密级：　　保密期限：

北京郵電大學

硕士学位论文

题目：　对 UCON 模型的改进及其在云存储访问控制系统中的应用

学　　号：　2012111436

姓　　名：　高磊

专　　业：　计算机科学与技术

导　　师：　秦素娟

学　　院：　网络技术研究院

2014 年 12 月 31 日

独创性（或创新性）声明

本人声明所呈交的论文是本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中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京邮电大学或其他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。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名：_____ 日期：_____

关于论文使用授权的说明

学位论文作者完全了解北京邮电大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属北京邮电大学。学校有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存、汇编学位论文。（保密的学位论文在解密后遵守此规定）

保密论文注释：本学位论文属于保密在__年解密后适用本授权书。非保密论文注释：本学位论文不属于保密范围，适用本授权书。

本人签名：_____ 日期：_____

导师签名：_____ 日期：_____

对 UCON 模型的改进及其在云存储访问控制系统中的应用

摘　要

当前云计算的发展十分迅速，人们也越来越关注云的安全问题。在云上，用户能够在任何地方任何地点通过互联网获取到数据，但是这种数据共享模式也给云带来了更多的风险。同时，云环境也给授权带来一些新的挑战。传统的访问控制模型不支持在访问控制过程中进行动态授权。UCON 由于具有属性的可变性以及持续授权的特性，被称为下一代访问控制模型。但是在云环境下，UCON 也存在很多不足之处，如何设计云环境下的安全访问控制方法是云计算的重要内容。

本文分析了 UCON 在云环境中的不足，并对其进行了改进：加入角色，简化了授权规则的定义，能很好的支持主体权限的变更；加入租户，使 UCON 具有隔离性；提出一种临时属性的概念，使得规则的定义更灵活。同时，还将会话这一组件引入到 UCON 中，使模型能更直观的体现持续授权这一特性。最后，本文基于改进的 UCON 模型实现了一个基于 HDFS 的云存储系统。该系统利用服务器集群、负载均衡和分布式缓存系统来提高系统的性能，并设计了一套消息传递机制来保证系统的可扩展性。实验分析该存储系统有很好的租户隔离性和动态授权特性。

关键词： 访问控制 UCON 云存储 HDFS XACML

IMPROVEMENT OF UCON MODEL AND ITS APPLICATION IN THE CLOUD STORAGE ACCESS CONTROL SYSTEM

ABSTRACT

With the rapid development of cloud computing, a growing numbers of researchers begin to pay close attention to the cloud security issues. Users can obtain their data at anywhere any time, which is the most important features of cloud computing. However, such data sharing model has also led to higher risk in data security. At the same time，new challenges on access control emerge with this new computing environment. Traditional access control model is not support in the process of access control for dynamic authorization. Usage control is presented and has been considered as the next generation control model with features like attribute mutability and continuity of control. But in a cloud environment, UCON also has many shortcomings. It is an important thing that designing a security access control mode in a cloud environment.

We improved the attributes of UCON: making role as an attribute of subject, simplifies the definition of the authorization rules; making tenement as an attribute of subject; proposing the concept of a temporary property. In this thesis, the session will be introduced into the UCON, making the features of the UCON(continuity of control) more obvious. Finally, we also implemented a cloud storage system based on HDFS. The system uses UCON model for access control. Taking into account the case of huge number of users, we build a service cluster, use load balancing and distributed caching system to improve system performance, and design a set of message passing mechanism to ensure system scalability. The results of the experimental show that the storage system can isolate different user's data and can control the on-going access.

KEY WORDS: Access control UCON Cloud storage HDFS XACML

目 录

第一章 绪论

本章节主要介绍云存储的背景，访问控制技术的研究现状，给出了本文的研究内容和所完成的主要工作，并对层次结构划分进行简要说明。

1.1 研究背景

随着云计算和大数据的发展，云计算和大数据越来越融入我们的生活。越来越多基于云计算的应用不断的出现，在传统的商业模式下，企业需要自己购买专业的设备来满足自己的计算和存储需求，而现在只需要向云服务商购买虚拟设备即可，这极大的降低了企业的成本。

云计算中的“云”是指散布在互联网上各种资源的统称[1]。云计算是也是一种超大规模的分布式计算技术。这种技术把一个庞大的计算任务拆分成许多小的计算任务，并且把这些任务分发给集群中的机器进行计算，最后把计算结果汇总返回给用户[2]。云计算具有在开销很低的情况下还能提供超级计算处理能力的特点，这吸引着人们对云计算进行投资、研发和应用，使之不断的成熟。技术的发展能降低系统搭建的成本，减少系统故障。

一些简单的云计算技术目前已经随处可见。例如搜索引擎，用户输入少量的关键词就可以拿到大量的相关信息。云计算不仅仅只能做资源的检索工作，还能为用户提供各种计算服务，人们通过手边的终端设备和网络在几秒之内就能处理超大量的数据，得到和“超级计算机”一样效果的计算服务[2]。例如分析 DNA 的结构，进行天气预测，以及各种各样的科学计算。目前，基于云计算的应用会越来越多，也越来越贴近普通人的生活，例如现在流行的网盘，极大的方便了人们存储个人资料。

然而，在 IT 世界里，被攻击的事件总是层出不穷。当存有敏感数据的 IT 系统被攻击的事件不断的增加时，云安全就变成了大家比较关心的课题。

云是复杂的分布式系统，那么如何做才能保证其安全呢？实践证明，最佳的云安全模型是身份访问管理（Identity and Access Management，IAM）[3]。例如 AWS（Amazon Web Service），在它提供的云服务上就具有 IAM 的功能，其他的一些云服务则使用第三方的 IAM 系统。为了确保云数据的安全，必须有这样一种技术，能保证一个合法的用户在正确的时间和有着正确的理由的情况下访问云上的资源。这意味着所有用户的身份都被系统知晓，并且需要用到访问控制技术。

1.2 研究现状

传统的访问控制技术例如自主访问控制（Discretionary Access Control，DAC），强制访问控制（Mandatory Access Control，MAC），基于角色的访问控制（Role-Based Access Control，RBAC）等都属于静态授权模型，即无法在访问的过程中对访问进行授权管理。

在自主访问控制模型中，文件的权限是由文件的拥有者决定的。一般使用访问控制列表来实现此模型。这种方式实现较为简单，但是在用户量很大的时候，访问控制列表也很庞大，由其是对用户权限需要变更的情况，维护起来较为困难。

强制访问控制模型是依靠主体和客体的安全属性来对访问进行控制的。主体和客体的安全属性都只能被系统修改。该模型的安全性较高，但是缺乏灵活性，往往应用于军事领域。

基于角色的访问控制模型最主要的特点是引入了“角色”这一概念，权限被分配给角色，角色被分配给主体，主体获得到相应的权限。相对于主体来说，角色拥有的权限更为稳定，不易变化。后来又有学者对 RBAC 进行了扩展，例如提出了角色继承的概念，子角色自动获得父角色的所有权限，解决了角色之间权限重合的问题。考虑到安全问题，有学者为其加入了约束规则，包括最小权限、互斥约束、先决条件等。RBAC 虽然得到了广泛的应用，但是它仍然属于静态授权类型，无法实现动态授权。

下一代访问控制模型使用控制（Usage Control，UCON）引入了主客体属性、条件和义务这些组件，具有属性的易变性和持续授权的特点，属于动态授权模型。但是 UCON 只是一个抽象的概念模型，在具体实现的时候要结合具体的使用场景进行一些改进。例如在多租户的环境下要具备数据隔离的特性，在多用户多资源的条件下，授权规则的定义要灵活性高、扩展性强。另外，UCON 的属性易变性可以通过属性这一组件很好的体现出来，但是却没有组件来体现其持续授权的特点。

当前主流的云存储产品使用的访问控制策略都属于静态授权模型，例如 openstack 和 Google storage 使用 ACL[4,5]，Amazon 的 S3 使用自主访问控制策略[6]。

1.3 本文的主要工作

本文对访问控制技术和云计算进行了理论研究，分析了 DAC、MAC、RBAC 等这些传统的访问控制模型，指出了它们的不足之处，其中传统的访问控制模型的一个共同缺陷是：不支持动态的访问授权，无法在访问中进行授权控制。

之后本文又分析了 UCON，UCON 虽然被称作是下一代的访问控制模型，但是在云环境下仍然有一些缺陷，例如在用户量较大的时候，授权规则定义的繁琐性，以及没有数据隔离性，因此本文根据在云环境下对访问控制的要求对 UCON 做出了一些改进，引入了角色属性、租户属性以及临时属性，从一定程度上解决了 UCON 在云环境下缺陷。另外本文还给新模型引入了一个新的组件：会话。会话表示一个正在访问的过程。会话的引入，使得 UCON 能更直观的体现出持续授权的特性。

在完成基础的理论研究后，本文还设计并实现了一个基于 UCON 模型的云存储系统。该系统使用了改进后的 UCON 进行访问控制管理，因此有租户隔离特性，且支持动态授权。在实现的过程中，考虑到用户访问量较大，又使用了一些关键技术来解决性能问题，例如用缓存系统解决 I/O 操作耗时的问题，用 MQ 解决通信问题，用负载均衡解决多台服务器对外提供统一服务的问题。另外在实现的过程中，也用到了现有的一些优秀框架，比如 Hadoop 分布式文件系统（HDFS）作为文件存储的主要载体，Balana（一个 XACML 的实现）中的 PDP 策略决策引擎，MongoDB 用来存储主客体的属性。

1.4 本文的结构安排

本文一共分为五章，各章内容安排如下：

第一章叙述云存储访问控制的研究背景、研究意义，目前研究现状，以及本论文所做研究内容。

第二章介绍了传统访问控制的相关研究，对授权的理论基础即访问控制模型的发展进行了概述。本章还介绍了云计算相关技术，如 Hadoop 分布式文件系统 HDFS；介绍了访问控制实现技术，如 XACML（可扩展的访问控制标记语言）。

第三章是提出了对 UCON 模型改进的方案，并且提出了一个基于改进后的 UCON 模型的云存储系统架构。本章首先介绍了 UCON 模型，介绍了它的核心组件及其新特性。之后分析了 UCON 模型在云环境下的不足之处，并提出了几项改进，主要是针对属性的改进，将角色、租户引入到属性当中，并分析了这样引入的优点以及好处；提出了一种临时属性的概念，使得授权规则的表述有更多的灵活性；引入了会话这一组件来体现 UCON 的动态授权特性。最后的小节中提出了基于改进后 UCON 模型的云存储系统架构，并介绍了每个模块的大致功能。

第四章重点说明了系统中的关键模块以及使用到的一些关键技术，例如缓存系统的设计，使用缓存系统来提高读取属性的效率，在此模块中主要考虑缓

存的一致性问题和置换算法；在使用了多台服务器提供决策服务的情况下，使用负载均衡算法来保证对外提供统一的服务；使用通信模块来保证系统的扩展性，分别设计了同步和异步的通信机制。

第五章是总结与展望，总结了全文的工作以及提出进一步的研究方向。

第二章 背景知识介绍

本章主要介绍了访问控制和云计算相关的基础知识，对构建云存储系统的一些基础知识进行了说明，包括 HDFS 和 XACML。

2.1 云计算与云存储简介

2.1.1 云计算基本介绍

目前，云计算现在还是一种发展中的技术，还没出现一个标准的定义。

维基百科给出的关于云计算的定义是[2]：共享的软硬件资源和信息可以通过基于互联网的计算方式提供给计算机和其它设备。用户不需要了解“云”的细节信息，只要按需交费，即可使用云服务商提供的服务。

云安全联盟（Cloud Security Alliance，CSA）给出的定义是[7]：网络、“资源池”化的计算、信息和存储等组成的服务一起构成了云。类似效用计算的按需分配和消费模式，云的这些服务和组件可以被迅速的设置、部署、缩减或者扩充。

美国国家标准与技术研究院（National Institute of Standards and Technology，NIST）定义[7]：云计算作为一种按需付费的模式，该模式能够提供按需的、可用的、便捷的网络访问，使用户进入一个可配置的计算资源共享池（资源包括网络，存储，应用软件，服务器，服务），并且用户只需和服务提供商进行很少的交互就能快速获取所需的计算资源。

虽然不同的人对云计算有着不同的定义，但是从上述定义中可以得到一些共同点：云计算将集群中服务器的计算能力集合起来，通过互联网的形式给用户服务，用户只需要按需付费，因此云计算不仅仅是一种技术，也是一种服务[7]。

云计算具有如下特征[1]：

(1) 多重租赁(分享资源)。在云计算以前，其他的计算模式都需要用户拥有专有的计算资源，云计算与此不同，它是一种共享资源的计算模式，可以共享网络，主机，存储等资源。

(2) 大规模的可扩展性。云计算的另一特点是规模超大，可以使成千上万台机器一起进行工作，能够提供超强的计算能力和大容量的存储服务。

(3) 弹性。用户需要的计算资源并非是一成不变的，云计算具有很好的弹性，允许用户增加自己的资源，当用户不需要时，就把资源释放，从整体上提高了

资源的利用率。

(4) 随用随付，按需付费。在以往，当用户需要进行大量的计算时，就不得不购买昂贵的设备。而在云计算中，用户只需要为他自己使用的资源付费，这样就大大降低了用户的开销。

当前社会，技术的变革日新月异，云计算技术也会不断的变化，文献[8]上指出了云的发展趋势：

(1) 在移动浪潮推动下，云计算将无处不在。现如今，手机、平板电脑的用户数量增多，在这种市场的需求之下，云服务提供商们会向移动端发力。

(2) 第三方云平台的数量将会增长。如同过去的操作体统和浏览器之间的争夺，将来会出现一些优秀的第三方应用程序，能很好的满足用户需求。这些第三方的应用程序能很好的促进云计算平台的增长。

(3) 新的价格战不可避免。目前的云计算产业，处于新一轮价格战的前夕。由于亚马逊对云计算研发较早，因此亚马逊在云服务领域具有一定的优势。但微软与谷歌一直在关注这个领域，这些 IT 巨头的加入，会引发起新一轮的竞争。

(4) 云计算的社会化特征增强.目前云计算正在想社会化模式靠拢。通过云端应用发布的数据，不仅可以用电子邮件来分享，而且能够满足社会化媒体的需求，例如 Facebook 和 Twitter。

(5) 云存储将成为主要的服务趋势。虽然在云计算的基础上可以提供很多服务，但是云存储越来越成为一个主要的云计算服务。Dropbox 和它的竞争对手们，可以在任何时间通过网络为企业提供存储服务。当前在云存储上的一个主要问题是安全问题，但是随着云安全情况的改善，云存储会变得更加普遍。

2.1.2 云存储基本介绍

云存储实际上是云计算提供的一种服务，把分布在互联网上的不同类型的设备集合起来，对外提供大容量的数据存储服务[9]。云存储用到了多种技术，例如：网格技术，分布式文件系统等。随着网络技术的发展，云存储越来越成为云计算众多服务当中最主要的一个[10]。

云存储具有如下优势[11]：

(1) 成本低廉。对于云存储的使用者来说，他使用的并非某个具体设备，而是一种虚拟的机器，这些虚拟的机器运行在由成千上万台机器组成的集群之上。对于使用者来说，有存储需求的时候，无需另行购买昂贵的专业的存储设备，只需要在云上购买容量即可，减少了成本投入。

(2) 存储智能化，使用效率更高。云存储使用到了虚拟化技术，这样能很好的解决存储资源利用率的问题，用户并非每时每刻都全部占用自己购买的全部空间，虚拟化技术可以很好的分配存储空间，保证资源不被浪费。

虽然云存储越来越成为一种重要的云计算服务，但目前它还存在以下几个问题[12]：

(1) 数据检索效率。由于存储系统中往往是存放数以 PB 级别的数据，存储设备更是成百上千台，在某些场景下，快速的检索到需要的文件也是一个重要的问题。例如在海量视频应用中，文件的数量非常之多，传统的文件系统是根据文件目录来定位文件，但是在分布式的系统下，目录和文件的数量是非常巨大的，这种方式不能提供一个高效率的检索服务。

(2) 网络带宽。前面已经提到，云存储是运行在集群之上的，用户是通过互联网的形式接入到云存储系统当中的，只有网络带宽足够，用户才能大量的下载或者上传数据，实现大容量数据的传输，这样用户才能真正享受云存储服务。

(3) 云上的安全问题。从云计算开始发展之初，安全一直是需要考虑的问题，同样，在云存储方面，安全问题也不容忽视。许多用户在云上放自己的敏感数据，不希望别人获取到，因此希望云服务提供商能保证系统的安全性能。目前，一些大型的云存储厂商正在努力提高自己产品的安全性，一些云存储产品提供的安全性水平要比大多数用户自己建立的数据中心要高。

(4) 基于云存储应用的发展。云存储系统给用户只能提供存储服务，这对普通人来说意义并不大。现在有许多第三方的应用是建立在云存储服务至上，这些应用才能真正的方便用户。比如视频网站的所有视频就放在云存储系统上，为用户提供视频播放服务。

多租户是指许多企业共享资源，包括软件、硬件、网络等[12]。所有的企业都把数据放到了共享的数据平台中，使用一套基础设施。但是每个企业都只被允许访问自己的数据。多租户模式包含以下几个特点[13]：

(1) 服务托管：企业只需要为自己所使用的服务付费，服务提供商负责应用、软件和硬件的安全问题进行维护和管理，例如防止一些网络攻击，保证数据完整性的措施等；

(2) 共享性：在多租户的环境下，所有的资源都是共享的，这种资源共享的模式有利于提高资源利用率，也降低了成本；

(3) 可配置性：每个企业都可以按照个性化的需求来配置策略，从而获得符合自己要求的服务；

(4) 安全性：存放在共享数据平台的数据一定要确保安全，不能被他人非法的窃取或者篡改。

多租户的模式大大降低了企业的信息化管理的开销，给传统的软件行业带来了巨大的变革，企业通过互联网就可以使用服务提供商提供的服务，不需要购买大型的专业的设备就可以使用到快速的服务。同时企业也节省了传统模式

中维护设备的开销，服务提供商具有专业的维护团队，这样就使企业的人力、物力和财力都得到了大大的节省。

在访问控制方面，相比于传统的软件模式，多租户模式的环境相对复杂，每个租户都有独自的、个性化的访问控制需求策略，因此多租户模式下的访问控制不仅需要保证访问控制策略的安全性，同时也必须考虑多租户的访问控制模型的灵活性和可靠性。

2.2 访问控制相关研究

传统的访问控制技术，主要有两种思路去实现[2]：

第一、访问控制列表（Access Control List，ACL）。ACL 是以客体为出发点，为每一个客体都附上了一张列表，列表上标明了某个主体有什么样的权限。当某个主体访问此客体的时候，如果列表上写明了该主体具有相应的权限，那就允许访问，否则拒绝访问。

第二、Capability。这种思路是以主体为出发点，主体持有 Capability。Capability 中定义了每个主体对客体有着什么样的权限，当该主体发起访问的时候，就检查主体的 Capability，如果有对应的权限，那么就准许访问。

这两种思路在不同的场景下有不同的应用，本文在此给出两种思路的不同之处[2]：

第一、认证方面。如果使用的是 ACL 的方式，则必须要有认证的过程。因为访问控制列表上面只写了主体所拥有的权限，所以系统必须知道访问者是哪个主体。而采取 Capability 方式就不比如此。

第二、授权操作。ACL 中的授权操作必须在客体访问的时候进行，只有主体访问某个客体，才能完成授权。而对于 Capability 这种方式来说，授权操作是独立的，没有必要直到主体开始访问客体的时候才进行授权，在主体开始访问之前我们就能决定该主体有什么样的权限。

第三、安全保障。对于 ACL 这种方式来说，必须保护好访问控制列表，如果某个主体能对某个客体的 ACL 加以篡改，那就极不安全。对于 Capability 方式来说，重点要确保主体的身份以及 Capability 的正确，因此需要签证和加密技术，要使得冒充其他主体或者篡改 Capability 非常困难。

下面介绍一些传统的访问控制模型。

一、自主访问控制模型

自主访问控制是指资源的拥有者对资源进行权限管理，这种访问控制方法是自主的，由资源的拥有者来决定谁能访问该资源，有何种权限[14]。

在 Linux 当中使用自主访问控制来实现对文件的权限管理[15]。对每个文件

都有一个 9 位的二进制数字来表示系统中用户对该文件所拥有的权限，每 3 位为一组，表示不同种类用户的权限，分别是：文件所有者的权限，同用户组的权限，其他非本用户组的权限。一组中的 3 位数组，每一位都代表了一种权限，分别是读、写和执行。使用这种实现方式的特点是：可以很容易的得到每个用户对某一文件的权限，但是很难得到某一用户对所有文件的权限，而且当用户数据量较大的时候，无法实现细粒度的权限管理。

实现自主访问控制的另一种方式是访问控制列表 ACL，ACL 使用了一个二维矩阵表示主体对客体的访问权限。矩阵中的每一列则代表了系统的客体（文件，资源等）[16]，矩阵中每一的行分别对应对于系统中的每一个主体（如用户、程序等实体），矩阵中每个元素代表该行的主体相对于该列中的客体的许可访问权限。使用这种实现方式，当文件数量和用户量都比较多的时候，矩阵列表也变得很庞大，维护起来相当困难。

二、强制访问控制模型

强制访问控制是一种系统强制主体服从访问控制策略。系统的主体（用户和其他主体）与系统客体（文件，数据等）都被标记了固定的安全属性（例如安全等级、访问权限等）[16]，在用户发起对某个客体的访问时，系统通过比较主体的安全级和客体的安全属性，从而决定某个客体能否被某个主体访问。这些安全属性是被系统强制加上的，主体无法修改自己的安全属性，也无法修改客体的安全等级。

MAC 的优点在于它的安全性能较好，系统中的主体、客体都无法改变自己的属性，可以阻止一些人为引发的恶意行为[17]。但是在实际应用中，MAC 无法给出一个灵活的访问策略。

三、基于角色的访问控制模型

在 RBAC 模型中，与其他访问控制模型的很大不同之处在于它具有“角色”的概念。角色相比于主体而言，它的权限更为固定，不易发生变化[18]。在自主访问控制模型中访问权限是直接分配给主体，而在 RBAC 中，主体先被分配某个角色，角色拥有权限，主体通过角色获得权限[18]。

Sandhu 等人先提出了一个简单的较容易理解的 RBAC 框架[20]，后来得到不断完善的模型 RBAC96[21]，该模型其实是一系列的访问控模型。RBA96 实现起来较容易，范围覆盖较广，是一个模型家族。RBAC96 的出现使 RBAC 在访问控制模型中具有了绝对优势地位。

实际上 RBAC96 模型有四种子模型：$RABC_0$，$RBAC_1$，$RBAC_2$，$RBAC_4$。

$RBAC_0$ 是 RBAC96 模型家族中支持基于角色访问控制模型的最小结构模型，它没有考虑约束条件和角色继承的概念[22]。

$RBAC_1$ 引入了角色分级的概念，即角色是分等级的，子角色可以自动继承父角色的所有权限。所有的角色就形成了一棵角色树，这样极大的方便了对角色的授权管理，并且很好的解决了对角色与角色之间权限重叠的问题[23]。

$RBAC_2$ 则更多的考虑了授权管理中的安全问题，加入了约束规则。$RBAC_2$ 主要有以下约束规则[24]：

(1) 最小权限。每个用户只能得到满足其需要的最小权限集，这条规则是为了防止权限滥用。

(2) 互斥约束。如果两个角色之间存在互斥关系，则用户只能得到其中一个角色，例如一个用户不能同时用户出纳和审计两种角色。权限与权限也存在互斥关系，一个角色不能同时拥有存在互斥关系的权限，例如对某文件的修改和审核权限不能交给同一个租户。

(3) 基数约束与角色容量。基数约束使指需要一种安全策略，来限制分配给一个用户的角色数目以及一个角色拥有的权限数目。角色容量使指一个角色被指派给的用户数量是受到限制的。

(4) 先决条件。这条规则对用户获得角色或者角色获得权限进行了限制。例如某个用户在获得某个角色之前必须先获得另一角色，某个角色想获得某个权限之前必须先得到另一权限。

$RBAC_4$ 既包含 $RBAC_2$ 的结构又包含 $RBAC_1$ 的结构，它既支持角色继承也支持约束条件。

2.3 HDFS 简介

HDFS 是一个可以运行在廉价机器上的分布式文件系统[25]。HDFS 有良好的容错性能，从一开始就被设计成可以部署在多台低成本的硬件上。HDFS 能提供高吞吐量的数据访问服务，适用于具有大数据集的应用程序。

2.3.1 数据块与集群节点

在文件系统的实现中，为了便于管理，设备往往将存储空间组织成为具有一定结构的存储单位。例如磁盘，文件是以块的形式存储在盘中，文件系统通过一个块大小的整数倍的数据块来使用磁盘。磁盘上的数据块管理对于用户来说是透明的。

HDFS 也有块的概念，不过是更大的单元，默认的 HDFS 数据块大小是 64MB[26]。和普通文件系统累死，HDFS 上的文件也进行了分块，块作为单独的存储单元，以 Linux 上普通文件的形式保存在数据节点的文件系统中。数据快是 HDFS 的文件存储处理的单元。

HDFS 的设计基于主从结构，具有一个主节点，用来记录和维持系统内的

所有文件，这个主节点称为 NameNode。它并不存储真正的文件的数据内容，只存储文件的元数据信息。真正的数据内容存储在从节点 DataNode 上。在 DataNode 上，文件以数据块(block)为单位来存储，每个数据块的大小一般是 64MB。NameNode 管理着整个文件系统的命名空间，控制客户端对文件及目录的访问操作，它将数据分散到各 DataNode 上，并将相关的映射信息记录下来，这些映射信息被记录在元数据之中。DataNode 则负责存储数据，并给客户端提供读写服务。HDFS 的架构如图 2-1 所示。

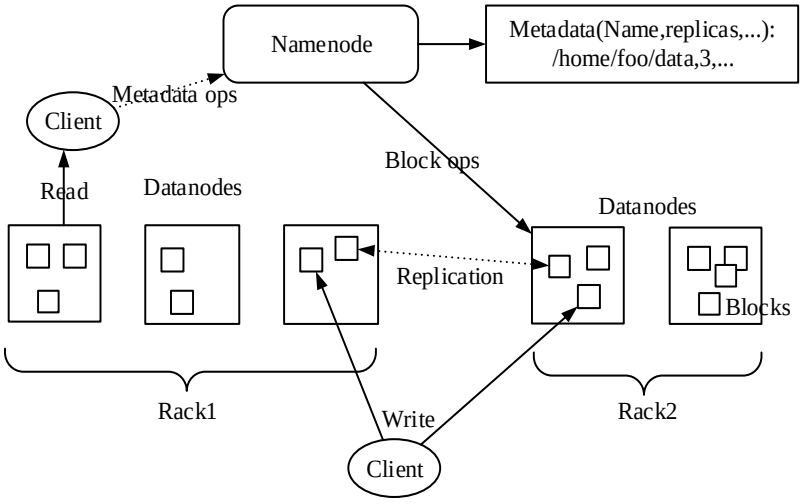

图 2-1 HDFS 架构[26]

HDFS 是基于 Java 语言开发的，任何具有 JAVA 运行环境的机器都可以运行 HDFS。由于 Java 语言有高度可移植以及跨平台的特性，HDFS 可以部署在很多不同系统的机器上。一个典型的部署示例是：一台专用机作为 NameNode，集群中的其它机器都是 DataNode。单 NameNode 的设计，使得集群的系统结构设计极大地被简化了。系统中的文件内容即不保存在 NameNode 上也不会流经 NameNode，客户端直接与 DataNode 交换数据。

DataNode 和 NameNode 之间有许多数据交互，例如 DataNode 在启动时会向 NameNode 进行注册，进入了正常状态之后会向 NameNode 发送心跳信息等。

DataNode 正常启动的时候，会向 NameNode 进行版本查询，用来保证他们之间的版本号一直。DataNode 正常启动了之后，会向 NameNode 发送注册信息，成为该 NameNode 的一个成员节点。注册成功之后，DataNode 会把它自身的所有数据块的信息上传给 NameNode 上，这样是为了帮助 NameNode 建立 DataNode 到文件数据块之间的映射。这些都结束之后，DataNode 对外提供正式的存储服务。

DataNode 在正常的运行期间，每隔一段时间要向 NameNode 发送一个心跳信息，表示自身处于正常的工作状态[27]。如果 NameNode 很长一段时间没有收

到某个 DataNode 的心跳信息，就会认为该节点已经失效。NameNode 也会想 DataNode 发送一些指令动作。例如如果删除一个文件时，在 NameNode 把该文件对应的数据块进行标记，如果存有该数据块的 DataNode 发送来心跳信息，NameNode 就会把删除指令发送给该节点。DataNode 收到指令后，就会删除数据块，把存储空间释放出来。

2.3.2 事务日志以及文件系统镜像

当客户端执行的每一个修改 HDFS 的目录树的操作后，NameNode 都会在 Editlog 写下日志，记录下此次操作，这样可以保证系统出现故障后恢复到正常状态。HDFS 能够按照日志的内容进行系统的恢复，方法是依次执行日志中的操作。但是 Editlog 文件的大小会随时间的变化而不断的增长，一旦发生了系统重启，根据日志来恢复系统的时间就很长。为了解决这个问题，HDFS 引入了 Fsimage（系统空间镜像），它是 HDFS 系统目录树以及元数据的一个存档，也称作是一个检查点（CheckPoint），每隔一段时间，系统就会更新 Fsimage，使其能保持较新的目录树和元数据。Fsimage 和 Editlog 一起解决了恢复系统较慢的问题。一旦系统需要恢复，首先把 Fsimage 的内容全部读取出来，然后在从 Editlog 中找到检查点时间之后的操作，这样就可以在较短的时间内恢复系统。

2.4 XACML

XACML，可扩展的访问控制标记语言，是由 OASIS 组织提出的一个标准[28]，它规定一个策略语言和一个访问控制决策请求/响应语言的标准格式，这两种语言都是基于 XML 书写的。其中策略语言用来描述一般的访问控制需求，以及一些标准的扩展点，用来定义新的功能，数据类型和组合逻辑等。使用请求语言来构建一个标准的查询请求，去询问是否允许执行特定的动作，响应语言用来解释系统返回的结构。响应当中包含了是否通过该请求的回答，有四种值：允许，不通过，无法确定（有错误发生或者请求当中的某些值丢失了，系统无法给出决策结果），不适用（服务器无法处理该请求）[29]。

如果有用户想对一个资源执行某个动作（读取，写入），其中资源是被保护起来的。如果使用 XACML 来做访问控制，其步骤如下：PEP 先按照用户发的请求属性生成一个标准格式的请求。PEP 把此请求发送给策略决策点 PDP，PDP 将会解析该请求，并寻找适应于该请求的策略，然后根据策略产生响应，响应中包含了此次决策的结果。之后响应就被返回给 PEP，PEP 根据响应的内容来准许或是拒绝用户的访问动作。其中 PDP 和 PEP 可以在同一个应用当中，也可能分布在不同的服务器上。XACML 中除了请求/响应语言和策略语言之外，还有一些其他的组件，例如找到一个适用于给定请求的策略，计算出该请求是

否满足策略中的规则。

XACML 有以下特点[30]：

(1) 它是一个标准。如果使用一个标准语言，那么这个语言一定被一些大型社区的专家和用户审查过，因此不会遇到那些能需要你的系统不断回滚的错误，也无需设计一个新的语言来解决自己遇到的一些棘手的问题。另外，随着 XACML 被越来越广泛的使用，它将更容易的与其他语言构建的应用交互。

(2) 它是在任何环境下都通用。它不仅仅能够为某些特定的环境下特定的资源提供一个访问控制，它能适应任何环境。策略被编写好了之后，可以被用到许多其他不同的应用中。策略的书写都使用共同语言的话，那么策略的管理将更加的容易。

(3) 它是分布式的。这意味着一个策略可以引用另一个在其他位置上的策略。这样的好处是不必要维护一个单一的策略，只需要维护一个整体的框架，不同的人或者不同的组织维护他们自己的子策略。XACML 知道如何正确的生成一个完整的策略。

(4) 它的表述能力很强大。虽然 XACML 提供了很多的扩展点，但是实际上在许多情况下不需要扩展，因为它本身提供了各种各样的数据类型，函数，和规则，策略就是由这些组件构成的。另外，以及由许多团队致力于扩展和配置文件，使 XACML 和其他的标准（SAML，LDAP）进行连接，这样就会增加 XACML 语言的使用方式。

下面介绍一下 XACML 当中的重点概念[30]：

一、Policy 和 PolicySet

所有的策略文件的一个根节点都是 Policy 或者 PolicySet。PolicySet 是一个容器，它可以包含其他的 Policy 或者 PolicySet，同样也可以引用远程位置上的策略。Policy 是指一个访问控制策略，它是通过一组规则来表示的。每个 XACML 都至少包含一个 Policy 或者 PolicySet。

由于一个 Policy 或者 PolicySet 都包含多条规则或者是多个 Policy，每一条规则或者每一个 Policy 最终做出的决策结果可能会不一样。因此 XACML 需要一个把所有结果整合到一起的方式。Combining Algorithms 可以很好的解决这个问题。每一个算法都代表了一种把多个结果组合成一个结果的方式。有两种 Combining Algorithms，PolicySet 会使用 Policy Combining Algorithms，而 Policy 会使用 Rule Combining Algorithms。例如，Deny Overrides Algorithm，如果使用这种算法，在多个结果中，只要有一个结果是 Deny（否决），那么最终的结果就会是 Deny。这些 Combining Algorithms 被用来构建成越来越复杂的策略，XACML 提供了七种标准算法，用户也可以自定义一些 Combining Algorithms

来满足自己的要求。

二、Target

PDP 的工作当中，需要找到一个适用于给定请求的合适策略。为了做到这一点，XACML 规定了 Target 节点，一个 Target 由主体，资源和动作组成。在 PolicySet、Policy 或者 Rule 中都包含 Target 节点，用于判断是否适用于某个请求。它们使用一个布尔函数去比较请求和 Target 中的值是否匹配。如果一个请求与 Target 中所有的条件都匹配成功，那么说明这个 PolicySet、Policy 或是 Rule 适用于这个请求。Target 除了是一种检测适用性的方法，它还是一种索引策略的方法，这十分有利于在大量的策略需要快速的筛选到用户所需要的策略。例如一个 Policy 中包含的 Target 可能只匹配来自某个特定服务的请求，当这种访问请求到达时，PDP 知道从哪里可以找到可能适用于该请求的 Policy。另外，一个 Target 可以被规定成适用于所有请求。

一旦一个 Policy 适用于某个请求时，该 Policy 的 Rule 将会被评估计算。一个 Policy 可以包含任意数量的 Rule，Rule 包含了 XACML 策略的核心逻辑。Rule 当中最重要的是 Condition，Condition 是一个布尔函数，如果 Condition 的评估计算结果是真，那么 Rule 就会返回其 Effect，Effect 的结果又四种情况，Permit（准许），Deny（拒绝），Error（错误），NotApplicable（Condition 并不适用于该请求）。一个 Condition 可能是非常复杂的，可以由非布尔的函数或属性嵌套而成。

三、Attribute、Attribute Values 和 Functions

属性是一些已知类型的值，例如一个 ID 号或日期时间。一个访问请求由以下几个元素构成：主体，客体，动作，环境。而属性代表了这些元素的特征，例如用户名，文件名或是文件的创建日期。一个从 PEP 发送到 PDP 的请求当中，几乎全部都是属性值，这些属性值被用于 PDP 的决策过程。

PDP 可以通过两种机制来查询请求中的属性值：AttributeDesignator 和 AttributeSelector。如果使用 AttributeDesignator 方式，PDP 会给出一个属性具体名称和一个具体的值，之后会从请求中查找匹配的特定属性，有四种类型的 Designator，分别是 SubjectAttributeDesignator、ResourceAttributeDesignator、ActionAttributeDesignator 和 EnvironmentAttributeDesignator。而另一种方式 AttributeSelectors 则允许 PDP 通过 XPath 查询的方式在请求中寻找属性值。XACML 提供了一个数据类型和 XPath 的表达式，可以用来寻找请求当中的属性集合。

AttributeDesignator 和 AttributeSelector 可能会返回多个值，因为 PDP 有可能解析了多个请求，所以 XACML 提供了一种特殊的属性类型 Bag。Bag 是一

个允许重复值的无需集合，用于存放 Designator 或者是 Selector 的返回结果。有可能最终的匹配结果只有一个属性，这种情况仍然会返回一个 Bag 类型的值。另一种情况是无匹配结果，这种情况会返回一个空的 Bag，Designator 或者是 Selector 还可以只返回一个 Error 标志。

一旦返回的 Bag 是非空的，那么 Bag 当中的那些属性值就检查是否与要求的属性值相等，然后 PDP 会做出决策。比较的过程由一些函数来完成。这些函数可以处理任意类型的属性值，其返回结果也可能是任意类型的属性值。函数有可能是嵌套的，一个函数可能会处理另一个函数的返回结果，这样就用户可以写出一些功能强大的函数。

2.5 本章小结

本章首先介绍了云计算和云存储的相关知识，云存储是云计算技术的产物，能够提供大容量的数据存储服务。近年来伴随着云计算技术的不断发展，有许多安全问题随之而来，许多大企业的云产品都有安全问题的发生。之后介绍了一些传统的访问控制模型，DAC、MAC 和 RBAC，分析了他们的特点以及不足之处，这些传统的访问控制模型都不能很好的适应云环境。第三小节介绍了 HDFS，它是一种分布式的文件系统，能提供大吞吐量存取数据的服务。最后介绍了 XACML（访问控制标记语言），介绍了它的特点以及核心概念。

第三章 基于改进 UCON 模型的云存储系统设计

本章主要分析了 UCON 在云环境下的不足之后，并针对这个问题，通过对 UCON 引入色、租户等属性，提出了一个改进的 UCON 模型。改进后的模型能更好的满足云存储系统下动态授权的访问控制需求，简化了授权规则的定义并且具有租户隔离性。最后基于改进后 UCON 模型设计了一个云存储系统，并详细分析了核心模块的功能。

3.1 UCON 模型

UCON 模型如图 3-1 所示：

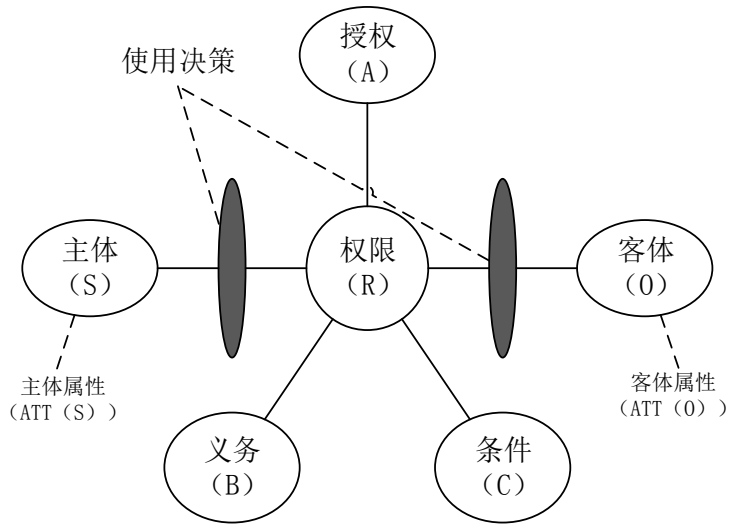

图 3-1 UCON 控制模型[31]

$UCON_{ABC}$ 模型(Authorizations，oBligations and Conditions)与传统访问控制模型不同之处在于它不仅有授权规则，而且还包含了义务和条件。UCON 核心模型由八种元素组成：主体，客体，权限，授权规则，义务，条件还有主体和客体的属性[32]。授权规则、条件和义务这三个因子共同决定了访问的决策结果。

下面分别给出每个组件的描述[31]：

主体（Subject）：主体是一个实体，简记做 S。主体可以是一个人，也可以是一个进程，它能对客体执行某些操作。

主体属性（Subject Attribute）：主体属性表示了主体的特性，简记为 ATT(S)。属性往往在授权规则中被用到。例如用户所在部门、职位、角色等等。

客体(Objects)：客体是一种具体的资源。简记为 O。例如文件系统中的文

件就是一种客体，它可以被用户读、写或是执行。

客体属性(Object Attributes)：客体属性用于表示客体的特性，简记为 ATT(O)。对于一个文件来说，它的属性可能如下：文件类型，创建时间，拥有者，是否属于机密数据等等。在授权规则的定义当中，需要用到这些属性。

权限(Rights)：它是主体拥有的对客体的一些操作权利，简记为 R。例如一个用户可以读取一个文件的内容，就称该用户对此文件拥有读权限。

授权规则(Authorization)：描述了一些规则，访问必须满足这些规则才能得到授权，简记为 A。授权规则包含主、客体属性和所请求的动作。授权之后，此次访问的主体和客体的属性也可能发生变化，影响下次的授权结果。

义务(oBligations)：义务是主体访问前或者访问过程中必须完成的动作，简记为 B。根据不同的决策过程，义务分为两种：使用前义务，也有使用中义务。使用前义表示主体访问客体前必须完成的动作，使用中义务表示的是在主体访问客体的过程中必须执行的动作。

条件(Conditions)：条件是系统的环境因素，简记为 C。这种因素与主客体是无关的，条件可以描述环境的一些属性和上下文信息。

UCON 与传统访问控制模型的最大的不同之处是它有两个新特性[31]：在访问过程中能持续授权和主客体属性的易变性。

持续授权是指对于那些主体正在对客体进行的访问，会持续不断的被系统检测，如果检测授权不通过的话，此次访问将会被撤销。

易变性是指主客体属性在访问时会发生变化。在主体开始访问客体之前，其属性值就发生了改变，这种情况称为属性的使用前更新。主体在访问客体的过程中，属性值发生了改变，这种情况称为属性的使用中更新，这些改变会被系统检测到，并重新检测是否还满足授权规则，如果不满足的话，此次访问可能会被撤销。属性的改变发生在使用过程之后，这种情况称为使用后更新，对下次主体访问客体的授权结果产生影响。

如上所述，UCON 模型引入了三个决策因子：授权(A)、义务(B)和条件(C)，因此也被称为 $UCON_{ABC}$ 模型。$UCON_{ABC}$ 并不是一个单一的模型，而是一个模型族。因为 $UCON_{ABC}$ 模型具有持续授权特性和属性易变性，UCON 对访问权限的决策可以发生在使用前或使用中，本文用 pre 表示使用前，用 on 表示使用中，preA 就表示在使用前按照授权规则来进行决策，onC 就表示在使用中按照条件来进行决策。UCON 中主客体的属性更新可以发生在使用前、使用中、使用后。本文用“0”表示在决策过程中没有属性更新，用“1”表示在使用前更新，“2”表示在使用中更新，“3”表示属性更新发生在访问之后。

表 3-1 展示了 UCON 模型族，有 24 种单模型，其中某些模型在现实中不

可能出现，这种模型就记为“N”，例如使用条件做决策的模型来说，属性的更新就无意义；对于另外一些模型则是现实中可能发生的，记为“Y”。

表 3-1 UCON 基本模型[31]

属性 / 决策	0（无更新）	1（使用前更新）	2（使用中更新）	3（使用后更新）
preA	Y	Y	N	Y
onA	Y	Y	Y	Y
preB	Y	Y	N	Y
onB	Y	Y	Y	Y
preC	Y	N	N	N
onC	Y	N	N	N

表 3-1 中列出的是 UCON 的各个单模式的情况，但是在实际系统中可能使用组合模型，如 $UCON_{preA1}$ 表示在使用前按照授权规则进行授权，并且主客体的属性需要更新；$UCON_{preC0onA2onB2}$ 表示在使用前用按照条件决策，在使用中按照授权规则和义务进行授权，并且在授权的过程主客体的属性是可变的。图 3-2 详细描述了 UCON 模型的各种组合。

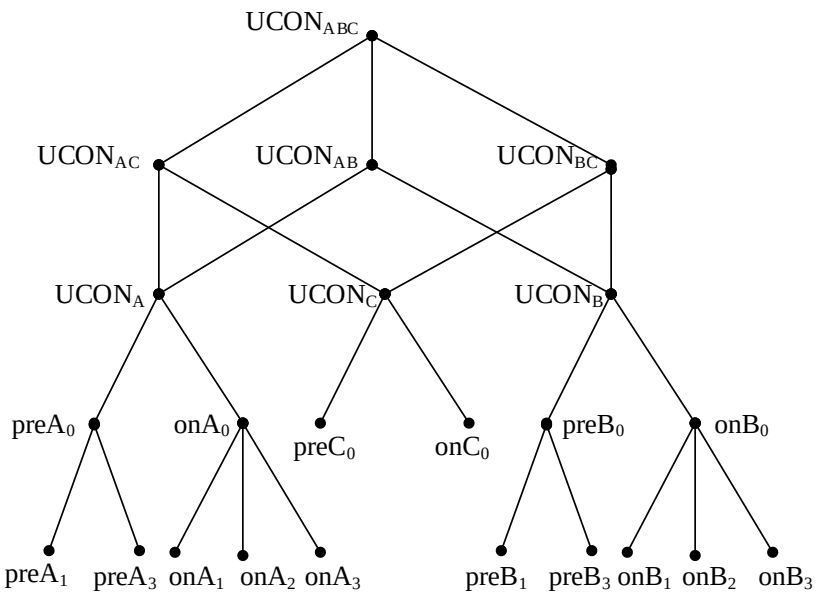

图 3-2 $UCON_{ABC}$ 模型的组合[31]

3.2 UCON 存在的问题以及针对属性的扩展

UCON 虽然能够支持动态授权，但是它仅仅是个抽象的模型，只是给出了一个理论上的框架。实现复杂，不易应用和管理。尤其是在云环境的具体应用

下，有以下不足之处：

(1) 云环境的主客体数量较多，授权规则的定义会十分的琐碎，并且一旦主体的权限发生变化，授权规则需要重新定义；

(2) 云环境有很多用户，不同用户之间的数据不能相互访问，需要访问控制模型有一定的隔离性。

因此，针对云环境下对访问空中模型的需求，本论文对 UCON 的属性做出了改进提出了 TR-UCON 模型：

(1) 将角色引入到 UCON 中，作为主体的属性，简化了授权规则的定义，也能很好的支持主体权限的变更；

(2) 将租户引入到 UCON 中，作为主客体的属性，使得 UCON 具有隔离性；

(3) 在 UCON 的主体中，引入临时属性的概念，使得在访问过程中具有更强大的授权规则。

3.2.1 将角色作为属性

在云存储系统中，用户往往要存放大量的数据，主体和客体的数量都十分的巨大，这样在定义策略的时候就十分的繁琐，并且如果某个主体的权限频繁的发生变化的时候，策略也需要跟着发生变化，这样就使得云上的访问控制系统极不灵活。

RBAC 模型的一大优点是具有了角色的概念，通过引入角色实现了用户与权限的逻辑分析，具体来说，它有以下优点[22]：

(1) 引入了角色的概念，实现了用户与权限的分离，这样很大程度上使授权管理的过程得到了简化；

(2) 由于降低了授权管理的复杂性，从而也减少了管理方面的开销；

(3) 方便企业定义出灵活的授权策略，也使的企业能定义出具有伸缩性的安全策略。

但是 RBAC 也有不足之处，角色是静态的概念，既不能实现动态的授权，在用户访问的过程中撤销访问资格，也没有考虑整个授权系统所处的环境条件。

因此本文提出，将角色引入到 UCON 当中，把角色作为 UCON 中主体的一个属性，使得新模型具有 UCON 和 RBAC 共同的优点。由于新模型的主体具有了角色属性，授权策略的定义会更加方便，也具有更好的扩展性。同 RBAC96 模型中的角色一样，新模型中的属性只需要规定用户的角色属性值就使得用户具有了某个角色的所有权限，方便了授权的过程。如果有主体的权限要发生变化的时候，只需要改变用户的角色属性值，无需改变已有的策略定义。这样既减少了授权策略定义的复杂性，又提高了灵活性，同时也极大的方便了授权的管理过程。

3.2.2 将租户作为属性

在云环境下，所有的租户都共享资源。由于云环境中使用了虚拟化技术，因此每个租户都感觉自己拥有独立的计算资源（CPU，内存，带宽）。在云存储系统中，每个租户都会感觉到自己拥有独立的存储空间，但实际上所有的存储空间都在一起。这样就为云环境下租户个人的数据带来了安全隐患，有些租户可能非法的窃取或者篡改其他租户的私密数据。而 UCON 模型也并未考虑到云环境下的安全问题，并不具备租户与租户之间的数据隔离这一特性。在云环境下选择合适的访问控制模型，很重要的一点就是考虑其安全性。

UCON 只是一个抽象的模型，为了解决上述问题，本文在 UCON 中引入了另一个属性：租户。与角色属性不同的是，租户属性是主体和客体都具备的，另外租户属性是不可变属性，主体或者客体的租户属性一旦被赋值，就不会再变化。因为在实际的应用当中，租户往往代表了某个企业，这个企业内部的所有的员工都具有相同的租户属性，同样，属于这个企业的数据也具有相同的租户属性。如果允许租户发生变化的话，一个用户可能就会访问到属于其他企业的数据，这是不允许的。另外主体和客体的租户属性可以有多个值，因为一个用户可能在多家企业就职，一份数据也可以被多家企业共享。

租户属性的引入，使特定的租户的主体只能访问特定租户下的资源，从而实现不同租户下的数据隔离。当一个主体访问云上的客体时，系统会检测主体以及被访问客体的租户，如果主体与客体的租户属性值不匹配，那么此次访问将会被拒绝，如果匹配成功，系统会继续检测此次访问的主客体是否满足授权规则，义务以及条件。

3.2.3 临时属性

临时属性，表示该属性只是在某个特定条件下某个特定时间段内存在，例如某些属性只在主体访问一个特定客体的时候中存在，访问结束或者访问其他客体都没有此属性。

在 UCON 中引入临时属性，使得授权规则的定义更灵活，也具备了更好的安全性。添加了临时属性，就可以更灵活的对主体的行为进行控制，例如某系统规定，对于某个特殊文件 A，连续的阅读时长不能超过 1 小时。所有阅读此文件的主体在进行访问的时候都有如下属性：readFileATime，表示该用户开始访问文件 A 的时间点。在访问的过程中，系统会不断的检测此属性与当前的系统时间的差值，如果超过 1 小时，那就结束此次访问，被取消访问资格的主体也不再具备这个属性。

但这样一来，UCON 中就有两种授权规则，一种是主体在访问客体之前进行检测的规则，此时主体还不具备临时属性，因此这种规则中也不涉及临时属

性，另一种是在主体访问客体过程中进行检测的规则，此时主体具有了临时的属性，这种规则含有临时属性。

3.3 将会话引入到 UCON

UCON 与传统的访问控制模型很大的不同点是具有属性易变性和连续授权的特性。其中由于属性在模型当中有定义，所以能很直观的理解属性的易变性，但是对于连续授权这一特性，在模型图当中则没有很好的表现出来。

所以把会话加入到 UCON 当中，作为一个组件，来表现 UCON 连续授权的特点。

会话的定义如下：表示一个主体正在访问客体的过程，包含访问的主体 subject，被访问的客体 object，以及访问的动作 action。一个会话被创建之后，就会被持续的检测，在会话的过程中，主客体的属性都有可能发生变化，可能会不符合授权要求，一旦检测到这种情况，该会话将被取消，此次访问过程也会结束。

会话有以下几种状态：初始化，访问中，检测中，结束。在主体发起访问之后，如果满足授权规则、义务、条件，就创建一个会话，此时会话进入初始化状态，之后开始访问，进入访问状态，之后每隔一段时间，或者是主客体的属性发生了变化，会话就进入检测状态，此时访问仍然继续进行，如果检测结果不通过，则会话被撤销，接着进入结束状态，如果通过，则访问会继续进行下去，一直到主体访问结束。状态转移过程如图 3-3 所示。

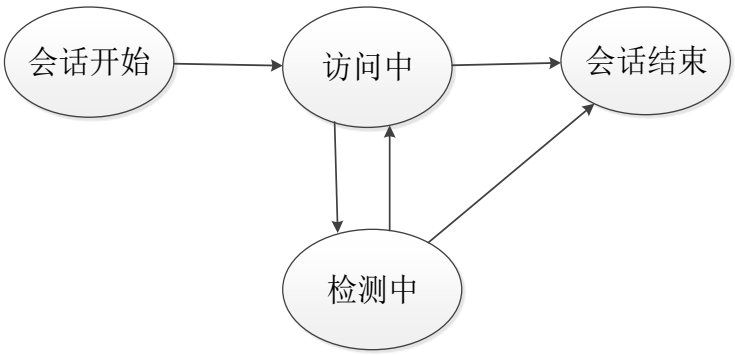

图 3-3 会话状态转移图

3.4 改进的 UCON 模型描述

本文提出了一个改进后的模型：TR-UCON，其模型图如图 3-4 所示：

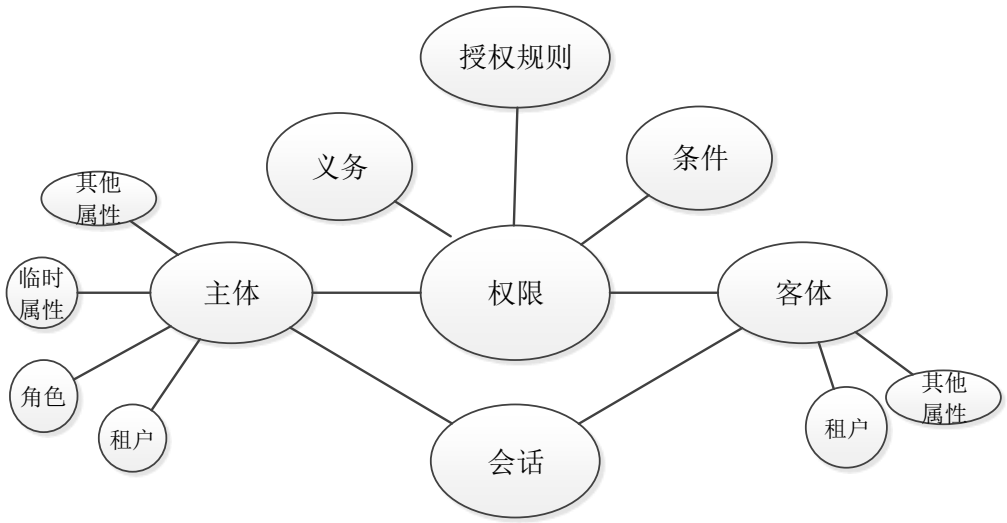

图 3-4 TR-UCON 模型图

该模型包含如下组件：主体 Subject、客体 Object、权限 Right、主体属性 ATT(S)、客体属性 ATT(O)、义务 oBligation、条件 Condition、授权规则 Authorization、会话 Session。

文献[31]中详细的给出了关于 UCON 各个单模型的描述。TR-UCON 模型的逻辑描述与 UCON 模型的基本相同，在本文中给出一些关新组件的说明：

主体的属性包括四部分：角色属性、租户属性、临时属性、其他属性；

客体的属性包括三部分：租户属性、临时属性、其他属性。

主体的角色属性具有角色等级和约束规则的概念。子角色拥有父角色的所有权限，关系为互斥的两个角色不能分配给同一个主体。

主客体的租户属性均为不可变属性，一个主体或者客体一旦生成，租户就已经确定，不能再改变。

主客体的临时属性只在特定的条件特定的时间下存在，临时属性方便定义出更强大更安全的授权规则。

会话组件由三部分组成：主体 Subject，客体 Object，动作 Action。主体发起访问客体的请求，如果请求被通过，那么就生成会话，并且在访问的过程中该会话不断的被检测，直到结束或者授权不通过，会话就会被撤销。

另外，新模型还增加了关于会话的一些操作：

(1) createSession(subject,object,action)，表示根据主体、客体和动作来创建一个会话。如果一个主体发起对客体的访问，系统首先会检测主客体的属性是否满足授权规则，主体是否执行了义务中规定的动作，当前的环境条件是否允许访问；如果以上条件都满足，系统就会允许此次访问，并且执行 createSession 操作，创建一个新的会话，并将其记录在检测队列中。

(2) checkSession(session)，表示在访问过程中检测会话；在主体对客体的访

问过程中，系统会不断的执行 checkSession 操作，首先通过会话拿到主体的属性、客体的属性、正在执行的操作，并检测义务和环境条件，此时主体的属性和客体的属性很可能发生了变化，因此其授权结果就有可能为不通过。

(3) cancelSession(session)，撤销会话；当系统执行完 checkSession 操作之后的结果为不通过的时候，或者主体正常的结束了访问时，就执行 cancelSession 操作，撤销掉该会话。

3.5 云存储系统架构

云存储系统模型架构图如图 3-5 所示：

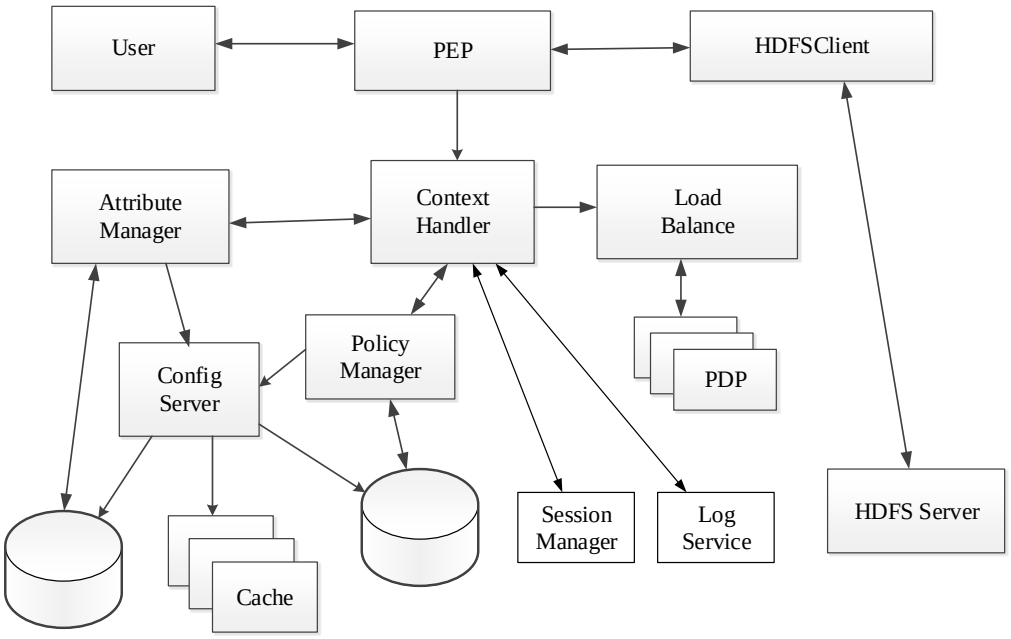

图 3-5 云存储系统模型架构图

PEP(Policy Enforcement Point)，这个模块是访问控制的入口，用户发起的访问会首先到达该模块，PEP 会按照标准格式生成请求，并将请求发给 PDP 模块，等待授权 PDP 模块的响应结果，如通过，该模块会去请求资源，返回给用户。

PDP(Policy Decision Point)，该模块接受来自 PEP 的请求，并根据请求查询到合适的策略，计算该请求是否满足策略，然后按照标准格式生成响应，并且将结果返回给 PEP 模块。该模块是整个架构的核心部分，在决策的过程中，PDP 会从属性管理中心(AM)以及策略管理中心(PM)读取数据，拿到主体、客体的属性以及相应的策略。

AM(Attribute Management)，属性管理中心，该模块管理了所有主体、客体

的属性。对外提供以下服务：属性读取，属性更新，属性添加，属性删除。

PM(Policy Management)，策略管理中心，该模块管理了所有的策略，对外提供策略读取服务。

SM(Session Management)，会话管理中心，由于 UCON 有持续授权的功能，因此需要使用会话来保存现有的所有访问。一个会话包括如下三部分：主体、客体和动作。在访问的过程中，会话会不断的被检测，并被 PDP 模块重新计算是否满足策略。

CH（Context Handler），这个模块用于存储和转发不同模块之间的通信。如果没有此模块，其余的各个模块由于要相互通信，会紧密的耦合起来，不利于扩展，因此从工程实现的角度考虑，CH 是用于解除不同模块之间的耦合。

LS（Log Service）日志服务，用来记录所有 PDP 做出的决定，日志的格式如下：主体，主体的属性值，客体，客体的属性值，决策用到的策略。以后会将这些日志进行审计。

ConfigServer 是缓存系统的控制节点。缓存系统用来存放属性和策略，由于缓存系统是分布式的，所以需要一个控制节点来决定数据存放的节点。

LoadBalance 负载均衡中心节点。由于计算量较大，所以搭建了多台 PDP 服务器，需要一个负载均衡节点。

3.5 本章小节

本章主要分析了 UCON 在云环境下的不足之处，对 UCON 的属性组件进行了扩展，引入了角色、租户、临时属性的概念，提出了新的模型：TR-UCON；给 UCON 引入会话这一概念，使得在模型之中，除了能表现出属性易变性的特点之外，也能表现出持续授权的特点。最后给出了基于改进 UCON 模型的云存储系统架构，分析了每个模块的主要功能。

第四章 系统实现的关键技术分析

本章主要基于上一章改进后的 UCON 模型的基础上，实现了一个面向企业用户的云存储系统。本章首先分析了系统在用户量较大的情况下可能出现的一些性能问题，然后给出了相应的解决方案。例如本文使用了消息队列来解决系统的扩展性问题，使用负载均衡来使多台机器对外提供统一的服务，使用缓存系统来解决主客体属性读取的性能问题。最后通过实验验证了改进后的授权系统的有效性。

4.1 策略决策模块

XACML 是 OASIS 提出了一个标准，但是 OASIS 并没有给出具体的实现。Balana 是一个开源的 XACML 的实现，基于 Sun's XACML 开发。Sun's XACML 用 Java 实现了所有 XACML1.1 标准必须实现的规则，而 Balana 可以支持 XACML3.0[36]。

Balana 中对于策略的寻找方式仅仅支持 ResourcePolicyFinder 方式，而本系统中由于使用了缓存系统，所以会有大量的 Policy 都需要存放在内存里，所以也需要一种从内存中查询策略的方式，因此需要对 Balana 做出一些扩展，添加一种查询策略的方式：InMemoryPolicyFinder。

在 InMemoryPolicyFinder 方式中，使用 HashMap 来存放所有的策略，其中 key 为策略的 URI，value 为策略本身。如果查询策略提供的参数为策略的 URI，可以在 O(1) 的时间内返回相应的策略。如果查询策略提供的参数为 EvaluationCtx，那就需要遍历整个 HashMap，逐个进行匹配。

4.2 消息通信

在架构中，很多模块之间都会涉及到消息的交互，如果模块与模块直接发送消息，在实现上，各个模块都耦合在一起，不利于扩展，有新需求的时候，也不方便改动，需要一个中间件来保证模块之间的通信。

4.2.1 同步与异步通信模块

模块与模块的通信大致分为两种：同步和异步。

对于异步的通信，实时性要求不高，消息可以在中间件当中存放一段时间也不影响结果，例如日志的存取，这种情况可以采用消息队列技术。ActiveMQ

是一个开源的中间件框架，在高并发的访问下仍然有很好的性能，而且消息不会丢失，未必处理的消息会被持久到硬盘中[37]。在本系统的 ActiveMQ 里，针对每个模块都有一个与之相对应的队列，每个模块都从与之对应的队列中读取消息，也可以向其他模块的队列发送消息，从而实现异步通信，图 4-1 是异步通信的结构模型。

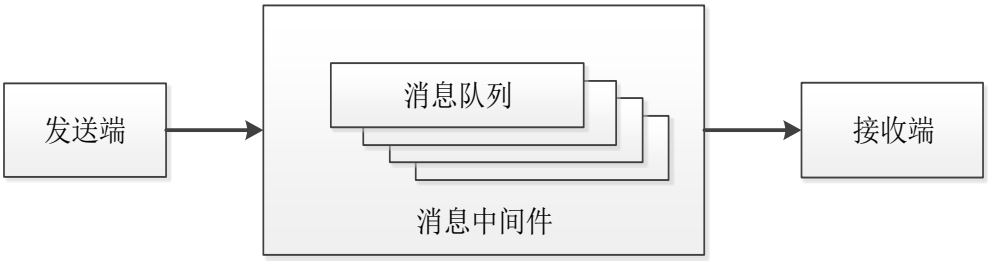

图 4-1 异步通信结构模型

对于同步的通信，对实时性要求很高，例如 PDP 模块与 PEP 模块的通信，PDP 要处理 PEP 的请求，这种情况需要快速的给用户返回决策结果，因此不能采用消息队列的形式进行通信。需要另一种通信方式：序列化与远程过程调用。PEP 与 PDP 的通信过程如下：PEP 在与 PDP 通信之前先要拿到 PDP 服务器的地址，与 PDP 所在的进程建立连接，PDP 将服务对象序列化，发送给 PEP，PEP 获取到服务对象，解序列化，通过调用服务对象的方法来使用服务。

在云环境下往往是多台机器构成的集群来提供服务，如果使用硬编码的方式将服务器的地址写入代码中，就会使得服务难以应对机器故障，也难以通过添加机器的方式来对服务进行横向扩展。所以本文引入了服务管理中心 ServerManager 来解决这个问题。

在系统中，所有的服务提供者启动时都向 ServerManager 注册服务信息（接口、版本、超时时间、序列化方式等），这样 ServerManager 上面就有了所有的服务信息。服务调用者启动的时候向 ServerManager 注册对哪些服务感兴趣，ServerManager 就把服务地址推送到服务调用者那里。调用者在调用时则根据服务信息的列表直接访问相应的服务提供者，而无需经过 ServerManager。

服务提供者每隔一段时间就会向 ServerManager 发送一个心跳信息来表示自己的存活状态。如果 ServerManager 发现某个服务提供者长时间未发送心跳信息，就会主动询问，如没有收到回应，说明该服务器可能出现故障，ServerManager 就会告知所有与该服务相关的服务调用者，图 4-2 是同步通信的服务架构。

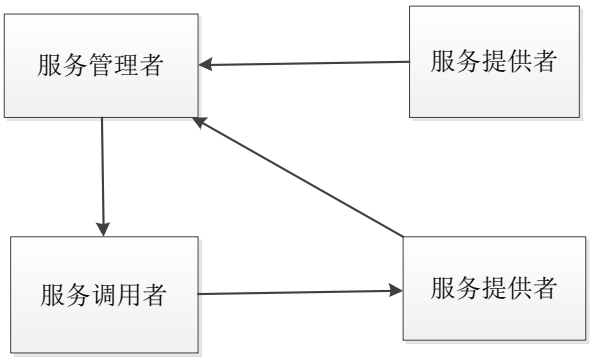

图 4-2 同步通信服务架构

4.2.2 通信模块的序列化机制

在同步通信中一个很重要的技术是序列化。

序列化（serialization）是指将结构化的对象转化为字节流，以便在网络上传输或者写入到硬盘进行永久存储；相对的反序列化（deserialization）是指将字节流转回到结构化对象的过程。

在分布式系统中进程将对象序列化为字节流，通过网络传输到另一进程，另一进程接收到字节流，通过反序列化转回到结构化对象，以达到进程间通信。

由于本系统是一个集群系统，对序列化机制的要求如下[38]：

(1) 紧凑，带宽是分布式系统中最为重要的资源之一，因此需要一个紧凑的序列化机制来降低带宽的使用率；

(2) 快速，在应用中，网络的 I/O 操作往往是性能的瓶颈，因此序列化和反序列化的速度必须很快，不能让传输速度成为应用的性能瓶颈。

Java 原生的序列化机制，其序列化结果包含了大量与类相关的信息，例如类的描述信息，类的版本 ID，对于拥有父类的类，父类的信息也会被保存下来；这些信息之后才是对象的数据。在这个过程中，序列化输出中保存了大量的附加信息，导致序列化结果膨胀，因此本系统需要一个新的序列化机制。

Hadoop 的序列化框架是专为集群系统开发的，主要目的在于减少时间和空间开销，它要比 java 原生的序列化高效的多。

Hadoop 的序列化机制的过程是先调用对象的 write()方法，把对象序列化到流中，反序列化的时候，调用对象的 readFileds()方法，把数据从流中读取出来。在 Java 原始的序列化机制中，对对象的反序列化的过程会不断地创建新的对象，这样会产生大量的对象。而在 Hadoop 的序列化机制中，对象是可以复用的，可以在同一个对象上得到多个反序列化的结果，而不是多个对象，这大大减少了 GC 中新生代对象的生成，减少了应用因为 GC 的执行而停顿的次数和时间，提高了应用的效率。

4.3 属性管理模块

每个主体以及客体都有一个 ID，其属性应该放在数据库中。但是每个主体拥有的属性数量不一样，客体的属性同样也是如此。因此，主体与客体的属性并不是一个结构化的数据，这种情况不适合使用关系型数据库来存放主客体的属性，MongoDB 能很好的满足这种需求，对于非结构化的数据，MongoDB 提供了非常好的存储方案。

4.3.1 使用 MongoDB 作为数据存储层

MongoDB 是一个基于分布式文件存储的数据库。它既不完全是关系数据库，也不完全是非关系数据库。是介于它们之间的一款数据库产品。它支持的数据结构比较松散，非常类似 json 类型的数据格式，因此可以存储比较复杂的数据类型。MongoDB 并不支持标准的 SQL，它有独特的查询语言，并且其功能非常强大，其表述形式类似于面向对象的查询语言，几乎可以实现标准 SQL 中单表查询的绝大部分功能，并且支持对数据的一列或者多列建立索引[39]。

在 MongoDB 中本系统用两个 Collection 分别用来存放主体的属性和客体的属性。

db.createCollection("subAttr")，创建一个名字为“subAttr”的 Collection，用于存放主体属性。

db.createCollection("objAttr")，创建一个名字为“objAttr”的 Collection，用于存放客体属性。

一个主体属性的实例：

{

SubID："300021"，

Name："john"，

Age："25"，

Role：" software development engineer"，

Tenement："Bat"

}

一个客体属性的实例：

{

ObjID："123845176"，

Tenement："Bat"，

Type："txt"，

SecuirtyLevel："high"，

Path："bat/fical"，

Filename："employee"

}

4.3.2 为数据库添加索引

由于云环境下用户量很大，文件系统存也存放了许多文件资源，因此存储在数据库中主客体的属性量也很大，为了提升检索效率，减少程序等待响应的时间，本系统对 ID 字段建立索引。

数据库的索引对于提高数据库检索的速度是立竿见影的，由于使用的特殊的数据结构来存储索引，使得在很短的时间内就可以查询到需要的数据[39]。如果没有索引的话，数据库的搜索引擎在检索数据时必须全盘扫描 Collection 中的记录，并且把每条记录都抽取出来与查询条件进行比对，最终选取那些符合查询条件的记录。 这种全集合扫描的查询效率是非常低的，特别在处理大量的数据时，查询可以要花费几十秒甚至几分钟，这对应用的性能是非常致命的。

索引是特殊的数据结构，索引存储在一个易于遍历读取的数据集合中，索引是对数据库表中一列或多列的值进行排序的一种结构。

db.subAttr.ensureIndex({SubID:1}) 对“subAttr”中的“SubID”字段建立一个升序索引。

db.objAttr.ensureIndex({ObjID:1}) 对“objAttr”中的“ObjID”字段建立一个升序索引。

针对索引的测试，使用 MongoDB2.6 版本，数据库中有 100 万条记录，每次查询 100 条，查询六次，记录查询耗时。无索引的情况下，查询耗时 60s 左右，而加入索引，耗时 3.5s 左右。结果如图 4-3 所示：

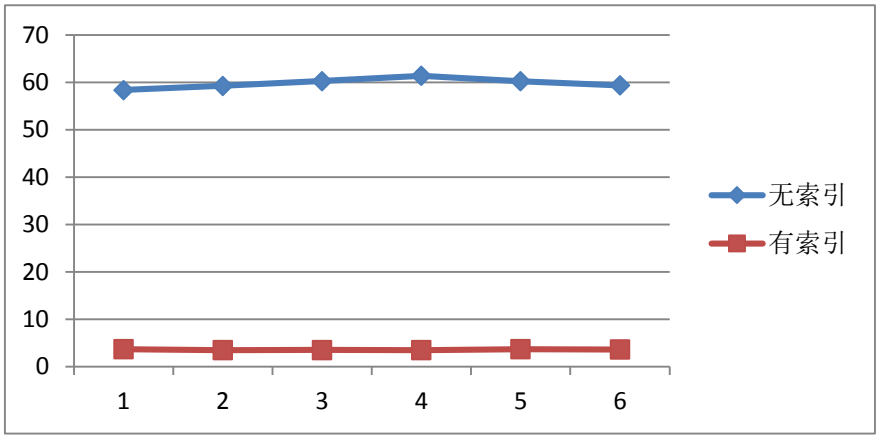

图 4-3 索引效果对比

4.4 负载均衡

由于是云存储平台，对策略的计算量应该很大，因此需要使用多台服务器

来对策略进行计算，也需要加入负载均衡模块，把不同的请求交给不同的服务节点。4.2.2 节当中描述了一个服务管理的节点 ServerManager，本系统在这个节点之前加入负载均衡的功能。

负载均衡的作用是把不同的请求分散到各个数据节点的技术，在分布式的应用中比较常见。负载均衡分为软件负载均衡和硬件负载均衡两种，目前，一些专业的负载均衡硬件虽然其效果很好，但是却很昂贵，所以近年来软件负载均衡大受亲睐。

在本系统中采取加权轮询算法，算法的流程如图 4-4 所示[40]：

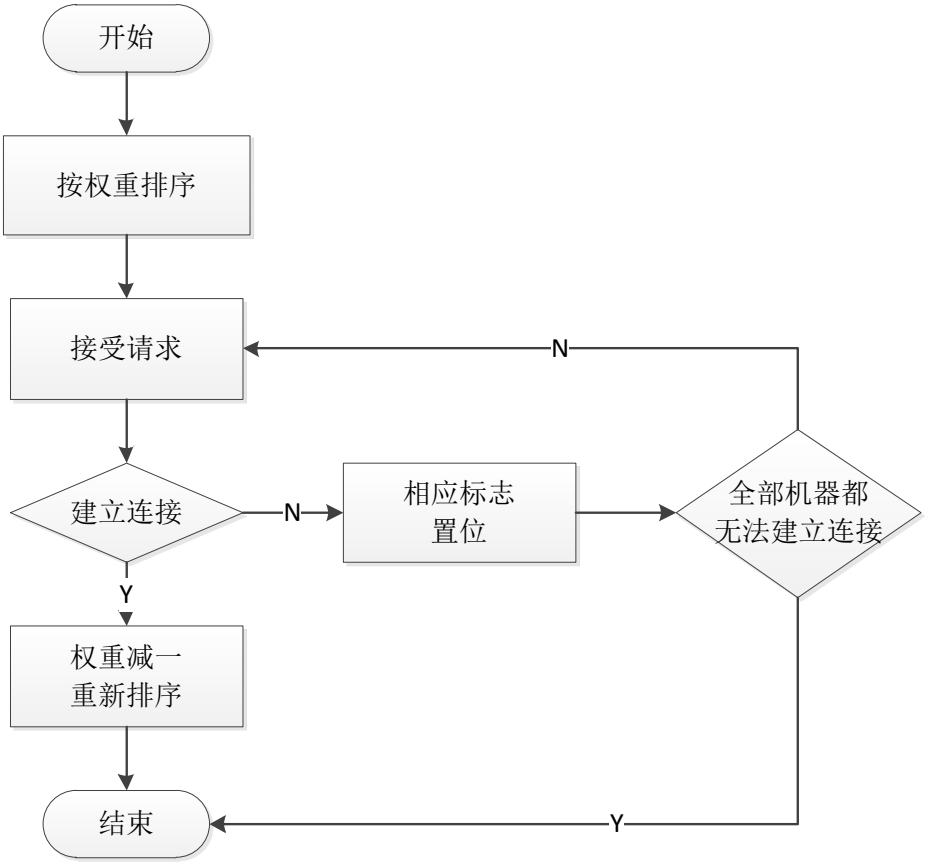

图 4-4 加权轮询算法流程图

加权轮询算法分为广度优先搜索和深度优先搜索，本系统采用的是深度优先搜索算法。首先将请求都分给权重最高的节点，该节点的权重减一，然后所有的节点按照权重的高低重新排序，下一个请求过来之后，重复上述过程，流程图如图 4-4 所示。

在上述流程中需要注意的一点是：有些节点会出现故障，当所有后端节点都出现故障时，本系统会不会再继续接受请求，以避免造成所有的节点都处在 timeout 的状态，造成死循环。

4.5 缓存系统的使用

在每一个主体对资源进行访问的时候，PDP 模块都要不断的计算，也需要获取到相关的属性值和策略，如果每次都从数据库或者文件系统中读取，时间开销会很大。为了解决这个问题本系统引入了缓存系统，缓存系统主要用于存放主客体的属性以及策略，这样在决策需要用到这些信息的时候，先在缓存系统中查找有无结果，如有，直接从缓存系统中读取。由于在会话当中的访问要不断的进行读取，因此所有会话当中的主客体属性以及策略都应该存放在缓存系统中。

4.5.1 缓存系统架构

由于缓存中需要存放的数据量很大，使用单机缓存不能很好的满足需要，因此本系统仿照 HDFS 的机制设计了一个分布式的缓存。

缓存系统由一个中心控制节点 ConfigServer 和一系列的数据节点 DataServer 组成。中心控制节点负责管理所有的数据节点，维护数据节点的状态信息。数据节点对外提供缓存服务，并每隔一段时间给中心节点发送一个心跳数据，汇报自身的情况。中心控制节点是单一的控制点，可以仿照 HDFS，使用一主一从的形式来保证可靠性。所有的数据节点地位是相同的。缓存系统的架构图如图 4-5 所示：

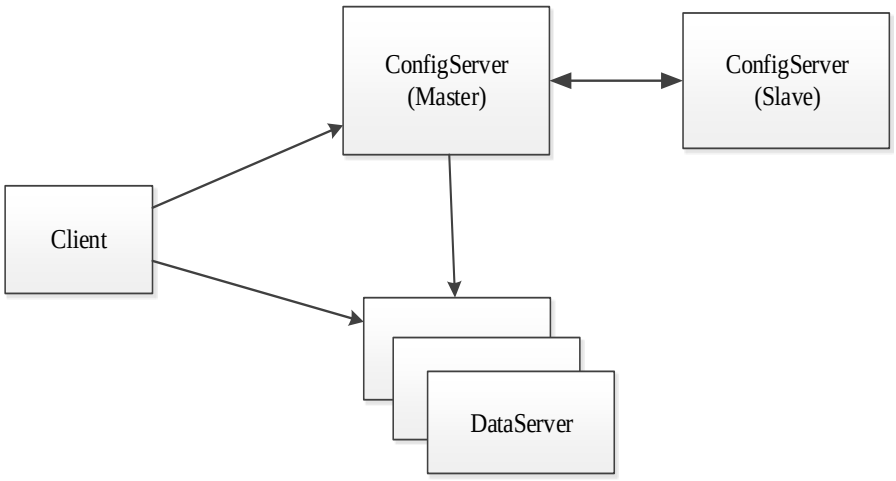

图 4-5 缓存系统架构图

分布式缓存不同于单节点缓存，对于一个数据对象 object，要选择一个节点来存放。传统的做法是先对这个数据对象使用哈希函数，得到一个哈希值，然后对节点数量进行取余，也就是 hash(object)%n，n 为节点的数量。这种方法有一个问题，一旦某一个节点出现故障，缓存系统的命中率就大大降低，因为出现故障后，节点的数量就为 n-1，所有的取余结果都与原来的不同，因此本

系统采用了一致性哈希算法[41]。

算法的过程如下：

首先定义一个具有 2^{32} 个数字的空间，数字的范围是 0~(2^{32}-1)，现在可以将这些数字头尾相连，形成一个闭合的环形。将 DataServer 节点均匀的放置到环形空间中，然后对于某个对象进行 hash 操作，哈希的结果一定在上述的环形空间中，然后以顺时针的方向计算，将对象 object 放到离自己最近的节点上。

采用这种算法，即使某个节点发生了故障，也不会影响其他节点的正常运转。

4.5.2 缓存置换算法

由于缓存系统的空间有限，不断的有新数据加入到缓存当中，旧的数据就需要被替换掉。由于会话中的主客体属性会被不断的用到，因此要尽量的保证这些数据在缓存系统中。

LRU(Least Recently Used) 最近最少使用算法，是内存管理中的一种页面置换算法。LRU 的基本思想是一旦需要替换的时候，就把最近未被使用的元素替换。基本的实现方法是：维持一个队列，队列中的元素被使用过之后就放到队尾，这样越靠近队尾的元素都是最近使用过的，越靠近队首的元素都是近期没被使用到的，因此需要发生替换的时候就把队首的元素删除掉。

由于会话中的主客体属性会被不断的读取，因此总是被最近使用到，不会被替换掉，因此使用 LRU 算法能很好的保证缓存的命中率。

本文对缓存系统的性能进行了测试，测试了直接读取数据库和从缓存中读取 100 个主体的属性所消耗的时间，加入缓存之后，提高了 97%的读取速度。图 4-6 展示了对比效果：

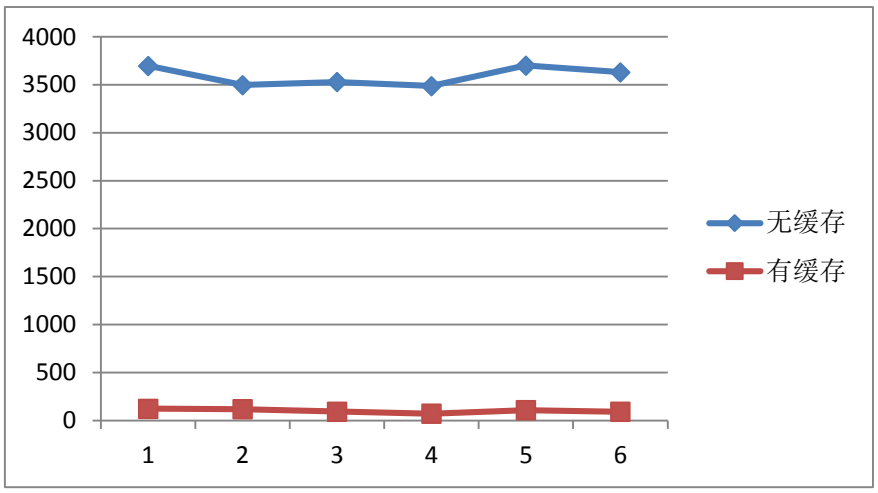

图 4-6 缓存效果对比

4.6 实验验证

实验目的：验证访问控制系统的有效性。

实验环境：由于时间仓促，本文只完成了系统的后台部分，因此在实验中，使用浏览器模拟前端界面，向 Tomcat 服务器发送请求，然后交给 PEP 模块，PEP 会对请求进行后续的处理，PEP 模块使用 Struts 搭建。

(1) 图 4-7 展示了实验界面，界面的左上方是主体以及主体的属性值，右上方为客体以及客体的属性值。中间是两种操作类型，最下面是授权的日志，系统所有的授权操作都会记录到日志中。当用户的角色属性具有某种权限时候，用户也拥有该权限，当用户的角色被更换时，其权限也随着发生变化。图 4-7 展示了正常情况下的主客体属性以及授权结果，图 4-8 展示了主体更新角色之后没有访问客体权限的情况，其授权结果是不通过。

图 4-7 正常情况下的访问结果

(2) 验证租户的隔离特性，图 4-9 中主体的租户 ID 是 2374135，而客体的租户 ID 是 2374136，他们属于不同的租户，主客体的 ID 不匹配，因此，这种情况下主体不具备访问客体的权限，授权日志给出了授权结果以及失败原因。

图 4-8 更换角色后的访问结果

图 4-9 不同租户下的访问操作

(3) 验证动态授权的过程。如图 4-10 所示，用户开始发出访问请求，授权通过，系统创建了一个 session，并把这个 session 加入到检测队列中，每隔一段时间重新检测，并记录授权结果，一直到某一次授权不通过或者是用户主动撤销访问。

图 4-10 会话检测的过程

4.7 本章小结

本章详细提出了云存储系统的模型架构，考虑到系统的效率问题，采用了集群提供服务的方式。在多台服务器提供服务的时候，往往需要一个中央节点来进行控制，例如分布式缓存系统中的 ConfigServer。在当前大用户量的环境下，分布式的应用是必不可少的。

第五章 总结与展望

5.1 本文工作总结

在云计算高度发展的同时，云计算中的安全问题也逐渐暴露出来。其中云存储作为云计算的一种主要服务，其安全问题更应加以重视，由于使用云存储可以大幅的降低企业的生产成本，因此越来越多的企业将其数据放在云端，其中可能包括一些敏感数据，但是传统的访问控制模型和技术并不能很好的满足云环境的需求，因而本文针对这个问题进行了相关的研究。

本文先是对传统访问控制模型和相关控制技术进行了调研。包括自主访问控制、强制访问控制和基于角色的访问控制，以上模型都是静态授权模型。下一代访问控制模型 UCON 引入了属性、条件和义务这三个组件，UCON 具有属性的可变性以及动态授权的特点，因此属于动态授权类型。但是 UCON 只是一个抽象的模型，在云环境下仍然有一些缺陷，例如：规则定义繁琐，不具备隔离性，在实际的实现时要加以改进。因此本文针对 UCON 的属性进行了扩展，加入了角色，租户和临时属性，使其克服了上述缺陷。除此之外，本文还对 UCON 引入了会话元素，使之很好的体现了持续授权的特性。

最后本文在基于 HDFS 上实现了一个云存储系统，并且在云存储系统内部实现了访问控制模块。在具体实现中加入了负载均衡，使得系统能同时对多个会话进行检测和重新授权；加入了缓存系统，用来保证系统读取主客体属性的性能；分别设计了同步和异步两种通信机制，用来保证系统的可扩展性。

5.2 进一步的工作

本文从理论和实现两方面的角度研究了 UCON 模型，在理论研究的过程中，发现 UCON 还存在一些其他的缺陷，例如在使用中更新的风险以及使用控制的并发性。风险问题是指某些属性只能由远端的客户端来提供，在传输给服务端的过程中可能会发生属性丢失或被人有意篡改的问题；并发问题是指多个访问可能要对同一个属性值进行修改，这样可能会引发冲突，并导致最终结果不正确；这两个问题在本文中都没有考虑到，需要进一步的研究。

另外本系统在实现的时候偏向于 $UCON_A$ 模型，对义务、条件这两个元素都没有对应的模块来支持，需要进一步的实现。

参考文献

[1] 刘松柏. 大数据及应用就是国家竞争力 [EB/OL]. http://www.chinacloud.cn/show.aspx?id=18787&cid=22.

[2] wiki. 云计算 [EB/OL]. http://zh.wikipedia.org/wiki/%E9%9B%B2%E7%AB%AF%E9%81%8B%E7%AE%97.

[3] ChinaCloud. 云安全的架构设计与实践之旅 [EB/OL]. http://www.chinacloud.cn/show.aspx?id=18369&cid=29.

[4] OpenStack. The auth system [EB/OL]. http://docs.openstack.org/developer/swift/overview_auth.html#access-control-using-keystoneauth.

[5] Google Cloud Storage. Access control [EB/OL]. https://cloud.google.com/storage/docs/accesscontrol.

[6] Amazon. AWS documentation [EB/OL]. http://docs.aws.amazon.com/zh_cn/AmazonS3/latest/dev/s3-access-control.html

[7] 于欣. 云计算中的访问控制技术研究[D]. 西安: 西安电子科技大学, 2013.

[8] ChinaCloud. 云计算的 6 个未来趋势 [EB/OL]. http://www.chinacloud.cn/show.aspx?id=18371&cid=22.

[9] 钱进进. 私有云安全存储技术的研究与实现[D]. 广州: 广东工业大学, 2013.

[10] 杨丽丽. 云存储网关的研究与实现[D]. 武汉: 华中科技大学, 2013.

[11] ChinaCloud. 云存储取得的突破性应用 [EB/OL]. http://www.chinacloud.cn/show.aspx?id=17339&cid=30.

[12] 任礼. 云计算的多租户存储分析[J]. 科学与财富, 2013，2013 (11): 323-323.

[13] 刘宇龙, 薛涛. 企业多租户云存储平台的设计与实现[J]. 西安: 西安工程大学学报, 2014, 28(2): 213-219.

[14] 李凤华, 苏锟, 史国振, 等. 访问控制模型研究进展及发展趋势[J]. 电子学报, 2012, 40(4): 805-813.

[15] 鸟哥. 鸟哥的 Linux 私房菜[M]. 北京: 人民邮电出版社, 2010.

[16] 魏娟. 云计算中基于角色的访问控制管理模型研究[D]. 长沙: 湖南大学, 2012.

[17] 程剑豪. 基于多元判决的动态访问控制模型的研究与设计[D]. 上海: 上海交通大学, 2009.

[18] 何康. 云计算环境下基于多目标规划的访问控制模型研究[D]. 长沙: 湖南大学, 2012.

[19] 李双. 一种扩展的基于角色的访问控制模型[J]. Computer Engineering and Applications, 2012, 48(19).

[20] Sandhu R S, Coyne E J, Feinstein H L, et al. Role-based access control models[J]. Computer, 1996, 29(2): 38-47.

[21] Sandhu R. Role activation hierarchies[A]//Proceedings of the third ACM workshop on Role-based access control[C]. ACM, 1998: 33-40.

[22] 罗鑫. 访问控制技术与模型研究[D]. 北京: 北京邮电大学, 2009.

[23] Arnab A, Hutchison A. Persistent access control: a formal model for drm[A]//Proceedings of the 2007 ACM workshop on Digital Rights Management[C]. ACM, 2007: 41-53.

[24] Martínez-García C, Navarro-Arribas G, Foley S N, et al. Flexible secure inter-domain interoperability through attribute conversion[J]. Information Sciences, 2011, 181(16): 3491-3507.

[25] Lee. Hadoop 介绍 [EB/OL]. http://blog.csdn.net/leechenglong/article/details/22790225.

[26] Apache. HDFS architecture guide [EB/OL]. http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[27] greatwqs. Hadoop HDFS 架构和设计 [EB/OL]. http://greatwqs.iteye.com/blog/1840321.

[28] OASIS. xacml-3.0-core-spec-os-en [EB/OL]. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf.

[29] 刘铁钢. 基于 XACML 的多租户访问控制的研究与应用[D]. 呼和浩特: 内蒙古大学, 2014.

[30] OASIS. A brief introduction to XACML [EB/OL]. https://www.oasis-open.org/committees/download.php/2713/Brief_Introduction_to_XACML.html.

[31] 姚冬梅. 基于 UCON 的云计算访问控制模型研究[D]. 南京: 南京大学, 2012.5.

[32] 李亚平, 周伟良. $UCON_{ABC}$ 模型中的委托授权方案研究[J]. 中国科学技术大学学报, 2012, 42(2): 154-160.

[33] 朱君礼. 基于 UCON 的云计算访问控制的研究与应用[D]. 北京: 北京邮电大学, 2013.

[34] 李萍. 基于 UCON 模型的 PMI 系统的研究与实现[D]. 上海: 上海交通大学, 2007.

[35] 谢辉, 张斌, 任志宇. 基于 UCON 模型的 PMI 体系结构[J]. 计算机工程与设计,2009, 2009 (7): 1590-1592.

[36] Asela. “Balana” The open source XACML 3.0 implementation [EB/OL]. http://xacmlinfo.org/2012/08/16/balana-the-open-source-xacml-3.

[37] Apache. ActiveMQ [EB/OL]. http://activemq.apache.org/.

[38] 蔡斌, 陈湘萍，著. Hadoop 技术内幕[M]. 北京: 机械工业出版社, 2013.

[39] W3SCHOOL. MongoDB 索引 [EB/OL]. http://www.w3cschool.cc/mongodb/mongodb-indexing.html.

[40] xiajun07061225. 负载均衡-加权轮询策略剖析 [EB/OL]. http://blog.csdn.net/xiajun07061225/article/details/9318871.

[41] cywosp. 理解一致性哈希算法 [EB/OL]. http://blog.csdn.net/cywosp/article/details/23397179.

致谢

两年半的研究生生活马上就要结束了，在读研期间，不论是学术上还是生活上，我都受益颇多。

首先要感谢我的导师秦素娟副教授在我的论文写作过程中给予的指导和帮助，也感谢网络安全中心其他几位老师的关怀，众位老师对我论文的不足之处提出了很多中肯的意见。秦老师治学严谨、为人平和、见识广博、看问题能切中要点，感谢张老师的严格要求，使我的论文质量有了保障。

感谢温老师、高老师和金老师让我们学习六项精进，也感谢秦老师日夜加班帮助我修改论文。感谢西门子中国研究院给予我的实习机会，在一年多的实习中，我的工程开发能力得到了很大提升，也为我将来的发展奠定了良好的基础，感谢我的主管栗静文对我的指导和鼓励。感谢我的父母，你们都是在我背后默默的支持着我的人，这种支持是我无以回报的，你们是我前行中最大的动力之一。

感谢网络技术研究院 2012 级 7 班的所有同学们，我们一起度过了美好的研究生生活，你们构成了我研究生生活中最美好的记忆，特别感谢 1508 的舍友们，你们对我包容很多。

感谢所有关心和帮助我的人！

最后感谢评审的老师，感谢你们在百忙之中抽出时间审阅本文并提取宝贵的建议。

作者攻读学位期间发表的学术论文目录

[1] 高磊. 基于风险的 UCON 访问控制模型研究 [EB/OL]. 北京: 中国科技论文在线[2015-01-04].