西南科技大学

Southwest University of Science and Technology

本科毕业设计（论文）

题目：基于云服务的任务调度与负载均衡系统研究

学生姓名: 潘天佑

学生学号: 20130224

专　　业: 计算机科学与技术

指导教师: 吴亚东教授 张晓蓉讲师

学院(部): 计算机科学与技术学院

基于云服务的任务调度与负载均衡系统研究

摘要

如今云服务厂商越来越多，大家所使用的云服务也越来越多，传统云服务虽然给使用者带来了便利，但对开发者而言，云服务系统有着维护成本高昂，吞吐量不易扩展，难以重复利用的缺点。在这样的背景下，本论文将研究云服务的定义，云服务的工作方式，使用现有技术，不断优化代码，实现一个拥有负载均衡，日志记录，JMX 监控，服务管理的云服务基础中间件系统，该系统将在完成传统云服务基本功能的前提下，灵活的管理云服务，并可通过简单的扩展，接入其他云服务系统。且在 github 上开源，作为其他类似中间件的实现参考。经过测试，该系统实现了指定的功能，通过了测试。该系统代码结构清晰，接入简单。适用于生产系统时，只需更改配置文件，注册相应节点即可。可方便的接入服务与 Zookeeper 节点相关联的云服务系统中。该系统为云服务提升吞吐量，系统重用，提出了一种解决方案。

关键词：中间件，Zookeeper，Thrift，负载均衡，任务调度

RESEARCH ON TASK SCHEDULING AND LOAD BALANCING SYSTEM BASED ON CLOUD SERVICE

ABSTRACT

Now more and more cloud service vendors, we use more and more cloud services, although the traditional cloud services to the user to bring convenience, but for developers, the cloud service system has high maintenance costs, throughput Difficult to expand, difficult to re-use the shortcomings. In this context, this paper will study the definition of cloud services, cloud services work, the use of existing technology, and constantly optimize the code to achieve a load balancing, logging, JMX monitoring, service management cloud services based middleware System, the system will be completed in the traditional cloud services under the premise of the basic functions of flexible management of cloud services, and through simple expansion, access to other cloud services system. And github on the open source, as other similar middleware implementation reference. After testing, the system implements the specified function, passed the test. The system code structure is clear, access is simple. Apply to the production system, just change the configuration file, register the corresponding node can be. Can be easily accessed in the cloud service system associated with the Zookeeper node. The system for cloud services to enhance throughput, system reuse, put forward a solution.

Key words: Middle Ware, Zookeeper, Thrift, Distributed, Load Balance

目录

第 1 章 绪论

近年来，随着计算机技术，Internet 的不断发展及企业对计算成本的不断要求，计算机应用系统由集中式向分布式发展。软件的体系结构也从 CS 模式转向了多层应用体系结构[1]。随着分布式的普及，网络的发展，云服务诞生了。

1.1 云服务的定义

云服务是一类依托于云计算平台的新兴网络服务[2]，它基于互联网的相关服务的增加、使用和交付模式，通常涉及通过互联网来提供动态易扩展且经常是虚拟化的资源。

云是网络、互联网的一种比喻说法。过去在图中往往用云来表示电信网，后来也用来表示互联网和底层基础设施的抽象。

云服务指通过网络以按需、易扩展的方式获得所需服务。这种服务可以是 IT 和软件、互联网相关，也可是其他服务。它意味着计算能力也可作为一种商品通过互联网进行流通。

1.2 云服务拟解决的问题

（1）云物联

“物联网就是物物相连的互联网”。这有两层意思：

第一， 物联网的核心和基础仍然是互联网，是在互联网基础上的延伸和扩展的网络；

第二， 用户端延伸和扩展到了任何物品与物品之间，进行信息交换和通信。

物联网的两种业务模式：

①MAI（M2M Application Integration), 内部 MaaS；

②MaaS（M2M As A Service), MMO, Multi-Tenants(多租户模型）。

（2）云安全

云安全(Cloud Security)是一个从“云计算”演变而来的新名词。

云安全的策略构想是，使用者越多，每个使用者就越安全，因为如此庞大的用户群，足以覆盖互联网的每个角落，只要某个网站被挂马或某个新木马病毒出现，就会立刻被截获。

“云安全”通过网状的大量客户端对网络中软件行为的异常监测，获取互联网中木马、恶意程序的最新信息，推送到 Server 端进行自动分析和处理，再把病毒和木马的解决方案分发到每一个客户端。

（3）云存储

云存储是在云计算(cloud computing)概念上延伸和发展出来的一个新的概念，是指通过集群应用、网格技术或分布式文件系统等功能，将网络中大量各种不同类型的存储设备通过应用软件集合起来协同工作，共同对外提供数据存储和业务访问功能的一个系统。

当云计算系统运算和处理的核心是大量数据的存储和管理时，云计算系统中就需要配置大量的存储设备，那么云计算系统就转变成为一个云存储系统，所以云存储是一个以数据存储和管理为核心的云计算系统。

1.3 云服务的表现形式

（1）IaaS：基础设施即服务

消费者通过 Internet 可以从完善的计算机基础设施获得服务。如 Vultr 提供 VPS 机器，用户可购买它们提供的云服务器，在云服务器上运行任何应用。即服务商提供一些基础设施如服务器，路由器等。

（2）PaaS：基础平台即服务

平台即服务。PaaS 实际上是指将软件研发的平台作为一种服务，以 SaaS 的模式提交给用户。因此，PaaS 也是 SaaS 模式的一种应用。但是，PaaS 的出现可以加快 SaaS 的发展，尤其是加快 SaaS 应用的开发速度。如腾讯云的各种服务。

（3）SaaS：软件即服务

它是一种通过 Internet 提供软件的模式，用户无需购买软件，而是向提供商租用基于 Web 的软件，来管理企业经营活动。如神箭手云爬虫能够提供一个开发 js 爬虫的框架，开发者直接在浏览器中写好代码运行即可

1.4 云服务与分布式的关系

（1）分布式系统

分布式系统（distributed system）是建立在网络之上的软件系统。正是因为软件的特性，所以分布式系统具有高度的内聚性和透明性。

因此，网络和分布式系统之间的区别更多的在于高层软件（特别是操作系统），而不是硬件。内聚性是指每一个数据库分布节点高度自治，有本地的数据库管理系统。

透明性是指每一个数据库分布节点对用户的应用来说都是透明的，看不出是本地还是远程。在分布式数据库系统中，用户感觉不到数据是分布的，即用户不须知道关系是否分割、有无副本、数据存于哪个站点以及事务在哪个站点上执行等。

（2）分布式处理

分布式处理系统包含硬件，控制系统，接口系统，数据，应用程序和人等六个要素。而控制系统中包含了分布式操作系统，分布式数据库以及通信协议等。

分布式计算环境是在具有多地址空间的多计算机系统上进行计算和信息处理的软件环境。

而分布式软件系统是支持分布式处理的软件系统，它包括分布式操作系统，分布式程序设计语言及其编译系统，分布式文件系统和分布式数据库系统等。而 CORBA，COM+等是设计分布式软件系统的一些技术。

简单来说，分布式处理就是多台相连的计算机各自承担同一工作任务的不同部分，在人的控制下，同时运行，共同完成同一件工作任务。由多个自主的、相互连接的信息处理系统，在一个高级操作系统协调下共同完成同一任务的处理方式。利用网络技术能把许多小型机或微机连接成具有高性能的计算机系统，使其具有解决复杂问题的能力。[1]

（3）云服务与分布式的关系

云服务通过网络来调用服务，面对大量的调用，以及大量的调用引起的负载压力，计算压力，数据库压力，网络 IO 压力等，往往都会通过分布式技术进行解决，来提升整个系统的瓶颈。

当云服务发展到一定规模时，需要分布式服务进行提升。因此可以说，云服务是分布式技术的一种应用场景，分布式是能够提升云服务系统瓶颈的一种解决办法。

1.5 分布式拟解决的问题

（1）地理异构

一个工作流程，可能调用多个程序，程序 A,B 都需要强大的 CPU 进行计算，但因为资源限制，往往同一个地方并没有充足的计算资源，假设 A，B 依次运行在同一台机器上，运行时间往往会超出期望。

这时候分布式就能够让 A，B 程序按照一定的逻辑，通过网络，调用异地的计算资源进行并行计算。解决了地理异构带来的资源分配不均问题。

（2）提升 IO 效率

将一个服务产生的数据写入一台机器，当有多个服务同时写入时，磁盘 IO 效率会下降许多，崩溃风险极大，通过分布式，可以让数据写入有规则的分布在多台机器上，最后在进行结果回收。

（3）避免网络带宽限制

云服务的数据传输通过网络进行，网络带宽固定，以往单机模式时提高吞吐量只能通过提升带宽，价格高昂，通过使用分布式，可以使用大量成本低廉的机器接入网络，提升吞吐量。

（4）降低负载

单机集中式处理下，随着任务的增加，机器 CPU，内存都会上升，一旦超过某个阈值时，就会宕机，给业务带来很大的伤害。通过分布式的负载均衡技术，在保证高处理率的前提下，依旧能让业务可运行。

（5）加快计算效率

通过分布式技术，同一个计算任务通过简单的 MapReduce 思想，在多个机器上并行执行，最后进行结果回收，大大的提升了计算效率。降低了生产成本。

1.6 现有的云服务厂商和分布式系统

（1）阿里云

阿里云创立于 2009 年，是全球领先的云计算及人工智能科技公司，为 200 多个国家和地区的企业、开发者和政府机构提供服务。截至 2016 年第三季度，阿里云客户超过 230 万，付费用户达 76.5 万。阿里云致力于以在线公共服务的方式，提供安全、可靠的计算和数据处理能力，让计算和人工智能成为普惠科技。

阿里云服务着制造、金融、政务、交通、医疗、电信、能源等众多领域的领军企业，包括中国联通、12306、中石化、中石油、飞利浦、华大基因等大型企业客户，以及微博、知乎、锤子科技等明星互联网公司。在天猫双 11 全球狂欢节、12306 春运购票等极富挑战的应用场景中，阿里云保持着良好的运行纪录。

（2）腾讯云

腾讯云有着深厚的基础架构，并且有着多年对海量互联网服务的经验，不管是社交、游戏还是其他领域，都有多年的成熟产品来提供产品服务。腾讯在云端完成重要部署，为开发者及企业提供云服务、云数据、云运营等整体一站式服务方案。

具体包括云服务器、云存储、云数据库和弹性 web 引擎等基础云服务；腾讯云分析（MTA）、腾讯云推送（信鸽）等腾讯整体大数据能力；以及 QQ 互联、QQ 空间、微云、微社区等云端链接社交体系。这些正是腾讯云可以提供给这个行业的差异化优势，造就了可支持各种互联网使用场景的高品质的腾讯云技术平台。

（3）七牛云

七牛云是国内领先的企业级公有云服务商，致力于打造以数据为核心的场景化 PaaS 服务。围绕富媒体场景，七牛先后推出了对象存储，融合 CDN 加速，数据通用处理，内容反垃圾服务，以及直播云服务等。目前，七牛云已经在为 50 多万家企业提供服务，亲历互联网创新创业发展的同时，也深入理解传统企业转型过程中的云服务需求场景，推出了有针对性的一系列行业解决方案。

（4）金山云

金山云将面向三大领域：

第一， 为创新型智能硬件公司提供云服务，借助手环、智能电视、智能摄像头等入口布局移动互联网；

第二， 为客户提供更大规模的存储、图片和视频服务；

第三， 在“互联网+”浪潮下，利用云计算优势，帮助传统行业和企业拥抱互联网，实现业务转型。

（5）Hadoop

Hadoop 是一个由 Apache 基金会所开发的分布式系统基础架构。
用户可以在不了解分布式底层细节的情况下，开发分布式程序。充分利用集群的威力进行高速运算和存储。

Hadoop 原本来自于谷歌一款名为 MapReduce 的编程模型包。谷歌的 MapReduce 框架可以把一个应用程序分解为许多并行计算指令，跨大量的计算节点运行非常巨大的数据集。使用该框架的一个典型例子就是在网络数据上运行的搜索算法。Hadoop 最初只与网页索引有关，迅速发展成为分析大数据的领先平台。

目前有很多公司开始提供基于 Hadoop 的商业软件、支持、服务以及培训。Cloudera 是一家美国的企业软件公司，该公司在 2008 年开始提供基于 Hadoop 的软件和服务。GoGrid 是一家云计算基础设施公司，在 2012 年，该公司与 Cloudera 合作加速了企业采纳基于 Hadoop 应用的步伐。Dataguise 公司是一家数据安全公司，同样在 2012 年该公司推出了一款针对 Hadoop 的数据保护和风险评估。

（6）Spark

Apache Spark 是专为大规模数据处理而设计的快速通用的计算引擎。

Spark 是 UC Berkeley AMP lab (加州大学伯克利分校的 AMP 实验室)所开源的类 Hadoop MapReduce 的通用并行框架，Spark，拥有 Hadoop MapReduce 所具有的优点；但不同于 MapReduce 的是 Job 中间输出结果可以保存在内存中，从而不再需要读写 HDFS，因此 Spark 能更好地适用于数据挖掘与机器学习等需要迭代的 MapReduce 的算法。

Spark 是一种与 Hadoop 相似的开源集群计算环境，但是两者之间还存在一些不同之处，这些有用的不同之处使 Spark 在某些工作负载方面表现得更加优越，换句话说，Spark 启用了内存分布数据集，除了能够提供交互式查询外，它还可以优化迭代工作负载。

Spark 是在 Scala 语言中实现的，它将 Scala 用作其应用程序框架。与 Hadoop 不同，Spark 和 Scala 能够紧密集成，其中的 Scala 可以像操作本地集合对象一样轻松地操作分布式数据集。

尽管创建 Spark 是为了支持分布式数据集上的迭代作业，但是实际上它是对 Hadoop 的补充，可以在 Hadoop 文件系统中并行运行。通过名为 Mesos 的第三方集群框架可以支持此行为。Spark 由加州大学伯克利分校 AMP 实验室 (Algorithms, Machines, and People Lab) 开发，可用来构建大型的、低延迟的数据分析应用程序。

第 2 章 基于云服务的任务调度与负载均衡系统的需求设计

2.1 云服务的需求

（1）跨平台

跨平台泛指程序语言、软件或硬件设备可以在多种作业系统或不同硬件架构的电脑上运作。跨平台最民生最简单的理解就是在一个熟悉的平台上面开发的软件或者程序，直接可以在其他平台上正常的运行显示而不需要对其原始文件或者原始代码进行修改。

只有通过跨平台技术，云服务才能方便的部署在不同的机器上（比如 WindowsServer，Linux），并能够正常的运行。

现有实现跨平台的方案有：

① 使用跨平台语言如 Java，Python，Qt 等实现跨平台；

② 使用 Thrift，将不同的开发语言联系起来，达到跨平台的目的。

实际生产环境中，不同的模块往往由不同的技术团队负责，这些技术团队的 leader 又会使用不同的语言，因此方案 2 往往作为云服务的跨平台实现。

thrift 是一个软件框架，用来进行可扩展且跨语言的服务的开发。它结合了功能强大的软件堆栈和代码生成引擎，以构建在 C++, Java, Go,Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, JavaScript, Node.js, Smalltalk, and OCaml 这些编程语言间无缝结合的、高效的服务。本系统最终是作为一个中间件的角色，参与到云服务业务中，因此决定采用 java 实现，通过调用 jar 包，运行在后台。与其他业务通过 thrift 进行通讯。

（2）隔离

在多个云服务中，往往存在一些共享的文件或变量，对于同一个机器上不同的服务，或者不同的机器上的同一服务，访问同一个资源时，必须对资源使用做好隔离，也就是做好“锁”，只有进行完善的资源隔离，进程隔离，线程隔离，分布式才能如期的运行，否则会出现各种各样的 BUG。

（3）分布式锁：

可以被一个以上任务使用的资源叫做共享资源。为了防止数据被破坏，每个任务在与共享资源打交道时，必须独占该资源。

分布式锁是控制分布式系统之间同步访问共享资源的一种方式。在分布式系统中，常常需要协调他们的动作。

如果不同的系统或是同一个系统的不同主机之间共享了一个或一组资源，那么访问这些资源的时候，往往需要互斥来防止彼此干扰来保证一致性，在这种情况下，便需要使用到分布式锁。

在分布式系统中，常常需要协调他们的动作。如果不同的系统或是同一个系统的不同主机之间共享了一个或一组资源，那么访问这些资源的时候，往往需要互斥来防止彼此干扰来保证一致性，这个时候，便需要使用到分布式锁。

集合是编程中最常用的数据结构。而谈到并发，几乎总是离不开集合这类高级数据结构的支持。比如两个线程需要同时访问一个中间临界区（Queue），比如常会用缓存作为外部文件的副本。

在 Java 中，分布式锁通过 ConcurrentHashMap 或其他技术实现，用来实现在分布式情况下的程序读写功能。

通过分析 Hashtable 就知道，synchronized 是针对整张 Hash 表的，即每次锁住整张表让线程独占，Java 中的 ConcurrentHashMap 允许多个修改操作并发进行，其关键在于使用了锁分离技术。它使用了多个锁来控制对 hash 表的不同部分进行的修改。ConcurrentHashMap 内部使用段(Segment)来表示这些不同的部分，每个段其实就是一个小的 hash table，它们有自己的锁。只要多个修改操作发生在不同的段上，它们就可以并发进行。

有些方法需要跨段，比如 size()和 containsValue()，它们可能需要锁定整个表而而不仅仅是某个段，这需要按顺序锁定所有段，操作完毕后，又按顺序释放所有段的锁。这里“按顺序”是很重要的，否则极有可能出现死锁，在 ConcurrentHashMap 内部，段数组是 final 的，并且其成员变量实际上也是 final 的，但是，仅仅是将数组声明为 final 的并不保证数组成员也是 final 的，这需要实现上的保证。这可以确保不会出现死锁，因为获得锁的顺序是固定的。ConcurrentHashMap 可以简单理解成把一个大的 HashTable 分解成多个，形成了锁分离。

（4）信号量：

信号量(Semaphore)，有时被称为信号灯，是在多线程环境下使用的一种设施，是可以用来保证两个或多个关键代码段不被并发调用。在进入一个关键代码段之前，线程必须获取一个信号量；一旦该关键代码段完成了，那么该线程必须释放信号量。

其它想进入该关键代码段的线程必须等待直到第一个线程释放信号量。为了完成这个过程，需要创建一个信号量 VI，然后将 Acquire Semaphore VI 以及 Release Semaphore VI 分别放置在每个关键代码段的首末端。确认这些信号量 VI 引用的是初始创建的信号量。

（5）透明

透明概念最早来源于防火墙的透明模式，顾名思义，首要的特点就是对用户是透明的(Transparent)，即用户意识不到防火墙的存在。要想实现透明模式，防火墙必须在没有 IP 地址的情况下工作，不需要对其设置 IP 地址，用户也不知道防火墙的 IP 地址。适用环境:1.服务器必须是真实互联网地址。2.需要保护同一子网上不同区域主机。

随着互联网的发展，渐渐出现了“业务透明”，“应用透明”等概念。

云服务对于用户来说，往往是按一个按钮，点击网页上的某个选项，来进行启动的。至于云服务背后调用了哪些服务，在哪些机器上运行，结果出现在哪台机器上，应当对用户都是透明的。

用户仅仅需要进行启动和终止，并能够观察结果或过程结果就可以了。

实现这种技术需要通过多个模块的合作，通过代理的方式，完成用户的任务并返回结果。

透明的好处有很多，比如，有一些服务不再需要了，只需要关闭这些服务即可。有服务新增，则只需要写服务逻辑即可。

服务或者说业务逻辑，与程序逻辑已经解耦了。这是指业务透明。而从安全上来说，用户无需知道自己到底访问了那个机器的 ip，对服务器集群信息也能进行一定程度上的保护。用户发出的信息全都转给代理，由代理进行下一步操作。

（6）容灾

云服务面对的业务环境，通常都十分严峻，用户不确定的调用时间，突发的调用高峰，因为程序 BUG 而引起的突发性崩溃，难以跟踪调试等，任何一条都有可能对业务造成很大的伤害，只有完备的容灾策略，才能保证服务的“高可用”，让用户可信赖。容灾技术在实现上，往往分为，数据库备份，主从库复制，宕机重启，状态监控等多个部分，具体实现也会因业务而有所不同。

容灾系统是指在相隔较远的异地，建立两套或多套功能相同的 IT 系统，互相之间可以进行健康状态监视和功能切换，当一处系统因意外（如火灾、地震等）停止工作时，整个应用系统可以切换到另一处，使得该系统功能可以继续正常工作。容灾技术是系统的高可用性

技术的一个组成部分，容灾系统更加强调处理外界环境对系统的影响，特别是灾难性事件对整个 IT 节点的影响，提供节点级别的系统恢复功能。

从其对系统的保护程度来分，可以将容灾系统分为:

数据级容灾是指通过建立异地容灾中心，做数据的远程备份，在灾难发生之后要确保原有的数据不会丢失或者遭到破坏。

应用级容灾是在数据级容灾的基础之上，在备份站点同样构建一套相同的应用系统，通过同步或异步复制技术，这样可以保证关键应用在允许的时间范围内恢复运行，尽可能减少灾难带来的损失，让用户基本感受不到灾难的发生，这样就使系统所提供的服务是完整的、可靠的和安全的。

业务级容灾是全业务的灾备，除了必要的 IT 相关技术，还要求具备全部的基础设施。其大部分内容是非 IT 系统（如电话、办公地点等），当大灾难发生后，原有的办公场所都会受到破坏，除了数据和应用的恢复，更需要一个备份的工作场所能够正常的开展业务。

（7）服务管理

①服务发现：作为服务中心，应当能感知到所有加入服务组（如根据 ip 规定的范围组，或根据某个程序逻辑，如 List，Map 等规定的组）的节点，并将该服务纳入监控的范围。

②服务注册：服务应当可插拔，随时可以让系统具备某种服务的处理能力，或关闭对某种服务的处理能力。注册表现为，用户或开发者，可以通过某条消息向服务中心注册一个服务，并让服务中心监控该服务。

③服务注销：当用户退出服务时，或不再使用某服务时，应当进行服务注销，或由开发者因业务原因对某些服务进行关闭。

④服务监控：服务中心应当能对服务所在的机器进行监控，通过某些策略（如轮询）更新机器状态。

（8）可伸缩与中间件

可伸缩性是分布式计算和并行计算中的重要指标，它描述了系统通过改变可用计算资源和调度方式来动态调整自身计算性能的能力。[11]

云服务需要可伸缩的部署，根据用户量的增加，灵活的改变集群，以适应不同阶段的需要，减少成本，提升效率。而实现中间件就是一种达成可伸缩功能的途径。

中间件是一种独立的系统软件或服务程序，分布式应用软件借助这种软件在不同的技术之间共享资源。中间件位于客户机/ 服务器的操作系统之上，管理计算机资源和网络通讯。是连接两个独立应用程序或独立系统的软件。相连接的系统，即使它们具有不同的接口，但通过中间件相互之间仍能交换信息。执行中间件的一个关键途径是信息传递。通过中间件，应用程序可以工作于多平台或 OS 环境。中间件是基于构件技术, 处于两种或多种软件之间(通常是在应用程序和操作系统、网络操作系统或数据库管理系统之间)传递信息的软件。它能使用一种脚本语言来选择和连接已有的服务从而生成简单程序的软件开发工具。

它使用系统软件所提供的基础服务（功能），衔接网络上应用系统的各个部分或不同的应用，能够达到资源共享、功能共享的目的。[3]

云服务需要大量的中间件，减少强依赖，以便部署在不同机器上。将不同的业务分布在不同的地方，以缩短单个任务的执行时间来提升效率。[4]

2.2 云服务的难点

（1）负载均衡

负载均衡建立在现有网络结构之上，它提供了一种廉价有效透明的方法扩展网络设备和服务器的带宽、增加吞吐量、加强网络数据处理能力、提高网络的灵活性和可用性。

负载均衡技术面临的三大问题就是服务器的负载状况的定义、如何获取以及获取后如何处理的问题[7]

负载均衡，英文名称为 Load Balance，其意思就是分摊到多个操作单元上进行执行，例如 Web 服务器、FTP 服务器、企业关键应用服务器和其它关键任务服务器等，从而共同完成工作任务。负载均衡在技术上的实现有许多种方案，比较常用的方案有：

通过一个定时轮询所有机器状态的脚本，来达到监控的目的，该脚本同时负责向外部提供一个“Safe”的机器节点。该节点从监控的所有机器中选出。为负载最小的节点；

通过 Zookeeper 的节点注册/销毁，来判断机器上对应的服务是否正常，使用一个 hashMap 来记录每个节点的状态。通过合适的算法（轮询，最小堆最大堆等），选择出合适的机器反馈给调用方。

（2）数据一致性

分布式中，多个机器同时访问数据库的同一内容，需要每台机器读到的内容都是一致的，只有数据一致性保证了，分布式系统才是“可靠的，稳定的”。

在技术上，数据一致性主要通过以下三个方面实现：

①DBA 的 SQL 语句；

②程序逻辑中正确的处理（错误回滚，事务提交等）；

③数据不一致时的处理策略（根据具体业务逻辑实现）。

（3）并行计算

分布式的一个应用场景就是海量计算，俗称“云计算”，在云计算中，核心技术主要为 MapReduce，通过该思想，将一个巨大的计算分成无数小计算，最后进行结果合并。

MapReduce 是一种编程模型，用于大规模数据集（大于 1TB）的并行运算。概念"Map（映射）"和"Reduce（归约）"，是它们的主要思想，都是从函数式编程语言里借来的，还有从矢量编程语言里借来的特性。

它极大地方便了编程人员在不会分布式并行编程的情况下，将自己的程序运行在分布式系统上。当前的软件实现是指定一个 Map（映射）函数，用来把一组键值对映射成一组新的键值对，指定并发的 Reduce（归约）函数，用来保证所有映射的键值对中的每一个共享相同的键组。

简单说来，一个映射函数就是对一些独立元素组成的概念上的列表（例如，一个测试成绩的列表）的每一个元素进行指定的操作（比如前面的例子里，有人发现所有学生的成绩都被高估了一分，它可以定义一个“减一”的映射函数，用来修正这个错误。

事实上，每个元素都是被独立操作的，而原始列表没有被更改，因为这里创建了一个新的列表来保存新的答案。

这就是说，Map 操作是可以高度并行的，这对高性能要求的应用以及并行计算领域的需求非常有用。

虽然他不如映射函数那么并行，但是因为化简总是有一个简单的答案，大规模的运算相对独立，所以化简函数在高度并行环境下也很有用.

在 Google，MapReduce 用在非常广泛的应用程序中，包括分布 grep，分布排序，web 连接图反转，每台机器的词矢量，web 访问日志分析，反向索引构建，文档聚类，机器学习，基于统计的机器翻译...等。

（4）调试

云服务中，用户调用的服务往往在不同机器上，甚至会先后调用不同机器上的不同服务，这給调试工作带来了极大的苦难，即使部署了测试环境，也很难模拟线上环境，导致实际生产环境中，常常发生难以处理的 BUG。

为了解决这个问题，在云服务系统中，需要实现：

①日志系统：这里一般使用 Log4j，Slf4j，LogBack 等的日志模块。对于强分布式的应用，往往自己实现日志组件，发送到 Redis，在进行统一过滤；

②异常捕获：一般语言的 TryCatch 即可实现；

③测试环境：这里一般使用 Docker 创建新的测试环境，来模拟生产环境。

2.3 功能需求

（1）整体功能

该系统包括的功能为，用户可远程调用对应服务并获取结果，服务将根据负载运行在合适的机器上，服务可动态修改。

整体的数据流如图 2-1 所示。

用户向服务管理系统发起调用，服务管理系统根据节点管理系统与负载均衡系统的状态，调用某台机器上的某个服务。

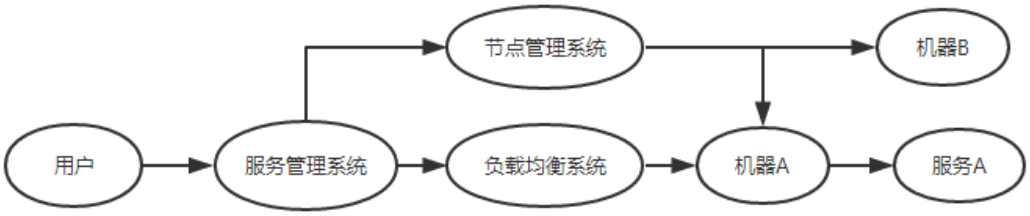

图 2-1 整体功能数据流图

（2）负载均衡

该系统包括的功能为，定时获取一组机器上的机器状态数据并存储，根据指定算法筛选出合适的机器，该部分的数据流如图 2-2 所示。

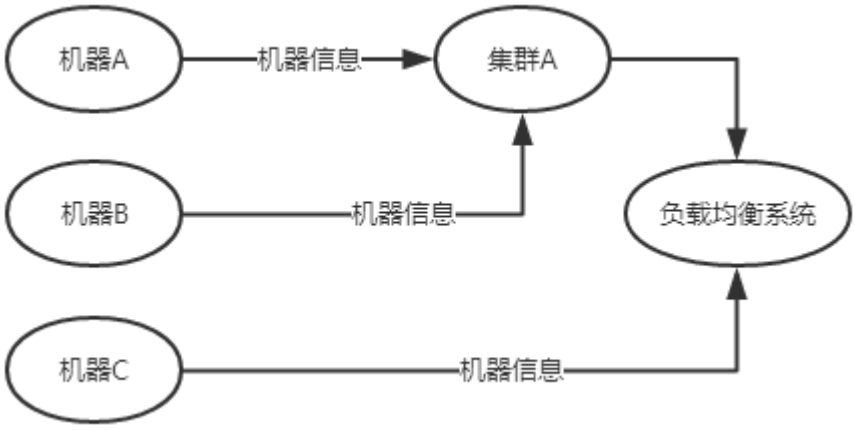

图 2-2 负载均衡数据流图

（3）节点管理

该系统包括的功能为，随时监听节点变化（服务变化），根据节点变化定期通知服务管理系统，更新相应的 Map，该部分的数据流如图 2-3 所示。

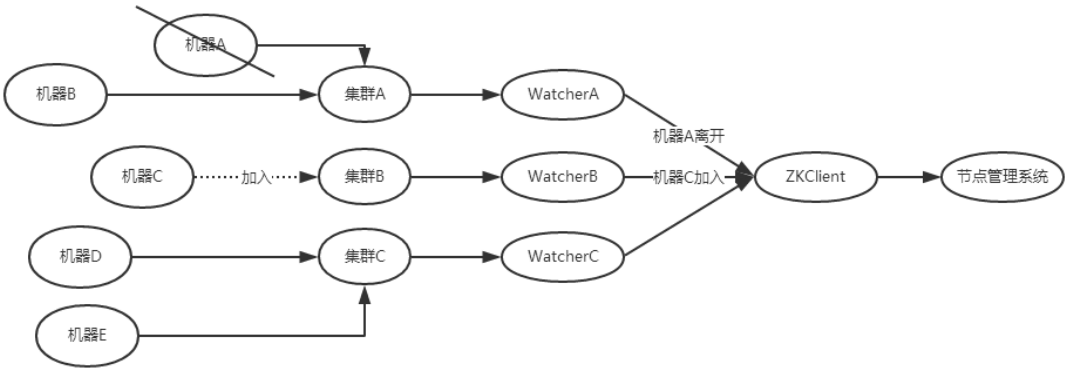

图 2-3 负载均衡数据流图

如图 2-3 所示，该图表示当机器 A 离开集群 A，机器 C 加入集群 B 时，ZKClient 收到通知并反馈给节点管理系统

2.4 本章小结

本章分析了云服务的需求与难点，分析了云服务面对的问题如容灾，中间件，数据一致性，负载均衡等，并给出了一些解决方式，同时从云服务的需求出发，就本系统的整体功能，负载均衡功能，节点管理功能，给出数据流图说明。

第 3 章 基于云服务的任务调度与负载均衡系统的总体设计

3.1 整体设计

实现一个服务管理中心，客户端能够给服务管理中心注册某个服务，并得到某个服务的响应。服务管理中心能够监控某个服务，并返回某服务的结果。且服务中心有以下功能：

（1） 负载均衡：多个云服务部署在不同节点上，用户调用时，采用负载最小的节点运行服务；

（2） 任务调度：能够将客户的任务合理的运行在适当机器上；

（3） 滚动日志：打印完整的日志，同时每过一段时间会压缩，并留档；

（4） 服务管理：将服务与节点绑定，完成服务的注册，删除，更新。

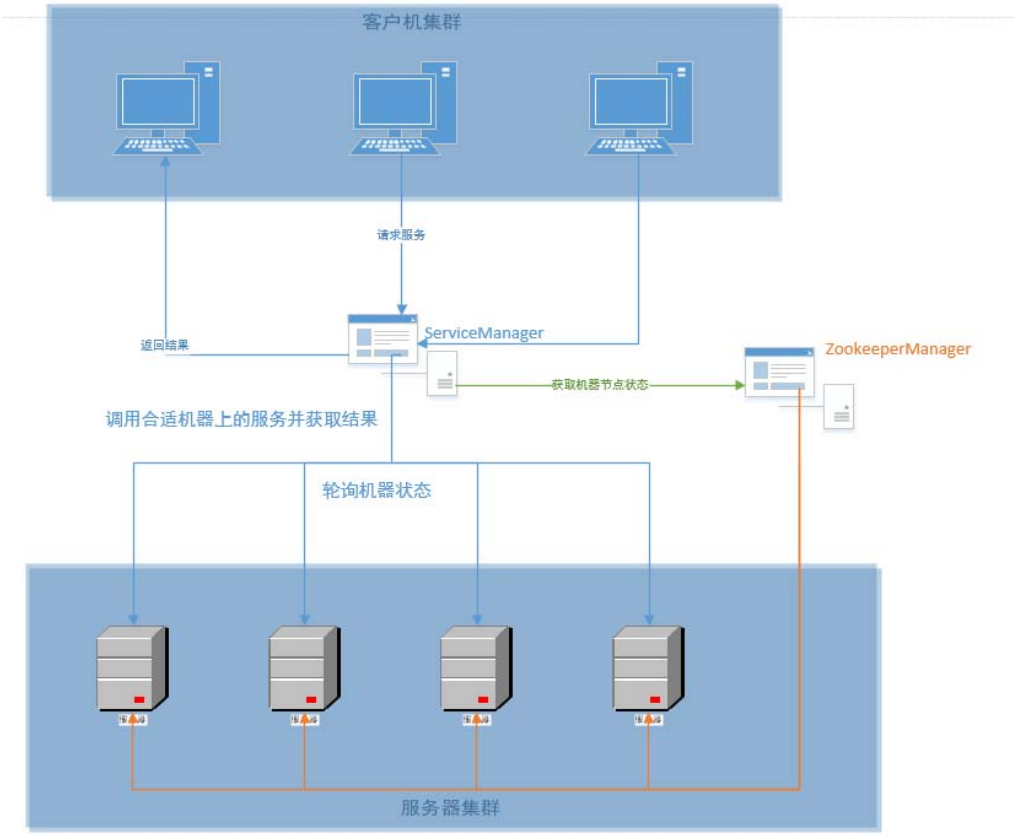

图 3-1 ServiceManagerBusiness

（2）项目结构

本项目由负载均衡（LoadBalance），节点管理（Zookeeper），任务调度（Thrift，SCServiceHandler），配置加载（SCConfig），这几个部分组成，监控通过 JMX 实现。这几个部分及其子部分如图 3-2 所示。

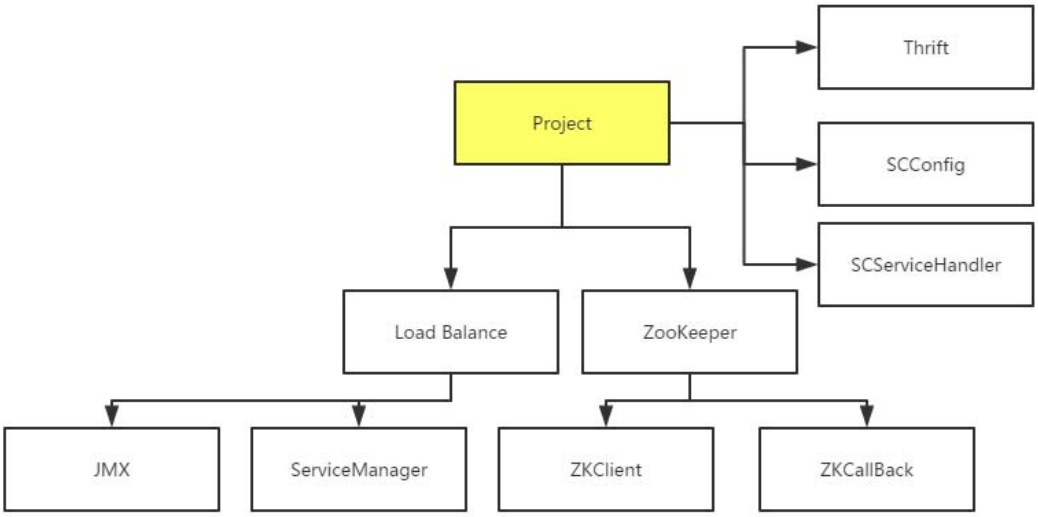

图 3-2 ServiceManager 项目结构

3.2 所用技术

（1）开发工具：IDEA，PyCharm。

（2）开发语言：Java，Python。

（3）第三方技术：Zookeeper，Thrift，Apache Curator。

3.3 模块概述

（1）总介绍

如表 3-1 所示。

表 3-1 项目包含模块

模块名	功能
SCConfig	实现读取配置文件
LoadBalance	负载均衡模块
Thrift	跨平台通信模块，实现了注册服务与调用服务的接口
Zookeeper	实现了一个包装好的 ZKClient 在高层次上进行节点操作
Logback	生成根据时间滚动的日志文件
ServiceHandler	处理 thrift 请求
SCMain	启动函数

（2）SCConfig：配置模块

该模块通过 Java 的 properties 进行属性读取，属性包括：

a) serverIP SC 的 ip；
b) serverPort SC 的 port；
c) thriftThreadMax thrift 最大线程数；
d) zkServer 部署 zk 的服务器 ip；
e) zkPort 部署 zk 的服务器端口；
f) zkTimeout zk 操作的超时限制；
g) JMXEnabled 是否开启 JMX；
h) ServerList 服务列表。

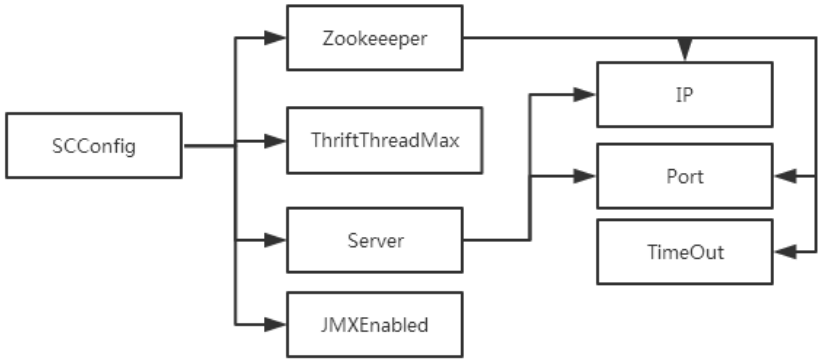

图 3-3 SCConfig 模块细节

（3）LoadBalance 负载均衡模块

Ruler 为算法模块该模，块可自由选择算法类型如轮询，随机选择，内存最小，CPU 最小等，选择后，该算法的 getNode 方法将采用合适的策略从一系列节点中，筛选出最合适的那个。

NodesGroup 是节点组，该模块包含了某个服务下一系列节点的信息，包括每个节点的 path，并时刻获取某个节点的实际机器负载信息并更新。

Nodes 是节点数据结构，该模块包含了一个节点的数据：Ip，Port，Path。

ServiceManager 是服务管理，该模块实现了一个定期更新机器数据的线程，以及一个 ManagerMap<serviceName, NodesGroup>，并且接收 ZK 的回调以更新 ManagerMap。

JMX 用于实现 JMX 监控，该模块遵循 JMX 标准，将 managerMap 交给 JMX 监控。

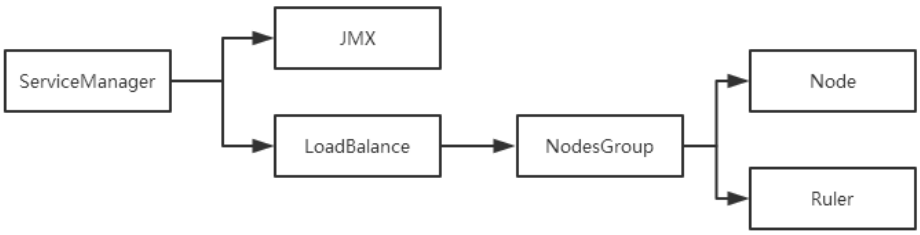

图 3-4 LoadBalance 模块实现细节

（4）Thrift 跨平台通讯模块

SCClientService 是 Thrift Client 接口，其他模块实现该文件的 handler 接口，处理 thrift 调用。

SCService 是 Thrift Server 接口其他模块实现该文件的 handler 接口，处理 thrift 调用。

（5）Zookeeper 模拟集群模块

ZKCallBack 为当节点变化时触发的回调接口，该模块包括三个接口，分别为节点添加，节点删除，节点更新，在对应 ZK 节点事件发生时，会被调用，该接口被 ServiceManager 实现。

ZKClient 为高层次 ZK 客户端，该模块是底层的 Zookeeper 的封装，接收外部命令，通过 ApacheCurator 的流式 API 对 zk 进行操作。

（6）LogBack

日志模块，设置滚动日志。

（7）ServiceHandler

thrift 信息处理类。

（8）SCMain

启动类，该模块执行顺序如图 3-5 所示。

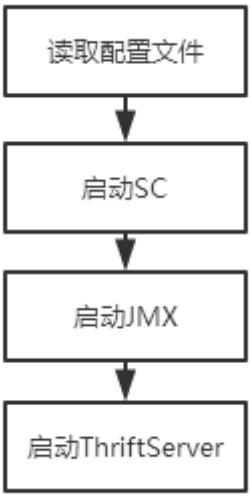

图 3-5 SCMain 启动顺序

3.4 本章小结

本章对系统的功能设计进行了分析，从整体设计上将系统分为负载均衡，任务调度，滚动日志，服务管理等几个模块。并给出实现一个客户端能够给服务管理中心注册某个服务，并得到某个服务的响应。服务管理中心能够监控某个服务，并返回某服务的结果这样的服务中心的解决方案。

第 4 章 基于云服务的任务调度与负载均衡系统的详细设计与实现

4.1 模块详细设计

4.1.1 任务调度模块

如图 4-1 所示，客户端发起一次 GetService 请求，请求 A 服务，ServiceManager 将在节点表中，找到对应 A 服务的节点组（集群）GroupB，再从 Group B 中根据 Ruler 模块选择出对应的 NodeC，调用 NodeC 上的服务（为了易于接入，该部分将会获取 NodeC 节点上的数据），并返回给 Client。关键代码见附录 1-1。

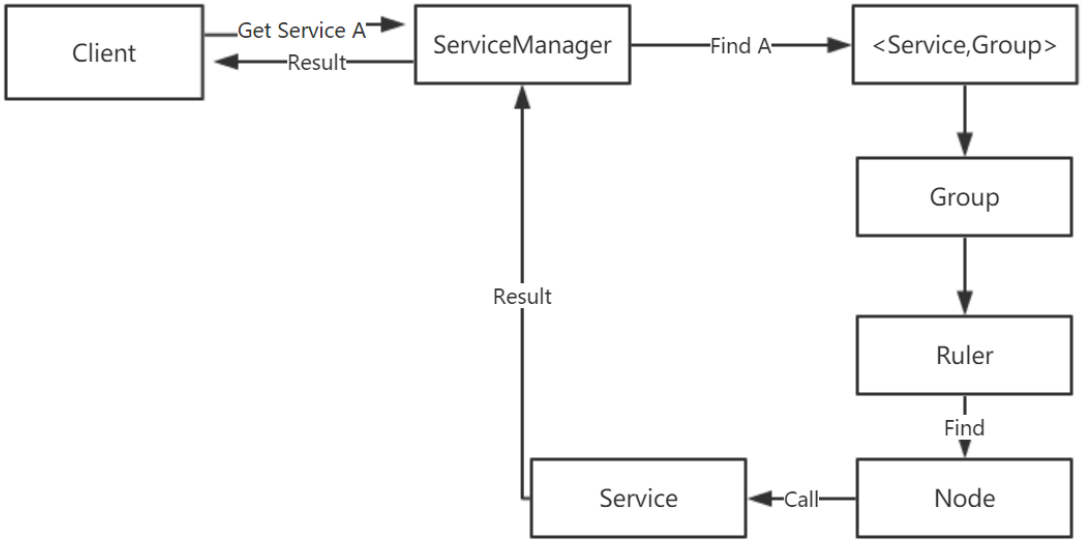

图 4-1 任务调度流程

4.1.2 服务管理模块

服务是通过节点来表现的，在程序中，每一个节点，表示一台机器，一个服务。一个 ZK 节点，包含一个 IP 地址和端口号，和自带的 Data 数据，IP 和端口号表示一台机器，Data 数据为服务名。

如图 4-2 所示，该图显示了当 Client 发起“Register Service A”的请求时，ServiceManager 会先寻找该服务是否存在于节点表中，如不存在则创建，调用 ZKClient 创建相应的 ZooKeeper 节点。在 ZKClient 节点创建成功后，将注册的服务添加到节点表中。否则返回错误。

关键代码见附录 1-2，给出了注册服务的示例。注册服务时，使用一个 GSON 格式的数据串。该数据格式如图 4-3 所示。Groups 表示集群，它包含了一系列 Nodes，一个 Nodes 由一系列 Node 组成。

Groups 是逻辑上的，由开发者任意划分集群。

Nodes 是服务组，它包含了同一个服务对应的所有可用节点。

Node 是具体的节点，它是 Nodes 中的一个单位，包括节点数据，ip，端口，节点路径。

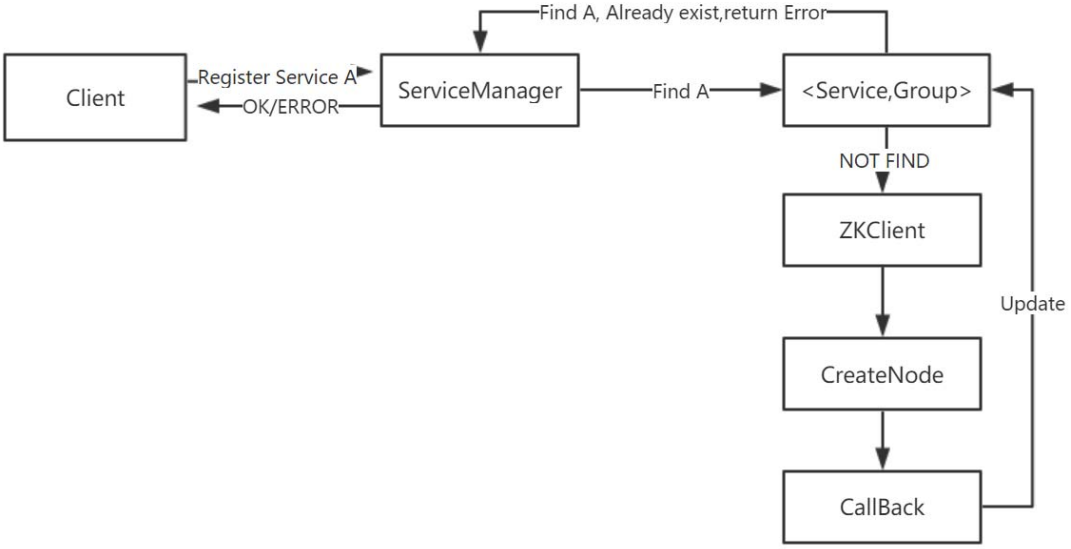

图 4-2 服务管理流程


```
{
    "groups":[
        {
            "nodes":[
                {
                    "data":"test-127.0.0.1-2181",
                    "ip":"127.0.0.1",
                    "path":"/zookeeper/@TEST/test0/zk-0",
                    "port":"2181"
                },
                {
                    "data":"test-127.0.0.1-2181",
                    "ip":"127.0.0.1",
                    "path":"/zookeeper/@TEST/test0/zk-1",
                    "port":"2181"
                }
            ],
            "path":"/zookeeper/@TEST/test0",
            "serviceName":"test0",
            "tag":"@@test0"
        },
```


图 4-3 数据 Model

4.1.3 负载均衡模块

如图 4-4 所示，ServiceManager 的 Map 中，储存了每个服务各自对应的所有节点的信息。ServiceManager 将独自开启一个线程，轮询 Map 中存储的所有节点信息，通过 thrift 接口，获取对应节点的机器信息，并更新。

关键代码见附录 1-3，给出了更新机器状态的关键代码。

实际效果如图 4-5 所示。数据用@隔开，依次为：

（1） 内存使用率

（2） CPU 使用率

（3） 网卡发送/接收所占比

（4） 磁盘读/写所占比

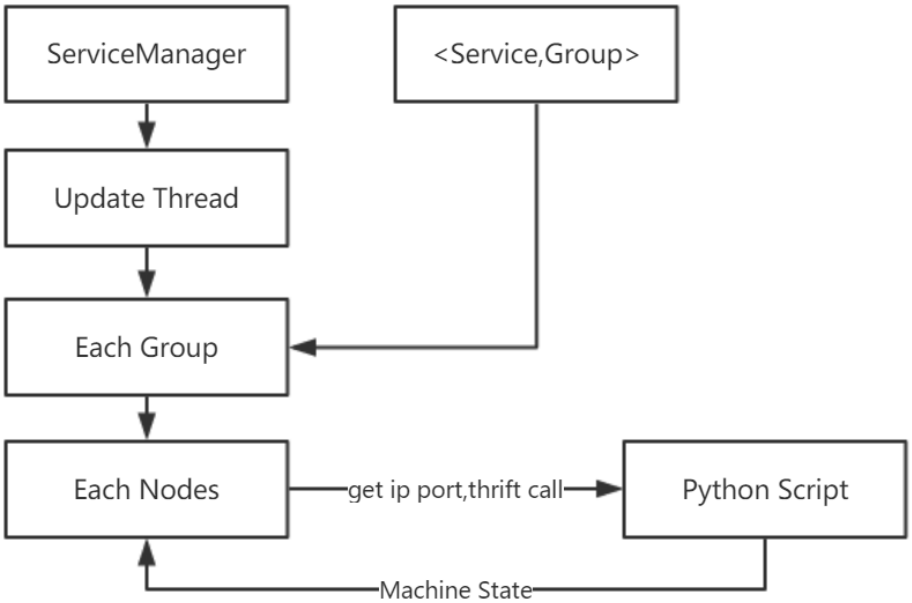

图 4-4 负载均衡流程

```
. java:115 ## get machine status : 28.2@6.0@1478@15
. java:115 ## get machine status : 28.2@25.9@1478@15
. java:115 ## get machine status : 28.2@0.0@1478@15
. java:115 ## get machine status : 28.3@33.3@1478@15
. java:115 ## get machine status : 28.2@23.1@1478@15
. java:115 ## get machine status : 28.2@20.0@1478@15
. java:115 ## get machine status : 28.2@0.0@1478@15
. java:115 ## get machine status : 28.3@20.0@1478@15
. java:115 ## get machine status : 28.2@7.7@1478@15
. java:115 ## get machine status : 28.2@9.1@1478@15
. java:115 ## get machine status : 28.2@38.5@1478@15
. java:115 ## get machine status : 28.2@0.0@1478@15
. java:115 ## get machine status : 28.2@0.0@1478@15
```

图 4-5 获取机器状态数据

4.1.4 日志模块

a) 概述

日志通过 LogBack 模块进行实现，只需要按照约定实现一个 XML 文件，并调用 Log.info 等 API 即可打印日志到指定输出。

b) 附录中给出了所用的 xml 文件。见附录 1-4。

c) 实际效果如图 4-6 所示。

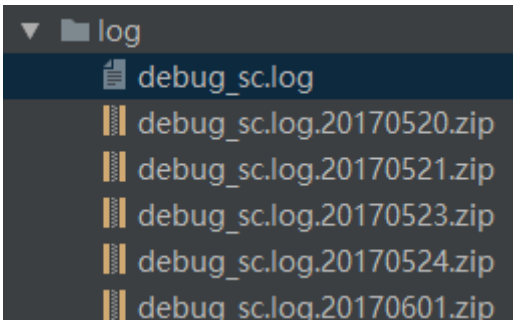

图 4-6 打印并滚动的日志

4.1.5 监控模块

d) 概述

通过 Java 的 JMX 技术实现。

e) 关键代码见附录 1-5，给出了 thriftCall 指标的计算方式

f) 实际效果如图 4-7、4-8 所示。

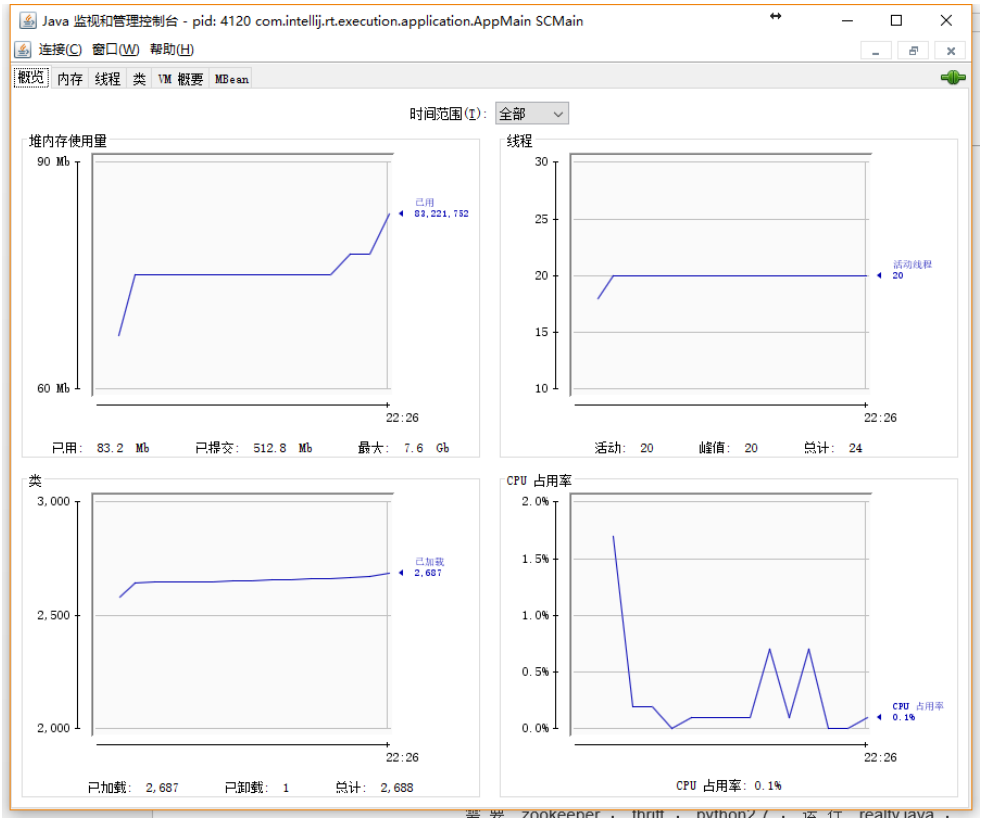

图 4-7 SC 程序的 JMX 监控界面 1

4.2 部署

（1）Client

需要 JDK 1.8，thrift，python2.7，运行 client.java，client.java 将会每隔一段时间就开启一定数量的 python client 进程，该进程会向 SC 发起 thrift 调用，进行 register service 或 getservice 调用。

（2）ServiceManager

需要 JDK1.8，thrift，zookeeper，运行 SCMain.java。

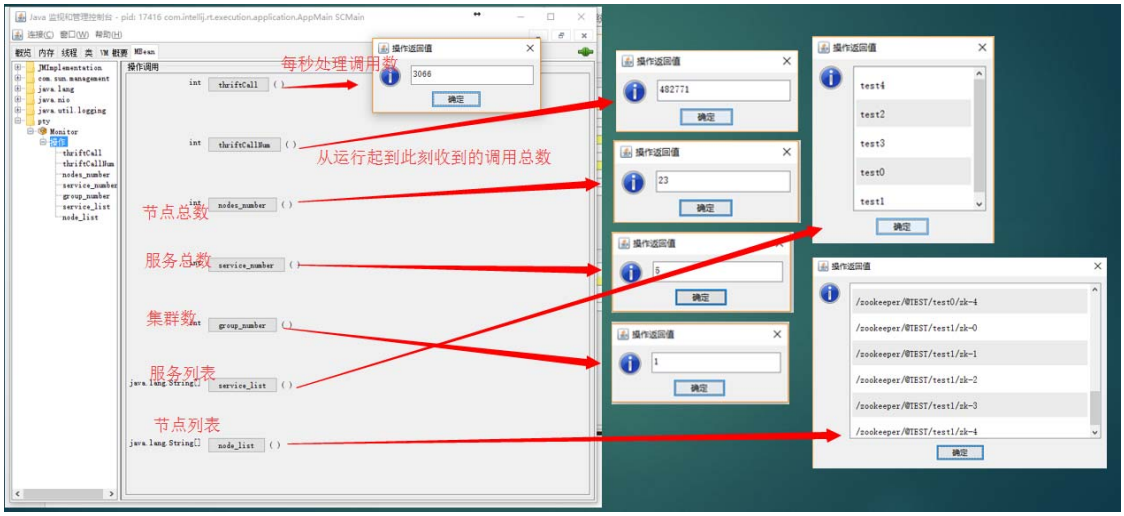

图 4-8 JMX 监控指标

（3）Server Nodes

需要 zookeeper，thrift，python2.7，运行 realty.java，machineStatue.py，machineStatue.py 会定期获取脚本所在机器上的负载信息如 CPU 使用率，内存使用率等。

（4）扩展

如图 4-9 所示。通过平行扩展，给同一个集群（Groups）或多个集群加上多个 ServiceManager，其中：

每个 ServiceManager 只接收 100 左右的连接数；

ServiceManager 之间的数据共享通过 Redis 消息队列或者数据库进行；

Connection Balance 负责把用户发起的链接均衡的分配到不同的 ServiceManager 上。

通过这种方式，随着用户量的增长，该系统的吞吐量也会不断的提升。可动态扩增，符合生产需要。

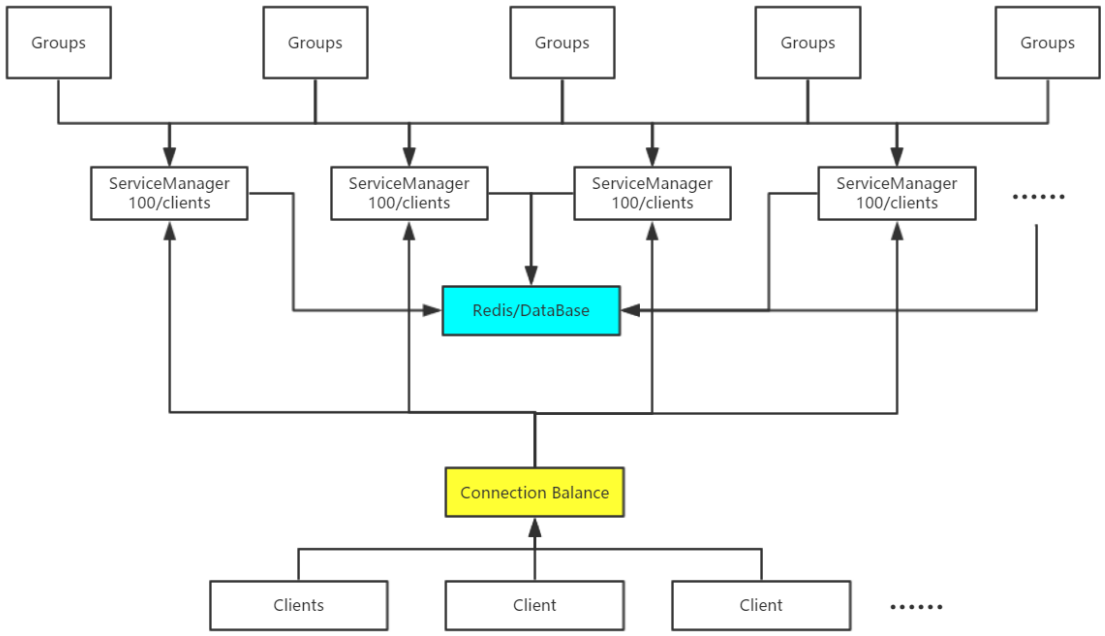

图 4-9 平行扩展

4.3 本章小结

本章详细解释了各模块设计，分别给出了任务调度，服务管理，负载均衡，日志，监控模块的概述，流程图，关键代码，以及一些具体效果。

任务调度，服务管理，负载均衡模块可通过调用流程了解其内在运行，日志模块的 xml 文件在附录中有，监控模块通过图片解释了监控的指标和监控运行界面。

此外讲明了如何部署程序，以及阐述了一种在生产环境中，面对不断增长的用户，或访问量动态变化的业务时，进行动态系统吞吐量扩展的方法。通过该方法，克服了 thrift 本身连接数的瓶颈。

第 5 章 基于云服务的任务调度与负载均衡系统的测试

5.1 功能测试

表 5-1 功能测试

序号	测试功能	测试方法	预期结果
1	服务注册	调用 Client 对应方法	监控界面出现新服务
2	服务删除	同上	监控界面服务消失
3	节点删除	同上	监控界面节点消失
4	服务获取	同上	Log 打印出返回的服务
5	日志打印	观察 log 日志	有对应的 log 信息
6	获取机器负载	观察 log 日志	有对应的机器负载信息
7	负载均衡	多次调用 getService	因随机策略，会返回不同节点

5.1.1 服务注册测试

初始状态如图 5-1 所示。程序刚启动时，没有建立任何节点，此时点击监控指标里的“Service_List”，出现如下对话框。

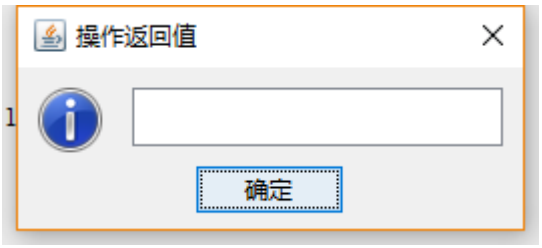

图 5-1 服务注册初始状态

注册节点后，再次点击监控指标的“Service List”，出现对话框如图 8-2 所示。

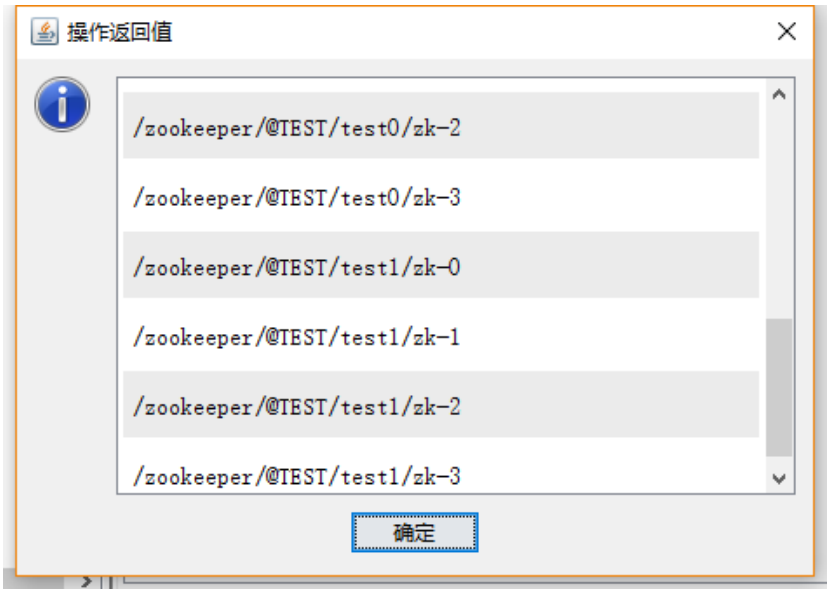

图 5-2 服务注册后状态

5.1.2 服务删除测试

注册服务之后，此时点击监控指标“Service List”，初始状态如图 5-3 所示。

图 5-3 服务删除初始状态

运行 Client，发起删除请求，删除 test2 后，点击“Service List”后如图 5-4 所示。

图 5-4 服务删除后状态

5.1.3 节点删除测试

从图 5-4 的状态开始，删除所有节点，删除后点击“Service List”后，如图 5-5 所示。

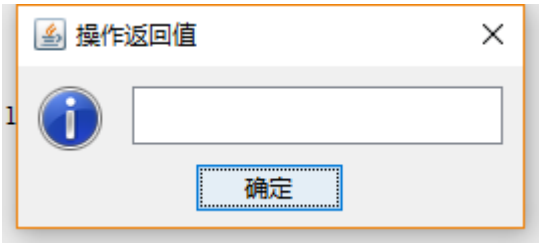

图 5-5 删除节点后状态

5.1.4 服务获取测试

Log 打印如下：

```
    Received 1
[/zookeeper/@TEST/test0/zk-0, /zookeeper/@TEST/test0/zk-1,
/zookeeper/@TEST/test0/zk-2, /zookeeper/@TEST/test0/zk-3]

    Process finished with exit code 0
```

5.1.5 日志打印测试

如图 5-6，该图显示了 Log 依次记录下程序运行开始，zookeeper Client 运行开始的过程。

```
meworkImpl.java:238 ## Starting
keeperClient.java:181 ## Starting
State.java:100 ## Starting
State.java:211 ## reset
t.java:100 ## Client environment:zookeeper.version=3.4.6-1569965, built on 02/20/2014
t.java:100 ## Client environment:host.name=10.10.4.203
t.java:100 ## Client environment:java.version=1.8.0_60
t.java:100 ## Client environment:java.vendor=Oracle Corporation
t.java:100 ## Client environment:java.home=C:\Program Files\Java\jdk1.8.0_60\jre
t.java:100 ## Client environment:java.class.path=C:\Program Files\Java\jdk1.8.0_60\jr
:\Program Files\Java\jdk1.8.0_60\jre\lib\jfxswt.jar;C:\Program Files\Java\jdk1.8.0_60
.1\httpcore-4.4.1.jar;C:\Users\Administrator\.m2\repository\org\apache\curator\curato
ic-0.9.28.jar;C:\Users\Administrator\.m2\repository\com\alibaba\fastjson\1.2.31\fast
t.java:100 ## Client environment:java.library.path=C:\Program Files\Java\jdk1.8.0_60\
\Program Files (x86)\NVIDIA Corporation\PhysX\Common;C:\Program Files\Microsoft SQL S
t.java:100 ## Client environment:java.io.tmpdir=C:\Users\ADMINI~1\AppData\Local\Temp\
t.java:100 ## Client environment:java.compiler=<NA>
t.java:100 ## Client environment:os.name=Windows 10
t.java:100 ## Client environment:os.arch=amd64
t.java:100 ## Client environment:os.version=10.0
```

图 5-6 日志打印 Log

5.1.6 获取机器负载测试

该图显示了一系列当获取到机器状态时打印的 Log。依次为内存使用率，CPU 使用率，网络使用所占最大值比例，磁盘所占最大值比例。（最大值可自定义）

```
)desGroup.java:115 ## get machine status : 28.2@6.0@1478@15
)desGroup.java:115 ## get machine status : 28.2@25.9@1478@15
)desGroup.java:115 ## get machine status : 28.2@0.0@1478@15
)desGroup.java:115 ## get machine status : 28.3@33.3@1478@15
)desGroup.java:115 ## get machine status : 28.2@23.1@1478@15
)desGroup.java:115 ## get machine status : 28.2@20.0@1478@15
)desGroup.java:115 ## get machine status : 28.2@0.0@1478@15
)desGroup.java:115 ## get machine status : 28.3@20.0@1478@15
)desGroup.java:115 ## get machine status : 28.2@7.7@1478@15
```

图 5-7 机器负载 Log

5.1.7 负载均衡测试

第一次调用 Log 输出：

```
    Received 1
test0//zookeeper/@TEST/test0/zk-1


    Process finished with exit code 0
```

第五次调用 log 输出：

```
    Received 1
test0//zookeeper/@TEST/test0/zk-0


    Process finished with exit code 0
```

5.2 压力测试

第一个指标，表明了该系统同一时间能维持多少用户的使用，第二个指标，表明同一个用户的操作频率能够多快。这两个指标共同决定了系统的响应速度。

此外，将通过人为改变机器负载（mem，cpu）的方式，或人为进行中断干扰，突然干掉某些节点等，验证系统的稳定性。

如图 5-8 所示。该调用为每秒处理的 getService 调用（即用户调用）

测试环境为：

（1）100 个用户连接；

（2）每个用户发送 10000 次请求。

结果为：

每秒处理 2811 次调用。且系统运行正常。

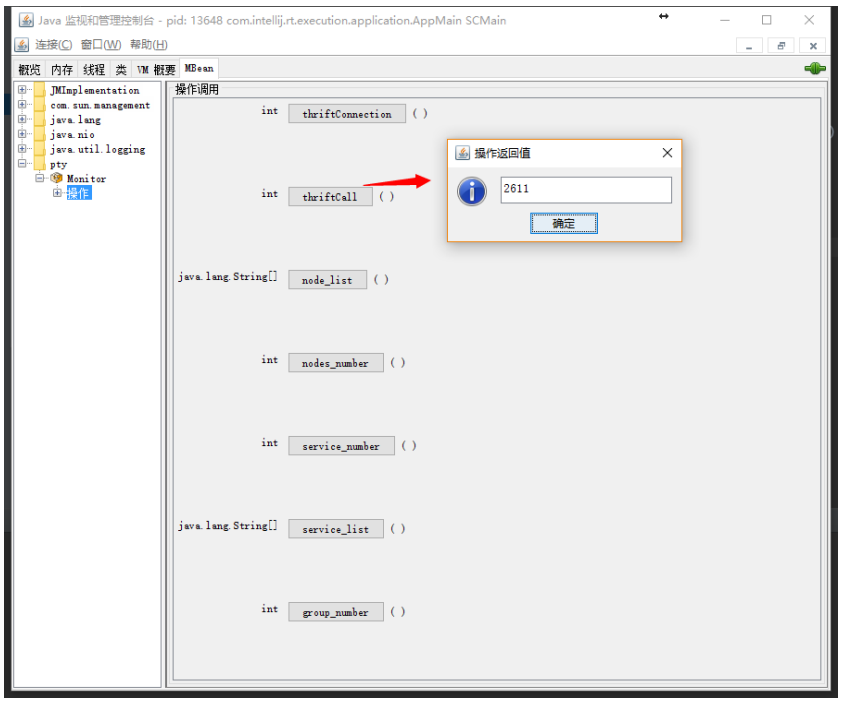

图 5-8 100clients 10000request test

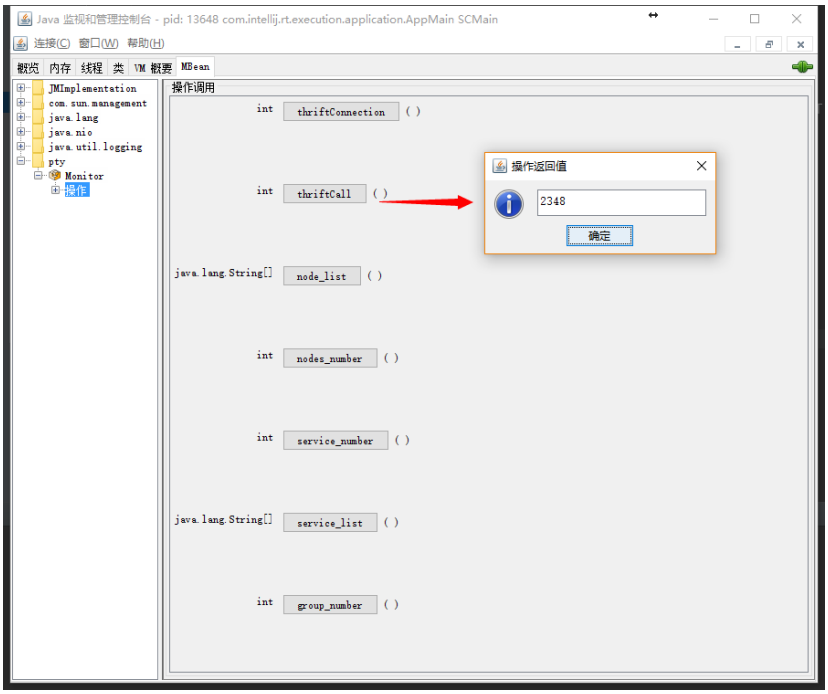

图 5-9 2000clients 10000requests

如图 5-9 所示。该调用为每秒处理的 getService 调用（即用户调用）。

测试环境为：

（1）2000 个用户连接；

（2）每个用户发送 10000 次请求。

结果为：每秒处理 2348 次调用。且系统运行正常。仅在初次调用时有偶发调用失败。当调用由无间断调用改为间隔 2s 时调用，偶发调用失败现象也消失了。

5.3 本章小结

本章节对系统进行了功能测试和压力测试。功能测试全部通过。

压力测试则分别测试了 100clients，10000request，和 2000clients，10000request 两种情况。

在第一种情况下，系统轻松运行。

在第二种情况下，最终在以下测试环境：仅在初次调用时有偶发调用失败。当调用由无间断调用改为间隔 2s 时调用，偶发调用失败现象也消失了。

在这两种情况下，处理能力均为 2900-3300 处理调用数/s。

第 6 章 结论

通过本次研究，使用 Zookeeper，Thrift，Java，Python 实现了一个对云服务进行任务调度和负载均衡的系统（管理中心），能够完成云服务的基本功能如服务注册，服务注销，服务发现，服务监控，负载均衡等。并提供了完备的时间滚动 log。

本论文对时下分布式技术，云服务技术的发展进行了一定程度的分析和归纳，分析了现有云服务拟解决的问题，分布式拟解决的问题，以及云服务与分布式的关系。并介绍了一部分现有的云服务厂商和分布式系统。

同时，本论文分析了云服务的需求与难点。并给出了本论文要实现的系统的各项数据流图。并根据需求对应的数据流图，给出了整体设计，将系统从大方面划分为负载均衡，配置加载，节点管理，任务调度，日志滚动几个部分。并就这几个部分分别给出对应的模块设计。

负载均衡部分，给出了一种通过 thrift 在 Java 与 Python 之间通讯，获取机器信息的方案。虽然获取机器信息也可以通过 Java 完成（早先版本为通过 java 实现），但一来实现复杂，需要针对 Linux 和 Windows 分别适配，此外效率比 Python 低。因此在往后的版本中更换为 python 实现。

配置加载部分则直接通过 Java 的 Properties 实现。日志滚动则使用 Logback 实现。

节点管理部分原先使用了自己写的 ZKClient，该 ZKClient 只能调用 Zookeeper 的基本 API，且监听到的变化为一个节点列表，每次在接收到监听变化时，均需要进行一次列表清洗，效率很低。最后改为 Apache Curator 实现。

任务调度则通过 thrift 实现，该方案解放了客户端的实现方式。因客户端与系统的交互通过 thrift 实现，所以客户端可用任意支持 thrift 的语言实现如 python，C++，JS 等等，给开发带来了极大的灵活性。

在每个部分的详细设计中，均给出了关键代码，并附带注释。并给出了实际部署的方法及扩展方案。

最后，本文对实现的系统进行了测试，通过测试，本论文所实现的系统可以作为一个中间件，接入云服务系统中，只需要修改配置文件，即可将该系统与对应服务的 ZK 节点关联起来。

该系统实现源代码托管在 github，累积 24 次 commit。从原先的版本渐渐详实，并去除赘余代码，同时采用了 Apache Curator 替换了原有 ZKClient 实现，并通过 python 将获取机器信息的 java 部分重写。

本系统仍旧有可提升的部分，在实际业务中，往往还有以下几点可以做：

（1） 流量优化：对文件传输进行压缩，对消息传输使用序列化，加快传输效率，减轻网络带宽压力；

（2） 日志集中：将日志重定向至 redis 等消息队列，并发送至单机或数据库。便于查看；

（3） 使用数据库进行节点表，服务的存储，而非存在内存中。

因本系统并未接触实际业务，因此并未实现上述功能，该系统的初级概念来自于上家实习公司，由于现已不在该公司，因此无法录制该系统在实际业务中的表现。谢谢。

参考文献

[1] 潘志庚, 姜晓红. 分布式虚拟环境综述[J]. 软件学报, 2000, 11(4):461-467
[2] 丁滟, 王怀民, 史佩昌,等. 可信云服务[J]. 计算机学报, 2015, 38(1):133-149.
[3] 周园春, 李淼, 张建,等. 中间件技术综述[J]. 计算机工程与应用, 2002, 38(15):80-82.
[4] 王文峰, 袁庆祝, 陆佃龙. 基于 Zookeeper 综合任务调度平台的设计与应用[J]. 信息技术, 2016(6):181-184.
[5] 韩冰, 祝永志.一种基于 Thrift 的跨平台单点登录实现方法[J].软件导刊,2014(2):48-50.
[6] 薛军, 李增智, 王云岚. 负载均衡技术的发展[J]. 小型微型计算机系统, 2003, 24(12):2100-2103.
[7] Tanembaum, Andrews. Distributed System : Principle and Paradigms[J]. 2002.
[8] Foster I, Kesselman C, Nick J M, et al. Grid Services for Distributed System Integration[J]. Computer, 2002, 35(6):37-46.
[9] Bernstein P A. Middleware: a model for distributed system services[J]. Communications of the Acm, 1996, 39(2):86-98.
[10] Hunt P, Konar M, Junqueira F P, et al. ZooKeeper: Wait-free Coordination for Internet-scale Systems[C]// 2010:653–710.
[11] 陈斌, 白晓颖, 马博,等. 分布式系统可伸缩性研究综述[J]. 计算机科学, 2011, 38(8):17-24.

附录

1-1 任务调度

```
public String getService(String serviceName) {
    if(serviceMap.containsKey(serviceName)){
        if(!serviceMap.get(serviceName).isEmpty()) {
            return serviceMap.get(serviceName).getService();
        }
        else{
            LOG.warn("Service NodeList is empty");
            return null;
        }
    }else{
        LOG.warn("Unregister service");
        return null;
    }
}
```

```
Node findSuitable(ArrayList<Node> list) {
    switch (type){
        case "RANDOM":
            return list.get(random.nextInt(list.size()));
        default:
            float score = 0;
            int index = 0;
            float min_score = 10000;
            for (int i = 0; i < list.size(); i++) {
                if(score < min_score) {
                    min_score = score;
                    index = i;
                }
                score = list.get(i).getCpu() * powerCPU +
                        list.get(i).getMem() * powerMemory +
                        list.get(i).getNet() * powerNetUsg +
                        list.get(i).getDisk() * powerDisk;
            }
            return list.get(index);
    }
}
```

1-2 注册服务

```
        @Override
        public String registerService(String serviceName, String xml) throws
TException {
            if(debug){
                return test;
            }
            LOG.debug("get thrift call : register Service");
            return manager.registerService(serviceName,xml);
        }
```

```
        public String registerService(String serviceName, String jsonString)
{
            Groups groups = JSON.parseObject(jsonString, Groups.class);
            String groupsPath = groups.getPath();
            String tag = groups.getTag();
            if(!zkClient.CreateNode(groupsPath,tag,true)){
                return "failed.";
            }
            for (Group group : groups.getGroups()) {
                String name = group.getServiceName();
                String group_path = group.getPath();
                if(!zkClient.CreateNode(group_path, group.getTag(),
true)){
                    return "failed.";
                }
                for (ServiceNode service : group.getNodes()) {
                    String ip = service.getIp();
                    int port = Integer.parseInt(service.getPort());
                    String data = name + "-" + ip + "-" + port;
                    String path = service.getPath();
                    if(!zkClient.CreateNode(path, data, true)) {
                        return "some node create failed.";
                    }
                }
            }
            return "register success.";
        }
```

1-3 更新机器状态

```
    void update(){
            for (Node s : serviceList) {
                String IP = s.getIP();
                int port = 9090;
                TTransport transport = new TSocket(IP, port, timeout);
                TProtocol protocol = new TBinaryProtocol(transport);
                // 按名称获取服务端注册的service
                LoadBalanceInterface.Client client = new
LoadBalanceInterface.Client(protocol);
                try {
                    transport.open();
                } catch (TTransportException e) {
                    LOG.warn("TTransportException when isSave {}",
e.getMessage());}
                try {
                    String response = null;
                    response = client.requestServiceSituation("test");
                    if (response != null) {
                        try {
                            LOG.debug("get machine status : {}", response);
                            String status[] = response.split("@");
                            s.setMem(Float.parseFloat(status[0]));
                            s.setCpu(Float.parseFloat(status[1]));
                            s.setDisk(Float.parseFloat(status[2]));
                            s.setNet(Float.parseFloat(status[3]));
                        } catch (Exception e) {
                            LOG.warn("get machine status failed. ip : {},
port : {}", s.getIP(), s.getPort());
                        }
                    } else {
                        LOG.warn("TTransport response is null. ip : {}, port
{}", IP, port);
                    }
                } catch (TException e) {
                    LOG.warn("TException when isSave {}", e.getMessage());
                } finally {
                    transport.close();
                }
            }
        }
```

1-4 日志 xml

```
    <?xml version="1.0" encoding="UTF-8"?>
    <configuration>
        <appender name="console-error"
class="ch.qos.logback.core.ConsoleAppender">
            <target>System.out</target>
            <filter class="ch.qos.logback.classic.filter.LevelFilter">
                <level>DEBUG</level>
                <onMatch>NEUTRAL</onMatch>
                <onMismatch>NEUTRAL</onMismatch>
            </filter>
            <encoder>
                <pattern>%date %-5level [%thread] %logger[%file:%line]
## %msg%n</pattern>
            </encoder>
        </appender>
        <appender name="file_debug"
class="ch.qos.logback.core.rolling.RollingFileAppender">
            <file>log/debug_sc.log</file>
            <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
                <level>debug</level>
            </filter>
            <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<fileNamePattern>log/debug_sc.log.%d{yyyyMMdd}.zip</fileNamePattern>
                <maxHistory>5</maxHistory>
            </rollingPolicy>
            <encoder>
                <pattern>%date %-5level [%X{traceId}] %file:%line
## %msg%n</pattern>
            </encoder>
        </appender>

        <root level="debug">
            <appender-ref ref="file_debug" />
            <appender-ref ref="console-error" />
        </root>
    </configuration>
```

1-5 Thrift 调用指标的监控代码

```
@Override
public int thriftCall() {
    int old_call = TestParams.getThrift_call();
    try {
        Thread.sleep(1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    return TestParams.getThrift_call() - old_call;
}
```

谢辞

首先要特别感谢我的指导老师吴亚东老师对我的无限关怀和悉心指导。在实验室给予了我充分的时间和条件做我想做的事，在实验室我写过 C++，写过 Qt，写过 App，写过游戏，和程序结下了不解之缘。同时在我出去实习时，也给我了很有用的建议。并且让我能够自己选题完成毕业设计。

同时，还要特别感谢我的指导老师张晓蓉老师，在我的论文撰写和修改过程中，张老师给予我方方面面的指导。指出了我论文中的不足，在张老师的帮助下，论文不再是毕设中的难题。张老师还给我们组内开会，讲了许多答辩时要注意的问题，让我受益匪浅。

毕业设计的完成，不仅是我的辛劳付出，同时也倾注了指导老师的心血与关怀。在此，我向吴亚东老师和张晓蓉老师致以衷心的感谢！

同时还要特别感谢西南科技大学游戏开发团队的杨文超师兄，蒋宏宇师兄，齐锦楠师兄，周和繁师兄。周和繁师兄是我大一时的导生，那时我们就认识了，在学习方向上给了我很多有用的建议，让我少走了不少弯路。齐锦楠学长在大三引荐我加入了游戏开发团队，让我真切的参与了一次实际项目的开发。杨文超师兄则与我一起做过项目，参加过比赛。蒋宏宇师兄则在毕设完成期间参与了组内审查，给我的毕设提出了不少建议。师兄们脾气又好又热心，和他们交流很愉快，也让我学习到了很多知识。融入实验室的过程很愉快。谢谢你们。

非常感谢四年来教过我的老师们，你们的专业态度和学术素养，让我学会了许多知识，特别感谢我大三下学期以及大四上学期因为实习而导致有些课程无法完成时对我宽容的老师们，对我交大作业的时间给予宽限。是你们让我有了出去实习，见识世界的机会。

非常感谢西南科技大学计算机科学与技术卓越班这个班集体，在这个班中大家非常融洽，没有隔阂，有什么问题在群里问都会有人解答。马上毕业就要分开了，但大学这段时光，这个班集体，我不会忘记。

最后，也是最重要的，我要感谢我的父母和家人，感谢你们在这二十多年里对我的养育和教育，是你们无怨无悔的付出才使我能有如此良好的成长和教育环境，是你们给了我精神上的，物质上的支持，我也终于不负期望，在毕业之后，有了一份体面的工作。从小学、初中、高中，再到大学，求学的这些年，看着你们一点一点有了白头发，感叹岁月不如人。但不论时间如何改变，我们之间的关系与情谊不会变。