·体系结构与软件技术· 文章编号: 1000-3428(2016)05-0080-05 文献标识码: A 中图分类号: TP391

基于可变长分块的分布式文件系统设计与实现

许艳艳[1],雷迎春[2],龚奕利[1]

(1. 武汉大学计算机学院,武汉 430072; 2. 中国科学院计算技术研究所,北京 100080)

摘 要: 分布式文件系统受传统磁盘文件系统影响,大多使用固定大小的对象或者块来组织和管理文件。定长的分块不适合随机写或插入写,开销大且性能差,但典型的用户约有 25% 的文件操作是随机写。针对上述现状,提出一种基于内容的可变长文件分块方法,使用拉宾指纹算法对文件进行分块,并根据文件的内容标识文件块。为更准确地指定写类型,提高写性能,给出与 POSIX 语义兼容的写接口。通过修改 Ceph 实现一种新的分布式文件系统 VarFS,提供文件的可变长分块并支持新的写接口。实验结果表明,由于减少网络数据传输量,VarFS 在随机写延迟和带宽消耗量上比 Ceph 减少了 1 个 ~2 个数量级。

关键词: 固定长分块; 可变长分块; 分布式文件系统; 随机写; 元数据服务器

中文引用格式: 许艳艳,雷迎春,龚奕利. 基于可变长分块的分布式文件系统设计与实现[J]. 计算机工程,2016,42(5):80-84,101.

英文引用格式: Xu Yanyan,Lei Yingchun,Gong Yili. Design and Implementation of Distributed File System Based on Variable-sized Chunk[J]. Computer Engineering,2016,42(5):80-84,101.

Design and Implementation of Distributed File System Based on Variable-sized Chunk

XU Yanyan[1],LEI Yingchun[2],GONG Yili[1]

(1. Computer School,Wuhan University,Wuhan 430072,China;
2. Institute of Computing Technology,Chinese Academy of Sciences,Beijing 100080,China)

【Abstract】 Inherited from traditional disk file systems,most Distributed File Systems(DFS) organize and manage files based on objects or chunks of fixed sizes,which is expensive and works poorly for random writing or inserting. But approximately 25% of file operations from a typical user are random writing. In order to change this status,this paper puts forward a file blocking method based on variable-sized contents,and it uses Rabin fingerprint algorithm to block the file and identify it according to its contents. A new writing interface which is semantic compatible with POSIX is proposed to accurately specify the writing type and improve writing performance. It presents a novel DFS named VarFS to implement all the design proposed above through modifying the Ceph. Experimental results show that VarFS can reduce the necessary reading and writing back data of update operation,which consequently alleviates amount of data transfer,accordingly it can achieve 1 ~ 2 orders of magnitude less latency and bandwidth consumption than Ceph on random writing and bandwidth consumption.

【Key words】 fixed-sized chunk; variable-sized chunk; Distributed File System(DFS); random writing; Metadata Server(MDS)

DOI: 10.3969/j.issn.1000-3428.2016.05.014

1 概述

基于云的文件系统和文件存储服务,如 Dropbox[1],Evernote[2] 等为用户提供了一个可靠和便捷的方法,使用户备份个人或工作文件到云存储服务器上,并在需要时及时更新这些文件。同时,云存储和基于云的文件系统已经成为适用于各种应用程序的基础层。例如,Facebook Messages(FM) 把它的数据存储在分布

基金项目: 国家"863"计划基金资助项目(2013AA12A301); 国家自然科学基金资助项目(61100020); 华为公司创新研究计划基金资助项目。

作者简介: 许艳艳(1989-),女,硕士,主研方向为分布式文件系统; 雷迎春,副研究员; 龚奕利(通讯作者),副教授。

收稿日期: 2015-04-20 修回日期: 2015-05-13 E-mail: yiligong@whu.edu.cn

式数据库中(HBase[3] 来源于 BigTable[4])，而 HBase 是基于分布式文件系统的(HDFS[5] 来源于谷歌文件系统(Google File System，GFS) [6])。另外，桌面虚拟化技术[7] 的成熟，在很大程度上依赖于大规模文件系统的发展。

现有的分布式文件系统，如 GFS，HDFS，把大文件分割成固定大小的块进行存储，通常是64 MB。还有一些文件系统，如 Ceph[8] ，Lustre[9] ，将文件分割成较小的定长对象，通常是 8 MB。这些系统文件一经创建，很少被更新，因此，比较适合读取和追加写操作，但这种假设对用户个人文件和许多其他的应用并不成立。一个典型用户大约有 25% 的文件访问是随机写。FM 有 90% 左右的文件小于15 MB，根据文献 [10] 的显示，随机 I/O 占有很高的比例。文献 [11] 显示 VDI(Vitrual Desktop Infrastructure) 存储负载很高，占到总 I/O 操作的 65% 。

对于通用 POSIX 写语义，如果用户在文件的中间插入一个字节或者几个字节，那么至少需要将待插入的字节和插入点之后的所有数据写回磁盘。在分布式环境中，用户和存储服务器间的网络通常是不稳定的，读取或者写回大量不必要的数据，不仅消耗客户端和服务器端的处理器资源，而且降低网络吞吐量。造成这种 I/O 操作性能差的关键原因是: (1) 文件系统缺乏足够的信息来识别不必要传输的数据; (2) 即使文件大部分内容未改变，固定大小分块方式会强制进行重新映射。在 POSIX 语义的基础上，为随机写给出新的接口，即rwrite( ) 和 prwrite( ) ，允许用户删除和增加具体的对象。这个接口可以执行覆盖写、追加写、插入和删除，使得文件系统可以更好地处理写操作。同时，新接口易与标准的 POSIX 接口兼容。

本文提出一种新的文件系统——VarFS，该系统包括 3 个关键设计点: (1) 把文件映射到对象; (2) 元数据组织; (3) 读写接口的设计与实现。VarFS 支持新的文件分块方式和随机写接口，并使用文件的内容来标识对象，提供数据去冗余功能。在 Ceph 的基础上实现了 VarFS，并对 VarFS 进行全面的文件操作测试。

2　VarFS 设计

新的 rwrite( ) 和 prwrite( ) 接口使得应用程序可以更准确、更灵活地控制写操作，其设计目标是让随机写仅仅影响直接修改的块或者有限的相邻块，从而减少数据的传输和移动。

客户端在提交数据之前，把数据重新划分，根据新接口的特性，将新对象与已存在的对象进行比较，判断对象是否存在。如果不存在，把对象写回磁盘，否则跳过此次写操作。例如，刚好在文件的开头删除一个字节，那么除了包含这个字节的第一个对象发生改变外，其他的对象都不变，因此，只需将第一个对象写回存储设备。

由于 VarFS 与标准的 POSIX 接口是兼容的，因此应用程序也是可兼容的。未修改的应用程序可以从新的设计中获益，但是得到的性能提升会差一些。如在随机写操作中，应用程序使用传统的读写逻辑，把文件从修改位置之后的全部数据写回数据服务器。新提出的系统 VarFS 通过对新对象进行检查，对于未修改的对象不写回(通常仅有很少的对象被真正修改) ，减少不必要的操作，提高系统的性能。

当然，不是所有的应用都可以从中获益。最差的情况是，应用程序从来不在文件中进行插入和删除操作，或者仅仅是在文件尾部进行追加操作与进行相等字节替换操作。对于通常的写负载环境，VarFS 可以节省大量带宽。

VarFS 的系统架构如图 1 所示。数据分为 3 层: 文件层，对象层和存储层。文件分割为可变大小的对象，对象被组织成扁平的结构，以文件的形式存储在本地文件系统中。VarFS 的一个关键设计是将文件依据其内容进行分块。这样做有 3 个优点: (1) 通过对象内容保持对象边界，例如文件中对象几乎不会受其他对象修改的影响。(2) 提供了一个便捷的方式判断客户端对象是否被修改。(3) 基于内容划分对象可以利用文件之间的共性，例如在线文件的自动保存、多次编译生成的对象文件和版本控制系统中的多版本文件。

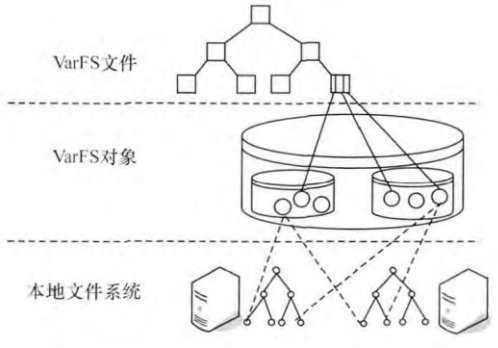

图 1　VarFS 系统架构

使用拉宾指纹算法[12] 对文件按内容进行分块。拉宾指纹是一个多项式，它是数据模上一个预定的不可约多项式。每个对象有一个哈希值，它通过 SHA-1 哈希函数[13-14] 计算得到。

元数据是描述文件系统组织和结构的数据，通

常包括目录内容、文件属性、文件块指针、物理磁盘的组织和状态等信息。每个文件都有一个 inode 用来记录与其相关的元数据信息。在 VarFS 中,元数据包括对象元数据和文件元数据,对象的元数据主要包括全局唯一的对象 ID、对象大小、位置、SHA-1 值和对象引用次数。文件元数据除了基本的属性外,还包括指向对象的指针。但为了更好地提高性能,VarFS 把定位对象位置的信息也存储在文件的 inode 中,包括 < 对象 ID,对象大小,对象位置 > 。

3 VarFS 实现

VarFS 是基于开源的分布式文件系统 Ceph 实现的。通过修改 Ceph 必要的部分来实现核心设计,使得系统可以工作。VarFS 大约修改了 6 000 行C ++ 代码。

图 2 显示了 VarFS 的整体实现架构。应用程序通过 Kernel 和 FUSE,然后调用读写接口访问 VarFS。客户端解释应用程序的请求,调用相应的进程。文件元数据服务器负责存储文件相关的元数据和回应相应的请求。对象元数据服务器以键值对的形式存储对象信息,并处理对象的增加、删除和去重查询。对象存储服务器(Object Storage Device,OSD) 是处理对象真正被存储的地方,使用 XFS 作为 OSD 的本地文件系统。

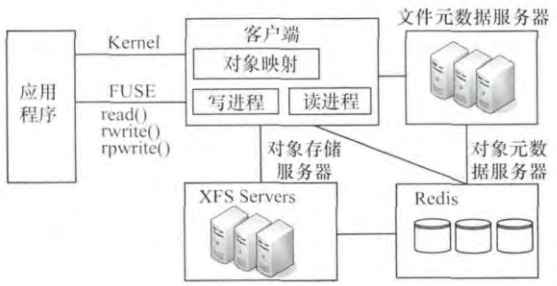

图 2 VarFS 的整体实现架构

在 Ceph 中,每个对象都有全局唯一的对象 ID,由 inode 编号(每个文件的 inode 有一个全局唯一的编号) 和对象编号(文件中的每个对象有一个唯一的序列号) 组成。CRUSH (Controlled Replication Under Scalable Hashing) 通过对象 ID 定位一个对象。在 VarFS 中,一个对象可以被多个文件包含,所以对象 ID 应该是独立于文件的全局唯一的编号。为了充分利用 Ceph 的源代码,VarFS 的对象 ID 由第一个创建此对象的文件 inode 编号和对象编号组成,后续使用到此对象的文件,均使用这个对象 ID,并将对象的版本号增加 1。利用 CRUSH 可以根据 VarFS 的对象 ID 定位到一个对象。

对于客户端和元数据服务器集群来说,对象存储集群可以被看作是一个单独的逻辑对象存储。VarFS 使用 Ceph 的 RADOS (Reliable Autonomic Distributed Object Store) 来处理对象放置、对象副本、扩展集群、故障检测和恢复。VarFS 除了在每个对象的末尾增加了一个标识对象的 SHA-1 值外,OSDs 的对象存储方式与 Ceph 是一样的。主要修改了 Ceph 的 2 个组件: 客户端和元数据服务器。

3.1 客户端实现

客户端主要做了如下 3 个部分的修改:

(1) 给应用程序提供了新的接口 random write。

(2) 增加了一个用于给文件进行按内容分块的模块,即拉宾指纹算法模块。

(3) 实现了读写处理函数,包括与文件元数据服务器和对象元数据服务器进行通信的处理(OSD 通信的方式与 Ceph 是一致的) 。

VarFS 客户端使用尽量大的缓存来存储可变对象和对象相关信息,用来快速识别对象。对象 ID、对象 SHA-1 值以及对象的位置信息被缓存在客户端。一旦有新的对象生成,首先查找本地缓存,如果有相同的对象,那么将新对象标记为 clean,并把新的文件映射信息发送到文件元数据服务器。被标记为 clean 的对象不需要将其发送到 OSDs。如果在本地没有匹配的对象,那么将新对象的 SHA-1 值发送到数据库进行查找。

3.2 元数据服务器实现

VarFS 的元数据服务器(Metadata Server,MDS) 集群管理文件的元数据信息。对象元数据信息主要记录与对象相关的属性信息、对象的查询回复、删除和增加对象的请求。

MDS 基于 Ceph 的元数据服务器实现。除了普通的属性(如文件字节大小、文件属主等) 外,VarFS 的 inode 还包括文件映射到对象的对象列表,列表项包括部分对象元数据信息。

元数据最重要的功能是根据文件的偏移量找到具体的对象。VarFS 元数据服务器将给定的偏移量,在对象列表中找到相应的对象。如果文件太大,被映射到大量的对象时,元数据的定位操作可能需要很长时间。在这种情况下,可以使用索引指针来加快查找速度。

4 实验评估

分布式文件系统的性能受到很多因素的影响。VarFS 主要是通过读写测试其性能。

4.1 实验配置

测试集群中所有的机器使用 8 核 2.13 GHz Xeon E5606 CPU,45 GB 内存,CentOS 6.3(内核版本

是 3.2.51）和 XFS 本地文件系统。机器运行在千兆网络上，使用 TCP 进行通信。集群由 4 台主机组成，Monitor 和 MDS 安装在一台机器上，Monitor 负责监控整个集群的状态变化，MDS 负责集群的元数据管理；OSD 安装在一台机器上，运行一个 OSD 实例，主要负责集群的数据和元数据的存储；另外两台机器是 Client，主要用来产生负载，多个 Client 实例共享一台物理主机，每台机器可以运行上百个 Client 实例。VarFS 客户端可以通过 FUSE（Filesystem in Userspace）[15] 挂载的方式进行访问。

为了避免机械硬盘和预取机制影响访问速度，使用 RAMDisk [16] 将数据存储在 OSD 的内存中，并禁用客户端的预取机制。

使用 IOzone [17] 对 VarFS 进行读写性能的测试，其中，IOzone 是一款被广泛用来进行文件系统读写性能测试的工具。除了标准的读写操作外，另外实现了随机写测试功能。所有的结果都是 10 组测试求平均值，每次运行都是随机选择一个 4 GB 大小的文件。为了确保比较的公平性，使用相同的实现和优化方法来比较 VarFS 和现存的文件系统。当从设计效率，特别是从数量级的性能提升方面，定性分析比定量分析更加有意义。

4.2　读、初始写和覆盖写的性能测试

首先测试系统的读、初始写和覆盖写性能。由于 OSD 使用的 RAMDisk，所有的数据都在 OSD 内存中，因此在实验结果中，顺序读和随机读的性能几乎是一样的，所以，仅仅显示了顺序读的结果。初始写是指创建一个文件，然后向文件的末尾不断写入内核缓存大小的数据，直到写完整个文件。标准的 Posix I/O 写（即覆盖写）用来和随机写进行区分。覆盖写操作和初始写是类似的，所以，只显示了初始写的性能。

在 VarFS 中，依据文件的内容，文件被分割成可变的对象。首先分析平均对象大小和请求大小对 VarFS 性能的影响，图 3 显示了 VarFS 在不同的请求大小和平均对象大小的情况下初始写的性能。随着请求大小的增加，吞吐量也增加，但是当请求大小增至 8 MB（选择内核缓存页为 8 MB）时，吞吐量开始保持稳定。请求大小越小意味着要执行更多的追加写操作去写完一个 4 GB 的文件。反过来，也就意味着需要多次获取最后一个对象，进行更多计算去映射对象。在达到内核缓存限制之前，请求的数据会从 FUSE 经过内核一次性发送给 VarFS，而一个 16 MB 的请求会被分成 2 个 8 MB 的请求发送给 VarFS，因此，16 MB 的请求大小与 8 MB 是相似的。

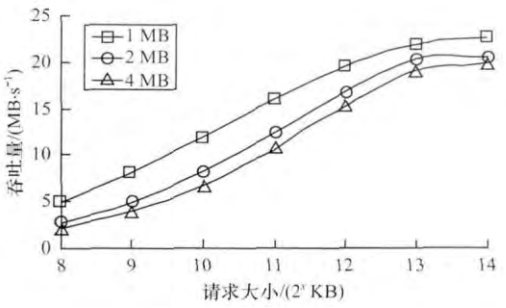

图 3　在不同请求大小和平均对象大小情况下初始写的性能

通过实验发现，平均对象越小，系统的性能越好。这是因为越小的平均对象，获取的最后一个对象的大小就相对越小，那么需要传输的数据量就越少。另一方面，对象越小，产生的元数据就越多，那么读写相同的数据量需要更多的元数据访问和对象操作。考虑到应用环境和性能，下面的实验选择平均对象大小等于 1 MB。

一个新的设计不增加额外的读开销是很重要的，所以，VarFS 的读性能不应该比 Ceph 的读性能差太多。图 4 为 2 种系统的实验结果，通过实验证明了 VarFS 和 Ceph 的读吞吐量是接近的。对于初始写，由于文件分块而引入额外的开销，导致 VarFS 的性能比 Ceph 差。图 5 显示了 Ceph 和 VarFS 的初始写的性能。但随着引入随机写操作，这个开销会被分摊而变小。

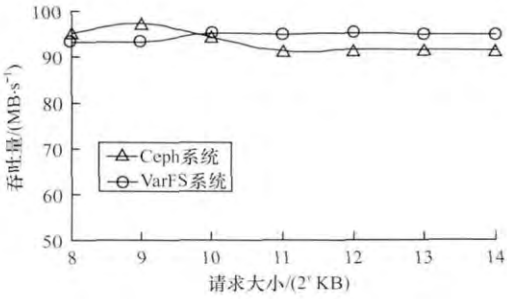

图 4　VarFS 与 Ceph 的读吞吐量测试结果

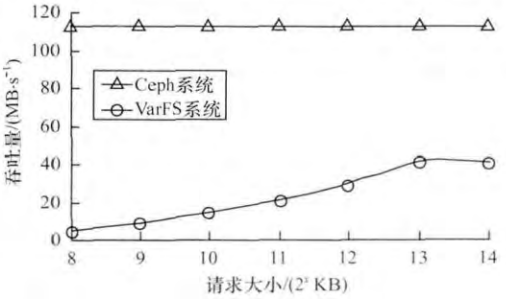

图 5　Ceph 和 VarFS 的 Initial Write 吞吐量测试结果

4.3　随机写

随机写操作是在文件的相同位置删除一些数据之后，再插入另外一些数据。除了插入的数据外，

Ceph 需要重写删除点之后的所有文件数据，而 VarFS 只需要重写删除点影响到的边界对象。

图 6 显示了 VarFS 和 Ceph 的随机写延迟。Ceph 的随机写延迟随着文件大小的增加而急剧上升，然而 VarFS 却保持在一个稳定的状态，它的随机写延迟保持在 0.5 s 以内。

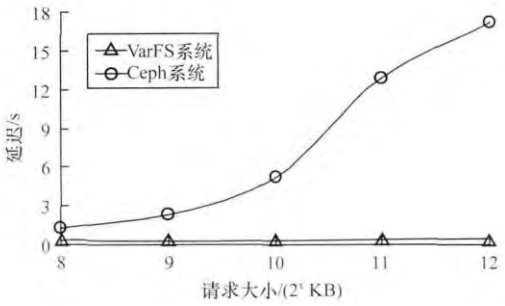

图 6 VarFS 和 Ceph 的随机写延迟测试结果

图 7 显示了 VarFS 的随机写延迟和数据传输量占 Ceph 的比例。在最差的情况下，VarFS 的延迟约占 Ceph 的 25%，数据传输量约占 5%。当文件较大时，VarFS 的性能表现得更好。在最好的情况下，VarFS 的随机写延迟仅占 Ceph 的 2%，数据传输量仅占约 0.5%。通过随机写测试发现，VarFS 较 Ceph 的性能有了较大提升。

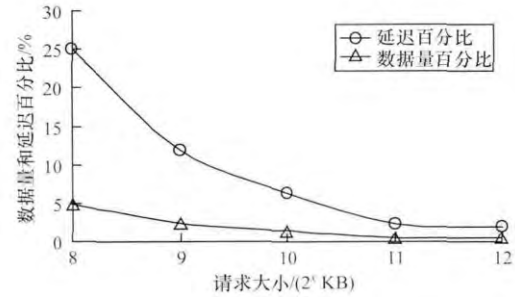

图 7 随机写延迟和数据传输量中 VarFS 占 Ceph 的比例

4.4 对象存储服务器性能

为了测试 VarFS 的 OSD 性能，实验中使用多个客户端实例，尽可能快地向系统发送插入和随机写请求。

Ceph 和 varFS 的每个 Client 进程均在 1 GB 文件的任意位置插入一块 8 MB 的数据。对于随机写，*mbytes*(删除的数据大小)和 *nbytes*(插入的数据大小)是 8 MB 内的随机值。图 8 显示了随机写的测试结果。VarFS 的吞吐量随着客户端进程数的增加而增加，当网络成为瓶颈时吞吐量达到峰值，而 Ceph 的吞吐量保持不变。即使在很少客户端的情况下，Ceph 也会传输大量的数据使得网络达到饱和，吞吐量基本保持不变。在两者的吞吐量都达到峰值的情况下，VarFS 约是 Ceph 的 70 倍。

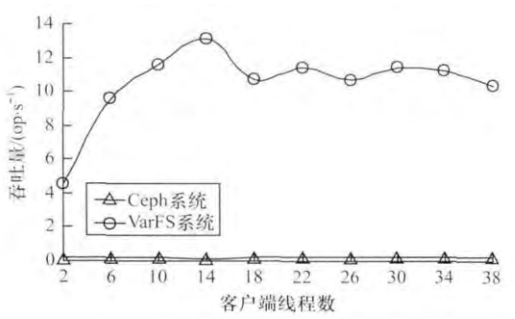

图 8 Ceph 和 VarFS 的 OSD 随机写吞吐量测试结果

5 结束语

本文给出一个新的接口来提高传统文件系统的写效率，并提出一个基于可变对象的文件系统——VarFS，该系统把文件按内容映射到对象。实验结果表明，结合新的接口，VarFS 可以减少由写语义引起的对象读和不必要的数据对象传输。同时，文件中间内容发生变化不会影响后续的对象。对象映射是相对稳定的，还可以用于重复数据删除。基于 Ceph 实现的 VarFS，比 Ceph 的随机写吞吐量和延迟的性能提升了 50 倍。另外，基于拉宾指纹的分块机制是比较耗时的操作，很多技术提高了拉宾指纹计算所产生的计算延迟，例如硬件加速、延迟写、并行计算。作为未来工作的一部分，下一步将探索更有效的基于内容的映射机制。

参考文献

[1] Dropbox [EB/OL]. [2015-03-20]. http://www.dropbox.com.
[2] Evernote [EB/OL]. [2015-03-20]. http://www.evernote.com.
[3] HBase [EB/OL]. [2015-03-20]. http://hbase.apache.org/.
[4] Fay C, Jeffrey D, Sanjay G, et al. Bigtable: A Distributed Storage System for Structured Data [C]//Proceedings of IEEE Conference on Operating Systems Design & Implementat. Washington D. C., USA: IEEE Press, 2006: 205-218.
[5] Shvachko K, Kuang H, Radia S, et al. The Hadoop Distributed File System [C]//Proceedings of IEEE Symposium on Mass Storage Systems & Technologies. Washington D. C., USA: IEEE Press, 2010: 1-10.
[6] Ghemawat S, Gobioff H, Leung S. The Google File System [C]//Proceedings of ACM Sigmetrics Inter-national Conference on Measurement & Modeling of Computer Systems. New York, USA: ACM Press, 2003: 29-43.
[7] 黄 华. 桌面虚拟化技术的现状及未来发展研究[J]. 福建电脑, 2009, 25(9): 38-39.

(下转第 101 页)

[4] Raniwala A, Tzi-cker C. Architecture and Algorithms for an IEEE 802. 11-based Multi-channel Wireless Mesh Network [C]//Proceedings of the 24th Annual Joint Conference of IEEE Computer and Communications Societies. Washington D. C., USA: IEEE Press, 2005: 2223-2234.

[5] 邱振谋,姚国祥,官全龙,等. 多信道无线 Mesh 网络的多播信道分配算法 [J]. 计算机工程,2011,37(6): 107-109.

[6] 邓雪波,王小强,陈　曦,等. 基于 QoS 和吞吐量公平的信道分配算法 [J]. 计算机工程,2012,38(6): 89-91.

[7] 张彤芳. MIMC 网络中基于多径路由的跨层信道分配技术研究 [D]. 南京: 南京理工大学,2014.

[8] Liu Lu, Cao Xianghui, Cheng Yu, et al. Energy-efficient Capacity Optimization in Wireless Networks [C]//Proceedings of INFOCOM'14. Washington D. C., USA: IEEE Press, 2014: 1384-1392.

[9] 赵志峰,郑少仁. Ad Hoc 网络体系结构研究 [J]. 电信科学,2001,17(1): 14-17.

[10] Lin C R, Gerla M. Adaptive Clustering for Mobile Wireless Networks [J]. IEEE Journal on Selected Areas in Communications, 1997, 15(7): 1265-1275.

[11] Baker D J, Ephremides A. The Architectural Organization of a Mobile Radio Network via a Distributed Algorithm [J]. IEEE Transactions on Communications, 1981, 29(11): 1694-1701.

[12] 王海涛,张学平. Ad Hoc 网络中的分簇算法 [J]. 数据通信,2003,32(4): 32-35.

[13] 杨卫东. 考虑节点能量状态的 Ad Hoc 网络分簇算法 [J]. 计算机工程,2010,36(12): 119-122.

[14] 王　雪,王　晟,姜爱国,等. 无线传感网络中的分簇融合决策方法 [J]. 控制与决策,2007,22(11): 1208-1212.

[15] 肖　磊,符云清,钟明洋,等. 兼容弱连通簇的 Ad Hoc 网络分簇算法 [J]. 计算机工程,2013,39(6): 107-110.

[16] Bahl P, Chandra R, Dunagan J. SSCH: Slotted Seeded Channel Hopping for Capacity Improvement in IEEE 802. 11 Ad-hoc Wireless Networks [C]//Proceedings of the 10th Annual International Conference on Mobile Computing and Networking. New York, USA: ACM Press, 2004: 216-230.

[17] Neely M J, Modiano E, Rohrs C E. Dynamic Power Allocation and Routing for Time-varying Wireless Networks [J]. IEEE Journal on Selected Areas in Communications, 2005, 23(1): 89-103.

[18] Alicherry M, Bhatia R, Li L. Joint Channel Assignment and Routing for Throughput Optimization in Multi-radio Wireless Mesh Networks [C]//Proceedings of the 11th Annual International Conference on Mobile Computing and Networking. New York, USA: ACM Press, 2005: 58-72.

[19] Kodialam M, Nandagopal T. Characterizing the Capacity Region in Multi-radio Multi-channel Wireless Mesh Networks [C]//Proceedings of the 11th Annual International Conference on Mobile Computing and Networking. New York, USA: ACM Press, 2005: 73-87.

[20] Naveed A, Kanhere S S. Cluster-based Channel Assignment in Multi-radio Multi-channel Wireless Mesh Networks [C]//Proceedings of the 34th IEEE Conference on Local Computer Networks. Washington D. C., USA: IEEE Press, 2009: 53-60.

编辑　刘　冰

(上接第 84 页)

[8] Weil S A, Brandt S A, Miller E L, et al. Ceph: A Scalable, High-performance Distributed File System [C]//Proceedings of the 7th Symposium on Operating Systems Design and Implementation. Washington D. C., USA: IEEE Press, 2006: 307-320.

[9] Donovan S, Symposium L, Kleen A, et al. Lustre: Building a File System for 1,000-node Clusters [C]//Proceedings of Linux Symposium. Washington D. C., USA: IEEE Press, 2003: 9-17.

[10] Harter T, Borthakur D, Dong S, et al. Analysis of HDFS Under HBase: A Facebook Messages Case Study [C]//Proceedings of the 12th USENIX Conference on File and Storage Technologies. Washington D. C., USA: IEEE Press, 2014: 199-212.

[11] Siegel E H, Okasaki M E, Kumar P, et al. Coda: A Highly Available File System for a Distributed Workstation Environment [J]. IEEE Transactions on Computers, 1990, 39(4): 447-459.

[12] Rabin M O. Fingerprinting by Bandom Polynomials, TR-CSE-03-01 [R]. Cambridge, USA: Harvard University, 1981.

[13] Standard S H. FIPS 180-1 Secure Hash Standard [S]. Department of Commerce, National Institute of Standards and Technology, 1995.

[14] 林雅榕,侯整风. 对哈希算法 SHA-1 的分析和改进 [J]. 计算机技术与发展,2006,16(3): 124-126.

[15] Szeredi M, Szeredi M. FUSE: Filesystem in Userspace [C]//Proceedings of the Usenix Winter Technical Conference. Washington D. C., USA: IEEE Press, 2009: 159-166.

[16] 龚　辉,徐学洲,曹荣禄. 基于 Ramdisk 的全内存式 Linux 系统的设计与实现 [J]. 微机发展,2005,15(4): 75-77.

[17] IOzone [EB/OL]. [2015-03-20]. http://www.iozone.org.

编辑　索书志