第 11 卷 第 3 期
2022 年 5 月
集 成 技 术
JOURNAL OF INTEGRATION TECHNOLOGY
Vol. 11 No. 3
May 2022

引文格式：
杨振宇, 吕敏, 李永坤. 纠删码存储下的离线批处理作业性能优化 [J]. 集成技术, 2022, 11(3): 85-97.
Yang ZY, Lv M, Li YK. Performance optimization of offline batch jobs in erasure-coded storage systems [J]. Journal of Integration Technology, 2022, 11(3): 85-97.

纠删码存储下的离线批处理作业性能优化

杨振宇 吕 敏* 李永坤

（中国科学技术大学计算机科学与技术学院 合肥 230026）

摘 要 随着互联网数据的爆发式增长，越来越多的分布式存储系统开始引入纠删码存储机制，以在提供数据可靠性的同时降低存储开销。但纠删码机制的引入改变了数据放置模式，从而影响分布式系统上层业务的数据访问和运行效率。在异构 Hadoop 集群环境中，一类典型的离线批处理作业——MapReduce 应用在条带式纠删码存储模式下需要从多个节点访问数据，该“一对多”的数据访问模式由于节点性能差异造成应用执行效率下降。对此，该文提出了一种基于异构环境的数据放置和任务分配策略。通过对异构集群中各节点的硬件参数和历史负载进行分析，将同一纠删码条带的数据块尽可能分布在性能相近的节点上；在系统进行任务分配时，针对各节点当前负载和运算能力确定节点的任务并发度，以平衡各节点计算资源的占用情况，从而避免因数据访问或计算过程中的资源竞争产生极端缓慢任务以致降低整个 MapReduce 应用的运行效率。实验结果表明，相比当前 Hadoop 默认的随机数据放置和任务分配策略，该文提出的异构感知数据放置策略和动态任务分配策略能够在不同类型的 MapReduce 应用中有效削弱任务的长尾效应，使得作业整体运行时间节约 10.5%～42%，验证了该方案的有效性。

关键词 分布式存储系统；纠删码存储系统；离线批处理作业；MapReduce 应用；数据布局；任务调度
中图分类号 TP 338.8 **文献标志码** A doi: 10.12146/j.issn.2095-3135.20211026001

Performance Optimization of Offline Batch Jobs in Erasure-Coded Storage Systems

YANG Zhenyu LV Min* LI Yongkun

(*School of Computer Science and Technology*, *University of Science and Technology of China*, *Hefei* 230026, *China*)

**Corresponding Author*: *lvmin05@ustc.edu.cn*

Abstract With the explosive growth of Internet data, many distributed storage systems have integrated erasure-coding mechanisms to ensure data reliability, while further reducing storage overhead. However,

收稿日期：2021-10-26 **修回日期**：2021-12-05
作者简介：杨振宇，硕士研究生，研究方向为分布式容错系统；吕敏（通讯作者），副教授，主要研究方向为分布式容错系统、图计算，E-mail：lvmin05@ustc.edu.cn；李永坤，副教授，主要研究方向为存储系统、虚拟化。

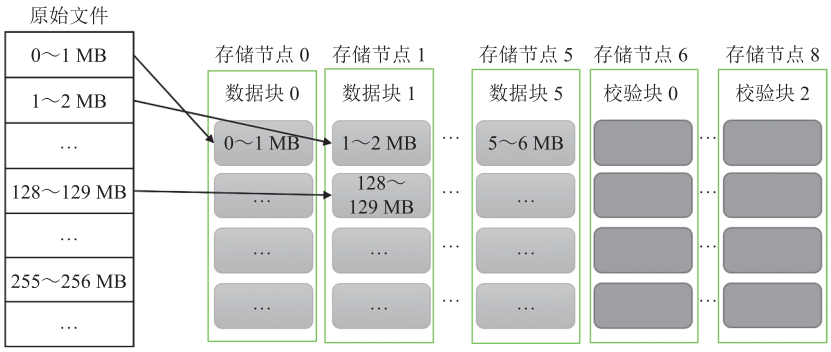
erasure-coding has changed the data placement scheme, thus affecting the data access of other services of the cluster. This paper proposes a new data placement scheme and a task scheduling strategy based on heterogeneous Hadoop cluster that can be better adapted to the “one-to-many” data access scenarios of a typical offline batch job——MapReduce applications. By analyzing the hardware parameters and historical load of each node in a heterogeneous cluster, the data blocks of the same erasure coded stripe are distributed as many as possible on nodes with similar performance. This way ensures that the data access pressure to each node of the cluster during the execution of the MapReduce job achieves relatively balanced state. In addition, when the system schedules tasks, the task concurrency of nodes is determined according to the current load and computing power of each node and so to avoid straggler task caused by heavy load in some nodes and optimize the progress of the MapReduce job. The experimental results show that compared with the default random data placement and task allocation strategy in Hadoop, the data layout strategy Heterogeneous-aware Data Placement Algorithm (HDPA) and the task allocation strategy Dynamic Task Allocation Algorithm (DTAA) proposed in this paper can effectively reduce the long tail effect of tasks in different types of MapReduce applications, thus reducing the running time by 10.5%~42%.

Keywords distributed storage systems; erasure-coded storage systems; offline batch jobs; MapReduce applications; data layout; task scheduling

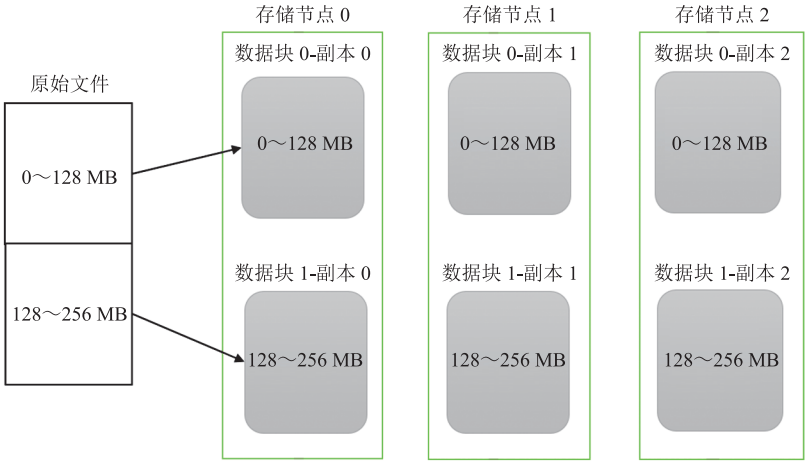
1 引 言

在主流的分布式存储系统中，为了保证数据的可靠性，往往采用多副本方式将原始数据的多个备份存储在不同节点或设备上以达到容错要求。但随着互联网数据的爆发式增长[1]，相较于原始数据量，多副本方式产生的存储开销成倍数增长，这已成为不可忽视的问题。当前，越来越多的分布式存储系统[2-3]开始引入纠删码机制[4]来保证系统的可靠性。纠删码通过将原始数据进行条带分组和编码来得到冗余的校验单元，并将整个条带的数据单元和校验单元分散放置在不同节点，在保证系统容错需求的同时，其存储开销大大降低。然而，纠删码机制的引入改变了数据的组织方式，对上层业务的数据访问产生了新的影响。图 1(a)展示了 Hadoop 分布式文件系统(Hadoop Distributed File System，HDFS)3.x 版本[3]中条带式纠删码存储模式下的数据放置[5]，与图 1(b)所示的 HDFS 在三副本存储模式下的数据放置相比：纠删码模式在保留 HDFS 数据块划分的同时，将原始文件进行了更细粒度的划分以进行条带编码；在副本模式下，一个数据块一般对应原始文件连续的一部分(数据块大小可进行配置)，而在纠删码模式下，一个数据块中的内容则对应原始文件中一段段不连续的数据。

条带式纠删码的数据布局直接影响上层业务的数据访问。以 Hadoop 基础架构下 MapReduce 应用的数据处理流程为例，MapReduce 将计算任务分为 Map 和 Reduce 两个阶段，在 Map 阶段并发地执行多个 Map 任务来对数据集进行访问和处理，每个 Map 任务处理原始数据集中连续的一部分。在 HDFS 三副本存储模式下，为了实现“计算向数据靠拢”的目的，每个 Map 任务处理的数据量默认与数据块大小保持一致，以尽可能减少计算任务所在节点向其他多个节点请求数据的情况。而在以图 1(a)为例的 RS-(6,3)



(a) RS-(6,3)条带式纠删码的数据布局



(b) 三副本存储模式的数据布局

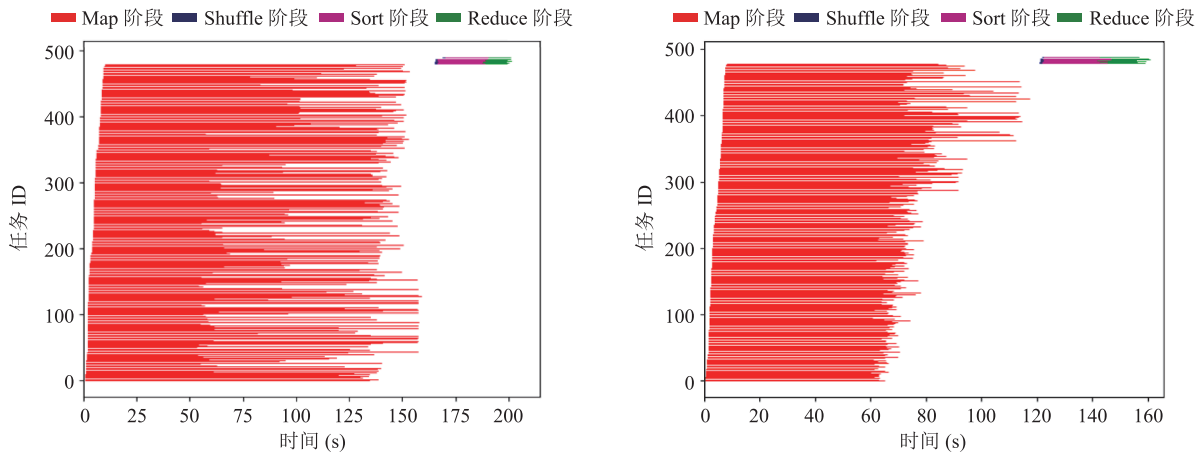
图 1　HDFS 三副本与 RS-(6,3)纠删码存储模式下的数据布局示意图

Fig. 1　Data layouts of replication storage mode and RS-(6,3) erasure-coded storage mode in HDFS

纠删码存储模式下，一个 Map 任务需要获取的数据对应分布在整个条带的所有数据块节点上，计算节点不可避免地要从多个存储节点访问所需要的数据。在大多数情况下，集群各节点的硬件设备(磁盘、网络带宽、内存容量等)存在异构情况，集群各节点运行时的负载状况也不尽相同。因此，当 Map 任务在访问纠删码存储模式下的数据时，节点性能差异导致的数据传输性能波动将会极大地影响任务的运行效率。如图 2 所示，结合表 1 的相关度量指标，相较于三副本存储模式，在纠删码存储模式下上层运行的 Map 任务呈现出更为严重的不均衡性，任务完成用时接近三副本存储模式的 3 倍。

目前，国内外学者针对 MapReduce 应用的性能优化主要聚焦于多副本存储模式下的数据放置策略。如 Wang 等[6]通过分析 HDFS 数据的历史访问记录，发掘数据间的关联性并重新组织数据布局，使得运行 MapReduce 应用时能最大化地利用数据本地性优势，尽可能减少跨网络的数据传输，提高 MapReduce 应用的运行效率。此外，针对集群的异构环境，Xiong 等[7]通过选取节点关键性能参数对异构集群进行逻辑分层，并结合数据访问热度决定副本数量和放置位置，从而提高异构环境下 MapReduce 应用的运行效率。

关于纠删码在存储中的应用，目前大量研



(a) RS-(6,3)纠删码存储模式　　(b) 三副本存储模式

图 2　RS-(6,3)纠删码与三副本存储模式下 MapReduce 的任务运行时间分布

Fig. 2　Task running time distribution of MapReduce applications in replication storage mode and RS-(6,3) erasure-coded storage mode

表 1　RS-(6,3)纠删码与三副本存储存储模式下 Wordcount 作业运行情况统计

Table 1　Wordcount Application's running statistics in replication storage mode and RS-(6,3) erasure-coded storage mode

存储模式	Map 阶段用时(s)	作业完成用时(s)	Map 任务平均用时(s)	最快 Map 任务用时(s)	最慢 Map 任务用时(s)	Map 任务完成标准差(s)
RS-(6,3)纠删码存储模式	161.5	207.2	89.1	48.0	153.4	33.6
三副本存储模式	120.5	162.0	79.9	56.8	109.2	17.0

究工作集中于优化纠删码机制引入的编解码操作在实际系统中的数据更新[8]及故障修复等性能[9]。如 Wang 等[10]提出一种基于异或网络计算的数据更新及修复流程，通过在复杂网络拓扑下选取合适的交换机卸载部分编解码操作，降低纠删码机制带来的网络传输流量，从而优化系统的更新和故障修复性能。此外，也有学者利用纠删码的编解码特性对系统访问请求进行负载均衡，如 Rashmi 团队[11]在内存对象缓存系统 Alluxio 中引入纠删码机制取代原来的选择性复制策略，当某些数据的访问热度较高时，将数据分片的一部分访问请求转移到访问条带的校验分片，再进行解码操作恢复原始数据，从而实现系统的访问负载均衡。

部分学者也尝试在 Hadoop、Spark 等分布式计算框架中通过引入纠删码进行中间结果的容错。如 Yao 等[12]将纠删码用于 Spark 的 Shuffle 阶段对弹性分布式数据集进行编码，避免部分节点发生故障时，系统运行的部分任务需要重新进行。Tandon 等[13]和 Wang 等[14]在分布式机器学习框架中引入了纠删码对分布式神经网络计算过程的中间梯度结果进行编码。通过引入冗余校验信息，主节点可在收取一部分从节点的计算结果后就恢复出全局结果，从而减少网络传输过程中因部分节点的延迟造成主节点过长的同步等待时间。Li 等[15]更改了 MapReduce 应用在 Shuffle 阶段的网络传输模式，通过增加本地重复性计算对临时结果进行编码，而后利用网络多播传输来降低 Reduce 任务需要跨网络获取的数据总量。

综上可知，纠删码存储策略虽然具有存储友

好的优势，但由于其存储模式的变化，上层业务的数据访问性能存在较大的优化空间。本文基于当前 HDFS 的条带式纠删码策略，通过考虑集群的异构情况和长期负载为每个条带选取合适的放置位置，并根据集群各节点运行时的负载选取合适的任务并发量，从而避免产生严重的“掉队”任务，影响整个 MapReduce 应用的执行效率。

2　异构感知的数据放置策略

2.1　异构感知的重要性

在实际的集群环境中，随着时间扩展，各节点的硬件设备、节点间网络拓扑往往是异构的[7]，并且由于各节点运行时负载不尽相同，统一配置的参数无法很好地适应异构环境下的任务处理。目前，已有许多学者针对集群异构情况，基于分布式机器学习系统[14]或分布式存储系统[7]设计了合适的任务分配和数据存储策略。在 Hadoop 中，条带式纠删码默认的随机放置策略仅仅考虑机架间的容错，对集群异构环境并未进行针对性的优化，对数据块和校验块未进行区分，在任务分配时也并未考虑系统运行时各类资源(CPU、内存、网络等)的占用情况。如图 3 所示，通过将测试数据集的部分条带完全放置于高性能固态硬盘(NVME SSD)(对应区域 2)，另一部分条带随机放置在普通机械硬盘并限制上行网络带宽来模拟节点数据访问性能差异(对应区域 1)，此外还在部分节点运行 Linux Stress 压力测试工具模拟较高的 CPU 和内存占用(对应区域 3)，可以观察到明显的 Map 任务运行用时差异。因此，根据集群各节点自身的硬件情况与运行负载来设计合适的数据放置策略以弥合 Map 任务的用时差异具有相当大的优化空间。

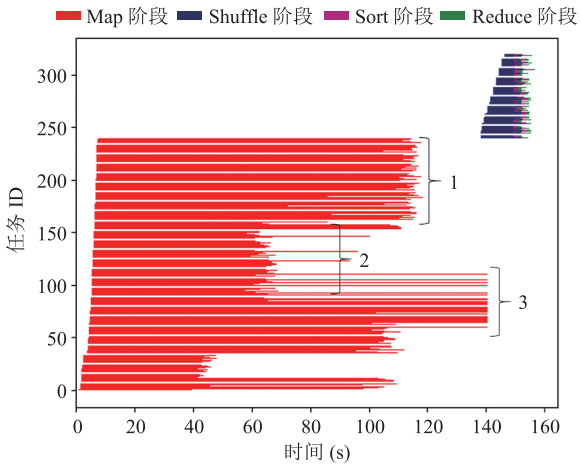


图 3　硬件异构对任务运行时间的影响

Fig. 3　Impact of heterogeneous hardware and load differences on task running time

2.2　数据放置算法

针对前面提到的集群异构环境，本文提出一种异构感知数据放置策略(Heterogeneous-aware Data Placement Algorithm，HDPA)：首先对各节点影响 HDFS 数据访问的关键硬件参数进行统计，而后根据各节点相关硬件的历史负载进行分析，为各节点设备计算出一个常量负载因子，最后基于上述两点对集群节点进行性能分组，使同一个条带的数据块尽可能位于同一个节点组内。

2.2.1　节点划分策略

根据 MapReduce 的数据访问模式可知，影响 Map 任务数据读取的因素主要有数据存放节点的磁盘顺序读写性能和网络上行带宽，以及任务执行节点的网络下行带宽等；影响 MapReduce 任务计算的因素主要有 CPU、内存，以及磁盘的顺序读写、随机读写性能等，其中关于 MapReduce 任务计算的负载均衡优化将在任务分配算法中实现。

本文先按照影响任务数据读取的节点数据传输性能将所有节点进行分组，数据传输性能将结合节点本身的硬件参数及历史负载来衡量。通过访问节点日志，对节点相关硬件设备较长一段时间的历史负载(利用 Node Exporter、Prometheus 以及 Grafana 等集群性能监控组件以分钟为粒度采集)进行分析，得到节点的长期后台数据访问负载情况，用常量化的负载因子 α 表示，单位为数据传输速度 MB/s，具体计算方式如下：

对于磁盘顺序读写，统计节点磁盘大于 80% 时间中的负载峰值作为长期后台负载，记作 α_{diskIO}。

对于网络传输，分别统计节点大于 80% 时间中的网络上、下行负载峰值，将其作为网络传输的长期后台负载，分别记作 $\alpha_{NetUP}, \alpha_{NetDown}$。

根据系统中所有节点的数据传输性能情况，确定节点分组的阈值 θ(单位为 MB/s)，节点将被划分为 G_{total} 组，每组内节点的数据访问性能差异不超过 θ。

$$G_{total}=\frac{\text{Max}\left(\text{Min}\left(R_{\text{disk}}-\alpha_{\text{diskIO}}, NetUP-\alpha_{NetUP}\right), \text{Min}\left(W_{\text{disk}}-\alpha_{\text{diskIO}}, NetDown-\alpha_{NetDown}\right)\right)}{\theta} \quad (1)$$

其中，$R_{\text{disk}}, W_{\text{disk}}$ 为磁盘的顺序读、写性能；*NetUP*, *NetDown* 为节点上、下行网络带宽(单位均为 MB/s)。

基于上述划分方式，本文通过设计合适的数据放置策略可以实现向 HDFS 写入数据时将同一条带的数据块尽量分布在数据传输性能相近的节点组，使 Map 任务向多个节点读取数据时不会因为节点性能差异过大而导致数据传输时间过长，从而有效缩短 Map 任务的数据访问用时。

2.2.2 数据放置策略

在为条带选取放置位置时，必须保证系统的可靠性。当前 HDFS 纠删码模式并不对数据块和校验块的放置进行区分，而是通过随机放置的方式尽可能容错多个机架，然而在真实环境中单磁盘、节点以及单机架故障这类情况占绝大多数[16]。因此，为了更容易为条带找到合适的存储节点，并尽量减少 MapReduce 应用运行产生的跨机架网络流量，鉴于多个机架同时发生故障的可能性较小，在满足实际需求的前提下，HDPA 将只保证单个机架的故障容错。与此同时，为了保证节点在正常状态下的数据访问性能，HDPA 在选取条带位置时将对条带的数据块和校验块进行区分。当 HDPA 在集群同一节点组内无法找到存放完整条带的节点数目时，将会优先将条带拆分并把校验块放在性能相对较弱的节点组。具体流程如算法 1 代码所示：

算法 1 HDPA 数据放置策略

```
输入：写入文件 F，纠删码策略 δ，划分阈值，集群各节点信息 NI
输出：无
1:  init: Node_Group = {}
2:  init: Stripe_Set = Compute stripe num(F,δ)
3:  for node_i in Cluster do:
4:      set node_i in NG_tmp = Compute node perf(NI_node_i)
5:      if NG_tmp not in Node_Group then:
6:          add NG_tmp to Node_Group
7:      endif
8:  end
9:  Refresh NG order(Node_Group)
10: for S_t in Stripe_Set do:
11:     for NG_i in Node_Group do:
12:         place_flag = Check NG avaliable(S_t,NG_i)
13:         if place_flag = true then:
14:             Select storage node(NG_i)
15:             Write current stripe(S_t)
16:             Refresh NG order(Node_Group)
17:         else:
18:             place_flag = Check datablock NG avaliable(S_t,NG_i)
19:             if place_flag = true then:
20:                 Select datablock storage node(NG_i)
21:                 Rest_Group = Refresh NG order(Node_Group\NG_i)
22:                 Select parityblock storage node(Rest_Group)
23:                 Write current stripe(S_t)
24:                 Refresh NG order(Node_Group)
25:             endif
26:         endif
27:     end
28: end
```

HDPA 算法替代默认随机放置策略集成在 HDFS 主节点的核心步骤如下：

Step 1. 在 Client 端写入文件，确定文件被划分的条带个数；在主节点端开始为每个条带选取合适的放置位置。该步骤对应算法 1 第 1～2 行的初始化设置。

Step 2. 设置节点性能分组阈值 θ，根据公式(1)可以确定集群的节点分组数 G_{total}，并确定各节点对应的分组 G_i，i=1, 2, …, *total*(其中 *total* 为集群划分的节点组总数)。该步骤对应算法 1 第 3～8 行，主节点将根据各从节点硬件资源配置将集群各节点进行分组。

Step 3. 主节点统计每一个节点组的平均存储

占用率：

$$SG_i=\frac{\sum_{N_j\in G_i}S\left(N_j\right)}{\left|G_i\right|}$$

其中，$S\left(N_j\right)$ 为节点组 G_i 中节点 N_j 的存储占用率。接下来主节点将对各节点组按存储占用率升序排序，具体在算法 1 第 9 行的函数中实现，用以确保接下来的数据放置对节点的存储占用达到相对均衡的状态。此外，主节点还将分别记录各节点上的数据块和校验块占该节点存储数据块的比例，记作 DBR_{N_j},PBR_{N_j}。

Step 4. 在为每一个条带选取放置位置时，将按 Step 3 顺序遍历节点组，检测该节点组是否能够存放当前条带，同时保证集群的单机架容错特性，即对于 RS-(k,m)编码的条带，该节点组内至少有 $\lceil\frac{k+m}{m}\rceil$ 个机架，且每个机架内最多可放置 m 个数据块或校验块。对应功能由算法 1 第 12 行函数进行处理。

Step 5. 若节点组满足 Step 4 的要求，则组内节点按 $<S\left(N_j\right),DBR_{N_j}>$ 这一有序数对的升序存放整个条带的数据块和校验块，以达到组内各节点的存储占用以及数据块、校验块占比均衡。完成当前条带位置选取后，更新节点组存储占用情况并返回 Step 3 继续处理下一个条带。该功能对应算法 1 第 13～16 行的处理分支。若节点组不满足 Step 4 的要求，则进入 Step 6。

Step 6. 如算法 1 第 16～25 行分支所示，若节点组无法存放整个条带，则考虑将条带的数据块和校验块分开存放，以提高各节点组的存储空间利用率。若当前节点组满足存放数据块的要求，则组内节点按 $<S\left(N_j\right),DBR_{N_j}>$ 升序来进行数据块放置，然后继续遍历余下节点组为条带校验块寻找放置位置，满足要求的节点组按组内节点 $<S\left(N_j\right),PBR_{N_j}>$ 的升序确定校验块位置；若当前节点组无法容纳条带数据块，则按优先级遍历余下节点组继续查找。

上述 HDPA 算法保证了 HDFS 集群正常读写性能和容错需求，同时提高了系统在 MapReduce 这类应用场景下的数据读取性能。当节点磁盘及网络负载总体较高时，上述 HDPA 算法优化了 Map 任务执行时因节点性能差异导致数据传输时间过长的问题。

3　负载感知的任务分配算法

当前 Hadoop 版本通过另一种资源协调者（Yet Another Resource Negotiator，YARN）框架对集群计算资源进行管理，用户通过在每个节点的配置文件中设置参数划分节点可用的计算资源（主要是 CPU 核心和内存可用量），因此 YARN 确定了集群的最大任务并发度（即可同时运行 Map/Reduce 任务的最大数量）。但这种确定的节点管理方式在一定程度上无法适用于集群的异构环境和运行负载。如图 3 的区域 3 所示，即使在相同硬件配置的集群中，在 CPU 负载较高的节点上运行相同数目的 Map 任务，运行时间也会被大大拉长，甚至会由于运行时间过长而导致任务失败。

与 HDPA 类似，本文基于集群异构场景和节点硬件负载情况，提出一种动态的任务分配算法（Dynamic Task Allocation Algorithm，DTAA），其核心在于 YARN 会在 MapReduce 应用进行任务分配前采集集群各节点的计算资源负载，并为每个节点确定合适的任务分配数量，通过动态调节集群的任务并发度避免因计算资源竞争造成任务运行效率低下的问题。

3.1　计算资源的动态限制

不同于 YARN 默认情况下，主节点通过各从节点访问节点本地的配置文件并向主节点发送心跳包来确定集群可用的计算资源情况（在 Hadoop 中称为 Container），DTAA 在各节点向主节点发送的心跳包中额外添加了 Preferred-Container 信

息，其含义是通过访问节点当前硬件负载情况，结合硬件参数，计算出在不影响节点运行效率的情况下节点自身可用的计算资源数量，以实现计算资源的动态限制。各节点的 Preferred-Container 数量(简记为 Num_{pc})计算如下：

$$Num_{pc}=\mathrm{Min}\left(\frac{v_{\mathrm{Cores}}\times\left(Num_{\mathrm{Core}}-\gamma_{\mathrm{cpuload}}\right)}{Num_{\mathrm{Core}}},\frac{v_{MEM}\times\left(RAM-\gamma_{\mathrm{ramload}}\right)}{RAM\times MEM_{\mathrm{task}}}\right) \quad (2)$$

其中，v_{Cores},v_{MEM} 分别是节点配置文件中设置的可用 CPU 核心数和内存分配容量；$\gamma_{\mathrm{cpuload}}$,$\gamma_{\mathrm{ramload}}$ 分别是节点上的 CPU、内存负载情况；Num_{Core},RAM 分别是节点 CPU 核心数和内存容量的硬件信息。由此可以根据节点负载情况对 YARN 的默认参数进行动态调整。

3.2 任务选取

根据 3.1 节的资源动态限制，可以得知集群的两类关键信息：MapReduce 应用需要处理的数据集合、当前集群的可用计算资源。接下来需要解决任务分配问题，存在两种情况：

(1)数据量小，需要分配的 Map/Reduce 任务数量小于集群并发度；

(2)数据量大，需要分配的 Map/Reduce 任务数量大于集群并发度，涉及到节点计算资源的分配、释放与再分配。

当前 Hadoop 在运行 MapReduce 应用时的默认策略是正常情况下 Map 任务访问条带的数据块，当 Map 任务运行效率过低或访问数据块失败时重新执行任务并访问条带的校验块进行数据修复，而重新执行和超时机制造成了集群资源的浪费与 Map 阶段用时的增加。

上述两种任务情况存在相同的问题。如图 4，Map 任务仅仅访问条带数据块(蓝色)，容易在不同节点产生数据读取热度差异，并可能产生部分节点磁盘负载和网络流量的瓶颈，影响任务的执行。因此，DTAA 引入主动降级读取机制，在 Map 任务运行速度过慢时根据任务进度缓慢的不同原因选取对应的解决策略。DTAA 在 Map 任务的执行逻辑中引入数据等待计时器，分别统计 Map 任务在数据读取和计算两个过程的用时情况，判断可能的瓶颈位置。若 Map 任务在数据读取时等待数据同步的用时相对较长，则额外随机访问条带的一个校验块来替换条带数据块响应最慢的节点，从而降低一定的时间开销。

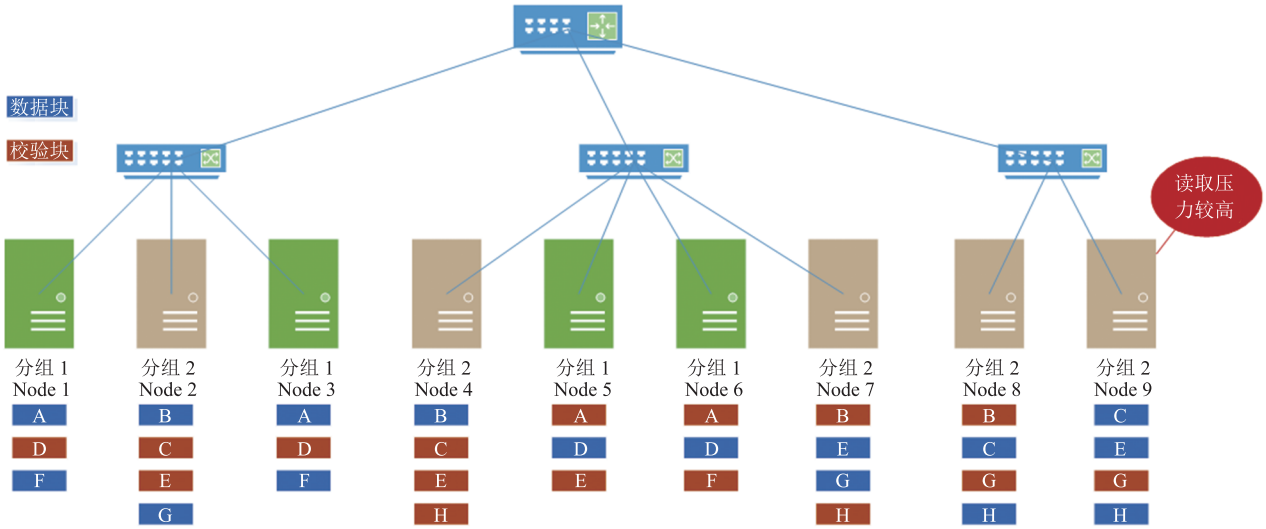


图 4 任务对应数据块访问不均衡

Fig. 4 Unbalanced datablock read in task running

针对上述情况，DTAA 均给出了对应的任务分配方式进行优化，具体流程如算法 2 中伪代码所示，步骤概括如下：

算法 2　DTAA 任务分配策略

输入：提交作业 $\mathcal{J}$，集群各节点负载信息 OL
输出：无
1: init: MapTask_Set, ReduceTask_Set based on submitted Job $\mathcal{J}$
2: init: each node default container $node_i\_CN$, Cluster total container C_{total}
3: for $node_i$ in Cluster do:
4: $node_i\_PCN$ = Compute Preferred-Container Num(OL_{node_i})
5: end
6: Calculate Cluster Total PC Num PC_{total}
7: if NUM(MapTask_Set) < PC_{total} then:
8: task allocation ratio ρ = NUM(MapTask_Set)/PC_{total}
9: for $node_i$ in Cluster do:
10: allocating $node_i\_PCN \times \rho$ maptask
11: end
12: else if PC_{total} < NUM(MapTask_Set) < C_{total} then:
13: task allocation ratio ρ = NUM(MapTask_Set)/C_{total}
14: for $node_i$ in Cluster do:
15: dynamic allocating maptask(PC_{total}, C_{total}, ρ)
16: end
17: else:
18: allocating $node_i\_PCN$ maptask to $node_i$
19: while still maptask to be allocated do:
20: allocating maptask to released Preferred-Container
21: endwhile
22: endif

Step 1. 各从节点定期采集自身 CPU、内存等负载情况，并根据公式(2)计算 Preferred-Container 数量，附加在向主节点发送的心跳包信息中。如算法 2 第 2～6 行所示，主节点获悉集群默认 Container 资源总量和 Preferred-Container 资源总量。

Step 2. 提交 MapReduce 应用时，主节点启动应用服务进程开始计算 MapReduce 应用的资源需求并向主节点申请资源。对应算法 2 第 1 行的初始化设置。

Step 3. 当 MapReduce 应用的任务规模小于集群 Preferred-Container 总数时，主节点将按各节点的 Preferred-Container 数量等比例分配任务。对应算法 2 中第 7～11 行条件分支。否则进行 Step 4 操作。

Step 4. 当 MapReduce 应用的任务规模小于集群默认的 Container 配置数量，主节点将按各节点 Preferred-Container 相对比例进行可用 Container 的分配与限制。对应算法 2 中第 12～16 行条件分支。例如，6 个节点的默认 Container 资源均为 8，MapReduce 应用的 Map 任务数为 34，在 6 个节点上计算出的 Preferred-Container 分别为 3、5、5、6、4、3，此时各节点对应的任务分配数量为 4、7、7、8、4、4。若 MapReduce 应用的任务规模超过集群默认的 Container 配置数量，则进行 Step 5 的操作。

Step 5. 若 MapReduce 应用的任务规模超过集群默认的 Container 配置数量，Map 任务则需多次分配运行，每一次任务分配数量以各节点的 Preferred-Container 数量为限制，直至所有 Map 任务分配完成。对应算法 2 中第 17～21 行条件分支。

4　实验结果与分析

4.1　实验环境配置及参数设置

本文实验在一个由 16 台服务器组成的集群上进行，每台服务器的 CPU 为两颗 Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz 组成的 NUMA 节点，标准配置 64 GB 频率为 2400 MHz 的服务器 ECC 内存和万兆网卡，每台节点挂载若干个 1 TB 希捷机械硬盘和三星 EVO860 SSD，每台节点均运行 CentOS 7.9 版本的 Linux 操作系统，集群上运行若干常见分布式系统以及容器环境(Ceph，Redis，Cassandra 等)。

本实验选取一台服务器作为主节点运行，其他机器作为数据节点和任务执行节点。通过对网卡带宽、挂载磁盘等进行设置以及在其他分布式系统上运行不同的 Benchmark 来模拟集群的软硬件异构。实验以 Hadoop-3.3.0 默认的随机数据放置和任务分配作为基准，数据块大小设置为 256 MB，通过在 Hadoop 系统中运行几类常见的 MapReduce 应用，对比本文提出的 HDPA

数据放置和 DTAA 任务分配策略对 MapReduce 应用运行速度的优化结果。

4.2 常见 MapReduce 应用运行情况对比

4.2.1 词频统计

词频统计(Wordcount)作业是 MapReduce 应用中常见的一类数据分析任务，即通过分片处理海量数据统计某些属性的情况(如天气分析、用户地域属性划分等)。本文用 HiBench 生成的 60 GB 文本数据作为测试数据集，实验结果如图 5 所示。

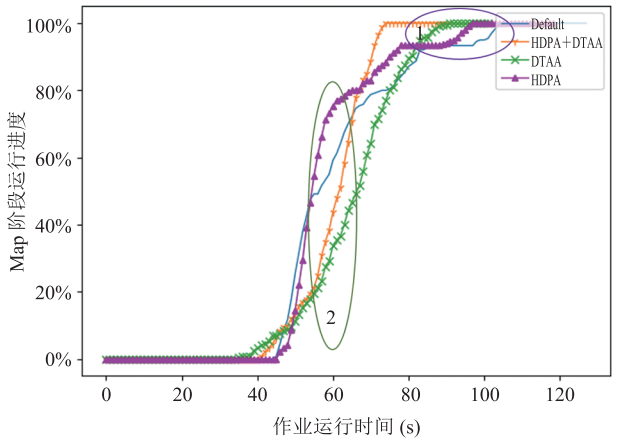


图 5 词频统计作业处理 60 GB 文本数据的作业完成用时

Fig. 5 Completion time of Wordcount Application processing 60 GB text data

结果表明，与 Hadoop 默认的数据放置和任务分配策略相比，本文提出的“HDPA＋DTAA”优化策略能够缩短 MapReduce 作业的运行时间。在三次重复实验中(如表 2 所示)，词频统计 MapReduce 作业在默认设置(蓝色线条)情况下和在“HDPA＋DTAA”方案(黄色线条)下的平均完成时间分别为 128.0 s 和 90.8 s，“HDPA＋DTAA”方案提升约 29%，其中 Map 阶段运行用时分别为 105.7 s 和 74.4 s。

此外，为了进一步分析数据布局和任务分配分别对 MapReduce 作业运行的影响，本文分别测试了 HDPA 和 DTAA 算法对 MapReduce 作业运行的优化效果，如图 5 中紫色和绿色线条所示，当各节点上的任务瓶颈出现在不同位置时，HDPA 和 DTAA 算法的优化效果也不尽相同。如图 5 紫色圈定范围 1 所示，在词频统计作业中监控各任务进程的 CPU、内存及磁盘 IO 占用信息和节点整体资源使用情况发现，对于极端长尾任务，往往由于计算节点 CPU、内存等计算资源发生较为严重的竞争，HDPA 数据访问的优化并不能带来明显的效果，而 DTAA 可以通过各节点均衡计算资源的占用有效缓和极端任务的产生；与之对应，从图中绿色圈定范围 2 可以看出，HDPA 能够有效缓和部分任务完成用时的“参差”情况，因为范围 2 的任务计算资源较为宽松，HDPA 能够均衡任务并发访问多节点数据时各节点的响应性能(与数据所在节点的磁盘和上行带宽占用相关)。综合各方面来看，HDPA 和 DTAA 能够分别对集群并发任务可能面临的不同瓶颈情况进行优化，从而协同达到更好的优化结果。

4.2.2 排序

排序(Sort)任务是 MapReduce 作业中数据全

表 2 Wordcount 应用运行情况统计

Table 2 Wordcount Application's running statistics

运行策略	Map 阶段用时(s)	Reduce 阶段(含 Shuffle)用时(s)	整体运行用时(s)	Map 任务平均用时(s)	最快 Map 任务用时(s)	最慢 Map 任务用时(s)	Reduce 任务平均用时(s)	最快 Reduce 任务用时(s)	最慢 Reduce 任务用时(s)
Hadoop 默认策略	105.7	17.7	128.0	63.8	43.3	84.9	5.7	4.3	17.6
HDPA＋DTAA	74.4	13.2	90.8	56.9	41.2	64.1	5.7	4.3	9.2

量处理的一类代表性任务(如数据分类、归档、批量修改等)，其特点是在纠删码策略下数据在 Map 访问、Shuffle、Reduce 写回阶段均会产生大量磁盘读写和跨网络传输。本文基于 60 GB 向量标签数据集进行 Sort 排序和分类写回，实验结果如图 6 所示：

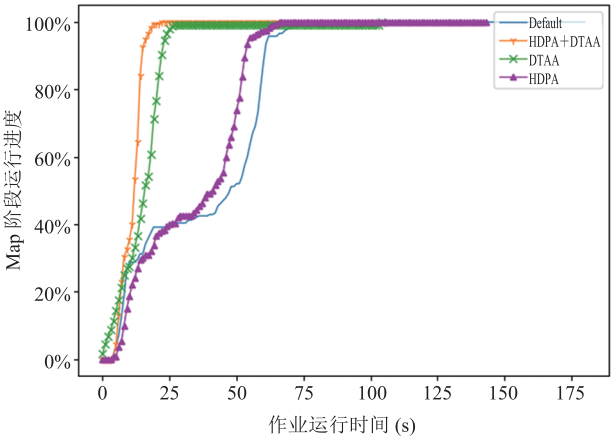


图 6　Sort 应用统计 60 GB 向量标签数据的作业完成用时

Fig. 6　Completion time of Sort Aplication processing 60 GB vector record data

在 Sort 排序应用中，“HDPA＋DTAA”方案的平均完成时间为 104.7 s，与采用默认策略的 180.6 s 相比，提速约 42.0%。一方面，是因为 Sort 排序应用的中间结果在数据量上约等于原始数据，因此在 Map 阶段会产生大量磁盘溢写，不均衡的数据放置会进一步加剧磁盘负载的热度差异；另一方面，是因为各节点的任务分配数量除了影响节点 CPU、内存等计算资源的占用外，也决定了中间结果数据量，从而影响节点磁盘负载。在默认策略下，上述两方面原因共同造成了极端 Map 任务的出现，延长了 Map 阶段的执行时间。如图 6 所示，在分别部署 HDPA 和 DTAA 的实验中，两种策略均体现了一定的优化效果，分别对应了上述两方面的因素。关于 Sort 应用的其他运行的详细信息如表 3 所示。

4.2.3　K-Means 机器学习算法

MapReduce 框架也可用于大规模机器学习任务。本文针对典型的机器学习算法 K-Means 进行测试，Hadoop 通过将 K-Means 聚类算法抽象为多次迭代的 MapReduce 任务得到最终结果，实验结果如图 7 所示。

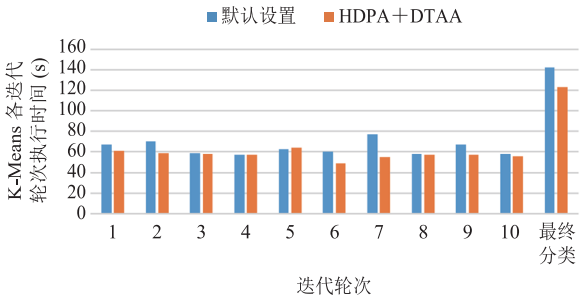


图 7　K-Means 应用计算 75 GB 数据 10 次迭代运行情况

Fig. 7　10 Iteration of K-Means with 75 GB data

K-Means 聚类算法每次迭代均会访问全部数据集进行聚类中心的更新，因此 Map 阶段占据了大部分执行时间。K-Means 应用运行时，数据放置和任务分配是否负载均衡将随着迭代次数增加 K-Means 应用的执行时间。在“HDPA＋DTAA”策略下，10 次迭代以及最终结果写回共计用时 696 s，相较于默认策略下用时 778 s 缩短了

表 3　Sort 应用运行情况统计

Table 3　Sort Application’s running statistics

运行策略	Map 阶段用时(s)	Reduce 阶段(含 Shuffle)用时(s)	整体运行用时(s)	Map 任务平均用时(s)	最快 Map 任务用时(s)	最慢 Map 任务用时(s)	Reduce 任务平均用时(s)	最快 Reduce 任务用时(s)	最慢 Reduce 任务用时(s)
Hadoop 默认策略	72.1	106.5	180.6	35.4	2.2	70.4	32.7	12.3	47.2
HDPA＋DTAA	22.3	81.3	104.7	8.7	1.6	21.3	30.8	12.0	41.1

表 4 Hadoop 默认策略和 HDPA＋DTAA 两种策略下 HDFS 数据吞吐量

Table 4 HDFS data throughput in the Hadoop default strategy and HDPA＋DTAA strategy

运行策略	数据写入吞吐量(MB/s)	数据写入平均IO 速率(MB/s)	数据写入速率标准差(MB/s)	数据读取吞吐量(MB/s)	数据读取平均IO 速率(MB/s)	数据读取速率标准差(MB/s)
Hadoop 默认策略	6.7	12.1	26.4	14.9	48.6	102.5
HDPA＋DTAA	7.0	12.5	18.0	15.9	50.4	98.7

10.5%，并且在每次迭代中的用时波动也更加稳定。

4.3 优化方案对 HDFS 数据吞吐量的影响

为验证本文 HDPA 方案对 Hadoop 集群自身的读写性能并未产生明显影响，本文通过向集群并发读写 200 个文件，每个文件设为 1 GB 大小，测试 HDPA 方案和默认数据放置策略下 Hadoop 集群的数据吞吐量。如表 4 所示，HDPA 数据放置算法取代了 HDFS 默认的随机放置，在数据写入的 IO 路径上会带来一定开销，主要原因在于节点分组和数据放置选取。由于节点分组时只考虑硬件性能参数和较长一段时间的平均负载，因此单次划分可作为系统较长一段时间的划分参考。此外，数据的放置位置选取主要包括节点组平均存储占用排序、机架级可靠性保障检测以及条带放置选取结束后的节点组平均存储占用更新等三部分，其计算开销相对较低，因此 HDPA 不会在文件写入时带来过长的额外延迟。并且，由于条带按节点性能放置，部分条带反而会更快完成写入或读取，因此 HDPA 不会对 HDFS 的总体读写性能造成明显消极影响。

5 结 论

本文提出了一种基于异构 Hadoop 集群的数据放置算法 HDPA 和任务分配策略 DTAA，分别从存储侧和计算侧两端解决在条带式纠删码存储模式下，上层 MapReduce 应用由于“一对多”数据访问模式和节点性能差异造成的任务运行用时增加的问题。实验结果表明，在 Wordcount、Sort、K-Means 等三类具有代表性的 MapReduce 应用中，通过对异构集群节点的数据访问性能进行评估和分组，以此作为纠删码条带放置位置的选取依据，能够有效提升 MapReduce 应用在 Map 阶段的数据访问性能；此外，根据集群节点各项资源负载情况对 Hadoop 的任务并发度进行动态调节，能够有效避免各节点任务间因资源竞争产生严重的长尾效应，最终验证了“HDPA＋DTAA”方案的可行性与有效性。目前，国内外针对 MapReduce 应用优化的主要场景大多基于多副本存储模式[6,17]，基于 Hadoop 条带式纠删码存储模式的相关工作相对较少，主要是一些问题的提出和在同构场景下的实验分析[18]，因此本文具有一定创新性。在未来工作中，我们将尝试采用不同评估标准对集群节点的性能进行更加精确的划分，并设计更加高效灵活的数据放置和任务分配策略来进一步提升本研究方案的优化效果。

参 考 文 献

[1] IDC. 数据新视界 [EB/OL]. (2017-03-07)[2021-10-09]. https://www.seagate.com/files/www-content/our-story/rethink-data/files/Rethink_Data_Report_2020_zh_CN.pdf.
IDC. New horizons of data [EB/OL]. (2017-03-07)[2021-10-09]. https://www.seagate.com/files/www-content/our-story/rethink-data/files/Rethink_Data_Report_2020_zh_CN.pdf.

[2] Ceph. Ceph documentation [EB/OL]. (2016-08-25)[2021-10-09]. https://docs.ceph.com/en/latest/rados/operations/erasure-code/.

[3] Apache Hadoop. HDFS erasure coding [EB/OL]. (2021-06-11)[2021-10-09]. https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HDFSErasureCoding.html.

[4] Plank JS. Erasure codes for storage system: a brief primer [J]. ;login:, 2013, 38(6): 44-50.

[5] Wang A. Introduction to HDFS erasure coding in Apache Hadoop [EB/OL]. (2015-09-23)[2021-10-09]. https://blog.cloudera.com/introduction-to-hdfs-erasure-coding-in-apache-hadoop/.

[6] Wang J, Shang PJ, Yin JL. Cloud Computing for Data-Intensive Applications [M/OL]. Springer, 2014: 153-179. https://link.springer.com/content/pdf/10.1007/978-1-4939-1905-5.pdf.

[7] Xiong RQ, Luo JZ, Dong F. Optimizing data placement in heterogeneous Hadoop clusters [J]. Cluster Computing, 2015, 18(4): 1465-1480.

[8] Shen ZR, Lee PP. Cross-rack-aware updates in erasure-coded data centers [C] // Proceedings of the 47th International Conference on Parallel Processing, 2018: 1-10.

[9] Li RH, Li XL, Lee PP, et al. Repair pipelining for erasure-coded storage [C] // Proceedings of the 2017 USENIX Annual Technical Conference, 2017: 567-579.

[10] Wang F, Tang Y, Xie Y, et al. Xorinc: optimizing data repair and update for erasure-coded systems with XOR-based in-network computation [C] // Proceedings of the 35th Symposium on Mass Storage Systems and Technologies, 2019: 244-256.

[11] Rashmi KV, Chowdhury M, Kosaian J, et al. EC-Cache: load-balanced, low-latency cluster caching with online erasure coding [C] // Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, 2016: 401-417.

[12] Yao X, Wang CL, Zhang MZ. EC-Shuffle: dynamic erasure coding optimization for efficient and reliable Shuffle in Spark [C] // Proceedings of the 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2019: 41-51.

[13] Tandon R, Lei Q, Dimakis AG, et al. Gradient coding: avoiding stragglers in distributed learning [C] // Proceedings of the 34th International Conference on Machine Learning, 2017: 3368-3376.

[14] Wang HZ, Guo S, Tang B, et al. Heterogeneity-aware gradient coding for straggler tolerance [C] // Proceedings of the 39th International Conference on Distributed Computing Systems, 2019: 555-564.

[15] Li S, Maddah-Ali MA, Avestimehr AS. Coded MapReduce [C] // Proceedings of the 53rd Annual Allerton Conference on Communication, Control, and Computing, 2015: 964-971.

[16] Ford D, Labelle F, Popovici FI, et al. Availability in globally distributed storage systems [C] // Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation, 2010: 61-74.

[17] Ahmad F, Chakradhar ST, Raghunathan A, et al. Shufflewatcher: Shuffle-aware scheduling in multi-tenant MapReduce clusters [C] // Proceedings of the 2014 USENIX Annual Technical Conference, 2014: 1-12.

[18] Darrous J, Ibrahim S, Perez C. Is it time to revisit erasure coding in data-intensive clusters? [C] // Proceedings of the 27th IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2019: 165-178.