Bloom Filter 及其应用综述*)

肖明忠　代亚非
(北京大学计算机科学技术系　北京100871)

摘　要　Bloom Filter 对数据集合采用一个位串表示并能有效支持集合元素的哈希查找操作。本文对 Bloom Filter 及其改进型进行了综述性分析研究,探讨了它的实用性。较为详细地阐述了它在 P2P 网络文件存储系统 OceanStore 和文本检索系统中的应用情况。最后指出了进一步的研究方向。

关键词　Bloom Filter,哈希查找

A Survey on Bloom Filters and its Applications

XIAO Ming-Zhong　DAI Ya-Fei
(Department of Computer Science & Technology, Peking University, Beijing 100871)

Abstract　Representation and location of information play a key role in many applications and the two processes are very related. Bloom Filter uses a bit strings to represent a data set and is able to locate an element in the set by the way of hash functions. This paper surveys all kinds of Bloom Filter, discusses their practicability and describes in detail their application on OceanStore system and text-retrieval system. Finally, some advices about future works are given.

Keywords　Bloom filters, Hashing locate

表示和查找信息是大多数计算机应用程序的核心,这两个过程是密切相关的。比如,采用顺序表表示信息,则可以使用折半查找;采用哈希表组织信息,则可以使用哈希查找。表示就是以计算机能处理的方式,把信息组织成为可计算的实体,查找就是确定一个具有特定值的元素是不是一个特定集合的成员[18]。

Bloom Filter 对数据集合采用一个位串表示并能有效支持集合元素的哈希查找操作[1~3]。由于其表示算法的随机特性,存在某元素不属于数据集合而被指称属于该数据集合的可能性[1,3],其大小记为误称率。只要这种可能性足够地小以致应用能容忍这种误差,由于其哈希查找的常数时间和少量的存储空间开销,使得它具有非常的实用价值。

Bloom Filter 自1970年提出以来[1],被广泛应用到各种计算机系统[1,3~6,15,16]之中表达庞大数据集及提高查找效率。近30多年来对 Bloom Filter 实质性的改进,我们认为仅有压缩型和计数型两种。最早提出的 Bloom Filter 没有考虑分布式网络环境下的应用特征,压缩型 Bloom Filter 引入算术编码技术,主要考虑如何减少网络节点间交互的信息量,同时又保持甚至获得更好的误称率。这两种 Bloom Filter 没有考虑集合元素的减少问题,计数型 Bloom Filter 对这一问题给予了考虑,并认为对大多数实际应用而言4位的计数器大小是足够的。

本文首先给出了1970年提出的 Bloom Filter 算法和数据结构的描述,然后进行误称率的分析。接下来分别就压缩型 Bloom Filter(2001年提出)和计数型 Bloom Filter(2000年提出)进行了描述与分析。最后,以 P2P 网络文件存储系统 OceanStore 和文本检索系统为例,阐述了它的应用情况。我们的目的在于综述 Bloom Filter 的各种细节问题以及实际应用中需要权衡的一些事情,希望对实际应用工作有所裨益。

1　Bloom Filter[1,2,6]

Bloom Filter 使用长为 m 的位串 V 表达数据集合 $A=\{a_1,a_2,\cdots,a_n\}$,设有 k 个具有均匀分布特性的哈希函数 h_i,且 $\forall x\in A, h_i(x)\in\{1,2,\cdots,m\}$,则:

①集合表示算法

```
BitString BF_reprent(Set A,int m){
  //对数据集合 A,采用长为 m 的位串 V 表示
  BitString V[1..m]=0;
  for(j=1;j<n+1;j++)
     for(i=1;i<k+1;i++) V[h_i(A[j])]=1;
  return V;
}
```

② 查找算法

```
Boolean BF_lookup(BitString V,ele){
    //返回0,if ele∉A;返回1,if ele∈A
    int i=1;
       while ((V[h_i++(ele)]==1) && (i<k+1));
       if (i<=k) return 0; else return 1;
}
```

③ 已知两集合的 Bloom Filter 表示,则其集合并的 Bloom Filter 表示算法

```
BitString BF_union(V1,V2){
    return V=(V1 | V2); //两个向量的按位或
}
```

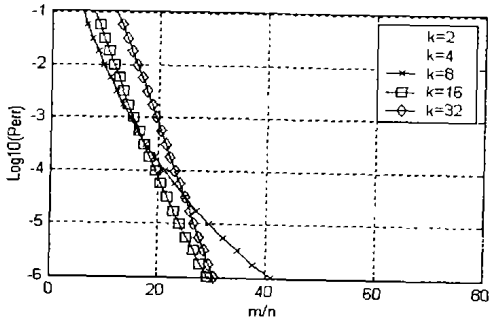
④ 集合元素增加,则 V 的更新算法

```
BitString BF_add(V,ele){
    for(int i=1;i<k+1;i++) V[h_i(ele)]=1;
    return V;
}
```

分析上述算法,不难发现:①假定集合元素是 URL 地址且每个地址约50个 ASCII 码字符,则直接存储这个集合需要400(n+1)个比特的内存空间[1],若采用 Bloom Filter 仅需 m

*)基金项目:国家重点基础研究发展规划973资助项目(G1999032706);国家863高科技发展计划资助项目(2001AA111013)。肖明忠　博士生,主要研究领域为网络与分布式系统。

个 bit 的存储空间(实际中,m≪400n);②查找算法时间复杂度为 $O(k)$。③实现两个集合并的 Bloom Filter 向量表示,时间复杂度为 $O(m)$。④增加集合元素时间复杂度 $O(k)$。值得一提的是,在实际系统中,k 和 m 均为常数。⑤根据表示算法,某位可能被多次置为1,但仅第一次设置真正起作用。节约空间的同时,引入了这个问题。由于这个问题,也导致了某个元素不属于集合而被指称属于的问题。这个误称概率的大小将在下节讨论,它需要我们在应用时,对一些算法参数作出权衡。



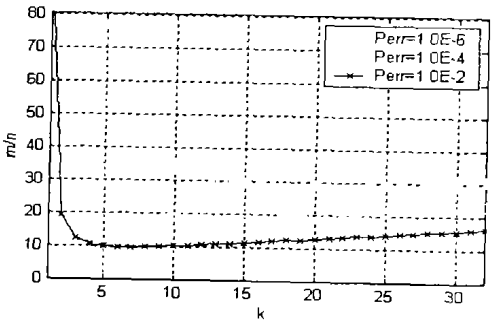


图1 左图反映 p_{err} 随 m/n 变化的曲线图,右图反映 k 与 m/n 间关系

2 Bloom Filter 查找误差分析[1~3,6]

如前所述,一个位被设置为1若干次,但仅第一次设置真正起作用。若 $ele\in A$,则 BF_lookup 一定返回1。问题是 BF_lookup 存在:$ele\notin A$ 却返回1的可能性。下面分析这一误称概率的大小。

在表示算法中,每次赋值使得某位为1的概率为:1/m,为0的概率为:1−1/m。语句 V[h_i(A[j])]=1共被执行 kn 次,所以算法结束时,某位仍为0的概率是:

$$p_0=(1-\frac{1}{m})^{kn}\approx e^{-kn/m} \tag{1}$$

从而,误称 $ele\in A$ 的概率就是:$\forall i\in\{1,2,\cdots,k\}$,均有 $h_i(ele)=1$的概率。即误称概率为:

$$p_{err}=(1-p_0)^k(1-(1-\frac{1}{m})^{kn})^k\approx(1-e^{-kn/m})^k$$
$$=e^{k\ln(1-e^{-kn/m})} \tag{2}$$

给定 m,n 的情况下,求 p_{err} 的最小值,即是求 $g=k\ln(1-e^{-kn/m})$的最小值。

$$\frac{dg}{dk}=\ln(1-e^{-kn/m})+\frac{kn}{m}\frac{e^{-kn/m}}{1-e^{-kn/m}} \tag{3}$$

显然,k=(m/n)ln2时,上式为0,即 g 和 p_{err} 均取最小值。再将(1)式代入 g,有:

$$g=-\frac{m}{n}\ln(p_0)\ln(1-p_0) \tag{4}$$

同样,g 关于 p_0 求导,当 $p_0=1/2$时,g 有最小值,从而 p_{err} 有最小值。所以:

$$p_{err\_min}=(1-\frac{1}{2})^k=(\frac{1}{2})^{m/n\ln2}=0.6185^{m/n} \tag{5}$$

然而实际应用中,通常 k 较小且固定,需要在 p_{err} 和 m 之间作权衡。图1左反映了 k 值固定情况下,如何在两者间权衡,同时也可以看出:k=8,16,32时,曲线比较接近,使用8个哈希函数并不比使用32个哈希函数差多少,但却可以节约近3/4的哈希计算时间。从右图中,可以看出:同样的 P_{err} 值,并非 k 越大越好;相同 k 值情况下,要获得较好的误称率,需要较大的 m。

从图1右中可以看出,固定 m/n,增大 k,可以获得更好的误称率。下一节将看到:若 Bloom Filter 用于网络节点间交换信息,固定 m/n,减少 k,还可以获得约70%的压缩率,大大减少了节点间的网络流量,这是以增大 P_{err} 为代价的。也可采用增大 m,减小 k、P_{err} 的方法获得大幅度的压缩率。

3 压缩型 Bloom Filter[2]

分布式系统要求尽可能减少节点间交互次数及每次交换的信息量[10],减少交换信息量的方法之一就是使用压缩技术。设可能传输的符号集合为字母表 Σ。符号 s 的编码位数,称为信息量(information content),记为 I(s),$I(s)=-\log_2^{pr(s)}$,pr(s)是符号 s 的出现概率。整个字母表中的每个符号的平均信息量称为熵(entropy),记为 H,则有:

$$H=\sum_s pr(s)\cdot I(s)=\sum_s -\mathrm{pr}(s)\cdot\log_2^{pr(s)} \tag{6}$$

已经表明,不可能有比熵值更好的平均编码长度。这就是香农编码原理[7]。

前述已知 $p_0=1/2$,即 k=(m/n)ln2时,P_{err} 有最小值。若选择最优 k 值,位串 V 每一位为0的概率为1/2,那么0和1在 V 中出现的概率均为1/2,从而有 H=1。所以,选择最优 k 值,V 将不可能获得任何压缩的效果。一个自然的想法就是改变 p_0,影响字母表 Σ={0,1}中,0和1出现的概率,达到压缩的目的,记 $H(p_0)=-p_0\log_2^{p_0}-(1-p_0)\log_2^{1-p_0}$。在这个极限熵值的情况下,对长为 m 的位串 V 进行压缩,则需要 $mH(p_0)$个比特位,记为 z。若 z=m,即不压缩,则 $p_{err\_min}=0.6185^{m/n}=0.6185^{z/n}$。根据(1)式,有:$k=-(m/n)(\ln p_0)$。再根据(2)式,所以:

$$p_{err}=(1-p_0)^k=(1-p_0)^{-\frac{m}{n}(\ln p_0)}=(1-p_0)^{-\frac{(\ln p_0)}{H(p_0)}(\frac{z}{n})}=$$
$$\exp(\frac{-(\ln p_0)\ln(1-p_0)}{(-\log_2^e)(p_0(\ln p_0)+(1-p_0)\ln(1-p_0))}\frac{z}{n}) \tag{7}$$

固定 z,n(z ≥ n),最小化 p_{err} 即最小化,$\exp(\frac{(\ln p_0)\ln(1-p_0)}{p_0(\ln p_0)+(1-p_0)\ln(1-p_0)})$,又即最大化下式:

$$\gamma=\frac{p_0}{\ln(1-p_0)}+\frac{1-p_0}{\ln p_0} \tag{8}$$

通过计算:

$$\frac{d\gamma}{dp_0}=\frac{1}{\ln(1-p_0)}-\frac{1}{\ln p_0}+\frac{p_0}{(1-p_0)\ln^2(1-p_0)}-\frac{1-p_0}{p_0\ln^2 p_0}$$
$$=0 \tag{9}$$

发现,当 $p_0=1/2$时,r 取最小值。以1/2为中值,p_0向两侧分别对称趋于0或1时,r 对称地趋向于越来越大。所以,求(8)式最大值,即是:

$$\lim_{p_0\to0}\gamma=-1(\because\lim_{p_0\to0}\ln(1-p_0)=-p_0) \tag{10}$$

将其代入(7)式有:

$$p_{err\_min}=e^{\left(\frac{1}{-\log_2^e}\right)\frac{z}{n}}=e^{-\frac{z}{n}(\frac{\ln2}{\ln2\cdot\log_2^e})}=e^{\frac{z}{n}\ln\frac{1}{2}}=0.5^{z/n}<0.6185^{z/n}$$

(11)

这表明，原来的 Bloom Filter 的最小误称率是可以改善的。根据(5)式和(11)式有：$(0.5)^{z/n}=(0.5)^{(m/n)\ln 2}$，从而 $z=m\ln 2$，将获得约70%的压缩率。

在前述的计算中，$p_0\to 0$和 $p_0\to 1$分别对应使用无限多个和0个哈希函数。实际中，最小哈希函数个数最小为1，而且出于节约哈希计算量的目的，显然我们希望$1\leqslant k\leqslant (m/n)\ln 2$。为了获得高效的压缩率，我们需要在 k、m 和 P_{err}间作权衡。如图2例所示，A 点和 B 点所在曲线分别绘出了 Bloom Filter 和压缩型 Bloom Filter 误称率与 k 的关系。从 A 点所在曲线可以看出：若选择 k=2，较最优点 k=6将获得约75%的压缩率，但 P_{err}从0.0216增大到约0.05。从 B 点所在曲线看，增大本地节点表示数据集合的位数，可以获得比原 Bloom Filter 的最小误称率还小的误称率(原来的最优点 C 变成了最坏点)，且可以使用更小的 k 值。如 B 点所示，本地节点使用14n 个比特位代表数据集合，则传输时，仅需压缩传输约7.923n 个比特位，压缩率为57%。同时获得0.0177的误称率，且仅使用了两个哈希函数。

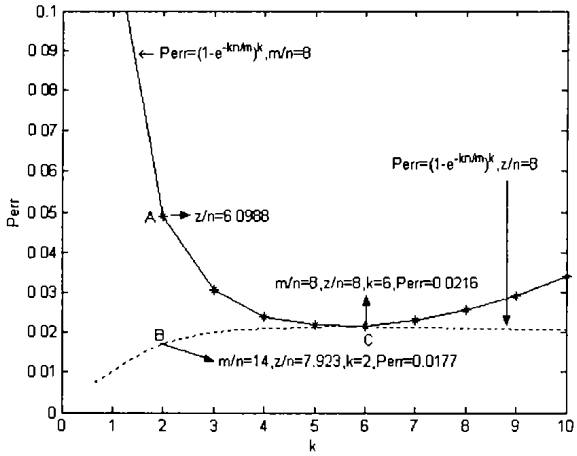


图2　压缩型 Bloom Filter 与 Bloom Filter 性能对比

总之，在分布式应用环境下，按(m/n)ln2选择最优 k 值不会获得任何的传输压缩率。相反，若增大本地节点表示信息的位数和选择较小的 k 值，不仅可以获得高效的传输压缩率同时还能获得更小的误称率。

4　计数型 Bloom Filter[3]

前述的 Bloom Filter 算法存在这么一个问题，即：没有考虑集合元素的减少问题。已知 $A,V,\exists\ d\in A$，如果 d 不再属于 A，则如何修改 V 向量？根据 A-d 重算向量 V，在某些应用场合是不可能的，自然可以考虑为向量 V 的每一维 $i(i\in\{1,2,\cdots,m\})$设置一个计数器，记为 $c(i)$，初值为0。当要增加集合元素 d 时，则 $c(h_j(d))=c(h_j(d))+1,(j=1\cdots k)$；集合元素删除时，$c(h_j(d))=c(h_j(d))-1,(j=1\cdots k)$。问题是计数器该设置多大，以致不会溢出？假设任一计数器 c 大于等于 i 的概率为 $\Pr(value(c)\geqslant i)$，则：

$$\Pr(value(c)\geqslant i)=m\binom{nk}{i}\frac{1}{m^{nk}}\leqslant m\binom{nk}{i}\frac{1}{m^{i}}=m\frac{nk(nk-1)\cdots(nk-i+1)}{i!m^{i}}\leqslant m\frac{(nk)^{i}}{i!m^{i}}\approx m\frac{(nk)^{i}}{\sqrt{2\pi i}(\frac{i}{e})^{i}m^{i}}\leqslant m(\frac{enk}{im})^{i}\frac{1}{\sqrt{2\pi}}\quad(\text{原文中无}2\pi)\tag{12}$$

$$\because 1\leqslant k\leqslant(m/n)\ln 2\quad\therefore\Pr(value(c)\geqslant i)\leqslant\frac{m}{\sqrt{2\pi}}(\frac{e\ln 2}{i})^{i}$$

当 $i=16$，则 $\Pr(value(c)\geqslant 16)\leqslant 0.55\times 10^{-15}\times m$。可见，设置4位计数器已经拥有了非常小的溢出概率。实践中，我们可以根据集合元素增加与删除的统计规律，选择适当的计数器大小。那种不断增加元素，使得计数器溢出，然后又不断减少元素，使得某位不该为0却为0的这种事件序列概率是非常小的[3]。Li Fan[3]认为4位计数器对于大多数应用来说，是足够的。

5　应用举例

Bloom Filter 已经应用到许多系统中，如：①协同 Web Cache 共享[3]使用计数型 Bloom Filter 表述网络节点缓存的 URL 数据集，节点间相互交换 V 向量，从而节点最后拥有所有的 V 向量，即拥有全局 URL 数据映像。任何对特定 URL 的访问，都会被指向最有可能的节点。节点间交互，可以使用广播协议、Gossip 协议[10]以及文[9]提到的算法；②当一个主文件由于性能或其他原因，大量更新的数据不能实时反映到主文件上，而又需要考虑数据的完整性时，可以考虑对更新的数据集合采用 Bloom Filters 表示(直接对更新数据集合进行访问，显然代价过高)。对主文件的访问，首先转向查找该向量，若向量相应的位均被置1，则访问更新数据集合，否则，转向主文件。根据前述算法的分析，数据的完整性显然能保证。花费一些内存空间和哈希函数计算为代价，实现大量更新数据的高速查找[1,6]，等等。限于篇幅，将仅以 P2P 网络文件存储系统 OceanStore 和文本检索系统为例，阐述它的实际应用情况。

5.1　在 OceanStore 中的应用

OceanStore 是一个广域的 P2P 网络文件存储系统[4]，它的对象定位过程分为两个阶段。第一阶段使用了 Bloom Filter 算法，每个 P_{err}节点拥有距其 $i(i=0..d)$个 hops 数的节点数据集合的 Bloom Filter 向量的信息，从而当某节点处理某个定位对象的请求时，它能够选择最佳的搜索方向，较 Gnutella[13]那样的 flooding 对象定位思路减少了不少的网络流量。在半径为 d 的范围内，若对象定位失败，将进入第二阶段的查找过程。在这一阶段，将使用 Tapstry[8]路由策略进行对象搜索。

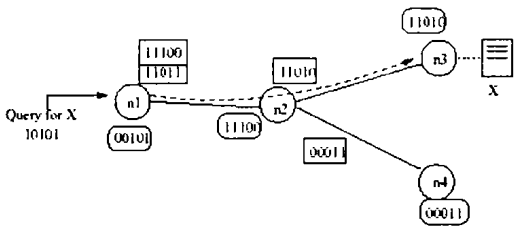


图3　OceanStore 中 Bloom Filter 的使用(引自文[12])

这里仅描述第一阶段的对象定位算法。每个节点拥有自己、直接邻居节点的 Bloom Filter 向量以及沿各直接邻居方向距其 j(j=2..d)个 hop 数的所有节点的 Bloom Filter 向量和。如图3所示，设有对象空间 objects={v|v[i]∈{0,1},i=1..5},k=3,m=n=3,d=2。节点 n1拥有的 V 向量有：V_0={00101}，V_1={11100}，V_2={11011=BF_union(11010,00011)}。当 n1收到对 x=10101∈*Objects* 的定位请求，假定有计算 $h_1(x)=1,h_2(x)=2,h_3(x)=4$，n1遍历 V_0、V_1和 V_2发现 $V_2(j),j=1,2,4$全为1，知沿 n2方向可以发现 x。于是，转发请求给 n2，n2执行同样操作，发现应该转发给 n3。n3最后响应查

找成功。为避免大的 hop 数搜索，请求包设有包生存期域(TTL，初始 TTL=d)，请求没被转发，则 TTL=TTL-1，若 TTL=0，丢弃请求且告知原请求者，以便引发第二阶段的对象定位过程。图中矩形框内为附近节点 V 向量(和)信息，圆形框内为本地数据集合的 Bloom Filter 向量。如何选择生成 Bloom Filter 向量的参数是这类系统的关键，若误称率过大，有退化为 flooding 算法的可能性。

5.2 在文本检索系统中的应用

自由文本检索[5,11]系统，将出现在一个文本中的单词集合，表述为一个 Bloom Filter 向量，不需要任何索引技术的支持就可以实现任意单词的组合查询。

假设系统文档集合为 documents 且每个文档是若干不同单词的集合，则根据 Bloom Filter 向量表示算法，容易得到每个文档的 Bloom Filter 向量 V。由所有 V 向量，构成一个位矩阵，矩阵大小|documents|×m。查询时，从请求中提取单词 x，并计算 $h_j(x)$，$(j=1\cdots k)$。若矩阵中第 $h_j(x)$，$(j=1\cdots k)$列均同时为1，则相应的行向量对应的文档带着$1-P_{err}$的概率即为所需。

制约这一思想广泛应用源于庞大的计算量，但是目前的索引技术约需与被索引文档集相当的存储空间且倒排文件的计算量也是不容乐观的[7]。所以，这一思想仍具深入研究的必要与实际应用的可能。Stanfill 和 Kahle[11]在并行计算机上实现了这一想法。限于当时计算机的处理能力，使用了65536个处理器且每个处理器仅仅承担约2个 Bloom Filter 的检索请求。

结束语 本文对 Bloom Filter 及其改进型进行了综述性分析研究，它们适用于对大规模数据集合表示与查找的应用背景。在应用允许的误差范围内，它们能有效节省集合元素表示的空间占用和获得常数级的查找开销。文中对如何调节各种算法参数进行了详细的讨论。在分布式应用环境下，按(m/n)ln2选择最优 k 值不会获得任何的传输压缩率。相反，若增大本地节点表示信息的位数和选择较小的 k 值，不仅可以获得高效的传输压缩率同时还能获得更小的误称率。计数型 Bloom Filter 主要适用于需要对集合元素进行删除的应用场合。分析中对均匀哈希函数的假定是合理的，存在 MD5、SHA-1[17,19]以及文[5]提到的一类哈希函数具有这样的特点。Bloom Filter 的应用不只限于本文所提到的例子，如：在蜂窝系统中的应用[14]、其他 P2P 系统[15,16]中的应用等等。我们的目的在于展示 Bloom Filter 的各种细节问题以及实际应用中需要权衡的一些事情，希望对实际应用工作有益，能否发展更好的替代品以及增强对模糊集合和多重集合的处理有待进一步研究。

参考文献

1 Bloom B. Space/time tradeoffs in hash coding with allowable errors. Communications of the ACM, 1970, 13(7): 422～426

2 Mitzenmacher M. Compressed Bloom Filters. In: Proc. of the 20th ACM Symposium on Principles of Distributed Computing (PODC2001), Aug. 2001

3 Fan L, Cao P, Almeida J, Broder A. Summary cache: a scalable wide-area web cache sharing protocol. IEEE/ACM transactions on networking, 2000, 8(3)

4 Kubiatowicz J, et al. OceanStore: An architecture for globe-scale persistent storage. In: Proc. of the 9th Intl. conf. on architectural support for programming languages and operating systems (ASPLOS 2000), 2000

5 Ramakrishna M V. Practical performance of Bloom Filters and parallel free-text searching. Communications of the ACM, 1989, 32(10): 1237～1239

6 Mulllin J K. A second look at Bloom Filters. Communiations of the ACM, 1983, 26(8): 570～571

7 Witten I H, Moffat A, Bell T. Managing Gigabytes (2nd Edition). Morgan Kaufmann, San Francisco, 1999

8 Zhao B Y, Kubiatowicz J, Joseph A D. Tapstry: An infrastructure for fault-tolerant wide-area location and routing. Computer Science Division University of California, (UCB/CSD-01-1141), April 2001

9 Balter M H, Leighton T, Lewin D. Resource discovery in distributed networks. In: Proc. of the 18th annual ACM symposium on priciples of distributed computing (PODC'99), 1999

10 Coulouris G, Dollimore J, et al. Distributed systems concepts and design (3 Edition), Addison Wesley, 2001

11 Stanfill C, Kahle B. Parallel free-text search on the connection machine system. Communication of the ACM, 1986, 29(12)

12 Rinaldi R, Waldvogel M. Routing and data location in overlay p2p networks: [IBM Research Report RZ3433]. July 2002

13 The Gnutella protocol specification v0.4. http://www.clip2.com

14 Yuen W H A, et al. A hybrid Bloom Filter location update algorithm for wireless cellular systems. In: IEEE Intl. Conf. on Communications. ICC(3) 1997. 1281～1286

15 Byers J, et al. Informed content delivery across adaptive overlay networks, ACM SIGCOMM'02, Aug. 2002. 19～23

16 Ledlie J, Taylor J M, Serban L, Seltzer M I. Self-Organization in Peer-to-peer Systems. 2002 SIGOps European Workshop, Bordeaux, France, Sep. 2002

17 The Secure Hash Algorithm Directory MD5, SHA-1 and HMAC Resources. http://www.secure-hash-algorithm-md5-sha-1.co.uk/

18 Shaffer C A. Data structures and algorithm analysis. Prentice Hall, 1997

19 NIST, FIPS PUB 180-1: Secure hash standard. http://cs-www.ncsl.nist.gov/cryptval/shs.html/, April 1995

(上接第175页)

2 邵维忠，梅宏. 统一建模语义 UML 述评. 计算机研究与发展，1999，36(4)：385～394

3 Bruel J-M, France R B. Transforming UML Models to Formal Specifications. In: Proc. of the Int. Conf. on OOPSLA'98 Vancouver, Canada, Oct. 1998

4 Breu R, Hinkel U, et al. Towards a Formalization of the Unified Modeling Language. In: Aksit M, Matsuoka S, eds. Proc. of ECOOP'97-Object Oriented Programming, 11th European Conf. Jyvaskyla, Finland, June Springer-Verlag, LNCS 1241, 1997

5 Araujo J. Formalizing sequence diagrams. In: Andrade, L., Moreira, A., Deshpande, A., eds. Proc. of the OOPSLA'98 Workshop on Formalizing UML. 1998

6 Geisler R. Precise UML semantics through formal metamodeling. In: Andrade, L., Moreira, A., Deshpande, A., eds. Proc. of the OOPSLA'98 Workshop on Formalizing UML. 1998

7 Baeten J C M, Weijland W P. Process Algebra. Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, Cambridge, 1990

8 Bergstra J A, Klop J W. Process algebra for synchronous communication. Information& Control, 1984, 60: 109～137

9 Mauw S, Reniers M A. An algebraic semantics of Basic Message Sequence Charts. The Computer Jounal, 1994, 37(4): 269～277

10 Plotkin G D. An operational semantics for CSP. In: Proc. of the Conf. on the Formal Description of Programming Concepts, volume 2, Garmisch, 1983