一种面向分布式文件系统的文件预取模型的设计与实现

师 明[1,3] 刘 轶[2] 唐歌实[1,3]
(北京航天飞行控制中心 北京 100094)[1]
(北京航空航天大学计算机学院中德联合软件研究所 北京 100191)[2]
(航天飞行动力学技术重点实验室 北京 100094)[3]

摘 要 如何为上层应用和计算提供稳定高效的文件I/O性能,是分布式文件系统性能研究的热点。文中分析分布式文件系统在设计机理上的共同特征,基于此提出一种通用型的启发式文件预取模型,并选取HDFS平台进行系统实现。启发式文件预取对上层应用透明,采用在文件系统内部建立预取线程池的方法,以组成文件块的数据存储文件为预取单位,在分布式文件系统内部实现。这种设计思路具有一定的普适性,适合推广应用于多种分布式文件系统。实验结果表明,所述的启发式文件预取,能够有效提升分布式文件系统的I/O性能。

关键词 分布式文件系统,文件预取,启发式,HDFS
中图法分类号 TP393 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.07.005

Design and Implementation of File Prefetching Module Oriented to Distributed File System

SHI Ming[1,3] LIU Yi[2] TANG Ge-shi[1,3]
(Beijing Aerospace Control Center, Beijing 100094, China)[1]
(Sino-German Joint Software Institute, School of Computer Science and Engineering, Beihang University, Beijing 100191, China)[2]
(National Laboratory of Science and Technology on Aerospace Flight Dynamics, Beijing 100094, China)[3]

Abstract How to provide a stable and efficient file I/O performance for the upper application and computing, is the performance research hotspot oriented to distributed file system. This paper analyzed the mechanism in the design of the distributed file system on the common features, presented a general-purpose file prefetching heuristic module, and selected HDFS platform system to implement. The heuristic file prefetching module services the upper application and accomplishes the implementation in the internal of distributed file system, using the method of establishing prefetching thread pool within the file system, and the data not block as prefetching unit. This idea has certain universality, and is suitable for a variety of distributed file systems. Experimental results show that the heuristic file prefetching method can enhance the distributed file system I/O performance effectively.

Keywords Distributed file system, File prefetching, Heuristic, HDFS

1 引言

面向云计算的分布式文件系统诸如GFS、HDFS、GPFS、DAFS、PVFS等,兼具存储容量大、高聚合I/O性能、容错性好等特点,在海量数据存储的应用领域获得广泛应用。目前,集群节点硬件性能发展迅速,CPU计算能力与I/O速度的不匹配问题愈加严重,I/O性能难以满足文件处理的需求,严重制约了云存储服务的应用和推广。如何为上层应用和计算提供稳定高效的文件I/O性能,是设计和优化面向云计算的分布式文件系统所亟需解决的问题。针对分布式文件系统I/O性能的研究,有两个热点值得注意:1)以顺序读为主的大规模数据的流式处理,如地理信息系统对数据的处理;2)海量文件处理的数据密集型的应用场景中,文件请求具有很强的随机性,如图片、音频搜索的后台服务。目前针对分布式文件系统性能优化的研究包括:Yue等[1]提出二级元数据管理方法来提高分布式文件系统的可用性;Mackey G.等[2]对HDFS小文件的元数据管理方式进行优化,以提高HDFS对小文件的存取效率;Yu等[3]发现文件分布模型对聚合I/O带宽有着显著的影响,提出一种基于用户视角的数据分布策略,并在其另一篇论文[4]中提出写操作分块和层次条带化的方法来提高分布式/并行文件系统的I/O性能;Yahoo[5]改进Hadoop多级缓存区的设计,简化JVM中的缓存区,把数据校验、数据压缩/解压的工作移至Native层进行,并使用效率更好的C语言加以实现;Facebook[6]对HDFS的文件访存过程进行优化,用C/C++代码对JVM从磁盘获取数据的部分流程进行了重写,以提高文件读写效率;Liu Xuhui[7]实现一类HDFS客

到稿日期:2013-09-11 返修日期:2013-12-06 本文受国家"十二五"863计划信息技术领域重大项目"云计算关键技术与系统"课题:以公众汉语服务为主的搜索引擎研制(2011AA01A205)资助。

师 明(1983—),男,硕士,工程师,主要研究方向为计算机体系结构与高性能计算、分布式文件系统及云计算,E-mail:shiming1983202@163.com。

户端，将小文件打包成大文件进行处理，以改善文件系统在处理海量小文件时的性能。研究点主要集中在系统实现和客户端应用优化等方面，多是有针对性的改进优化措施，通用性和适用性有限。

预取技术一直以来是研究文件系统性能提升的重要方法，也为分布式文件系统的 I/O 性能瓶颈提供了一种优良的解决途径，它通过一定的预取策略，将数据提前从本地硬盘读取至预取缓存池中，通过并行化的文件访问操作，有效地隐藏磁盘的寻址时间、寻道时间以及数据传输时延。Linux 即在其内核(2.6.24 版本之后)集成了一种按需预取算法，拟解决 CPU 等待的问题，为应用程序的文件访问提供性能输出支持。该技术应用于分布式文件系统的研究成果较少，典型的研究包括：GPFS 提出和实现了一种智能预取机制。该技术和后台写(write-behind)技术相结合，利用其高效的客户端数据缓存，通过多服务进程的调度，可以有效地降低读写延迟。Bo Dong[8] 等在 HDFS 系统上，提出一种基于文件关联性的预取技术，根据数据存储结构分析前后文件读请求之间的关联关系，把存在关联关系的全部 metadata、block 和 replica 都作为预取范围。上述方法对大文件连续顺序访问的 I/O 性能提升明显，而面对海量文件处理的场景，由于数据读取具有随机性，该预取机制会引发大量的寻址操作，反而会降低文件系统性能。而且，GPFS 是一款商业软件，其预取机制的设计思想，也鲜有被其他文件系统所借鉴。

在设计机理上，面向云计算的分布式文件系统有很多共同特点：结构上一般可以分为元数据管理节点、存储节点和客户端 3 个组成部分；使用较大的文件管理单元(文件块)进行数据存储的统一管理，组成文件块的数据存储文件在物理磁盘上多是连续的；文件目录集中管理，并提供单独的目录空间，对文件 I/O 请求统一进行处理；上层应用或客户端文件读取的数据流程类似，并采用多层次的文件缓冲结构设计；实现单一的文件映像，一个文件数据可以分布于不同的磁盘或存储节点，而文件存放位置的分布情况对用户是透明的，用户更关注效率而不是实现细节。这些在文件管理、数据访问流程、文件缓存管理等方面的共性，为文件系统性能的研究带来便利，可以方便地研究通用型的预取设计方案，在具体的技术实现上，再根据不同文件系统平台加以区别。

本文提出一种通用型的启发式文件预取模型，并选取 HDFS 平台进行系统实现。实验结果表明，本文所述的启发式文件预取，能够有效提升分布式文件系统的 I/O 性能。

2 总体设计

分布式文件系统文件访问流程一般为：本地缓存区查找数据—与存储节点建立连接—存储节点根据文件请求触发读线程—定位文件块存储位置—调用 Linux 内核函数读取数据存储文件—数据存放至本地缓存区。文件预取的设计针对存储节点中的文件读取操作进行性能优化设计。

2.1 预取总体框架

启发式文件预取的运行框架如图 1 所示。

启发式文件预取部署在云计算环境中的存储节点，响应来自分布式文件系统上层应用的文件读取请求，以组成文件块的数据存储文件为预取对象，并通过 Linux 系统的内核系统调用的支持，将预取数据读取并存放至分布式文件系统的内部缓存区中，同时记录文件历史，访问历史记录，构建 LS 预测模型。

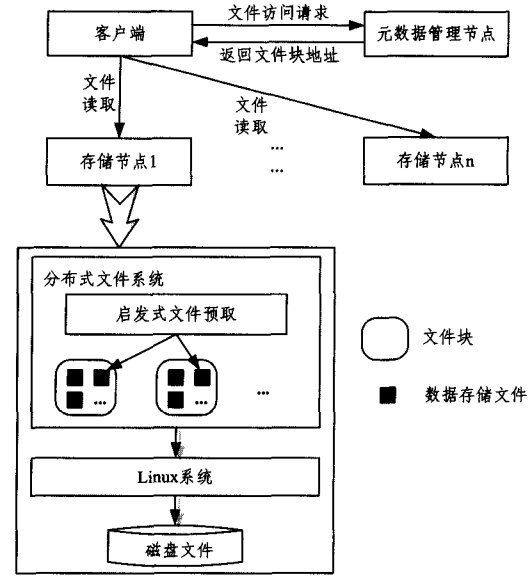

图 1 启发式文件预取的运行框架

2.2 启发式文件预取逻辑结构

启发式预取机制在逻辑结构上分为线程池的管理和创建、监控触发、模式识别、预取量管理、缓存区数据管理、用户管理和配置 7 个功能子模块，如图 2 所示。

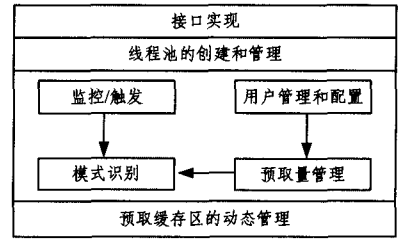

图 2 启发式预取的逻辑结构

接口实现：包括获取当前访问文件的信息，以及调用 Linux 内核函数执行强制预取指令两个部分。

线程池的创建和管理：创建和管理预取线程，响应文件读函数的调用。

监控/触发：嵌入在程序预取请求的响应线程中，在文件 I/O 请求到来时，检查缓存中页面是否满足预取触发条件。

模式识别：由一组独立的判定函数组成，对文件访问请求进行分支判断，识别不同访问模式。

预取量管理：创建系统与用户的通信机制，提供预取量管理的相关因子配置说明，用户可通过修改分布式文件系统的配置文件配置与预取量相关的参数。

缓存区数据的动态管理：嵌入在分布式文件系统的文件读主线程中，提供文件预取的管理窗口，同时对预取数据缓存区执行动态管理策略。

用户管理和配置：为用户提供交互窗口，后者可根据具体的使用环境进行参数优化。

2.3 技术路线对比

启发式文件预取和 Linux 的预取技术，在设计思想、优化策略等诸多方面存在联系和区别，在此对这两种技术进行比较，见表 1。

表 1　与 Linux 按需预取技术的对比

	预取响应	工作层次	数据存放	支持范围	应用场景
Linux	以 pagecage 为中心的缺页判断	Linux 内核 VFS 层	内核缓存区域，系统管理	顺序读	对上服务于应用程序请求，对下独立于不同的文件系统
分布式文件系统	开辟预取线程池，由主线程触发	存储节点的文件读主线程	分布式文件系统内部缓存区，预取机制动态管理	顺序读/随机读	对上响应应用程序请求，对下依赖于不同操作系统内核函数的支持

启发式文件预取和 GPFS 智能预取有着显著区别，详细比较见表 2。启发式文件预取适用性更好，推广应用价值更高。

表 2　与 GPFS 智能预取技术的对比

	预取对象	缓存区管理	预取实现方式	应用场景
GPFS（智能预取）	Block	申请缓存区，单独管理	线程池	大规模的连续顺序访问
启发式文件预取	数据存储文件	分布式文件系统内部缓存区，清除过期页面，执行动态管理	预取线程池；posix _ fadvise 内核系统调用的支持	响应上层所有的访问请求

3　基于 HDFS 的启发式文件预取技术实现

HDFS 是当前主流的分布式文件系统之一，获得 Yahoo、Facebook 以及淘宝、百度等众多 IT 厂商的青睐。选取 HDFS 进行系统实现，验证文件预取系统设计的正确性和可用性，具有说服力。HDFS 含 NameNode、DateNode、Client 3 个组织要点，分别执行元数据管理节点、存储节点和客户端的职能。

3.1　DataNode 节点文件访问静态模型

Client 从 NameNode 获取所需 Block 的位置信息，并与 DataNode 建立连接进行数据传输。DataNode 会首先调用 FSInputStream 构建一个文件流实例，从 Client 获取所要读取 block 的信息，包括 block 的 ID、数据偏移量、数据的长度、客户端的标识等，完成读取操作的初始化工作。同时，实例化 FSInputChecker 对象并对 DataNode 上传输的数据进行 checksum 的校验操作。BlockReader 类进行 block 中数据存储文件的读取操作，并根据 hadoop 数据流传输协议对数据进行解析。文件访问的静态模型如图 3 所示。

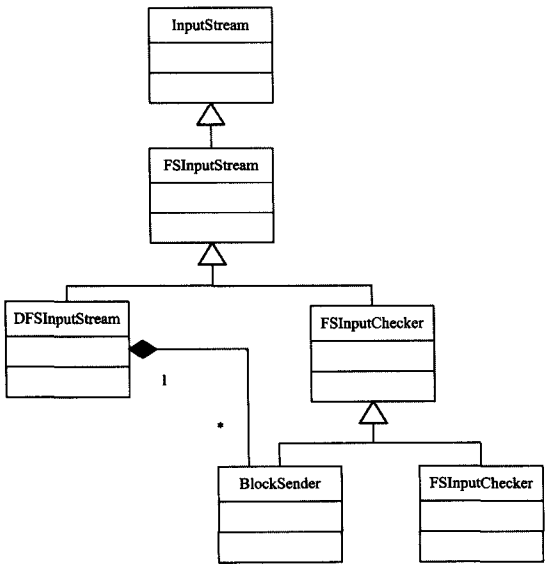

图 3　DataNode 文件读取静态模型

启发式文件预取即加载在 BlockReader 类中。

3.2　基本数据结构

定义变量参数用以保存文件访问信息：

```
long curPos //当前访问的页面位置
long lastOffset//上次访问结束的页面位置
long Offset//上一次访问页面的偏移量
long readaheadLength//一次预取的页面长度
long maxOffsetToRead//最大可预取量，用于判定是否达到文件末尾
```

3.3　预取执行流程

由 HDFS 文件读函数的主线程发出文件预取请求，传递文件信息以及预取请求的相关变量。预取缓存池根据预取队列的运行情况，判断和执行预取线程。预取线程间的执行互不干扰，独立管理各自的内部缓存区域，便于管理从 Linux 内核读函数返回的数据文件。预取线程执行初始化内部缓存区、分析文件句柄、监控触发、模式识别、预取量管理、强制预取请求提交等完整工作流程。其中，模式判别模块会根据用户打开文件位置的变化，依照不同的判别条件，执行不同的预取操作。预取执行流程如图 4 所示。

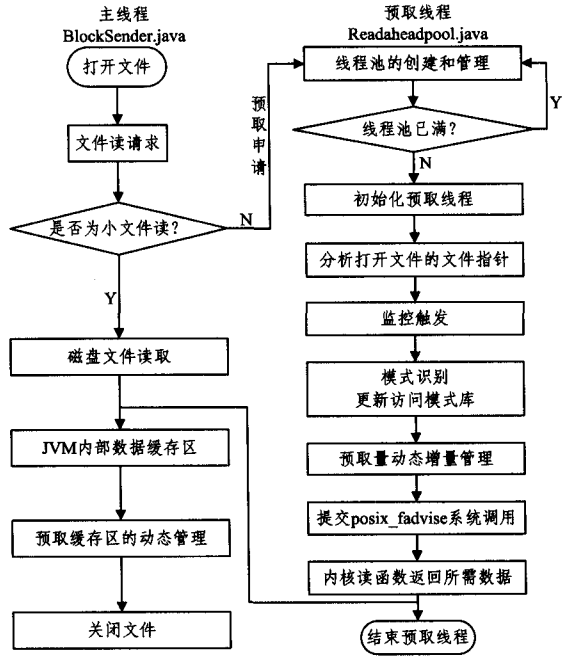

图 4　启发式预取的执行流程

3.4　模块实现要点

3.4.1　接口实现

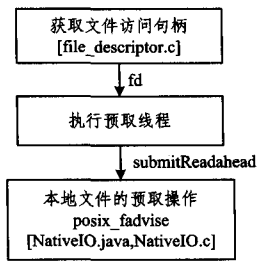

图 5　接口实现的子模块组成

文件预取涉及到 HDFS 与 Native library 的交互，从实现上是一个混合编程的工程。与 Native library 交互的部分包括：获取文件访问句柄（图 5 中 file_descriptor. c）以及通过操

作系统读取本地磁盘文件(图 5 中 NativeIO. java 和 NativeIO. c)。Hadoop 中,与操作系统内核函数交互的函数都是通过 JNI 技术实现的。在接口实现中,没有修改其 JNI 连接,只是在现有相关函数中添加功能函数的代码,重新编译生成 JNI 访问连接来实现目的。接口实现子模块的组成如图 5 所示。

3.4.2 线程池的管理和创建

利用线程池能够带来 3 个好处。1)降低资源消耗。通过重复利用已创建的线程降低线程创建和销毁造成的消耗。2)提高响应速度。当任务到达时,任务可以不需要等到线程创建就能立即执行。3)提高线程的可管理性。线程是稀缺资源,如果无限制地创建,不仅会消耗系统资源,还会降低系统的稳定性,使用线程池可以对其进行统一的分配、调优和监控。

使用有界队列对线程池进行管理,以增加系统的稳定性和预警能力,避免因任务积压引发内存抛弃异常的问题。用户可以根据应用程序执行情况、集群规模、节点性能等因素合理配置队列长度,并考虑用不同规模的线程池运行不同类型的任务。修改线程池大小和任务队列长度后,只需对 Hadoop 源码进行重新编译和发布即可。

3.4.3 监控触发

如图 6 所示,在内部缓存区的文件流中设置一个标识,用来判定下一次预取时机是否到来。标识用 nextOffset 表示:nextOffset=lastOffset+readaheadLength/2。预取标识是在上一次预取 I/O 时设置的,用来指示应用程序已经用尽了足够的提前读窗口,此时应读进更多的页,命中它意味着进行下一个预取 I/O 的时机已经来到,因而应立即调用预取例程,进行异步预取,同时清除 nextOffset 标识。

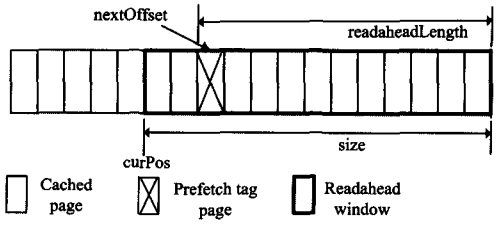

图 6 启发式预取监控触发示意图

3.4.4 模式识别

一个文件访问流对应一组预取请求序列,记文件读请求序列为 $R=\{(step,size)\mid i=0,1,2,\cdots n\}$,其中,$step$ 表示两次预取操作在业内地址的偏移量。预取的起始页面和文件请求序列的位置关系为 $R_m'=R_{m+1}\geqslant 0$,m=0 时,表示读请求位于文件的首字节,从序列中的第 2 个读请求开始预取算法才能判断它为一个时空连续的文件访问流,并将预取请求置于 Initial read 位置。

在预取线程的实现 Readahead. java 中定义变量:

```
stepP=offset-lastOffset;//文件读取的步长
posStepP=stepP;//记录上一次文件读取的步长
```

$stepP>0$ 时,文件 I/O 请求序列为正向顺序。对于前后 3 个读请求 R_{a-2}、R_{a-1} 和 R_a,若 R_a 的开始位置恰好是 R_{a-1} 的结束位置,文件 I/O 为顺序读。文件访问的特征的判定条件见表 3。

表 3 启发式预取模式判定条件

访问模式	处理方案
首次访问文件头部首次预取	lastOffset=Long. MIN_VALUE
遇到访问标识 nextOffset 的顺序读	Offset=readaheadLength; lastOffset=Offset;
到达文件末尾	Math. min(readaheadLength, maxOffsetToRead-curPos)< 0
小文件读	在主线程向预取线程池发送预取请求之前,由判断函数 isLongRead()判断文件访问流的对象是否为小文件读,若是则不执行预取线程

这里需要注意的是小文件读的判定,设定在主线程向预取线程池发送预取请求之前进行,如果是小文件,则不执行预取线程,这样可以节省线程资源。一般认为,在一个 block 内的数据存储文件的文件访问是连续的。如果是大量小文件(<256k,自定义大小,可更改)的随机读取,典型的应用有数据库操作等,文件访问过于零散,若执行文件预取则会带来大量额外的文件寻址和定位操作,影响分布式文件系统的效率,给系统性能带来不利影响,所以对小文件不进行预取操作。

小文件的判定函数 isLongRead()嵌入在 HDFS 文件读的主线程函数 BlockSender()中,执行如下判断:(endOffset-offset)>LONG_READ_THRESHOLD_BYTES(256k)

3.4.5 预取量管理

对于一个有效预取请求序列 $R'=\{(step_i,size_i)\mid i=0,1,2,\cdots,n\}$,称 R_0' 为初次预取,R_1',R_2',…,R_n' 为后续预取。预取量则采用恒量管理的方法,每次预取的长度即为用户设定的预取量大小,并设定该值应该小于 maxOffsetToRead。一般设定 maxOffsetToRead 是一个绝对值很大的常数值(Long. MAX_VALUE),这样,预取一般以用户设定的预取量大小进行。

1)初次预取窗口大小等于文件访问流中 I/O 请求的大小。

2)从第二个有效读请求开始,每次的预取量大小等于主线程传递的预取请求大小。

3)判定预取窗口是否已经到达文件末尾,若已经达到文件末尾,则预取操作返回空值。

3.4.6 预取缓存区的动态管理

预取缓存区的管理实现函数 manageOsCache()嵌入在 HDFS 文件读的主线程函数 BlockSender()中,用以提供文件预取的管理窗口,同时对预取数据缓存区执行动态管理策略。其中,预取的管理窗口用到了函数 isLongRead(),后者实现对数据存储文件大小的判定,是模式识别中的一个实现。

定义相关变量如下:

```
long lastCacheDropOffset;//上次释放缓存区的页面位置
static final long CACHE_DROP_INTERVAL_BYTES = 1024 * 1024;//数据缓存区管理单位
long LONG_READ_THRESHOLD_BYTES=256 * 1024;//数据缓存区的最小值,用于判定文件是否为小文件读
long nextCacheDropOffset;//释放缓存区的标识
```

预取量管理函数的执行流程如图 7 所示。

函数 manageOsCache()实现两个功能:向预取线程池提出文件预取请求;对预取缓存池进行动态管理。

当判定函数 isLongRead()返回非空值并且当前 block 有效(根据 BlockID 判断)时,向预取线程池提出文件预取请求,后者将文件读请求加入预取队列。

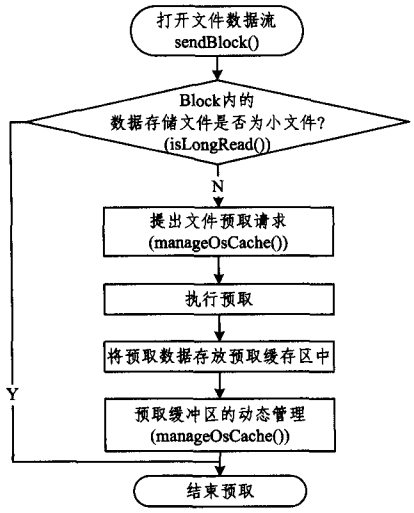

图 7　预取量管理函数的执行流程

通过动态地删除过期的文件页面，对预取缓存池进行动态管理，工作机制如图 8 所示。

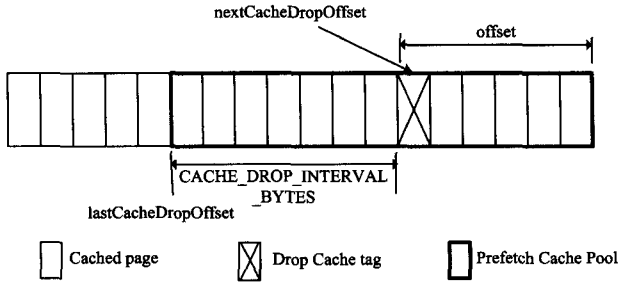

图 8　预取缓存池的动态管理

在缓存区文件队列中，设定一个标识，用来判定下一次清理预取缓存池的时机。标识用 nextCacheDropOffset 表示：nextCacheDropOffset = lastCacheDropOffset + CACHE_DROP_INTERVAL_BYTES。缓存区清理标识是在本次文件读取完成后设置的，用来指示文件读的主线程，预取数据已经过期，后续跟进的读取操作将不会再使用这部分数据，可以更新缓存区域，以节省缓存空间。命中它意味着进行下一次预取缓存区清理的时机已经到来，应该立刻更新缓存区域，同时清除 nextCacheDropOffset 标识。

3.4.7　用户管理和配置

通过在 DataNode.java 中添加配置函数类成员函数，把预取量大小(dfs.datanode.readahead.bytes)作为配置项加入 hdfs-site.xml 配置文件中，为用户提供交互窗口。

4　性能测试与结果分析

在 4 台 IBM 刀片机上部署 HadoopHadoop 0.20.203 版本，采用 4 个 DataNode+1 个 NameNode 的组合，其中一个服务器同时承担 NameNode 和 DataNode 两个角色。

有效性测试：启发式文件预取的预取对象是组成文件块的数据存储文件，由于分布式文件系统都是以文件块为单位进行文件访问，因此忽略了分布式环境下不同文件 I/O 请求模式(顺序读、逆序读、跳步读等)对文件预取性能的影响。

影响因素测试：环境变量会对分布式文件系统的 I/O 性能产生影响，测试方案拟选定文件大小、预取量大小、预取线程池大小、Map 任务多少 4 个影响因素，分别进行测试，分析影响分布式文件系统预取效果提升的因素，提出针对性的优化措施。

测试工具：TestDFSIO。该程序使用 MapReduce 框架模拟多路的并发读写，生成统计信息，是一个分布式的 I/O 基准，用于测试 HDFS 的 I/O 性能。

定义 TestDFSIO 测试的数据集：预取量大小 dfs.datanode.readahead.size={0,15,30,50,100,200}(byte)，文件大小 fileSize={1,5,10,40,100,200}(Mb)，测试文件数量 nrFiles={4,6,10,20,40,80}。

4.1　有效性测试

随机选取预取量：dfs.datanode.readahead.bytes=50，比较同一组测试数据下，使用启发式预取/不适用启发式预取两种 HDFS 下的文件吞吐率(Throughput Mb/sec)。文件数量为 6，文件大小的取值范围为{1,5,10,40,100,200}(Mb)，包含了典型的小文件、一般文件和大文件几种选型。测试效果如图 9 所示。

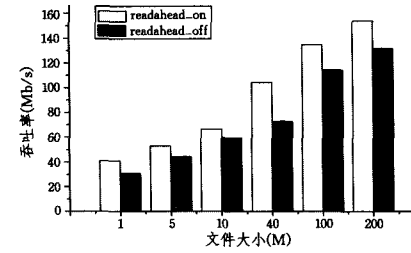

图 9　文件预取有效性测试

由结果分析可知，使用启发式预取后 HDFS 的吞吐率获得明显提升，其中，文件大小为 40M 时，性能优化效果最好，达到 43.97%。在实验样本内，HDFS 吞吐率性能平均提升约 27.49%。

4.2　影响因素测试

4.2.1　文件大小

选取预取量：dfs.datanode.readahead.bytes=30，比较同一组测试数据下，使用启发式预取\不适用启发式预取两种 HDFS 下的吞吐率。文件数量为 20，文件大小的取值范围为 fileSize={1,5,10,40,100,200}(M)。性能比较如图 10 所示。

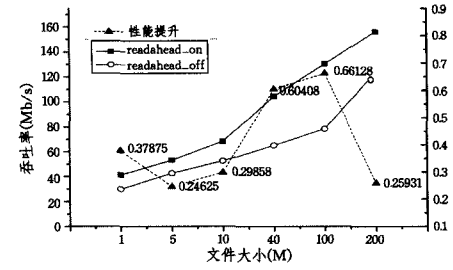

图 10　文件大小对预取性能提升的影响

由结果分析可知，随着文件大小的增加，使用文件预取的 HDFS 文件性能也呈现线性增长的趋势，支持“HDFS 更适合处理大数据文件”的事实。同时，文件大小对预取效果产生较为明显的影响，预取用于小文件(<=64M)处理效果更优，在 block 大小附近，如图中选取的 100M，HDFS 进行数据访问时，由于不会产生过多的寻址操作，预取效果最好。而对于单个的大数据文件，预取效果容易淹没在文件流式访问的大数据量中，对系统性能的提升比例反倒会下降。

4.2.2　预取量大小

选取预取量 dfs.datanode.readahead.bytes={1,5,30,

50,100,200}，比较同一组测试数据下，使用启发式预取\不适用启发式预取两种 HDFS 下的吞吐率大小。文件数量为 20，单个文件大小为 40M。性能比较如图 11 所示。

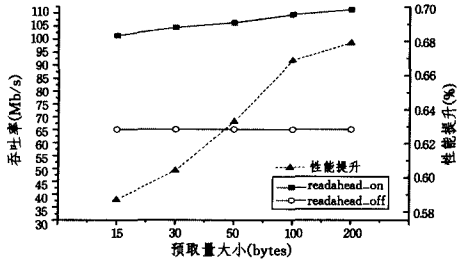

图 11　预取量大小对预取性能提升的影响

由结果分析可知，随着预取量的增加，HDFS 的吞吐率获得稳定的线性增长，性能提升越来越明显，预取量大小对文件系统的预取效果有着比较明显的影响。实验中选取了相对保守的预取值，在进行大规模的文件读操作过程中，可适当增大预取量，提升系统 I/O 性能。

4.2.3　线程池规模

选取预取量 dfs. datanode. readahead. bytes＝15，比较同一组测试数据下，使用启发式预取\不适用启发式预取两种 HDFS 下的吞吐率。文件数量为 20，单个文件大小为 5M。线程池的取值范围 POOL_SIZE＝{4,8,16,32}，对应地，创建线程池的最大数的取值范围 MAX_POOL_SIZE＝{8,16,32,64}。性能比较如图 12 所示。

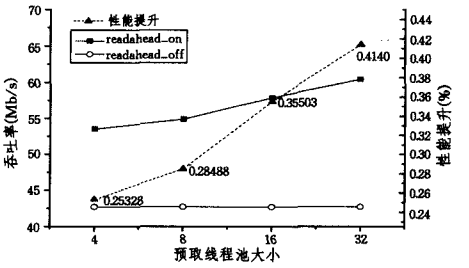

图 12　线程池大小对预取性能提升的影响

由结果分析可知，随着预取线程池的增加，HDFS 的性能提升呈现线性增长的趋势，线程池大小会对预取效果产生比较明显的影响。增大线程池，意味着主线程可以调用更多的预取线程获取资源。实际应用中，如果集群节点的负载较小，可以适当增大线程池的大小来提高系统性能，但同时也要注意过多的预取线程会影响 JVM 运行的稳定性。

4.2.4　Map 任务数量

选取预取量 dfs. datanode. readahead. bytes＝30，比较同一组测试数据下，使用启发式预取\不适用启发式预取两种 HDFS 下的吞吐率。单个文件大小为 10M。TestDFSIO 会根据文件数量的多少来决定开辟多少 Map 任务进行处理，这里用文件数量来表示 Map 任务的多少，取值范围 nrFiles＝{4,6,10,20,40,80}。性能比较如图 13 所示。

由结果分析可知，随着文件数量的增多，HDFS 的吞吐率呈现线性降低的特征，与之相对应，采用启发式预取后的文件系统吞吐率也会随之发生下降的趋势。在性能提升上，过多的文件数量会产生更多的 Map 任务进行数据处理，在节点数量一定的情况下，增加 Map 任务不能提升 HDFS 的吞吐率。

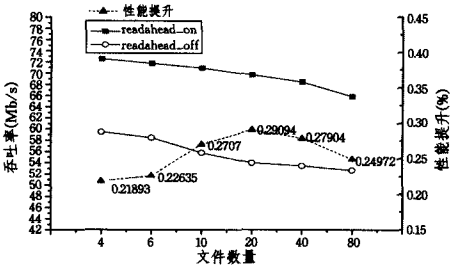

图 13　Map 任务数量对预取性能提升的影响

结束语　分布式文件系统 I/O 性能的提升，是当前云计算研究的热点之一。本文深入研究 GFS、GPFS、HDFS 等典型的面向云计算分布式文件系统的基础结构，基于分布式文件系统在文件管理、数据访问流程、文件缓存管理等设计机理中的共同特征，提出一种通用型的启发式文件预取模型，并选取典型的文件系统 HDFS 平台进行系统实现，以验证总体设计方案和预取模型的有效性。启发式文件预取以存储节点为工作节点；由接口实现、监控触发、模式识别、预取量管理以及用户管理和配置等模块组成；采用在分布式文件系统内部建立预取线程池的方法，响应上层应用程序所有的文件访问请求；预取数据存放在内部数据缓存区中并执行动态管理；以组成文件块的数据存储文件为预取单位，通过记录文件访问信息构建 LS 预测模型。这种设计思路适合推广应用于多种分布式文件系统。实验结果同时表明，本文提出的文件预取的方法，以及基于 HDFS 的具体实现，能够有效提升分布式文件系统的 I/O 性能。

参 考 文 献

[1] Yue Yin-liang, Feng Dan, Wang Juang, et al. High Availability Storage System Based on Two-level Metadata Management[C]// Proceedings of Frontier of Computer Science and Technology (FCST 2007). 2007:41-48

[2] Mackey G, Sehrish S, Wang Jun. Improving Metadata Management for Small Files in HDFS[C]// Proceedings of Cluster Computing and Workshops. 2009:1-4

[3] Yu Wei-kuan, Oral H S, Canon R S, et al. Empirical Analysis of a Large-Scale Hierarchical Storage System[C]// Euro-Par 2008-Parallel Processing. 2008:130-140

[4] Yu Wei-kuan, Jeffrey S V, et al. Performance Characterization and Optimization of Parallel I/O on the Cray XT[C]// The 22nd IEEE International Parallel and Distributed Processing Symposium(IPDPS2008). 2008:1-11

[5] Dittrich J, Quiane R J A, Jindal A, et al. Hadoop++: Making a Yellow Elephant Run Like a Cheetah (Without It Even Noticing)[J]. Proceedings of the VLDB Endowment, 2010(3):515-529

[6] OMalley O. The Anatomy of Hadoop I/O Pipeline[EB/OL]. http://developer. yahoo. com/

[7] Liu Xu-hui, Han Ji-zhong, Zhong Yun-qin, et al. Implementing WebGIS on Hadoop: A case study of improving small file I/O performance on HDFS[C]// Cluster Computing and Workshops (CLUSTER'09). 2009:1-8

[8] Dong Bo, Qiu Jie, Zheng Qing-hua, et al. A Novel Approach to Improving the Efficiency of Storing and Accessing Small Files on Hadoop: A Case Study by PowerPoint Files[C]// Services Computing Conference(SCC). 2009:65-72