面向云计算的键值型分布式存储系统研究

孙 勇[1],林 菲[2],王宝军[1]
(1.浙江交通职业技术学院信息学院,浙江杭州 311112; 2.杭州电子科技大学软件工程学院,浙江杭州 310018)

摘 要: 对于数据密集型的云计算应用,基于磁盘的存储系统很难同时满足它们对性能与可用性的需求.本文提出了一种以内存为主设备、以磁盘为辅助设备的键值型分布式存储系统 M-Cloud,能提供大数据读写、备份及恢复等存储服务功能.M-Cloud 通过将数据全部装入服务器集群内存中的方式提高系统整体性能,并设计了分区线性哈希算法以实现负载均衡和高扩展性,设计了相应的数据备份与故障快速恢复策略以保证系统可靠性.仿真实验结果表明,M-Cloud 具有较高的性能与可用性,对系统进一步改进和优化后具有应用于实际生产环境中的潜力,可为用户提供高质量的存储服务.

关键词: 云计算; 键值; 分布式存储系统; 线性哈希; 故障恢复
中图分类号: TP302 **文献标识码:** A **文章编号:** 0372-2112 (2013)07-1406-06
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2013.07.025

Study on the Key-Value Distributed Storage System for Cloud Computing

SUN Yong[1], LIN Fei[2], WANG Bao-jun[1]
(1. *Department of Information Technology, Zhejiang Institute of Communications, Hangzhou, Zhejiang* 311112, *China*;
2. *School of Software Engineering, Hangzhou Dianzi University, Hangzhou, Zhejiang* 310018, *China*)

Abstract: In view of data intensive cloud computing applications, disk-based storage systems are difficult to meet their demand for performance and availability simultaneously. This paper puts forward a key-value distributed storage system named M-Cloud, which uses memory as main device and disk as auxiliary when providing storage services (read, write, backup and recovery) for big data. M-Cloud improves overall system performance by loading entire data in the memory of the cluster, and designs PBLH (partition-based linear hashing) algorithm to achieve load balancing and high scalability. M-Cloud also designs relative backup and fast crash recovery strategy in order to ensure high system reliability. The simulation results show that M-Cloud has high performance and availability, with further optimization, the system has the potential in real production environment usage, can provide users with high-quality storage services.

Key words: cloud computing; key-value; distributed storage system; linear hashing; crash recovery

1 引言

云计算[1]将各类软硬件设施封装为资源,以服务形式通过互联网提供给各类用户进行按需使用.云服务提供商必须以相关软硬件的统计复用和批量采购为基础,构建并运营超大型数据中心,同时为整个系统提供高效的管理框架[2].作为云计算的重要基础部件,存储系统的应用场合及需求均不断增加,推动了许多云存储技术[3,4]的出现和发展.基于磁盘的存储系统已经很难满足云计算需求,而内存具有低延时、高吞吐的特点,只是由于容量价格等因素限制,尚未被作为首要的存储设备.

利用内存计算技术提高分布式存储系统的整体性能已是当前学术界的一个研究热点.谷歌和雅虎公司的搜索引擎将全部的搜索项索引保存在内存中,谷歌的 Bigtable[5]技术允许把全部"列族"数据载入内存,但它们仅将内存作为缓存使用.传统关系型数据库处理大数据能力的不足已日益凸显,NoSQL[6]在此背景下以其强可扩展性应运而生,但它们还是以磁盘作为首要存储设备,系统性能并未提升.Memcached[7]是一种基于缓存技术的键值型存储系统,具有处理速度快的特点,但不保证数据持久性,2010 年发生的一次故障导致基于 Memcached 技术的 Facebook 网站丢失了 28TB 数据,服务终止了 2.5 小时.

收稿日期:2012-07-26;修回日期:2013-01-20;责任编辑:李勇锋
基金项目:浙江省自然科学基金(No.LY12F02017);浙江省教育厅科研项目(No.Y201223461)

本文提出一种以内存为首要存储设备,以磁盘为备份存储设备的键值型分布式存储系统(M-Cloud),做出研究贡献如下:(1)通过将全部数据保存在内存中和磁盘备份的方式在提高系统性能的同时实现了数据持久性;(2)通过多备份磁盘加多目标内存的并行快速故障恢复策略保证了存储系统的高可用性与可靠性.

2　M-Cloud 系统架构及相关定义

与其它分布式存储系统相比,M-Cloud 主要有 2 个特点:第一是将所有数据保存在服务器(存储节点)的内存中,磁盘仅作为备份设备使用;第二是支持动态伸缩的大规模服务器集群,并将所有机器的内存作为一个统一存储系统对外提供服务.

2.1　数据模型

M-Cloud 本质上是一种改进的键值型存储系统,将数据以键值对的形式存储在内存中,利用备份设备(磁盘)并行地实现数据持久性与故障恢复.

定义 1　在 M-Cloud 中,所有数据均被封装成由 $< oid, bid, value >$ 三元组表示的对象 Obj. 其中,*oid* 为访问 Obj 的唯一标识符(键);*bid* 标识了保存 Obj 的容器(桶);*value* 为可变长度字节数组(值),存储原始数据.

M-Cloud 不解析数据内容,不识别任何数据结构,这使其可处理所有数据类型. 因为数据是无结构的,所以 M-Cloud 不支持条件查询等操作,将此类任务交由上层应用完成,以牺牲传统数据库丰富功能(例如条件查询、事务)为代价来获取高可扩展性.

M-Cloud 不直接写入 Obj 数据,而是将其压缩后(默认使用可变字节编码[8])存储以节省空间. 此外,由于许多操作可直接针对压缩数据进行,所以读取性能也会有相应的提高,因为解压缩的 CPU 开销不会超过逻辑上更高内存带宽所带来的增益.

2.2　系统体系结构

M-Cloud 包含一个超级服务器(Super Node)、多个存储服务器(Storage Node)及其中的若干个"桶"(Bucket)和多个备份服务器(Backup Node),如图 1 所示.

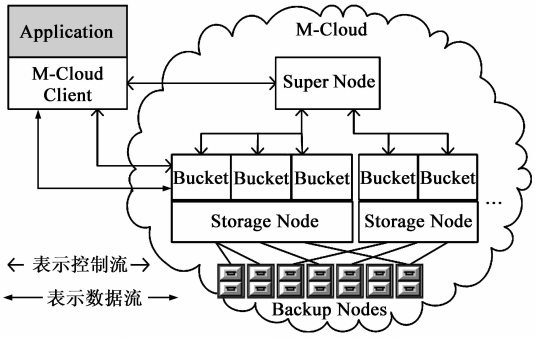


图1　M-Cloud系统体系结构图

定义 2　Super Node 负责管理所有 Storage Node 元数据,主要包括节点状态信息、映射信息. Super Node 还管理系统范围内的活动,例如 Storage Node 的扩展与缩减、系统性能监视等. Super Node 周期性地与每个 Storage Node 通讯,以获得实时状态信息. 为防止单点失效,可以通过备份[9,10]的方式实现 Super Node.

定义 3　Storage Node 负责在内存中管理 Obj 集合,包括维护 Obj 地址哈希表、实现读写改删操作、数据备份等工作. 将以上工作交由每个 Storage Node 负责可以减轻 Super Node 的负担,实现去中心化.

定义 4　基本哈希算法在选择存储节点时不考虑节点间性能差异. 为解决此问题,引入 Bucket 概念,每个 Bucket 容量相似,都属于某个 Storage Node,外部用户只与 Bucket 进行数据交互. 通过 Bucket 与 Storage Node 多对一的配置可实现处理能力权重配置和负载均衡.

定义 5　Backup Node 负责在磁盘中备份 Obj,并在 Storage Node 发生故障时与其它 Backup Node 协作参与故障恢复工作. 在实现上,Backup Node 与 Storage Node 可以是同一台物理服务器.

3　负载均衡与可扩展性

对内存资源的合理利用将对 M-Cloud 性能产生决定性影响,其中以负载均衡最为重要. M-Cloud 的服务器集群具有大规模、动态性等特点,所以负载均衡的核心任务就是在 Storage Node 不断加入、退出、失效的情况下,可以将 Obj 均匀地分布在所有的 Bucket 中.

3.1　Partition-Based Linear Hashing 算法

M-Cloud 在传统的线性哈希算法基础上设计了一种新算法,称之为 Partition-Based Linear Hashing,简称 PBLH 算法. PBLH 将所有 Bucket 分成 p 个分区,每个分区的容量近似相等. 分区指导原则是尽量将隶属于不同 Storage Node 的 Bucket 划入同一分区中,如算法 1.

算法 1　分区算法

输入:分区数 p、Storage Node 元数据.

输出:分区表 *PT*.

Step 1　根据 p 与 Bucket 数量 *BN* 计算每个分区中的 Bucket 数量 *PBN*,即 $PBN = BN/p$,初始化 *Map*,并设 $index = 0$.

Step 2　依次从每个 Storage Node 中抽出一个 Bucket,记为 b,在映射表中插入新项 $Map \leftarrow < b, index >$.

Step 3　$index = (index + 1)\%p$,如果完成全部 Bucket 的处理,则执行下一步;否则执行 Step 2.

Step 4　合并映射表 *Map*,据此创建并返回 *PT*.

在每个分区中,PBLH 使用一个独立的线性哈希函数计算 Obj 到 Bucket 的映射,即

$$H(oid)=\begin{cases}M(oid)\%2^{i+1}, & \text{if } M(oid)\%2^{i+1}<N\\ M(oid)\%2^{i}, & \text{else}\end{cases} \tag{1}$$

其中：N 为分区内 Bucket 总数，$i=\lfloor\log_2(N)\rfloor$

函数 M 的作用是为 Obj 生成一个 $b(2b>N)$位长度的键(默认采用 SHA-1[11])．对于一个 Obj，PBLH 从 p 个分区中选择负载最小的 Bucket 作为它的容器，具体步骤如算法 2．

算法 2 分区线性哈希算法

输入：分区数 p、obj(待处理的 Obj)．

输出：存放 obj 的 Bucket．

Step 1 在每个分区中，将 obj 的 oid 代入式(1)，计算出 $H(oid)$，并生成长度为 p 的候选 Bucket 集合．

Step 2 向 p 个候选 Bucket 所在的 $s(s\leq p)$个 Storage Node 发送负载查询请求．

Step 3 从 Step 2 的计算结果中选取负载最小的 Bucket 作为 obj 的容器，并根据该 Bucket 更新 obj 的 bid 属性．

Step 4 返回 $obj.bid$．

3.2 PBLH 算法的可扩展性

为提高可扩展性，哈希函数应能动态调整，并使重定位工作最小化．目前具备此种能力的哈希算法包括一致性哈希[12]、可扩展性哈希[13]和线性哈希 3 种，多数的 P2P 系统均使用一致性哈希，文献[13]则应用可扩展性哈希实现了一种目录数据结构．

与另外两种算法相比，线性哈希更适合于在局域网环境下使用．同时线性哈希比一致性哈希有更高的均衡性．PBLH 又引入了分区的概念，从而进一步提高了系统负载的均衡性，扩展方法如算法 3．

算法 3 扩展算法

输入：原分区表 OPT、nsn(新增加的 Storage Node)．

输出：新分区表 PT．

Step 1 初始化 Map，并设 $index=0$．

Step 2 从 nsn 中抽出一个 Bucket，记为 b，在映射表中插入新项 $Map\leftarrow<b,index>$．

Step 3 重定位分区 $index$ 中 b 所占用的键值空间．

Step 4 $index=(index+1)\%OPT.p$，如果完成 nsn 中全部 Bucket 的处理，则执行下一步；否则执行 Step 2．

Step 5 合并映射表 Map，据此创建并返回 PT．

可以看出，算法 3 以一种“轮询”的方式将新加入的 Bucket 依次指派到各个分区中，好处是可以动态地保持各分区的容量平衡．对于每个新的 Bucket，PBLH 根据式(1)自动将某个已存在的 Bucket 一半的键值空间移交给它，而其它 Bucket 均不受影响．

3.3 PBLH 算法分析

PBLH 主要作用是实现数据分布的动态平衡性，以保证系统可用性与可扩展性．为分析算法，定义如下：

$$LI=W(Bucket_{\max})-\sum_{i=1}^{n}W(Bucket_i)/n \tag{2}$$

其中 LI (Load Imbalance)用于衡量系统负载的不均衡性，W 为计算 Bucket 中 Obj 总容量的函数，$Bucket_{\max}$表示系统当前负载最大的桶，n 为桶的数量．

假设所有 Obj 大小相似，每次将一个 Obj 随机放到一个 Bucket 中，假设共有 m 个 Obj 且 $m>>n$，则

$$LI\leq\sqrt{m\ln n/n} \tag{3}$$

而如果将每个 Obj 随机放到 $k(k\geqslant 2$，即分区数)个 Bucket 中负载最小的桶中(PBLH)，则

$$LI\leq\ln n/\ln k+O(1) \tag{4}$$

以上推导过程可参见文献[14]．因为在实际境下 $m>>n$ 基本成立，从式(3)看出传统单路选择型算法的 LI 将随着数据量(即 m)的增长而增长．但从式(4)可看出作为多路选择型的 PBLH，LI 与 m 无关，决定 LI 的因素只有 n 和 k，这是 PBLH 算法的优点之一．PBLH 的另一个优点是可以自动消除系统扩展造成的不均衡性．

定理 1 当 Bucket 数量扩展时，PBLH 会在创建新 Obj 的过程中逐渐地自动降低 LI．

证明 假设处于均衡状态下的 M-Cloud 共有 p 个分区，每个分区平均有 n 个桶，每个桶装有大小相等的 m 个 Obj，且具有大小相等的键值空间(即从 PBLH 映射到所有桶的概率均等)．设系统扩展速率为∂，即每个分区在每次扩展中增加$\partial n(0<\partial\leqslant 1)$个 Bucket，每个新 Bucket 根据算法 3 接管了某个旧 Bucket 一半的键值空间．

设 CS 为所有新加入和被分裂的 Bucket 集合，则 $|CS|=2p\partial n$，设 RS 为所有余下的 Bucket 集合，则 $|RS|=p(1-\partial)n$，根据上述定义，如下两个条件将满足：(1) CS 中的每个 Bucket 中装载了 $m/2$ 个 Obj，RS 中的每个 Bucket 有 m 个 Obj；(2)由于 CS 中每个 Bucket 负责的键值空间是 RS 中的一半，所以 CS 中每个 Bucket 被哈希映射的概率是 $1/2n$，RS 的相应概率则是 $1/n$．

设 D 为 Super Node 可并行处理的 Obj 的总数量(CPU 核数)，设 $DALL_{CS}$、DPB_{CS}分别表示 CS 接收的 Obj 数量与 CS 中每个 Bucket 接收的 Obj 的平均值，则

$$DALL_{cs}:=\sum_{i=1}^{D}iP\{X=i\},\ DPB_{cs}=DALL_{cs}/|CS| \tag{5}$$

其中，X 为服从二项分布 $B(p,\partial)$的随机值，概率函数 $P\{X=i\}=C_p^i\partial^i(1-\partial)^{p-i}$．设 DPB 表示系统全部 Bucket 接收的 Obj 的平均值，可知

$$DPB=D/(pn(1+\partial)) \tag{6}$$

设 R 为 DPB_{CS}与 DPB 之间的比值，由上述推导可知

$$R=DPB_{cs}/DPB=DALL_{cs}(1+\partial)/2D\partial \tag{7}$$

从式(7)可知,当 $R=1$ 时,所有 Bucket 将接收相同数量的 Obj;当 $R<1$ 时,CS 中 Bucket 接收的 Obj 数量小于 RS,即系统将向不均衡方向发展;当 $R>1$ 时,CS 将接收更多的 Obj,系统将向均衡方向发展,LI 值将逐渐降低.

定理 1 说明,只要 $p \geqslant 2D(R>1)$,键值空间被分裂的 Bucket 的负载最终将与未被分裂的 Bucket 趋向于平衡,通过大量模拟测试表明,$p=2D+1$ 是一个最优取值.

4　数据的存储与备份

数据的存储与备份是 M-Cloud 最基本的功能,通过"分段"+"追加"+"缓冲"三种方法相结合的方式实现了高效的存储管理,如图 2 所示.

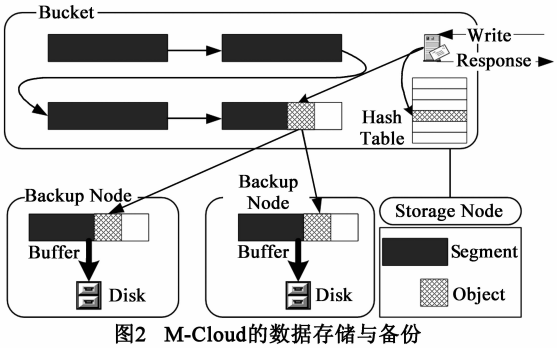


图2　M-Cloud的数据存储与备份

4.1　分段机制

M-Cloud 将每个 Bucket 的内存空间划分成固定大小的 Segment 并组成链表,Bucket 负责维护一张哈希表(记为 BHT),用于记录 Obj 与 Segment 的映射关系,以支持内存中的数据随机访问. Segment 同时也是进行数据备份与故障恢复的基本单位,每个 Backup Node 会在内存中维护一个用于缓冲数据的 Segment(记为 Buffer).

4.2　数据的写入与备份

当 Bucket 接收到数据(记为 *wo*)写请求时,将执行下列操作(见图 2):(1)以追加方式将 *wo* 写入到 Segment 链表的尾部;(2)在 BHT 中写入(或更新)*wo*. *oid* 和相应的 Segment 地址信息;(3)将 *wo* 写入到与其对应的 β(备份数,默认值为 2)个 Buffer 中;(4)写请求成功并直接返回;(5)当某个 Buffer 写满时,Backup Node 会将其中的数据一次性地写入磁盘中,并释放该 Buffer 以等待下次写入.

为防止数据丢失,Backup Node 必须保证 Buffer 中数据的持久性,可以通过备用电源(为 Buffer 写入磁盘提供足够的电量)等方式实现. 使用上述基于缓冲的备份方式使得数据写入无须等待磁盘操作即可完成. 但由于 Buffer 的容量有限,其中的数据最终还是要存入到磁盘中,所以系统总吞吐量会受到磁盘写带宽的限制,解决方法是为 Backup Node 加入更多的磁盘通过并行访问的方式提高吞吐量.

4.3　数据的读取

Super Node 的逻辑单一性极大简化了系统设计,可以在全局信息协助下精确定位每个 Obj,但为避免其成为系统瓶颈,客户端并不通过 Super Node 直接读写数据,而只是向其查询 Bucket 的位置信息,最小化所有同 Super Node 的交互是 M-Cloud 的一项重要设计原则. 以下是一次数据(记为 *ro*)读取流程示例:(1)如果客户端缓存中已存在 *ro* 的 *key*,则执行第(3)步;(2)将 *ro*. *oid* 发送至 Super Node 执行算法 2,获得 *ro*. *bid*,并把 *oid* 和 *bid* 属性作为 *key* 缓存在客户端中;(3)客户端发送读请求至由 *bid* 标识的 Bucket,获得相应 *value*,此后不必再和 Super Node 通讯.

5　故障恢复

M-Cloud 是服务于云计算应用的,当某个 Storage Node 产生故障时,由于内存易失性,其中保存的数据都将丢失. 这时需要启动故障恢复机制,以快速重建内存数据.

5.1　Fast Parallel Recovery 算法

某个 Storage Node 的故障恢复时间是由数据传输时间(记为 *DTT*)来衡量的:将故障节点所有备份数据从其 Backup Node 传输到 Bucket 所花费的总时间. *DTT* 又由数据读取时间(备份数据从 Backup Node 的磁盘读取到内存的时间,记为 *DRT*)和网络传输时间(备份数据从 Backup Node 传输至目标 Bucket 的时间,记为 *NTT*)组成. 设 *Cap* 为故障节点的内存容量,*DRate* 为磁盘读取速率,*NRate* 为网络传输速率,*SN* 和 *DN* 分别为备份磁盘数量和目标 Bucket 数量,可知

$$DTT = DRT + NTT = \frac{Cap}{DRate \times SN} + \frac{Cap}{NRate \times DN} \quad (8)$$

进行快速故障恢复的关键是最大可能地利用集群中的海量资源来降低 *DTT*,图 3 列出了 3 种可选方案.

假设 *Cap* 为 16GB, *DRate* 为 100MB/s, *NRate* 为 1Gbps,在方案(1)中 $SN=3$, $DN=1$,根据式(8)可得出 $DTT \approx 3$min;方案(2)中,$SN=1000$, $DN=1$, $DTT \approx 2$min,但通过大幅提高备份磁盘数量已使 *DRT* 在 1s 之内;方案(3)中,$SN=1000$, $DN=100$, $DTT \approx 1.44$s,在方案(2)

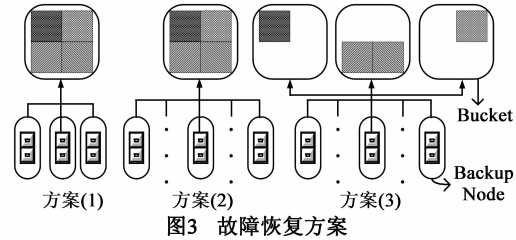


图3　故障恢复方案

基础上通过进一步提升并行率,将总恢复时间控制在2s内.

通过上述分析可知,在其它参数一定情况下,尽可能地提高 *SN* 与 *DN* 的值即可实现快速故障恢复,由算法2及其证明可知,被恢复数据会均匀地分散到各个Bucket中,即 *DN* 值优化,同时在数据备份的过程中已经保证了 *SN* 值的最大化(4.2节),以下为故障恢复算法的流程.

算法4 Fast Parallel Recovery 算法

输入:故障 Storage Node(记为 *crash*).

Step 1 Super Node 查找 *crash* 中所有 Obj 的主备份(从 Obj 的 β 个备份中指定)数据,并向相应的 Backup Node 发出数据恢复指令.

Step 2 每个 Backup Node 为其中的所有主备份数据根据算法2计算目标 Bucket,并向其转发数据恢复指令.

Step 3 各个目标 Bucket 并行地执行数据恢复工作.

Step 4 各个目标 Bucket 代替 *crash* 开始提供存储服务.

5.2 故障检测

M-Cloud 主要通过两种方式检测 Storage Node 的故障,第一种方式是在客户端访问某个节点失败后向 Super Node 发出故障报告,故障恢复模块被触发执行;第二种方式是由 M-Cloud 主动、周期性地探测各个 Storage Node 的"心跳"信息,在发现故障后及时进行恢复,以避免由多点失效造成的永久性数据丢失,为防止 Super Node 负载过重,将"故障检测工作执行者"的角色从 Super Node 开始以"接力"方式依次向状态正常的 Storage Node 传递. Super Node 只负责在收到故障报告后进行相应的处理工作,并迅速地发起故障恢复.

6 模拟实验与结果分析

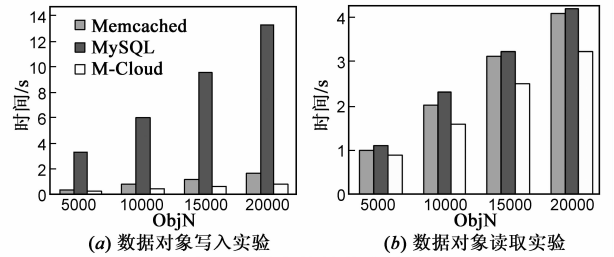
实验主要目的是为了验证 M-Cloud 能适用于云计算环境并具备较好的性能及可用性,课题组实现了 M-Cloud 原型及其各个重要算法并利用澳大利亚墨尔本大学开发的开源系统 CloudSim[15] 进行了模拟实验. 实验平台软硬件配置为 Windows 7 + Intel i5 2.53GHz(4 核) + 内存 2GB. 实验中使用的各项参数及说明如表1.

表1 实验参数说明

参数	说明
SNN	Storage Node 总数(CloudSim 节点数)
Cap	Storage Node 的平均内存容量,单位为 GB
ObjN	待处理的 Obj 总数
Size	Obj 的平均尺寸,单位为 MB
p	M-Cloud 分区数
β	每个 Obj 的备份个数

实验1 数据存储性能实验

本次实验中,我们使用 Memcached 和分布式 MySQL Cluster 作为比较对象,对于 Memcached 及 M-Cloud,将以 < *Obj*. *oid*, *Obj*. *value* > 作为键值对进行存取;对于 MySQL,创建一张包含 *oid* 和 *value* 字段的临时表,以记录形式访问. 取 *SNN* = 100, *Cap* = 16, *Size* = 1, $p = 9$, $\beta = 2$,在对 20000 个 Obj 存取过程中进行 4 次记录,并将上述过程重复 10 次取平均值,得到的实验结果分别如图4所示.



(*a*) 数据对象写入实验 (*b*) 数据对象读取实验

图4 数据存储性能实验结果

实验1表明,在一定集群规模情况下,Memcached 和 M-Cloud 均表现出较快的数据存取速度,但由于受外部程序库及副本备份的影响,Memcached 在数据处理量加大的情况下逐步落后于 M-Cloud. 同时 Memcached 由于不支持数据持久化,影响了系统可靠性,而 M-Cloud 通过异步备份保证了数据持久性和系统对性能的要求.

实验2 故障恢复实验

本次实验包含3组测试, *Size* 值分别为 1、0.5 和 0.25, *SNN* = 1000, *Capacity* = 16, $p = 9$, $\beta = 2$. 首先建立写满 16GB 内存数据的 Storage Node(包含1个 Bucket,记为 *crash*),然后通过命令关闭掉 *crash*,最后执行故障恢复程序,在多个 Storage Node 中重建 *crash*,得到的实验结果如图5所示.

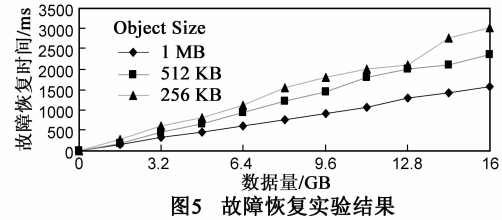


图5 故障恢复实验结果

实验2表明,在总数据量一定的情况下,大尺寸 Obj 具有更快的恢复时间,这是由于小尺寸 Obj 需要更多的 BHT 重建时间. 但同时也不宜将 Obj 的尺寸设置得过大,因为某些应用程序存在着对大量小规模数据随机读取的操作需求,对于这些应用程序,过大的 Obj 会造成带宽、存储等资源浪费. M-Cloud 中 Obj 尺寸上限默认值为 1MB,上层应用程序应在此基础上,通过对性能、故障恢复时间、资源利用率、可用性等各项系统指

标的综合权衡,进行相应的参数设置.

7 总结

本文提出了一种键值型分布式存储系统 M-Cloud,将数据全部保存在服务器集群的内存中以进行快速的存取处理,同时在磁盘上备份数据以保证持久性并为故障恢复提供支持.由于受实验条件和环境的限制,研发工作尚不完善,未来将在 M-Cloud 编程模型及其自治性等方面继续展开更深入的探索与研究.

参考文献

[1] Armbrust M, Fox A, Griffith R. Above the clouds: A Berkeley view of cloud computing[OL]. http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf, 2009-02-10.

[2] Schaffner J, et al. Predicting in-memory database performance for automating cluster management tasks[A]. Proceedings of the 2011 IEEE 27th International Conference on Data Engineering[C]. Hannover: IEEE, 2011.1264-1275.

[3] 吴吉义,傅建庆,平玲娣,谢琪.一种对等结构的云存储系统研究[J].电子学报,2011,39(5):1100-1107.
WU Ji-yi, FU Jian-qing, PING Ling-di, XIE Qi. Study on the P2P cloud storage system[J]. Acta Electronica Sinica, 2011, 39(5):1100-1107. (in Chinese)

[4] 王丽娜,等.云存储中一种基于布局的虚拟磁盘节能调度方法[J].电子学报,2012,40(2):266-272.
WANG Li-na, et al. A data assured deletion approach adapted for cloud storage[J]. Acta Electronica Sinica, 2012, 40(2):266-272. (in Chinese)

[5] Fay Chang, et a1. Bigtable: a distributed storage system for structured data[J]. ACM Transactions on Computer Systems, 2008, 26(2):1-26.

[6] Wikipedia. NoSQL[OL]. http://en.wikipedia.org/wiki/NoSQL, 2012.

[7] Dormando. Memcached: a distributed memory Obj caching system[OL]. http://www.memcached.org/, 2012.

[8] Transier F. Algorithms and Data Structures for In-memory Text Search Engines[D]. Karlsruhe Germany: a PhD thesis of University of Karlsruhe, 2010.

[9] 百度百科.双机热备[OL]. http://baike.baidu.com/view/132705.htm, 2012.

[10] 张民贵,刘斌.IP 网络的快速故障恢复[J].电子学报,2008,36(8):1595-1602.
ZHANG Min-gui, LIU Bin. Fast failure recovery of IP networks[J]. Acta Electronica Sinica, 2008, 36(8):1595-1602. (in Chinese)

[11] Wikipedia. SHA-1[OL]. http://en.wikipedia.org/wiki/SHA1, 2012.

[12] Witold Litwin. Linear hashing: a new tool for file and table addressing[A]. Proceedings of the sixth international conference on very large data bases[C]. Montreal: VLDB, 1980. 212-223.

[13] Minwen Ji, et al. Archipelago: an island-based file system for highly available and scalable internet services[A]. Proceedings of the 4th Conference on USENIX Windows Systems Symposium[C]. Berkeley: USENIX Association, 2000.1-1.

[14] Petra Berenbrink, et al. Balanced allocations: the heavily loaded case[J]. SIAM Journal on Computing, 2006, 35(6):1350-1385.

[15] R N Calheiros. CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms[R]. NY, USA: Software: Practice and Experience, Wiley Press, 2010.

作者简介

孙　勇　男,1977 年 4 月出生,辽宁盘锦人,副教授.研究方向为分布式计算系统、无线传感器网络等.
E-mail: sy@zjvtit.edu.cn

林　菲　女,1977 年 11 月出生,浙江永康人,副教授.研究方向为并行分布式计算、软件工程等.
E-mail: linfei@hdu.edu.cn