(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 113901007 A
(43)申请公布日 2022.01.07

(21)申请号 202111220222.0

(22)申请日 2021.10.20

(71)申请人 杭州电子科技大学
地址 310018 浙江省杭州市钱塘区白杨街道2号大街

(72)发明人 路锦 曾艳 赵乃良 张纪林 袁俊峰 万健 张雪容 沈鸿辉

(74)专利代理机构 浙江千克知识产权代理有限公司 33246
代理人 周雷雷

(51)Int.Cl.
G06F 16/172(2019.01)
G06F 16/13(2019.01)
G06F 12/0871(2016.01)
G06F 12/0862(2016.01)

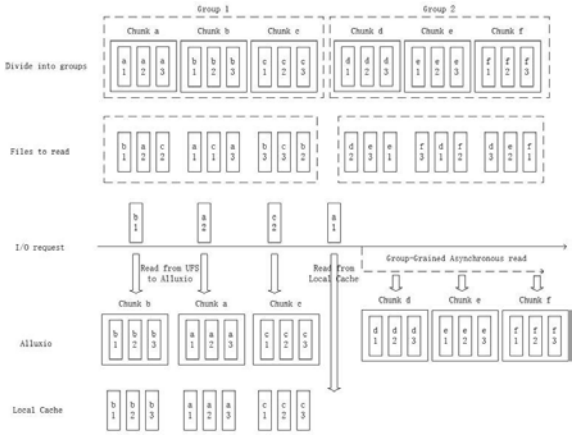
权利要求书1页 说明书4页 附图2页

(54)发明名称

一种面向AI训练的海量小文件分布式缓存方法

(57)摘要

本发明公开了一种面向AI训练的海量小文件分布式缓存方法,以实现海量小文件高性能分布式缓存。本发明首先对小文件按照拟合AI训练中Batch Size特征的规则合并为chunk;其次,分析chunk的缓存状态,对小文件序列进行双层shuffle操作;最后,在AI训练读取数据时,针对重复I/O采用Local Cache短路读取,并在Local Cache短路读取时机开启异步分组预读。本发明通过高效利用缓存解决面向AI训练的海量小文件随机读取难题,在面向AI训练的场景下,显著提升数据访问速率与缓存命中率,减少AI训练的迭代时间。



CN 113901007 A

1.一种面向AI训练的海量小文件分布式缓存方法,其特征在于该方法包括以下步骤:

步骤1:在客户端创建以key-value键值存储的Local Cache,并在分布式存储设备中创建Alluxio缓存;

步骤2: 在AI训练的数据存储阶段,对数据集基于拟合Batch Size特征进行chunk合并操作;

步骤3:在AI训练迭代前,分析chunk的缓存状态,采取chunk shuffle+group内小文件shuffle的双层shuffle操作,生成本次迭代的遍历序列;

步骤4:在AI训练迭代时对重复I/O采用Local Cache短路读取,并同时,采用异步分组预读方法将后续chunk预读到Alluxio缓存。

2.根据权利要求1所述的一种面向AI训练的海量小文件分布式缓存方法,其特征在于:所述的步骤2中所述对数据集基于拟合Batch Size特征进行chunk合并,即将小文件按照2的N次方合并规则合并为chunk。

3.根据权利要求1所述的一种面向AI训练的海量小文件分布式缓存方法,其特征在于:所述的步骤3中所述的分析chunk的缓存状态,即在shuffle操作的初始过程通过缓存状态将chunk划分为两个序列,分别是位于缓存中的chunk序列,以及不在缓存中的chunk序列,并分别对这两个序列采取双层shuffle操作。

4.根据权利要求1所述的一种面向AI训练的海量小文件分布式缓存方法,其特征在于:所述的步骤3中所述的对chunk中的小文件序列采取chunk shuffle+group内小文件shuffle的双层shuffle操作,具体是:

第一层shuffle操作为chunk shuffle操作,即获取序列中的所有chunk,并在序列中整体shuffle;

第二层shuffle操作为group内小文件shuffle操作,即将序列中的所有chunk分组,分别获取每个group中的小文件序列,在group内对小文件整体shuffle。

5.根据权利要求1所述的一种面向AI训练的海量小文件分布式缓存方法,其特征在于:所述的步骤4中所述重复I/O是指:当前I/O请求所访问的目标文件已存在于Local Cache,则此次I/O请求为重复I/O;所述Local Cache短路读取是指:在客户端拦截此次发起的I/O请求,从Local Cache中取出目标小文件,随后在Local Cache中删除此文件。

6.根据权利要求1所述的一种面向AI训练的海量小文件分布式缓存方法,其特征在于:所述的步骤4中所述的采用异步分组预读方法将后续chunk预读到Alluxio缓存,具体是:判断Local Cache中是否存储了当前group中的全部chunk,若已经存储,则后续访问均为Local Cache短路读取,因此采用异步分组预读,在后续访问的同时,对底层存储系统发起对下一个group中的chunk的 I/O请求,将chunk预读到Alluxio缓存。

一种面向AI训练的海量小文件分布式缓存方法

技术领域

[0001] 本发明面向智慧城市,对于人脸识别、视频查询、智能存储等场景,设计了一种面向AI训练的海量小文件分布式缓存方法,以实现海量小文件高性能分布式缓存。

背景技术

[0002] 近年来,随着全球范围内社会经济与科学技术的高速发展,AI技术在安防领域的大规模落地与应用,强有力地助力了平安智慧城市发展。与此同时,平安智慧城市中诸如人脸识别、行人跨境追踪等场景对AI技术提出了挑战。通常AI训练任务需要的文件数量都在几百万甚至上千万的量级,例如Google OpenImages数据集包含900万个图像,Tencent ML-Images数据集包含近1,769万个图像。通常需要对百亿级规模的海量小文件进行缓存处理,然后采用ResNet50,ResNet101等大规模AI网络模型进行分布式训练。因此,针对海量小文件进行大规模AI训练是AI发展的重要趋势,而针对AI训练的海量小文件高性能分布式缓存方案是当前亟待解决的重要问题。

[0003] UC Berkley于2018年提出的一种开源分布式缓存系统Alluxio。该系统在数据存储时支持大文件分块存储,数据访问时,采用按访问频率采取MEM、SSD、HDD的分层缓存,并在每层独立采取缓存替换策略。但该系统在缓存策略上默认使用传统的LRU替换策略,由于AI随机访问特性,访问的数据不存在热点数据,因此传统缓存替换策略无法提高缓存命中率,不适用于AI训练时的随机访问场景。

[0004] 商汤提出一种主从架构系统DIESEL。该系统提出了将海量小文件合并数据块存储的方案,并且提出了组内shuffle来代替AI训练随机访问,但该系统由于AI任务在首次访问数据块时仍然存在缓存缺失现象,需要等待较长的磁盘I/O时间,影响数据访问速率。

发明内容

[0005] 本发明的目的为了克服面向AI训练时存在的:数据随机访问速度过低、缓存命中率较低问题,提出了一种面向AI训练的海量小文件分布式缓存方法。

[0006] 本发明步骤如下:

[0007] 步骤1:在客户端创建以key-value键值存储的Local Cache,并在分布式存储设备中创建Alluxio缓存。

[0008] 步骤2:在AI训练的数据存储阶段,对数据集基于拟合Batch Size特征进行chunk合并操作。

[0009] 步骤3:在AI训练迭代前,分析chunk的缓存状态,对chunk中的小文件序列采取chunk shuffle+group内小文件shuffle的双层shuffle操作,生成本次迭代的遍历序列。

[0010] 步骤4:在AI训练迭代时对重复I/O采用Local Cache短路读取,并同时,采用异步分组预读方法将后续chunk预读到Alluxio缓存。

[0011] 本发明具有的有益效果是:

[0012] 1、本发明在数据存储阶段按照拟合Batch Size特征的规则合并为chunk,即采用2

的N次方规则合并小文件,合并后的chunk结构更加拟合AI数据读取。

[0013] 2、分析chunk的缓存状态,采用chunk shuffle+group内小文件shuffle的双层shuffle,解决随机访问难题,在每次迭代中优先利用缓存中的chunk数据,高效利用缓存。

[0014] 3、在AI训练迭代时对重复I/O采用Local Cache短路读取,并同时,采用异步分组预读方法将后续chunk预读到Alluxio缓存,增大缓存命中率,提升数据访问速率。

附图说明

[0015] 图1为小文件合并流程图;

[0016] 图2为双层shuffle流程图;

[0017] 图3为AI训练迭代流程图。

具体实施方式

[0018] 下面结合附图,对本发明的具体实施方案作进一步详细描述。

[0019] 本发明包括以下步骤:

[0020] 步骤1:创建Local Cache和Alluxio缓存。

[0021] 在客户端创建以key-value键值存储的Local Cache,并在分布式存储设备中创建Alluxio缓存。

[0022] Alluxio缓存:Alluxio缓存支持block单元存储,主要存储合并后的chunk数据,每当应用程序访问到一个不在Alluxio缓存中的chunk时,会把这个chunk从底层存储中取出,并存到Alluxio缓存中,以便后续访问。

[0023] Local Cache:Local Cache以key-value键值存储,位于客户端,主要存储chunk解析出的所有小文件,每当从Alluxio缓存取出一个chunk,chunk中小文件会被解析并存储在Local Cache中。

[0024] 步骤2:在AI训练的数据存储阶段,对数据集基于拟合Batch Size特征进行chunk合并操作。

[0025] AI训练都需要准备和存储数据集,本发明在数据存储阶段将小文件合并为chunk,来保证缓存系统可以更好的管理和存储小文件,具体步骤如下。

[0026] 如图1所示,首先获取所有小文件序列,并对小文件序列进行初始shuffle,以便保证每个chunk中的小文件分布均匀。

[0027] 其次将小文件按照拟合Batch Size特征的规则合并为chunk。AI训练遍历数据时,往往会以批次为单位进行数据读取,由于GPU对2的幂次的批次可以发挥更佳的性能,因此batch size通常被设置成16、32、64、128等。基于此特征,采用将小文件按照2的N次方合并规则合并为chunk,N可以由用户自定义。假设Batch Size设置为2的M次方,在数据访问中普遍存在以下三种情况。(1)若M=N,那么访问一个批次的数据则为一个chunk的小文件数量组成。(2)若M>N,那么访问一个批次的数据由k个chunk的小文件数量组成,$k=2^{M-N}$。(3)若M<N,那么访问k个批次的数据由一个chunk的小文件数量组成,$k=2^{N-M}$。因此,该合并规则普遍情况均能很好的配合AI训练的数据访问。

[0028] 最后在客户端保存小文件到chunk的映射文件,当应用程序发起对某一小文件的I/O请求时,则通过该映射信息转换为对chunk的I/O请求。

[0029] 步骤3:在AI训练迭代前,分析chunk的缓存状态,对chunk中的小文件序列采取chunk shuffle+group内小文件shuffle的双层shuffle操作,生成本次迭代的遍历序列。

[0030] 在小文件合并为chunk后,每个chunk所包含的都是固定的小文件序列,在AI训练迭代前,通常需要对数据进行shuffle以此满足遍历数据的随机性。为了能够更好的拟合AI训练的随机访问操作,本发明设计了一个可靠的shuffle策略。

[0031] 首先基于chunk的缓存状态将chunk分为两个序列,分别对应着位于缓存中的chunk序列(图2中的序列1),以及不在缓存中的chunk序列(如图2中的序列2)。接下来对序列1和序列2进行双层shuffle。

[0032] 第一层shuffle:第一层shuffle为chunk shuffle。将每一个序列中的chunk顺序进行打乱。以图2中序列2为例,序列2经过第一层shuffle后,chunk内的小文件顺序不变,而序列中的chunk顺序发生了改变。

[0033] 第二层shuffle:第二层shuffle为group内小文件shuffle。分别将两个序列中的chunk分组,若干个chunk分为一个group,获取每个group内的全部小文件序列并对小文件进行group内shuffle操作。以图2中序列2为例,经过分组,chunk被分为了group2与group3,group2与group3中的小文件分别在group内进行了彻底的shuffle,但group之间仍然为顺序排列。

[0034] 最后,将两个小文件序列前后拼接,组成本次迭代中AI训练的文件访问序列,此时访问序列不仅进行了合理的打乱,还将缓存中已经存在的数据优先放在了访问序列前部,高效利用了缓存中已有的数据。

[0035] 步骤4:在AI训练迭代时对重复I/O采用Local Cache短路读取,并同时,采用异步分组预读方法将后续chunk预读到Alluxio缓存

[0036] 当准备完AI训练的遍历序列,则可以开始进行数据遍历,本发明在AI训练迭代时,对重复I/O采用Local Cache短路读取,并同时,采用异步分组预读方法将后续chunk预读到Alluxio缓存,从而提高缓存命中率和数据访问速率,具体步骤如下。

[0037] 步骤1:在访问一个group的某一个小文件,判断该小文件是否已存在于Local Cache,若不存在执行步骤2,若存在,则执行步骤3。

[0038] 步骤2:将对该小文件的I/O请求转换为对chunk的I/O请求。该I/O请求会从底层存储系统中访问chunk,并存入Alluxio缓存。随后解析chunk并从中取出目标小文件,同时将当前chunk中所有的小文件存入Local Cache。接下来跳转到步骤5。

[0039] 步骤3:若目标小文件已存在于Local Cache,则说明该小文件对应的chunk已被访问过,此时判断为重复I/O,在客户端拦截此次发起对该小文件的I/O请求,采取Local Cache短路读取的方式,直接在Local Cache中取出目标小文件,随后执行步骤4。

[0040] 步骤4:为了避免Local Cache耗尽,当从Local Cache访问完某个小文件后,会从Local Cache中删除当前文件,随后跳转到步骤5。

[0041] 步骤5:判断此时是否已经访问完一个group中的全部chunk,即图3中访问group1中的chunk c后,Local Cache中存储了chunk a、chunk b、chunk c中的全部小文件,则后续访问均为Local Cache短路读取。因此采取异步分组预读,在后续访问的同时,对底层存储系统发起对下一个group中的chunk的I/O请求,将chunk预读到Alluxio缓存。并同时跳转到步骤1继续访问后续文件。

[0042] 本发明在AI训练迭代时对重复I/O采用Local Cache短路读取,因此应用程序不需要反复发起I/O请求访问同一个chunk,这减少了I/O请求数量,加速数据访问。并同时,采用异步分组预读方法将后续chunk预读到Alluxio缓存,提升缓存命中率,加快缓存响应。

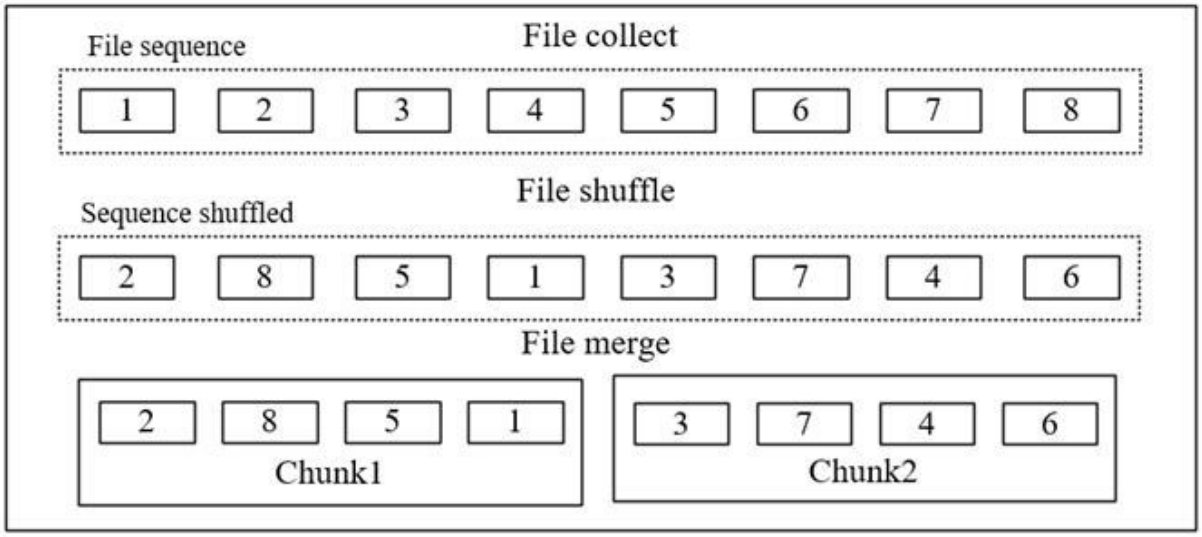

File sequence
File collect
1 2 3 4 5 6 7 8
File shuffle
Sequence shuffled
2 8 5 1 3 7 4 6
File merge
2 8 5 1
Chunk1
3 7 4 6
Chunk2

图1

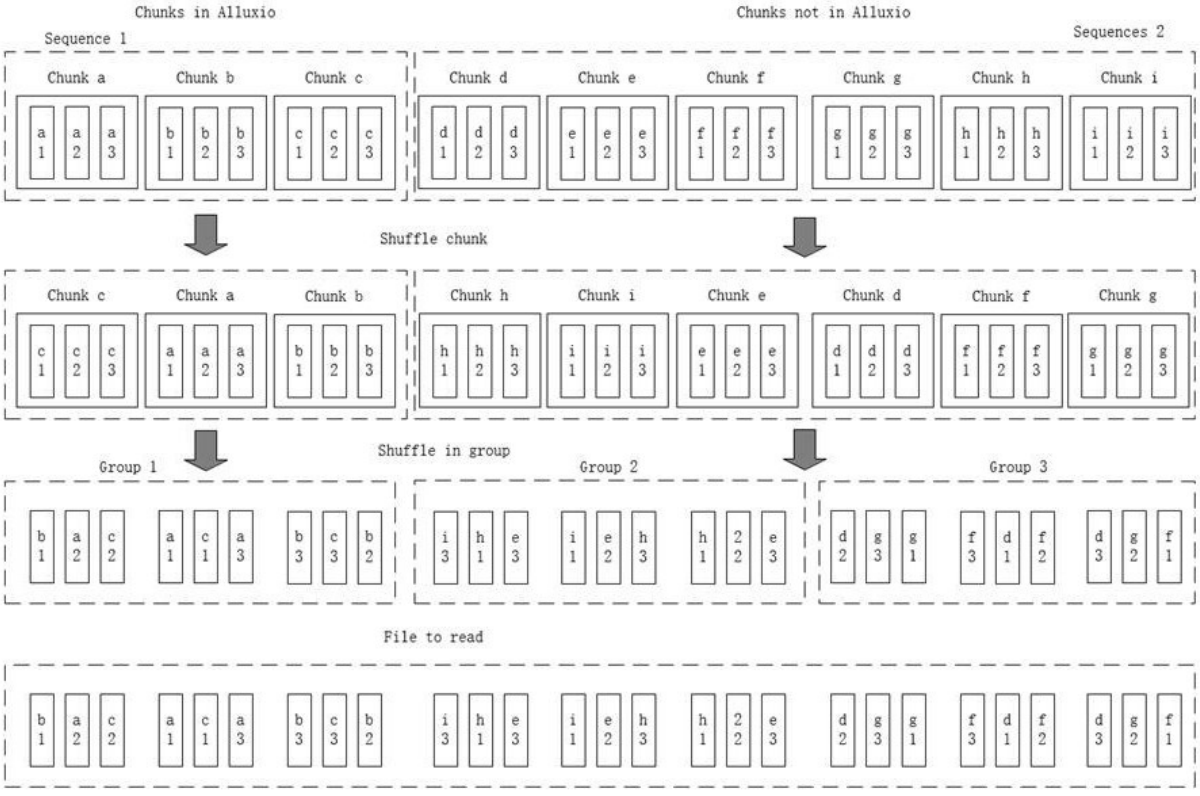

Chunks in Alluxio
Chunks not in Alluxio
Sequence 1
Sequences 2
Chunk a
Chunk b
Chunk c
Chunk d
Chunk e
Chunk f
Chunk g
Chunk h
Chunk i
Shuffle chunk
Shuffle in group
Group 1
Group 2
Group 3
File to read

图2

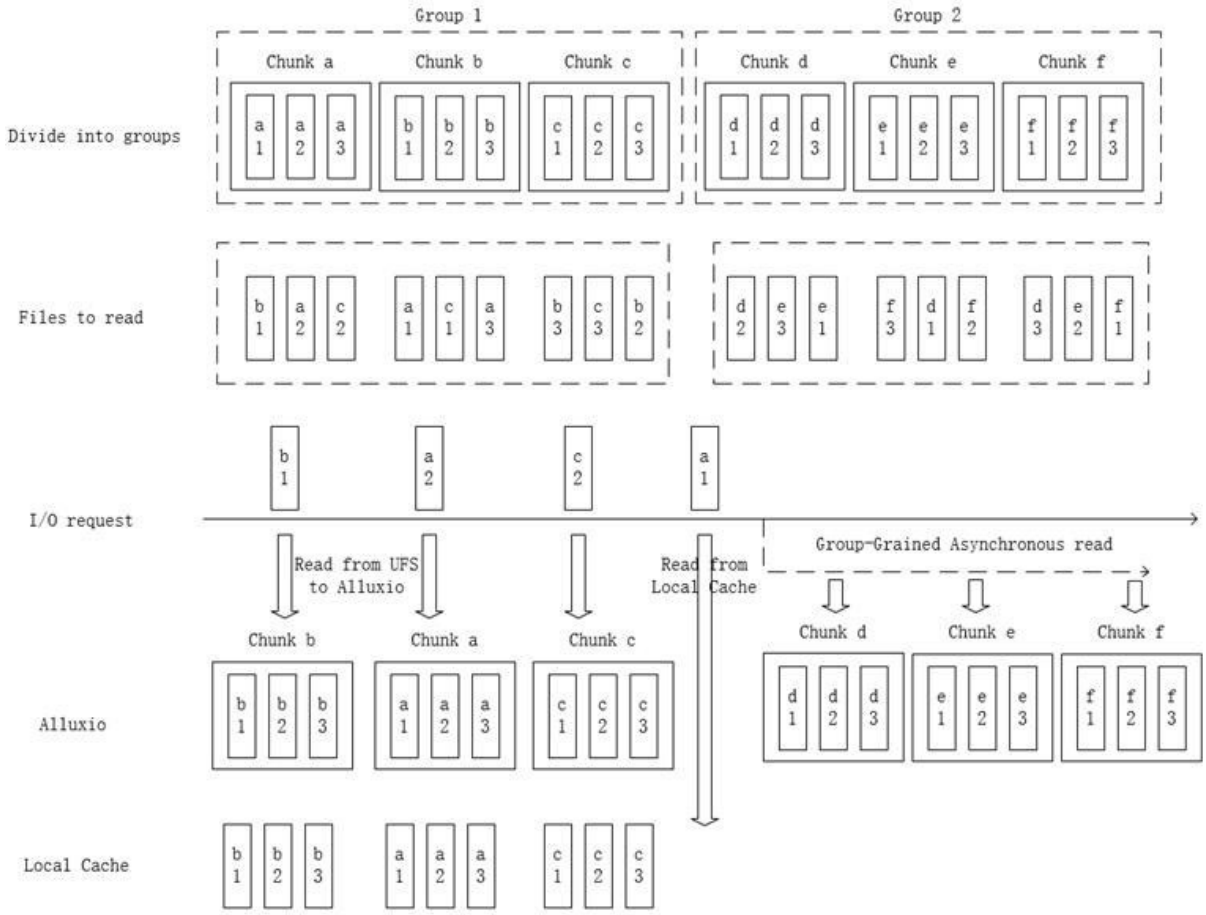

Group 1
Group 2
Chunk a
Chunk b
Chunk c
Chunk d
Chunk e
Chunk f
Divide into groups
Files to read
I/O request
Read from UFS to Alluxio
Read from Local Cache
Group-Grained Asynchronous read
Alluxio
Local Cache

图3