(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 104243425 B
(45)授权公告日 2018.09.04

(21)申请号 201310244635.1

(22)申请日 2013.06.19

(65)同一申请的已公布的文献号
申请公布号 CN 104243425 A

(43)申请公布日 2014.12.24

(73)专利权人 深圳市腾讯计算机系统有限公司
地址 518057 广东省深圳市南山区高新区高新南一路飞亚达大厦5-10楼

(72)发明人 庄奇东

(74)专利代理机构 广州三环专利商标代理有限公司 44202
代理人 郝传鑫 熊永强

(51)Int.Cl.
H04L 29/06(2006.01)
G06F 17/30(2006.01)

(56)对比文件
CN 101446984 A,2009.06.03,
CN 1932821 A,2007.03.21,
CN 102438020 A,2012.05.02,
CN 101706814 A,2010.05.12,
CN 103095688 A,2013.05.08,

审查员 文华胤

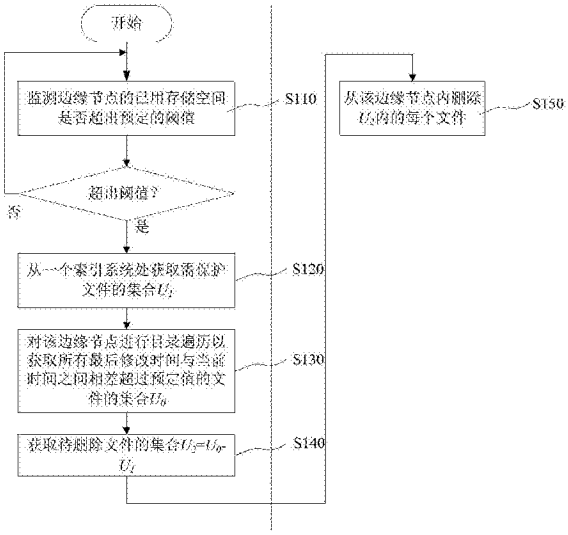
权利要求书5页 说明书21页 附图19页

(54)发明名称

一种在内容分发网络中进行内容管理的方法、装置及系统

(57)摘要

本发明涉及一种在内容分发网络中进行内容管理的方法及装置，上述方法包括：监测边缘节点的已用存储空间是否超出预定的阈值，若是则执行以下步骤：从一个索引系统处获取需保护文件的集合U_1；对该边缘节点进行目录遍历以获取所有最后修改时间与当前时间之间相差超过预定值的文件的集合U_0；获取待删除文件的集合$U_2=U_0-U_1$；以及从该边缘节点内删除U_2内的每个文件。根据上述的方法及装置，在内容分发网络中进行内容管理具有硬件开销低、速度快，且不影响服务可用率、不引起服务质量下降的优点。



CN 104243425 B

1.一种在内容分发网络中进行内容管理的方法,其特征在于,包括:

监测边缘节点的已用存储空间是否超出预定的阈值,若是则执行以下步骤:

向代理服务器发送淘汰预警信号;

该代理服务器根据该淘汰预警信号向索引系统发送获取文件请求并相应获取需保护文件的集合U_1;以及

接收该代理服务器返回的该需保护文件的集合U_1;

对该边缘节点进行目录遍历以获取所有最后修改时间与当前时间之间相差超过预定值的文件的集合U_0;

获取待删除文件的集合$U_2=U_0-U_1$;以及

从该边缘节点内删除U_2内的每个文件。

2.如权利要求1所述的在内容分发网络中进行内容管理的方法,其特征在于,该需保护文件的集合U_1包括:在过去第一预定时间内至少存在某天访问次数大于1的文件集合、过去第二预定时间内上传但未发布的文件的集合、以及所有需要永久保存的文件的集合。

3.如权利要求2所述的在内容分发网络中进行内容管理的方法,其特征在于,与该代理服务器之间的网络传输采用加密方式进行处理。

4.如权利要求2所述的在内容分发网络中进行内容管理的方法,其特征在于,还包括:

接收来自该代理服务器的目录遍历请求及该预定值,对该边缘节点进行目录遍历以获取所有最后修改时间与当前时间之间相差超过预定值的文件的集合U_0是根据来该目录遍历请求进行的。

5.如权利要求2所述的在内容分发网络中进行内容管理的方法,其特征在于,还包括:该代理服务器将该第一预定时间及第二预定时间发送至该索引系统。

6.如权利要求2所述的在内容分发网络中进行内容管理的方法,其特征在于,还包括:在该代理服务器中根据用户输入获取该预定值或者读取预先的设定信息以获取该预定值。

7.如权利要求1所述的在内容分发网络中进行内容管理的方法,其特征在于,还包括:接收一个客户端发送的文件,以及将接收到的文件存储在连接至该边缘节点的存储器内;

该文件的存储路径为:/DATAx/(CHANNEL/)y/z/filename;

其中DATAx表示索引号为x的存储器的根目录,CHANNEL表示频道名,且CHANNEL子目录可选,其后下一级子目录为y,filename为该文件的文件名,且x、y、z满足以下条件:

$$\begin{cases} x = \mathrm{DiskID}, & \mathrm{DiskID} \le 256/\mathrm{N}; \\ y = hash_1(filename) \bmod A, & 96 \le A \le 128; \\ z = hash_2(filename) \bmod B, & 96 \le B \le 128. \end{cases}$$

N表示边缘节点的数目,$hash_1$与$hash_2$为两个不同的哈希函数。

8.如权利要求7所述的在内容分发网络中进行内容管理的方法,其特征在于,在对该边缘节点进行目录遍历之前还包括:

通过网络文件系统将连接至该边缘节点的存储器内的文件挂载至一个虚拟路径下,该虚拟路径为:

/data/k/(CHANNEL/)y/z/filename;

其中,CHANNEL子目录可选,其后下一级子目录为y,k≤10000,且k与x之间为均匀的多

对一映射关系。

9.如权利要求7-8任一项所述的在内容分发网络中进行内容管理的方法，其特征在于，对该边缘节点进行目录遍历时采用广度优先遍历。

10.如权利要求7-8任一项所述的在内容分发网络中进行内容管理的方法，其特征在于，对该边缘节点进行目录遍历时采用非递归遍历。

11.如权利要求1所述的在内容分发网络中进行内容管理的方法，其特征在于，对该边缘节点进行目录遍历时，对于同一个存储器内的目录，其遍历过程串行进行，而不同的存储器的遍历过程并行进行。

12.如权利要求1所述的在内容分发网络中进行内容管理的方法，其特征在于，对该边缘节点进行目录遍历的过程中还将遍历进程绑定至特定的处理器核心。

13.如权利要求1所述的在内容分发网络中进行内容管理的方法，其特征在于，对该边缘节点进行目录遍历的结果按照索引节点、文件索引节点所在块号、或者文件目录项所在块号进行排序。

14.一种在内容分发网络中进行内容管理的装置，其特征在于，包括：

监测模块，用于监测边缘节点的已用存储空间是否超出预定的阈值，若是则调用文件删除模块；

该文件删除模块包括：

第一获取单元用于：

向代理服务器发送淘汰预警信号；

该代理服务器根据该淘汰预警信号向索引系统发送获取文件请求并相应获取需保护文件的集合U_1；以及

接收该代理服务器返回的该需保护文件的集合U_1；

目录遍历单元，用于对该边缘节点进行目录遍历以获取所有最后修改时间与当前时间之间相差超过预定值的文件的集合U_0；

第二获取单元，用于获取待删除文件的集合$U_2=U_0-U_1$；以及

文件删除单元，用于从该边缘节点内删除U_2内的每个文件。

15.如权利要求14所述的在内容分发网络中进行内容管理的装置，其特征在于，该需保护文件的集合U_1包括：在过去第一预定时间内至少存在某天访问次数大于1的文件集合、过去第二预定时间内上传但未发布的文件的集合、以及所有需要永久保存的文件的集合。

16.如权利要求15所述的在内容分发网络中进行内容管理的装置，其特征在于，与该代理服务器之间的网络传输采用加密方式进行处理。

17.如权利要求15所述的在内容分发网络中进行内容管理的装置，其特征在于，该文件删除模块还包括：

接收单元，用于接收来自该代理服务器的目录遍历请求及该预定值，该目录遍历单元是根据该目录遍历请求进行目录遍历操作。

18.如权利要求15所述的在内容分发网络中进行内容管理的装置，其特征在于，该代理服务器还包括发送模块，用于将该第一预定时间及第二预定时间发送至该索引系统。

19.如权利要求15所述的在内容分发网络中进行内容管理的装置，其特征在于，该代理服务器还包括获取模块，用于根据用户输入获取该预定值或者读取预先的设定信息以获取

该预定值。

20.如权利要求14所述的在内容分发网络中进行内容管理的装置，其特征在于，还包括：

文件存储模块，用于接收一个客户端发送的文件，以及将接收到的文件存储在连接至该边缘节点的存储器内；

该文件的存储路径为：/DATAx/(CHANNEL/)y/z/filename；

其中DATAx表示索引号为x的存储器的根目录，CHANNEL表示频道名，且CHANNEL子目录可选，其后下一级子目录为y，filename为该文件的文件名，且x、y、z满足以下条件：

$$\begin{cases} x = \mathrm{DiskID}, & \mathrm{DiskID} \le 256/\mathrm{N}; \\ y = hash_1(filename)\ mod\ A, & 96 \le A \le 128; \\ z = hash_2(filename)\ mod\ B, & 96 \le B \le 128. \end{cases}$$

N表示边缘节点的数目，$hash_1$与$hash_2$为两个不同的哈希函数。

21.如权利要求20所述的在内容分发网络中进行内容管理的装置，其特征在于，还包括目录挂载模块，用于在该目录遍历单元进行目录遍历之前通过网络文件系统将连接至该边缘节点的存储器内的文件挂载至一个虚拟路径下，该虚拟路径为：

/data/k/(CHANNEL/)y/z/filename；

其中，CHANNEL子目录可选，其后下一级子目录为y，k≤10000，且k与x之间为均匀的多对一映射关系。

22.如权利要求20-21任一项所述的在内容分发网络中进行内容管理的装置，其特征在于，该目录遍历单元对该边缘节点进行目录遍历时采用广度优先遍历。

23.如权利要求20-21任一项所述的在内容分发网络中进行内容管理的装置，其特征在于，该目录遍历单元对该边缘节点进行目录遍历时采用非递归遍历。

24.如权利要求14所述的在内容分发网络中进行内容管理的装置，其特征在于，该目录遍历单元对该边缘节点进行目录遍历时，对于同一个存储器内的目录，其遍历过程串行进行，而不同的存储器的遍历过程并行进行。

25.如权利要求14所述的在内容分发网络中进行内容管理的装置，其特征在于，该目录遍历单元对该边缘节点进行目录遍历的过程中还将遍历进程绑定至特定的处理器核心。

26.如权利要求14所述的在内容分发网络中进行内容管理的装置，其特征在于，该目录遍历单元对该边缘节点进行目录遍历的结果按照索引节点、文件索引节点所在块号、或者文件目录项所在块号进行排序。

27.一种在内容分发网络中进行内容管理的系统，包括：

主控服务器、索引服务器、代理服务器；

该主控服务器用于监测该内容分发网络的边缘节点的已用存储空间是否超出预定的阈值，若是则：

向该代理服务器发送淘汰预警信号；

该代理服务器根据该淘汰预警信号向该索引服务器发送获取文件请求并相应获取需保护文件的集合U_1；以及

接收该代理服务器返回的该需保护文件的集合U_1；

对该边缘节点进行目录遍历以获取所有最后修改时间与当前时间之间相差超过预定值的文件的集合U_0；

获取待删除文件的集合$U_2=U_0-U_1$；以及

从该边缘节点内删除U_2内的每个文件；

该索引服务器用于根据请求返回该需保护文件的集合U_1。

28.如权利要求27所述的在内容分发网络中进行内容管理的系统，其特征在于，该需保护文件的集合U_1包括：在过去第一预定时间内至少存在某天访问次数大于1的文件集合、过去第二预定时间内上传但未发布的文件的集合、以及所有需要永久保存的文件的集合。

29.如权利要求28所述的在内容分发网络中进行内容管理的系统，其特征在于，该代理服务器，用于在该主控服务器与该索引服务器之间提供通讯。

30.如权利要求29所述的在内容分发网络中进行内容管理的系统，其特征在于，该代理服务器采用加密方式处理该主控服务器与该索引服务器之间的通讯。

31.如权利要求29所述的在内容分发网络中进行内容管理的系统，其特征在于，该主控服务器还用于：

接收来自该代理服务器的目录遍历请求及该预定值，该主控服务器是根据该目录遍历请求进行目录遍历操作。

32.如权利要求29所述的在内容分发网络中进行内容管理的系统，其特征在于，该代理服务器还用于：将该第一预定时间及第二预定时间发送至该索引系统。

33.如权利要求29所述的在内容分发网络中进行内容管理的系统，其特征在于，该代理服务器还用于：根据用户输入获取该预定值或者读取预先的设定信息以获取该预定值。

34.如权利要求27所述的在内容分发网络中进行内容管理的系统，其特征在于，该主控服务器还用于：

接收一个客户端发送的文件，以及将接收到的文件存储在连接至该边缘节点的存储器内；

该文件的存储路径为：/DATAx/(CHANNEL/)y/z/filename；

其中DATAx表示索引号为x的存储器的根目录，CHANNEL表示频道名，且CHANNEL子目录可选，其后下一级子目录为y，filename为该文件的文件名，且x、y、z满足以下条件：

$$\begin{cases} x = \mathrm{DiskID}, & \mathrm{DiskID} \le 256/\mathrm{N}; \\ y = hash_1(filename) \bmod A, & 96 \le A \le 128; \\ z = hash_2(filename) \bmod B, & 96 \le B \le 128. \end{cases}$$

N表示边缘节点的数目，$hash_1$与$hash_2$为两个不同的哈希函数。

35.如权利要求34所述的在内容分发网络中进行内容管理的系统，其特征在于，该主控服务器还用于：在该进行目录遍历之前通过网络文件系统将连接至该边缘节点的存储器内的文件挂载至一个虚拟路径下，该虚拟路径为：

/data/k/(CHANNEL/)y/z/filename；

其中，CHANNEL子目录可选，其后下一级子目录为y，$k \le 10000$，且k与x之间为均匀的多对一映射关系。

36.如权利要求34-35任一项所述的在内容分发网络中进行内容管理的系统，其特征在

于，该主控服务器对该边缘节点进行目录遍历时采用广度优先遍历。

37.如权利要求34-35任一项所述的在内容分发网络中进行内容管理的系统，其特征在于，该主控服务器对该边缘节点进行目录遍历时采用非递归遍历。

38.如权利要求27所述的在内容分发网络中进行内容管理的系统，其特征在于，该主控服务器对该边缘节点进行目录遍历时，对于同一个存储器内的目录，其遍历过程串行进行，而不同的存储器的遍历过程并行进行。

39.如权利要求27所述的在内容分发网络中进行内容管理的系统，其特征在于，该主控服务器对该边缘节点进行目录遍历的过程中还将遍历进程绑定至特定的处理器核心。

40.如权利要求27所述的在内容分发网络中进行内容管理的系统，其特征在于，该主控服务器对该边缘节点进行目录遍历的结果按照索引节点、文件索引节点所在块号、或者文件目录项所在块号进行排序。

一种在内容分发网络中进行内容管理的方法、装置及系统

技术领域

[0001] 本发明涉及内容分发网络(Content Delivery Network,CDN),特别涉及一种在内容分发网络中进行内容管理的方法、装置及系统。

背景技术

[0002] 内容分发网络的目的是通过在现有的互联网中增加一层新的网络架构,将热点资源或用户所需的资源分发到最接近用户的边缘节点上,使用户可以就近取得所需的内容,解决互联网网络拥挤的状况,提高用户访问资源的响应速度。

[0003] 图1所示为一个典型的CDN架构示意图,其中,DC表示数据中心,AC表示区域中心,OC表示边缘节点。当用户请求在距用户网络拓扑距离最近的OC节点命中资源时,由OC直接提供服务;若不命中,则OC节点中的Web服务器返回HTTP302回复,并把用户重定向到DC或AC,这个过程称作一次“回源”。

[0004] 在一个CDN系统中,提高服务质量(Quality of Service,QoS)的一个重要手段就是尽可能降低回源的概率。但是,如果持续不断地把资源分发到CDN边缘节点,那么由于其存储容量有限,一段时间之后必然达到存储容量的极限而导致服务质量下降。另外,随着时间的推移,有些热点数据逐渐变成冷而失去在CDN边缘节点存储的必要。所以就必须维持边缘节点的存储量不能超过一特定阈值,或者在达到某阈值后执行主动淘汰。另外,一些新业务的上线等场景又要求CDN边缘节点具有更强的主动淘汰的能力。

[0005] 一个最直接的想法是把OC机房(即CDN的边缘节点)里所有文件按最后访问时间排序,或按类似最近最少使用(Least Recently Used,LRU)的规则维护一个所有文件信息的LRU链表,执行淘汰时,删除最后访问时间最早的或访问链表LRU端的一个或若干个文件。

[0006] 但是,一直维护一个机房所有文件的信息所费内存空间比较大。而前一种方案每有一次用户访问请求都需要付出O(logn)的调整全局数据结构的额外代价;后一种方案虽然是O(1)的,但如若一旦程序意外崩溃或者有特殊需要人为重启,就要付出O(1)的时间来重建LRU链,但此时的LRU链只具有不能表征访问情况的临时信息,且会对淘汰过程产生滞后影响。

[0007] 为了避免上述方案的开销,一个粗略的替代实现方案是利用文件系统中的atime属性。atime表示某文件最后一次被访问的时间戳,如果开启atime,那么只需要遍历文件系统中的所有文件,并且在此过程中删除早于一个既定时刻的所有文件即可完成类似于LRU方式的淘汰。然而,而系统更新atime所带来的开销却是巨大的。因为系统每访问一次某个文件,就要对这个文件更新一个新的atime时间值。这里所说的访问,并不是从用户角度来看的打开一个文件,而是系统底层的每一次打开(open)和读取(read)等操作。每对文件进行一次读操作,都要引起一个对磁盘的写操作,即使我们要读的内容已经存在于内存的页缓存(Page Cache)中,还是要对磁盘进行一个写操作。这样引起的开销确实是巨大的,这些写操作会使磁盘更多的处于忙碌状态,这对系统性能(因为磁盘在完成一个写操作的时候会暂时阻止其他的写操作)以及电量消耗都是不利的。所以在实际互联网服务系统中,

atime通常是关闭的。虽然Linux2.6.20和2.6.24两个版本的内核分别针对atime做了一些延迟更新的策略，但其在提供海量文件满足海量并发用户的访问需求的系统中所能起到的作用甚微。

[0008] 所以就需要设计一种开销低、速度快，且不影响服务可用率、不引起服务质量下降的高效淘汰策略。

发明内容

[0009] 有鉴于此，有必要提供一种在内容分发网络中进行内容管理的方法、装置及系统，其硬件开销低、速度快，且不影响服务可用率、不引起服务质量下降。

[0010] 一种在内容分发网络中进行内容管理的方法，包括：监测边缘节点的已用存储空间是否超出预定的阈值，若是则执行以下步骤：从一个索引系统处获取需保护文件的集合U_1；对该边缘节点进行目录遍历以获取所有最后修改时间与当前时间之间相差超过预定值的文件的集合U_0；获取待删除文件的集合$U_2=U_0-U_1$；以及从该边缘节点内删除U_2内的每个文件。

[0011] 一种在内容分发网络中进行内容管理的装置，包括：监测模块，用于监测边缘节点的已用存储空间是否超出预定的阈值，若是则调用文件删除模块；该文件删除模块包括：第一获取单元，用于从一个索引系统处获取需保护文件的集合U_1；目录遍历单元，用于对该边缘节点进行目录遍历以获取所有最后修改时间与当前时间之间相差超过预定值的文件的集合U_0；第二获取单元，用于获取待删除文件的集合$U_2=U_0-U_1$；以及文件删除单元，用于从该边缘节点内删除U_2内的每个文件。

[0012] 一种在内容分发网络中进行内容管理的系统，包括：主控服务器及索引服务器；该主控服务器用于监测该内容分发网络的边缘节点的已用存储空间是否超出预定的阈值，若是则从该索引服务器处获取需保护文件的集合U1；对该边缘节点进行目录遍历以获取所有最后修改时间与当前时间之间相差超过预定值的文件的集合U0；获取待删除文件的集合U2=U0-U1；从该边缘节点内删除U2内的每个文件；该索引服务器用于根据请求返回该需保护文件的集合U1。

[0013] 上述的在内容分发网络中进行内容管理的方法、装置及系统具有以下优点：

[0014] 1.不需要在CDN的边缘节点维护资源的全局索引，更不需要按访问频度等信息来对资源进行排序，保持CDN边缘节点逻辑功能设计的最简单化，减少边缘节点的硬件资源消耗，从而提升边缘节点的运行效率。

[0015] 2.CDN边缘节点中的每台网络服务器和存储服务器均不需要开启atime，这样防止了每对文件进行一次读操作，都要引起一个对某磁盘块的写操作。不仅降低了不必要的磁盘写，提升了并发服务性能，而且降低了能耗。

[0016] 为让本发明的上述和其他目的、特征和优点能更明显易懂，下文特举较佳实施例，并配合所附图式，作详细说明如下。

附图说明

[0017] 图1所示为一个典型的CDN架构示意图。

[0018] 图2为一个视频网站的访问情况示意图。

[0019] 图3为一个视频网站的访问请求的累积分布函数。

[0020] 图4为若干影片票房收入和公映时间的关系示意图。

[0021] 图5至图7分别为三种不同的资源访问模式示意图。

[0022] 图8为本发明实施例提供的在内容分发网络中进行内容管理的方法及装置的运行环境示意图。

[0023] 图9为Unix/Linux文件系统中文件树的示意图。

[0024] 图10为索引节点的内容示意图。

[0025] 图11为目录项的结构示意图。

[0026] 图12为一个文件相关的i-节点、目录项和数据块的示意图。

[0027] 图13为文件路径解析过程的示意图。

[0028] 图14为仅包含两个媒体文件的目录项所在的块的示意图。

[0029] 图15为一种目录遍历的流程示意图。

[0030] 图16为图15的目录遍历的部分步骤流程示意图。

[0031] 图17为另一种目录遍历的流程示意图。

[0032] 图18为第一实施例提供的在内容分发网络中进行内容管理的方法流程图。

[0033] 图19为第二实施例提供的在内容分发网络中进行内容管理的方法流程图。

[0034] 图20为第三实施例提供的在内容分发网络中进行内容管理的方法流程图。

[0035] 图21为文件删除操作时目录项的变化示意图。

[0036] 图22为第四实施例提供的在内容分发网络中进行内容管理的方法流程图。

[0037] 图23至图27分别为全部资源、UGC资源、新闻、音乐、及影视等各类资源的302回源统计示意图。

[0038] 图28为第五实施例提供的在内容分发网络中进行内容管理的装置的结构框图。

[0039] 图29为代理服务器的结构框图。

[0040] 图30为第六实施例提供的在内容分发网络中进行内容管理的系统的结构框图。

具体实施方式

[0041] 为更进一步阐述本发明为实现预定发明目的所采取的技术手段及功效，以下结合附图及较佳实施例，对依据本发明的具体实施方式、结构、特征及其功效，详细说明如后。

[0042] 幂律分布

[0043] 1999年9月，Barabasi小组在《自然》上发表了一篇通讯(Albert,R.,Jeong,H.and Barabási,A.L..″Diameter of the World-Wide Web.″Nature(London)401,no.6749(1999):130.)，指出互联网的出度分布和入度分布都与正态分布有很大的不同，而是服从幂律分布。更为重要的是，文中揭示了幂律分布产生的两个机理。不同于ER随机图(Erdős-Rényirandom graph)，实际网络具备两个重要特性：第一是增长(Growth)特性，即网络的规模是不断扩大的。第二是优先连接(Preferential attachment)特性，即新的节点更倾向于与那些具有较高连接度的节点相连接，这种现象也成为“富者更富”或者“马太效应(Matthew effect)”。一个月之后，Barabasi小组又在《科学》上发表文章(Albert-László Barabási&Réka Albert(October1999).″Emergence of scaling in random networks″.Science286(5439):509-512.)指出，包括电影演员网络和电力网络在

内的其他许多实际网络的度分布也都服从与泊松分布有很大差异的幂律分布。

[0044] 中国科学技术大学的汪秉宏等人的实证研究总结出:“通过各种不同的数据收集方法,人们的研究涉及市场交易、网站浏览、电影点播、欣赏网络音乐、手机通讯、在游戏及虚拟社区中的行为、计算机指令的使用行为等,包含了商业行为、娱乐行为、日常使用习惯等众多的人类行为,在这些行为中,都较好地服从幂律分布。”(请参见周涛,肖伟科,任捷,汪秉宏网络集团度的幂律分布复杂系统与复杂性科学2007年02期;樊超,郭进利,韩筱璞,汪秉宏人类行为动力学研究综述复杂系统与复杂性科学2011年02期)

[0045] 现以某视频网站为例,统计了某一周时间内该视频网站的一个子系统在某城市的媒体文件的访问情况,如图2所示。进一步计算表明,媒体文件的访问请求次数和其排名在双对数坐标下的皮尔逊线性相关系数为0.93,可近似认为其服从Zipf分布。

[0046] 图3为访问请求的累积分布函数,从中可以看出,访问频数排名在前20%的媒体文件贡献了超过86%的访问量,此亦即符合“二八”法则。

[0047] 进一步统计最近一个月内该视频网站全部子平台全量产生流量的资源的每日访问计数,发现:超过56%的媒体资源虽然在过去30天内有访问请求,但每日的请求次数均不超过1;超过77%的媒体资源的每日请求次数均不超过3。由此看见,访问呈现出显著的长尾分布,但大部分文件属于冷门资源。

[0048] 指数衰减

[0049] 图4显示了互联网电影资料库(Internet Movie Database,IMDB)数据库中的若干影片票房收入和公映时间的关系。与此类似,视频网站等内容服务平台上的资源访问也呈现出相近情况。一个媒体资源通常在引入13天之内达到访问最高值,此后便以近似指数函数的方式衰减。

[0050] 虽然各种不同类别的媒体资源的衰减参数各不相同,甚至个别资源会出现几个高峰,但深入统计人为划分出的状态之间的转移情况后发现:如果一个资源文件在第i-3日及之前有访问,第i-2和第i-1日无访问,则其在第i日仍无访问的概率超过0.75;在过去5日之内无访问而在下一日有访问的概率则小于0.01。这个结果表明:一旦一个资源变冷,则接下来出现访问的可能性很小。

[0051] 图5、6、7显示了三种不同的资源访问模式。其中,大部分新闻视频和绝大部分用户生成(User Generated Content,UGC)视频都符合图5的模式,即视频在引入1-3天内达到访问峰值,此后骤减,再在之后以指数方式衰减直零。一部分新闻视频和大部分不太热门的影视视频符合图6的访问模式,总体以指数方式平缓衰退,但中间由于某些原因会形成若干次局部峰值。一小部分极热资源(无论编辑上传资源还是UGC资源),会出现如图7所示振幅较大但访问频度始终维持在较高值的现象。

[0052] 但无论哪种情况,一旦某个资源在某时访问频度降低到一个足够小的值,那么其后几乎必然满足指数衰减;一旦某个资源连续几日访问频数为0,那么其后一日有访问的概率极小。

[0053] 以上统计亦适用于图片、静态资源等平台,只是统计出来的参数有所差异。

[0054] 图8为本发明实施例提供的在内容分发网络中进行内容管理的方法及装置的运行环境示意图。边缘节点601为一个内容分发网络的一个边缘节点,其内包括网络服务器602、主控服务器603、存储服务器604、以及缓存服务器606。

[0055] 网络服务器602用于边缘节点601中面向用户的网络接入，即接收用户的请求并返回相应的资源。在一个实例中，网络服务器的配置是：1个4核CPU，8G内存，1*1TSATA硬盘。主控服务器603硬件结构上可与网络服务器602相同，其功能亦类似，其不同之处在于还承担一些边缘节点601的管理功能，例如本发明实施例提供的在内容分发网络中进行内容管理的方法即可由主控服务器603触发执行。

[0056] 存储服务器604用于边缘节点601中的数据存储，其可挂接多个存储器605如硬盘。在一个实施例中，存储服务器604的配置是：2个4核CPU，16G内存，主机总线适配器，12*1T SATA硬盘(605)。

[0057] 缓存服务器606用于边缘节点601中的热点资源缓存。在一个实例中，缓存服务器606的配置是：2个4核CPU，36G内存，4*500GB SATA硬盘，512G*2(华为™)或160G*8(因特尔™)固态硬盘。

[0058] 网络服务器602以及主控服务器603均可通过网络文件系统管理存储服务器604以及缓存服务器606内存储的文件。

[0059] 边缘节点601通过互联网与代理服务器607相连，代理服务器607通过企业网与索引系统608相连。代理服务器607可以基于以下技术构建：一、双向TCP(Transmission Control Protocol，传输控制协议)代理；二、HTTP上的CGI(Common Gateway Interface，通用网关接口)代理；三、基于ASN.1(Abstract Syntax Notation One，抽象语法标记)协议框架的代理。

[0060] 为了保证安全，代理服务器607与边缘节点601或者索引系统608的通讯采用加密的方式进行处理。例如，如果采用前两者，传输时须携带有参数校验字段以防止黑客注入，保证安全。生成校验字段的一个实施例是基于块的对称加密算法(如Extended Tiny Encryption Algorithm，XTEA)，或者基于流的等对称加密算法(如chacha)。而如果采用第三者，因为ASN.1本身就是一种可灵活实现编码/解码(加密/解密)的协议框架，所以不需要额外实现校验字段。

[0061] 索引系统608内包括请求记录数据库609。请求记录数据库用于存储所有边缘节点601内文件的访问记录、用户生成的文件以及需要永久保存的文件等信息。在一个实例中，请求记录数据库609包括以下几个子库：DBhot、DBpub、以及DBperm。

[0062] DBhot记录文件的访问记录，至少包括文件名、所属频道、访问次数等项，每天一个数据表。由于对全局访问的统计有稍许时延，且需要归并各边缘节点和区域中心的访问记录日志，所以需要在每天访问的低谷时刻(如凌晨4点)进行数据的校验计算和重新入库。

[0063] DBpub记录每天由编辑新上传但还没有发布推广的文件，同DBhot，一日一数据表。

[0064] DBperm通常只包含一个数据具有时变特征的数据表，记录该段时间需要保护防止删除的文件信息。

[0065] 上述的DBhot数据库，是由一个内容分发网络内的多个(如几十个)边缘节点、多个(如十个左右)区域中心/数据中心节点，根据服务用户的服务器产生的用户访问日志，每间隔一段时间(如15分钟)上报一次，再合并，而每天凌晨把前一天的上报数据统计归并，再重新入库的。故而这个在索引系统608内的“全量访问记录”，是缺少当天访问的数据的，或者至少当天访问的记录是不准、有延迟的。

[0066] CDN内文件的存储

[0067] 可以理解，要在CDN中发布的文件需要首先分发至各个边缘节点内的存储服务器中。在实际的分发过程中，文件的发布可以是由专门的分发系统完成。换言之，每个边缘节点内均包括上述分发系统的客户端，用于接收来自发布源推送的文件。进一步地，在接收到推送的文件后，将其存储至存储服务器中的某个目录下。

[0068] 在一个实例中，分发系统根据以下策略决定文件将要存储的路径：

[0069] /DATAx/(CHANNEL/)y/z/filename

[0070] 上面路径中，CHANNEL为频道名，DATAx为索引为x的磁盘的根路径，filename为要存储的文件的文件名。可以理解，频道名并不是必须的，或者频道还可以具有多级目录结构，本领域普通技术人员可以根据实际需要任意采用合适的目录结构。

[0071] x、y、z满足以下条件：

[0072] $$\begin{cases} x = \mathrm{DiskID}, & \mathrm{DiskID} \le 256/N; \\ y = hash_1(filename)\ mod\ A, & 96 \le A \le 128; \\ z = hash_2(filename)\ mod\ B, & 96 \le B \le 128. \end{cases}$$

[0073] 上式中，N表示一个边缘节点内存储服务器的数目，DiskID取值不大于256/N是由于NFS文件系统的最大挂载磁盘数是256。而一般一个边缘节点内每个存储服务器下连接的存储器数是相同的。例如，在一个实例中，一个边缘节点内包括20台存储服务器，每个存储服务器接入12个存储器，总存储器的数目为240。

[0074] $hash_1$和$hash_2$为能保证散布均匀的两个哈希函数。在一个实例中，可选用BKDR、BobJenkins或SuPerHash等字符串哈希函数。

[0075] 至于计算y、z值的公式中模数的范围，是为了提升目录遍历的性能，其具体原理请参见后续NFS目录遍历部分。

[0076] 如图8所示，每个网络服务器602(或者主控服务器603)均可通过NFS管理存储服务器内存储的文件。其具体的方式是，通过NFS把若干台存储服务器上的不超过256块的存储器挂载到自己的一个虚拟目录k下。其上看到的路径如下：

[0077] /data/k/CHANNEL/y/z/filename

[0078] 其中，k≤10000，且k与x之间建立一个均匀的多对一的映射。此映射关系的设计可参考一致性哈希技术(David Karger,Eric Lehman,Tom Leighton,Rina Panigrahy,Matthew Levine,Daniel Lewin.Consistent hashing and random trees:distributed caching protocols for relieving hot spots on the World Wide Web[C].Proceedings of the twenty-ninth annual ACM symposium on Theory of computing.STOC'97,New York,NY,USA:ACM,1997,654-663.http://doi.acm.org/10.1145/258533.258660，此处一并引入作为参考)或者其引入虚拟节点概念的改进(Giuseppe DeCandia,Deniz Hastorun,Madan Jampani,Gunavardhan Kakulapati,Avinash Lakshman,Alex Pilchin,Swaminathan Sivasubramanian,Peter Vosshall,Werner Vogels.Dynamo:amazon's highly available key-value store[C].Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles.SOSP'07,New York,NY,USA:ACM,2007,205-220.http://doi.acm.org/10.1145/1294261.1294281，此处一并引入作为参考)。

[0079] 经过以上的设计，分发系统分发到CDN系统中的每一个文件都将存储到一个确定的逻辑路径下，而且因为文件数量巨大及哈希的均匀散步效果，文件目录树可形成一个高度有限、有效层次为2的平衡树。这样就使得在遍历时可做基于数据划分的并行化，而且各计算单元负载均衡，另外目录遍历时栈的逻辑深度亦可控。从而可提升目录遍历时的性能。

[0080] 在以上的存储路径设计中，在存储服务器中采用的是2级哈希均匀散布，而通过NFS映射后，文件的路径是3级哈希均匀散布。然而，本发明实施例并不限于上述的实例，例如，还可以采用更多级数的哈希均匀散布。

[0081] NFS上的并行目录遍历

[0082] 计算机系统中CPU与输入/输出系统(I/O)之间持续扩大的性能差距影响着并行程序设计。磁盘传输带宽的改善速度跟不上CPU计算能力的增长，数据访问延迟的改善更是严重地滞后。为此，Linux等操作系统采取了预取技术来提升磁盘I/O性能，可以有效地将来自应用程序的同步的小I/O转化为异步的大I/O，从而减少访问延迟，并促成I/O的并行化。

[0083] 但是，单磁盘的I/O操作如果并发的话，操作系统内核的互斥锁依然会保证I/O操作的串行性。另外，虽然Linux系统会在并发I/O请求中有效地识别和处理顺序流，但是对磁盘上所有文件的遍历操作很容易让缓存失效。例如，Linux操作系统内核有一个数据结构dcache用来缓存此前解析出的目录项，它采用LRU策略来管理，但是如果初始LRU链表为空的话，一个超过k+1长度的遍历会使长度为k的dcache的命中率为0。

[0084] 此外，边缘节点上要删除的文件逻辑上都是冷文件，另一方面，统计数据也表明，边缘节点上存储的大部分文件也较冷；但显然dcache的LRU链表上都是热文件的目录项。

[0085] 因此，dcache在目录遍历这种操作中，dcache不命中的概率几乎为100%。

[0086] 再者，并发I/O请求带来的磁头臂的来回寻址又会加剧性能的恶化。事实也表明，很多文件系统，在其上遍历的性能在超过2个线程之后反而会随着线程数的增加而降低。

[0087] 因此，在NFS上进行文件(目录)遍历，单个磁盘上I/O的操作应选择串行，而各磁盘间的遍历应变为并行。

[0088] 在常见的Unix/Linux文件系统中，索引节点(又称i-节点或inode)、目录项(dirent或directory entry)、数据块(data block或block)构成了一颗庞大的文件树，如图9所示。

[0089] inode表示文件除文件名外的所有元信息，如mtime(最后修改时间)等项，并提供了到数据块的多级索引方式，如图10所示。

[0090] dirent表示文件/目录名和inode的对应关系。在Linux2.6.4之后，增加了一个字节的表示文件/目录类型的域，见图11。

[0091] block是数据块，磁盘读写即以block为单位，通常大小为4K字节。值得注意的是：一个inode或一个dirent，也必然处在某个block中。

[0092] 图12示出了在一个磁盘的线性地址区域中，文件xxx.mp4相关的i-节点、目录项和数据块直接的关系。

[0093] 如果某文件(或其祖先目录)的目录项不在目录项缓存dcache中，那么对文件完整路径的解析将会是一件代价繁重的工作，会造成多次的磁盘访问，如图13所示。而如前文分析，本发明涉及的目录遍历需求，即属于这种绝大多数目录项都不在目录项缓存dcache中的场景。

[0094] 可以理解，目录遍历实质是个无环图遍历过程，所以可有深度优先和广度优先两种策略可以选择。在绝大多数文件系统中，目录项也是一个文件，那么访问目录下某个文件时，需要逐个读取目录数据中的目录项并与目标进行匹配获得文件的inode号。

[0095] 当目录下文件数较少时，则在目录下查找文件需要的数据块也较少，而且是直接索引；当目录下文件数比较多的时候，需要访问的块数会更快的增加，而且后面得到存储数据的物理块号需要多级索引，这也是在目录下不应放太多文件的原因。如果将拥有很多文件的目录均分成多个子目录，多一级目录会多一次(或多次，具体依赖于子目录下文件数量)磁盘块访问，但在子目录中查找文件的磁盘访问开销会小很多。

[0096] 在Linux操作系统中，由于目录项的长度均为4的整数倍，而文件名本身就是文件全文或一定规则的若干分片哈希得来，即使考虑分片的后缀，也基本可以看成是定长的。实际场景中计算得到目录项长度通常不超过32。通常块大小为4KB，考虑到两个优化指标：1.使目录项能够在读取inode之后的第一个数据块中得到；2.使目录分级高度最小，即目录项所在的块被充分利用。最终计算得到取值96-128之间的数较为合适。为考虑可扩展性，在一个实例中，可取较小的100。图14示出了仅包含两个媒体文件的目录项所在的块(可结合图11来看)。

[0097] 在本发明对文件目录结构的设计中，由于末级目录中的文件数目较少，所以采用广度优先遍历得到的效率会稍高一些，但从全文件系统的遍历来看，与深度优先差别不是特别大。

[0098] 目录遍历也有递归和非递归两种方式可以选择。一方面，递归算法可以简化算法设计上的复杂性，但是在运行时会占用大量的栈(stack)空间，分配栈空间的方法也比分配堆(heap)空间的方法耗费要大很多，同时也会产生大量的函数调用代码，耗费大量额外时间用以维护中间现场状态，且有栈溢出的风险。但根据上文分析，目录遍历这种较为密集的I/O操作获得并行反倒适得其反，所以应该采用非递归方可获得更好的性能。

[0099] 参阅图15，在一个实例中，在主控服务器603中对一个边缘节点进行目录遍历包括以下步骤：

[0100] 步骤S1501，为边缘节点内的每个存储器创建一个目录遍历进程；

[0101] 步骤S1502，等待上述目录遍历进程的遍历结果；以及

[0102] 步骤S1503，归并所有目录遍历进程的遍历结果。

[0103] 更加详细地，采用类C++伪代码描述上述过程如下：

[0104]
```
for(i=0;i<TOTAL_DISK&&!inf.eof();i++)
    {
```

[0105]
```
        getline(inf, filename);
        {
```

```
                StartingScanDirs.push_back(filename);
                if(StartingScanDirs[count].c_str() == "")
                    break;
                int *arg;
                arg = (int*)malloc(sizeof(int));
                *arg = count;
[0106]          //创建一个线程扫描该目录（步骤S1501）
                result = pthread_create(&thread[count], NULL,
                    ScanDirectory, (void*)arg);
                count++;
            }
        }
```

[0107] 上述代码中，TOTAL_DISK表示存储器的数目，filename表示存储待遍历目录的文件的文件名。由上述可知，每个存储器对应于一个唯一的目录。而如前所述，在主控服务器603中，可通过NFS将存储器的要目录映射至多个虚拟目录中。因此，在上述代码执行之前，还可包括获取每个存储器唯一目录的步骤。

[0108] //等待所有线程扫描结束(步骤S1502)

[0109] for(j=0;j<count;j++)

[0110] pthread_join(thread[j],NULL);

[0111] fpAllFile=fopen("filelist.txt","w");

[0112] //将各目录扫描到的合并遍历输出(步骤S1503)

[0113] for(i=0;i<count;i++)

```
        {
            for(itSetNode = ScanResults[i].begin(); itSetNode !=
[0114]          ScanResults[i].end(); itSetNode++)
            fprintf(fpAllFile,  "%s\n",  (*itSetNode).c_str());
```

[0115] }

[0116] fclose(fpAllFile);

[0117] 参阅图16，在一个实例中，在目录遍历进程中可执行以下步骤：

[0118] S1601，将当前进程绑定至特定的处理器核心；

[0119] S1602，初始化一个目录栈；

[0120] S1603，起始目录入栈；

[0121] S1604，判定目录栈是否为空，若为空结束遍历，否则执行以下步骤直至目录堆栈为空：

[0122] S1605，获取目录栈顶部的目录并使最后入栈的目录出栈；

[0123] S1606，读取该目录栈顶部的目录的所有子项目；以用

[0124] S1607，若子项目为目录则置入该目录栈，否则记录在扫描结果中插入一条文件记录。

[0125] 更加详细地，采用类C++伪代码描述图16所示的过程如下：

[0126] void*ScanDirectory(void*arg)

[0127]

```
{
    int id;
    id =*(int*) arg; //解析出进程 ID
    CPU_ZERO(&cpu[id]);
    //线程按序号绑定到后面的逻辑 CPU，但避免绑定至前两个处理
        器核心（步骤 S1601）
    CPU_SET(id%(total_cpu_num-2) + 2,  &cpu[id]);
    if(pthread_setaffinity_np(pthread_self(),  sizeof(
        cpu_set_t),  &cpu[id]) < 0)
            perror("pthread_setaffinity_np");
    DIR* handle;
    //初始化一个目录栈（步骤 S1602）
    stack<string> FolderSet;
    string CurrentFile,  CurrentFolder;
    struct stat buf;
    struct dirent* direntp;
```

[0128]

```
    if(StartingScanDirs[id].empty())
    {
        return NULL;
    }
```

[0129] //起始目录进栈(步骤S1603)

[0130] FolderSet.push(StartingScanDirs[id]);

[0131] //步骤S1604

[0132]
```
while(!FolderSet.empty())
{
    //步骤 S1605，获取目录栈顶部的目录并使最后入栈的目录
    出栈;
    CurrentFolder = FolderSet.top();
    FolderSet.pop();
    //步骤 S1606，读取该目录栈顶部的目录的所有子项目
    if((handle = opendir(CurrentFolder.c_str())) == NULL)
        continue;
```

[0133]
```
    else
    {
        while((direntp = readdir(handle)) != NULL)
        {
            //步骤 S1607，若子项目为目录则置入该目录栈，
            否则记录在扫描结果中插入一条文件记录
            if((direntp->d_name[0]== '.' && (direntp->
                d_name[1]== '\0' || (direntp->d_name[1]=='.
```

```
                    ' && direntp->d_name[2]== '\0'))))
                        continue;
                    CurrentFile = CurrentFolder + "/" + direntp->
                        d_name;
                    if(stat(CurrentFile.c_str(), &buf) != 0)
                        continue;
                    if(S_ISDIR(buf.st_mode))
                    {
                        FolderSet.push(CurrentFile);
                    }
                    else
                    {
                        string s = buf.st_mtime;
                        ScanResults[id].insert(pair < int,  string >(
                        direntp->d_ino,  CurrentFile + " " + s));
                    }
                }
            }
            closedir(handle);
        }
    }
```

[0134]

[0135] 上述的目录遍历过程可以适用于各种存储器如磁盘、固态存储器(Solid State Disk,SSD)光盘等等。但可以理解,不同各类的存储器具有不同的特性。目录遍历过程还可以针对不同种类的存储器做进一步优化。

[0136] 例如,参阅图17,在另一个实例中,对一个磁盘进行目录遍历可包括以下步骤:

[0137] S1701,初始化文件队列及目录队列,根目录入目录队列;

[0138] S1702,判断目录队列是否为空,若为空则进行步骤S1703,否则执行遍历子步骤;

[0139] 步骤S1703,判断文件队列是否为空,若为空则结束遍历,否则进行步骤S1704。

[0140] 步骤S1704,文件队列出列,在遍历结果中插入出列的文件,并返回步骤S1703。

[0141] 遍历子步骤包括:

[0142] S1705,目录队列出列;

[0143] S1706,获取出列的目录的子项目列表,对子项目列表排序;

[0144] S1707,对于每个子项目,若为目录则入目录队列,若为文件则入文件队列,并返回步骤S1702。

[0145] 更加详细地,采用类C++伪代码描述上述过程如下:

```
scan(root_path)
{
    //步骤 S1701，初始化文件队列及目录队列
    FileQueue, DirQuene;
    dirent d(root_path);
    //根目录入目录队列
    DirQuene.push(d);
```

[0146]

```
    FilePosMap; //用于存储扫描结果
    //步骤 S1702，判断目录队列是否为空
    while(!DirQuene.empty())
    {
        //目录队列出列
        d=DirQuene.pop();
        if(d.isdir())
    FileQuene.push(d);
else
{
    if(d.d_name!=".")
```

[0147]

```
        path=d.parent + '/' + d.d_name;
    else
        path=d.parent;
}
```

[0148] //获取出列的目录的子项目列表,对子项目列表排序

[0149] read_dir(path,dir_list);

[0150] sort(dir_list);

[0151] d.setdir();

[0152] it=dir_list.begin();

[0153] //对于每个子项目,若为目录则入目录队列,若为文件则入文件队列

[0154]
```
while(it!=dir_list.end())
    {
        if(it-d_name!="." or "..")
        {
            if(it->isdir())
                DirQuene.push(*it);
[0155]      else
                FileQuene.push(*it);
        }
        ++it;
    }
}
```

[0156]
```
//判断文件队列是否为空
    while(!FileQuene.empty())
    {
        //文件队列出列，在遍历结果中插入出列的文件
[0157]  d=FileQuene.pop();
        FilePosMap.insert(d.d_ino,d.full_path());
    }
}
```

[0158] 上述用于获取某个目录的子项目的函数read_dir伪代码如下：

[0159]
```
read_dir(path,dir_list)
```

[0160]

```
{
    fd=open(path);
    written=0, offset=0;
    buf[BLOCK_SIZE];
    //通过系统调用尝试读取 BLOCK_SIZE 长度的目录项到 dirp 所指向的目录项结构体中
    written=syscall(SYS_getdents, fd, dirp, BLOCK_SIZE);
    while(written>0)
    {
        if(offset<written)
        {
            os_dirent *s;
            s=buf+offset;
            dirent d;
            d.d_ino=s->d_ino;
            d.d_name=s->d_name;
            d.d_type=*(s+s->d_reclen-1);
            d.parent=path;
            //在返回结果中加入读取的目录项
                dir_list.push_back(d);
                offset += s->d_reclen;
            }
```

[0161]

```
            written=syscall(SYS_getdents, fd, dirp, BLOCK_SIZE);
        }
    }
```

[0162] 大量的实验也表明，在多数对文件元信息进行操作的程序中，stat()调用会消耗大量时间，而标准Unix API中，readdir()返回的是文件名的字典序，而和位置序无关，这样更加剧了stat()调用的消耗，使得在获取文件元信息的过程中，磁头臂不停地来回寻址，即使是找同一目录下不同文件的信息也如此。

[0163] 在上述的实例中，为充分提升性能，重写了自定义的read_dir()函数，先尽可能读取整块的所有目录项，再按inode排序，这样目录项信息的查找也变成了有序的，充分提高了性能。统计结果显示，在磁盘中进行目录遍历时，图17所示的方法相比于图15所示的方法效率可以提升30%至300%。

[0164] 在上述的遍历过程中省略了文件mtime是否符合备淘汰条件的判断。但可以理解，本领域普通技术人员可自行添加相应的判定过程。

[0165] 第一实施例

[0166] 图18为第一实施例提供的在内容分发网络中进行内容管理的方法，其例如可由图8所示的主控服务器603执行。上述的方法包括以下步骤：

[0167] 步骤S110、监测边缘节点的已用存储空间是否超出预定的阈值，若是则执行以下步骤S120至步骤S150。步骤S110例如可由部署在主控服务器603内的代理程序执行。如上所述，主控服务器603可以通过NFS将存储服务器604内的存储器挂载至一个虚拟目录内，然后通过查询各目录的属性，即可获取存储服务器604的存储空间占用情况，当已用存储空间超过预定的阈值如90%时，即可触发删除程序，即执行步骤S120至步骤S150。

[0168] 步骤S120、从一个索引系统处获取需保护文件的集合U_1。参阅图8，索引系统608的数据库内可存储所有边缘节点的访问记录，并可存储需要保护的文件的相关信息。在一个实例中，U_1可包括：在过去第一预定时间(t1)内至少存在某天访问次数大于1的文件集合、过去第二预定时间(t2)内上传但未发布的文件的集合、以及所有需要永久保存的文件的集合。

[0169] 由于索引系统608可位于企业网内部，而所有边缘节点均直接接入互联网。为了保证网络安全，主控服务器603与索引系统608之间的网络通讯可以通过代理服务器进行。因此，步骤S120具体地可包括以下步骤：

[0170] 主控服务器603将该淘汰预警信号发送至代理服务器607；

[0171] 代理服务器607获取淘汰参数t1、t2、及t3；淘汰参数t1、t2、及t3可以是由用户输入的，还可以是通过读取默认的配置信息得到。

[0172] 代理服务器607向索引系统608发送获取文件请求以及淘汰参数t1、t2。

[0173] 索引系统608根据获取文件请求以及淘汰参数t1、t2返回上述的需要保护的文件集合U_1。

[0174] 代理服务器607接收索引系统608返回的需保护文件的集合U_1并将U_1返回至主控服务器603。

[0175] 步骤S130、对该边缘节点进行目录遍历以获取所有最后修改时间与当前时间之间相差超过预定值的文件的集合U_0。

[0176] 目录遍历的具体过程可参见上述的上的NFS上的并行目录遍历部分，在此不再赘述。

[0177] 步骤S130可以是根据目录遍历请求进行的，上述的目录遍历请求可以是代理服务器607发出的。例如，在代理服务器607接收到上述的淘汰预警信号后，即可向主控服务器603发送目录遍历请求以及参数t3。

[0178] 步骤S130中的预定值可为上述的参数t3。在一个实例中，上述的t3可为大于1的整数，例如30天。

[0179] 步骤S140、获取待删除文件的集合$U_2=U_0-U_1$。

[0180] 步骤S150、从该边缘节点内删除U_2内的每个文件。

[0181] 步骤S140及步骤S150例如可由主控服务器603进行。

[0182] 可以理解，如图15所示，步骤S110与步骤S150实质上可以是并行进行的，亦即，步骤S110与步骤S150可以由两个独立的进程分别执行。进一步地，步骤S110、步骤S120至步骤S140、以及步骤S150可以分别由独立的进程执行。例如，步骤S110由一个监控进程执行，步骤S120至步骤S140由一个待删除文件获取进程执行，而步骤S150由一个文件删除进程执行，这些进程分别独立运行，但可接受其他进程传输的参数。例如，文件删除进程接收待删除文件获取进程发送的待删除文件的的集合作为参数。

[0183] 此外，可以理解，在步骤S150中，若主控服务器603在删除文件时失败，其原因可能是存储器故障或者网络故障。因此，在删除失败时，还可向对应的存储服务器604发出文件删除请求，由存储服务器604执行文件删除操作，以避免网络故障导致的文件删除失败。

[0184] 本实施例的在内容分发网络中进行内容管理的方法具有以下优点：

[0185] 1.不需要在CDN的边缘节点维护资源的全局索引，更不需要按访问频度等信息来对资源进行排序，保持CDN边缘节点逻辑功能设计的最简单化，减少边缘节点的硬件资源消耗，从而提升边缘节点的运行效率。

[0186] 2.CDN边缘节点中的每台网络服务器和存储服务器均不需要开启atime，这样防止了每对文件进行一次读操作，都要引起一个对某磁盘块的写操作。不仅降低了不必要的磁盘写，提升了并发服务性能，而且降低了能耗。

[0187] 3.通过调整几个简单而又含义明确的参数，可以灵活控制需要淘汰的文件数量。

[0188] 第二实施例

[0189] 图19为第二实施例提供的在内容分发网络中进行内容管理的方法的流程图，其与第一实施例相似，不同之处在于，在步骤S120后还进行以下步骤：检测是否有损坏的存储服务器，若是则执行步骤S210，否则执行步骤S130a。

[0190] 步骤S130a，通过NFS对该边缘节点进行目录遍历以获取所有最后修改时间与当前时间之间相差超过预定值的文件的集合U_0。

[0191] 如上所述，主控服务器603可通过NFS将不同存储服务器604挂载至一个虚拟目录下，因此遍历可直接在主控服务器603中进行。

[0192] 步骤S210，由损坏的存储器对应的存储服务器604进行目录遍历，并将目录遍历结果上传至主控服务器603。从而，主控服务器603可归并目录遍历结果，以得到上述的集合U_0。

[0193] 根据本实施例的在内容分发网络中进行内容管理的方法，可以针对存储器损坏做出更好的处理。

[0194] 第三实施例

[0195] 图20为第三实施例提供的在内容分发网络中进行内容管理的方法的流程图，其与第一实施例相似，不同之处在于，在步骤S140与步骤S150之间还包括：步骤S310，对待删除的文件的集合U1中的文件按一定顺序排序，例如其inode、本文件inode所在块号、本文件目录项所在块号等进行排序。如此可使在进行文件删除操作时磁头尽量沿一个方向行进，避免寻道时磁头臂耗时的机械运动，从而可加速文件删除的速度。

[0196] 一并参阅图14及图21，文件被删除时对目录项的操作是，将其前面的目录项的记录长度值增大到指向被删除文件的后一个目录项的起始处。例如图21所示的状态相比于图14所示的状态，删除了文件bbb.flv，则前一个目录项(对应于aaa.mp4长度从32增加到4072(32+4040))。

[0197] Ext3文件系统在删除文件时，不清除目录项中的i-节点号，但将i-节点中的文件大小设置为0并清除i-节点中的块指针以及间接块中的块指针；而Ext2文件系统正好相反，清除目录项中的i-节点号但不清除i-节点中的文件占用块数及块指针和间接块指针。

[0198] 根据所做的大量实验，inode号-物理块号-逻辑块号均呈现线性关系或呈现极高的线性相关度。且在众多Linux文件系统中，

[0199] inode(a)<inode(b)->block(inode(a))≤block(inode(b))

[0200] 这样便可根据这些信息，适当调度I/O，减少磁头臂来回寻址的时间。

[0201] 第四实施例

[0202] 图18为第四实施例提供的在内容分发网络中进行内容管理的方法的流程图，其与第一实施例相似，不同之处在于，在步骤S120与步骤S130之间还包括步骤S410，将进行目录遍历的进程绑定在特定的处理器核心上。

[0203] 上述的特定的处理器核心是指逻辑CPU。在一个实例中，对于一个进程，按以下公式确定其逻辑CPU的序号：id%(处理器核心总数-2)+2。其中，id为当前的进程标识(ID)。

[0204] 可以看出，在上述公式，逻辑CPU的序号永远不可以能为1及2，亦即目录遍历的进程永远不会绑定至序号为1与2的处理器核心上。这么做的目的是避免将目录遍历的进程绑定至前两个处理器核心上，从而可避免目录遍历进程影响服务器中已有的处理网络连接、统计测速等进程的正常服务，保证合理的响应时间。

[0205] 而将录遍历的进程绑定在特定的处理器核心上是为了避免遍历不同磁盘的线程在上下文切换中损失缓存和流水线断开的开销。

[0206] 根据本实施例的在内容分发网络中进行内容管理的方法，可提升目录遍历过程的性能，减少目录遍历操作所消耗的时间。

[0207] 可以理解，以上各实施例仅为上述的在内容分发网络中进行内容管理的方法的示例性说明，并非用以限制其范围。本领域普通技术人员可结合实际情形采用、组合上述的各实施例得到不同的技术方案。

[0208] 基于上述的各种优点，根据本发明实施例提供的在内容分发网络中进行内容管理的方法可以极大提升内容管理尤其是文件主动删除的效率。例如，若采用传统的脚本程序进行内容删除，由于只能为串行，一个边缘节点内的目录遍历需要约10小时，文件删除则需要超过一个白天的时间，如果取t1为90天，t2为30天，那么获取需保护的文件的集合的时间约为5分钟，这样总共需要的时间往往超过一天。而采用本发明实施例提供的在内容分发网络中进行内容管理的方法，目录遍历耗时2至4分钟，文件删除耗时4至10分钟，由于获取需要保护的文件以及目录遍历两个步骤可以并行，即使算上网络传输延时，总的消耗时间也不超过15分钟。由于消耗的时间很短，选择一个用户访问量最低的时段，即可将地文件删除操作对用户访问的影响降低到最小程度。

[0209] 在一个实际操作实例中，在某一天8:30在一个边缘节点上执行文件删除操作，淘汰约1/5的存储量，全部资源、UGC资源、新闻、音乐、及影视等各类资源的302回源次数(每1

分钟统计)和上一日同时刻比较分别如图19至图23所示。其中,深色的线条表示上一日的数据,浅色的线条表示进行文件删除操作后的数据。

[0210] 从图19至图23可以看出,只有新闻资源产生了在可控范围内的比较明显的回源请求,仔细调查原因如下:

[0211] 1、当天发生了诸多热闹新闻事件,而资源尚未来得及分发至各边缘节点;

[0212] 2、一些和最近事件语义相关的其他很早的新闻,被相关性推荐系统推荐而引发了诸多用户请求。

[0213] 可以理解,这两个问题都不是文件删除操作时所能够考虑到的问题。综上所述,根据本发明实施例的在内容分发网络中进行内容管理的方法,可以在不明显增加回源数的前提下快速删除大量的文件,可有效维护边缘节点的正常运转。

[0214] 第五实施例

[0215] 第五实施例提供一种在内容分发网络中进行内容管理的装置,参阅图24,上述装置包括:监测模块10以及文件删除模块20。

[0216] 监测模块10用于监测边缘节点的已用存储空间是否超出预定的阈值,若是则调用文件删除模块20。

[0217] 文件删除模块20包括:

[0218] 第一获取单元21,用于从一个索引系统处获取需保护文件的集合U_1;

[0219] 目录遍历单元22,用于对该边缘节点进行目录遍历以获取所有最后修改时间与当前时间之间相差超过预定值的文件的集合U_0。

[0220] 目录遍历单元22对边缘节点进行目录遍历时可采用广度优先遍历,以及非递归遍历。此外,对于同一个存储器内的目录,其遍历过程串行进行,而不同的存储器的遍历过程并行进行。对边缘节点进行目录遍历的过程中还将遍历进程绑定至特定的处理器核心。而进行目录遍历的结果可按照索引节点进行排序。

[0221] 第二获取单元23,用于获取待删除文件的集合$U_2=U_0-U_1$;以及

[0222] 文件删除单元24,用于从该边缘节点内删除U_2内的每个文件。

[0223] 在一个实例中,上述需保护文件的集合U_1包括:在过去第一预定时间内至少存在某天访问次数大于1的文件集合、过去第二预定时间内上传但未发布的文件的集合、以及所有需要永久保存的文件的集合。

[0224] 在一个实例中,第一获取单元21用于进行以下操作:

[0225] 向代理服务器发送淘汰预警信号;

[0226] 该代理服务器根据该淘汰预警信号向该索引系统发送获取文件请求并相应获取该需保护文件的集合U_1;以及

[0227] 接收该代理服务器返回的该需保护文件的集合U_1。

[0228] 如图25所示,代理服务器可包括接收模块510、获取模块520、发送模块530、及返回模块540。

[0229] 接收模块510用于接收第一获取单元21发送的淘汰预警信号;获取模块520用于获取淘汰参数(如t1、t2、及t3)。发送模块530用于:向索引系统发送获取文件请求以及参数t1及t2;向主控服务器603发送目录遍历请求及参数t3。返回模块540用于将从索引服务器获取的需保护的文件的集合返回至主控服务器603。

[0230] 文件删除模块20还可包括：接收单元25，用于接收来自该代理服务器的目录遍历请求及该预定值，该目录遍历单元22是根据该目录遍历请求进行目录遍历操作。

[0231] 此外，如图24所示，上述装置还可包括文件存储模块30及目录挂载模块40，文件存储模块30用于接收一个客户端发送的文件，以及将接收到的文件存储在连接至该边缘节点的存储器内；

[0232] 该文件的存储路径为：/DATAx/(CHANNEL/)y/z/filename；

[0233] 其中DATAx表示索引号为x的存储器的根目录，CHANNEL表示频道名，filename为该文件的文件名，且x、y、z满足以下条件：

[0234] $$\begin{cases} x = \mathrm{DiskID}, & \mathrm{DiskID} \le 256/\mathrm{N}; \\ y = hash_1(filename)\ mod\ A, & 96 \le A \le 128; \\ z = hash_2(filename)\ mod\ B, & 96 \le B \le 128. \end{cases}$$

[0235] N表示边缘节点的数目，$hash_1$与$hash_2$为两个不同的哈希函数。

[0236] 目录挂载模块40，用于在该目录遍历单元进行目录遍历之前通过网络文件系统将连接至该边缘节点的存储器内的文件挂载至一个虚拟路径下，该虚拟路径为：

[0237] /data/k/(CHANNEL/)y/z/filename；

[0238] 其中，$k \leqslant 10000$，且k与x之间为均匀的多对一映射关系。

[0239] 根据本实施例的在内容分发网络中进行内容管理的装置，可提升目录遍历过程的性能，减少目录遍历操作所消耗的时间。

[0240] 第六实施例

[0241] 参阅图30，第六实施例提供一种在内容分发网络中进行内容管理的系统，其包括主控服务器100、代理服务器200以及索引服务器300。

[0242] 主控服务器100结构及功能与图8所示的主控服务器603相似，代理服务器200以索引服务器300结构及功能类似于图8所示的代理服务器607及索引系统608，可一并参考图8及相关描述。

[0243] 具体地，主控服务器100用于监测该内容分发网络的边缘节点的已用存储空间是否超出预定的阈值，若是则：从该索引服务器处获取需保护文件的集合U_1；对该边缘节点进行目录遍历以获取所有最后修改时间与当前时间之间相差超过预定值的文件的集合U_0；获取待删除文件的集合$U_2=U_0-U_1$；以及从该边缘节点内删除U_2内的每个文件。

[0244] 主控服务器100还用于：接收一个客户端发送的文件，以及将接收到的文件存储在连接至该边缘节点的存储器内；

[0245] 该文件的存储路径为：/DATAx/(CHANNEL/)y/z/filename；

[0246] 其中DATAx表示索引号为x的存储器的根目录，CHANNEL表示频道名，filename为该文件的文件名，且x、y、z满足以下条件：

[0247] $$\begin{cases} x = \mathrm{DiskID}, & \mathrm{DiskID} \le 256/\mathrm{N}; \\ y = hash_1(filename)\ mod\ A, & 96 \le A \le 128; \\ z = hash_2(filename)\ mod\ B, & 96 \le B \le 128. \end{cases}$$

[0248] N表示边缘节点的数目，$hash_1$与$hash_2$为两个不同的哈希函数。

[0249] 主控服务器100还用于：在进行目录遍历之前通过网络文件系统将连接至该边缘节点的存储器内的文件挂载至一个虚拟路径下，该虚拟路径为：

[0250] /data/k/(CHANNEL/)y/z/filename；

[0251] 其中，k≤10000，且k与x之间为均匀的多对一映射关系。

[0252] 在一个实例中，主控服务器100对该边缘节点进行目录遍历时采用广度优先遍历。

[0253] 在一个实例中，主控服务器100对该边缘节点进行目录遍历时采用非递归遍历。

[0254] 在一个实例中，主控服务器100对该边缘节点进行目录遍历时，对于同一个存储器内的目录，其遍历过程串行进行，而不同的存储器的遍历过程并行进行。

[0255] 在一个实例中，主控服务器100对该边缘节点进行目录遍历的过程中还将遍历进程绑定至特定的处理器核心。

[0256] 主控服务器100对该边缘节点进行目录遍历的结果按照索引节点、文件索引节点所在块号、或者文件目录项所在块号进行排序。

[0257] 代理服务器200用于在主控服务器100与索引服务器300之间提供通讯。具体地，代理服务器200用于：将该第一预定时间及第二预定时间发送至该索引系统；根据用户输入获取该预定值或者读取预先的设定信息以获取该预定值。

[0258] 索引服务器300用于根据请求返回该需保护文件的集合U_1。

[0259] 主控服务器100还用于：接收来自代理服务器200的目录遍历请求及该预定值，主控服务器100是根据该目录遍历请求进行目录遍历操作。

[0260] 根据本实施例的系统，可提升目录遍历过程的性能，减少目录遍历操作所消耗的时间。

[0261] 此外，可以理解，上述各实施例的装置仅为示例性说明，并不对本发明实施例提供的在电子装置中启动应用程序的装置做出任何限制，本领域普通技术人员可以将以上各实施例进行组合、稍加变化而得出新的技术方案，这些技术方案也应包含在上述的内容分发网络中进行内容管理的方法、装置及系统的范围内。

[0262] 此外，本发明实施例还提供一种计算机可读存储介质，其内存储有计算机可执行指令，上述的计算机可读存储介质例如为非易失性存储器例如光盘、硬盘、或者闪存。上述的计算机可执行指令用于让计算机或者类似的运算装置完成上述的电子装置中启动应用程序的方法。

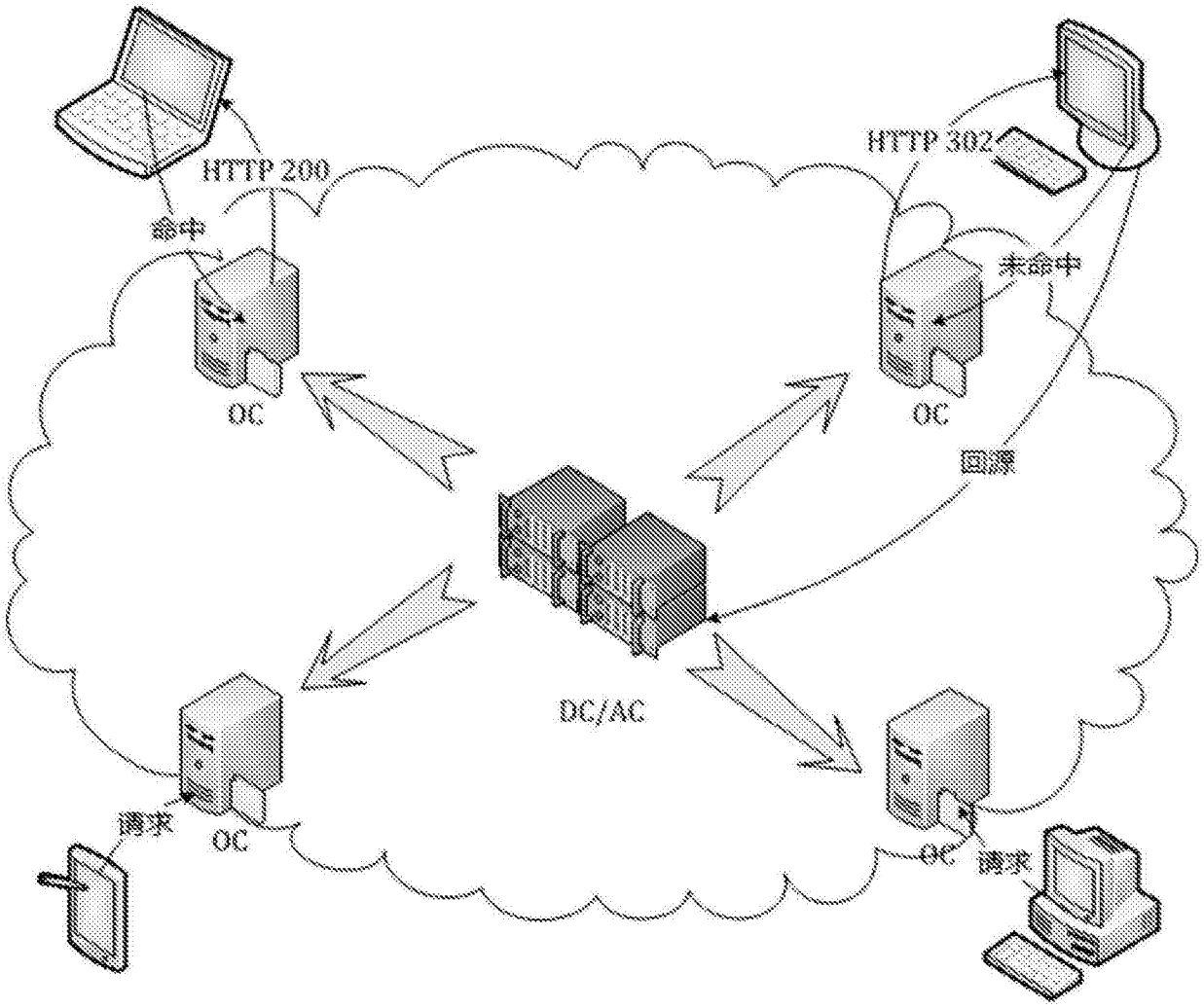


图1

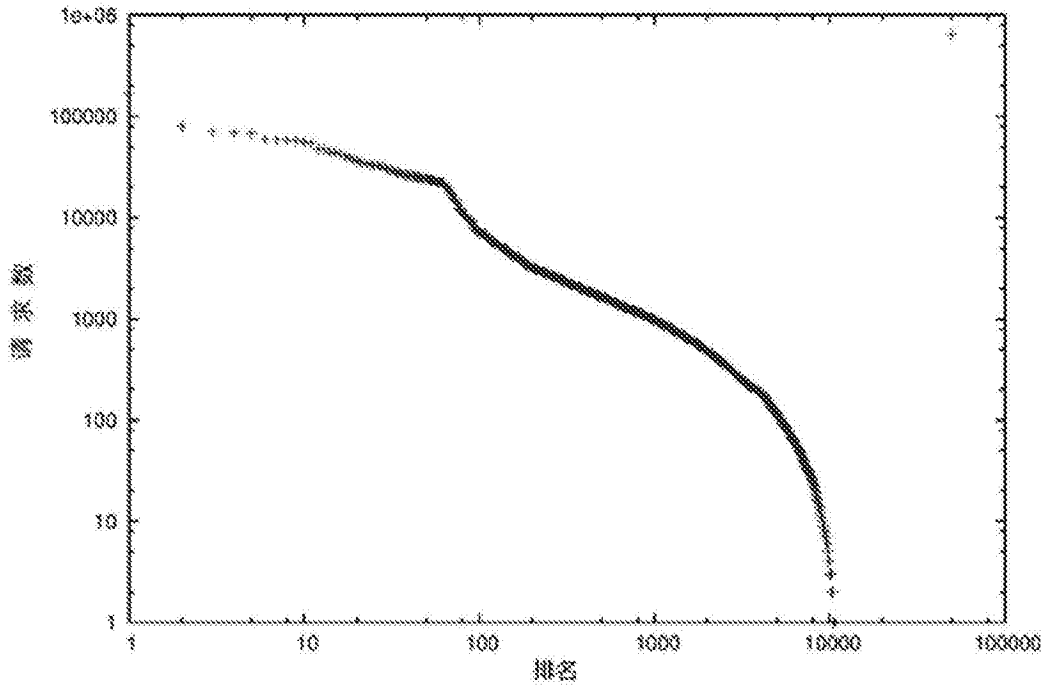

1e+06
100000
10000
1000
100
10
1
1
10
100
1000
10000
100000
排名

图2

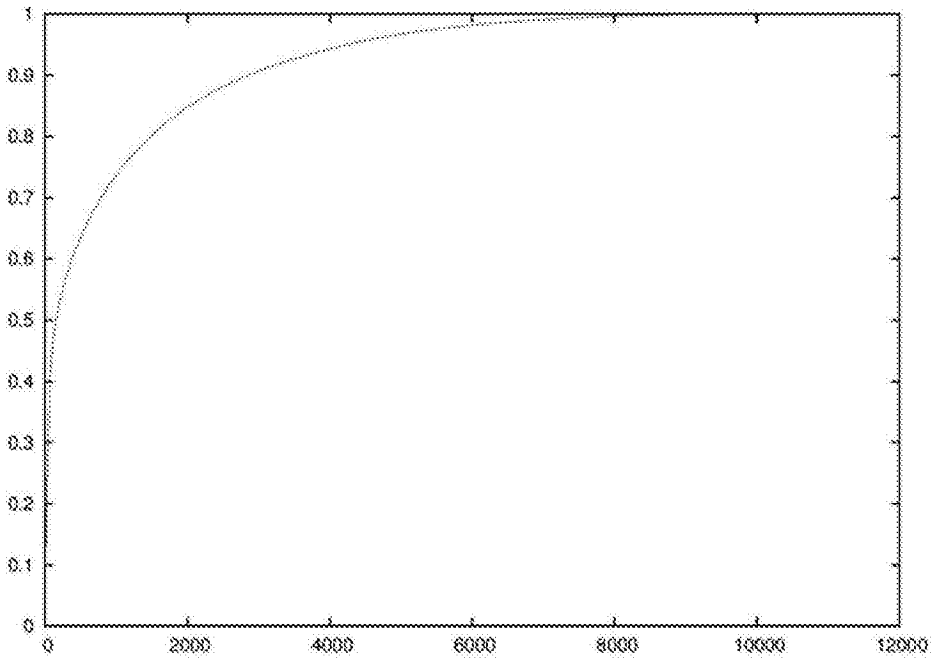

1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
2000
4000
6000
8000
10000
12000

图3

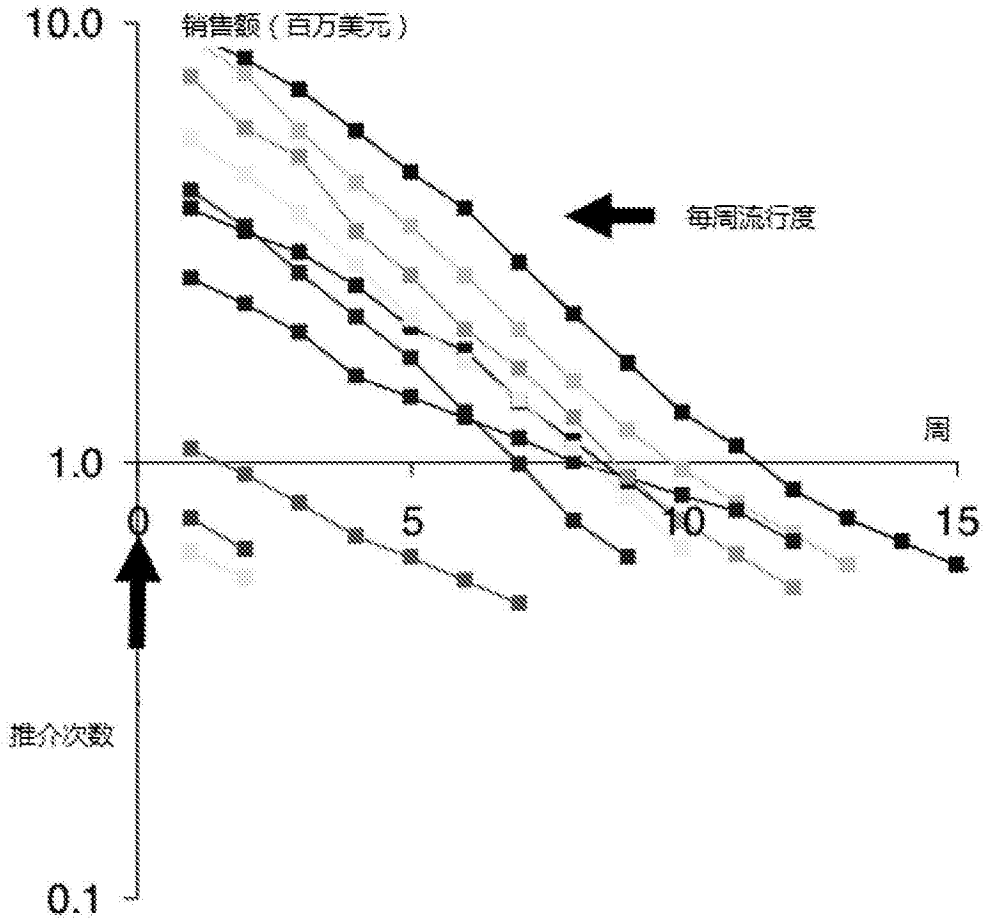


图4

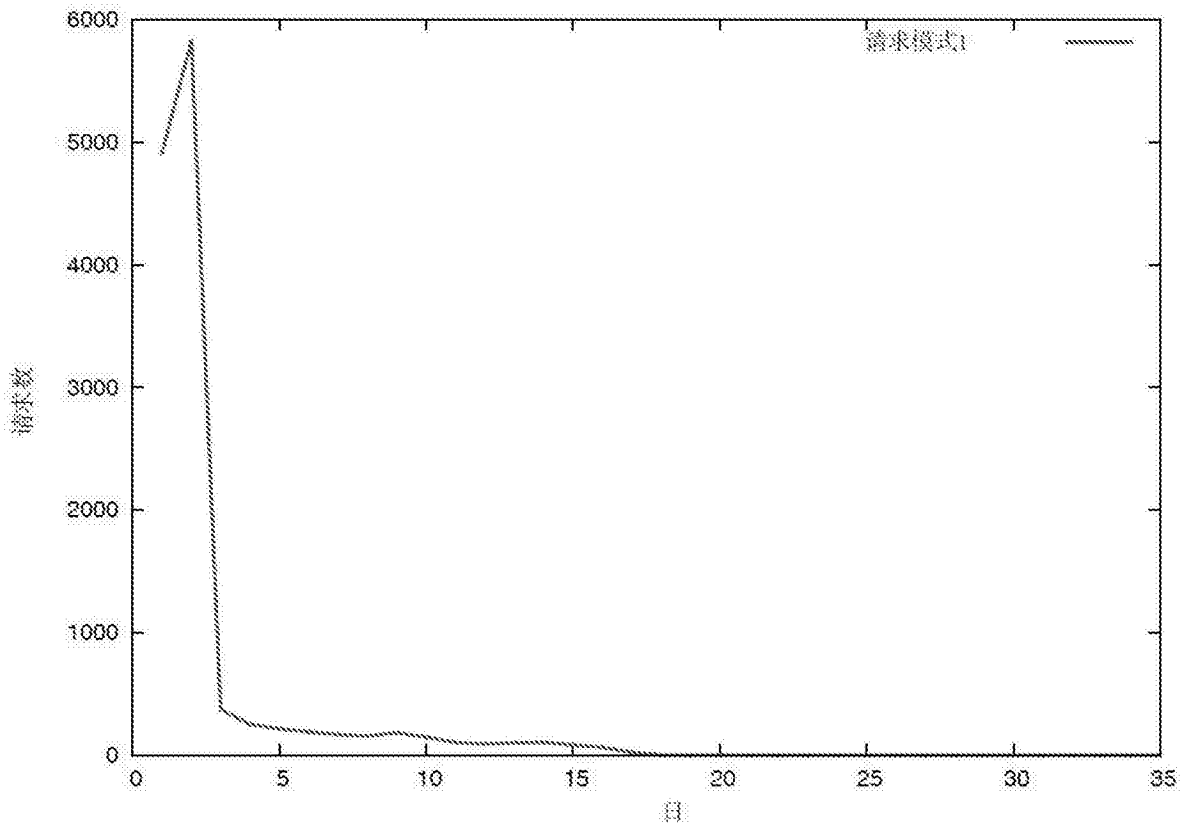


图5

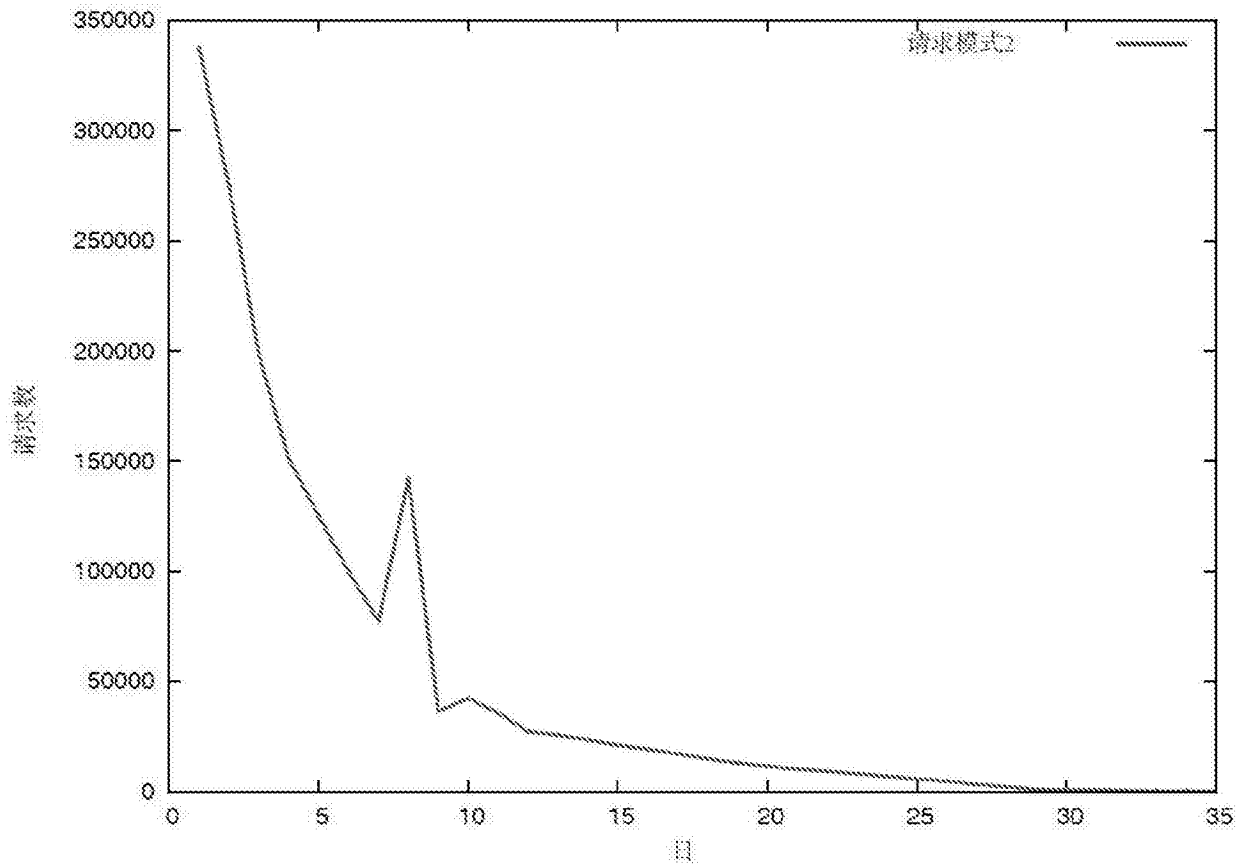

350000
300000
250000
200000
150000
100000
50000
0
0
5
10
15
20
25
30
35
请求数
日

图6

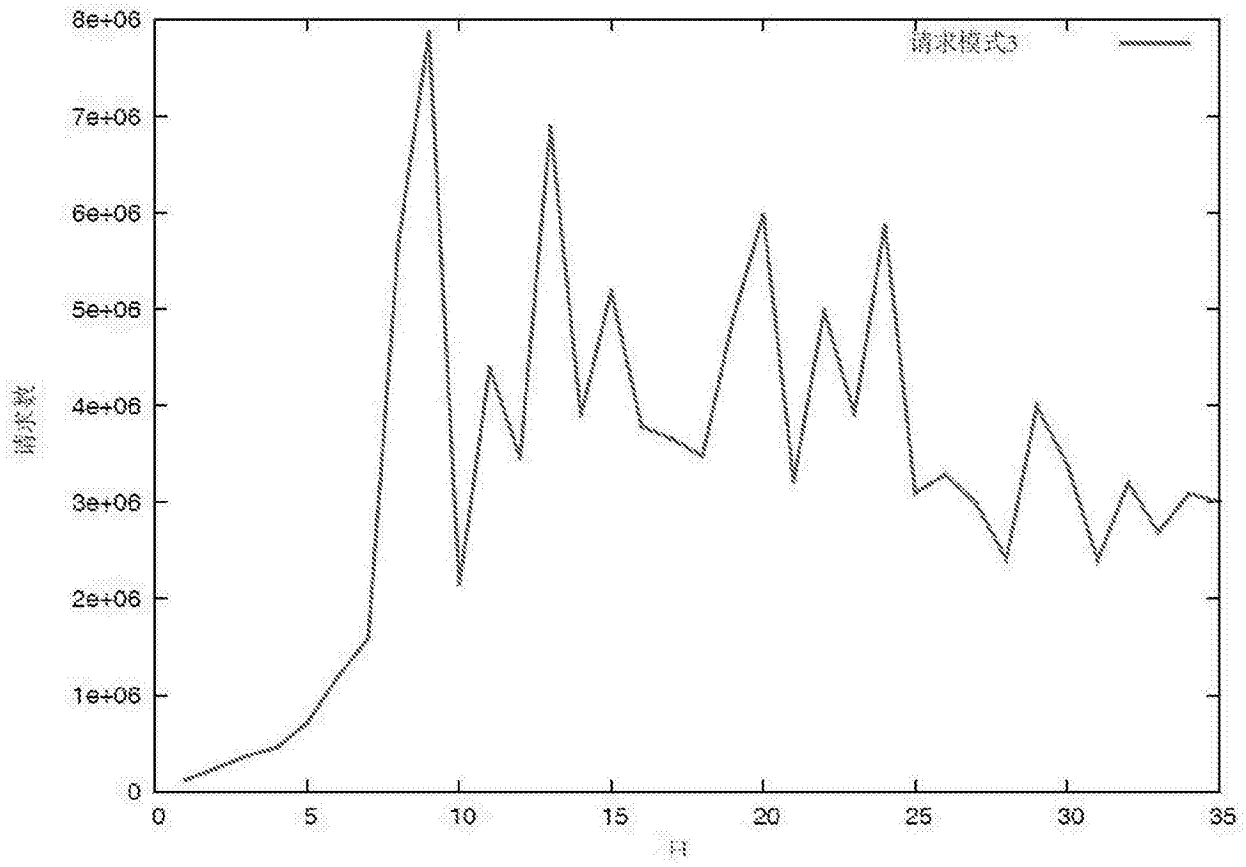

请求模式3
请求数
8e+06
7e+06
6e+06
5e+06
4e+06
3e+06
2e+06
1e+06
0
0
5
10
15
20
25
30
35

图7

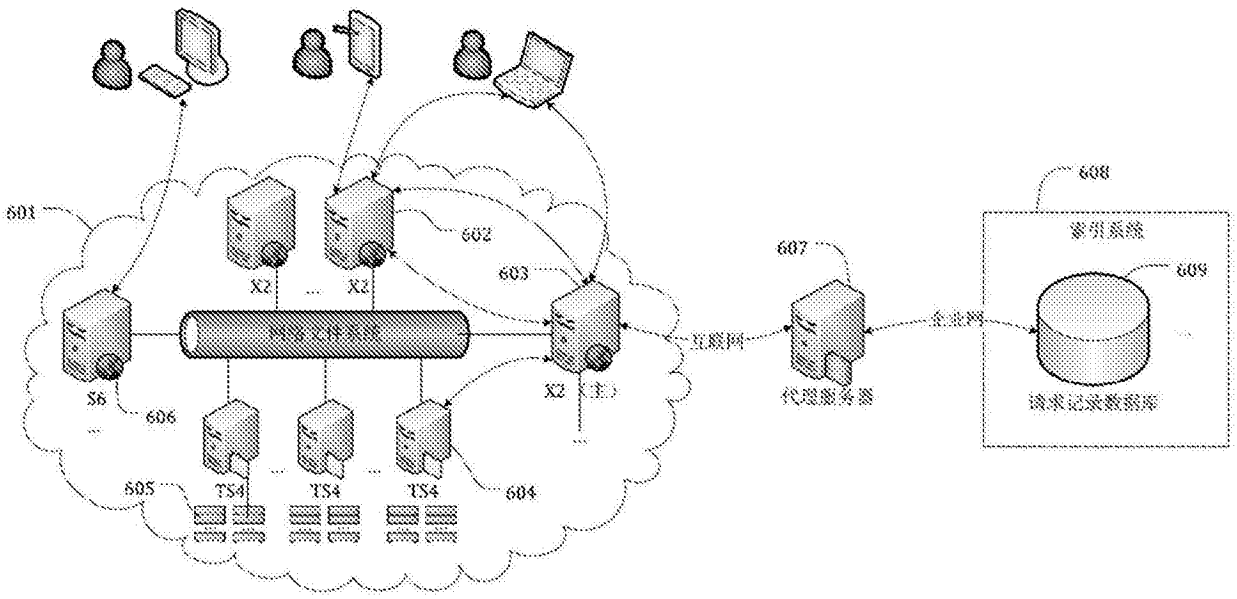

601
602
603
604
605
606
607
608
609
互联网
代理服务器
索引系统
请求记录数据库
X2
X2
X2（主）
S6
TS4
TS4
TS4

图8

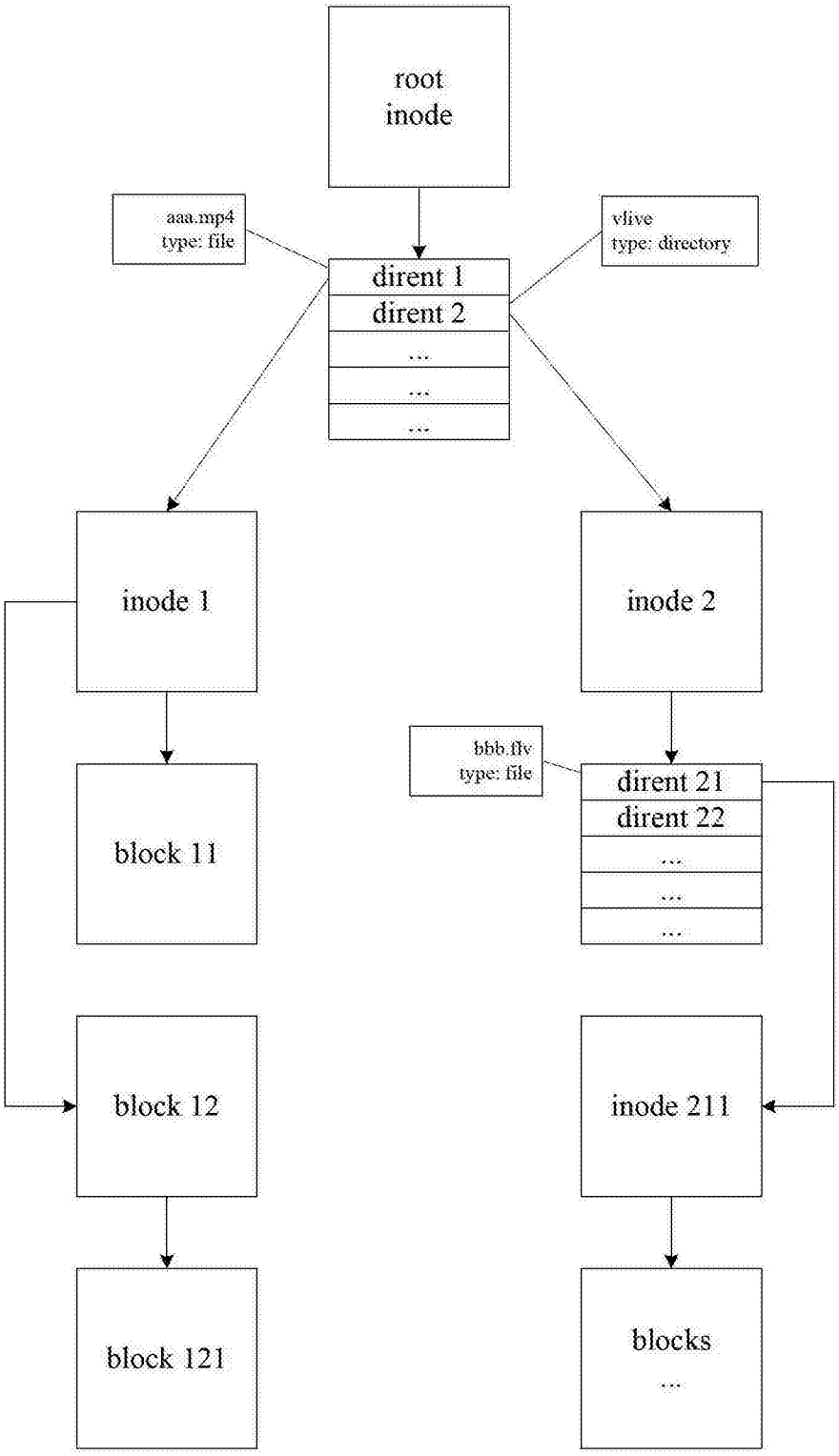

root
inode
aaa.mp4
type: file
vlive
type: directory
dirent 1
dirent 2
...
...
...
inode 1
inode 2
block 11
bbb.flv
type: file
dirent 21
dirent 22
...
...
...
block 12
inode 211
block 121
blocks
...

图9

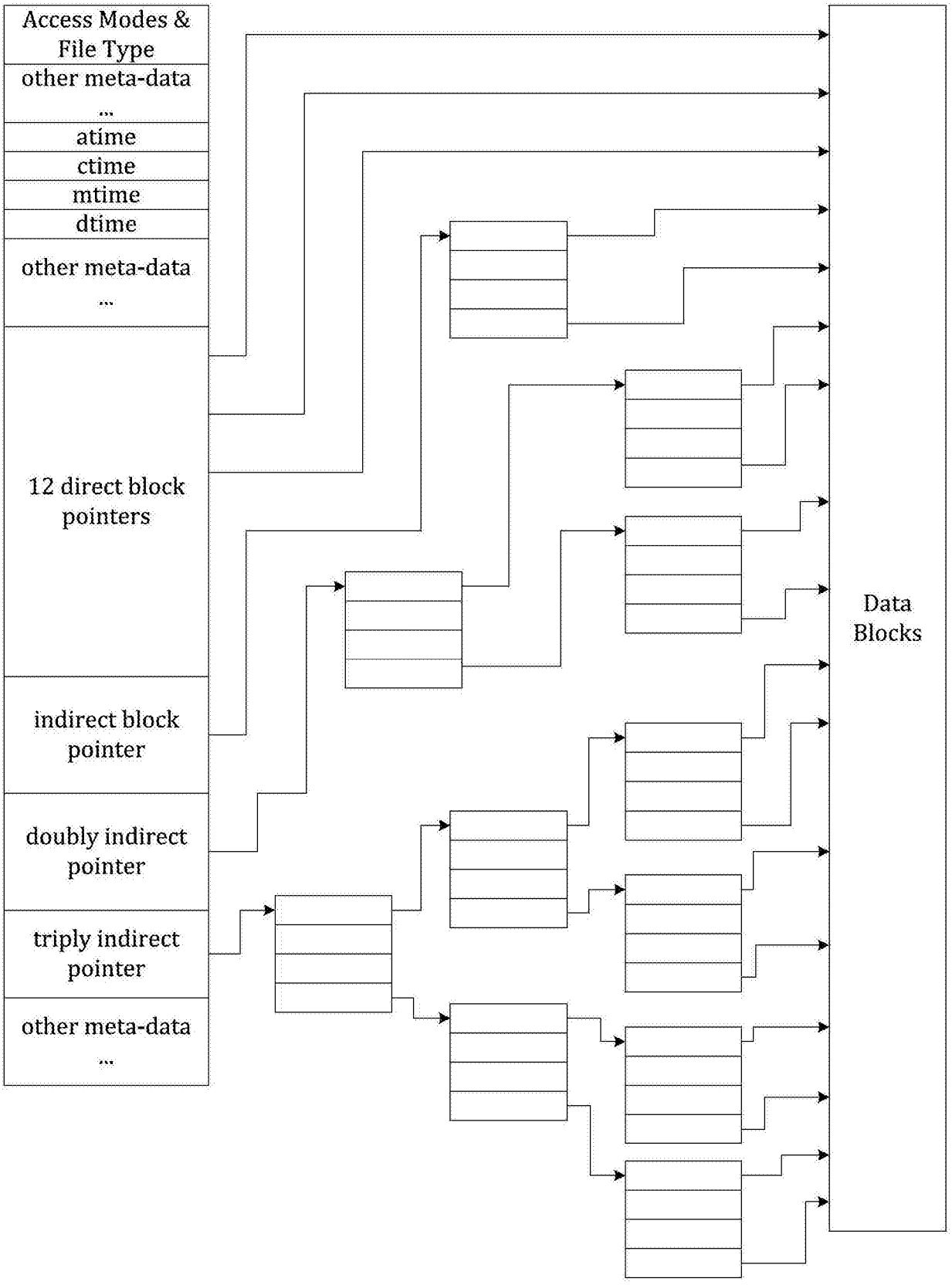

Access Modes & File Type
other meta-data
...
atime
ctime
mtime
dtime
other meta-data
...
12 direct block pointers
indirect block pointer
doubly indirect pointer
triply indirect pointer
other meta-data
...
Data Blocks

图10

inode number	offset	len	type	filename

图11

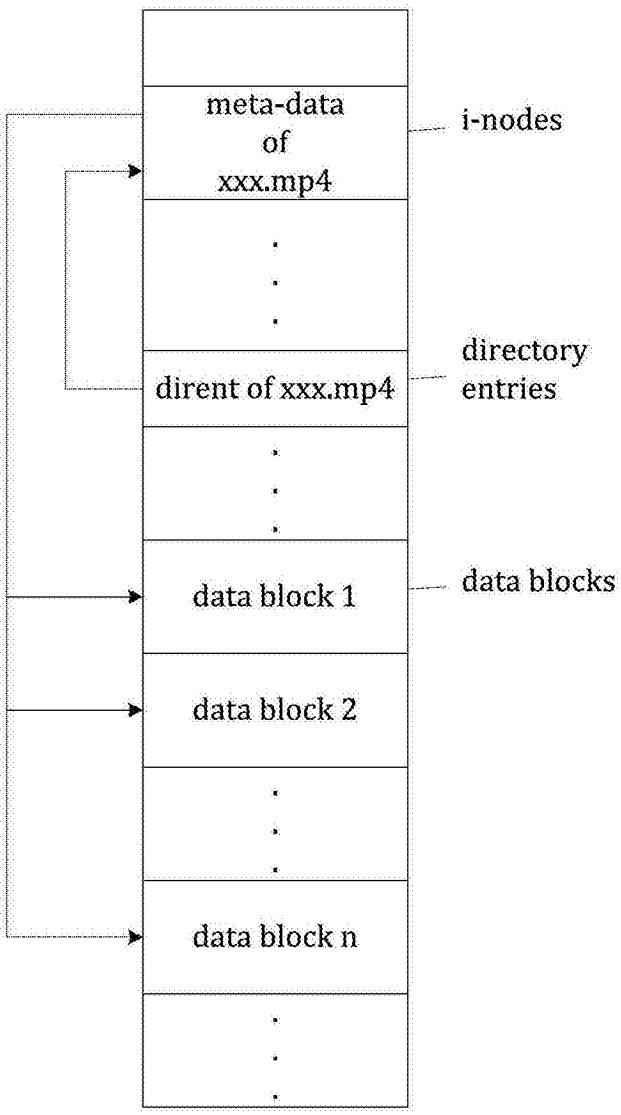


图12

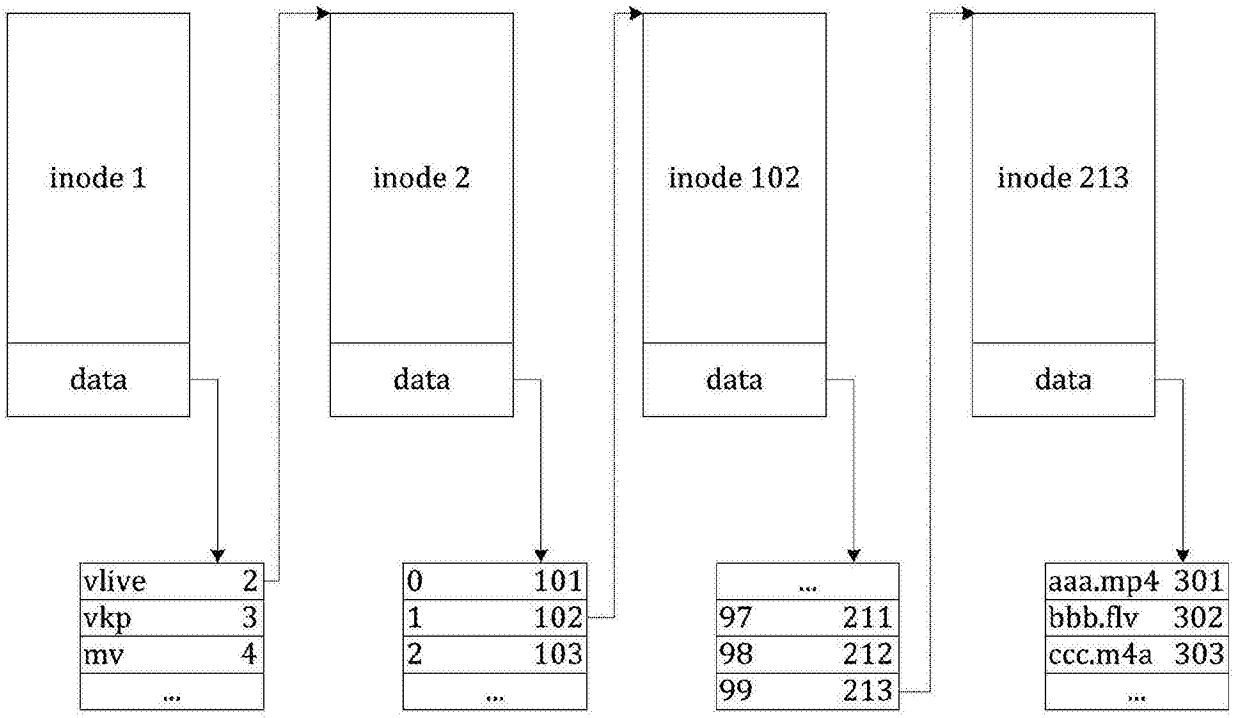

inode 1
inode 2
inode 102
inode 213
data
data
data
data
vlive 2
vkp 3
mv 4
...
0 101
1 102
2 103
...
...
97 211
98 212
99 213
aaa.mp4 301
bbb.flv 302
ccc.m4a 303
...

图13

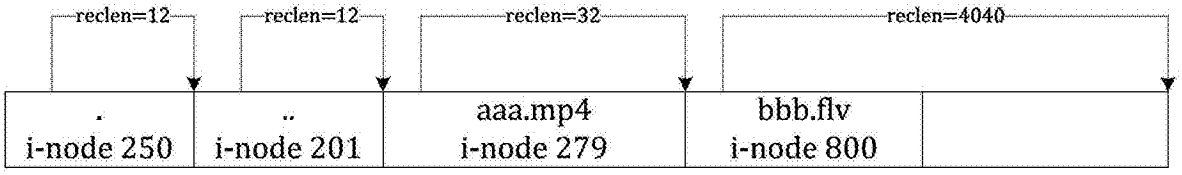

reclen=12
reclen=12
reclen=32
reclen=4040
.
i-node 250
..
i-node 201
aaa.mp4
i-node 279
bbb.flv
i-node 800

图14

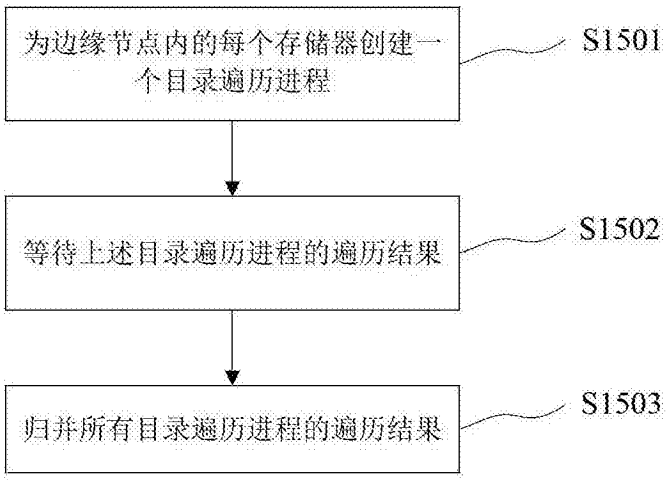

为边缘节点内的每个存储器创建一个目录遍历进程
S1501
等待上述目录遍历进程的遍历结果
S1502
归并所有目录遍历进程的遍历结果
S1503

图15

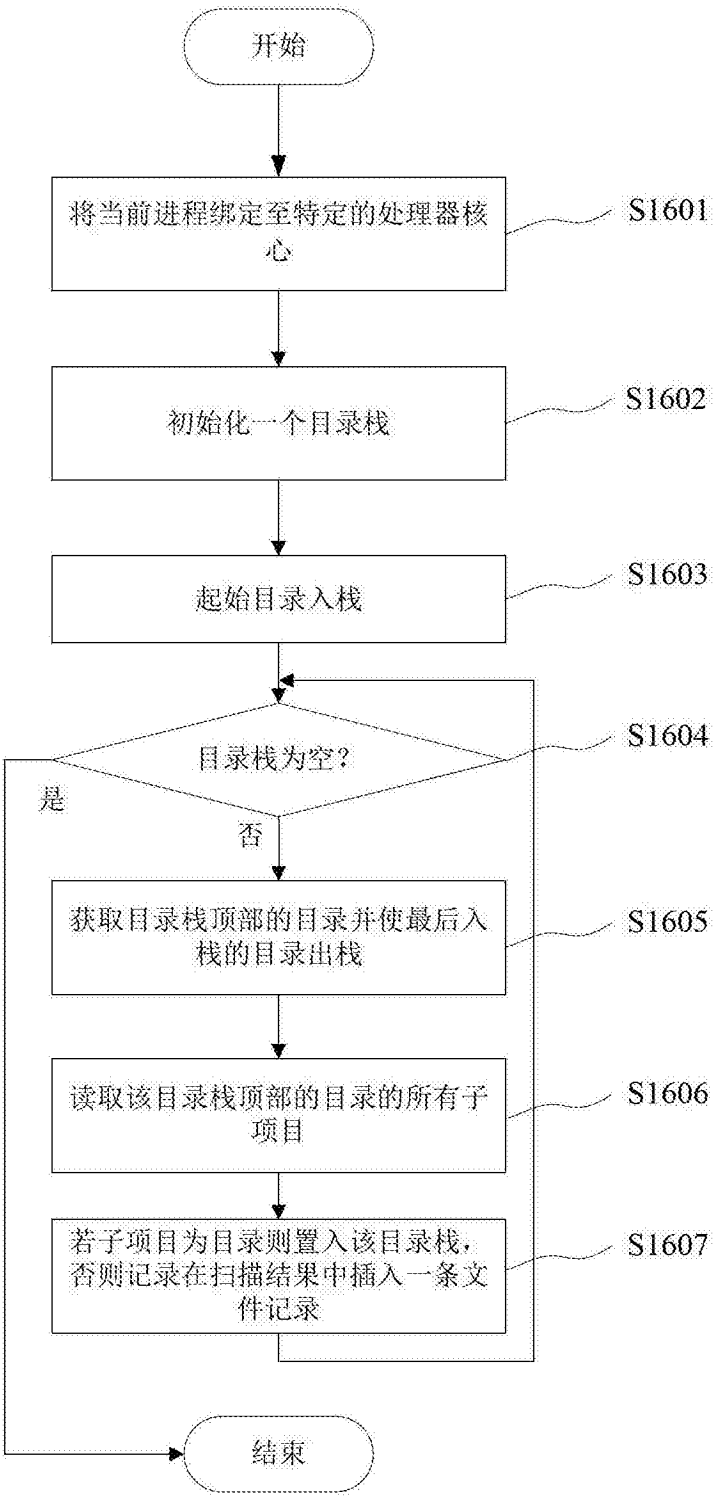

开始
将当前进程绑定至特定的处理器核心
S1601
初始化一个目录栈
S1602
起始目录入栈
S1603
目录栈为空？
S1604
是
否
获取目录栈顶部的目录并使最后入栈的目录出栈
S1605
读取该目录栈顶部的目录的所有子项目
S1606
若子项目为目录则置入该目录栈，否则记录在扫描结果中插入一条文件记录
S1607
结束

图16

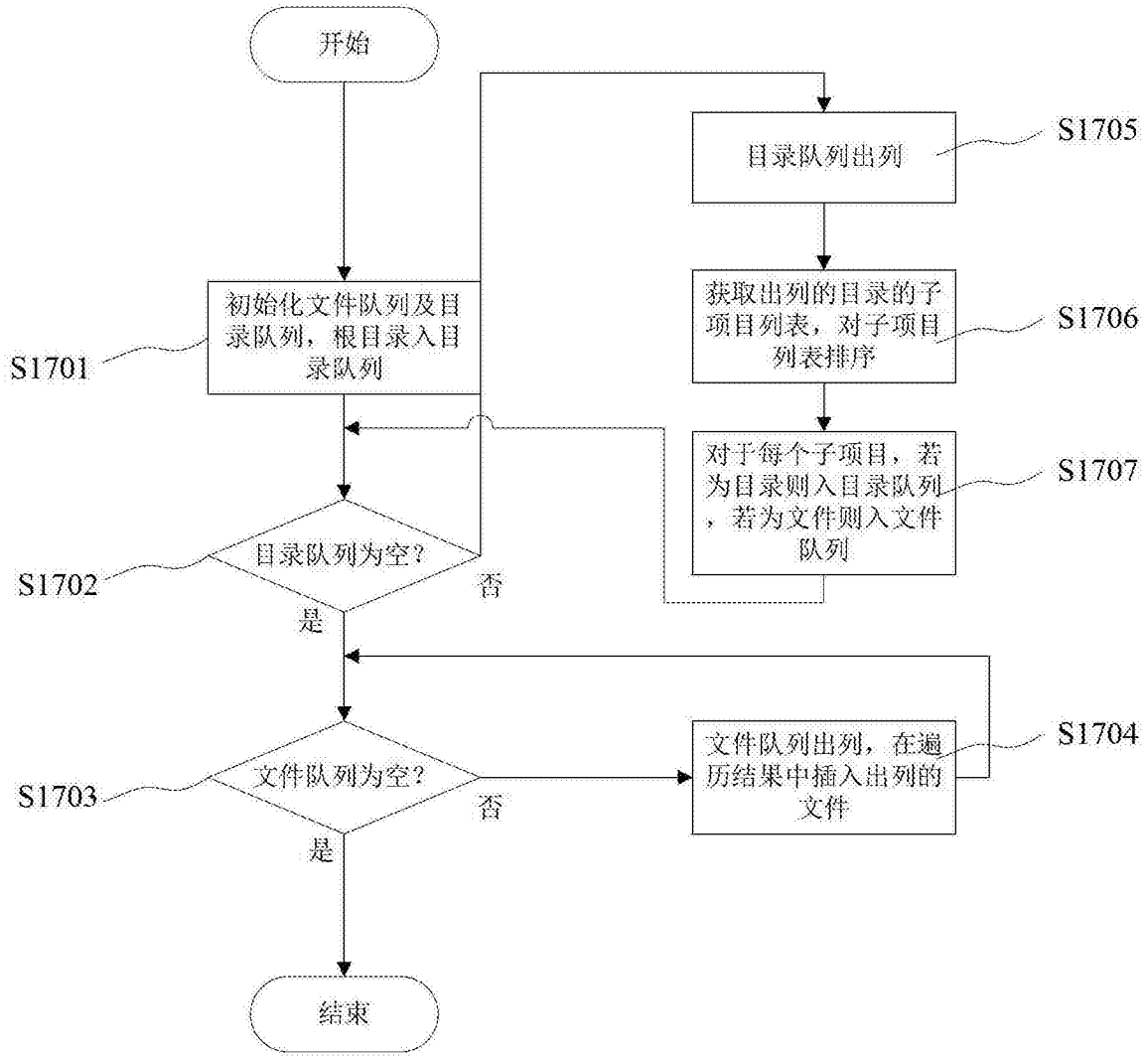

开始
初始化文件队列及目录队列，根目录入目录队列
S1701
目录队列为空？
S1702
否
是
文件队列为空？
S1703
否
是
结束
目录队列出列
S1705
获取出列的目录的子项目列表，对子项目列表排序
S1706
对于每个子项目，若为目录则入目录队列，若为文件则入文件队列
S1707
文件队列出列，在遍历结果中插入出列的文件
S1704

图17

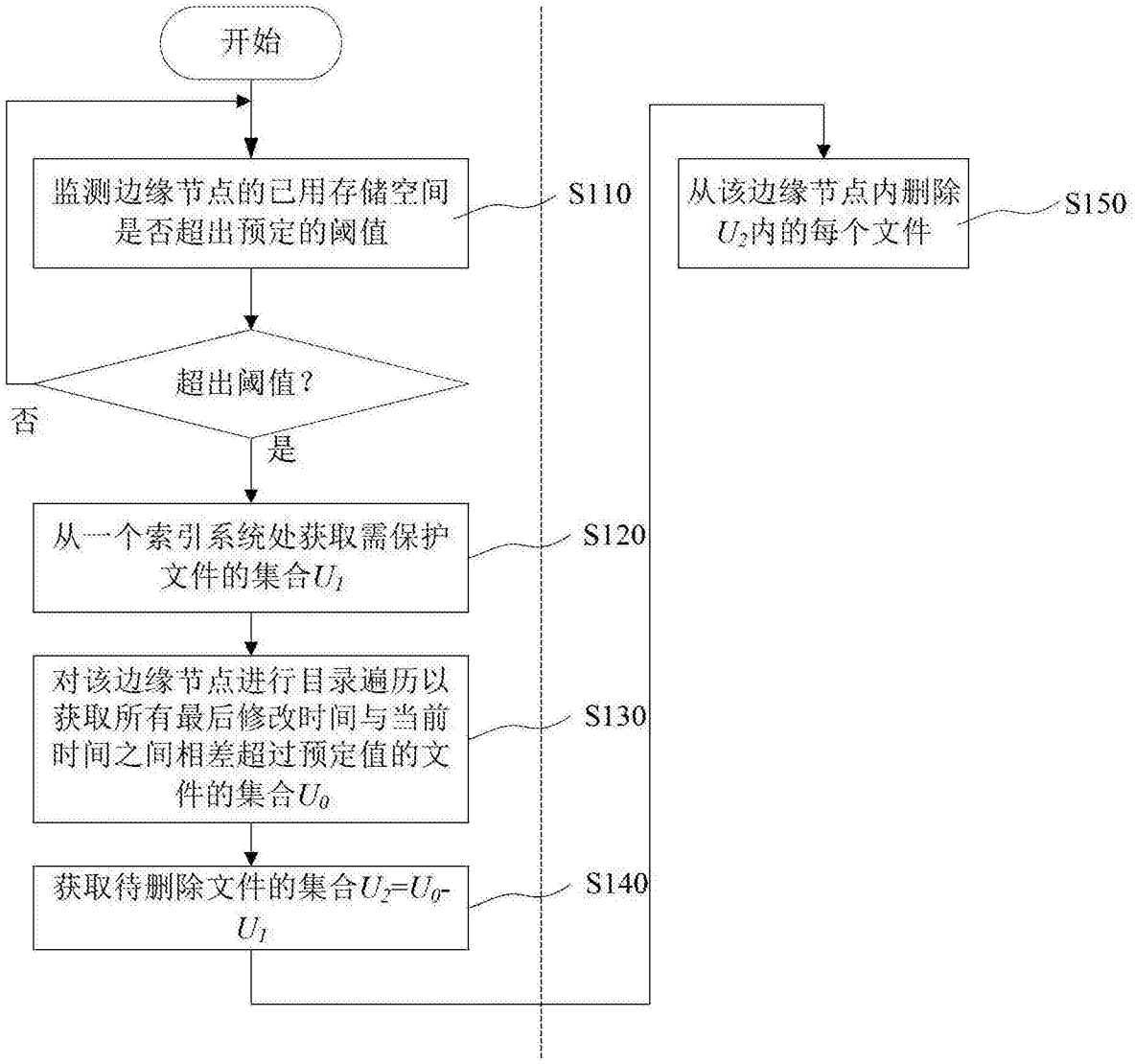

开始
监测边缘节点的已用存储空间是否超出预定的阈值
S110
超出阈值？
否
是
从一个索引系统处获取需保护文件的集合U_1
S120
对该边缘节点进行目录遍历以获取所有最后修改时间与当前时间之间相差超过预定值的文件的集合U_0
S130
获取待删除文件的集合$U_2=U_0-U_1$
S140
从该边缘节点内删除U_2内的每个文件
S150

图18

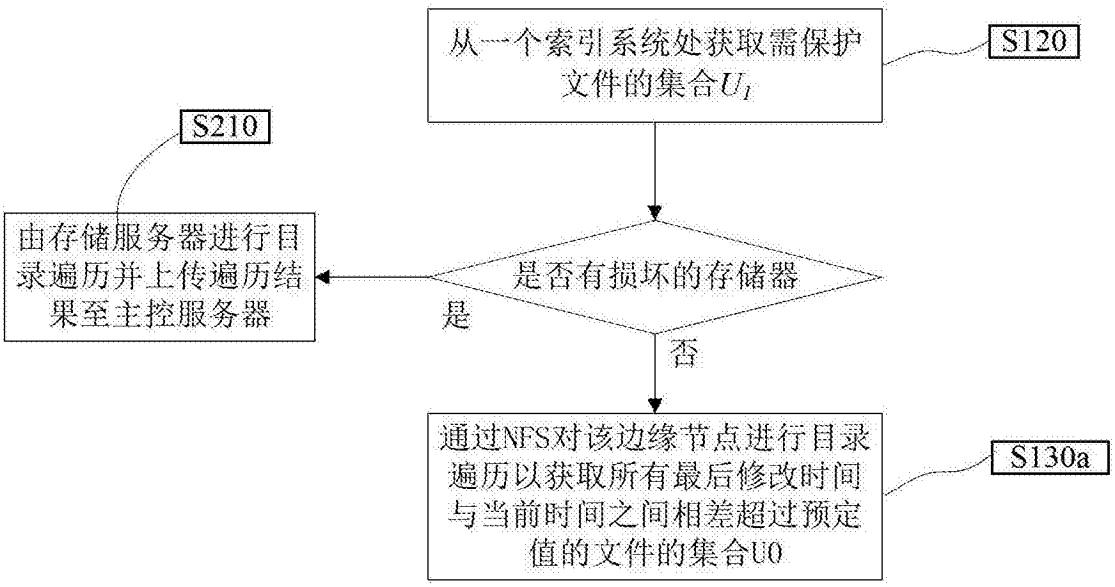

从一个索引系统处获取需保护文件的集合U_1
S120
S210
由存储服务器进行目录遍历并上传遍历结果至主控服务器
是否有损坏的存储器
是
否
通过NFS对该边缘节点进行目录遍历以获取所有最后修改时间与当前时间之间相差超过预定值的文件的集合U0
S130a

图19

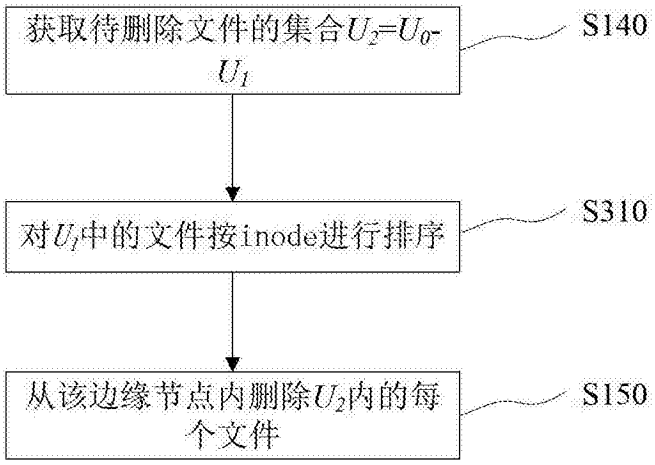

获取待删除文件的集合U_2=U_0-U_1
S140
对U_1中的文件按inode进行排序
S310
从该边缘节点内删除U_2内的每个文件
S150

图20

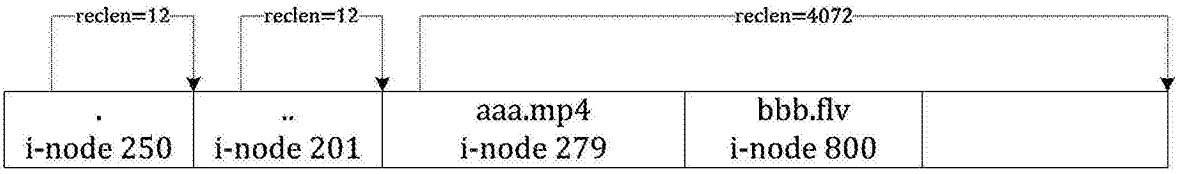

reclen=12
reclen=12
reclen=4072
.
i-node 250
..
i-node 201
aaa.mp4
i-node 279
bbb.flv
i-node 800

图21

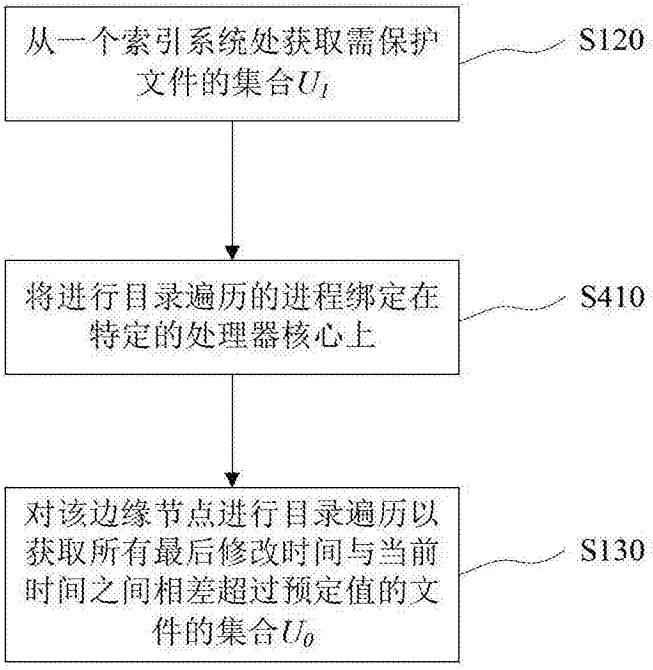

从一个索引系统处获取需保护文件的集合U_1
S120
将进行目录遍历的进程绑定在特定的处理器核心上
S410
对该边缘节点进行目录遍历以获取所有最后修改时间与当前时间之间相差超过预定值的文件的集合U_0
S130

图22

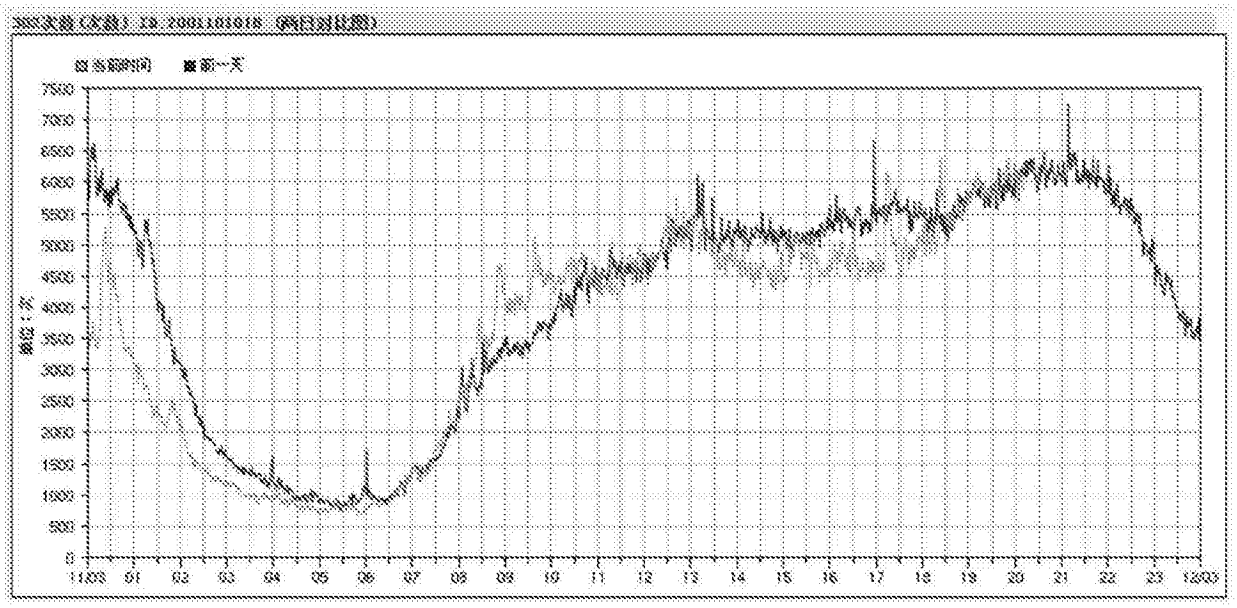


图23

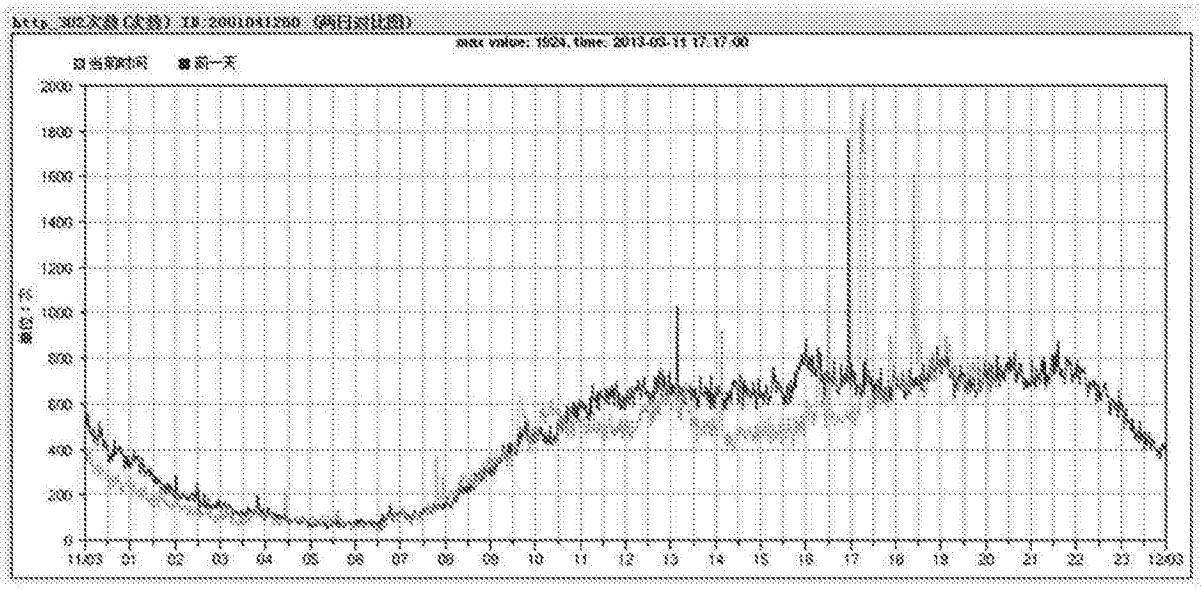


图24

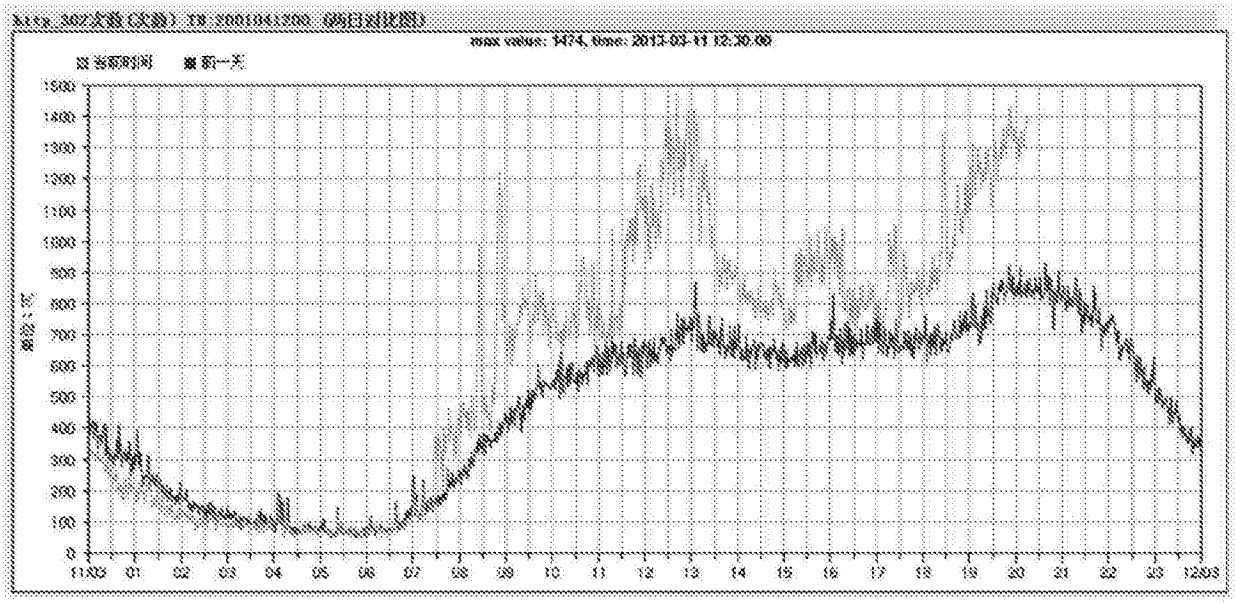


图25

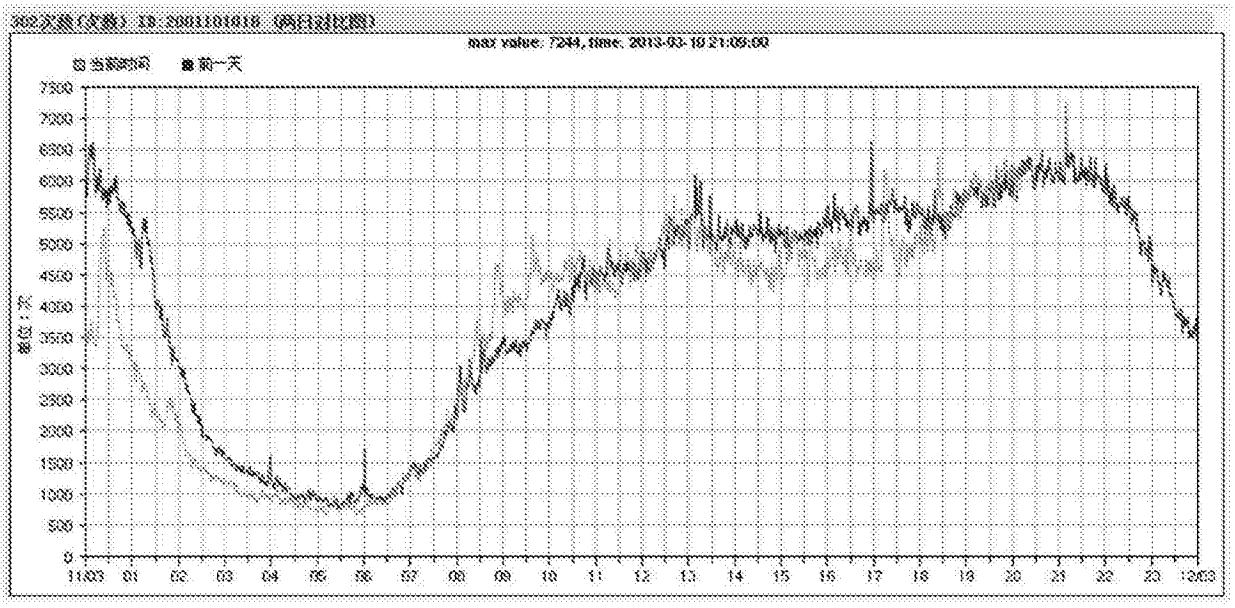

max value: 7244, time: 2013-03-10 21:05:00
当前时间
前一天

图26

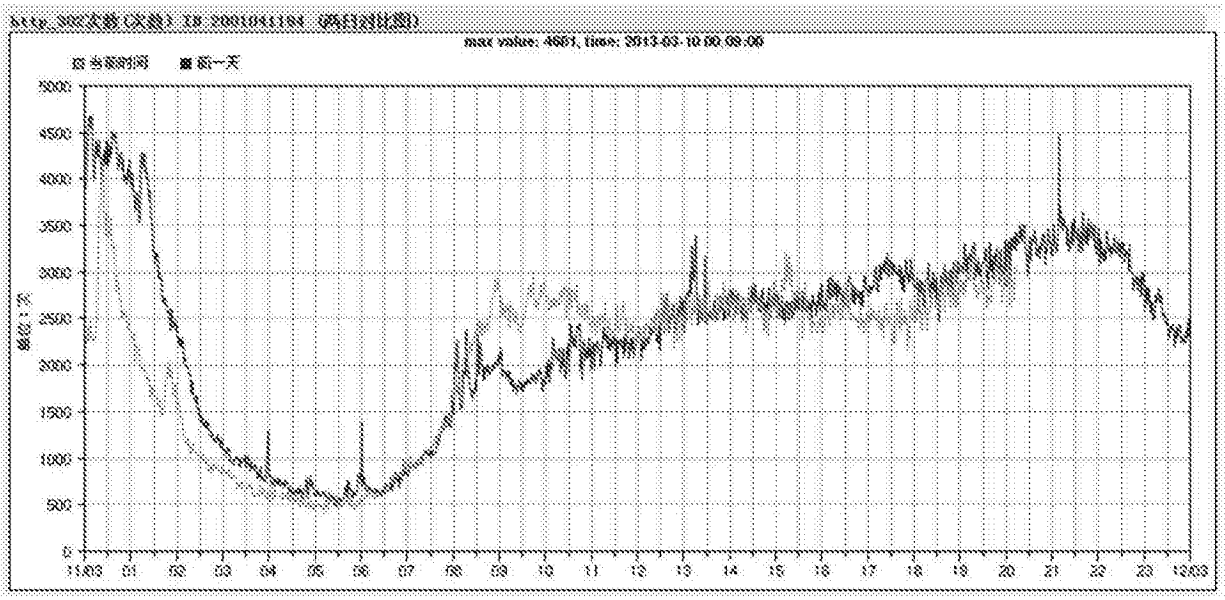

max value: 4661, time: 2013-03-10 09:00:00
当前时间
前一天

图27

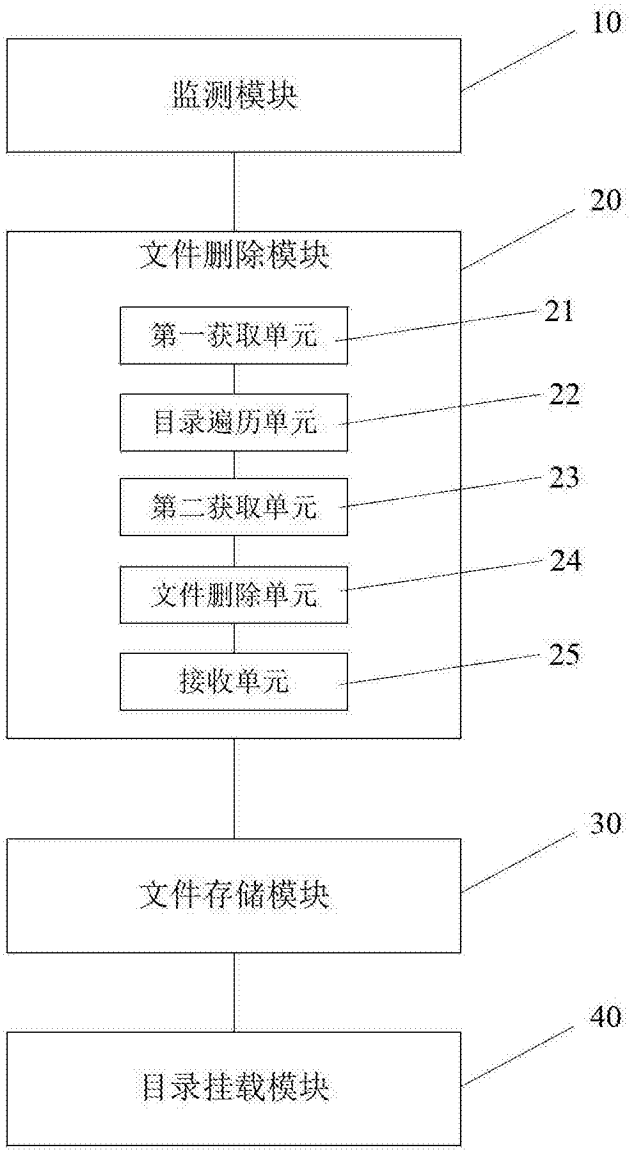

10
监测模块
20
文件删除模块
21
第一获取单元
22
目录遍历单元
23
第二获取单元
24
文件删除单元
25
接收单元
30
文件存储模块
40
目录挂载模块

图28

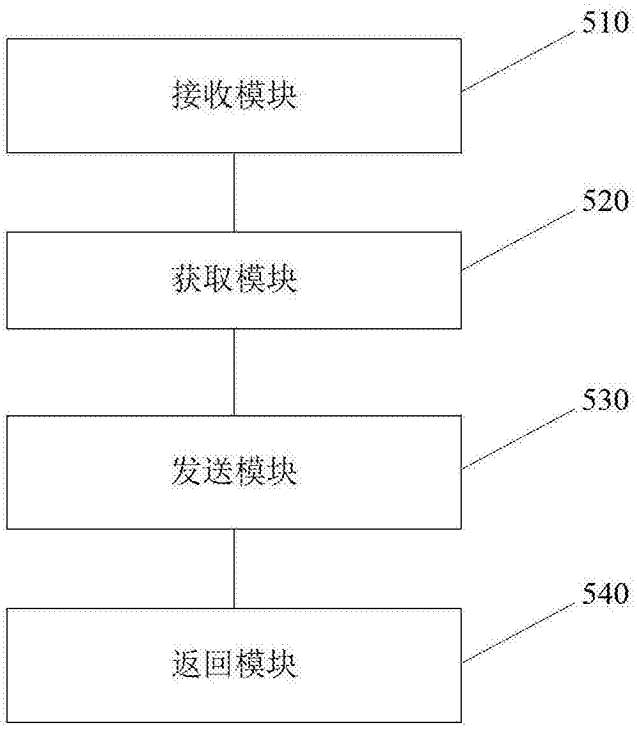

510
接收模块
520
获取模块
530
发送模块
540
返回模块

图29

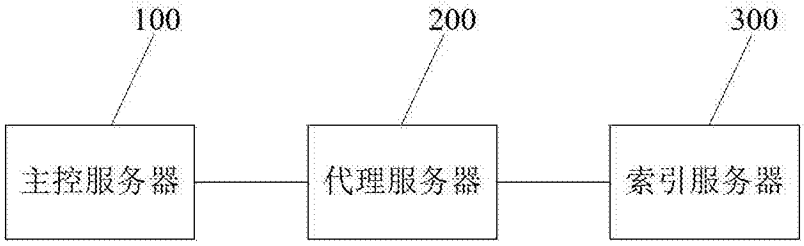

100
200
300
主控服务器
代理服务器
索引服务器

图30