执行时间依次为 1 时间窗口、1/2 时间窗口 1/4 时间窗口)、2 次、1 次(1 时间窗口).测试结果表明:同一时刻执行算法次数越多,则预取的准确率越高.但执行时间的长短是有限度的,预取的代价不能在一个足够小的时间范围内被容忍.根据公式(4),我们同时运行 ln$n$ 次算法.

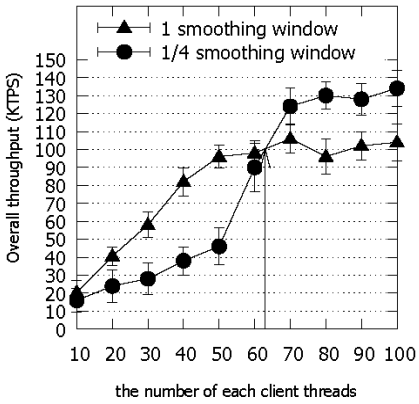

Fig.9 Accuracy comparison of algorithm 1 with different length adjusting periods

图 9 算法 1 不同长度时间内准确率对比

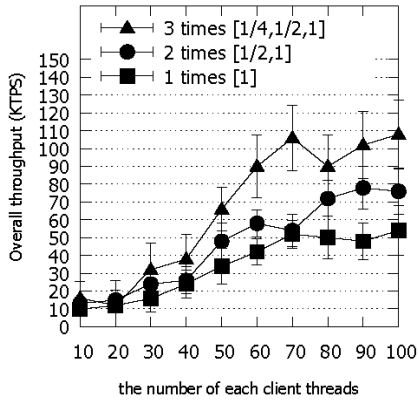

Fig.10 Accuracy comparison of algorithm 2 with different concurrent adjusting times

图 10 算法 2 不同运行次数的准确率对比

综合图 9 图 10 两组实验可以看出,两个预取算法的准确性都很高.对于算法的准确性而言,我们在一个时间滑动窗口同时运行两个算法.测试结果如图 11 所示,算法 2 达到 100KTPS 吞吐量的时间为 8 分钟,而算法 1 则慢了 5 分钟;同时,算法 2 的最大吞吐量为 140KTPS,即,算法 2 在时效性方面优于算法 1.

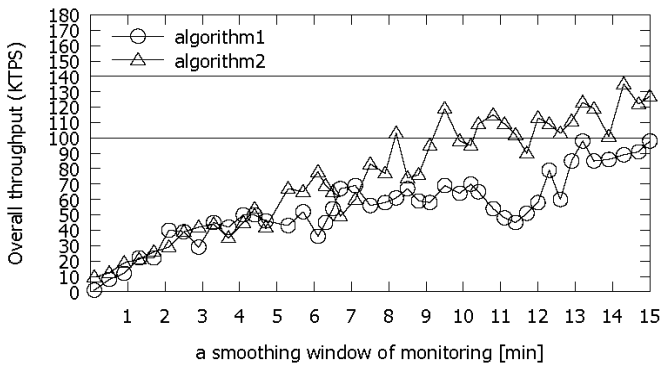

Fig.11 Comparison quickness between two algorithm

图 11 两种算法的时效性对比

4.3 PARC与其他缓存替换算法的性能对比

采用平均响应时间作为对比测量 PARC,ARC,LRU 性能的指标.基于大量的实际测试与参考文献数据,一次常规的元数据请求响应时间为 10ms 以内,所以我们将这个值(10ms)作为判断一个元数据服务器性能是否出现下降的基准线.测试结果如图 12 所示,算法 PARC 基本达到基准线,算法 ARC 差一些,两者的测试差距正是预取算法作用的表现.LRU 算法忽略了请求负载的变化频度,所以从这组实验看,LRU 算法基本是无效的.同时,这组实验也表现了 FLS 负载模式对元数据集群服务的负面影响,如方块曲线.

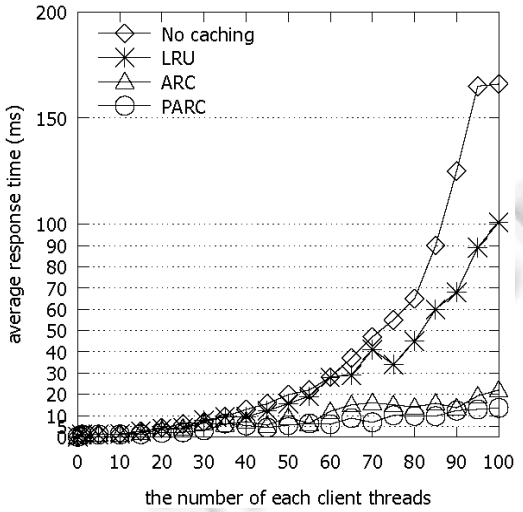

Fig.12 Compare performance of different cache algorithm

图 12 不同缓存替换算法性能对比

4.4 ACCS与强一致性策略的性能对比

缓存一致性策略采用 pull-based 方式实现.一致性协调器负责检测服务器端与缓存端数据一致性,即,同一个元数据缓存内容如果被检测出与服务器不一致,则根据更新时间确定该缓存是否失效:如果大于预设时间间隔,则视为失效缓存;如果小于或等于预设时间间隔,则视为可用缓存.

本组实验设置时间间隔分别为 1s,5s,10s 以及无时间间隔,每个客户端并发线程数量设置为 100,实验运行 20 次,测试结果如图 13 所示.随着允许不一致的时间间隔的减少,元数据平均响应时间越来越长,且呈不稳定态势逐渐明显.

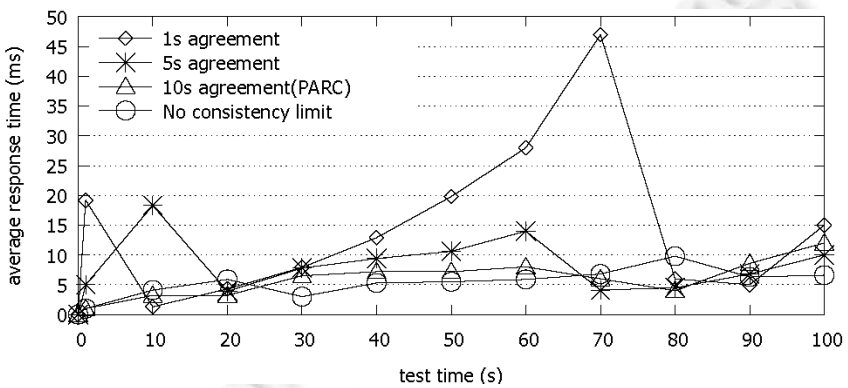

Fig.13 Comparison performance different consistency

图 13 不同一致性策略的性能对比

4.5 本文框架与子树、哈希性能对比

采用总体吞吐量作为对比测量子树、哈希以及我们的框架性能的指标.实验运行 20 次,每个客户端并发线程数量设置为 100.测试结果如图 14 所示:负载均衡效果最差的是子树架构,最好的是我们的两层架构.子树划分的元数据分布式管理架构在 FLS 负载模式中处于低性能的情况,哈希结构中出现的负载不均衡情况是由于同一时刻多个客户端同时访问一个目录或者文件造成的.本文提出的两层架构弥补了上诉两种架构的不足,独立

缓存层可以有效应对对突发性高负载模型引起的 FLS 场景.同时,从实验结果的误差情况看,我们的两层架构的误差范围也小于另外两种架构,这说明两层架构具备很好的稳定性.

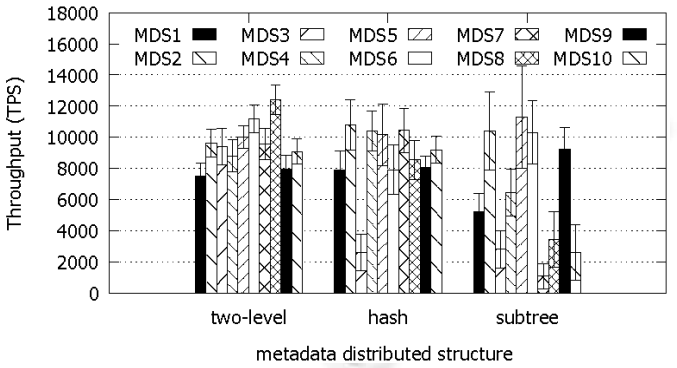

Fig.14 Comparison performance of different structure

图 14 3 种元数据管理架构的性能对比

5 相关工作

5.1 元数据动态负载均衡技术

已有研究工作运用动态调整的思想解决动态负载均衡问题,例如:

- 文献[14]提出的 DROP 采用 0-1knapsack 模型鉴别过载节点和空闲节点,然后执行一个迁移操作,重新平衡集群负载,并且在这个过程中保证元数据的目录关系不被破坏.该方法本质上依靠存储的负载均衡带动请求的负载均衡,同时也用到了缓存技术;
- 文献[15]提出在集群前端放置一个小体积的缓存专门管理热点,从而达到均衡全局访问负载的目的.该工作主要证实了一个观点:前端缓存大小与数据量大小无关,而与集群节点数量相关.一系列的验证实验都是围绕对抗性负载模式展开的,但是此方案不是针对元数据特点设计实现的;
- 另一个架构是文献[3]提出的两层无中心元数据管理模型,用一个全局副本策略管理最近被访问到的元数据.本质上,这是一个用准确率换取性能的方法.

以上分析的几个代表性的相关工作从 3 个主流的方面给出了解决动态负载均衡的思路,分别是再分布、缓存以及多副本技术.

本文方法同样用到了缓存、副本技术,但不同之处是:建立了一个完全独立的分布式缓存层,专门管理热点;同时,首次提出利用预取技术提高元数据缓存的性能.方案中的缓存层可以部署在不同的位置,适用于不同的系统.方法可以灵活应用于不同的 SLOs,在面对负载倾斜频繁变化问题时灵活度更高.

5.2 预取与缓存结合技术

预取技术最初是被用来解决 IO 延时问题[16]的,本文的目的是利用预取技术提高特殊场景中缓存的性能.预取与缓存结合的方法被广泛研究[16−18],文献[12]中,结合多种缓存替换算法,分析了预取的作用,得出的结论是,预取技术可以有效提高缓存性能.从文献[12]我们也受到启发,选择 ARC 算法作为基础算法.文献[16,18]支持此观点,不过他们提出,错误的预取会浪费缓存资源.基于这个观点,他们设计了一系列的规则,尽量降低错误预取对缓存性能的影响.

与以上工作相比,本文方法是基于概率的策略集成预取和缓存技术.这是缘于元数据的特点——预取错误的元数据带来的代价远小于预取错误的数据带来的代价.同时,ARC 算法[19]中的多队列机制也有利于我们调整预取对缓存的影响.

5.3 元数据一致性维护技术

元数据是新型分布式文件系统数据访问模型的关键组成部分,元数据层的强一致性,是系统准确实现数据副本一致性策略的基础.已有研究工作从如何降低元数据的强一致性维护成本方面展开了广泛的研究[4,20],文献[4]中,从减少两阶段提交协议中不必要的日志写入次数以及消息传递次数来减少强一致性开销.文献[20]观察到元数据分布式事务中子操作可以并发执行,中间结果可延时、批量提交等特点,提出一种面向实际应用的一致性放松协议,保持系统可扩展性的同时提高系统元数据服务性能.文献[21]同样采用缓存弱一致性技术提升元数据服务性能,客户端缓存的一致性维护机制是租约,不同时间长度的租约构成了一组弱一致性策略,此策略可以有效避免大量客户端缓存强一致性维护带来的性能开销.然而,此一致性策略应用范围方面存在一定的局限性,负载倾斜频繁变化时,即,热点位置频繁变化时,缓存一致性租约长度不能及时调整.ACCS 采用实时监测与阶段性统计的手段,可以感知负载倾斜变化,可以及时调整缓存一致性租约长度.文献[9]同样仅对读操作放松一致性要求,对于写操作仍然采用强一致性策略,将客户端读操作的一致性控制分为严格顺序一致、新旧偏差一致以及客户端自定义规则.ACCS 与文献[9]的区别是一致性控制粒度可以细化到会话级,这个区别对于突发性高负载模式有不可忽视的作用.本文针对突发性高负载模式,研究管理热点元数据过程中的一致性维护问题,首要考虑的是分布式元数据服务的性能指标.通过对元数据进行分类,将强一致性管理集中在写操作与后端服务器中实现,前端缓存采用了动态可调的弱一致性策略提高性能.

6 结束语

本文首先在元数据分布式管理框架之上设计实现了分布式缓存层,以有效应对潜在的各类热点以及突发性高负载模式.然后,我们提出了新的预取方法 TPPS 与新的缓存替换算法 PARC.在 FLS 场景中,预取技术与缓存技术的结合,有效地提高了缓存层的性能,在一致性方面考虑了时间偏差与顺序偏差因素对缓存性能的影响.最后,通过在 HDFS 元数据集群环境中的一系列实验结果,证明了本文提出框架、方法的有效性.

References:

[1] Shvachko K, Kuang H, Radia S, Chansler R. The hadoop distributed file system. In: Proc. of the IEEE Mass Storage Systems and Technologies (MSST). Incline Village, 2010. 1−10.

[2] Ghemawat S, Gobioff H, Leung ST. The Google file system. In: Proc. of the SOSP. 2003.

[3] Weil SA, Pollack KT, Brandt SA, Miller EL. Dynamic metadata management for petabyte-scale file systems. In: Proc. of the SC. 2004. [doi: 10.1109/SC.2004.22]

[4] Xiong J, Hu YM, Li GJ, Tang RF, Fan ZH. Metadata distribution and consistency techniques for large-scale cluster file systems. TPDS, 2011,22(5):803−816. [doi: 10.1109/TPDS.2010.154]

[5] Weil SA, Pollack KT, Brandt SA, Miller EL. Dynamic metadata management for petabyte-scale file systems. In: Proc. of the SC. IEEE, 2004. [doi: 10.1109/SC.2004.22]

[6] Beaver D, Kumar S, Li HC, Sobel J, Vajgel P. Finding a needle in Haystack: Facebook' sphotostorage. In: Proc. of the OSDI. 2010.

[7] Zhu YF, Jiang H, Wang J, Xian F. HBA: Distributed metadata management for large cluster-based storage systems. TPDS, 2008, 19(6):750−763. [doi: 10.1109/TPDS.2007.70788]

[8] Ari I, Hong B, Miller EL, Brandt SA, Long DDE. Managing flash crowds on the Internet. In: Proc. of the MASCOTS. 2003. 246−249. [doi: 10.1109/MASCOT.2003.1240667]

[9] CalvinFS: Consistent WAN replication and scalable metadata management for distributed file systems. In: Proc. of the FAST. 2015.

[10] Roselli D, Lorch JB, Anderson TE. A comparison of file system workloads. In: Proc. of the USENIX Technical Conf. 2000. 41−54.

[11] KVShvachko. HDFS scalability: The limits to growth. Usenix35, 2010,35(2):6−16. www.usenix.org/publications/login/2010-04/openpdfs/shvachko.pdf

[12] Xu QQ, Arumugam RV, Yong KL, Mahadevan S. DROP: Facilitating distributed metadata management in EB-scale storage systems. In: Proc. of the MSST. 2013.

[13] Cormen TH, Leiserson CE, Rivest RL, Stein C. Introducetion to Algorithms. 3rd ed., Edinburgh University Press, 2011.

[14] Xu QQ, Arumugam RV, Yong KL, Mahadevan S. DROP: Facilitating distributed metadata management in EB-scale storage systems. In: Proc. of the MSST. IEEE, 2013.

[15] Fan B, Lim H, Andersen DG, Kaminsky M. Small cache, big effect: Provable load balancing for randomly partitioned cluster services. In: Proc. of the SOCC. 2011. [doi: 10.1145/2038916.2038939]

[16] Cao P, Felten EW, Karlin AR, Li K. Implementation and performance of integrated application-controlled file caching, prefetching, and disk scheduling. ACM Trans. on Computer Systems, 1996,14(4):311−343. [doi: 10.1145/235543.235544]

[17] Butt AR, Gniady C, Hu YC. The performance impact of kernel prefetching on buffer cache replacement algorithms. In: Proc. of the SIGMETRICS. 2005. [doi: 10.1145/1064212.1064231]

[18] Teng WG, Chang CY, Chen MS. Integrating Web caching and Web prefetching in client-side proxies. IEEE Trans. on Parallel and Distributed Systems, 2005,16(5):444 −455. [doi: 10.1109/TPDS.2005.56]

[19] Megiddo N, Modha DS. ARC: A self-tuning, low overhead replacement cache. In: Proc. of the FAST. 2003.

[20] Yi LT, Shu JW, Ou JX, Zhao Y. Cx: Concurrent execution for the cross-server operations in a distributed file system. In: Proc. of the MSST. 2012. [doi: 10.1109/CLUSTER.2012.65]

[21] BIndexFS: Scaling file system metadata performance with stateless caching and bulk insertion. In: Proc. of the SC. Bestpaper, 2014. [doi: 10.1109/SC.2014.25]

[22] Nicolae B, Moise D, Antoniu G, Bouge L, Dorier M. BlobSeer: Bringing high throughput under heavy concurrency to hadoop map/reduce applications. In: Proc. of the IPDPS. 2010. 1−11. [doi: 10.1109/IPDPS.2010.5470433]

[23] Babaioff M, Immorlica N, Kleinberg R. Matroids, secretary problems, and online mechanisms. In: Proc. of the SODA. 2007. 434−443.

[24] Chen T, Xiao N, Liu F. Adaptive metadata load balancing for object storage systems. Ruan Jian Xue Bao/Journal of Software, 2013,24(2):331−342 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4177.htm [doi: 10.3724/SP.J.1001.2013.04177]

附中文参考文献:

[24] 陈涛,肖侬,刘芳.对象存储系统中自适应的元数据负载均衡机制.软件学报,2013,24(2):331−342. http://www.jos.org.cn/1000-9825/4177.htm [doi: 10.3724/SP.J.1001.2013.04177]

孙耀(1982－),男,吉林桦甸人,博士生,主要研究领域为网格分布式计算,软件工程,大数据存储.

叶丹(1971－),女,博士,研究员级高工,博士生导师,CCF 高级会员,主要研究领域为网络分布式计算,软件工程.

刘杰(1982－),男,博士,副研究员,CCF 专业会员,主要研究领域为网络分布式计算,软件工程.

钟华(1971－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为网络分布式计算,软件工程.