# 基于RAMCloud的大文件存储方法的研究与实现

刘钢标 张延园 林奕 樊鑫 邢新疆
西北工业大学计算机学院 西安 710129

## 0.引言

近年来，随着信息技术的快速发展，在科学计算、商业计算等众多应用领域中产生了规模巨大的数据，而且数据量仍在以较快的速度增加。如何存储和管理这些数据？分布式文件系统的出现很好地解决了这个问题。作为云存储的核心基础，分布式文件系统已经成为了计算机应用领域的研究重点，受到了当前学术界和企业界的广泛关注。现存的主流分布式文件系统大多都是基于磁盘的，它们虽然为海量数据的存储提供了解决方案，但是由于受到磁盘I/O性能的约束，它们远远不能满足一些大规模网络应用程序和集群计算框架（如Spark[1]，MapReduce[2]，Storm[3]等）的低延迟和高吞吐率的需求，从而严重影响了计算机系统的性能发挥，这种情况在云计算环境下尤为明显[4]。

与此同时，内存和网络技术两个领域取得了显著的进展。随机访问存储器的价格不断下降，其容量却以惊人的速度增加。网络技术也发生了巨大的变化，网络带宽、速度都获得了很大幅度的提高。文献[5-7]的研究表明，在高速局域网络中对网络主机内存中的数据的存取速度已经远远高于直接存取本机外存速度。因此，通过高速网络，将集群中服务器的内存整合起来，便可得到一个大容量的，基于内存的存储空间。基于这个思想，很多新兴的存储系统开始出现，RAMCloud[8]便是其中之一。针对RAMCloud无法直接存储大文件的问题，本文通过对其增加大文件管理模块的方式对其改进。实验结果表明，集成该模块后的系统能够满足大文件存储的需求，并且与基于磁盘的存储系统相比，具有明显的性能优势。

## 1.相关工作

### 1.1分布式文件系统

分布式文件系统的出现解决了大规模数据的存储问题。目前，比较具有代表性的分布式文件系统有Google的GFS[9]（Google File System）及其开源实现HDFS[10]（Hadoop[11] Distributed File System）和Lustre[12]等。GFS是Google云计算核心技术体系的底层，它为相关技术(MapReduce计算模型、BigTable[13]数据库等)的实现提供了有效的支撑；HDFS是Apache基金会所开发的Hadoop体系的数据存储管理的基础，它是一个高度容错的文件系统，能检测和应对硬件故障，能够部署在低成本的通用硬件之上，通过流式数据访问来提高系统吞吐率，适合对延迟要求较低的应用程序；Lustre是一个开源的基于对象存储的分布式文件系统，特别适合用来解决海量数据的存储问题，最多可以支持上万个客户端，PB级的存储量，具有较好的安全性、扩展性和管理性，同HDFS相比，Lustre文件系统一般运行在高性能计算机系统之上，对硬件设备要求相对较高。

### 1.2缓存技术

缓存技术的出现在一定程度上缓解了磁盘的性能瓶颈问题：通过将经常需要访问的数据调入到内存中。在理想的情况下，通过这种方式便可获得接近于内存的访问速度和接近于磁盘的存储容量。随着数据量和内存容量的不断增大，越来越多的数据被放到内存中，这种做法会削弱缓存带来的益处[14]。另外，缓存命中率也是影响系统性能的重要因素，它和缓存替换策略有着直接的联系。不同缓存替换策略的应用场景有所不同，如果缓存策略选择不当，可能会导致内存和外存之间进行频繁的数据交换，很容易使缓存失效率达到10%以上，这将大大降低系统的性能。在运用缓存技术时，如何保证内外存之间的数据一致性，也是一个特别需要注意的地方。所以，使用缓存技术可以在一定程度上缓解因外存I/O瓶颈引起的系统性能问题，但是随着数据量的增大，其中存在着管理复杂，低效，内存利用率低的问题。

### 1.3内存云（RAMCloud）

近年来，基于内存的存储系统层出不穷，如RAMCloud、Memcached[15]和Redis[16]等。RAMCloud是由斯坦福大学的John Ousterhout教授带领开发的一个基于内存的分布式键值存储系统。通过高速网络，将数据中心的每一台服务器的可用内存整合起来，进行统一管理，形成一个可存储海量数据的存储系统。将所有的数据都放到内存中，从而突破了磁盘吞吐量小、读写速度慢的瓶颈。尽管内存的价格相对较高，但是使用这种方式能大大提高系统性能。这种架构在处理数据密集型应用（社交网络、电商和搜索引擎等）中具有明显的优势。为了提高内存利用率，RAMCloud采用了日志结构方式来管理内存[17]。RAMCloud目前还只是一个原型系统，仅提供了一个简单的键值存储（key-value）数据模型，应用场景非常有限，不能存放大文件，仅适合存储大规模小数据对象（通常为几十KB或者更小）。

## 2 系统设计和实现

### 2.1设计目标

本文所描述的系统—RCDSS，其主要设计目标是：在满足大文件存储需求的同时，解决传统的分布式文件系统中普遍存在的I/O性能瓶颈问题，力图给应用提供快速的文件服务；在保证系统性能的前提下，同时兼顾系的统通用性和可靠性，尽可能地降低对上层应用带来的影响。

### 2.2系统架构

改进后系统主要分两部分：RAMCloud原生存储系统系统和Librcdfs模块。一个RAMCloud集群由一个协调服务器（Coordinator）和若干个存储服务器（Storage Server）组成。协调服务器的功能与HDFS中的NameNode结点的功能类似，负责管理集群的配置信息，它不参与数据的读写操作，所以不会成为系统的性能瓶颈。存储服务器的主要作用是提供数据存储服务，它主要包括两个模块：主模块（Master Module）和备份模块（Backup Module）。主模块负责管理存储服务器的内存，并处理客户端的读写请求。备份模块则负责将本存储服务器和其他存储服务器的数据备份到本地磁盘，它不会与客户端进行数据交互。只有当存储服务器崩溃后需要数据恢复的时候才会用到磁盘中的备份数据。另外，为了保证系统的可靠性，需要对协调服务器中的集群配置信息进行实时备份（如保存到Zookeeper分布式存储系统中），并启动若干个备用协调服务器。RAMCloud集群的架构如图1所示，

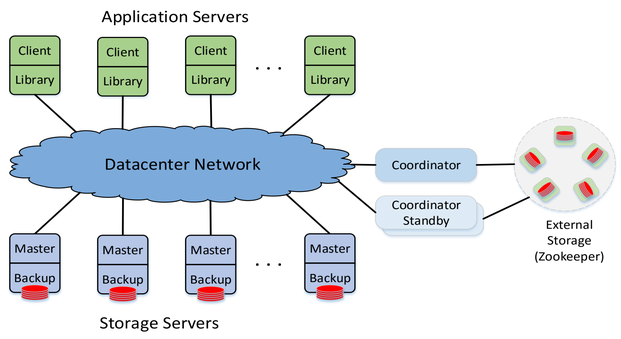


图1 内存云架构

Librcdfs模块位于RAMCloud集群和客户应用程序之间，是用户访问RCDSS的入口。它为用户提供了包括connect()，close()，put()，get()和delete()等在分布式文件系统中常用的接口。同时它还负责大文件存储策略的具体实现：当用户调用put()接口将大文件上传至RCDSS时，它把此文件拆分成一系列小数据块（默认1MB），分别存储到RAMCloud中；当用户通过get()接口从RCDSS中下载文件时，它从RAMCloud中读取数据块，并按照一定的顺序合并成一个大文件。RCDSS系统的总体架构图如图2所示，

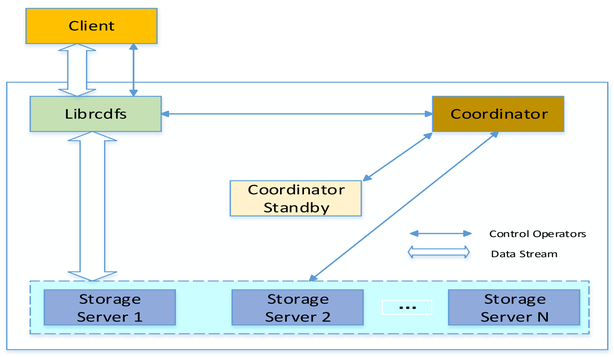


图2 RCDSS系统总体架构

在RCDSS中，用一张表来表示一个文件，表以文件的绝对路径命名。每张表由若干个固定大小数据块和一个元数据块组成。每个数据块与一个文件片段相对应，大小默认为1MB。元数据块存放在表的末尾，其中存放着文件的属性：文件所有者、创建和最后访问时间、数据块个数以及文件校验值等信息。元数据块类的定义如下：

```
class Property{
private:
int part_count;   //文件块数
int part_size;    //文件块大小，默认1M
string ctrate_time;//文件创建时间
stirng last_visit_time;//文件最后访问时间
string group;   //文件所有者所属的组
string owner;          //文件所有者
string checksum;       //文件校验值
/*其它数据成员*/
public:
int getCount();                 //返回文件块数
int getBlockSize();     //返回文件块的大小
/*其它成员函数*/
};
```

一个文件在RCDSS中的具体组织方式如图3所示，

/usr/foo/file_test

| Key_0 | File_Block_0 |
|---|---|
| Key_1 | File_Block_1 |
| Key_2 | File_Block_2 |
| ⋮ | ⋮ |
| Key_N | File_Block_N |
| Property | File_Property |

图3 RCDSS中文件的组织方式

### 2.3 提供的文件服务

RCDSS为用户提供了常用的对文件的操作方法，如文件的上传，下载和删除等。RCDSS提供的接口与HDFS类似，通过调用connect()，close()，put()，get()，和delete()等库函数来对文件进行操作。本部分重点描述文件的上传和下载功能的详细设计和实现。

文件的上传和下载功能通过Librcdfs提供的put()和get()接口来实现。上传和下载文件的详细流程如图4、图5所示，

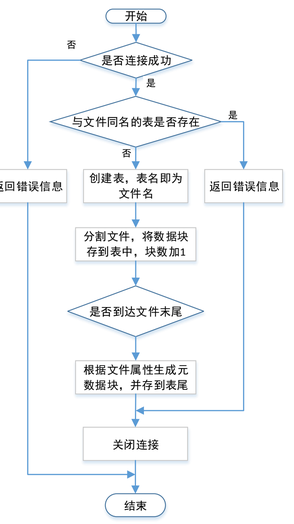


图4 上传文件至RCDSS流程图

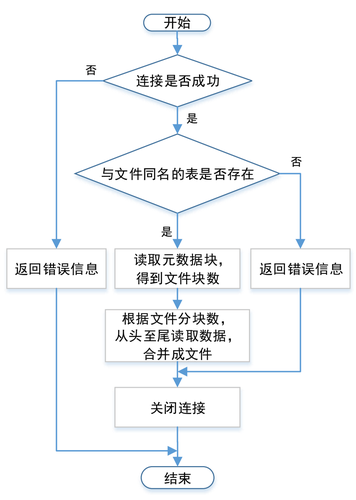


图5 从RCDSS中下载文件流程图

## 3实验结果和分析

RCDSS主要是面向拥有着高速网络的服务器集群而设计的，较高的网络带宽是使RCDSS的性能得以最大程度发挥的保证。为了模拟足够高的网络带宽，消除网络带宽对实验结果的影响，本实验在单机模式下进行，即客户端与RCDSS服务器在同一台机器上。

为了测试系统在大文件存储方面的可用性，测试中使用的文件大小在8MB至2GB之间，通过多次将文件写入和读取，然后统计时间平均值的方式来对系统进行测评。针对不同大小的文件，RCDSS和HDFS读写性能对比如图6，图7所示，

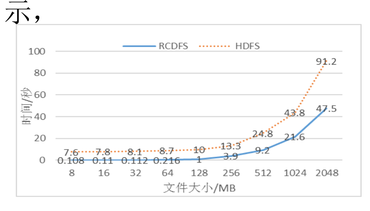


图6 RCDSS和HDFS读性能对比

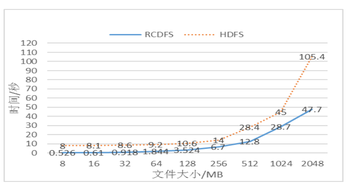


图7 RCDSS和HDFS写性能对比

由以上测试可见，RCDSS和HDFS相比，在读写方面均有着显著的性能优势。特别是当文件较小时，RCDSS的读写性能是HDFS的数十倍左右，出现这种情况主要是因为：HDFS系统本身调度所需的时间占了较大比重的缘故，而RCDSS系统调用时间远少于HDFS。当文件较大时（1GB及以上），系统本身调用时间在整体时间中的比例变小，RCDSS较HDFS的性能提升比例趋于稳定：RCDSS的读取和写入速度均是HDFS的2倍左右。

## 4 结束语

本文首先介绍了设计和实现本系统的背景以及动机，然后介绍了其设计目标，系统架构以及一些主要功能的具体实现。最后对本系统进行了性能测试，发现其相比HDFS有着显著的性能优势。显然，本系统现在还只是一个原型系统，在接下来的时间里，我们还有许多工作要做。随着对该系统的进一步开发，我们将会对系统的功能进行完善，在可靠性、并发控制和安全性等方面进行进一步的研究。后续工作准备把RCDSS集成到HDFS中当作缓存系统来使用，从而达到提高系统效率的目的。