The Study of MIB Management Algorithm in Extensible Agent

XU Xudong, QIN Riqiang, WANG Yaxiu
Computer college, Beijing University of Technology, Beijing, China
e-mail: xuxudong@bjut.edu.cn, qinriqiang198575@163.com, wangyx019@163.com

Abstract: MIB is the main operating target of extension agent. MIB management performance directly determines the overall performance of the agent. By analyzing the characteristics of extension agent's MIB, this paper proposes a multi-domain binary sort tree storage strategy and studies the search and traversal algorithm of MIB node under the storage strategy. Analysis shows that the strategy can effectively improves search and traversal efficiency of MIB node. For solving the issue of MIB region registered conflict, this paper proposes region decomposition strategy and ensures the flexibility and reliability of the agent.

Keywords: storage; register; extension agent; MIB

可扩展代理中 MIB 管理算法的研究

徐旭东，秦日强，王亚秀
北京工业大学计算机学院，北京，中国，100124
e-mail: xuxudong@bjut.edu.cn, qinriqiang198575@163.com, wangyx019@163.com

【摘　要】MIB 是可扩展代理的主要操作对象，MIB 管理性能直接决定了可扩展代理的整体性能。本文通过分析可扩展代理 MIB 的特点，提出一种多域二叉排序树 MIB 结点存储方案，并研究了该存储方案下的 MIB 结点查找、遍历算法；分析表明，该方案能有效提高了 MIB 管理结点的查找、遍历效率；针对可扩展代理中的 MIB 区域注册冲突问题，提出区域分解策略，保证了可扩展代理的灵活性与可靠性。

【关键词】存储；注册；可扩展代理；MIB

1 引言

可扩展代理的主要操作对象是 MIB[1] (Management Information Base)，网络管理系统要实现的管理功能，最终是通过访问 MIB 中的管理对象实现的。从本质上看，MIB 管理性能的优劣直接决定了整个网络管理系统性能的好坏。可扩展代理中 MIB 的管理主要有三部分：存储、访问及注册；MIB 的存储策略决定了管理对象的查找访问效率，查找访问效率决定了整个可扩展代理的性能；通过注册，主代理实现管理对象的扩展，保证可扩展代理的灵活性。

基金项目：北京市教育委员会学科与研究生教育建设项目资助
作者简介: 徐旭东（1961-），男，副教授、硕士,主研方向:计算机网络与通信；秦日强，男，硕士，研究方向：计算机网络管理；王亚秀，女，硕士，研究方向：计算分析.

针对用户自定义的网络管理对象，本文研究了可扩展代理中 MIB 管理算法，提出了多域二叉排序树 MIB 存储方案，研究了该存储方案下的 MIB 存储、遍历算法，并详细讨论了可扩展代理中注册区域冲突的解决方案，保证 MIB 管理的可靠性。

2 MIB 存储

根据 RFC 的标准定义，MIB 采用与域名系统结构相似的树形结构形式的组织起来，树中的每个结点都有一个唯一标识 OID(Object Identifier)，该标识采用从根结点到该结点的路径唯一标识。不同的代理实现方案，MIB 存储结构不同，其中多路径树与 MIB 结构最相似，但这种结构的查找效率不高，遍历复杂，为此本文采用多域二叉排序树存储结构，提高整体效率。

978-1-935068-06-8

2.1 字典序定义

可扩展代理中管理对象包括两部分：SNMP 框架中标准 MIB 对象及 AgentX 协议[2]框架中的 AgentXNode 对象，都用对象标识符 OID 来表示。一个具体的管理请求操作，最终是通过对网络管理对象 OID 的查找访问操作完成的。为了方便数据报文处理及 MIB 的查找访问等操作，特引入 OID 的字典顺序定义 1，来表示结点之间的关系。

定义 1：假设两个对象标识符 oid_X 和 oid_Y，其中 $oid_X=(x_1.x_2.x_3...x_m)$，$oid_Y=(y_1.y_2.y_3...y_n)$，若存在 $k(k\le m,k\le n)$，使得 $x_i=y_i(1\le i<k)$ 且 $x_k<y_k$，或者存在 $k(1\le k\le m,\ m<n)$，使得 $x_k=y_k$，则 $oid_X<oid_Y$；整个 MIB 中的管理对象结点可分为四类：简单结点、数据结点、表格 *entry* 结点和组结点；简单结点不保存实际数据而只包含一个数据结点。数据结点分两种：简单数据结点和表格数据结点。简单数据结点与简单结点一一对应，它保存实际数据[3]；表格数据结点保存对应表格中的某行数据；表格 *entry* 结点指向表格数据的入口。由于数据结点保存子代理注册的 MIB 管理信息，因此作为本文主要的研究对象，多个结点组成一个 *context* 域，整个 MIB 由一个或多个 *context* 域组合而成。

2.2 表格数据结点的定义

MIB 表格数据结点是通过行和列来描述的，表格 *entry* 结点指向表格数据的入口,表格数据结点由于行、列索引唯一确定。

定义 2：设任意含有 α 行 β 列数据的表格 T，$\alpha,\beta\in Z$ 且 $\alpha\ge 0,\beta\ge 1$。设表格 T 的 OID 为 oid_T，第 m 行 n 列单元格对象 $T(m,n)$ 的 OID 值为 $oid_{m,n}$，则：

$$oid_{entry}=oid_T.id_{entry} \quad (1)$$

$$oid_{m\times n}=oid_{entry}.index_n.index_m \quad (2)$$

其中 oid_{entry} 是表格 *entry* 结点 OID，id_{entry} 为表格 *entry* 结点 *id*，$index_n$ 为第 n 列索引值，$index_m$ 是第 m 行的索引值。

由管理对象 OID 的定义可知，简单数据结点在整个 MIB 中是唯一存在的，但对于表格数据结点，由于表格中的所有结点有着共同的表格 OID 前缀 oid_T，如果每个表格结点都完整地存储它的 OID，则一个有 m 行 n 列的表格，表格前缀 oid_T 要重复存储 $m\times n$ 次；当表格很大时，这样的存储方式显然是对存储空间的巨大浪费，降低了表格结点的存储效率，同时搜索树的结点数和键值的比较时间也大大增加，降低了管理操作对结点的访问效率。

3 多域二叉排序树存储

为了便于管理对象结点的查找、遍历，平衡查找与遍历的性能，本文提出一种多域二叉排序树存储方案，主代理运行期间，整个 MIB 结构维护着一个 *context* 集合，每个 *context* 域中的结点集合，按 OID 字典序，构成一个二叉排序树，其结构图如图 1 所示。

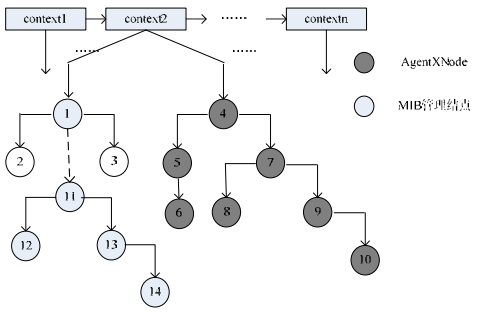

Figure 1. Master agent MIB storage structure
图 1. 主代理 MIB 存储结构图

二叉排序树便于结点的查找、遍历，适合于可扩展代理对 MIB 管理对象的访问操作。非表格数据结点按 OID 大小，分布在所在 *context* 域对应的二叉排序树中；对于表格数据结点，为了提高结点存储效率，本文引入表格数据结点的 AVL 树存储，表格 *entry* 结点是排序二叉树中的结点，保存着指向该 AVL 子树根结点，如图 1 中结点 1 为表格 *entry* 结点，指向以结点 11 为根的 AVL 子树。表格数据结点以列、行索引构成的 OID 大小顺序存储，这样避免了表格结点前缀 oid_{entry} 的重复存储，大大减少了表格数据结点 OID 的存储空间，同时也减少了查找遍历时 OID 值的比较次数，提高了效率。

采用该存储结构，非表格数据结点的查找过程比较简单，首先根据请求报文中公共体字符串或 *contextname*，找到该结点所在的 *context* 域，根据该结点 OID，搜索该 *context* 域对应的二叉排序树，找到相应的结点即可。表格数据结点的查找过程较复杂，设查找结点的 OID 为 $oid_{entry}.index_c.index_r$（$index_c$，$index_r$ 分别是该表格数据结点的列索引及行索引）则结点查找过程如下：

(1) 根据请求报文找到该结点所在的 *context* 域，

搜索相应的二叉排序树，找到 OID 为 oid_{entry} 的表格 *entry* 结点，如果找到该结点，转步骤(2)，否则，返回空值，查找失败。

(2)以 $index_c.index_r$ 为 OID，搜索表格 *entry* 结点所指向的 AVL 子树，如果找到该结点，则查找成功，返回相应值，否则，返回空值，查找失败。

二叉排序树存储的性能优越，结点查找时间复杂度为 $o(\log n)$，与大多数结点查找算法(如哈希算法、二分法等)相比，查找效率高，易于实现结点遍历；AVL 树结构存储表格数据结点，可减少存储空间的使用，提高性能；树型结构本身便于结点的动态添加及删除，适合可扩展代理动态扩展。

4 MIB 遍历

MIB 遍历是所有网络管理系统不可或缺的部分，一般通过 *GetNext* 请求操作来完成，其作用是查找 PDU 中给定 OID 后继对象的值。请求 OID 的后继对象是按字典序定义给出的所有 OID 大于请求 OID 的对象中最小的那个对象。

通过上述 MIB 存储结构分析可知，对于查找给定 OID 后继对象，这里把查找范围限制在同一个 *context* 域，只须中序遍历相应的查找搜索树即可。表格数据结点的遍历顺序是先列后行，这与上述列、行索引结构一致。在列号不变的情况下，OID 的后继结点即是目标结点；若未能找到后继结点，但列号未达到最大，则下一列的最小索引行结点为目标结点。设当前请求 OID 为 *cur*，则遍历算法如下：

(1) 若 *cur* 为简单数据结点或组结点，通过中序遍历该二叉排序树结点序列，找到 *cur* 的后继结点 *seq*，令 $cur = seq$，若 *cur* 为表格 *entry* 结点，转步骤(2)，否则，重复此步骤，直到该二叉排序树的最后一个结点，返回空值。

(2) 若 *cur* 为表格 *entry* 结点，根据前面讨论的表格结点的存储策略，中序遍历表格 *entry* 结点指向的 AVL 树，这里保证表格数据结点 OID 小于表格 *entry* 结点所在二叉排序树的后续结点 *seq*，令 $cur = seq$，若 *cur* 为非表格 *entry* 结点，转步骤(1)，否则，重复此步骤，直到该二叉排序树的最后一个结点，返回空值。

根据该算法，逐一遍历 MIB 维护的 *context* 集合，即可实现整个 MIB 的遍历。

5 MIB 注册

5.1 注册冲突

可扩展代理中，MIB 结点的动态注册是核心，只有子代理成功完成注册，主代理才能获得足够的信息，才知道如何将 SNMP 请求报文转化成相应的 AgentX 报文，才能进行正确的消息转发[4]。由于主代理允许不同的子代理注册同一区域，因此在子代理注册过程中会出现区域重叠，造成 MIB 注册区域冲突。如何解决区别冲突是可扩展代理框架中的关键问题。

子代理通过发送 AgentX_Register_PDU 来实现注册流程。在主代理端，如果新注册的 MIB 子树对原有注册的子树没有影响则直接返回 AgentX_Response 报文；否则按以下流程进行处理。

(1) 如果不同的子代理注册的 MIB 域完全相同，即重复域，则进行 MIB 注册域优先级的比较；如果该优先级也相同，则主代理拒绝此次注册请求；否则，以优先级高的 MIB 域作为有效域。

(2) 如果当前子代理注册的 MIB 域与已存在的 MIB 域产生重叠，即重叠域，则需要对原有区域及现有区域进行合理取舍，即区域分解。

5.2 区域分解

每个子代理可以与主代理建立一个或多个会话(Session)，同一个会话可以注册一个或多个 MIB 区间。针对重叠区域注册，MIB 区域分解有两种情况：区域包含和区域相交。

设存在两个 Session 进行区域注册，Session A 注册区域 *A*[*oid1*,*oid2*]，Session B 注册区域 *B*[*oid3*,*oid4*]，且 Session A 已成功注册，主代理 MIB 已存在区域 *A* 对应的 AgentXNode—*originNode*。

5.2.1 区域包含

如果区域 *A* 包含区域 *B*，则当 Session B 进行注册时，主代理 MIB 进行区域分解，如图 2 所示。

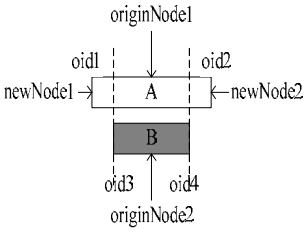

Figure 2. Regional decomposition when *A* contains *B*.

图 2. *A* 包含 *B* 的区域分解

978-1-935068-06-8 © 2009 SciRes.

Session A 注册的区域被分解成三部分：[*oid1*,*oid3*)，[*oid3*,*oid4*]和(*oid4*,*oid2*]，而对应的AgentXNode-*originNode* 被分解成个三个AgentXNode：*newNode1*，*originNode1* 和 *newNode2*；其中 *newNode1*，*newNode2* 作为新的 AgentXNode 注册到 MIB 子树中。如果区域 *B* 的注册优先级大于区域 *A* 的注册优先级，则用 *originNode2* 替换原来的 *originNode*，否则用 *originNode1* 替换 *originNode*，并更新主代理的消息分发模块。

如果区域 *B* 包含区域 *A*，如图 3 所示，则对区域 *B* 进行分解，处理过程与前面所述类似，在此不再赘述。

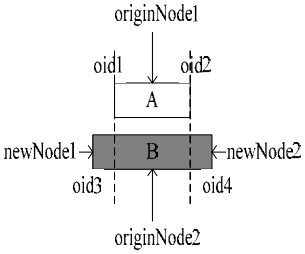

Figure 3. Regional decomposition when *B* contains *A*
图 3. *B* 包含 *A* 的区域分解

5.2.2 区域相交

区域相交有两种：左相交与右相交；当区域 *A* 的下界大于区域 *B* 的下界，且小于区域 *B* 的上界，则称左相交；当区域 *B* 的下界大于区域 *A* 的下界，且小于区域 *A* 的上界，则称右相交。

对于左相交，Session B 注册区域 *B*，区域分解如图 4 所示。区域 *A* 被分解成两部分：[*oid1*,*oid4*]，(*oid4*,*oid2*]；区域 *B* 也被分解成两部分：[*oid3*,*oid1*)，[*oid1*,*oid4*],。分解产生的两个 AgentXNode：*newNode1* 和 *newNode2*，作为新的 AgentXNode，插入到适当的

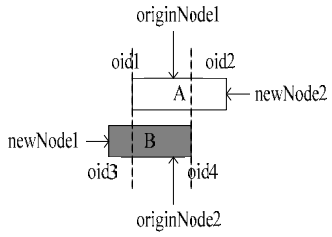

Figure 4. Regional decomposition of the left intersection of region
图 4. 左相交区域分解

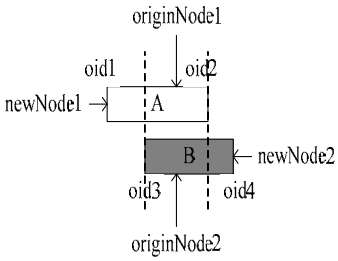

Figure 5. Regional decomposition of the right intersection of region
图 5. 右相交区域分解

MIB 子树中；而对于重叠域，则通过比较注册优先级，用 *originNode1* 或 *originNode2*，来替换原来的 *originNode*，同时更新主代理消息分发模块。右相交区域分解如图 5 所示，结点处理过程与左相交相似，在此不再赘述。

6 结论

本文提出了一种多域二叉排序树 MIB 结点存储方案，并研究了该存储方案中 MIB 结点的存储、遍历算法；通过引入表格数据结点的 AVL 树形结构存储，能有效降低表格结点存储空间的消耗，提高查找、遍历效率；同时，讨论了 MIB 区域分解算法，有效解决了可扩展代理中 MIB 注册区域冲突问题，保证了系统的可靠性。

致谢

特别感谢中创信测 3G Traffic 项目组各位老师的指导和帮助及软件学科部的廖湖声老师，感谢提供宝贵意见和帮助的所有同学。

References (参考文献)

[1] McCloghrie K, Management information base for network management of TCP/IP-based internet: MIB-II[S].RFC1213, IETF, 1991.

[2] Daniele M,Wijnen B,Ellison M,Francisco D, Agent Extensibility (AgentX) Proctocol[S].RFC2741, January 2000

[3] Zhou Jian, Zhang Xiao-tong, Wang Qin，Dynamic MIB Structure of SNMP and High Performance Search Algorithm[J], Computer Engineering, 2008, 34(2), P171-174.
周剑，张晓彤，王沁，SNMP 协议动态 MIB 结构与高效查找算法[J]，计算机工程，2008，34(2)，P171-174

[4] Xiao Zhi-bin, The Research and Implementation of Multi-Agent based on SNMP[Z] , Dissertation of Xi'an University of Electronic Science and Technology, 2007, 5-3: P35-43.
肖志彬，SNMP 协议中多代理的研究与实现[Z]，电子科技大学学位论文，2007，5-3: P35-43.