第 35 卷 第 2 期 计 算 机 工 程 2009 年 1 月
Vol.35 No.2 Computer Engineering January 2009
·软件技术与数据库· 文章编号：1000—3428(2009)02—0070—04 文献标识码：A 中图分类号：TP393

服务网格授权决策的$UCON_A$模型

桂劲松[1]，陈志刚[1]，胡玉平[2]
(1. 中南大学信息科学与工程学院，长沙 410083；2. 广东商学院信息学院，广州 510320)

摘 要：基于授权谓词决策的使用控制模型表达能力较弱。该文给出一种委托凭证模型细粒度表达决策结果，用委托凭证处理过程的状态组合替换原来的简单访问状态。决策组件根据请求时系统状态输出合理的委托凭证，根据系统状态的变化对委托凭证处理状态的转换进行决策。该方法有效避免了相同访问请求重复产生委托凭证，使委托凭证可以真实反映授权的实际需求。
关键词：服务网格；授权决策；委托凭证

$UCON_A$ Model for Service Grid Authorization Decision-making

GUI Jin-song[1], CHEN Zhi-gang[1], HU Yu-ping[2]
(1. School of Information Science and Engineering, Central South University, Changsha 410083;
2. School of Information, Guangdong University of Business Studies, Guangzhou 510320)

【Abstract】 To keep free from weak capability of express of the usage control model based on authorization predication decision($UCON_A$), a delegation certification model is proposed to express decision result in a fine-grained manner, and delegation certification processing statuses are defined to replace the simple access status. Decision component can make the reasonable delegation certification based on the system status when a request arrives, and also make decision to change the delegation certification processing status when the system status is changed. This method effectively avoids that the same access requests generate the delegation certification repeatedly, and the delegation certification really reflects actual demands of authorization.
【Key words】 service grid; authorization decision-making; delegation certification

使用控制(Usage CONtrol, UCON)概念和相应的概念模型$UCON_{ABC}$分别由文献[1-2]提出。文献[3]对$UCON_{ABC}$进行了形式化描述。在上述模型中，当一个访问请求被递交到决策组件时，决策组件仅输出简单的决策结果，即“允许”或“拒绝”。该方法在传统授权机制下可行，但其授权功能过分集中于决策组件，不利于授权负荷较重的网格环境中多个独立授权过程的并发执行。本文针对服务网格授权的新特点，研究并改进基于授权谓词决策的UCON，使其能满足服务网格授权决策的要求。

1 服务网格授权的新特点

按需、动态、即时构建服务虚拟组织(Virtual Organization, VO)，从而进行问题求解是网格发展的必然趋势。VO 必须对聚集的资源进行统一授权管理，但从资源角度来看，每个资源只关心其自身的访问控制策略和运行于其上的应用，因此，需要在 VO 中设置一个授权执行组件来充当全局控制器。该设置可以合理分担决策组件的繁重工作，实现多个独立授权过程决策与执行的并行，并能极大简化资源本地域执行组件的工作。此时，决策组件输出的授权决策结果不是简单的“允许”或“拒绝”，而是一个表示细粒度权限的授权证书，即委托凭证。

定义 1 集合$U=\{u_1,u_2,\cdots,u_m\}$表示所有用户，集合$R=\{r_1,r_2,\cdots,r_n\}$表示所有角色，集合$UA\subseteq U\times R$是从U到R的多对多映射。常规用户是由系统管理员指派角色的用户，委托用户是通过委托指派角色的用户。UA划分为UAO(表示用户与被授予角色的关系)和UAD(表示用户与被委托角色的关系)。

定义 2 角色树是委托授权的基本单位，表示为$r_{tree}=(r_0(r_1(r_{11}(r_{111}(\cdots),\cdots,r_{11n}(\cdots)),\cdots,r_{1m}(\cdots)),\cdots,r_k(r_{k1}(\cdots))))$。

定义 3 在委托票据$dt=(uad,pt,n,ae,dep)$[4]中，用r_{tree}替代$uad=(u,r)\in UAD$中的r，并将r看成r_{tree}的未给出树形结构的缩写式。

定义 4 委托传播树表示为$spr_{tree}=(dt_0(dt_1(dt_{11}(dt_{111}(\cdots),\cdots,dt_{11n}(\cdots)),\cdots,dt_{1m}(\cdots)),\cdots,dt_k(dt_{k1}(\cdots))))$。树的根节点拥有从常规用户得到的委托票据$dt_0$。$dt_1$, dt_2,$\cdots$, dt_k是根节点可签发的委托票据，根节点可以将部分或全部dt_0角色树委托给dt_1, dt_2,$\cdots$, dt_k的持有者。dt_{11}, dt_{12},$\cdots$, dt_{1m}是dt_1持有者可签发的委托票据，dt_1的持有者能将部分或全部dt_1角色树委托给dt_{11}, dt_{12},$\cdots$, dt_{1m}，其余依此类推。

定义 5 委托凭证由$dc=(n_d,n_b,spr_{tree})\in DC$组成。其中，$n_d$表示委托权限传播的许可步数，即委托深度；$n_b$表示每步中权限被委托的许可用户数，即委托宽度。

2 适用于服务网格授权决策的$UCON_A$

2.1 现有$UCON_A$的改进

文献[3]定义了如下概念：(1)系统状态定义为对一组变量的一次赋值，其中的变量包括主体属性、客体属性、系统属性 3 类。(2)实体(主体或客体)表示为一个有限属性集。(3)函数定义为一个或多个属性与常量构成的表达式。函数在形式上是从一组属性值到一个新值的映射。(4)单一使用过程用(s,

基金项目：国家自然科学基金资助项目(60573127)
作者简介：桂劲松(1968－)，男，讲师、博士研究生，主研方向：网络与信息安全，网格安全；陈志刚，教授、博士、博士生导师；胡玉平，教授、博士后
收稿日期：2008-07-25 **E-mail：**jsgui06@163.com

o,*r*)表示，其中，*s* 表示主体；*o* 表示客体；*r* 表示权限。(5)(*s*,*o*,*r*)的状态被定义为特殊系统变量，并定义函数 state(*s*,*o*,*r*)为从{(*s*,*o*,*r*)}到{*initial, requesting, denied, accessing, revoked, end*}的映射，其中，6 种状态的语义参见文献[3]。(6)谓词是由变量、函数、常量构成的逻辑表达式。其语义是从系统状态到布尔值的映射。若状态中指派的属性值属于谓词，则状态属于谓词。(7)使用控制行动包括更新属性值的行动 *preupdate*, *onupdate*, *postupdate* 和改变单一使用过程状态 *state*(*s*,*o*,*r*)的行动 *tryaccess*(*s*, *o*, *r*), *permitaccess*(*s*, *o*, *r*), *denyaccess*(*s*, *o*, *r*), *revokeaccess*(*s*,*o*,*r*), *endaccess*(*s*,*o*,*r*)，其语义参见文献[3]。

为了适应服务网格授权的新特点，本文将状态值 *accessing*(访问状态)细化为图 1 虚线框中的状态组合，并将 *permitaccess*(*s*,*o*,*r*)(允许访问)的语义改为将状态值 *requesting*(请求状态)改变为“凭证请求”状态，即将主体 *s* 对客体 *o* 的权限 *r* 的请求转换为主体 *s* 对委托凭证 *dc* 的请求。到达状态值 *revoked*(撤销状态)的前提是“凭证撤销”而不是“访问进行中”。

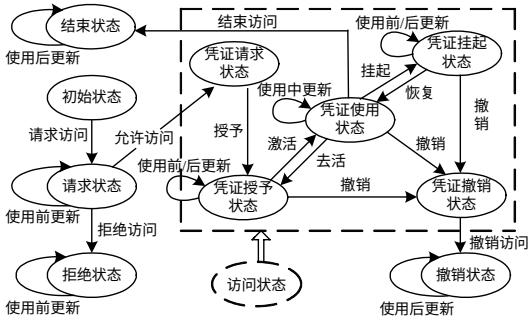

图 1　单一使用过程的状态变迁

定义 6 凭证处理过程用(*s*,*dc*)表示，其中，*s* 为主体；*dc* 为委托凭证。

定义 7 凭证处理状态集{request_dc(凭证请求)，grant_dc(凭证授予)，using_dc(凭证使用)，hold_dc(凭证挂起)，revoke_dc(凭证撤销)}用于表示(*s*,*dc*)的状态。其中，request_dc 的语义是(*s*,*dc*)已经产生并正在等待系统授予 *dc*；grant_dc 的语义是 *dc* 已被授予并随时可以被激活使用；using_dc 的语义是 *dc* 正在执行组件的使用；hold_dc 的语义是 *dc* 正在更新，暂时不能被执行组件使用；revoke_dc 的语义是 *dc* 已被系统撤销，不能再被执行组件使用。

定义 8 (*s*,*dc*)的状态表示为特殊系统变量，并定义函数 *state*(*s*,*dc*)为从{(*s*,*dc*)}到凭证处理状态集的映射。

定义 9(改变(*s*,*dc*)状态的使用控制行动) *grant*(*s*,*dc*)(授予)由系统执行，若其值为 true 则表示主体 *s* 授予委托凭证 *dc*。*activate*(*s*,*dc*)(激活)由系统执行，若其值为 true 则激活主体 *s* 的委托凭证 *dc*。*inactivate*(*s*,*dc*)(去活)由系统执行，若其值为 true 则去活主体 *s* 的委托凭证 *dc*。*hold*(*s*,*dc*)(挂起)由系统执行，若其值为 true 则挂起主体 *s* 的委托凭证 *dc*。*restore*(*s*,*dc*)(恢复)由系统执行，若其值为 true 则恢复主体 *s* 的委托凭证 *dc*。*revoke*(*s*,*dc*)(撤销)由系统执行，若其值为 true 则撤销主体 *s* 的委托凭证 *dc*。

定义 10 一个$\mathrm{UCON_A}$逻辑模型的抽象形式定义为三元组 $M_A=(S,P_A,A_A)$，其中，*S*是系统状态序列集；P_A是由主体和客体属性构成的授权谓词的有限集，A_A是使用控制行动的有限集。

定义 11 在$\mathrm{UCON_A}$中，逻辑公式 ϕ 由授权谓词和使用控制行动通过逻辑连接符和时态操作符构成。BNF 语法为 $\phi ::= a \mid p \mid (\neg\phi) \mid (\phi\wedge\phi) \mid (\phi\rightarrow\phi) \mid \boxdot\phi \mid \Diamond\phi \mid \bigcirc\phi \mid \phi U\phi \mid \boxminus\phi \mid \blacktriangle\phi \mid \Theta\phi \mid \phi S\phi$，其中，*a* 是使用控制行动；*p*是授权谓词；$\neg,\wedge,\rightarrow$ 是逻辑连接符，其语义分别是“非”、“与”、“蕴涵”；$\boxdot,\Diamond,\bigcirc,U,\boxminus,\blacktriangle,\Theta,S$ 是时态操作符，其语义分别是“Always”，“Eventually”，“Next”，“Until”，“Has-always-been”，“Once”，“Previous”，“Since”。

时态操作符语义的详细解释和使用示例参见文献[3]。为了适应服务网格授权的新特点，本文对文献[3]形式化的 8 种基于授权谓词的核心模型进行改进并分别命名为$\mathrm{sg_preA_0}$, $\mathrm{sg\_preA_1}$, $\mathrm{sg\_preA_2}$, $\mathrm{sg\_preA_3}$, $\mathrm{sg\_onA_0}$, $\mathrm{sg\_onA_1}$, $\mathrm{sg\_onA_2}$, $\mathrm{sg\_onA_3}$。统称上述模型为$\mathrm{SG_UCON_A}$。

2.2　$\mathrm{SG_UCON_A}$模型

2.2.1　$\mathrm{sg_preA_0}$模型

$\mathrm{sg_preA_0}$模型在使用前进行使用控制决策，并在使用前、中、后都无属性更新。使用控制策略表示为

(1) $permitaccess(s,o,r)\rightarrow_{\blacktriangle}(tryaccess(s,o,r)\wedge(pa_1\wedge pa_2\wedge\cdots\wedge pa_i))$

(2) $grant(s,dc)\rightarrow_{\blacktriangle}(permitaccess(s,o,r)\wedge(pa_1\wedge pa_2\wedge\cdots\wedge pa_i))$

(3) $activative(s,dc)\rightarrow_{\blacktriangle}(grant(s,dc)\wedge(pa_1\wedge pa_2\wedge\cdots\wedge pa_i))$

在上述策略中，pa_1, pa_2,…,pa_i是由主客体属性构成的授权谓词。行动*permitaccess*将(*s*,*o*,*r*)的状态值*requesting*改变为(*s*,*dc*)的状态值*request_dc*；行动*grant*将(*s*,*dc*)的状态值*request_dc*改变为*grant_dc*；行动*activate*将(*s*,*dc*)的状态值*grant_dc*改变为*using_dc*。此组策略表明，行动*permitaccess*为true意味着在当前系统状态前，*tryaccess*和授权谓词必须为true；行动*grant*为true意味着在当前系统状态前，*permitaccess*和授权谓词必须为true；行动*activate*为true意味着在当前系统状态前，*grant*和授权谓词必须为true。

2.2.2　$\mathrm{sg_preA_1}$模型

$\mathrm{sg_preA_1}$模型在使用前进行使用控制决策，且一个或多个主体或客体属性被更新。它使用策略(2)和如下 2 个策略：

(4) $permitaccess(s,o,r)\rightarrow_{\blacktriangle}(tryaccess(s,o,r)\wedge(pa_1\wedge pa_2\wedge\cdots\wedge pa_i))$
$\wedge_{\blacktriangle} preupdate(attribute)$

(5) $activate(s,dc)\rightarrow_{\blacktriangle}(grant(s,dc)\wedge(pa_1\wedge pa_2\wedge\cdots\wedge pa_i))$
$\wedge_{\blacktriangle} preupdate(attribute)$

策略中的“Once”操作符 $\blacktriangle$ 不涉及单一使用过程中 *tryaccess* 之前的状态。在策略(4)中，行动 *permitaccess* 在 *tryaccess* 之后执行，且在 *preupdate* 之前要求授权谓词为 true。行动 *permitaccess* 为 true 意味着在当前系统状态前，*tryaccess*、授权谓词、*preupdate* 必须为 true。在策略(5)中，行动 *activate* 在 *grant* 之后执行并且在 *preupdate* 之前要求授权谓词为 true。行动 *activate* 为 true 意味着在当前系统状态前，*grant*、授权谓词、*preupdate* 必须为 true。

2.2.3　$\mathrm{sg_preA_2}$模型

$\mathrm{sg_preA_2}$模型在使用前进行使用控制决策，且在使用中，一个或多个主体/客体属性被更新。更新不能改变当前正在进行中的使用过程的决策，但可以影响来自某个主体或对某个客体的其他进行中或随后的访问。它使用策略(1)~策略(3)和如下 2 个策略：

(6) $activate(s,dc)\rightarrow\Diamond(onupdate(attribute)\wedge\Diamond(endaccess(s,o,r)$
$\vee revoke(s,dc)))$

(7) $revokeaccess(s,o,r)\rightarrow_{\blacktriangle} revoke(s,dc)$

在策略(6)中，若行动 *activate* 为 true，就会造成如下情况：行动 *onupdate* 为 true 并最终导致行动 *endaccess* 为 true

或行动 *revoke* 为 true。在策略(7)中，行动 *revokeaccess* 为 true 意味着在当前系统状态前 *revoke* 必须为 true。

2.2.4 sg_preA$_3$模型

sg_preA$_3$模型在使用前进行使用控制决策，且在使用后，一个或多个主体/客体属性被更新。它使用策略(1)~策略(3)和策略(8)：

(8) $endaccess(s,o,r) \rightarrow \Diamond postupdate(attribute)$

主体结束访问后，由系统执行行动 *postupdate*。若行动 *endaccess* 为 true，则最终导致 *postupdate* 为 true。

2.2.5 sg_onA$_0$模型

sg_onA$_0$模型在使用中进行使用控制的检查和决策，且自始至终无属性更新。它使用策略(2)和策略(9)：

(9) $\square(\neg(pa_1 \wedge pa_2 \wedge \cdots \wedge pa_i) \wedge (state(s,dc) = usin\,g\_dc) \rightarrow revoke(s,dc))$

或 $activate(s,dc) \rightarrow (pa_1 \wedge pa_2 \wedge \cdots \wedge pa_i) U (revoke(s,dc) \vee endaccess(s,o,r))$

2.2.6 sg_onA$_1$模型

sg_onA$_1$模型在使用中进行使用控制的检查和决策，且在使用前，一个或多个主体/客体属性被更新。它使用策略(7)、策略(9)和如下 9 个策略：

(10) $permitaccess(s,o,r) \rightarrow_{\triangle} tryaccess(s,o,r) \wedge_{\triangle} preupdate(attribute)$

(11) $grant(s,dc) \rightarrow_{\triangle} permitaccess(s,o,r)$

(12) $activate(s,dc) \rightarrow_{\triangle} grant(s,dc) \wedge_{\triangle} preupdate(attribute)$

(13) $\square(\neg(\wedge_n pa_n) \wedge (state(s,dc) = usin\,g\_dc) \rightarrow inactivate(s,dc))$

(14) $activate(s,dc) \rightarrow_{\triangle} (grant(s,dc) \wedge (pw_1 \wedge pw_2 \wedge \cdots \wedge pw_i)) \wedge_{\triangle} preupdate(attribute)$

(15) $\square(\neg(\wedge_n pw_n) \wedge (state(s,dc) = grant\_dc) \rightarrow revoke(s,dc))$

(16) $\square(\neg(\wedge_n pa_n) \wedge (state(s,dc) = usin\,g\_dc) \rightarrow hold(s,dc))$

(17) $restore(s,dc) \rightarrow_{\triangle} (hold(s,dc) \wedge (pw_1 \wedge pw_2 \wedge \cdots \wedge pw_i)) \wedge_{\triangle} preupdate(attribute)$

(18) $\square(\neg(\wedge_n pw_n) \wedge (state(s,dc) = hold\_dc) \rightarrow revoke(s,dc))$

其中，$pw_i(1 \le i \le n)$是基于主体、客体、系统属性定义的谓词。

2.2.7 sg_onA$_2$模型

sg_onA$_2$模型在使用中进行使用控制的检查和决策，且在使用中，一个或多个主体/客体属性被更新。它使用策略(7)、策略(9)、策略(13)、策略(15)、策略(16)、策略(18)和如下 3 个策略：

(19) $activate(s,dc) \rightarrow_{\triangle} (grant(s,dc) \wedge (pw_1 \wedge pw_2 \wedge \cdots \wedge pw_i))$

(20) $restore(s,dc) \rightarrow_{\triangle} (hold(s,dc) \wedge (pw_1 \wedge pw_2 \wedge \cdots \wedge pw_i))$

(21) $activate(s,dc) \rightarrow \Diamond(onupdate(attribute) \wedge \Diamond(endaccess(s,o,r) \vee revoke(s,dc)))$ 或 $activate(s,dc) \rightarrow onupdate(attribute) U (endaccess(s,o,r) \vee revoke(s,dc))$

2.2.8 sg_onA$_3$模型

sg_onA$_3$模型在使用中进行使用控制的检查和决策，且在使用后，一个或多个主体/客体属性被更新。它使用策略(7)~策略(9)、策略(13)、策略(15)、策略(16)、策略(18)~策略(20)和策略(22)：

(22) $revokeaccess(s,o,r) \rightarrow \Diamond postupdate(attribute)$

SG_UCON$_A$策略可由以上所述核心模型规范表示。

3 应用实例分析

在e-Learning Grid中，构建一个软件培训虚拟组织VO$_{ST}$需要聚集广域分布的课件资源。本文将课件封装成网格服务，并将其授权表示为发起者调用操作的权限。这些服务对外提供 4 个基本操作，即读、下载、写、上传。上述操作的使用权限分别表示为*R,D,W,U*并分别被指派给 4 个基本角色r$_R$, r$_D$, r$_W$, r$_U$。

自治域AD$_1$愿意共享基础课服务。AD$_1$的资源组织片断如图 2 所示。以“应用数学AM”为例，其课件被封装成网格服务并指派给角色r$_{AM}$，而r$_{AM}$继承了角色r$_R$, r$_D$, r$_W$, r$_U$，依此类推。角色r$_{MT}$继承了角色r$_{AM}$, r$_{AS}$, r$_C$。

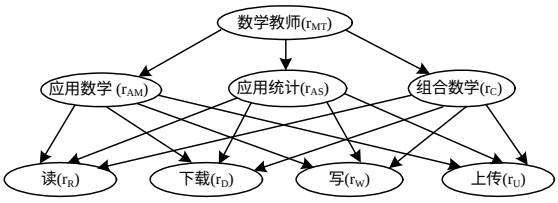

图 2 基础课角色层次示例

相同地，自治域AD$_2$愿意共享专业课服务。AD$_2$的资源组织片断及角色层次关系如图 3 所示。

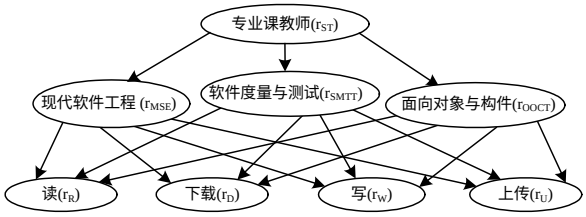

图 3 专业课角色层次示例

VO$_{ST}$根据自己的目标与AD$_1$协商获得“应用数学AM”和“应用统计AS”的读权和下载权，有效期为 20007 年 7 月 1 日~2007 年 8 月 31 日，其委托凭证为

$$dc_{VO_{ST} \leftarrow AD_1} = (n_d, n_b, (dt_{VO_{ST} \leftarrow AD_1}))$$

其中，委托票据为

$$dt_{VO_{ST} \leftarrow AD_1} = ((VO_{ST}, r`_{MT\_tree}), ([07-07-01, 07-08-31], P), n, ae, dep)$$

其中，$r`_{MT\_tree} = (r_{MT}(r_{AM}(r_R, r_D), r_{AS}(r_R, r_D)))$; $pt = ([07-07-01, 07-08-31], P)$；$pt$ 的详细解释参见文献[4]。$r`_{MT_tree}$表示由图 2 中完整的角色树 r_{MT_tree} 剪枝得到的子角色树。

VO$_{ST}$根据自己的目标与AD$_2$协商获得“现代软件工程MSE”、“软件度量与测试SMTT”和“面向对象与构件OOCT”的读权和下载权，有效期同前，其委托凭证为

$$dc_{VO_{ST} \leftarrow AD_2} = (n_d, n_b, (dt_{VO_{ST} \leftarrow AD_2}))$$

其中，

$dt_{VO_{ST} \leftarrow AD_2} = ((VO_{ST}, r`_{ST\_tree}), ([07-07-01, 07-08-31], P), n, ae, dep)$；

$r`_{ST\_tree} = (r_{ST}(r_{MSE}(r_R, r_D), r_{SMTT}(r_R, r_D), r_{OOCT}(r_R, r_D)))$；

$r`_{ST\_tree}$的解释类似 $r`_{MT_tree}$。

上述委托凭证未给出具体值的参数，在本例中对授权无限制。VO$_{ST}$制定的一条授权策略如下：注册用户可以浏览课件；每人每周可对每门课的课件浏览 5 次，每次浏览不能超过 30 min；每门课的课件被同时浏览的人数不能超过 60 个。

笔者定义的部分属性如下：主体属性包括“委托凭证*dc*”、“主体类别*cid*”、“浏览时间*bt*”、“浏览次数bn_o(用课件名做下标)”、“开始使用时间start”，其中，*cid*能取 2 种值，即“注册用户*reg*”和“注册并缴费用户*enr*”。客体属性包括“正在浏览的主体数量*bsn*”。系统属性包括“系统时钟clock”。定义常量为*DD*=07-07-01，*UD*=07-08-31。客体*o*的取值范围为$o \in \{AM, AS, MSE, SMTT, OOCT\}$。

上述授权策略用SG_UCON$_A$核心模型规范表示如下：

(1) $permitaccess(s,o,R) \rightarrow_{\triangle} (tryaccess(s,o,R) \wedge (s.CID = reg) \wedge (o \in \{AM, AS, MSE, SMTT, OOCT\}) \wedge preupdate(s.dc))$；

$preupdate(s.dc):switch(o)$

$\{case\ \ o==AM:\ dc=(n_d,n_b,(dt_{VO_{ST}\leftarrow AD_1}(dt_{s\leftarrow VO_{ST}})))$，$dt_{s\leftarrow VO_{ST}}=((s,(r_{AM}(r_R))),([DD,UD],P),n,ae,dep)$；

$case\ \ o==AS:\ dc=(n_d,n_b,(dt_{VO_{ST}\leftarrow AD_1}(dt_{s\leftarrow VO_{ST}})))$，$dt_{s\leftarrow VO_{ST}}=((s,(r_{AS}(r_R))),([DD,UD],P),n,ae,dep)$；

$case\ \ o==MSE:\ dc=(n_d,n_b,(dt_{VO_{ST}\leftarrow AD_2}(dt_{s\leftarrow VO_{ST}})))$，$dt_{s\leftarrow VO_{ST}}=((s,(r_{MSE}(r_R))),([DD,UD],P),n,ae,dep)$；

$case\ \ o==SMTT:\ dc=(n_d,n_b,(dt_{VO_{ST}\leftarrow AD_2}(dt_{s\leftarrow VO_{ST}})))$，$dt_{s\leftarrow VO_{ST}}=((s,(r_{SMTT}(r_R))),([DD,UD],P),n,ae,dep)$；

$case\ \ o==OOCT:\ dc=(n_d,n_b,(dt_{VO_{ST}\leftarrow AD_2}(dt_{s\leftarrow VO_{ST}})))$，$dt_{s\leftarrow VO_{ST}}=((s,(r_{OOCT}(r_R))),([DD,UD],P),n,ae,dep)\}$；

(2) $grant(s,dc)\rightarrow_{\triangle} permitaccess(s,o,R)$；

(3) $activate(s,dc)\rightarrow_{\triangle}(grant(s,dc)\wedge(o.bsn<60)\wedge(s.bn_o<5)\wedge preupdate(o.bsn)\wedge preupdate(s.bn_o)\wedge preupdate(s.bt)\wedge preupdate(s.start))$，

$preupdate(o.bsn):o.bsn=o.bsn+1$，

$preupdate(s.bn_o):s.bn_o=s.bn_o+1$，

$preupdate(s.bt):s.bt=0$，

$preupdate(s.start):s.start=sys.clock$；

(4) $\Box((state(s,dc)=using\_dc)\wedge(s.bt<30)\rightarrow onupdate(s.bt))$，

$onupdate(s.bt):s.bt=sys.clock-s.start$；

(5) $\Box((s.bn_o\le 4)\wedge(s.bt>30)\wedge(state(s,dc)=using\_dc)\rightarrow inactivate(s,dc))$;

(6) $inactivate(s,dc)\rightarrow\Diamond postupdate(o.bsn)$，

$postupdate(o.bsn):o.bsn=o.bsn-1$；

(7) $\Box((s.bn_o>4)\wedge(s.bt>30)\wedge(state(s,dc)=using\_dc)\rightarrow hold(s,dc))$;

(8) $hold(s,dc)\rightarrow\Diamond postupdate(o.bsn)$；

(9) $restore(s,dc)\rightarrow_{\triangle}(hold(s,dc)\wedge(s.bn_o==0)\wedge preupdate(o.bsn)\wedge preupdate(s.bn_o)\wedge preupdate(s.bt)\wedge preupdate(s.start))$；

(10) $\Box((sys.clock>UD)\wedge(state(s,dc)=using\_dc)\rightarrow revoke(s,dc))$；

(11) $\Box((sys.clock>UD)\wedge(state(s,dc)=grant\_dc)\rightarrow revoke(s,dc))$；

(12) $\Box((sys.clock>UD)\wedge(state(s,dc)=hold\_dc)\rightarrow revoke(s,dc))$；

(13) $revokeaccess(s,o,r)\rightarrow_{\triangle} revoke(s,dc)$；

(14) $activate(s,dc)\rightarrow\Diamond endaccess(s,o,r)$；

(15) $revokeaccess(s,o,r)\vee endaccess(s,o,r)\rightarrow\Diamond postupdate(s.dc)$，$postupdate(s.dc):s.dc=null$.

4 结束语

本文通过实例分析并展示改进模型的表达能力。在有效使用期中，委托凭证只在主体第 1 次请求时产生，除非主体主动撤销，否则必须到有效使用期满时才会将其销毁。在销毁前，委托凭证可以依据系统状态的变化转换到不同凭证处理状态。

参考文献

[1] Park J, Sandhu R. Towards Usage Control Models: Beyond Traditional Access Control[C]//Proceedings of the 7th ACM Symposium on Access Control Models and Technologies. Monterey, California, USA: ACM Press, 2002: 57-64.

[2] Park J, Sandhu R. The UCONABC Usage Control Model[J]. ACM Transactions on Information and System Security, 2004, 7(1): 128-174.

[3] Zhang Xinwen, Parisi-Presicce F, Sandhu R, et al. Formal Model and Policy Specification of Usage Control[J]. ACM Transactions on Information and System Security, 2005, 8(4): 351-387.

[4] 徐　震, 李　斓, 冯登国. 基于角色的受限委托模型[J]. 软件学报, 2005, 16(5): 970-978.

(上接第 69 页)

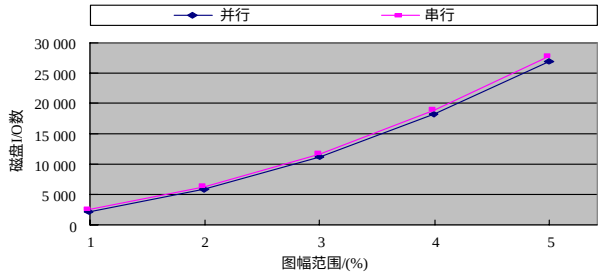

图 4　磁盘 I/O 对比

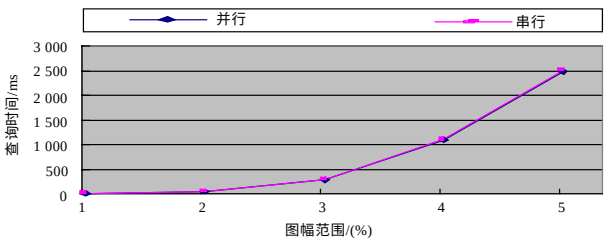

图 5　查询时间对比

由图 4、图 5 可以看出，并行构建的 R 树，其磁盘 I/O 次数和查询时间均高于串行 R 树。这是因为并行 R 树在构建之前人为地将图幅分区，使同一个分区的数据在 R 树中处于相对集中的索引包内。当查询窗口落在分区内部时，搜索路径会相对减少，因此，提高了查询效率。当查询窗口落在分区之间，最不理想的情况是查询窗口跨越 4 个分区，搜索路径会相对增加。但各分区数据量的极度不均衡将导致各子 R 树深度不同，合并后的树不再是平衡树，查询效率将降低。因此，R 树的并行构建必须基于合理的数据分区，以保证并行计算时的负载平衡以及合并后 R 树的平衡。

4 结束语

R 树索引的创建是并行度很高的计算过程，具体如下：按空间数据范围将数据分区存储后，对各分区进行并行处理，充分利用数据库的并发能力，提高并行计算过程中各子处理过程的数据访问性能，采用常用的 DCSO 策略，对结果进行合并输出。

参考文献

[1] 张明波, 陆　锋, 申排伟, 等. R 树家族的演变和发展[J]. 计算机学报, 2005, 28(3): 289-300.

[2] Kamel I, Faloutsos C. On Packing R-trees[C]//Proceedings of the 2nd International Conference on Information and Knowledge Management. Washington, D. C., USA: [s. n.], 1993.

[3] 张立立. 海量地理空间数据的高性能计算[Z]. 中国科学院地理资源与环境研究所, 2005.

[4] Theodoridis Y, Sellis T. A Model for the Prediction of R-tree Performance[C]//Proceedings of the 15th ACM Symposium on Principles of Database Systems. Montreal, Canada: ACM Press, 1996: 161-171.