一种对等结构的云存储系统研究

吴吉义[1,2],傅建庆[1],平玲娣[1],谢 琪[2,3]
(1.浙江大学计算机科学与技术学院,浙江杭州 310027;2.杭州师范大学电子商务与信息安全重点实验室,浙江杭州 310036;
3.英国伯明翰大学计算机学院,英国伯明翰 B15 2TT)

摘 要: 相对于当前各种主从(Master/Slave)结构的GFS,HDFS,Sector等云存储系统,提出了一种对等结构的云存储系统MingCloud,并采用Kademlia算法构建了原型系统.MingCloud能提供数据存储、读取、删除、搜索等云存储服务功能,并能保证系统中数据的安全性与可靠性.仿真实验结果表明,MingCloud具有较高的可用性和性能,系统通过改进和优化后能实际应用于互联网动态开放环境,并为用户提供较高质量的云存储服务.

关键词: 云存储;对等结构;Kademlia;存储系统;云计算
中图分类号: TPy302.7 **文献标识码:** A **文章编号:** 0372-2112 (2011) 05-1100-08

Study on the P2P Cloud Storage System

WU Ji-yi[1,2], FU Jian-qing[1], PING Ling-di[1], XIE Qi[2,3]
(1. *School of Computer Science and Technology, Zhejiang University, Hangzhou, Zhejiang* 310027, *China*;
2. *Key Laboratory of E-Business and Information Security, Hangzhou Normal University, Hangzhou, Zhejiang* 310036, *China*;
3. *School of Computer Science, University of Birmingham, Birmingham B*15 2*TT, UK*)

Abstract: Relative to the current Master/Slave computing model cloud storage file systems, including GFS(Google File System), HDFS(Hadoop Distributed File System), Sector, we proposed a general model of Peer-to-Peer based Cloud Storage system named MingCloud, and constructed a prototype system based on Kademlia algorithm. Highlighting the system security and reliability of data, our system can provide data storage, read, delete, search and other cloud storage services. Results from our simulation experiment show that MingCloud system has high availability and performance, which can be actually applied to the Internet dynamic and open environment after improving and optimizing, and to provide higher quality of cloud storage service.

Key words: cloud storage; peer-to-peer; Kademlia; storage system; cloud computing

1 引言

存储技术先后经历了从磁带、磁盘、RAID到存储网络系统的发展历程,目前存储系统已经成信息系统的重要基础.近几年,海量数据存储需求应用场合不断增加,直接推动了高性能存储技术的出现和发展,产生了如Google File System (GFS)[1],Hadoop Distributed File System (HDFS)[2,3],S3[4],SAN等一些典型的存储技术,并在云计算[5]中得到充分应用.

虽然有关分布式存储的研究工作已经进行了很长的时间,但分布式存储技术要在完全实用的程度上达到互联网环境中海量文件存储的目标,还有很长的路要走.当前,分布式存储中如数据容错、路由效率、信任安全、系统扩展、搜索算法、访问性能等很多问题都迫待研究解决,在云存储中这些问题也依然存在.作为近3年分布式存储领域的最新研究热点,学术界有关云存储的研究工作主要包括,美国伊利诺伊大学(University of Illinois)的Robert L. Grossman等提出并实现了一种基于高性广域网的计算和存储云Sector/Sphere[6,7],实验测试显示性能方面优于Hadoop[7].澳大利亚墨尔本大学(University of Melbourne)的James Broberg等[8,9]设计提出了MetaCDN,用于集成不同提供商的云存储服务为内容制作商(content creators)提供统一的高性能低成本内容分布式存储与分发服务.RSA实验室(RSA Laboratories)的Kevin D Bowers等[10]提出了一种高可靠性、完整性云存储模型HAIL,并进行了安全性和效率方面的实验.英国南安普敦大学 (University of Southampton)的David Tarrant等[11]提出了一种本地存储与云存储动态融合的知识库存储模型.此外,Jin Li等[12]进行了ERC(Erasure Resilient Code) 编码在对等云存储中的应用研究,验证了RSC

收稿日期:2010-08-30;修回日期:2010-11-21
基金项目:国家自然科学基金(No.61070153);浙江省教育厅科研计划项目(No.20071371);浙江省自然科学基金(No.6110317)

(Reed Solomon Code)编码的有效性.国内清华大学(Tsinghua University)也设计并实现了由分布式文件系统 Carrier 和数据共享服务系统 Corsair[13]组成,为该校师生提供个人数据存储、社区型数据分享以及公共资源数据下载等服务的云存储平台.

2　主从结构的云存储系统

目前的云存储系统基本都采用 Master/Slave 结构,包括 Google 的 GFS、Yahoo 采用的 HDFS 和 Amazon 的 S3 等.此外,还包括 Microsoft 的 SkyDrive,Sun 的 Honeycomb,HP 的 Upline,EMC 的 Atoms 等.其中 HDFS,KFS[14],Sector 为参考 GFS 进行实现的开源项目.这里将以 GFS 和 HDFS 为例分析主从结构云存储系统的基本原理.

Google 在云计算领域取得的巨大成功,相当程度上归功于以 GFS[1]为核心的先进云存储平台.GFS 是一个分布式文件系统,它能够处理大规模的分布式数据,图 1 所示为 GFS 的体系结构.系统中每个 GFS 集群由一个主服务器(Master)和多个块服务器(Chunkserver)组成,被多个客户端(Client)访问.主服务器负责管理元数据,存储文件和块的名空间、文件到块之间的映射关系以及每一个块副本的存储位置;块服务器存储块数据,文件被分割成为固定尺寸(64M)的块,块服务器把块作为 Linux 文件保存在本地硬盘上.为了保证可靠性,每个块被缺省保存 3 个备份.客户端通过主服务器向块服务器发送数据请求,而块服务器则将取得的数据直接返回给客户端.

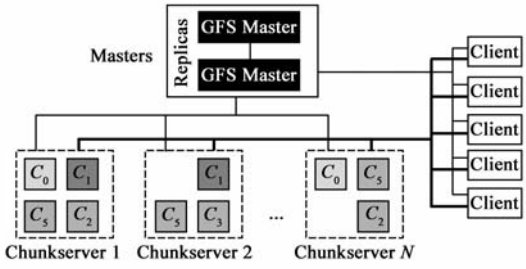

图1　Google File System体系结构

如图 2 所示,HDFS[2,3]也采用了类 GFS 的主从(Master/Slave)结构,每个集群(Cluster)由一个名字节点(NameNode)、多个数据节点(DataNode)、多个客户端(Client)组成.NameNode 主要负责管理文件系统的命名空间、集群配置信息和存储块的复制等,将文件系统的元数据(Metadata)存储在内存中,这些信息主要包括了文件信息、每个文件对应的文件块信息、每个文件块(Block)所在 DataNode 信息等.

NameNode 执行的文件操作包括打开、关闭、重命名、目录维护等,它也决定 Block 与 DataNode 间的映射.在内部,一个文件被分为一个或多个 Block.DataNode 的责任是满足 Client 的读写请求,执行由 NameNode 发出的针对 Block 的建立、删除、复制等指令.DataNode 将 Block 存储在本地文件系统中,并保存 Block 的 Metadata,同时周期性地将所有存在的 Block 信息发送给 NameNode.客户端(Client)主要是从文件系统获取文件的应用程序.

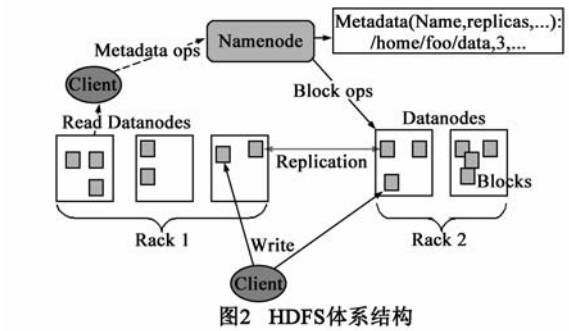

图2　HDFS体系结构

3　提议的对等结构云存储系统

相对于当前各种主从结构的 GFS,HDFS,Sector 等云存储系统,提出了一种对等结构的云存储系统.对等云存储系统 MingCloud 主要由系统运营商提供的私有节点(Private Node)和互联网高动态开放环境中公共用户计算机节点(Public Node)构成.私有节点是系统内最稳定的节点,与普通节点相比具有更强的计算及存储能力,并且能保证无物理故障情况下长时间在线,因而是具有非常高信誉的存储节点.私有节点也可以认为是对等网络中的超级节点,主要是为了提高系统可用性和整体服务质量.根据系统存储节点的规模,需要划分为不同的域(Domain),各设置一个主服务器(Master Server)用于保存用户的目录信息、文件索引信息等元数据文件,并对完成所在域用户的认证.

系统中涉及到的实体主要是节点(Node)、文件(File)和块(Block)三种,分别采用 NodeID,FileID,BlockID 作为标识符.此外,系统使用 UserID 唯一标识一个用户.

3.1　体系结构

MingCloud 采用一种分层的体系结构,从功能的角度划分为物理层(Physical Layer)、路由层(Router Layer)、数据层(Data Layer)、会话层(Session Layer)、应用层(Application Layer)等五层,如图 3 所示.

(1)物理层(Physical Layer):负责系统中存储节点的物理连接,主要由 Internet 范围内能提供闲余存储资源和计算能力的零散桌面计算机(存储节点)构成,当然还包括连接它们的底层物理通信网络其相关设备.各存储节点贡献各自的存储空间和计算资源,是构成 MingCloud 存储系统的基本元素,是文件存储的基本实体和路由转发的中间节点.

(2)路由层(Router Layer):负责系统中各存储节点

的互通性,采用 CAN, Chord, Pastry, Tapestry, Kademlia 等结构化路由算法,将系统中松散的节点结合到一起,形成一个结构化的分布式 P2P 覆盖网络,建立起节点地址空间与文件地址空间之间的映射关系.在后续的内容介绍和实验过程中,我们主要采用了整体性能相对较优的 Kademlia 或其改进算法作为 MingCloud 系统的路由层协议.

图3 MingCloud云存储系统的分层体系结构

(3)数据层(Data Layer):负责用户文件的分块、副本管理以及各种元数据的存储管理,用户的文件在该层被划分为固定大小的数据块(如 GFS 中为 64M),并且每个数据块以完全冗余副本的形式分散存储在覆盖网络各存储节点上.文件分块存储不仅能提高整个系统的负载平衡,同时还能实现数据的并行下载.为确保并提高系统的可靠性和响应能力,该层采用了完全副本冗余和纠删码冗余等策略,并运用缓存技术提高数据块的查询速度.

(4)会话层(Session Layer):负责对用户的认证、数据的加密以及目录管理.会话层根据对用户的登录信息,提供该用户的独立目录空间.采用对称与非对称相结合的方式对用户的私有文件数据进行加密,以保证用户数据的安全性.

(5)应用层(Application Layer):负责为系统用户提供文件存储服务接口,MingCloud 为用户提供了一个虚拟的云存储空间,容量在 2G 到 20G 甚至更大,主要根据用户的等级和存储资源提供方面的贡献动态调整.应用层屏蔽了下层会话、复制、路由等技术细节,用户可以像使用本地存储系统一样来访问 MingCloud 存储系统.在应用层中,可以利用系统下层提供的文件存储功能,开发各种新的应用.

3.2 文件组织

文件和目录是 MingCloud 系统最基本的组成部分.MingCloud 系统中,文件以用户为单位进行组织.用户访问存储系统时,看到的是独有的目录空间或文件名字空间.系统支持私有文件和共享文件两种文件形式.系统为每个用户文件分配一个 FileID,并进行文件分块,各分块及其副本分布在系统中的各节点上.文件的元数据项包括 FileID、文件名、文件大小、创建者、创建时间、访问权限列表、数据块列表及其位置信息等.

根据文件(或块)内容生成 FileID,在文件内容与文件标识符之间形成一个映射,也将文件属性(文件属主、文件名等)和文件标识符在逻辑上分离开来.文件内容相关的 FileID 主要用于发现系统中由不同用户共享出来的内容相同的文件,减少这类"重复文件"(Duplication File)对系统存储空间的无谓消耗.

目录文件用于记录用户存储在系统的文件目录树结构、fileID 与文件名映射、文件所有者、共享方式等信息.用户登录存储系统时,系统根据其 UserID 唯一对应的目录文件 fileID 取得目录文件,并生成该用户的独立目录空间.目录文件属于私有文件,只有文件所有者才能访问其所有的目录文件.目录文件也有一个 fileID,并像普通文件一样被存放在系统各节点上.由于目录文件非常重要,系统不仅为该文件保存数量较多的副本,还在用户经常登录的节点保存目录文件的缓存(Cache)文件.

3.3 路由算法

Kademlia 是一种基于 DHT 的路由算法,它通过异或算法(XOR)度量对等网络中节点之间的距离,使用"K 桶"路由表,建立一种全新的 DHT 拓扑结构.

在 Kademlia 网络中,每个节点都有一个唯一的 160 位长的 ID 值作为标识符.每个节点都保存一个 < key, value > 对,其中 key 为对象标识符即 ID 值,value 为对应文件的名称、所在节点的地址信息等.整个 Kademlia 网络被映射成一颗高为 160 的二叉树,每个节点被一一映射成二叉树的叶子.

Kademlia 的路由表是通过 160 个称为 K 桶的表格构造起来的,其中第 $i(0 \leqslant i < 160)$ 个 K 桶中保存了若干个和自己距离范围在区间$[2^i, 2^{i+1})$内的一些节点信息,其中节点之间的距离就是对两个节点 ID 的异或值.

每个 K 桶内部信息存放位置是根据上次看到的时间顺序排列,最近看到的放在头部,最后看到的放在尾部.每个桶都有不超过 k 个的数据项.

Kademlia 技术的最大特点之一就是能够提供快速的节点查找机制,并且还可以通过参数进行查找速度的调节.假如节点 x 要查找 ID 值为 y 的节点,Kademlia 按照如下递归操作步骤进行路由查找:

(1)计算到 y 的距离:$d(x, y) = x \oplus y$;

(2)从 x 的第$\lfloor \log d \rfloor$个 K 桶中取出 α 个节点的信息,同时进行节点查询操作,其中 α 是为系统优化而设立的一个参数.如果这个 K 桶中的信息少于 α 个,则从附近多个桶中选择距离最接近 d 的总共 α 个节点;

(3)对接受到查询操作的每个节点,测量自己和 y 的距离,以及 K 桶中节点和 y 的距离.如果发现自己离 y 最近,则回答自己是最接近 y 的;否则从自己对应的

K 桶中选择 α 个节点的信息给 x;

(4)x 对新接受到的每个节点都再次执行节点查询操作,此过程不断重复执行,直到每一个分支都有节点响应自己是最接近 y 的;

(5)通过上述查找操作,x 得到了 k 个最接近 y 的节点信息.

表 1　结构化路由算法整体性能比较

算法	路由复杂度	路由表大小	拓扑结构	〈key,value〉存放	容错性	扩展性	负载平衡性
Chord	$O(\mathrm{dxN1}/d)$	$O(d)$	O 维环形	存储在负责管理 key 所在空间的节点上	其他有效邻居节点路由	优	维护区域大的节点负载大
CAN	$O(\log N)$	$O(\log N)$	Ring	后继节点	用其后继列表中第一个正常节点替换失效节点	好	负载平衡
Pastry	$O(\log N)$	$O(\log N)$	Plaxton	键值最近的节点	路由表或邻居节点表通过某种策略选择一个节点代替	好	能够灵活地平衡负载
Tapestry	$O(\log N)$	$O(\log N)$	Plaxton	键值最近的节点	邻居节点启动接管机制	好	能够灵活地平衡负载
Kademlia	$O(k)$	$O(\log k)$	覆盖网	距离 key 最近的 k 个节点	容错性很好	好	负载平衡

由于每次查询都能从更接近 y 的 K 桶中获取信息,这样的机制保证了每一次递归操作都能够至少获得距离减半(或距离减少 1bit)的效果,从而保证整个查询过程的时间复杂度为 $O(k)$,这里 k 为存储网络全部节点的数量.从表 1 可知,Kademlia 整体性能相对较好.

针对 Kademlia 算法的改进,我们的工作主要集中于几个方面:①在消息中附加信息,由于每条信息都包含发送节点在 Kademlia 中的 NodeID,同时也允许在消息中附加了一些其他有关发件节点的有用信息.我们为每条消息附加了时间戳信息用于测量节点之间往返传输的时间.有了这些附加信息,用户可以通过一个低延时路径发送路由查询,并从多个存储资源节点选择延迟最低的一些节点完成数据存储或读取.②路由表自适应调节机制设计,以缩短查询延迟,增强网络对抖动的恢复能力.③通过增加快速查找表、添加权重等缓存策略,使重复查询的资源快速定位,以提高对热点资源的查询速度,降低了系统的开销,提高了路由效率.

3.4　基本操作

以下的操作均假定用户 User_a 已经在 Node_a 成功登录,系统已根据用户的 UserID 和目录文件 fileID 为其生成的目录空间.

3.4.1　文件数据存储

文件数据存储是系统最基本的功能,用户对本地的某一具体实体文件进行“保存”或“上传”操作后,文件数据存储过程发生.为了操作的友好性,系统也支持用户直接“拖动”本地文件到存储系统的方式.

用户在系统进行文件存储操作的一般过程描述如下:

(1)选择要存入系统的文件,假设为 F_a;

(2)根据文件内容由 SHA-1 算法获得 F_a 的 FileID;

(3)把文件划分为固定大小(64M)的数据块 B_i,并对各块分别进行以下操作:

①利用 SHA-1 对数据块进行哈希运算,计算得到值 H_i;

②采用 AES 对称加密算法以 H_i 为密钥对 B_i 进行加密,得到加密后的数据块 B_{ei};

③再次利用 SHA-1 对 B_{ei}进行哈希运算,计算得到值 EH_i;

④以 EH_i 为该数据块的 BlockID 把块存储到相应的存储节点 Node_i 上;

⑤Node_i 接收数据完毕,在自己的子集合里选择存储节点,存储该数据块的副本;

⑥Node_i 向 Node_a 返回数据块 B_{ei}存储成功的消息,同时修改自己及其他副本节点上 F_a 对应的元数据项;

(4)Node_a 收到全部数据块存储成功的消息后,将 F_a 的 FileID、文件名、文件大小、创建者、创建时间、访问权限列表、数据块列表(包括 H_i 和 EH_i)及其位置等信息保存到用户目录文件,并更新远程目录文件.

事实上,上述过程主要针对私有文件,没有考虑用户所存储的文件在系统已经存在的情况,若存储的文件是共享文件,则只是简单修改目标节点元数据文件中的共享计数器(shared-link counter)加 1,直接返回存储成功的消息,并把文件的相关信息更新入用户目录文件.

3.4.2　文件数据读取

文件数据的读取是存储的逆过程,用户登录系统后获得的目录空间提供了文件数据读取接口,目录文件可以支持一次性加载或分层逐级加载.用户对目录中的某一具体实体文件进行“打开”或“下载”操作后,文件数据读取过程发生.

文件数据读取过程是用户从存储文件对应的数据块的各存储节点处将文件数据下载到本地或下载后加载到本地应用程序的操作,一般过程描述如下:

(1)根据 FileID 获得所读取文件 F_a 在目录文件上

的节点信息;

(2)根据数据块列表信息,对文件 F_a 的各数据块 B_i 分别进行以下操作:

①由 EH_i 获得加密后数据块 B_{ei};

②用 EH_i 验证密文;

③用 H_i 解密数据;

④对明文进行验证.

(3)对解密后的数据块进行合成,读取文件成功.

3.4.3 文件数据删除

MingCloud 存储系统支持用户级的"文件移除(Remove File)"和系统级的"文件删除(Delete File)"操作.在这个问题上,私有文件和共享文件系统是区别处理的.私有文件和共享文件一般都只进行文件移除操作,对某文件进行文件移除操作后,该文件将从用户目录中移除,但文件元数据和各数据块及副本并没有真正从系统中删除.对于私有文件,文件移除类似于 OS 中的放入回收站操作,在一定的时间(如 3 个月)内文件可以被"还原"或"恢复",超过固定期限系统自动执行系统级的"文件删除"操作,并真正从系统中彻底删除该文件的元数据及对应数据块的全部副本.当然,由于存储节点的动态加入和离线,系统中可能出现删除失败的情况,如自动执行"文件删除"操作的某些时间点部分存储节点刚好离开,所以系统需要采用一定的回收机制.共享文件一般不进行系统级删除,只有当该文件的共享用户列表为空后,再经过固定期限后进行删除.

3.4.4 文件数据搜索

文件搜索是一般是针对存储系统中的共享文件的操作,是对备选文件(及其副本)存放位置定位的过程.搜索过程主要由系统中的各主服务器(Master Server)协作完成,并向用户节点返回搜索获得的备选文件列表.由于无法事先获得所搜索文件的 FileID,用户一般只能根据文件名、文件类型等属性进行搜索.由于不同 FileID 的文件可以采用相同的文件名,使得搜索很多时候是无效的操作.虽然系统搜索到了与文件名匹配的文件,但却不是所需的文件.针对这个问题,系统采用了对共享文件增加关键词和文件描述的方法,并向用户提供文件共享中的各用户对文件质量的点评或反馈信息.

文件数据的更新操作虽然是一个完整的存储系统必备的功能,但也一直是对等分布式存储系统的难题.后续研究中需要考虑的问题包括共享文件更新与读取操作的并发问题、共享文件的多用户协作更新问题、多数据块文件的更新问题等.

3.5 冗余一致性维护

与传统的集群存储系统相比,在对等存储系统(开放或封闭)中节点发生暂时或永久失效的概率更高,而且节点间的传输带宽又相对低于 SNA 和 NAS 等存储系统,其数据冗余度问题更加敏感.我们的实验数据显示,完全副本冗余较适合于存储无需进行分块的小文件,而纠删码冗余则能够满足当前各种大小文件存储的需要.MingCloud 系统能够同时支持完全副本和纠删码两种冗余方案,在本文中主要是以完全副本冗余进行仿真与性能评价的.

在实际的开放式对等存储系统中的存储节点永久失效也是非常频繁的.如果不修复永久失效节点上的副本,则数据冗余度必将逐渐降低,最终导致数据丢失.当系统发现一个节点离开时,无法知道该节点是暂时离开还是永久失效.

为解决由于存储节点离开或失效带来的数据不一致问题,在系统中专门设置了冗余一致性维护模块,负责实时监控各域(Domain)中用户存储文件有效副本的个数或采用纠删码冗余文件分块数.当判断的冗余副本数低于系统所需要的冗余度时,系统将执行冗余数据修复操作,将数据恢复至目标冗余度.该模块主要采用时间阈值法判断节点是否永久失效,当节点离线时间超过阈值 T 时就判定其永久离开系统,反之判定其为暂时离开系统.

此外,为保证存储在系统中的数据不丢失,系统通过信誉机制对域中存储节点进行分级,并为每个域配置 1 个由运营商提供的主服务器(Master Server),2 - 3 个由高信誉度公共用户提供的候备服务器(Candidate Server),都具有较长的在线时间,较强的计算及存储能力,更具有良好的信誉,能提供大量的存储资源,诚实地传递、响应其它节点的请求信息.用户进行数据存储时,有一个主副本将存储在这些高信誉度和高可靠性的存储节点上,可以基本保证在其他节点全部失效最坏情况下数据不丢失.

4 实验与性能分析

4.1 理论分析

在动态变化的网络环境中,设存储节点在线的概率(即节点可用性)为 p,在任意时候获取到已保存数据的概率(即系统可用性)为 q.则某节点失效的概率为 $1-p$.当采用 k 桶大小为 k 的 Kademlia 协议进行 P2P 路由时,网络中保存了 k 份完全冗余的数据.因此这 k 份数据同时失效的概率为 $(1-p)^k$,即有效性为 $1-(1-p)^k$.当文件被分为 b 份分别保存时,要保证这 b 份数据同时有效的概率为 $(1-(1-p)^k)^b$,即 $q=(1-(1-p)^k)^b$.可见,随着 q 随着 p、k 的增加而增加,随着 b 的增加而减少.在接下来的实验中,我们将首先保存文件,然后改变网络状态,使部分节点离线,部分已离线

节点重新上线,最后对文件进行读取.我们对上述操作重复多次,从而得出读取成功和总读取次数之间的比值,作为系统可用性的实验结果.

4.2 实验环境与目的

实验主要目的是为了验证提议对等结构云存储系统 MingCloud 能适用于互联网开放、动态网络环境并具备较好的可用性及性能,同时也为了验证 MingCloud 能提供用户数据的持久云存储服务.

到目前为止,公认成熟的 P2P 模拟器不多,代表性包括 p2pSim,3LS,GnutellaSim 和 PeerSim 等.考虑到扩展性问题,本文采用了 Java 语言编写的开源模拟工具 Peersim.同时课题研究的基础性工作,我们用 Java 实现了 Kademlia,Chord 算法,并集成到 Peersim 模拟系统中.CAN,Pastry 和 Tapestry 算法实现采用了 Peersim 官方网站提供的源码.模拟系统运行在双核 CPU 2.4GHZ,内存 6G 的服务器上.软件环境为 Window XP,JDK1.6 和 Java 开发工具 Eclipse 3.0.实验中使用的各项参数及说明如表 2.

表 2　实验中使用的各项参数及说明

参数	说明
N	存储节点总数
p	节点可用性
availability	系统可用性
filesize	文件大小,单位为 M
bandwidth	带宽,单位为 M/s
latency	消息的平均延时,单位为 ms
a	路由协议中的查询并发数
k	副本数量,同时也是 k 桶的容量

仿真实验主用从文件分块数量、网络节点总数、副本数量分别对系统可用性的影响和副本数量对系统性能的影响等方面对提议系统进行评估.

实验分为初始化和文件读写两个过程.在初始化过程中,首先构造出 N 个在线节点,给每个节点随机分配 160 位长的 id,并初始化每个节点的 160 个 k 桶,保证 k 桶内数据尽量随机.设置常用参数后,根据参数中的节点可用性,随机关闭部分节点(比如:节点可用性为 0.4,则关闭 60% 的节点).在文件读写过程中,先任选一个节点保存新文件,再根据节点可用性随机开启关闭同样数量的节点,然后再任选一个节点随机读取文件.读写过程重复 1000 次,统计其读取成功率或平均读写文件时间.

4.3 实验结果与分析

4.3.1 文件分块数量对系统可用性的影响

本次实验中,我们取 $N = 1000$, $p = 0.6$, $filesize = 1$, $bandwidth = 1$, $latency = 25$, $a = 3$, $k = 6$,实验结果如图 4 所示.

如图 4 所示,系统可用性随着分块数的增多而逐渐下降.这是因为随着分块数的增加,所有分块都存在于网络中的概率就会下降.所以应该尽可能减少分块数,最好是不进行分块.因此系统比较适合于保存小文件.

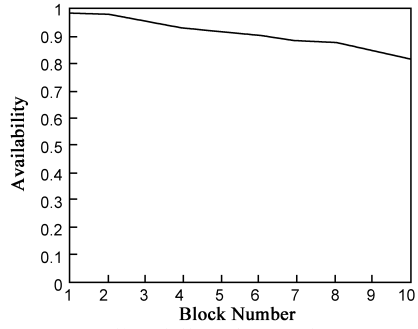

图4　文件分块数对系统可用性的影响

4.3.2 存储节点总数对系统可用性的影响

本次实验,取 $p = 0.6$, $blocknum = 5$, $filesize = 1$, $bandwidth = 1$, $latency = 25$, $a = 3$, $k = 6$.此处可以取 $blocknum$ 为 1,这样可以使系统可用性达到最高,但是为了更好地观察到实验效果,取了 $blocknum = 5$,之后的实验也类似.

实验结果如图 5 所示,网络节点总数对系统可用性影响不大,而在网络节点总数很少(实验中为 100 左右)时,系统可用性很高.这主要是因为选取了相对较高的节点可用性,以及相对较大的副本数量.使得在网络节点总数很少时,所有包含副本的节点同时下线的概率很低.随着网络总节点数增加到一定数量(本实验中为 1000 左右),系统可用性就与网络节点总数几乎无关.

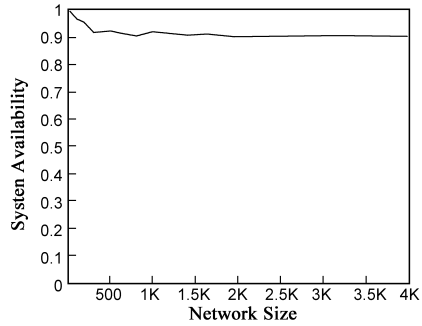

图5　网络节点总数对系统可用性的影响

4.3.3 副本数量对系统可用性的影响

本次实验,取 N = 1000, blocknum = 5, filesize = 1, bandwidth = 1, latency = 25, a = 3.节点可用性 p 分别取 0.4,0.6,0.8,得到的实验结果如图 6 所示.

由图 6 可知,随着副本数量的增加,系统可用性急剧增加,到了一定数量(本实验中为 8)以后达到稳定.当然,副本数的增加也会导致系统性能的下降,接下来来看一下副本数量对系统可用性的影响.

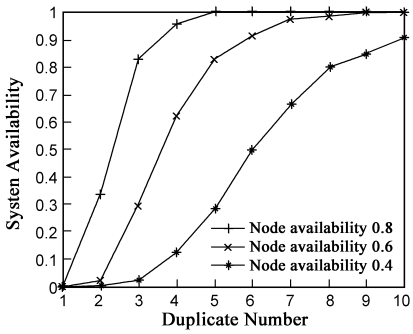

图6 副本数量对系统可用性的影响

4.3.4 副本数量对系统性能的影响

本次实验中,取 $N = 1000$, $p = 0.6$, *blocknum* = 5, *filesize* = 1, *bandwidth* = 1, *latency* = 50, $a = 3$.

由图7可知,平均存储时间随着副本数量几乎线性上升,而平均读取时间基本保持稳定.随着副本增加,本地节点需要向多个远程节点拷贝副本,导致保存时间急剧增加.而平均读取时间相对稳定,反而在副本数为1时,平均读取时间最小.这是由 Kademlia 协议的特性引起的.由于节点的 k 桶大小等于副本数,于是当节点的 k 桶(路由表)很小,每次保存文件就很难找到真正适合的节点,而是存放在本地,从而使得读取的时间变得很短.根据图6和图7的实验结论,可以找出一个副本数量的平衡点.例如在实际对等应用 BitCommit 系统中,副本数量一般取8.

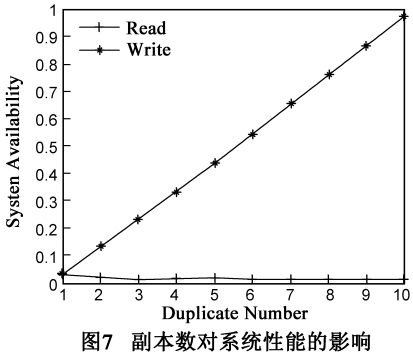

图7 副本数对系统性能的影响

5 结论以及下一步工作

评价云存储系统的一般标准包括可用性、可靠性和持久性等方面.尽管本文提出了一种具有较高的可用性和性能的对等结构云存储系统,但由于受实验条件和环境的限制研究工作还不完善.课题组只进行了简单的仿真实验,并未开发完整的系统并在现实的 Internet 上进行部署,与当前商业化的云存储系统尚存在一定的距离和可比性.因此,它的系统的可用性还需要做进一步的测试.在本文工作基础上,未来将对下面几个问题展开更深入的研究.

(1)路由算法的优化:针对 Kademlia 协议存在的网络拓扑失配、路由表更新迟缓、负载失衡等缺陷,进行优化和改进,进一步提高对等云存储系统路由算法的性能.

(2)冗余策略的研究:在当前采用的副本策略基础上,引入线形编码(linear code)、纠删码(Erasure code)等冗余编码方法,研究如何在较低存储空间需求下提供很高可靠性的云存储冗余策略[15].

(3)安全性方面的研究:MingCloud 主要通过用户认证和对存储数据进行分块加密保证安全性,但增加了系统的复杂程度,也影响了系统的性能,需要研究引入更高效的加密算法和认证机制[16].

(4)存储节点信任与信誉机制研究[17]:对已有信任与信誉计算模型进行改进和应用,在构建存储节点集群时把社会关系(如亲戚、同学、同事等)考虑在内,如只在好友节点之间提供私有文件的云存储服务.信任和信誉信息的安全存储与传输等.

此外,在存储空间的贡献方面,需要考虑如何有效激励系统中的节点贡献自己的存储空间.因为缺乏有效激励的问题会使存储系统陷入恶性循环.还需要考虑在大量节点出错的情况下如何保证系统的可靠性.

参考文献

[1] Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung. The Google file system [A]. Proc of the 19th ACM Symposium on Operating Systems Principles [C]. New York: ACM Press, 2003.29 – 43.

[2] Dhruba Borthaku. The Hadoop Distributed File System: Architecture and Design [EB/OL]. http://hadoop.apache.org/common/docs/r0.16.0/hdfs_design.pdf, 2011.

[3] Hbase Development Team. Hbase: Bigtable-Like Structured Storage for Hadoop Hdfs [EB/OL]. http://wiki.apache.org/hadoop/Hbase, 2011.

[4] Amazon. Amazon Simple Storage Service [EB/OL]. http://www.amazon.com/s3, 2011.

[5] 吴吉义,平玲娣.云计算:从概念到平台[J].电信科学,2009,(12):23 – 30.
WU Ji-yi, PING Ling-di. Cloud computing: Concept and platform [J]. Telecommunications Science, 2009, (12): 23 – 30. (in Chinese)

[6] Yunhong Gu, Robert L Grossman. Sector and sphere: The design and implementation of a high-performance data cloud [J]. Philosophical Transactions of the Royal Society, 2009, 367A: 2429 – 2445.

[7] Robert L Grossman, Yunhong Gu. Data mining using high performance data clouds: Experimental studies using sector and sphere [A]. Proc of the 14th ACM SIGKDD [C]. Las Vegas: ACM Press, 2008.920 – 927.

[8] James Broberg, Rajkumar Buyya, Zahir Tari. Creating a 'cloud storage' mashup for high performance, low cost content delivery [A]. Proc of the 6th International Conference on Service-Oriented Computing [C]. ICSOC 2008, Australia, Springer, LNCS 5472, 2009. 178 – 183.

[9] James Broberg, Zahir Tari. MetaCDN: Harnessing storage clouds for high performance content delivery [A]. Proc of the 6th International Conference on Service-Oriented Computing [C], ICSOC 2008, Australia, Springer, LNCS 5364, 2008. 730 – 731.

[10] Kevin D Bowers, Ari Juels, Alina Oprea. HAIL: A High-Availability and Integrity Layer for Cloud Storage [EB/OL]. http://eprint.iacr.org/, 2011.

[11] David Tarrant, Tim Brody, Leslie Carr. From the Desktop to the Cloud: Leveraging Hybrid Storage Architectures in Your Repository [EB/OL]. http://eprints. ecs. soton. ac. uk/17084/1/or09. pdf, 2011.

[12] Jin Li. Erasure resilient codes in peer-to-peer storage cloud [A]. Proc. of the International Conference on Acoustics, Speech, and Signal Processing [C]. Toulouse, France: IEEE CS, 2006. 233 – 236.

[13] Likun Liu, Yongwei Wu, Guangwen Yang, Weinun Zheng. Zettads: A light-weight distributed storage system for cluster [A]. Proc of the 3rd China Grid Annual Conference [C]. Dunhuang, Gansu: IEEE CS, 2008. 158 – 164.

[14] Kosmos File System (KFS) [EB/OL]. http://kosmosfs. sourceforge. net/, 2011.

[15] Lian Q, Chen W, Zhang Z. On the impact of replica placement to the reliability of distributed brick storage systems [A]. Proc of the 25th IEEE Conf. on Distributed Computing Systems [C]. Columbus, USA: IEEE CS, 2005. 187 – 196.

[16] 张薇,马建峰,王良民,郭渊博.门限 Byzantine quorum 系统及其在分布式存储中的应用[J].电子学报,2008,36(2):314 – 319.
ZHANG Wei, MA Jian-feng, WANG Liang-min, et al. Threshold byzantine quorum system and distributed storage [J]. Acta Electronica Sinica, 2008, 36(2): 314 – 319. (in Chinese)

[17] 李勇军,代亚非.对等网络信任机制研究[J].计算机学报,2010,30(3):390 – 405.
LI Yong-Jun, DAI Ya-Fei. Research on trust mechanism for peer-to-peer network [J]. Chinese Journal of Computers, 2010, 30(8): 390 – 405. (in Chinese)

作者简介

吴吉义　男,1979 年生于浙江诸暨,2007 年至今在浙江大学计算机学院攻读博士学位.现为杭州市电子商务与信息安全重点实验室主任,高级工程师,中国电子学会高级会员.主要研究方向包括云存储、SaaS 技术架构等.
E-mail: Dr_PMP@yahoo.com.cn

傅建庆　男,1982 年生于浙江绍兴,现为浙江大学计算机学院博士研究生.主要研究方向包括云存储技术、移动网络安全等.
E-mail: Jianqing_fu@zju.edu.cn

平玲娣　女,1946 年生浙江杭州,现为浙江大学计算机学院教授,博士生导师.主要研究方向为新一代互联网络、移动网络安全等.
E-mail: ldping@cs.zju.edu.cn

谢　琪　男,1968 年生于浙江上虞.2005 年获浙江大学应用数学博士学位,现为杭州师范大学教授,英国伯明翰大学访问学者.主要研究方向为密码学、电子服务安全等.
E-mail: qixie68@yahoo.com.cn