基于分层布隆过滤器的发布订阅自动发现算法

耿 宏，李勇猛+

（中国民航大学 电子信息与自动化学院，天津 300300）

摘 要：针对当前简单发现算法（SDP）中线性链表式存储结构对多层节点查找时延偏高和内存占用大的问题，提出一种基于分层布隆过滤器的发布/订阅自动发现算法（HBF_ADA）。使用动态布隆计数树（DBCT）型数据结构，根据添加元素个数动态创建节点，把待查询元素录入为集合，利用集合的相交运算判断元素是否存在于网络；对每个字节增加一个计数器，记录对应字节被置位的次数，解决布隆过滤器元素删除困难的问题。实验结果表明，HBF_ADA 算法的时延参数和内存消耗明显低于 SDP 和 BF_SDP，能够删除已存在的节点信息，在允许低误报率的情况下，满足现实应用的需求。

关键词：数据分发服务；自动发现算法；分层布隆过滤器；发布/订阅；误报率

中图法分类号：TP393 **文献标识号**：A **文章编号**：1000-7024（2019）12-3494-06

doi：10.16208/j.issn1000-7024.2019.12.022

Publish subscribe automatic discovery algorithm based on hierarchical Bloom filter

GENG Hong, LI Yong-meng+

(College of Electronic Information and Automation, Civil Aviation University of China, Tianjin 300300, China)

Abstract: Aiming at the problems that the linear linked list storage structure in the current simple discovery protocol (SDP) has high latency and large memory occupation in searching for multi-layer nodes, a publish/subscribe automatic discovery algorithm based on hierarchical Bloom filter (HBF_ADA) was proposed. A data structure based on dynamic Bloom count tree (DBCT) was used, which dynamically created nodes according to the number of joined elements, and the queried elements were added to the set, the set intersection operation was applied to determine whether the element existed in the network. To solve the problem that the Bloom filter element is difficult to delete, a counter was added to each byte to record the number of times the corresponding byte was set. The simulation results show that the latency and memory consumption of the proposed algorithm are significantly lower than SDP and Bloom filter simple discovery protocol (BF_SDP). The existing node information can be deleted. When low false positive rate is allowed, the algorithm meets the needs of practical applications.

Key words: data distribution service (DDS); automatic discovery algorithm; hierarchical Bloom filter; publish/subscribe; false positive rate

0 引 言

数据分发服务（data distribution service，DDS）是国际对象管理组织（object management group，OMG）针对分布式系统提出的以数据为中心的通信模型[1]。该模型采用发布/订阅通信模式，允许参与通信的末端节点相互分离，即各节点能够动态地加入和退出分布式应用，这就实现了分布式系统时间、空间和同步关系 3 个方面的完全解耦。目前，DDS 已被广泛应用于航空、航天等分布式实时控制系统。自动发现协议（automatic discovery algorithm，ADA）是 DDS 的核心技术，通过该协议能够对分布式系统中动态加入的节点及时发现并识别。

针对自动发现过程，国内外已提出多种自动发现协议。RTI（real-time innovations）根据 RTPS 标准提出一种简单发现算法[2]（simple discovery protocol，SDP），该算法包

收稿日期：2018-12-07；**修订日期**：2019-01-10

基金项目：中美绿色航线合作基金项目（GH201661279）

作者简介：耿宏（1964-），男，天津人，硕士，教授，研究方向为航空系统优化与仿真；+通讯作者：李勇猛（1992-），男，河北唐山人，硕士研究生，研究方向为虚拟维修仿真。E-mail：1299135391@qq.com

含参与者发现和端点发现两个阶段，在小型分布式系统中能发挥很好的作用，但在中大规模分布式系统中，尤其是在节点众多且匹配率不高的情况下，效果欠佳。An Kyoungho等提出一种基于布隆过滤器的自动发现算法[3] (Bloom filter simple discovery protocol，BF_SDP)，该算法将端点匹配过程提前，即在参与者发现阶段完成，因此对于不匹配的节点信息不需要再发送给远程参与者，从而减少了内存资源的浪费。但对于多层节点的分布式系统，BF_SDP 无法对子节点进行识别，且无法删除已存在的节点信息。翟海波等提出一种基于服务力向量[4]的自动发现算法，该算法利用 QoS（quality of service）策略进行匹配，减少了不必要信息的传输和存储。

在分布式系统中，节点信息大多存储在线性链表式结构中，当查找某个特定元素时需遍历整个链表才能找到相应元素，但对于多层节点，链表式结构查找时延偏高且内存消耗过大。为了实现对分布式系统中各层节点的高效查询并降低内存消耗，本文提出一种具有分层数据结构的自动发现算法。

1　算法原理

1.1　分层布隆过滤器

布隆过滤器[5] (Bloom filter，BF) 是一种节约空间的二进制数据结构，它能够将集合中的元素 $x \in S(S=\{x_1, x_2, x_3, \cdots, x_n\})$ 映射到二进制向量 V 中，也可以用于判断某个元素 x_i 是否属于集合 V。但 BF 存在一定的误报率 (false positive，FP)，即将不属于 S 中的元素误认为属于 S。对于 m 位长的 BF，使用 k 个哈希函数，添加 n 个元素后，误报率 FP 由式 (1) 计算得出

$$FP=\left[1-\left(1-\frac{1}{m}\right)^{kn}\right]^k \approx (1-e^{-\frac{kn}{m}})^k \tag{1}$$

根据计算[6]可得出，当 $k=\frac{m}{n}(\ln 2)$ 时，FP 取得最小值，即 $FP_{\min}=2^{-\frac{m}{n}(\ln 2)}$。

BF 支持集合元素的添加和查询，但不支持元素的删除。多个 BF 之间的组合和拆分可以通过位运算“*OR* (或)”和“*AND* (与)”实现[7]。

定理 1　如果 $\beta(A)$，$\beta(B)$ 和 $\beta(A \cup B)$，$\beta(A \cap B)$ 使用相同的 m、H 和 k，则 $\beta(A \cup B)=\beta(A) \cup \beta(B)$，$\beta(A \cap B)=\beta(A) \cap \beta(B)$。

分层布隆过滤器[8] (hierarchical Bloom filter，HBF) 是在布隆过滤器的基础上提出的一种树形数据结构，该结构支持多层节点的查询。HBF 以命名空间 (Namespace) 为基础，每个节点 (Node) 都是命名空间的一个子集，一旦数据结构被建立，就可以将集合中的各个元素添加到相应节点，HBF 仅需建立一次，即能重复使用。

表 1 给出文中使用的变量定义。

表 1　文中使用的变量定义

名称	定义
x	元素
S	元素 x 所构成的集合
N	集合 S 所对应的命名空间 $\forall x_i \in S, x_i \subseteq \mathrm{N}$
N	命名空间大小 $\lvert\mathrm{N}\rvert=N$
H	哈希函数
β	布隆过滤器
k	哈希函数个数
n	S 中元素个数 $\lvert S\rvert=n$
m	布隆过滤器位长
Γ	布隆过滤树
$N_\perp$	Γ 中叶节点存储的元素个数
β_{ij}	Γ 中第 i 行第 j 列所对应的布隆过滤器

1.2　数据存储结构

在分层布隆过滤器的基础上，定义一种树形数据结构—动态布隆计数树[9] (dynamic Bloom count tree，DBCT)，该数据结构能够根据添加元素的个数动态调整大小，以节省内存空间。同时，又通过对布隆过滤器中每个字节后增加一个计数器，以支持元素的删除。首先引入布隆过滤树 (Bloom filter tree，BFT) 的概念。

1.2.1　BFT 数据结构

BFT 是一个二进制树形数据结构，用 Γ 表示，有 $\log_2\left(\frac{N}{N_\perp}+1\right)$ 层，BFT 的每个节点都对应一个 BF，用于存储命名空间的一部分，且每个 BF 含有相同的参数 m，H 和 k，给出 BFT 的具体定义：

定义 1　给定命名空间 N 及其大小 N，BF 的大小为 m，哈希函数 $H_i(1 \leqslant i \leqslant k)$，每个节点存储元素个数为 $N_\perp$ $(N_\perp \leqslant N)$，BFT 用符号 $\Gamma(N \cdot m \cdot H \cdot N_\perp)$ 表示，则 $\Gamma=\left\{\beta_{ij} : 1 \leqslant i \leqslant \log_2\left(\frac{N}{N_\perp}+1\right), 1 \leqslant j \leqslant 2^i\right\}$。

图 1 为 N=14 的布隆过滤树，每个节点对应一个大小为 10 个字节的 BF，且每个 BF 存储两个元素，查询元素集合 $S=\{A1, B3, C6\}$，对应查询过滤器 $b=[1001001011]$，查询过程从根节点开始逐次向下直至最后一层叶节点。

1.2.2　计数结构

由于 BF 为二进制数据结构，不同的元素经过 Hash 后可能得到相同的位编号，即同一位置被置位多次，这就导致 BF 不支持元素的删除。本文引入布隆计数树 (Bloom count tree，BCT) 的概念，即对 BFT 中每个字节后增加一个计数器，使其支持元素的删除。BCT 工作方式和 BF 相同，但它可以记录元素的添加和删除，当添加元素时，被置位字节计数器加 1；当删除元素时，被置位字节计数器减

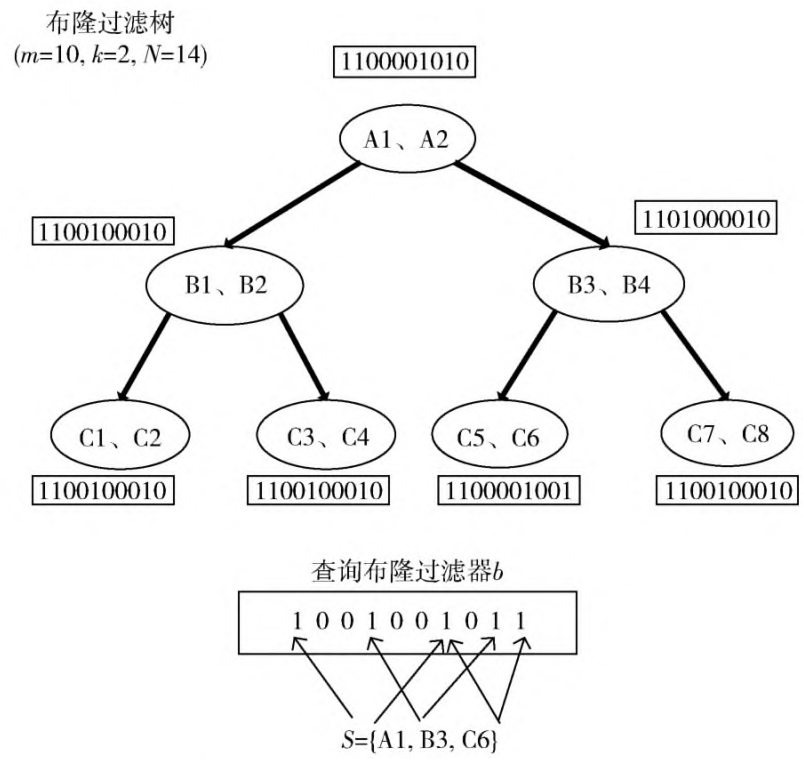

图 1 布隆过滤树和查询布隆过滤器 b

1，直到计数器减为 0 时，相应的字节才会被置为 0。为了避免计数器内存溢出，我们要确定计数器的内存大小，假设集合中有 n 个元素，k 个哈希函数，m 个计数器，计数器初始值为 0，用 $c(i)$ 表示第 i 个计数器，则第 i 个计数器增加 j 次的概率为二项式随机分布

$$P(c(i)=j)=\binom{nk}{j}\left(\frac{1}{m}\right)^{j}\left(1-\frac{1}{m}\right)^{nk-j} \tag{2}$$

第 i 个计数器大于 j 的概率可以限定为

$$P_r(\max(c)\geqslant j)\leqslant m\binom{nk}{j}\left(\frac{1}{m}\right)^{j}\leqslant m\left(\frac{enk}{jm}\right)^{j} \tag{3}$$

k 取最优值 $\frac{m}{n}(\ln 2)$，可得

$$\Pr(\max(c)\geqslant j)\leqslant m\left(\frac{e\ln 2}{j}\right)^{j} \tag{4}$$

若计数器的字节数为 4，则达到 16 时会溢出，概率[10]为

$$\Pr(\max(c)\geqslant 16)\leqslant 1.37\times 10^{-15}\times m \tag{5}$$

此概率值已足够小，满足大多数应用要求。

1.2.3 动态数据结构

在大多数分布式系统中，集合元素是动态加入和退出的，这就导致在建立 BF 时无法确定最佳变量值 m 和 k，进而产生较大的 FP，且浪费大量内存资源。针对此问题，本文提出动态数据存储结构，该数据结构的建立是基于 $s\times m$ 字节的动态矩阵，矩阵包含 s 个大小为 m 字节的 BF，其中，s 随元素的加入和退出动态变化，s 初始值为 1，当添加元素的个数超过给出的限定值时，s 值加 1，即新的 BF 被创建，元素会顺次添加到新的 BF 中。

2 算法描述

针对 SDP 中线性链表式存储结构对多层节点查找时延高且内存占用大的问题，本文提出基于分层布隆过滤器的自动发现算法 HBF_ADA，该算法对 SDP 算法中的参与者数据包结构进行改进，即将线性链表式存储结构改为 DBCT 结构用于存储节点信息，但对 SDP 中原有的两个发现阶段：简单参与者发现阶段（simple participant discovery protocol，SPDP）和简单端点发现阶段（simple endpoint discovery protocol，SEDP）不作改动。首先给出 DBCT 构建过程。

2.1 DBCT 构建过程

DBCT 是一种树形数据结构[11]，它能够记录元素的添加和删除，且能够根据命名空间的使用情况而动态变化。DBCT 的每个节点都有一个 BF，用于存储命名空间的一部分。DBCT 的构建算法见表 2。

表 2 DBCT 构建算法

算法 1：DBCT 构建算法

输入：参数 N、N' 和 $N_{\perp}$
输出：DBCT 结构 T
$Q\leftarrow Node\left(\log_2\left(\frac{N}{N_{\perp}}+1\right),1\right)$;
$Q'\leftarrow Node\left(\log_2\left(\frac{N'}{N_{\perp}}+1\right),1\right)$;
/*将命名空间 N 和已使用命名空间 N' 分别存储到 $\log_2\left(\frac{N}{N_{\perp}}+1\right)\times 1$ 维和 $\log_2\left(\frac{N'}{N_{\perp}}+1\right)\times 1$ 维 BF 中*/
repeat
  $node(a,b)\leftarrow dequeue(Q)$;
  if($node(a,b)\bigcap Q'\neq\phi$) then
    creat tree node $\beta_{a,b}$;
    insert $node(a,b)\bigcap Q'$ in $\beta_{a,b}$;
    $C_{a,b}=C_{a,b}+1$;
    /*把相交元素添加到该节点所对应的 BF 中，且节点中被置位字节计数器加 1*/
    if($b<\log_2 N$)then
      enqueue(Node(a,b));
      enqueue(Node(a,b+1));
    /*若 Q 的行数大于 DBCT 行数，继续建立下层节点，直至 Q 中元素全部被取出*/
    end
  end
until Q is empty;

2.2 DBCT 查询过程

首先将待查询元素集合 S 添加到查询布隆过滤器（query Bloom filter，QBF）中，然后判断 QBF 与 DBCT 各节点所对应 BF 是否相交，若相交，则说明相交元素存在于 DBCT 中，若不相交，则说明集合 S 中对应元素不存在于 DBCT 中，查询过程从根节点开始逐次向下直至最后一层叶节点[12]。DBCT 中元素查询如图 2 所示。

给出两个布隆过滤器 β_1、β_2 相交的公式[13]

$$\overline{S}^{-1}(t_1,t_2,t_{\wedge})=\frac{\ln\left(m-\frac{t_{\wedge}\times m-t_1\times t_2}{m-t_1-t_2+t_{\wedge}}\right)-\ln(m)}{k\times\ln\left(1-\frac{1}{m}\right)} \tag{6}$$

其中，t_1 为 β_1 中被置 1 的字节数，t_2 为 β_2 中被置 1 的字节

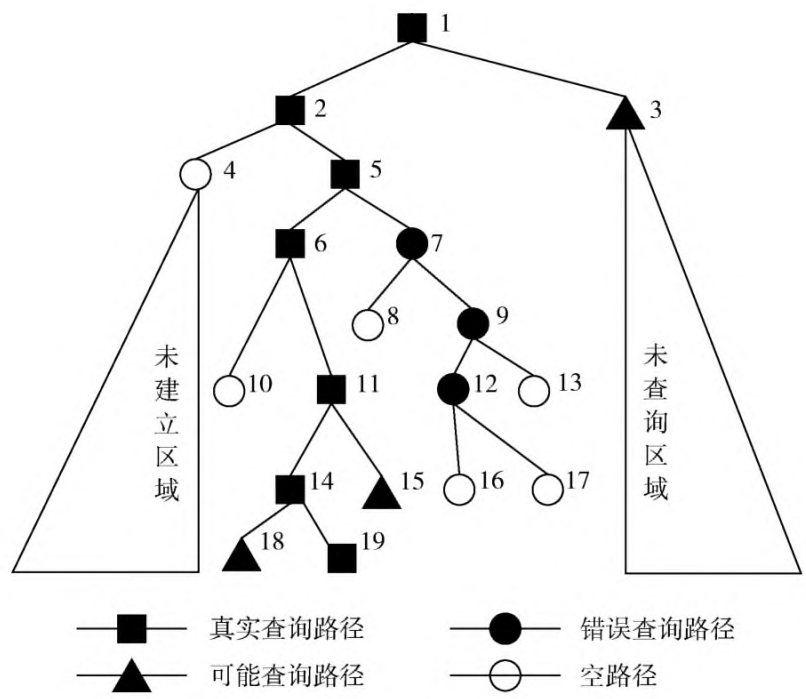

图 2　DBCT 中元素查询

数，m 为布隆过滤器的大小，k 为哈希函数的个数，$t_\wedge$ 为 β_1 和 β_2 做 AND 运算后被置 1 的字节数，当 $\overline{S}^{-1}=0$ 时，β_1 和 β_2 不相交；当 $\overline{S}^{-1}\neq 0$ 时，β_1 和 β_2 相交。DBCT 中元素的查询算法见表 3。

2.3　HBF_ADA 算法

与 SDP 和 BF_SDP 算法相比，HBF_ADA 算法的不同之处主要在于参与者数据包结构的变化，即将线性链表式存储结构改为 DBCT 结构用于存储节点信息。HBF_ADA 算法具体步骤如下：

（1）构建全局命名空间 N 和已使用命名空间 N′。

（2）构建本地参与者 DBCT 结构，并将各节点描述信息添加到 DBCT 中，包括主题名、主题类型和 QoS 策略等。

（3）向所有远程参与者发送本地参与者 DBCT 结构，同时接收远程参与者发送的 DBCT 结构。

（4）对收到的远程参与者 DBCT 结构进行解析，并在本地信息库中保存解析信息，进入 SEDP 阶段。

（5）在整个 SPDP 阶段，当本地或远程参与者信息发生变化时，将变化信息重新添加到 DBCT 中，转到步骤（3）继续执行。

3　仿真实验与分析

针对本文提出的 HBF_ADA 算法，仿真实验从实时性、内存消耗和元素删除 3 方面进行测试。为了便于实验验证和结果分析，本文引入两个概念：时延均值（average delay value，ADV）和节点匹配率（node match ratio，NMR）。

时延均值[14]是软件实时性的重要指标，在网络通信中，时延均值指的是数据从网络的一端发送到另一端所用的平均时间。在本文中，时延均值的具体定义如下：

定义 2　时延均值是指以本地参与者建立数据包开始，到远程参与者接收数据包且匹配成功终止，整个过程所用的平均时间，由式（7）计算得

$$ADV=\frac{1}{n}\sum_{i}^{n}T_i \tag{7}$$

式中：T_i 为本地参与者第 i 次发送数据所用时间，n 为发送次数。

由于 BF 存在误报率，即将不属于集合 S 的元素误认为属于 S，所以在新节点加入网络时被发现的概率并不会达到理论值 100%。针对此问题，本文提出节点匹配率（node match ratio，NMR）的概念，定义请参见文献 [15]：

仿真实验以分布式虚拟维修系统为背景，共设置 3 组实验，每组实验分布 10 个节点，每个节点上运行 5 个应用

表 3　DBCT 中元素的查询

算法 2：DBCT 中元素的查询算法

输入：DBCT 结构 T 和查询布隆过滤器 β_q
输出：查询已存在元素集合 S'
if T is leaf then
　　$S'\leftarrow\phi$;
　　for j：1…k do
　　$m\leftarrow\beta_j$;
　　$q\leftarrow\beta_q$;
　　if $\{m\}\bigcap\{q\}\neq\phi$
　　　　$S'\leftarrow(\{m\}\bigcap\{q\})$;
　　end
　　/ * 对于叶节点的查询，首先将节点中的所有元素取出，若取出的元素所构成的集合与 QBF 有交集，则相交元素在 DBCT 中 * /
else
　　$n_q=ComputeNum(\beta_q)$;
　　$n_l=ComputeNum(T.left)$;
　　$n_r=ComputeNum(T.right)$;
　　$n_{lq}=ComputeNum(\beta_q\bigcap T.left)$;
　　$n_{rq}=ComputeNum(\beta_q\bigcap T.right)$;
　　/ * 计算各节点所对应 BF 中被置 1 的字节数 * /

$$P_{left}=\frac{\ln\left(m-\frac{n_{lq}\times m-n_l\times n_q}{m-n_l-n_q+n_{lq}}\right)-\ln(m)}{k\times\ln\left(1-\frac{1}{m}\right)}$$

$$P_{right}=\frac{\ln\left(m-\frac{n_{rq}\times m-n_r\times n_q}{m-n_r-n_q+n_{rq}}\right)-\ln(m)}{k\times\ln\left(1-\frac{1}{m}\right)}$$

　　/ * 对于非叶节点的查询，首先判断 QBF 与左右两个子节点是否相交 * /
　　if($P_{left}=0$ and $P_{right}=0$) then
　　　　return Null;
　　　　/ * 若两侧节点与 QBF 均不相交，则返回 Null * /
　　else if($P_{left}=0$ and $P_{right}\neq 0$) then
　　　　return DBCT_Query($T.right,\beta$);
　　else if($P_{right}=0$ and $P_{left}\neq 0$) then
　　　　return DBCT_Query($T.left,\beta$);
　　　　/ * 若仅有一侧子节点与 QBF 相交，则沿该路径继续查询 * /
　　else
　　　　return DBCT_Query($T.right,\beta$);
　　　　return DBCT_Query($T.left,\beta$);
　　　　/ * 若两侧子节点与 QBF 均相交，则沿左右连个路径继续查询，直至最后一层叶节点 * /
end

程序。为保证测试结果的准确性，本文采用一对一的测试方式[16]，即新节点加入网络时，已存在的节点会向新节点连续发送 100 次数据。对所用时间求平均值，记录实验中发送的数据包大小，并尝试将已存在网络中的节点删除，观察是否成功。

分析结果如下：

(1) 图 3 为时延均值变化趋势。由图 3 可知，当系统层数小于 4 时，节点结构简单，SDP 算法中的线性链表式结构查询效率更高，因此 SDP 算法的 ADV 小于 BF_SDP 和 HBF_ADA 算法。当层数大于 4 时，3 种算法的 ADV 都会随着系统节点层数的增加而增大，但由于 HBF_ADA 算法采用 DBCT 分层数据结构，对多层节点的查询效率更高，因此 ADV 更小，且随着层数的不断增大，HBF_ADA 算法优势越明显。因此，HBF_ADA 算法具有更好的实时性。

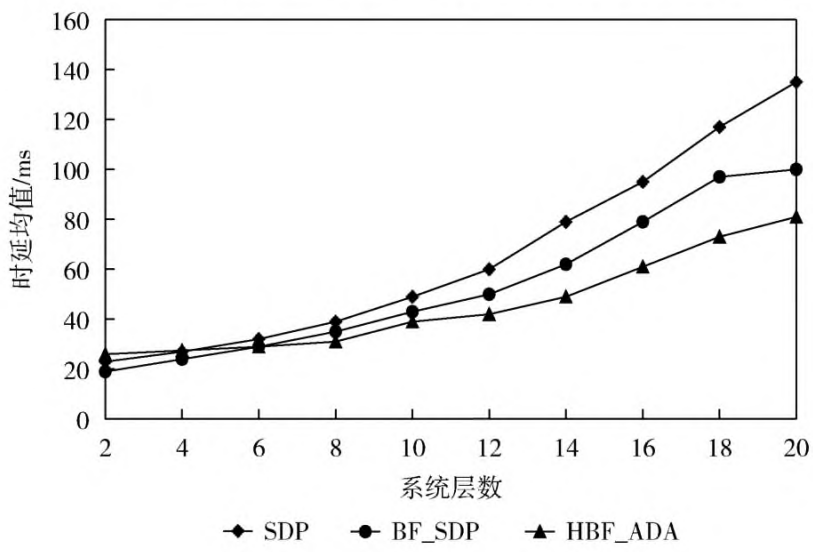

图 3 时延均值变化趋势

(2) 图 4 为节点负载变化趋势。由图 4 可知，在 SDP 算法的发现过程中，节点需保存所有远程节点发送的数据包信息，因此 SDP 算法中节点负载并不会随着 NMR 的增大而变化，且始终大于 HBF_ADA 和 BF_SDP 算法，因此 SDP 算法内存消耗最大。而 HBF_ADA 算法的 DBCT 为动态数据结构，能够根据节点的匹配情况动态存储远程节点数据包信息。因此，随着 NMR 的增加，HBF_ADA 算法节点负载明显小于 BF_SDP 算法，内存消耗更小。

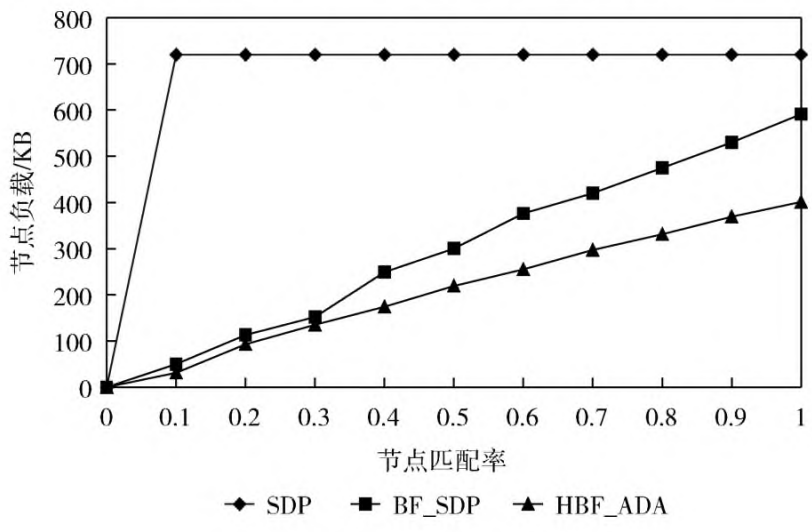

图 4 节点负载变化趋势

(3) 通过实验可以得出，HBF_ADA 和 SDP 能够将已存在网络中的节点删除，而 BF_SDP 无法删除已存在的节点。

4 结束语

目前，简单发现算法 SDP 中采用线性链表式数据结构存储和查询节点信息，但该结构查找多层节点时效率偏低且内存占用率过高。本文在 SDP 算法的基础上提出一种基于分层布隆过滤器的自动发现算法 HBF_ADA，该算法在不改变原有 SDP 发现过程的基础上，引入动态树形数据结构 DBCT。经实验验证，在相同条件下，与 SDP 和 BF_SDP 算法相比，该算法有更低的时延和内存消耗。但该算法在查询节点时存在误报的情况，所以下一步将要研究如何降低或消除 HBF_ADA 算法的误报率。

参考文献：

[1] Tu Yibin, Hao Jianguo, Zhou Xiaoyuan, et al. Research on improvement of KD-RTI communication mechanism base on DDS [C] //International Conference on Information Science and Control Engineering, 2016: 543-546.

[2] Object Management Group. The real-time publish-subscribe wire protocol DDS interoperability wire protocol specification version 2.1 [EB/OL]. [2017-09-05]. http://www.omg.org/spec/DDSI/2.1.

[3] Kyoungho An, Aniruddha G, Schmidt D, et al. Content-based filtering discovery protocol (CFDP): Scalable and efficient OMG DDS discovery protocol [C] //Proceedings of the ACM International Conference on Distributed Event-Based Systems. Mumbai: ACM, 2014: 130-141.

[4] ZHAI Haibo, ZHUANG Yi, HUO Ying. Publish/subscribe automatic discovery algorithm based on service ability vector [J]. Computer Engineering, 2014, 40 (9): 52-54 (in Chinese). [翟海波，庄毅，霍瑛. 基于服务力向量的发布/订阅自动发现算法 [J]. 计算机工程，2014，40 (9)：52-54.]

[5] Hua Xingbian, Zheng Xinxia. Applied-information technology in improving Bloom filter-based discovery protocol for DDS middleware in consideration of QoS [J]. Advanced Materials Research, 2014, 1044 (10): 457-460.

[6] Naror M, Yogev E. Bloom filters in adversarial environments [C] //Annual Cryptology Conference. Santa Barbara: Springer, 2015: 565-584.

[7] Khaefi MR, Kim D. Node discovery scheme of DDS using dynamic Bloom filters [C] //Proceedings of the IEEE Emerging Technology and Factory Automation. Barcelona: IEEE, 2014: 1-4.

[8] Adina C, Daniel L. Bloofi: Multidimensional Bloom filters [J]. Elsevier Information Systems, 2015, 54 (10): 311-324.

[9] Yoon M, Son J, Shin S. Bloom tree: A search tree based on Bloom filters for multiple-set membership testing [C] //IEEE

Conference on Computer Communications. Toronto：IEEE INFOCOM，2014：1429-1437.

[10] Qian Y，He Z，Zhang D. TIP：Time-efficient identification protocol for unknown RFID tags using Bloom filters [C] // IEEE International Conference on Parallel and Distributed Systems. IEEE，2016：151-158.

[11] Athanassoulis M，Ailamaki A. BF-tree：Approximate tree indexing [C] //Proceeding of the VLDB Endowment. Hawaii：IEEE INFOCOM，2014：1881-1892.

[12] Liu X，Qi H，Li K，et al. Sampling Bloom filter-based detection of unknown rfid tags [J]. IEEE Trans Comm，2015，14 (7)：1432-1442.

[13] Liu Xiulong，Qi Heng，Li Keqiu，et al. Sampling Bloom filter-based detection of unknown RFID tags [J]. IEEE Transactions on Communications，2015，4 (63)：1432-1442.

[14] Boutet A，Frey D，Guerraoui R，et al. Privacy-preserving distributed collaborative filtering [J]. Computing，2016，98 (8)：827-846.

[15] Neha Sengupta，Amitabha Bbagchi，Srikanta Bedathur，et al. Sampling and reconstruction using Bloom filters [J]. IEEE Transactions on Knowledge & Data Engineering，2017，7 (30)：1327-1330.

[16] Chul-Hwan Kim，Gunjae Yoon，Wonjoon Lee，et al. A performance simulator for DDS networks [C] //International Conference on Information Networking. Cambodia：IEEE，2015：122-126.