文章编号：1007-757X(2017)06-0011-06

IP 地址聚合算法的研究与分析

阮晓龙，　路景鑫
（河南中医药大学 网络信息中心，郑州 450008）

摘　要：数据聚合是当今信息化应用中常见技术之一，结合数据聚合思想，针对庞大的 IP 地址信息进行多种聚合算法的研究与实现，并通过可视化的数据分析，对聚合算法进行性能评估。

关键词：IP 地址；聚合算法；性能分析
中图分类号：TP311　　文献标志码：A

Research and Analysis of IP Address Aggregation Algorithms

Ruan Xiaolong, Lu Jingxin
(Network Information Center, HACTCM, Zhengzhou 450008, China)

Abstract: Data aggregation is one of common technologies in today's information technology application. Based on the idea of data aggregation, this paper does a variety of research and implementation of aggregation algorithm for the IP addrese of the vast information, and through the visualization of data analysis, it makes performance evaluation for the aggregation algorithms.

Key words: IP addresses; Aggregation algorithms; Performance analysis

0　引言

随着计算机网络的迅猛发展，基于 IP 地址的安全管理和访问控制的应用方式愈加常态化，IP 地址的聚合应用也逐渐趋于多元化、复杂化，如 IP 地址查询、路由优化、IP 黑名单管理等，都将用到聚合算法。

IP 地址聚合是将一组 IP 地址聚合成一个或多个不能再次聚合的 IP 地址块的过程。进行 IP 地址聚合，主要起到减少地址条目数，提高应用效率的作用。本文将提出三种 IP 地址聚合算法，通过不同算法的实现以及对算法的性能评估，进而充分了解 IP 地址聚合的基本原理和算法之间的差异性。

1　聚合算法概述

1.1　聚合定义

不是所有的 IP 地址或网络段地址都可以聚合，只有当网络前缀相同且连续的地址才能被聚合，形成新的聚合地址块。IP 地址进行聚合的主要思想及判断依据如下所述。

给定两个 IP 地址 p1＝addr1/len1，p2＝addr2/len2，其中 addr 表示 IP 地址，len 表示掩码长度，令 len1≤len2，则有：

a)如果 len1＜len2 且 addr1/len1＝addr2/len1，则称为 p2 对于 p1 是可聚合的，聚合结果为 p1；

b)如果 len1＝len2 且 addr1/(len1－1)＝addr2/(len2－1)，则称 p1 和 p2 是可聚合的，聚合结果为 p3＝addr1/(len1－1)。

1.2　聚合算法的实际应用

在网络环境或实际应用中，经常会见到聚合算法在不同场景下的应用。常见的应用场景如下。

(1)在网络设备中的应用

在网络设备中的应用主要体现在路由器/路由交换机上。当网络中业务量增加时，设备路由表数目也逐渐增多，路由表的增长严重影响路由性能。为减轻设备的负担和提高路由效率，需通过聚合算法将路由汇聚成更少条目的路由，这就是所谓的“路由聚合”。路由聚合最明显的好处是缩小路由表的“尺寸”，通过路由聚合，将大大减轻网络设备的负担以及提高设备的路由效率。

(2)在业务系统中的应用

由于其安全性问题，目前许多业务系统上都配置了访问控制列表。访问控制列表机制是针对每一个 IP 地址设定具体的规则权限，当访问控制列表需要配置的条目数增加时，可通过聚合算法将多地址聚合成较少数量的地址块，从而减少访问控制列表的条目数。

(3)在安全设备中的应用

在安全设备中的应用主要体现在防火墙上。防火墙作为一种内部与外部网络之间的网络安全设备，其添加的策略规则是针对单一地址增加的访问权限。当防火墙中添加的策略规则增多时，可根据相同规则对其作用的 IP 地址进行聚合，以起到减少策略规则数，降低防火墙设备压力的作用。

2　聚合算法的实现

本文采用“除二阶乘”、“除二阶乘优化”和“二进制比对”

基金项目：河南省教育厅自然科学研究(15B520013)
作者简介：阮晓龙(1981-)，男，讲师，研究方向：计算机网络、计算机软件、Web 技术。

3 种算法进行 IP 地址聚合。

2.1 “除二阶乘”聚合算法

(1)算法思想

“除二阶乘”聚合算法是将 IP 地址转换为整数类型，根据相同的网络前缀，对整数类型的 IP 地址进行位移运算，判断 IP 地址能否聚合。当两个 IP 地址能够聚合时，然后对聚合后的 IP 地址块进行输出，得到聚合结果。该算法主要思想是用整数在程序中进行直接运算。

(2)具体实现

使用“除二阶乘”聚合算法进行地址聚合时，主要的实现过程，如图 1 所示。

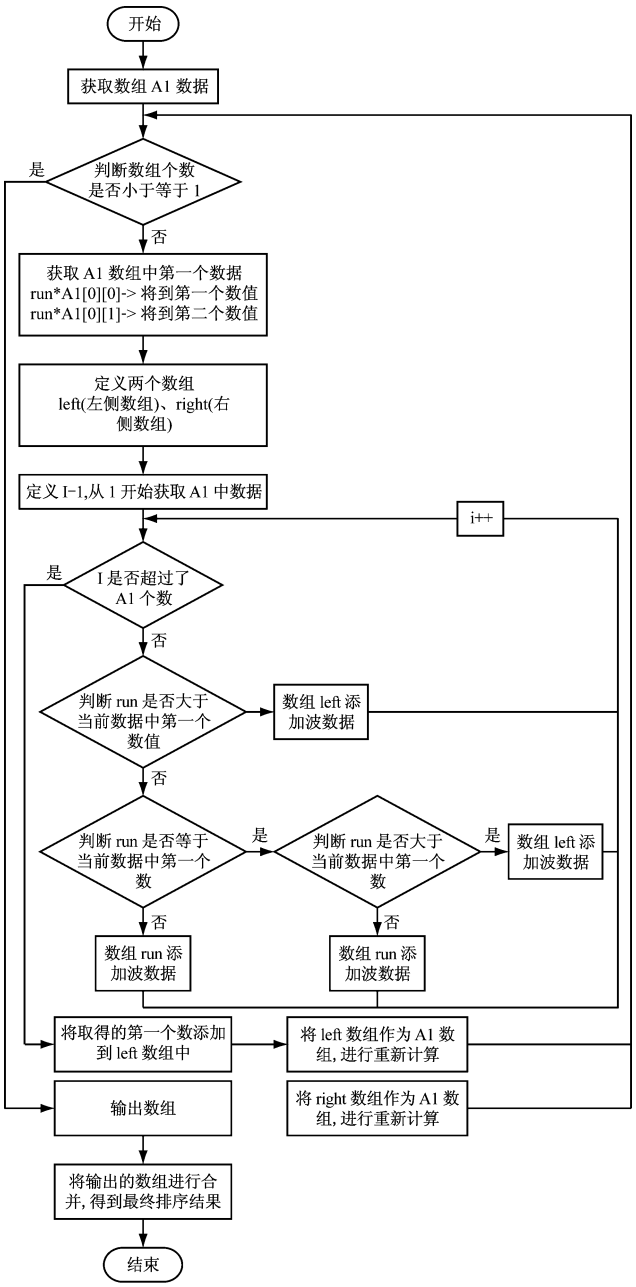

图 1 “除二阶乘”聚合算法过程

“除二阶乘”聚合算法主要分为 4 个步骤。

第一步：原始数组整理。程序中获取一组需要聚合的 IP 地址，在进行聚合前首先要“洗数据”，将原始数组中的数据进行排序后并去除包含关系。

“排序”的主要算法逻辑如下：

①将点分式加前缀表示的 IP 地址(如：192.168.1.2/32)以“/”进行分割，从左到右得到两个数值 ip 和 prefix，这两个数值分别代表 IP 地址和网络前缀，将得到的 ip 值转换成整数类型，存入数组中；

②“排序”算法主要满足以下两点规则：

a)ip 数值越小的数据排序越靠前；

b)ip 数值相同的数据，prefix 数值越小排序越靠前。

③本过程中使用的排序方法为快速排序法，具体的排序流程，如图 2 所示。

第二步：数据重新组合。由于上步操作中，已经去除数组的包含关系，所以能够聚合的地址，其网络前缀也必相等。本步骤将整理后的数组重新调整，将相同网络前缀的地址放在一起，并将网络前缀作为“键值”，最终形成以“网络前缀”为键值的数组 A4。

第三步：地址聚合计算。相同网络前缀的地址进行聚合时主要的聚合算法逻辑如下所述。

①将地址进行从小到大排序，排序结束后，依次取出一个地址，与上个地址进行聚合比对，若可以聚合则进行聚合运算，否则将该地址添加到输出数组中，具体的聚合算法过程如下：

a)取出 ip 数值，将 ip 数值除以 2 的(33-prefix)次方，并将结果进行“向下取整”运算，计算方法为：

quotient=floor(ip/pow(2,(33- prefix)))；//获取子网掩码前所有数值

b)如果与上个数值的 quotient 相等，则说明两个地址能够聚合，并将网络前缀数减 1，取上个地址的 ip 与网络前缀数减 1 的值，作为聚合后的结果，将结果添加到数组 A5 中；

c)如果与上个数值的 quotient 不相等，则说明两个地址不能聚合，将该地址添加到数组 A6 中；

d)判断数组 A5 是否为空，若为空，则输出数组 A6，A6 则为聚合后的结果；若不为空，则将 A5 按照上述方法继续进行聚合计算。

第四步：递归聚合计算。当获取到聚合后数组时，还需要再次做聚合计算，判断输出的数组能否再进行聚合。当再次聚合计算完成后，判断两次数组的地址块数量是否发生变化。

如果地址块数量发生变化，则说明本次计算进行了地址聚合，并将聚合后的结果再次做递归计算；如果地址块数量没有发生变化，则说明本次计算没有进行地址聚合，将结果数据进行输出，输出的聚合结果最终满足“形成一个或多个不能再次聚合的 IP 地址块”的聚合定义。

(3)实现代码

“除二阶乘”聚合算法中根据相同网络前缀数进行聚合计算，具体实现代码如下所示。

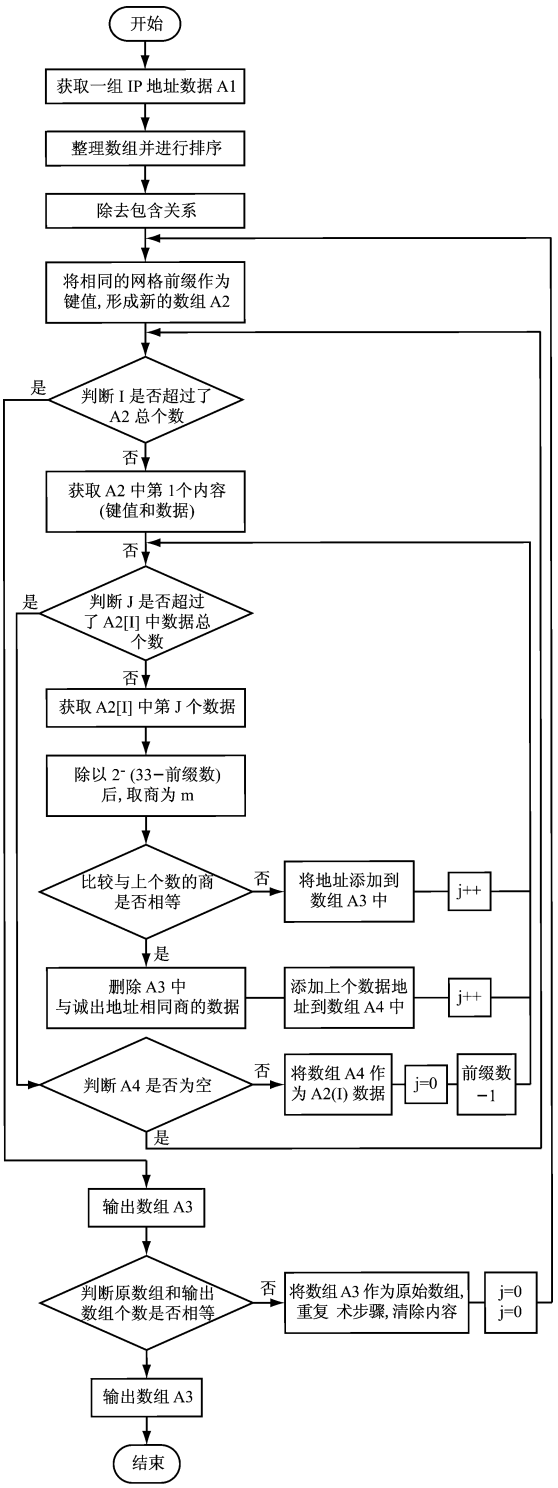

图 2　排序流程

```
//相同网络前缀数的地址聚合计算
function getTempArray(arr){
    $tempArray=array();
    $outputArray=array();
    $tempquotient=0;
    $len = count(arr);
   for($i = 0; $i<len; $i++) {
        //遍历获取数组中数据,并进行计算
        $quotient=floor( $arr[ $i]/pow(2,(33- prefix)));
        //对上个地址的"商"进行比较,判读两个地址能否合并
        if ( $quotient == $tempquotient) {
          array_pop(outputArray);
          //取上个地址,并将网络前缀数减 1 作为聚合后结果
            $tempArray[] = ( $outputArray[ $i-1][0], prefix-1);
        }
        else {
        //如果比较不相等,则对该地址的"商"以及网络前缀数进行赋值
        $tempquotient = quotient;
        $netmask = prefix;
        //将不能合并的地址添加到输出数组中
        $outputArray[]=array(arr[i],netmask);
        }
      }
     return tempArray;
   }
```

2.2　"除二阶乘优化"聚合算法

(1)算法思想

"除二阶乘优化"聚合算法是将 3.1 中的聚合算法进行优化,其主要优化之处是将 IP 地址按网络前缀数从高到低进行聚合,在聚合计算过程中,网络前缀数高的 IP 地址聚合完成后,将形成"聚合后"数组和"不能聚合"数组。

"聚合后"数组将聚合结果与网络前缀数低的 IP 地址数组进行数组合并,并做聚合计算;"不能聚合"数组中的 IP 地址在后续的聚合过程中也不会发生聚合,所以将这些地址直接作为输出数组进行输出。

根据该聚合算法与"除二阶乘"聚合算法的逻辑思想对比,可以看出该算法在每次聚合过程中参与计算的 IP 地址块数量将逐渐减少,从而减少程序的计算量,提高计算效率。

(2)具体实现

使用"除二阶乘优化"聚合算法对 IP 地址进行聚合,主要的实现过程,如图 3 所示。

"除二阶乘优化"聚合算法主要分为三个步骤,分别是原始数组整理、数据重新组合和地址聚合计算。

在"原始数组整理"和"数据重新组合"两个过程中与"除二阶乘"聚合算法保持一致,"地址聚合计算"将通过网络前缀从高到低优化的聚合算法,将大大减少数据的聚合计算时间。

(3)实现代码

"除二阶乘优化"聚合算法的实现过程中,将网络前缀从高到低依次进行聚合的具体实现代码如下。

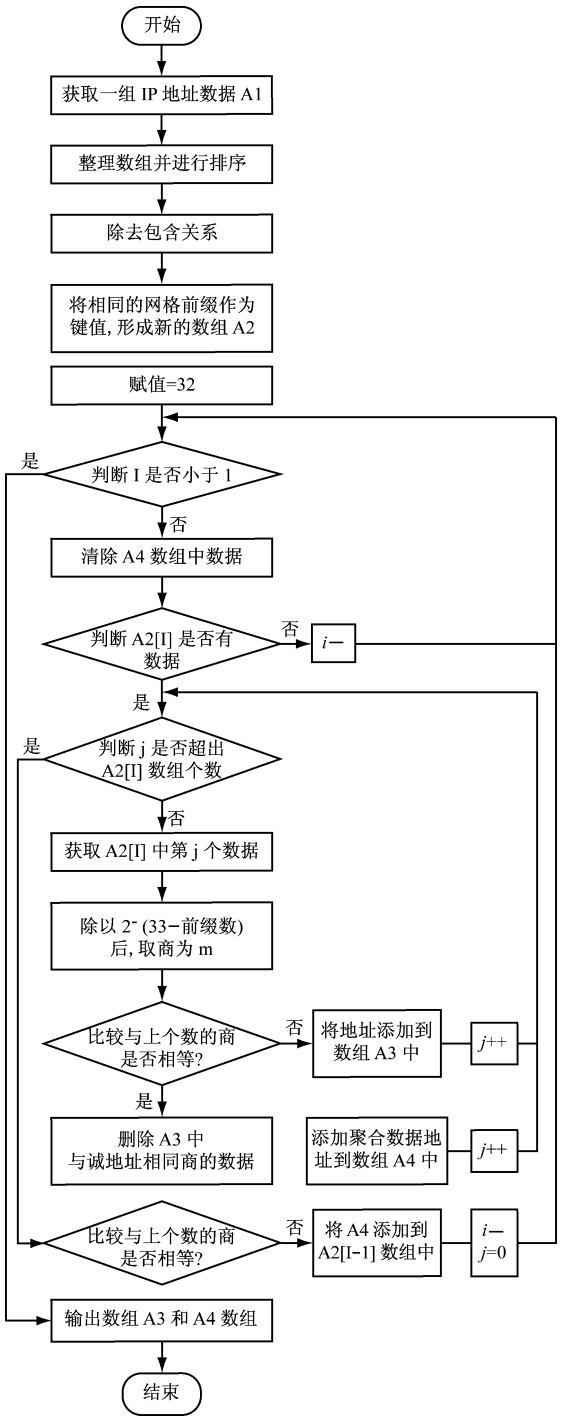

图 3　“除二阶乘优化”聚合过程

```
//递归进行聚合计算
function recursion(){
  //按网络前缀数从高到低依次进行地址聚合
  for($i=32;$i>=0;$i--){
    //判断是否存在该网络前缀数的数组
    if(isset(this->allArray[i])
      && !empty(this->allArray[i])){
      //获取到该数组中的数据
      $array= $this->allArray[i];
      sort($array);
      //获取聚合后的结果
        $result = $this-> getTempArray ( $array, $i);
      //判断网络前缀数减 1 的数组是否存在
      if( !isset( $this->allArray[ $i-1])){
        //如果不存在，初始化一个数组
        $this->allArray[i-1]=[];
      }
      //将聚合后的结果添加到网络前缀数减 1 的数组中
$this->allArray[ $i-1]=array_merge(this->allArray[ $i-1],result)
        }
      }
    }
```

2.3　**“二进制比对”聚合算法**

(1)**算法思想**

“二进制比对”聚合算法，主要是将 IP 地址转换成二进制字符串，根据相同的网络前缀，对转换的二进制 IP 地址进行比较，从而判断 IP 地址能否聚合。当 IP 地址能够聚合时，将网络前缀数减 1，并结合聚合后 IP 地址进行输出，得到最终聚合结果。

该算法采用了换算二进制的方法，虽然该算法过程较为复杂，但算法的逻辑思想较为简单，能够让人们更易理解聚合计算原理。

(2)**具体实现**

使用“二进制比对”聚合算法对 IP 地址进行聚合，主要的实现过程，如图 4 所示。“二进制比对”聚合算法与“除二阶乘”聚合算法具体实现过程基本一致，在运算的过程中将数据进行二进制比对处理，最终活动聚合结果。

(3)**实现代码**

“二进制比对”聚合算法中去除包含关系，需计算每个地址块的范围，具体实现代码如下。

```
//获取地址块范围方法
function getRange(ip,net){
  //定义最大值字符串
  $max=" 11111111111111111111111111111111";
  //定义最小值字符串
 $min=" 00000000000000000000000000000000";
  //根据网络前缀数计算地址二进制数
  $ipstr=getNetString( $ip, $net);
  $max1=substr( $max, $net,(32-$net));
  $min1=substr( $min, $net,(32-$net));
  //获得地址最小值的字符串
  $minbit= $bit1. $min1;
  //根据二进制计算最小 IP 地址
  $minIP=getBitToIP(minbit);
  //获得地址最大值的字符串
  $maxbit= $bit1. $max1;
```

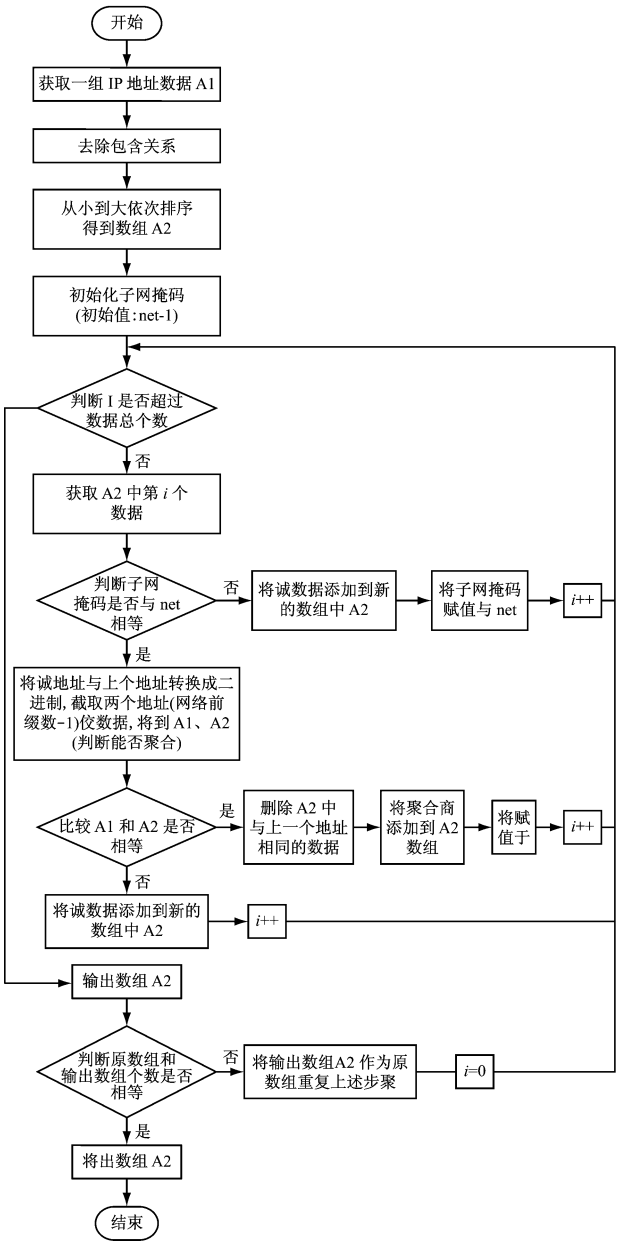

图 4　“二进制比对”聚合算法过程

```
    //根据二进制计算最大 IP 地址
    $ maxIP=GetBitToIP( $ maxbit);
    //添加地址范围数组
    $ result=array( $ minbit, $ maxbit);
    return result;
}
```

3　聚合算法的性能分析

为了评估不同算法的计算性能，本文对不同算法进行了多指标点的性能分析，具体分析指标点的详细介绍，如表 1 所示。

在测试分析过程中，受测试条件约束，本文在测试千万数量级时，不再进行“二进制比对”聚合算法的测试且其他两种算法仅进行 1 至 5 千万数据量的分析。

(1)时间复杂度分析

对万数量级、十万数量级、百万数量级、千万数量级的 IP 地址进行聚合，不同算法聚合计算所消耗的时间对比，如图 5、图 6、图 7、图 8 所示。

通过对时间复杂度的对比分析，可得出以下几点结论：

①数量级增大时，聚合算法所消耗的时间也将线性增加；

②算法优化，可以提升计算效率，无论什么级别数量的地址进行聚合时，“除二阶乘优化”聚合算法所消耗的时间相比“除二阶乘”聚合算法消耗的时间明显缩短。

(2)CPU 频率使用量分析

对万数量级、十万数量级、百万数量级、千万数量级的 IP 地址进行聚合，不同算法聚合计算所消耗的 CPU 频率使用量对比，如图 9 所示。

通过对 CPU 频率使用量对比图进行分析，可得出 CPU 频率使用量在增加到最大频率之前时，3 种算法占用服务器资源的大小关系为：“二进制比对”>“除二阶乘”>“除二阶乘优化”。

(3)CPU 使用率分析

对 CPU 使用率进行分析主要为了查看不同算法随着时间的增长，CPU 使用率的增长速率变化情况。本次实验将取 100万作为基本数量级，并获取服务器在开始计算后，前

表 1　性能分析指标点详细介绍

分析指标		时间	CPU 频率使用量	CPU 使用率	内存使用量	内存使用率
图表类型		折线图	折线图	折线图	折线图	折线图
坐标轴意义	X	聚合地址个数	聚合地址个数	时间	聚合地址个数	时间
	Y	聚合计算消耗时间	CPU 频率使用量大小	CPU 使用率大小	内存使用量大小	内存使用率大小
测量方法		控制变量法，不断改变计算数量，查看时间消耗情况	控制变量法，不断改变计算数量，查看 CPU 频率使用情况	进行相同数量的地址聚合，查看 CPU 使用增长速率	控制变量法，不断改变计算数量，查看内存使用情况	进行相同数量的地址聚合，查看内存使用增长速率
目的说明		比较不同算法的聚合计算能力，判断能否支持庞大数量的 IP 地址聚合	比较不同算法在不同数量级地址聚合时，消耗服务器 CPU 频率使用量的变化趋势	比较不同算法在进行聚合计算时，调用服务器 CPU 资源的能力	比较不同算法在不同数量级地址聚合时，消耗服务器内存资源的变化趋势	比较不同算法在进行聚合计算时，调用服务器内存资源的能力

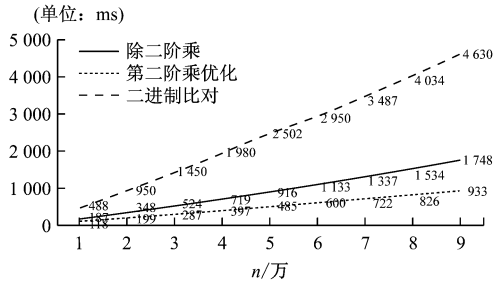

图 5　万数量级地址聚合时间对比图

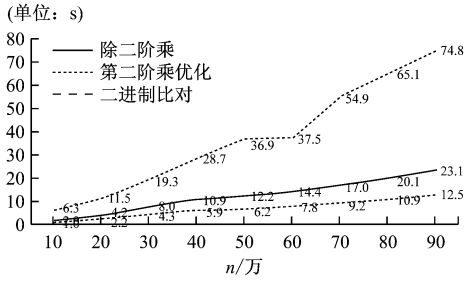

图 6　十万数量级地址聚合时间对比图

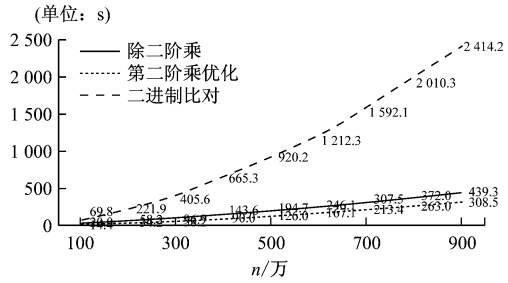

图 7　百万数量级地址聚合时间对比图

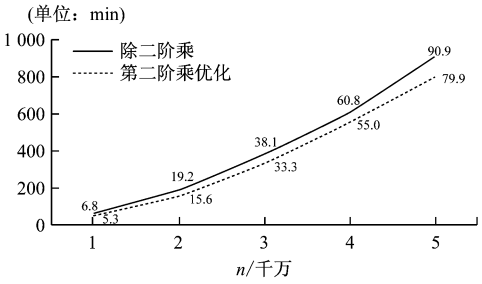

图 8　千万数量级地址聚合时间对比图

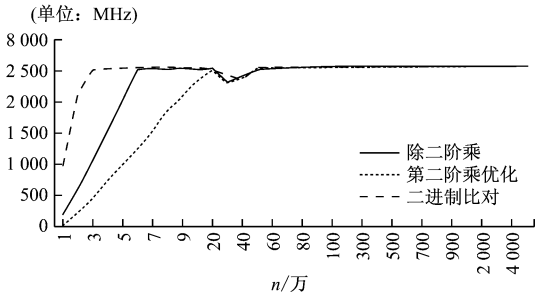

图 9　消耗 CPU 频率使用量对比图

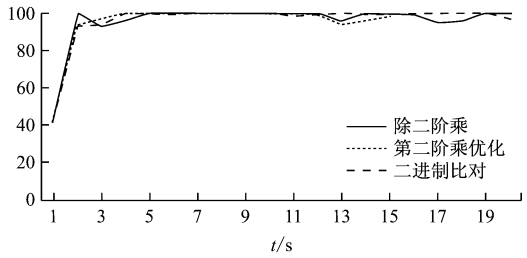

图 10　消耗 CPU 使用量对比图

20 s 的监控数据进行分析，分析结果，如图 10 所示。

通过对 CPU 使用率对比图进行分析，可得出“除二阶乘”聚合算法的 CPU 使用率增长速度大于其他两种算法，但是总体上三种聚合算法的 CPU 使用率增长速度基本保持一致。

(4)内存使用量分析

对万数量级、十万数量级、百万数量级、千万数量级的 IP 地址进行聚合计算，不同算法所消耗的内存使用量对比，如图 11 所示。

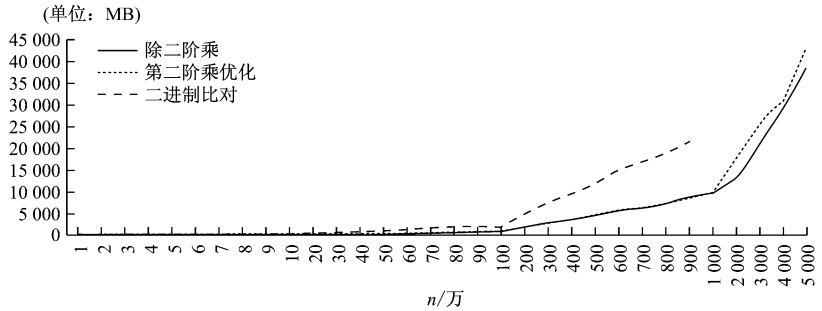

图 11　IP 地址聚合所消耗内存使用量对比图

通过内存使用量对比图可得出服务器内存的使用量会随着聚合地址的数量增大而增大，当聚合地址的数量增加到 100 万以后，服务器内存使用量的增率明显提高。

(5)内存使用率分析

对内存使用率进行分析主要为了查看不同算法随着时间的增长，内存使用率的增长速率变化情况。本次实验将取 100 万作为基本数量级，并获取服务器在开始计算后，前 20 s 的监控数据进行分析，分析结果，如图 12 所示。

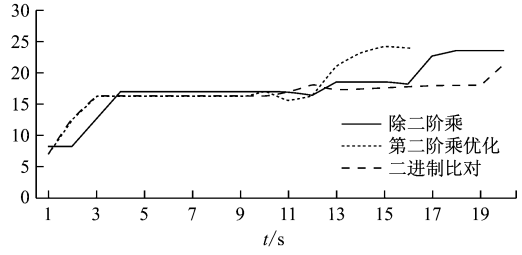

图 12　不同算法所消耗内存使用率随时间变化趋势图

（下转第 73 页）

表 2　使用 100 W 负载校正后的功率

功率	仪器输出功率（平均值）	功率（平均值）	误差
1	0.99	1.07	8.08%
2	1.99	2.06	3.52%
5	5.01	5.12	2.20%
10	9.99	9.88	−1.10%
50	49.97	49.79	−0.36%
100	100.04	100.11	0.07%
250	250.27	251.32	0.42%
500	500.11	501.21	0.22%
1 000	999.96	998.79	−0.12%
2 000	1 999.95	1 998.31	−0.08%

为获取小功率时的计量准确度，建议选用小功率的负载进行校准。

3.2　智能插座测试

智能插座设计目标通过功率计量切断插座内其余插孔电源。测试实验方法：将测试设备接入主孔位进行功率计量，并接入功率计分别测试运行功率、待机功率 10 次，记录并求取平均值。选用稳压电源，电烙铁作为负载接入联锁孔位，通过手机根据不同测试设备的功率平均值设置判决门限，延时时间 10 秒，具体设置，如表 3 所示。

表 3　智能插座测试设置值

测试设备	运行功率（平均值）	待机功率（平均值）	智能插座设置值
联想便携式计算机 X230i(不含电池)	38.9 W	3.57	15 W
联想启天 M7160 主机＋联想 L2250pwD 显示器＋公牛 GN316 电插排	189.85 W	47.26 W	70 W

测试结果：智能插座设置后，根据测试设备待机或运行状态，插座能够有效切断或恢复对联锁孔位的负载供电。

4　总结

基于功率计量的智能插座，可以有效提高传统插座的智能化水平，通过功率计量有效切断相关电器电源，解决了生产办公、日常生活中电器管理的不便。智能插座可以有效应用于智能家居、日常办公、实验开发等环境，提高管理效能、节省电量。

参考文献

[1] 韩强，陈金周，冯小军，等. 智能节能插座的研制[J]. 家电科技，2014(8)：74-76.

[2] 孙杰，胡乃军，郭志卓. 基于物联网的智能插座设计[J]. 中国科技信息，2015(5)：96-98.

[3] 高雄. 智能家居系统中智能插座的设计[J]. 信息技术，2016(10)：168-171.

[4] Cho W T，Ma Y W，Huang Y M. A Smart Socket-Based Multiple Home Appliance Recognition Approach over IoT Architecture[J]. Journal of Internet Technology，2015，16(7)：1227-1238.

[5] 孙茂超. 物联网环境下智能插座的设计[J]. 电子制作，2015(6)：53.

[6] 陈园园，袁焕丽. 基于单片机的脉冲频率测量系统的设计[J]. 智能计算机与应用，2016，6(4)：83-84.

[7] 殷勤奋，汤宇. 浅析几种基于单片机的数字频率测量仪的设计[J]. 教育教学论坛，2011(14)：229-230.

[8] 李伟，李一真，王志梁，等. 基于物联网技术的智能插座设计与实现[J]. 山东电力技术，2015，42(6)：55-58.

（收稿日期：2016.12.30）

（上接第 16 页）

通过对内存使用率对比图进行分析，可得出“二进制比对”聚合算法的内存使用率增长速度大于其他两种算法，说明了“二进制比对”聚合算法在计算过程中消耗大量的服务器内存资源。

4　总结

本文主要论述了 IP 地址聚合算法的基本原理及应用模式，并通过三种不同算法对不同级别数量的 IP 地址进行对比分析。通过本文不仅能够充分的了解 IP 地址聚合算法的实现原理，而且通过不同级别数据量的测试对比分析结果，能够充分体现出算法对大数据量计算性能的重要性。

参考文献

[1] 查普尔(Chappell，L. A)，蒂特尔(Tittel，E.). TCP/IP 协议 原理与应用[M]. 马海军，吴华，译. 北京：清华大学出版社，2005.

[2] 谢希仁. 计算机网络[M]. 电子工业出版社，2003.6.

[3] 陈秀莉. 浅论无类域间路由与可变长子网掩码技术[J]. 安徽电气工程职业技术学院学报，2004(3)：90-92.

（收稿日期：2017.03.24）