高技术通讯 2012 年 第22 卷 第10 期:1035 ~ 1040

doi:10.3772/j.issn.1002-0470.2012.10.006

基于分布式索引和目录聚合的海量小文件存储研究①

马 灿②*** 孟 丹* 熊 劲*

(*中国科学院计算技术研究所高性能计算机研究开发中心 北京 100190)

(**中国科学院研究生院 北京 100049)

摘 要 针对海量小文件访问问题的挑战,提出了用基于分布式索引和目录聚合的分布式文件系统——超虚拟文件系统(HVFS)来管理数十亿个小文件的方法,以支持高并发、高吞吐、低延迟的访问。重点讨论了目录索引、目录存储问题,提出了利用可扩展哈希索引来降低延迟、提高扩展性,利用日志结构和列存储的目录聚合来提高吞吐率的方法。测试结果表明,HVFS 的存储性能能够线性扩展,82 节点上峰值创建速度接近百万每秒,与 GIGA + 相比有 200% 以上的提升,小文件 I/O 性能与 OrangeFS 相比有 60 倍以上的提升,充分验证了分布式索引和目录聚合方法的有效性。

关键词 小文件,海量存储,分布式索引,目录聚合,分布式文件系统

0 引 言

新兴互联网应用展示了其与众不同的数据存储和访问特征。除了高效存储大文件(达到 PB 级)之外,还能够高效支持百万到上亿用户并发访问较小的文件(典型的在 1MB 以内)。例如,社交网站 Facebook 保存的图片超过600 亿张;电子商务网站"淘宝"保存的图片超过 200 亿张,图片的平均大小仅为 15KB。随着微博、社交网站的普及,并发存取大量小文件(微博、评论、图片、音视频)的需求还在不断增长。可以预见,在未来几年之内,支持 Web 服务的可扩展存储系统必须支持对海量小文件的高吞吐、低延迟访问。然而,传统的分布式文件系统和并行文件系统并不能满足上述需求,因为它们都将设计重心放在对大文件 I/O 访问带宽的优化上,而不是小文件访问延迟的优化上。通常而言,由于目录项、索引节点和数据分离存储,每次访问都需要多次访问底层设备,小文件访问的代价非常高。

对小文件访问而言,现有的分布式文件系统和近年来的研究系统存在着两个缺陷。一是目录索引结构低效。文件系统 Lustre[1]、并行虚拟文件系统(PVFS)[2]等使用 B + 树来索引单个目录,但 B + 树无法在有数亿目录项的大目录中很好扩展。正是由于单个目录元数据索引结构的低效,用户通常选择把文件分散存储在多层目录中以提高性能,这进一步加大了路径查询和管理的开销。二是文件存储映射低效。在多数分布式文件系统中,如 Ceph[3]、GIGA +[4]等,文件被映射为本地文件系统的一个或者多个文件,称之为 1 – *N* 映射。任何文件访问都需要再次经历本地文件系统的查询处理,因而加大了访问延迟。近年来,小文件存储问题受到了广泛重视,Carns[5], Kuhn[6]等人提出了合并提交和消减元数据的方法,有效缓解了 PVFS 的小文件访问问题;Liu[7]等人提出了在 Hadoop 分布式文件系统(HDFS)外部建立索引、聚合小文件的方法,减轻了 HDFS 的小文件压力。Facebook 和淘宝研发了各自的定制存储系统 Haystack[8]和 TaobaoFS 来存储百亿级别的图片文件。上述系统,或针对特定文件系统,或使用特定接口、一致性模型,很难应用于其他系统。针对上述缺陷,本文提出,基于分布式索引和目录聚合的分布式文件系统——超虚拟文件系统(hyper virtual file system, HVFS)来管理小文件,采用分布式可扩展哈希索引目录中的文件,基于目录聚合存储小文件,以降低访问延迟和提高并发小文件访问的吞吐率。

① 863 计划(2009AA01A129)资助项目。

② 男,1984 年生,博士生;研究方向:分布式文件系统和容错系统;联系人,E-mail: macan@ ncic.ac.cn

(收稿日期:2011-09-13)

1 分布式目录索引

1.1 问题分析

在 POSIX 文件系统中,目录表和索引节点表通常是分离的以支持两种不同类型的查询。通过路径名的查询由目录表处理,而通过索引节点号的查询则由索引节点表处理。使用两个不同的索引表导致了文件创建、查询延迟上升,从而影响了小文件的创建、查询性能。此外,常用的 B + 树索引方法也很难有效扩展到海量环境下。

针对上述问题,我们为目录提出了一种更灵活的模型(如图 1 所示),称作 xTable,消除了多个索引表带来的问题,并将索引复杂度由 B + 树的 $O(\log_t N)$ 降低到 $O(1)$。目录表通过主键索引,主键是文件名与目录随机扰动值的哈希值。同时,每个文件赋予一个唯一标识符,称作通用唯一标识符(universally unique identifier,UUID),我们用它作为文件的索引节点号。通过这种方法,基于文件名的访问可以通过计算哈希值寻址,而基于索引节点号的访问则还需要同时提供文件的哈希值以确定元数据位置。此外,基于 xTable 模型的目录索引结构还可以更好地支持文件扩展属性、标签以及多维数据。

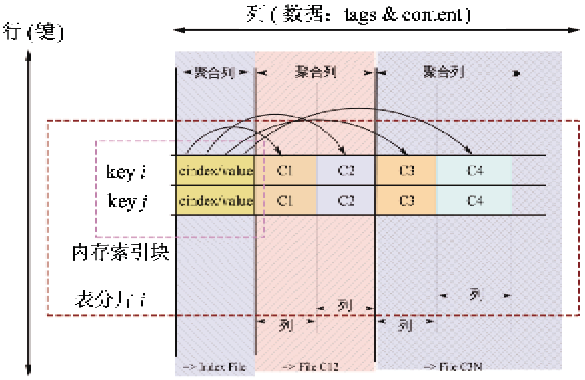

图 1 目录表的数据模型及存储划分方法

为了利用多个结点的并行性,超虚拟文件系统(HVFS)将单个目录划分为多个分片,通过采用可扩展哈希[9]的方法,即使随着负载增长,访问的时间复杂度也总是 $O(1)$ 的。

1.2 基于分布式可扩展哈希的多级目录索引

为了更好地支持 Web 应用丰富的语义检索要求,目录表采用了与 Google BigTable[10] 相似的面向列的数据模型,如图 1 所示。目录表通过主键索引,并可以拥有多个列。每列保存了不同的信息,如元数据、数据、属性或者标签。列可以在目录项的生命周期内任何时候加入或删除。每次访问都需要提供特定的行列对 < row, col > 以供定位具体的格(cell)。固定数量的行和列组成一个表分片(table slice,TS)。表分片是缓存与存储之间交互的最小粒度。多个列组成聚合列(combined column)并保存为一个文件(如 Index File, File C12, File C3N 等)以利用共同查询的局部性。此外,表分片是结构化的,拥有哈希表可以快速定位表中每一行。

为了支持大规模的小文件并发访问,目录表的每个访问都会被自动、动态划分到缓存集群中。目录表的请求接口包括以下几种原子操作:create, lookup, update, delete 和 list。每个目录表是随着新项的加入而动态增长的,支持最多 2^{64} 个目录项。系统中的目录表和请求被分布到所有的缓存服务器上以利用更多的内存资源,提高并发吞吐率。随着表中新项的不断加入,表分片会逐渐变大,当超过一定阈值时,表分片会自动分裂以利用更多的资源提高并发度。

为了加快定位过程,每个表都有两级索引来寻址:A 用来定位缓存和存储服务器;B 用来定位精确的行。索引 A 完成从表分片到服务器的寻址。表分片到缓存和存储服务器是两个独立的映射。为了提高系统的负载均衡能力,容忍结点故障,在映射时采用了一致性哈希[11]的方法。索引 B 完成在表分片内部的定位。传统的文件系统,如 Ceph、GIGA + 和 SkyFS[12],分片内部的索引依赖于本地文件系统,效率低下。HVFS 的表分片内部是高度结构化的,通过位图区管理内部的空闲空间,哈希表索引快速定位每一项,和读写锁区域来同步并发访问。

如图 2 所示,通过计算文件名的哈希值获得了 V,该值仅用来定位,而精确区别每一项仍依赖于唯一标识。将 V 分为两个部分:Vh 和 Vl。其中,Vl 用于索引 B 的寻址,表分片中的哈希表可以快速回答该项是否存在。Vh 用于索引 A 的寻址,定位该项所属的表分片和服务器。例如,通过 Vh 查询客户端本地的位图确定该项所属分片 TS(该结果可能是不精确的,比如 TS 已经发生了分裂,但是 TS 必然拥有完整的分裂历史,结果可以逐步修正),然后用 TS 在哈希环 0 中查询找到缓存服务器 X,最后将该请求发往 X。如果该分片已经加载到服务器 X 的内存中,则可以直接服务;否则,还需要进一步到哈希环 1 中查询以找到存储服务器 Y。

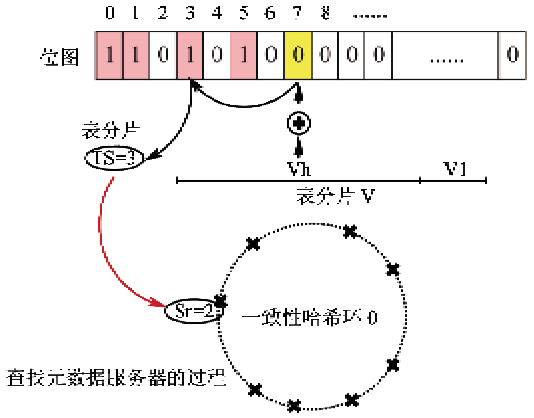

图 2　通过 Key 定位缓存、存储服务器的过程

HVFS 的目录表分布与定位工作基于 GIGA + 的方法,并在其基础上做了如下的增强。缓存服务器在收到不精确的客户端请求时,不再直接向客户端返回故障和位图更新,而是将请求路由到更精确的缓存服务器上,以此减少请求处理的延迟。由于每一跳都会更加接近正确的服务器,同时可扩展哈希是指数分裂的,因此路由次数最多为 $\log_2(N)$(N 为目录表分片的个数)跳。此外,在正确的缓存服务器处理完请求之后,应答会直接返回给客户端,同时携带最新的位图信息。这样,客户端后续的请求就无需再次经历多跳。

2　目录聚合存储

2.1　问题分析

分布式文件系统中的 1 – N 映射方法为每个文件创建了一个或多个本地文件系统的映射文件。任何一次小文件访问都需要在本地文件系统再次通过路径名进行查找,引入了多次 I/O 操作,从而带来了不必要的开销,增加了访问延迟。

针对上述问题,我们提出了基于目录的聚合方法,将同一目录的所有文件数据、元数据等信息聚合存储,避免了一般分布式文件系统采用分离、独立文件存储映射所带来的外部索引开销,消除了 1 – N 映射问题。基于目录聚合后,文件访问无需再通过路径名查找即可获得准确的数据偏移位置,这样不仅减小了小文件的访问延迟,同时也缩减了外部文件系统的元数据压力。

此外,对于小文件而言,将元数据和数据合并存储能够有效利用访问的空间局部性。因此,在需要时,我们采用列存储的方法将文件元数据和数据,或者共同访问的多个列合并在一起以提高局部性。更进一步,为了利用时间局部性,提升 I/O 写效率,新写入的数据和元数据都以追加的方式顺序写入到存储文件中。

2.2　基于日志结构与列存储的目录聚合存储

由于 1 – N 映射问题,小文件存储不但浪费空间,还影响 I/O 性能,因此我们将单目录内的所有小文件数据、元数据等信息聚合存储在多个追加文件中。

如图 3 所示,每个目录在本地文件系统上表示为一个独立的目录。文件 MD(MetaData)记录了目录表的元数据,如当前服务器负责的表分片区间等。文件 Range 保存了表分片的指针偏移。这些元数据文件通过 mmap 的方法映射到内存中,以降低访问延迟,减少内存耗用。文件 TSS(table slice store)中记录了提交的表分片,而文件 CS(columnar store)等保存了合并后的列数据,列数据的索引保存在表分片中。追加文件 TSS 和 CS 通过在内存中保持最后一个块以合并多个追加操作,充分利用磁盘带宽。

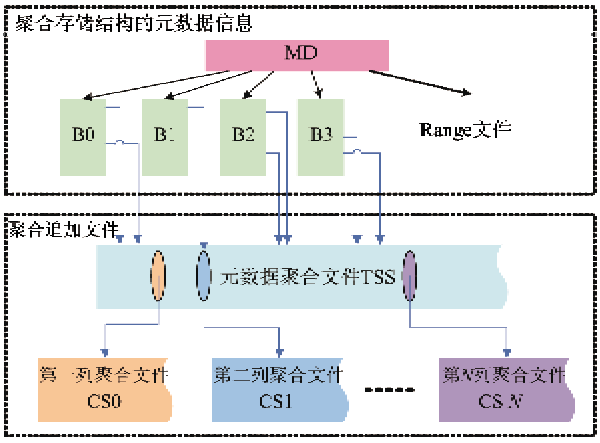

图 3　基于日志结构和列存储的目录存储结构

为了利用时间和空间的局部性,表分片根据其所在表的不同写入到不同追加文件 TSS 中(per-table append-only)。这种日志结构的顺序写入模式可以最大化地利用磁盘带宽。仅在事务的提交阶段,元数据 Range 才被更新以指向新写入的分片。注意到新分片每次都写入到新位置而没有覆盖旧分片,因此,部分写或者未提交的快照不会导致数据不一致。

3　性能评价

我们在中等规模的集群上测试了 HVFS 文件系统。配置为 4 核 1.6GHz Xeon CPU,4GB 内存,千兆

以太网，两层交换机互联，SATA 7200RPM 磁盘，Linux Kernel 2.6.16。磁盘的基本 I/O 性能如表 1 所示，其中随机读写在 128MB 的数据集上完成，读写大小在 0 到 1000 字节（读写粒度越小，随机性能越差）的区间内均匀分布。

表 1 测试集群的磁盘基准性能

ST31000524NS SATA Disk	
顺序写	42.15 MB/s
顺序读	59.12 MB/s
随机写	682 IOPS
随机读	3812 IOPS

3.1 目录表性能与分析

我们在 82 个节点上测试了目录表的性能。每个结点都运行服务器和客户端。每个客户端并发执行 160 个线程，在同一个目录中做 2000000 次操作，包括插入、查找、数据写入、数据读取和删除，共计产生1.64亿个文件。文件的长度在 0－1000 字节的区间中均匀分布，总数据集的大小是100.5GB。图 4 显示了目录表在高并发的压力下依然有着出色的聚合性能，聚合创建速度接近百万每秒。

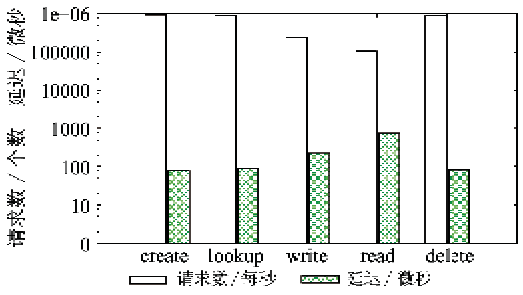

图 4 目录表的基本性能

对于小文件 I/O 而言，从图 4 可以看到，平均单个结点的每秒读写（I/O）操作次数（IOPS）为 2984，是表 1 中硬盘随机写性能的 4 倍。这是因为 HVFS 将并发的随机写转换为顺序写，从而带来了吞吐率提升。

小文件的并发读，由于客户端和存储服务器引入的两次随机化，很难转换为顺序读。因此，我们最好能够做到每次读取仅发出一次 I/O。如图 4 所示，平均单个结点的 IOPS 为 1293。这个结果比表 1 中的差，其原因与基准测试中小数据集太小（128MB）和并发度（单线程）有关。

测试结论 1：HVFS 目录表能够在高并发（＞10000）环境下达到低延迟（＜1ms）的小文件访问性能。

3.2 迁移开销分析

由于采用了动态分裂的目录表管理方法，在遇到大量文件创建时，可能会发生大规模的分片迁移。但是，通过构造合适的哈希方法来组织分片的分裂，我们可以有效地控制分片迁移的发生。为了评价迁移对系统性能带来的影响，我们在 82 个节点上执行了同目录内总计 1 亿文件的创建测试。

如图 5 所示，在创建的过程中，创建的性能与本地分裂的速率呈正相关。与此同时，远端分裂（即需要迁移分片的分裂）随着系统的执行则呈现了减少的趋势。仅在前 20s 发生了较大规模的需要迁移分片的分裂，而在此之后需要迁移的分片个数大大下降，分片更倾向于在本地存储。

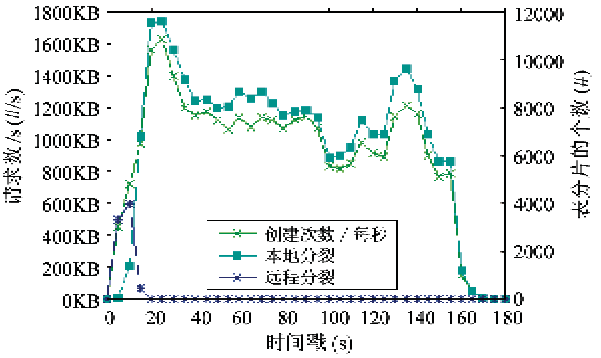

图 5 创建过程中的分片分裂

测试结论 2：分片迁移对系统性能影响很小。

3.3 HVFS 单结点基准测试与分析

针对单节点小文件的 I/O 性能，我们对比了 ext3（开启 dir _ index 选项，以提高元数据性能）和 ReiserFS。测试使用 Postmark 程序，在同一个目录下，完成 1000000 次文件的创建、读写和删除，以及 1% 的事务操作。测试文件的大小在 1B－1000B，1KB－10KB，10KB－100KB，100KB－1MB 的四个区间内正态分布。

如图 6（a）所示，在较小的文件区间内，HVFS 的运行时间相对较长，而随着文件的增大，HVFS 的运行时间逐渐取得优势。这是因为 HVFS 是基于用户空间文件系统（FUSE）用户态客户端，其余文件系统是核心态的，因此，HVFS 的元数据访问延迟较大。当文件大小增大时，元数据访问时间占总时间的比重下降，因此 HVFS 体现出更好的性能，约为 ext3 的 2 倍。图 6（b）展示了在有 100 万文件的目录中进行事务操作时，HVFS 与 ext3、ReiserFS 文件系统的

对比。在较小的文件区间上,HVFS 有着更高的事务性能。如在 1 – 1000 字节的区间上,HVFS 的事务性能大约是 ext3 的 5.6 倍。随着文件的增大,数据 I/O 时间占据了较大的比重,三种文件系统的事务性能逐渐接近。

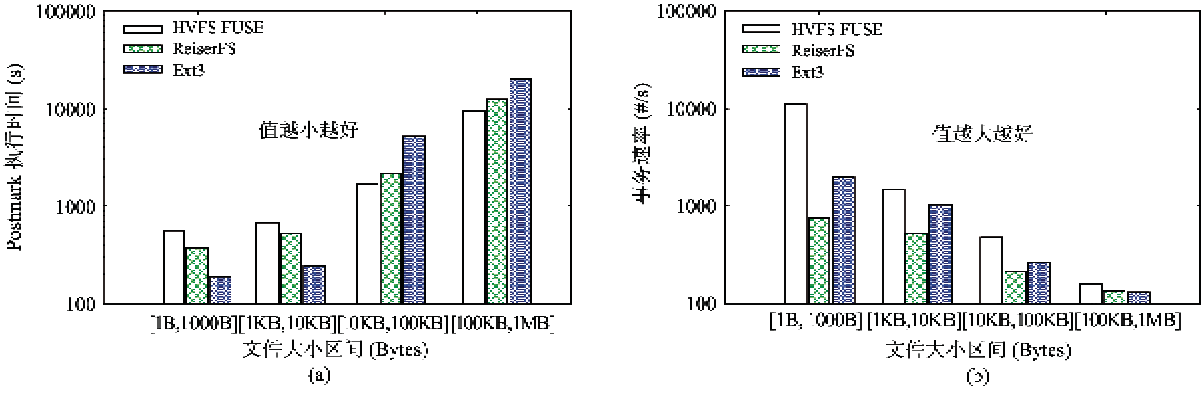

图 6　HVFS 与 ReiserFS、Ext3 在文件区间 0B – 1000B, 1KB – 10KB, 10KB – 100KB, 100KB – 1MB 上的 I/O 性能对比

测试结论 3:单节点 HVFS 在大规模小文件读写上有着明显超越本地文件系统的优势。

3.4　HVFS 多结点对比测试与分析

HVFS 支持多元数据服务器,因此,我们对 HVFS 元数据服务的扩展性和小文件 I/O 的扩展性进行了测试和分析。测试在 32 个服务器和 32 个 FUSE 客户端上完成。每个服务结点同时运行了元数据服务和存储服务,两者通过 TCP 套接字进行通信。

首先,我们对比了 HVFS 与 GIGA + 等存储系统的元数据性能和扩展性。客户端分布在 32 个节点上,每结点运行 4 个 mdtest 进程,服务器的规模从 1 增大到 32。图 7 中 GIGA + 、HBase 的对比数据来自于文献[4]中,Ceph 的对比数据来自于文献[3]中。HBase 用来模拟 Google 下一代文件系统 Colossus 采用 BigTable 来管理元数据的方法。由图 7 可知,随着元数据服务器个数的增长,HVFS 表现出了良好的线性扩展性。同时,由于采用了更好的表分片分裂机制和结构化、聚合的存储方法,在同等硬件配置下,HVFS 的性能约为 GIGA + 性能的 2 倍以上(CPU 频率归一化之后)。此外,与 HBase、Ceph 的对比也说明 HVFS 的性能远好于 HBase 和 Ceph。

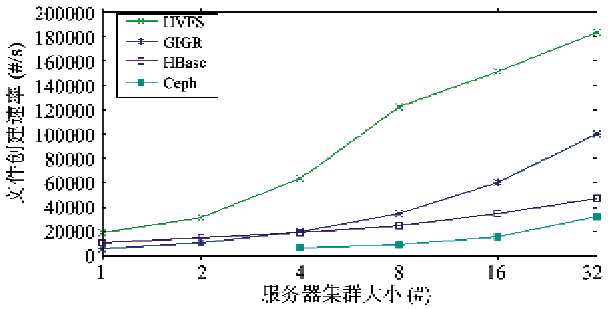

图 7　与 GIGA +, Ceph, HBase 的元数据性能、扩展性对比

其次,我们对比分析了 HVFS 与并行文件系统 OrangeFS(即现在的 PVFS2 主分支)的 I/O 性能。如图 8 所示,我们测试了两个系统创建小文件的速率和写入带宽。试验表明,对于小文件创建而言,HVFS 的创建速度和文件写入带宽大约为 OrangeFS 的 60 倍。此外,随着文件大小的增大,HVFS 的小文件写入带宽呈现了增长的态势,当文件大小为 512KB 时,聚合小文件的写入带宽达到了 1GB/s。

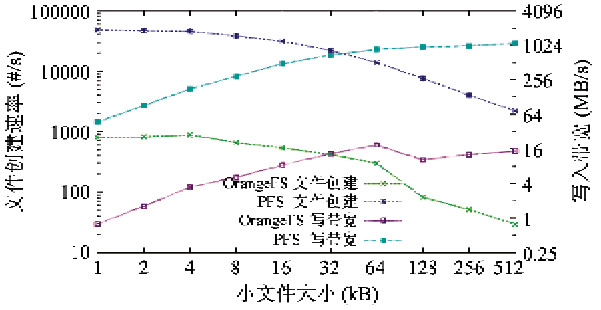

图 8　HVFS 与 OrangeFS 的小文件读写性能对比

测试结论 4:HVFS 的多元数据服务性能有着很好的线性扩展性,并能够提供领先的小文件创建和聚合写入速度。

4　结　论

随着互联网应用的发展,数据中心内部大小文

件的比重逐渐发生了变化,小文件的访问问题日益受到重视。本文提出了基于分布式索引和目录聚合的分布式文件系统 HVFS。研究了分布式可扩展哈希索引文件系统目录的方法,基于目录聚合存储小文件的方法。实验表明上述方法可以有效降低小文件的访问延迟,提高并发度,达到透明、自动的扩展性。文件系统元数据性能为 GIGA + 的 2 倍以上,小文件 I/O 性能为 ext3 的 5 倍以上,并随着结点的增加线性扩展。未来的工作是与更多的分布式文件系统对比,测试更多的文件大小区间以及动态可重构和容错子系统。也准备探索文件系统的语义查询的接口,使得用户可以方便地通过插件和流处理的方法完成业务逻辑。

参考文献

[1] Donovan S, Huizenga G, Hutton A, et al. Lustre: Building a File System for 1,000-node Clusters. In: Proceedings of the Linux Symposium, Ottawa, Canada, 2003. 1-10

[2] Carns P, Ligon W, Ross R, et al. PVFS: a parallel file system for Linux clusters, In: Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, USA, 2000. 317-327

[3] Wei S, Brandt S, Miller E, et al. Ceph: a scalable, high-performance distributed file system. In: Proceedings of 7th Symposium on Operating Systems Design and Implementation, Seattle, USA, 2006. 307-320

[4] Patil S, Gibson G. Scale and concurrency of GIGA +: file system directories with millions of files. In: Proceedings of the 9th USENIX Conference on File and Storage Technologies, Berkeley, USA, 2011. 177-190

[5] Carns P, Lang S, Kunkel R, et al. Small file access in parallel file systems, In: Proceedings of International Symposium on Parallel and Distributed Processing Systems, Washington, DC, USA, 2009. 1-11

[6] Kuhn M, Kunkel J, Ludwig T. Directory-based metadata optimizations for small files in PVFS. In: Proceedings of the 14th International Euro-Par Conference on Parallel Processing, Las Palmas de Gran Canaria, Spain, 2008. 90-99

[7] Liu X H, Han J Z, Zhong Y Q, et al. Implementing WebGIS on hadoop: a case study of improving small file I/O performance on HDFS. In: Proceedings of 2009 IEEE International Conference on Cluster Computing, New Orleans, USA, 2009. 1-8

[8] Beaver D, Kumar S, Li H C, et al. Finding a needle in haystack: Facebook's photo storage. In: Proceedings of the 9th USENIX Symposium on Operating System Design and Implementation, Vancouver, Canada, 2010. 1-8

[9] Fagin R, Nievergelt J, Pippenger N, et al. Extendible Hahsing: a fast access method for dynamic files. *ACM Transaction on Database Systems*, 1979, 4(3): 315-344

[10] Fay C, Jeffrey D, Sanjay G, et al. Bigtable: a distributed storage system for structured data. In: Proceedings of the 7th USENIX Symposium on Operating System Design and Implementation, Seattle, USA, 2006. 205-218

[11] Karger D, Lehman E, Leighton T, et al. Consistent Hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web. In: Proceedings of the 29th Annual ACM Symposium on Theory of Computing, El paso, USA, 1997. 654-663

[12] Xing J, Xiong J, Sun N H, et al. Adaptive and scalable metadata management to support a trillion files. In: Proceedings of the Conference on High Performance Computing Networking, Storage, and Analysis, Portland, USA, 2009. 1-11

Research on enormous storage for small files based on distributed indexing and directory aggregation

Ma Can*,**, Meng Dan*, Xiong Jin*

(*National Research Center for Intelligent Computing Systems, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

(**Graduate University of Chinese Academy of Sciences, Beijing 100049)

Abstract

To solve the problem of access to a vast amount of small files created in Web service, a method for managing billions of small files using the hyper virtual file system (HVFS), a distributed file system based on distributed indexing and directory aggregation, is proposed to provide both high throughput and low latency file access. The measures of using distributed extendible hash indexing to improve system scalability, using log structure format and columnar storage to exploit temporal and spatial locality in directory aggregation, are also presented. The evaluation indicates that the HVFS can scale linearly and obtain 1 million file creations per second with 82 nodes. Compared to GIGA +, it can improve more than 200%. Compared to the small file I/O performance of OrangeFS, it can improve more than 6000%.

Key words: small files, enormous storage, distributed indexing, directory aggregation, distributed file system