1. 论文完成情况

论文中包含的工作主要分为三个部分：适用于云环境的多模式访问控制(基于身份的访问控制，基于属性的访问控制和基于动态属性的访问控制三种访问控制的协同工作)，用户权限回收引起机密数据安全隐患的解决方案和系统的性能测试。

目前，已经在开源云存储 swift 上实现基于身份的访问控制，基于属性的访问控制的权限判断以及相应的性能开销测试，同时也设计并实现一套适用于解决用户权限回收引起安全隐患的方案。

2. 课题研究内容

本课题研究的内容是在现有作为数据备份的云存储系统上，结合基于身份访问控制与基于属性访问控制的优点，针对用户个人空间及组空间，构建一种适用于满足用户个性化需求的数据共享的多模式访问控制机制，以此实现安全的数据访问；此外，在以上基础上，为实现访问控制的扩展性和自适应性，为满足云环境中属性集复杂多变、用户频繁出入的特性以及属性值本身动态增减性特征，结合之前的访问策略，设计一套面向整个云储存的访问控制方案，实现云存储中数据的安全的多样化访问控制需求。另一方面，针对云环境中用户频繁变动引起的权限回收可能引起的机密数据安全隐患问题，研究一种高效的立即权限回收机制，设计一种基于密文掷乱处理算法，在云端进行代理密文掷乱算法处理，以此实现高效、安全、可靠的立即回收机制。

本课题分两步完成：首先实现云存储的多模式访问控制，然后在此基础上实现用户权限回收解决方案。

3. 多模式访问控制方案

随着用户上网终端的多样化，用户对云存储系统的多元化需求日益突显。云存储的功能将不仅局限于存储服务，用户对于面向特定用户数据共享的需求与日俱增。为满足用户社交化的多元需求，云环境中多元化数据共享必将成为云存储平台的主流服务。然而目前针对数据共享的访问控制策略主要利用实现预定好的身份信息或静态属性信息实现，因此无法根据云环境中用户的访问行为及其对云资源的贡献大小来动态调整其访问权限，因此无法体现云用户多元化需求的变化，也不利于开放环境下激励用户分享更多有价值的资源。

结合云用户的特点，以及不同数据空间对访问控制策略的需求不同，设计出一种灵活、简洁的访问策略统一描述语言，抽象用户行为特征，将用户行为属性和静态属性相结合，实现一种多模式访问控制模型，以解决用户多样化的数据共享需求。该模型的整体结构如图 1 所示。

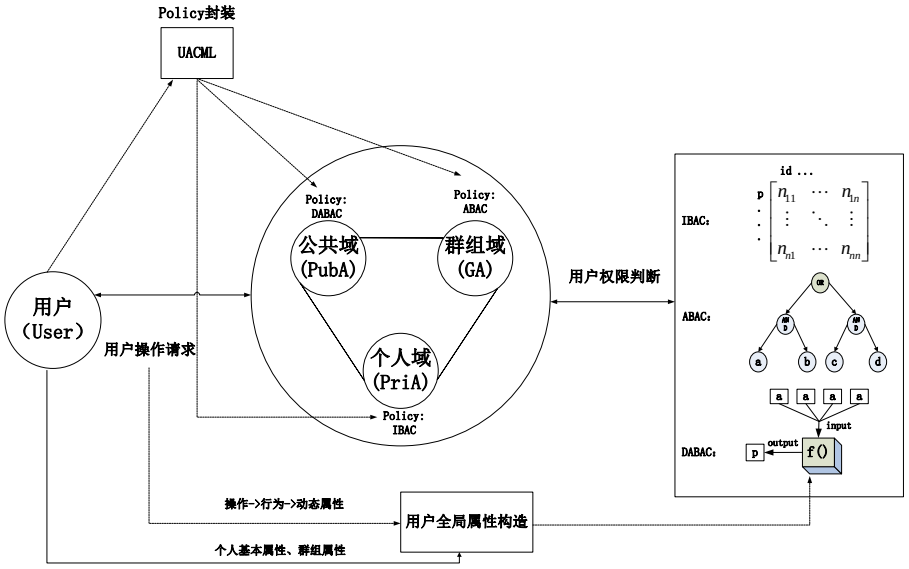


图 1 多模式访问控制模型

根据图 1 可以看出，多模式访问控制模型主要分为两个部分，一访问策略调度，用于实现不同访问策略的制定和选择；二是用户权限判断，针对不同的访问策略以此采用对应的权限判断规则，实现用户的权限判断。

3.1 多模式访问策略描述语言

多模式访问策略描述语言(DACML)是一套针对本系统自行设计的访问策略描述性规则，主要是用于实现本系统中三种不同的访问控制策略的统一规划和授权调度。DACML 主要借鉴 XACML（可扩展的访问控制标记语言），采用其中的标签来记录策略相关信息，通过标签之间的层级关系以及标签的逻辑组合来实现不同的访问策略整合，在实际调用时，根据标签值进行访问策略的选择，并节后 3.2 节的权限判断方法来实现整个访问权限的判断。

3.1.1 XACML

XACML 是一种用于决定请求/响应的通用访问控制策略语言和执行授权策略的框架，它在传统的分布式环境中被广泛用于访问控制策略的执行。在典型的访问控制框架中，有策略执行点 PEP(Policy Enforcement Point)和策略决定点 PDP(Policy Decision Point)。PEP 用于表达请求和执行访问控制决定。PDP 从 PEP

处接受请求，评估适用于该请求的策略，并将授权决定返回给 PEP。其形式如图 2 所示

```
<Policy PolicyID=Policy1
  PolicyTarget GroupName=University1
   RuleCombiningAlgId="permit-overrides">
  <RuleID=R1 Effect=Permit>
    <Target>
      <Subject Designation ={Educatee, Professor, Teacher}>
      <Resource FileType ={Score}>
      <Action AccessType ={Read, Write}>
    </Target>
    <Condition Time≤12:00>
  </Rule>
</Policy>
```

图 2 XACML 示例

3.1.2 DACML

DACML 是一种形如 XACML 的访问策略标记语言，是专用于本系统的访问策略标记描述语言。图 3 给出一个 DACML 的示例。

DACML 中的几个重要标签，其中 method 主要用于选择采用的具体访问策略，其值为 ABAC，ABAC，DABAC 中一个，white 标签表示白名单，在白名单的用户可以直接访问数据，即便其不满足访问策略规则；black 标签表示黑名单，黑名单中的用户无权访问数据；policy 标签为具体的访问策略逻辑表达式。

DACML 的具体使用规则如下：

1) 首先判断用户的连接请求，读取对应的 method 标签值，确定访问策略方案

2) 判断用户是否在 white 和 black 标签值中，若存在，则可以直接判断用户的访问权限；否则在非 IBAC 访问策略下继续执行步骤 3，若是 IBAC 则直接拒绝用户访问请求

3) 读取 policy 标签逻辑表达式，判断用户的属性集是否满足 policy 标签的逻辑表达式，若不满足表示用户无权访问；否则表示用户拥有合法的访问权限

注：在 DABAC 中，policy 中的 item 值通常为动态属性，进行逻辑表达式匹配时将不仅仅是简单的是或否进行决定，具体可以参看 3.3 节

```
<DACML
    id = policyid1
    method = IBAC/ABAC/DABAC  <!-- 三选一 -->
    <white list="user1,user2,user3" />
    <black list="user11,user12" />
    <rule>
        <item name=item1 attr="job" value="java,c++,c#" />
        <item name=item2 attr="age" value=">22" />
        <item name=item3 attr="sex" value="man" />
        <item name=item4 attr="work years" value=">3" />
    </rule>
    <policy>
        <cell name="p1" value="(item1 and item2) or (item3 or item4)" />
        <cell name="p2" value="item1 and item3" />
    </policy>
</DACML>
```

图 3 DACML 示例

3.2 用户权限判断

用户权限判断主要是指根据制定的访问策略判断用户是否拥有合法的访问权限。根据 3.1.2 节所述可以看出，访问权限判断的流程如下

1) 通过 method 标签获得具体的访问控制
2) 查看用户是否在白名单中
3) 查看用户是否在黑名单中
4) 判断用户是否满足 policy 标签的逻辑表达式

3.2.1 IBAC 权限判断

IBAC 主要采用传统的矩阵列表来记录可访问或拒绝访问的用户名单，以此实现细粒度的访问控制，其结构如图 4.

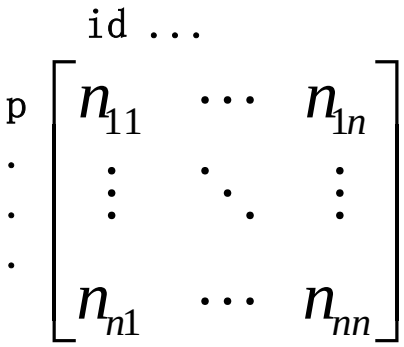


图 4 权限矩阵表

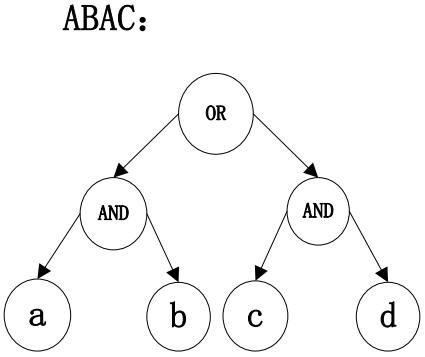


图 5 属性

3.2.2 ABAC 权限判断

ABAC 权限判断利用如图 5 的属性树的结构对用户权限进行判断，其中树的叶子节点为属性组成，非叶子节点则为 and,or,not,in 等逻辑词组成，若用户的属性集合满足属性树的一个逻辑成立的最小子树结构，则表示用户拥有合法的访问权限，否则表示用户无权访问。其具体算法如表 1

表 1 ABAC 权限判断算法

Algoritm1　generateTree(rule)	**Algoritm2　match(node, attres)**

step1 :　　/*初始化两个堆栈*/
InitStack(string_stack);　/*存放符号的堆栈*/
InitStack(node_stack);　/*存放树节点的堆栈*/
step2 :　　/*对输入的字符串进行扫描*/
while(attr = get(Policy)) // 获得属性
switch(attr) {
case'(':　将'('压入 string_stack;　break;
case')':　以 string_stack 栈顶元素为值创建一个二叉树结点;
弹出 string_stack 栈顶元素;
二叉树结点的左右指针指向 node_stack 栈顶的两个结点;
弹出 string_stack 栈顶的两个节点;
break;
case'and':　将'and'压入 string_stack;　break;
case'or':　将'or'压入 string_stack;　break;
default:　以该串为值创建一个二叉树结点，并压入 node_stack;break;
}
Step3:
以 string_stack 栈顶元素为值创建一个二叉树结点 root
其左右孩子分别指向 node_stack 栈顶的两个元素
return root;
	if(root -> value == "not"){
	return match(root->lchild, attrs) ? 0 : 1;
	}**else if**(root -> value == "and"){
	return (match(root->lchild, attrs) +
	match(root->rchild, attrs))
	== 2 ? 1 : 0;
	}**elseif**(root -> value == "or"){
	if(match(root->lchild, attrs) ||
	match(root->rchild, attrs))
	return 1;
	else
	return 0;
	}**else** {
	if (node -> value **in** attres) // 叶子属性在用户属性集内，返回 1
	return 1;
	else
	return 0;
	}

3.2.3 DABAC 权限判断

DABAC 利用已知访问权限与访问策略和用户属性全集对应关系，组成训练集，以此为基础构建如图 6 所示基于动态属性访问权限判断模型,从而实现用户权限判断.

训练集 D = {Xi ,Yi}m,其中 Xi 为用户属性与访问策略的二元组，Yi 为用户权限，利用机器学习构建权限判断模型 y=f(x)，因此用户操作权限为：P = f(x,y)

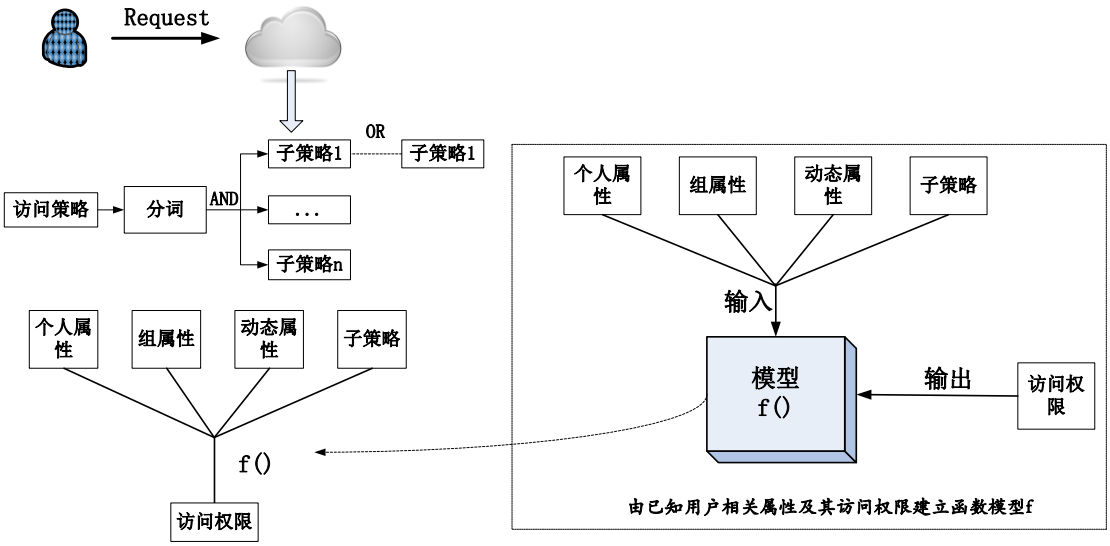


图6 DABAC权限判断模型

4．用户权限回收

云存储系统在为用户提供数据存储和业务服务的同时，其云端数据的安全性一直是用户和企业管理者尤为关注的重要问题。为保障用户数据的私密性，当前的云存储安全框架一般是对数据进行加密存储，云服务提供商使用相应的访问控制策略来判断用户对该数据是否拥有合法的的访问权限，密钥的管理和分发由可信第三方负责。虽然这在一定程度上加强了云端数据的安全性，但密文存储的形式也会引发新的问题。当需要变更用户对某数据的访问权限时，为保证回收权限的用户持有该数据的密钥无效，一般的方案是对该数据进行重加密处理，然后进行密钥的更新与重新分发。由于云环境中用户规模庞大且用户权限可能频繁变更，数据的重加密和密钥的重分发将会带来极大的性能开销。如果将重加密放在用户上传数据时进行，由于用户执行上传数据操作时间是不确定的，导致数据的重加密时机也是不确定的，而这期间已被回收权限的用户依然可以访问该数据，可能出现回收权限用户恶意向系统写脏数据，此外此方案无法解决只读数据的重加密问题。

由于用户权限回收引起的重加密问题，我们拟设计一种采用基于密文的置乱算法，采用立即回收方案，对密文进行代理置乱算法处理，并利用令牌机制保存置乱规则，合法用户利用解密密钥及令牌以获得原始数据。

置乱算法原理如下：

1) 云端首先将上传的密文数据分为$(n\times n-m)$块，利用随机算法生成m块大小与密文数据块大小相同的二进制置乱块；
2) 将生成的n块置乱块随机插入密文，得到新的密文并保存，同时维护一张$n\times n$矩阵表 A，其中置乱块对应内容为 0，密文块对应内容为 1，如：

$$\begin{bmatrix} n_{11} & \cdots & n_{1n} \\ \vdots & \ddots & \vdots \\ n_{n1} & \cdots & n_{nn} \end{bmatrix} \text{其中} n_{ij} \text{取 1 或者 0}$$

3) 令牌保存一张 n 阶方阵 B，其满足公式:，其中 I 为单位阵，计算并将得到的方阵作为密文元数据保存；
4) 用户获取原始数据，首先得利用授权中心赋予的令牌与获得密文元数据进行矩阵乘积运算，得到矩阵，然后利用得到的矩阵内容去除插入密文的置乱块；最后利用解密密钥获得原文；

   原理如下：

$$A\times B=C \qquad B\times B=nI$$

$$C\times B=A\times B\times B=A\times(B\times B)=A\times(nI)=nA$$

5) 用户权限回收时，由授权中心发给云端原始令牌及一个新的令牌，云端首先利用原始令牌剔除密文中的置乱块，然后利用新的令牌对密文数据进行如步骤 1、2 的代理置乱处理，并更新元数据中的矩阵表；合法权限用户则通过原始密钥及更新的令牌进行数据访问。

5.测试

5.1 测试环境

系统： ubuntu 12.10 64bi

内存： 4GB

硬盘: 500GB

处理器： Pentium® Dual-Core CPU E5400 @ 2.7oGHz x2

5.2 测试内容

测试内容包括功能测试和性能测试两个方面，功能测试主要是指客户端的正常数据访问请求，性能测试则主要是指添加访问策略后的性能开销。

5.3 功能测试

功能演示图如下：

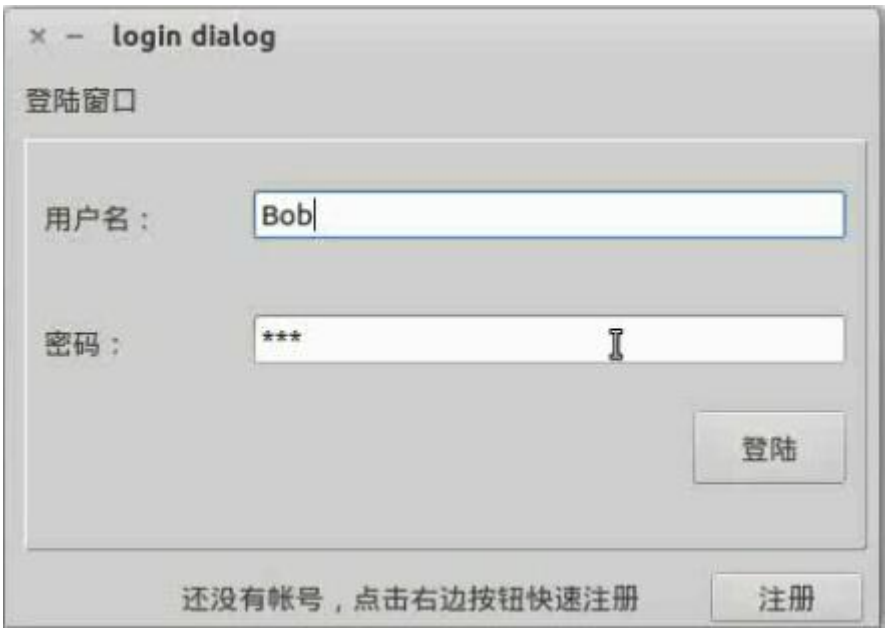


图 7 系统界面

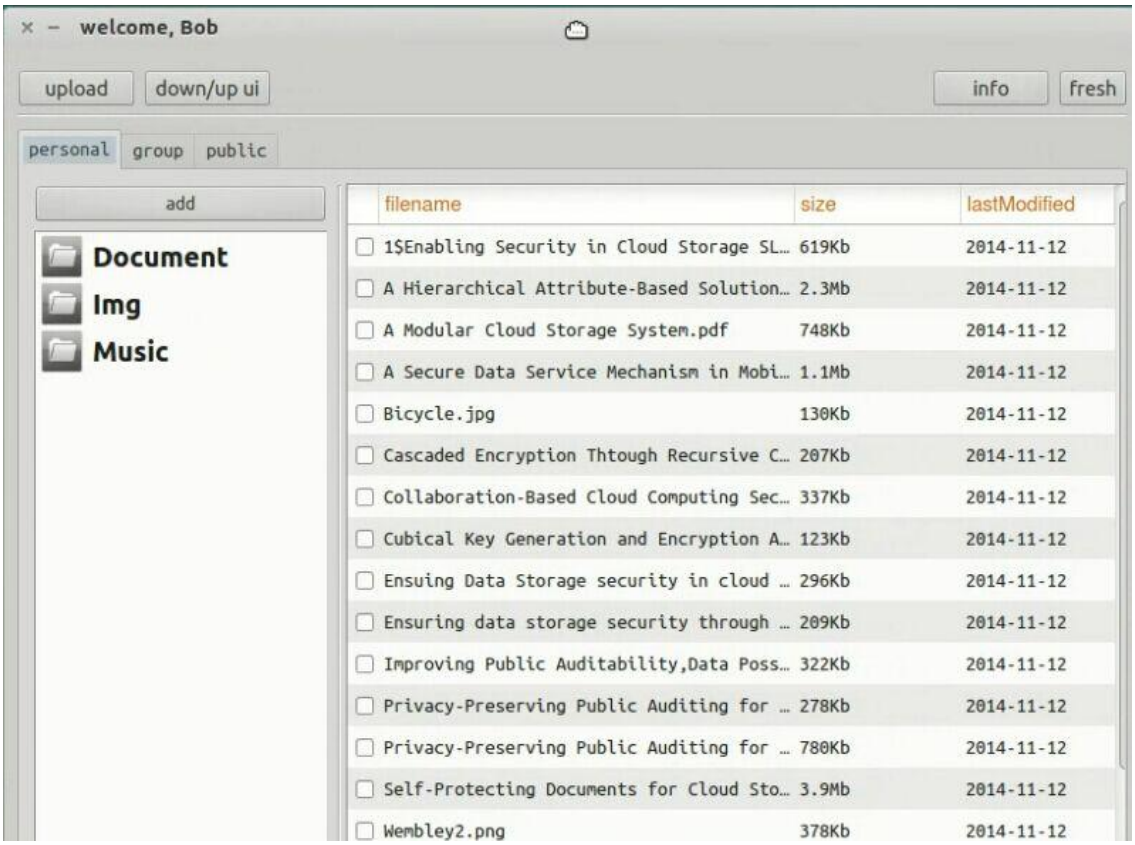


图 8 组空间制定访问策略

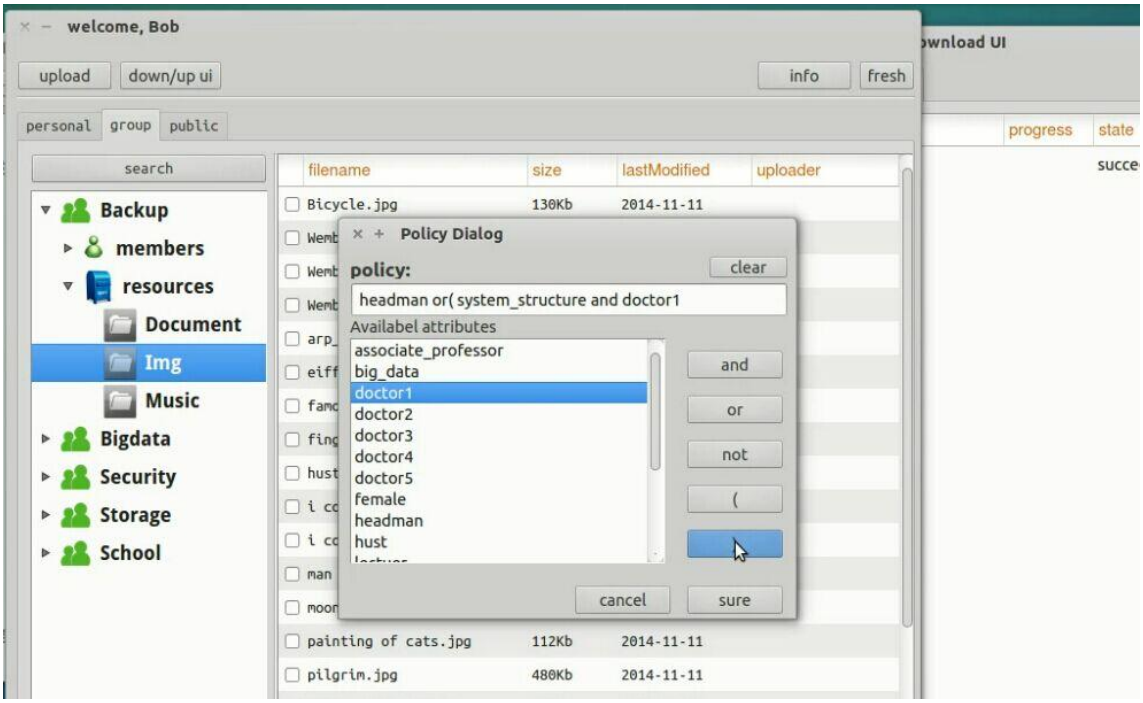


图 9 上传下载

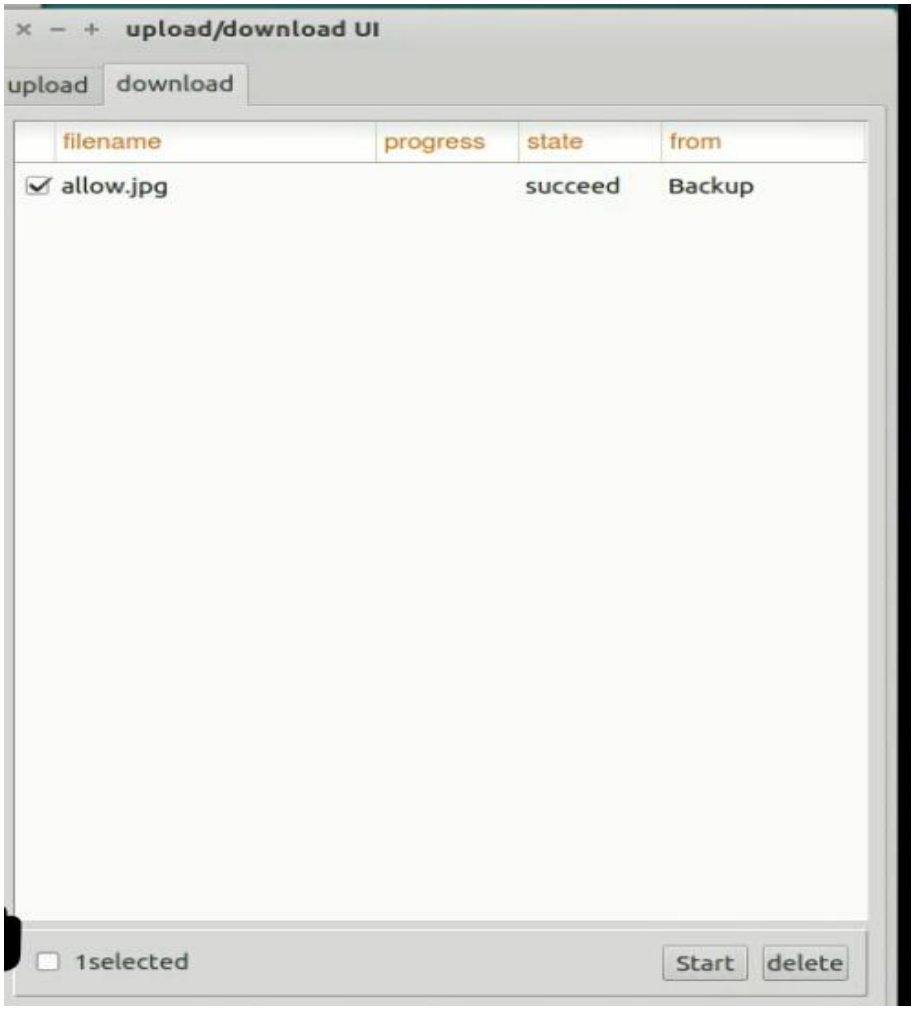
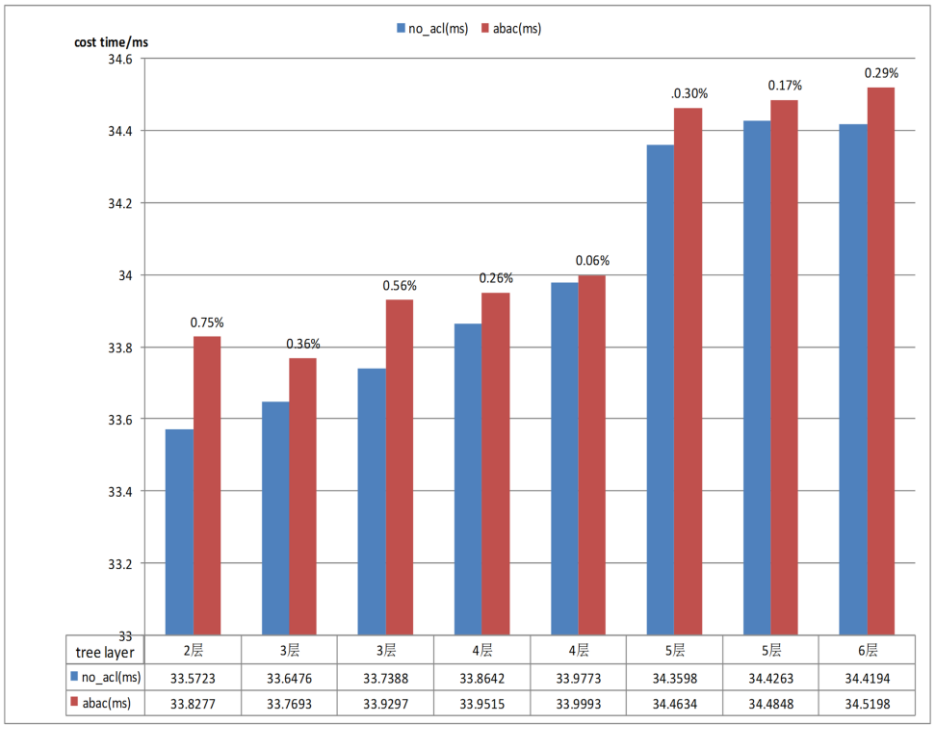


图 10 下载界面

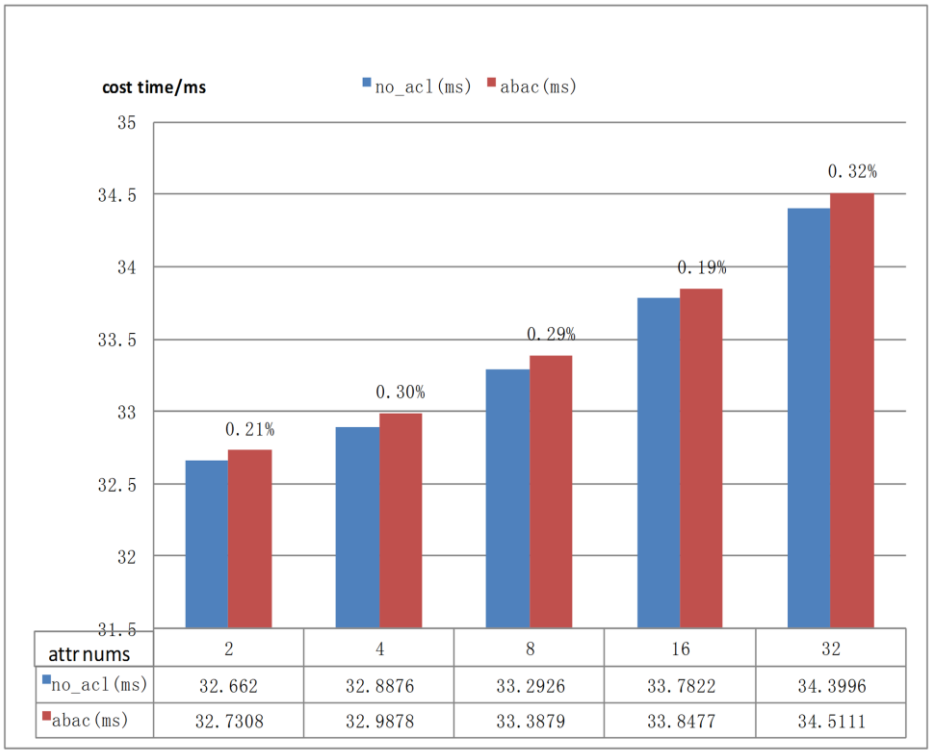
5.4 性能测试

主要对 ABAC 的性能开销进行了测试，将 ABAC 的访问控制开销与无访问策略进行了对比，可以看出增加访问策略，其所占的性能开销依然不超过 1%.



tree layer	2层	3层	3层	4层	4层	5层	5层	6层
no_acl(ms)	33.5723	33.6476	33.7388	33.8642	33.9773	34.3598	34.4263	34.4194
abac(ms)	33.8277	33.7693	33.9297	33.9515	33.9993	34.4634	34.4848	34.5198

ABAC不同访问策略复杂度的性能开销



attr nums	2	4	8	16	32
no_acl(ms)	32.662	32.8876	33.2926	33.7822	34.3996
abac(ms)	32.7308	32.9878	33.3879	33.8477	34.5111

ABAC中不同用户属性集合的性能开销

研　究　生　签　字 _____

指　导　教　师　签　字 _____

院(系、所)领导签字 _____

年　　月　　日