对象存储系统中的柔性对象分布策略

王 芳 张顺达 冯 丹 曾令仿

(华中科技大学 计算机科学与技术学院, 湖北 武汉 430074)

摘要: 为了使对象存储系统在处理不同大小文件时兼顾并行性和负载均衡,提出一种能够结合哈希算法和分片算法的优点,同时尽量避免其缺点的柔性分布算法.柔性分布算法将大小文件的边界值界定为512 Kbyte,小文件直接映射成一个对象并使用哈希策略映射到一个设备中;大文件分割成多个对象,分别放置在不同的设备里.实验结果显示:柔性分布算法在不同规模的系统中开销最小,且性能受设备数增加的影响较小.

关 键 词: 对象存储系统; 对象分布策略; 负载均衡

中图分类号: TP302.1 文献标识码: A 文章编号: 1671-4512(2007)03-0046-03

Hybrid object allocation policy for object storage systems

Wang Fang Zhang Shunda Feng Dan Zeng Lingfang

(College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

Abstract: In order to satisfy both load balance and parallel operation when dealing with file requests of different size in object storage system, a hybrid mapping algorithm is proposed which is characterized by hashing and fragment-mapping approaches and avoids their shortcomings. The bound of big or small file was set as 512 Kbyte. Small files are mapped though hashing function directly to one object storage device (OSD), while big files are mapped to multiple objects and stored onto OSDs. Experiment results indicate that this hybrid algorithm has the lowest workload and lest influence when device number increasing.

Key words: object storage systems; object mapping strategy; workload balance

面向对象的存储系统将文件视为对象的集合,这些对象分布在具有自我管理功能的智能设备OSD中.对象放置到不同的设备中可以使系统具有更高的容量、吞吐量、可靠性和可扩展性[1].目前对于类似Lustre, GFS, AFS, Coda和GPFS[2~6]的分布式文件系统的层次管理、可扩展性和可靠性的研究很多,但对于提高对象放置策略效率的研究则相对较少.

对象分布策略主要有两类:第一类是采用哈希函数将一个文件映射为一个对象并放置到一个设备中;第二类采用分片映射策略,将文件数据平均分配到多个设备中.结合以上两者的优点提出一种柔性分布策略,即:小文件直接映射成一个对象并使用哈希策略映射到一个设备中;大文件分割成多个对象,分别放置在不同设备里.这样既利用了设备并行带来的性能好处,又减少了不必要的文件分割所引入的系统开销.下文将讨论大小文件的界限、文件分片数和设备选择等影响因素.

1 算法设计

1.1 大小文件的边界值

文献[1]中研究人员开发对象存储设备的文件系统——OBFS时,通过对分布式文件系统负

收稿日期: 2005-12-26.

作者简介: 王 芳(1972-),女,副教授;武汉,华中科技大学计算机科学与技术学院(430074).

E-mail: wangfang@mail.hust.edu.cn

基金项目: 国家重点基础研究发展计划资助项目(2004CB318201);国家自然科学基金资助项目(60303032).

载数据特征值的分析, 发现负载中约有 85 % 的对象大小是 512 Kbyte, 15 % 的对象小于 512 Kbyte, 由此在 OBFS 设计中将大小文件的边界值界定为 512 Kbyte. Roselli 等[7] 跟踪过类似的系统, 研究发现 60 % ~ 70 % 的数据为小于 512 Kbyte 的文件. 基于上述结果, 本研究将小于 512 Kbyte 的文件当作小对象, 其余作为大对象处理.

1.2　将一个文件映射成对象

OSD 的数目与并行度之间存在复杂关系. 设备越多则传输通道越多, 有利于并行传输. 但是建立连接的系统开销对系统整体性能有负面影响, 数据同步和将文件片段重新拼装成文件也将花费不少 CPU 时间.

可以用下面的公式描述这种关系,

$$T_p/T = (na + b/n + c)/(a + b), \qquad (1)$$

式中: T_p 为并行传输文件的总时延; T 为串行传输文件的总时延; n 是文件分片对应的设备数; a 是开销因子; b 是消息传输时间, b= 消息大小/网络带宽; c 是对象间的协调时间, 包括同步、校验等, 是关于 n 的函数, 但其随 n 的变化非常小, 可以近似地认为 c 是一个常数.

网络上总的传输延迟为发送方开销、接收方开销、消息传输时间以及飞行时间之和. 根据文献[7~ 10], 互连网络类型不同(SAN, LAN 或者 WAN 等), 飞行时间和传输时间也不尽相同. WAN 中飞行时间相对较长, 发送方和接收方开销可被忽略. 可把发送方开销、接受方开销和飞行时间简化为一个开销因子, 即式(1)中的 a.

na 表示假设连接 n 个设备的发送方开销是连接单个设备的 n 倍, 而 b/n 则表明 n 个设备并行传输的带宽是单个设备的 n 倍.

式(1)进一步变形为

$$\frac{T_p}{T} = n\frac{a}{a+b} + \frac{1}{n}\frac{b}{a+b} + \frac{c}{a+b}. \qquad (2)$$

上式的倒数 T/T_p 是并行传输相对于串行传输的加速比, 加速比最大等价于使 T_p/T 最小. 关注将一个文件映射为多少个对象可以获得最大加速比, 即 n 对 T_p/T 的影响, 找到最佳的 n.

开销因子 a 只有在消息非常大时才会有较大的开销, 这里假设其不受消息长度影响. 可以取一种典型的情况: 带宽为 1 000 Mbit/s, 开销因子是 80 μs. 式中的 $c/(a+b)$ 因子对 n 并没有影响, 在讨论中暂时忽略它.

令 $F(n) = na/(a+b) + (1/n)[b/(a+b)]$, 对其求导并令 $dF(n)/dn = 0$, 可以得到 $n = \sqrt{b/a}$. 由于 $d^2F(n)/dn^2 > 0$, $F(n)$ 在 $n = \sqrt{b/a}$ 处有最小值, 因此

$$T_p/T \geqslant 2\sqrt{ab}/(a+b) + c/(a+b). \qquad (3)$$

在 $n = \sqrt{b/a}$ 时得到传输最大加速比. 在传输时间 b 很大, 或开销因子 a 很小时, n 将变得很大. 这表明若传输时间很小, 则没有必要将文件分片并行传输. 在千兆网环境下, 当文件很大时才有必要并行传输.

图 1 显示各种文件大小(大于 512 Kbyte) 下

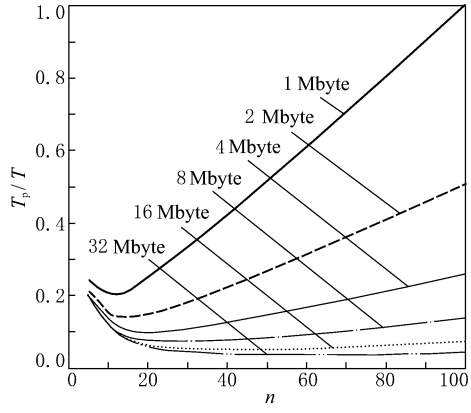


图 1　时间比 T_p/T 和文件最佳分片数 n 的关系

T_p/T 与文件分片数 n 的关系. 对于 1 Mbyte 和 2 Mbyte 的文件, 当 $n \geqslant 10$ 时 T_p/T 的变化率已经不明显了. 对于 4 Mbyte, 8 Mbyte 和 16 Mbyte 的文件, 当 $n \geqslant 20$ 时 T_p/T 的变化不明显. 对大于 32 Mbyte 的文件, $n \geqslant 40$ 时 T_p/T 几乎不随 n 变化了. 因此当文件大小分别为 1~ 2 Mbyte, 4~ 16 Mbyte 和 32~ 1 000 Mbyte 时, 文件最佳分片数 n 分别设为 10, 20 和 40.

1.3　OSD 设备的选取

大型的对象存储系统拥有众多性能各异的 OSD 设备. 随机选择合适的 OSD 是一种实现简便而且能够保持系统负载均衡的较好方法, 但它不能保证充分利用最优设备. 将设备按某些参数进行排序, 例如设备速度、空闲空间或忙程度等, 选择前 n 个设备. 这样保证了当前状态最好的设备优先使用, 但需要确保在众多设备中进行排序的开销不会过大而影响系统性能.

冒泡排序算法中, 在 N 个对象中选取前 n 个对象所需的时间复杂度为

$$\sum_{i=1}^{n}(N - i) = \frac{2N - (n+1)}{2} \times n, \qquad (4)$$

可以得出 n 对时间复杂度的影响比 N 更显著. 因为 n 比较小(10, 20 或者 40), 而且元数据服务器通常都配备大容量内存和高性能 CPU, 所以排序算法不会对系统整体性能有明显的影响.

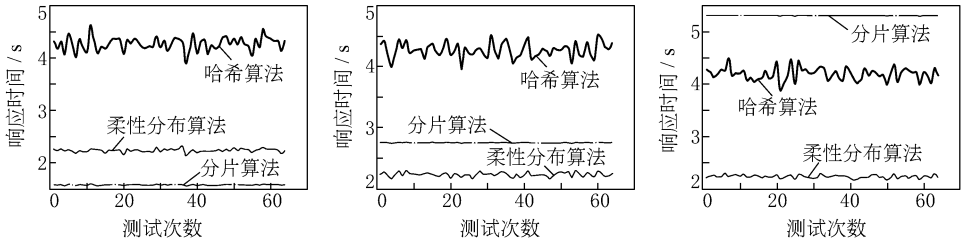
2 仿真实验与分析

2.1 仿真环境

所有的实验程序运行于 2.4 GHz Intel Celeron CPU, 512 Mbyte RAM 的 PC 机上, 操作系统是 Red Hat Linux 9.00. 通过 Matlab 按指数分布生成一个表示文件大小的随机数组, 然后将对象分布算法应用于这些文件, 得出相应传输时延. 假设开销因子为 80 μs, 网络带宽为 1 000 Mbit/s, 对小文件采用哈希算法, 大文件采用分片算法, 排序选择适当的存储设备. 按系统规模不同, 分别仿真包含 16, 32 和 64 个 OSD 的系统.

2.2 实验结果分析

图 2 描述了包含 16, 32 和 64 个 OSD 的系统中, 分别采用哈希算法、分片算法和柔性对象分配策略时的传输总时延.



(a) 16 OSD 系统 (b) 32 OSD 系统 (c) 64 OSD 系统

图 2 三种算法在不同规模系统的传输延迟

仿真过程中 Matlab 生成的文件大小符合 LLNL 所描述的负载分布. 在 16OSD 系统中, 分片算法的总时延均值是三种算法中最小的. 但是当设备数目增加时, 发送和接收端开销增大, 分片算法性能会逐渐下降. 在 32OSD 系统中分片算法的总时延均值已经超过柔性对象分配策略. 而在 64OSD 系统中分片算法的总时延均值已经是三种算法中最大的了. 哈希算法的总时延 4.0~ 4.6 s 之间, 柔性对象分布算法则时延 2.0~ 2.3 s, 它们在三个不同规模系统中的变化并不大. 柔性对象分布算法比哈希算法快是因为充分利用了设备间的并行性. 从总体上看, 柔性对象分布算法比其他两种算法更加稳定, 并且在对象存储系统拥有大量 OSD 时表现最好. 仿真结果证实三种算法中的柔性对象分布算法在规模扩展变化的系统中开销最小, 并且其性能受设备数增加的影响较小.

参 考 文 献

[1] Wang Feng, Brandt S A, Miller E L, et al. OBFS: a file system for object-based storage devices[C]// 21st IEEE/12th NASA Goddard Conference on Mass Storage Systems and Technologies. Los Alamitos: IEEE Comput Soc, 2004: 283-300.

[2] Satyanarayanan M, Kistler J J, 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.

[3] Schmuck F, Haskin R. GPFS: a shared-disk file-system for large computing clusters[C]// Proceedings of the FAST'02 Conference on File and Storage Technologies. Berkeley: USENIX Assoc, 2002: 231-244.

[4] Roselli D, Lorch J R, Anderson T E. A comparison of file system workloads[C]// Proc of the 2000 USENIX Annual Technical Conference. Berkeley: USENIX Assoc, 2000: 41-54.

[5] Soltis S R, Ruwart T M, O'Keefe M T. The global file system[C]// Proceedings of the 5th NASA Goddard Conference on Mass Storage Systems and Technologies. College Park: [s.n.], 1996: 188-203.

[6] Braam P J. The coda distributed file system[J]. Linux Journal, 1998, 6(2): 46-51.

[7] Hennessy J L, Patterson D A. Computer architecture: a quantitative approach[M]. 3rd ed. San Francisco: Morgan Kaufmann Publisher, Inc, 2002.

[8] Gilbson G A, Meter R V. Network attached storage architecture[J]. Communications of the ACM, 2000, 43(11): 37-45.

[9] Farley M. SAN 存储区域网络[M]. 2 版. 孙功星, 译. 北京: 机械工业出版社, 2001: 179-182.

[10] Seigle M. Storage area networks in video applications[J]. SMPTE Journal, 2001, 110(4): 236-239.