第36卷 第15期 计 算 机 工 程 2010年8月
Vol.36 No.15 Computer Engineering August 2010
·博士论文· 文章编号：1000—3428(2010)15—0032—04 文献标识码：A 中图分类号：TP301

一种高效的DNS日志压缩算法

王艳峰[1,2,3]，王 正[1,2,3]，阎保平[2,3]
(1. 中国互联网络信息中心，北京 100190；2. 中国科学院计算机网络信息中心，北京 100190；3. 中国科学院研究生院，北京 100049)

摘 要：CN顶级域名的DNS日志从分布式站点传输到数据处理中心时，对海量数据存储和传输带宽提出极大挑战。针对该问题，提出一种高效的DNS日志压缩算法，利用DNS查询类型的冗余性和DNS查询时间、IP地址和域名等的重复性进行DNS日志压缩。实验结果证明了DNS日志压缩算法在DNS实时监控和分析系统中部署的有效性和高效性。

关键词：域名系统；日志压缩；时间空间本地性

High-efficient DNS Log Compression Algorithm

WANG Yan-feng[1,2,3], WANG Zheng[1,2,3], YAN Bao-ping[2,3]
(1. China Internet Network Information Center, Beijing 100190; 2. Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190; 3. Graduate University of Chinese Academy of Sciences, Beijing 100049)

【Abstract】In the deployment of real time Domain Name System(DNS) monitoring and analysis system, the large volume CN top level domain DNS log heavily burdens its transmission from the distributed sites to the data processing center. The redundancy of DNS query type and the repetitiveness of DNS query time, IP and domain name are utilized for the DNS log compression to facilitate its transmission. For IP and domain name compression, the algorithm relies on their concentration property as well as their time and space locality.

【Key words】Domain Name System(DNS); log compression; time and space locality

1 概述

域名系统(Domain Name System, DNS)主要完成域名到IP地址的解析。国家顶级域名是指以国家或地区代码为结尾的域名[1]，中国互联网络信息中心(CNNIC)负责管理中国.CN国家域名系统的注册、解析和运行。

BIND作为目前最流行的DNS服务器软件[2]，其日志记录了详细的DNS查询请求信息，其格式如下：

03-Mar-2009 00:00:01.797 queries: info: client 80.12.255.138#58347: query: yourcatfree.cn IN A -E

其中，“03-Mar-2009 00:00:01.797”为查询请求到达的时间；“80.12.255.138”为终端(一般为递归服务器)IP 地址；“yourcatfree.cn”为查询的域名；“IN”为资源类别(class)；“A”为资源类型(type)；“-E”为DNS查询包某些字段的信息；“-”表示不请求递归；“E”表示支持EDNS0[3-4]。

目前CN国家顶级域名服务器在全球共部署10个分节点，分节点服务器所生成的DNS查询日志会定期上传到中心节点，由中心节点负责进一步的分析和处理。为了保证中心节点能够准确实时地展示日志中所蕴含的域名请求状况，分节点的日志数据需要及时传输到中心节点，但目前各分节点每天日志数据量最高达到60 GB以上，其1 h数据量的峰值更达到约5 GB，而且传输网络的带宽是有限且不可靠的。本文提出一种高效的DNS日志压缩算法，可以有效缓解海量数据传输和传输带宽之间的矛盾。

2 DNS日志压缩算法

从中心节点处理日志数据的实时监控的目标出发，原始日志的有用字段可以归结为：时间，IP地址，域名和类型，其他信息可以直接省略。如图1所示，在压缩端，对这4个字段分别进行压缩，并把压缩后的字段串联起来构成压缩后的日志。

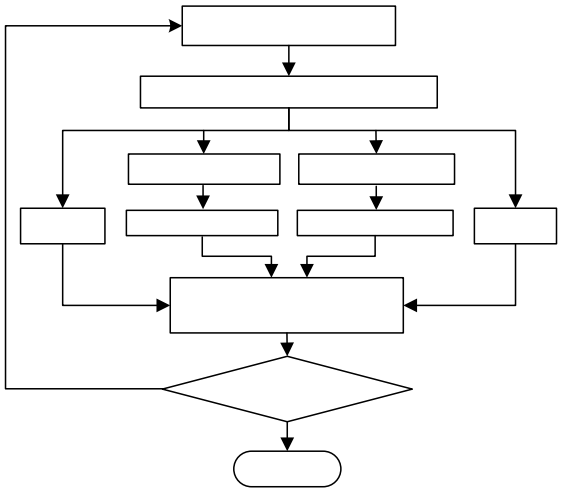

图1 DNS日志压缩流程

如图2所示，在解压端对这4个字段分别进行解压，并把解压后的字段串联起来构成解压后的日志。下面分别介绍

基金项目：国家发改委CNGI基金资助项目(CNGI-09-03-04)；中国科学院知识创新基金资助项目(0814081105)；中国科学院计算机网络信息中心青年基金资助项目(CNIC_QN_08006)

作者简介：王艳峰(1973－)，男，高级工程师、博士研究生，主研方向：资源定位与寻址技术，下一代互联网技术；王 正，博士研究生；阎保平，研究员、博士、博士生导师

收稿日期：2010-03-15 **E-mail：**wyf@cnnic.cn

各个字段的压缩和解压方法。

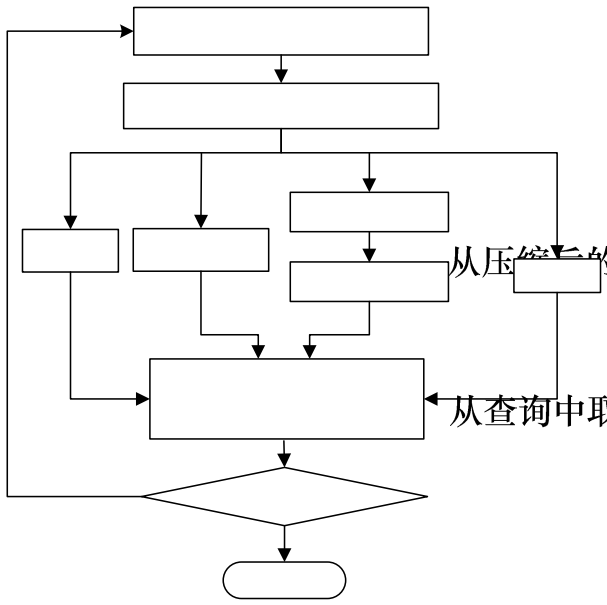

图 2　DNS 日志解压流程

2.1　时间压缩

各个分节点的平均查询量从每秒几百到六十多次不等，从日志的格式可知，在同一秒内，查询的时间字段的日期、小时、分钟和秒的整数部分都是相同的，不同之处仅是秒的小数部分。因此，仅保留秒的小数部分即可。又由于一秒内查询的秒的小数部分都是保持一种递增的关系，直至到下一秒的查询时，这种递增关系才被打破。实际上，只要查询量不至于太小，这种规律就一直存在。因此，可以根据这种简单的递增关系确定秒的小数部分之外的时间。定理给出了这种递增关系不再有效的条件，实际上，以目前 DNS 各分节点的查询量，一秒内没有查询的情况绝不可能发生。在压缩端，仅保留日志文件第一个查询全部时间(确定时间的起点)，以及其他查询的秒的小数部分即可。在解压端，通过递增关系的判断来确定日期、小时、分钟和秒。解压算法的流程如图 3 所示。

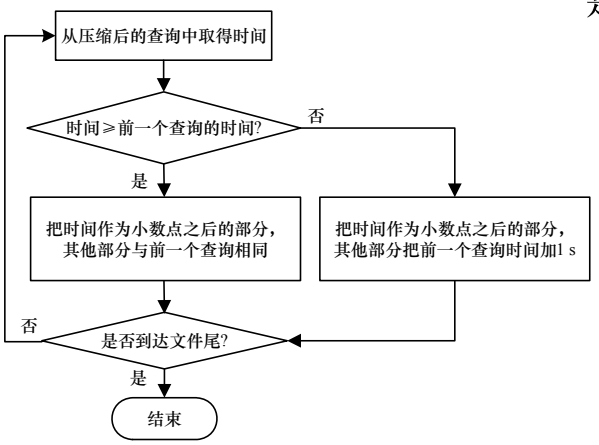

图 3　时间解压流程

定理　当且仅当时间上相邻的 2 个查询的间隔时间大于或等于 1 s 时，DNS 日志中秒的小数部分的递增关系才会打破。

2.2　类型压缩

DNS 区域文件包含描述区的资源记录，资源记录的语法为[3]:

{name} {TTL} addr-class record-type record-specific-data

其中，record-type 表示资源记录的类型，常用的有 A、MX、AAAA、PTR、CNAME 等。尽管目前定义的资源记录类型有几十种，但对各种资源记录类型的查询量是极不均衡的。由 CN 顶级域名服务器 DNS 查询类型分布，统计得到排名前 12 的类型占了总查询量的 99.99%以上，因此，只需压缩这些类型即可。除了 A 类型以外，其他 11 种类型的字符都文件中读取这些类型映射到一个各不相同的长度为 1 的字符，就达到了压缩的目的。在解压端，只需作反向映射即可。

2.3　IP地址和域名压缩算法

以下重点介绍 IP 地址和域名的压缩算法。本质上，IP 地址和域名的压缩算法充分利用了 IP 地址和域名的查询的重复性，而 CN 查询的重复性又有其自身的特点，这些特点为本文算法的设计提供了依据。

根据对 CN 顶级域名服务器日志的统计分析，发现 IP 地址和域名的查询具有以下两方面的特征:

(1)IP 地址和域名的查询量极不均衡，排名靠前的少量 IP 地址和域名访问量极高，而绝大部分 IP 地址和域名的访问量却很少。

图 4 是某节点日志中出现的全部域名的访问量的分布曲线。其中，横坐标为域名查询量排名的对数值，纵坐标为相应被查询频率的对数值，由两者之间的这种近似线性关系可知两者遵从幂律分布:

$$P(r \mid \alpha) \propto r^{-\alpha} \tag{1}$$

其中，$r = 1{:}\ S$ 为 *Rank* 值，S 为域名总数；α 为该分布的参数。由图 4 可见，域名空间的 0.1%的域名占据了大约 98%的访问量，这体现出域名域名查询量极高的集中特性。而 IP 地址的访问量同样呈现出极高的集中性。

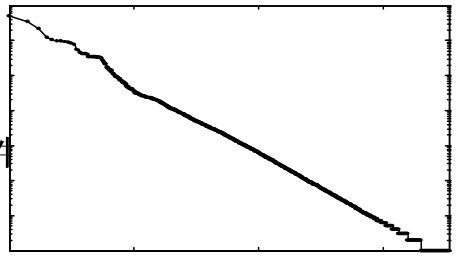

图 4　双对数坐标下某节点域名查询量分布

从总体上看，高查询量的域名和 IP 地址表明它们在日志中极高的重复性，而这种重复性为日志压缩提供了可能性。

(2)IP 地址和域名查询呈现出时间和空间本地化的特征。时间本地化主要是指就某个 IP 地址而言的 DNS 查询量常常有一定的突发性，即同一个 IP 地址短时间内连续发出多个查询。而空间本地化主要是指对同一个域名的查询往往由某个 IP 地址连续发出。这 2 种效应综合作用的结果是 IP 地址和域名查询往往都集中在某个较小的时间段重复出现，而这为提高压缩算法的时间和空间效率提供了基础和可能性。

对 IP 地址和域名的压缩和解压采用了相同的算法，以下仅以 IP 地址为例，域名类同。

算法的主要思想是维持一个包含最近出现的 IP 地址列表，这样与之重复的 IP 地址只需指向之前最先出现的那个即

可。当相同 IP 地址集中在一个时间段出现时，其重复命中率会相当高，从而取得较大的压缩率。

对于 IP 地址列表的数据结构，本文采用简单的数组结构。与链表相比，尽管在 IP 加入和删除操作过程中，数组内部有可能出现未利用的空元素，即空间利用率相对较低，但考虑到需要频繁进行 IP 的加入和删除，链表的这种开销比数组大得多(删除 IP 时，数组元素置空，加入 IP 时，为空数组元素赋值)，因此，采用数组的结构，实际上是以牺牲空间换取更高的时间效率。

IP 地址数组的一个问题是其空间的有限性。一般来说，IP 地址数组记录的 IP 数量越多，其重复命中率就越高。一种极端的情况是，当 IP 地址数组记录了日志中的所有 IP 地址空间时，其重复命中率就为 100%。但显然，过大的 IP 地址数组空间是不合理的，一方面这会增加存储的空间(一般在内存中)，另一方面查找 IP 地址数组的效率也更低(正比于空间的增长)。因此，由 IP 地址和域名查询的时间和空间本地化的特征，令 IP 地址数组只包含最近一段时间内的新出现或重复出现的 IP，这种作法会在保持一个有限的 IP 地址数组空间的同时令大部分的重复 IP 命中。这里涉及一个“最近”的定义问题，为了尽可能减少复杂的时间处理，令连续出现的 IP 数量为最近的量度，即定义“最近”的范围为近 K 个查询。如果 IP 地址数组中的某个 IP 地址经历了超过 K 个查询后，仍没有重复，则把此 IP 从数组中删除。这种作法的假定是，如果某 IP 短期内没有重复，则在较长的一段时间内也不会有重复发生。

对原日志文件中的 IP 地址，压缩处理的结果有 2 种：保留原文和记录数组位置。在压缩端，当处理一个查询的 IP 地址时，首先把它与 IP 地址数组的每一个非空 IP 比较。如果命中，则把此 IP 地址记录为相应的数组位置。否则，在压缩后的日志中保留此 IP，并把它写入数组。

为了记录数组中 IP 经历的无重复次数，对数组中每一个非空的 IP，其每次未命中无重复次数需要进行一个自增的操作，如自增后超过 K，则立即删除。而对于数组中命中的 IP，其无重复次数应立即归零。

对于当前数组中还没有出现的 IP，在完成数组中所有元素的比较后，应当把它写入数组中的空位置。因此在数组元素比较的同时，应记录所遇到的数组中首个空位置，如果遍历数组后，仍未发现空位置，则应该为数组增加一个元素，并写入 IP。

数组的长度是动态变化的，取决于其最后一个非空元素的位置。在从前向后遍历数组比较 IP 的过程中，所记录的最后一个非空元素的位置即为数组的长度。数组中最后一个 IP 在经历 K 次无重复查询后，将被删除(置空)，因此，数组长度会减少 1 或更多；而如前述，如果 IP 未能在数组中找到重复元素，数组长度会增加 1。为了减少比较 IP 时遍历数组的复杂度，理想的情况是数组长度与数组中非空元素的个数差别不大，即数组的非空率很高。而实际上，须考虑一种数组长度远超过数组中非空元素的个数的最坏情况。令日志文件的总体重复率维持一个很高的水平，而在某一时间段内，所有 IP 均不重复，因此，数组长度达到其上限 K(如连续的不重复个数超过 K，则数组中的元素会被依次删除，而其长度保持 K 不变)，令其最后一个元素为长期反复出现的 IP，则其在很长一段时间内，都不会被删除，数组长度保持 K 不变，而由于数组中其他元素多次重复，因此数组中实际平均非空元素个数远小于 K，这会为 IP 比较增加很大的不必要的复杂度。而 DNS 查询时间和空间本地化的特征使得这种情况出现的几率很低。因为 IP 反复重复只会限制在局部的某个时间段，对任何一个 IP 而言，在经历了某个时间段集中的反复查询后，总有机会出现一个足够长的未被查询的时间段，从而从数组中删除掉。IP 压缩算法的伪代码如算法 1 所示。

算法 1 IP 压缩算法伪代码

```
Get a new IP from the query record
// Initialize the boolean variables
Set IP hit to false
Set null hit to false
                    Set found the first null to false
                For each IPi in the IPlib of length L
                    {
                      if (IPi is not null)
{
                          // Record the last non-null IP
the last non-null IP←i
// Judge if IP hit
                                  if (IP = = IPi)
                                  {
                                Set IPi's non-repeatability count to zero
                                      Set IP hit to true
                                      IP hit position←i
                                      break
                                  }
// Plus the count if not hit
else
{
                                IPi's non-repeatability count ++
                                // Set null when reaching the max count
                              if (IPi's non-repeatability count > max)
{
                                      Set IPi to null
                                  }
                                }
                                }
// Record the first null
else if (not found the first null)
{
                          Set the first null to i
                          Set null hit to true
                              Set found the first null to true
                        }
                        }
                  // Save afterwards IP comparison when IP hit
                        if (IP hit)
{
  For each IPi in the IPlib of length L starting from IP hit position
{
                                if (IPi is not null)
{
                          the last non-null IP←i
                          IPi's non-repeatability count ++
                          if (IPi's non-repeatability count > max)
{
                                    Set IPi to null
                                  }
                                }
        else if (not find the first null)
{
                                Set the first null to i
                                Set null hit to true
                                  Set found the first null to true
                              }
                            }
                  // Record the IP position in IPlib only when IP hit
IP ← # IP hit position
// Set the position of the last non-null IP as the new length of IPlib
                              L ← the position of the last non-null IP
                        }
else
{
            // If there are no nulls in IPlib, extend IPlib
if (not null hit)
```

```
{
            Set the first null to L+1
            L++
          }
           IPlibfirst null ← IP
            Set IPlibfirst null's non-repeatability count to zero
        }
```

IP 解压算法的过程与压缩算法相反，伪代码如算法 2 所示。如果压缩后日志文件的某个 IP 为指针，则只需指向 IP 数组中相应位置的 IP 即可。如果压缩后日志文件的某个 IP 为原始保留的 IP，则需要把此 IP 加入到数组，完成与压缩时相同的数组重构。

算法 2 IP 解压算法伪代码

```
Get a new IP from the compressed query record
// Initialize the boolean variables
Set null hit to false
// If IP is not a pointer, insert IP to IPlib
if(IP is not a pointer)
            {
                For each IPi in the IPlib of length L
{
                if(IPi is not null)
                {
        // Set the position of the last non-null IP as IPlib's length
L ← i
                    IPi's non-repeatability count ++
                    if(IPi's non-repeatability count > max)
                    {
                        Set IPi to null
                    }
                  }
                  // If IPi is null, insert IP
else
{
                IPlib[i] ← IP
                Set IPlib[i] to non-null
                Set IPlib[i]'s non-repeatability count to zero
                    IP hit position←i
                    Set null hit to true
                    break
                  }
                }
        // If null hit, there is no need to insert IP any more
if(null hit)
                {
  For each IPi in the IPlib of length L starting from IP hit position
{
                    if(IPi is not null)
                    {
                    L ← i
                    IPi's non-repeatability count ++
                    if(IPi's non-repeatability count > max)
                        {
                          Set IPi to null
                        }
                    }
                  }
                }
                // If not found null in IPlib, extend IPlib
else
                {
                IPlib[L] ← IP
                Set IPlib[L] to non-null
                Set IPlib[L]'s non-repeatability count to zero
                    L++
                }
              }
        // If IP is a pointer, set IP as the corresponding one in IPlib
else
            {
IP ← IPlib[pointer]
            }
```

数组重构的过程需要完成三方面的操作。首先需要找到 IP 数组中的第一个非空的元素，把当前 IP 填入这个元素。如果遍历 IP 数组后仍未找到非空元素，则把 IP 数组的长度增加 1，把当前 IP 填入新增的元素。其次需要为 IP 数组中其他所有非空元素中的 IP 非重复计数增加 1，当计数达到最大的非重复计数 K 时，应把此元素置空。最后应在遍历过程中把每一个非空元素的位置赋给数组长度，这样最终的数组长度就为最后一个非空元素的位置。

3 实验结果分析

用 Java 实现了算法。实验环境为 Intel 双核处理器 1.86 GHz，1 GB 内存，Windows XP 操作系统。

实验数据集为 CN 顶级域名服务器韩国国家互联网发展局(National Internet Development Agency of Korea, NIDA)节点的 BIND 日志文件。选择的日志文件的查询时间从 03-Mar-2009 00:00:01.797~03-Mar-2009 08:00:01.804。文件的大小为 286 007 KB，查询量为 2 955 698。

令最大无重复次数分别为 100、1 000、2 000、3 000，压缩和解压算法的运行时间如图 5 所示。可见，运行时间随最大无重复次数而增加，但解压时间的增量比压缩时间小。压缩和解压的总时间近似与最大无重复次数呈线性关系。由于 IP 和域名的压缩和解压是算法复杂度的主要部分，这一结果表明 IP 和域名的比较次数与算法的复杂度有近似线性的关系。因此，从总的效果来看，最大无重复次数越大，消耗的 CPU 时间越多。

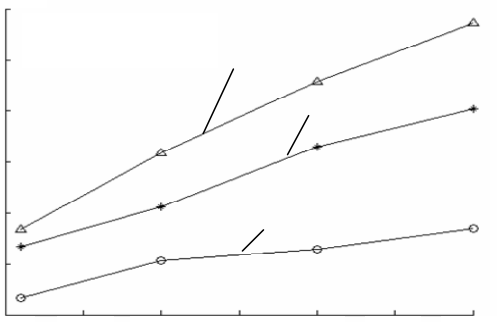

图 5 算法运行时间与最大非重复次数的关系

4 结束语

在实时 DNS 监控和分析系统的部署中，CN 顶级域名 DNS 日志的巨大数据量对从分布式站点到数据处理中心的传输带来了很大的负担。本文利用 DNS 查询类型的冗余性和 DNS 查询时间、IP 和域名的重复性进行 DNS 日志压缩，减轻了日志传输的负担。

参考文献

[1] Albitz P, Liu C. DNS and BIND[M]. Sebastopol, USA: O'Reilly Media, 1998.

[2] Vixie P. BIND Software[EB/OL]. (2009-03-16). https://www.isc.org/software/bind.

[3] Mockapetris P. Domain Names Implementation and Specification[S]. RFC 1035, 1987.

[4] Vixie P. Extension Mechanisms for DNS(EDNS0)[S]. RFC 2671, 1999.

编辑　任吉慧