计 算 机 研 究 与 发 展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202330541
61(3): 571−588, 2024

面向存算联调的跨云纠删码自适应数据访问方法

张凯鑫　王意洁　包　涵　阚浚晖
（并行与分布计算全国重点实验室（国防科技大学）　长沙　410073）
（国防科技大学计算机学院　长沙　410073）
（zhangkaixin19@nudt.edu.cn）

An Adaptive Erasure-Coded Data Access Method for Cross-Cloud Collaborative Scheduling of Storage and Computation

Zhang Kaixin, Wang Yijie, Bao Han, and Kan Junhui
（*National Key Laboratory of Parallel and Distributed Computing* (*National University of Defense Technology*), *Changsha* 410073）
（*College of Computer Science and Technology*, *National University of Defense Technology*, *Changsha* 410073）

Abstract　Nowadays, the increasing demand for cross-cloud collaborative scheduling of storage and computation puts high demands on cross-cloud data access speed. Therefore, cross-cloud data access methods based on data redundancy techniques (erasure coding and multiple-duplicate) with high cross-cloud data access speed are gaining attention. Among them, the cross-cloud data access method based on erasure coding has become a hot research topic because of its low storage overhead and high fault tolerance. In order to improve the data access speed by shortening the transmission time of coded blocks, existing cross-cloud data access methods based on erasure coding introduce caching techniques and optimize the coded data access scheme. However, due to the coarse granularity of cache management and the lack of coordinated optimization of cache management and coded data access scheme, the existing methods suffer from low cache hits, low cache hit efficiency, and high access volume of coded blocks with low transmission speed, which prolong the coded block transmission time. To this end, we first propose an IPFS-based cross-cloud storage system framework (IBCS) that can realize fine-grained cache management based on IPFS data slice management mechanism, and thus can improve cache hits. Then, we propose an adaptive erasure-coded data access method for cross-cloud collaborative scheduling of storage and computation (AECAM) that evaluates the transmission speed of each coded block during data access based on the distribution of coded blocks (including cached coded blocks) and data access nodes, and accordingly formulates a coded data access scheme that can avoid accessing low transmission speed coded blocks. In addition, AECAM identifies coded blocks that are easily selected in the coded data access scheme and have low transmission speed, and caches them near the data access nodes, thus improving both cache hits and hit efficiency. We build a cross-cloud storage system for collaborative scheduling of storage and computation (C2S2) based on IBCS and AECAM. Compared with existing erasure-coded storage systems that introduce caching, experiments in a cross-cloud environment show that C2S2 can improve data access speed by 75.22%−81.29%.

收稿日期：2023-06-21；**修回日期**：2023-12-04
基金项目：科技创新 2030——“新一代人工智能”重大项目 (2022ZD0115302)；国家自然科学基金项目 (61379052)；国家教育部科研创新基金项目 (2018A02002)；湖南省自然科学杰出青年基金项目 (14JJ1026)
This work was supported by the National Key Research and Development Program of China (2022ZD0115302), the National Natural Science Foundation of China (61379052), the Science Foundation of Ministry of Education of China (2018A02002), and the Natural Science Foundation for Distinguished Young Scholars of Hunan Province (14JJ1026).
通信作者：王意洁 (wangyijie@nudt.edu.cn)

Key words cross-cloud collaborative scheduling of storage and computation; erasure coding; data access technology; IPFS; cache

摘 要 日益旺盛的跨云存算联调需求对跨云数据访问速度提出较高要求. 因此，跨云数据访问速度较高的基于数据冗余技术（纠删码和多副本）的跨云数据访问方法逐渐受到关注. 其中，基于纠删码的跨云数据访问方法因其存储开销较低、容错性较高而成为当前研究热点. 为通过缩短编码块传输用时以提高数据访问速度，现有基于纠删码的跨云数据访问方法尝试引入缓存技术并优化编码数据访问方案. 然而，由于现有方法的缓存管理粒度较粗且未协同优化缓存管理与编码数据访问方案，导致其存在缓存命中量低、缓存命中增效低、低传输速度编码块访问量大等问题，使得其编码块传输用时仍较长. 为此，首先提出了一种基于星际文件系统（interplanetary file system, IPFS）的跨云存储系统框架（IPFS-based cross-cloud storage system framework, IBCS），可基于 IPFS 数据分片管理机制实现细粒度的缓存管理，从而可提高缓存命中量. 然后，提出一种面向存算联调的跨云纠删码自适应数据访问方法（adaptive erasure-coded data access method for cross-cloud collaborative scheduling of storage and computation, AECAM）. AECAM 以编码块（含缓存编码块）与数据访问节点的分布为依据评估数据访问过程中各编码块的传输速度，并据此制定可避免访问低传输速度编码块的编码数据访问方案. 此外，AECAM 可识别出其制定编码数据访问方案时易选中且实际传输速度较低的编码块，并将其缓存在数据访问节点附近，从而可同时提高缓存命中量和命中增效. 最后，基于 IBCS 和 AECAM 构建了面向跨云存算联调的存储系统（cross-cloud storage system for collaborative scheduling of storage and computation, C2S2）. 跨云环境下的实验表明，相较于现有引入缓存的基于纠删码的存储系统，C2S2 可以将数据访问速度提高 75.22%~81.29%.

关键词 跨云存算联调；纠删码；数据访问技术；星际文件系统；缓存

中图法分类号 TP391

随着数字经济生态和云计算技术的蓬勃发展，用户的云计算需求量迅速扩大且需求越发多样化. 由于单一云厂商服务能力有限，逐渐难以满足用户的全部需求. 跨云调度计算任务可有效缓解单一云厂商服务能力不足以满足用户需求的问题. 为此，研究者提出了云际计算技术，以云厂商之间的开放协作为基础，有效破解云厂商锁定问题，方便开发者通过软件定义方式定制云服务，为计算任务跨云调度创造了基础条件.

然而，由于跨云调度的计算任务通常需要跨云访问数据（即跨云存算联调），且在诸如机器学习、气象预报、基因测序、粒子物理等领域的计算任务的数据集往往高达数百 TB[1]，因而跨云调度后的计算任务的执行效率受到跨云数据访问速度的严重制约. 基于数据冗余（纠删码和多副本）技术的跨云数据访问方法是提高数据跨云访问速度的重要技术途径之一. 相较于多副本[2-5]，纠删码具有更高的数据容错性和更低的数据冗余度[6-7]. 因此，基于纠删码的跨云数据访问方法成为了当前的研究热点. 基于纠删码的跨云数据访问方法的基本思想为：首先，把原始数据对象分为 k 个数据块；然后，把 k 个数据块编码为 n 个编码块（这 n 个编码块中可能包含 k 个数据块），使用这 n 个编码块中的任意 $n-d+1$ 个编码块均可重构出 k 个数据块；接着，把 n 个编码块存储到多个云上；当某个数据访问节点需要访问数据时，选择若干云请求 $n-d+1$ 个编码块，用这些编码块重构出 k 个数据块，并将这些数据块重组为原始数据对象[8].

由于云间网络带宽通常远低于云内网络带宽，而基于纠删码的跨云数据访问方法需要跨多个云传输编码块. 因此，基于纠删码的跨云数据访问方法的编码块传输用时通常较长，使得其数据访问速度较慢.

基于纠删码的跨云数据访问方法主要可从 2 个方面来缩短编码块传输用时：

1）合理制定编码数据访问方案[9-13]. 由于纠删码可使用任意 $n-d+1$ 个编码块重构出 k 个原始数据块，因此基于纠删码的跨云数据访问方法可选择与数据访问节点间延迟较低（或带宽较高）的云上的编码块来重构数据，从而缩短编码块传输用时.

2）合理缓存编码块[14-20]. 基于纠删码的跨云数据访问方法可通过把常被访问的编码块缓存在相应的数据访问节点附近，以缩短整体的编码块传输用时.

在引入编码块缓存的情况下，基于纠删码的跨云数据访问方法的编码块传输用时显著受到缓存命中量（从数据访问节点对应的缓存中读取的数据总

量)、缓存命中增效(缓存命中时编码块传输用时的缩短量)、低传输速度编码块访问量的影响.

然而,现有基于纠删码的跨云数据访问方法存在2个方面的不足,使得其缓存命中量和缓存命中增效较低且低传输速度编码块访问量有待降低:

1)缓存管理粒度较粗.现有基于纠删码的跨云数据访问方法以编码块为单位管理缓存,缓存编码块只存在命中和未命中2种状态,无法利用部分命中的缓存编码块,因而其缓存命中量较低.

2)未协同优化缓存管理与编码数据访问方案.现有基于纠删码的跨云数据访问方法在选择加入缓存的编码块时,未综合考虑编码数据访问方案制定算法选择各编码块用于重构数据的概率以及各编码块的实际传输速度,不利于提高缓存命中量和缓存命中增效;或者在制定编码数据访问方案时,未充分考虑缓存对各编码块传输速度的影响,不利于准确评估各编码块传输速度,因而难以有效减少低传输速度编码块访问量.

现有基于纠删码的跨云数据访问方法存在的缓存命中量低、缓存命中增效较低、低传输速度编码块访问量大的问题,导致其编码块传输用时较长,进而使得其数据访问效率仍有待提升.

为此,本文首先提出了一种基于星际文件系统(interplanetary file system, IPFS)的跨云存储系统框架(IPFS-based cross-cloud storage system framework, IBCS).IBCS基于IPFS的数据分片管理机制实现细粒度缓存管理,可提高缓存命中量.此外,在引入缓存后,同一编码块的多个副本可能同时存储在多个云上,IBCS可充分利用这些副本提高编码块的传输并行度,从而减少低传输速度编码块.

然后,在框架IBCS下,本文提出了一种面向存算联调的跨云纠删码自适应数据访问方法(adaptive erasure-coded data access method for cross-cloud collaborative scheduling of storage and computation, AECAM),可通过协同优化缓存管理与编码数据访问方案提高缓存命中量和缓存命中增效.具体提出2个关键算法:

1)编码数据访问方案自适应制定算法(adaptive formulating algorithm of erasure-coded data reconstruction scheme, AFERS).AFERS首先根据IPFS集群的各个存储节点与数据访问节点间的传输延迟、各编码块在各存储节点里的存储状态(含正常存储、长期缓存、临时缓存等)、数据访问节点的类型(云内计算节点、云外计算节点),综合评估指定节点访问数据的过程中各编码块的传输速度.然后,根据各编码块传输速度的评估值,自适应制定可避免访问传输速度评估值低的编码块的编码数据访问方案.

2)基于数据访问过程感知的缓存管理算法(cache management algorithm based on data access process-aware, CADA).CADA根据数据访问过程中AFERS对各编码块传输速度的评估值,以及被传输的编码块的实际传输速度,决定各编码块的缓存优先级.优先在数据访问节点附近缓存易被AFERS选中(传输速度预估值高)且实际传输速度相对较低的编码块,因而能同时提高缓存命中量及缓存命中增效.

1 相关工作

1.1 纠删码数据访问方案

纠删码数据访问方案可由二元组⟨*type*,*blocks*⟩组成.其中*type*为数据访问方案的类别,包括直接数据访问方案和降级数据访问方案;*blocks*为访问数据时需获取的编码块的列表.

1)直接数据访问方案

如图1所示,直接数据访问方案是指数据访问节点直接获取被访问数据对象的k个数据块,然后将它们重组成被访问数据对象.直接访问方式无需进行解码运算且实现简单,因而大部分分布式存储系统(GFS(Google file system)[3], Lustre[4], Ceph[5]等)均默认采用这种数据访问方式.

对于直接数据访问方案,需获取的编码块固定为k个数据块.因此,直接数据访问方案只有1套.

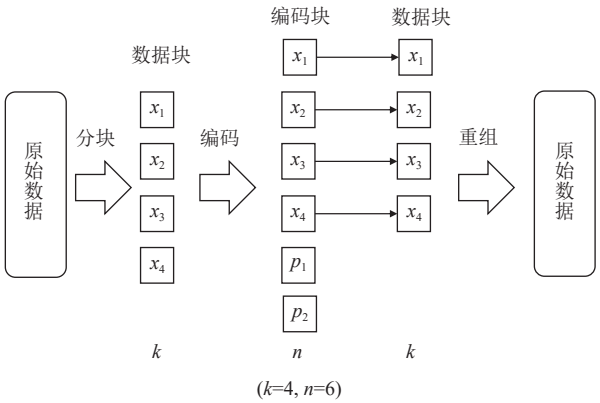


Fig. 1 Illustration of direct data access scheme

图1 直接数据访问方案示意图

2)降级数据访问方案

如图2所示,降级数据访问方案是指数据访问节点获取被访问数据对象的任意$n-d+1$个编码块,并将其解码为k个数据块,然后用解码出的数据块重组

出原始数据对象. 由于降级访问会带来额外的计算开销，因此在大部分存储系统中，仅当无法正常获取数据块时才会采用降级数据访问方案[6].

对于降级数据访问方案，需获取的编码块列表 *blocks* 的可能取值有 C_n^{n-d+1} 组. 因此，降级数据访问方案共有 C_n^{n-d+1} 套.

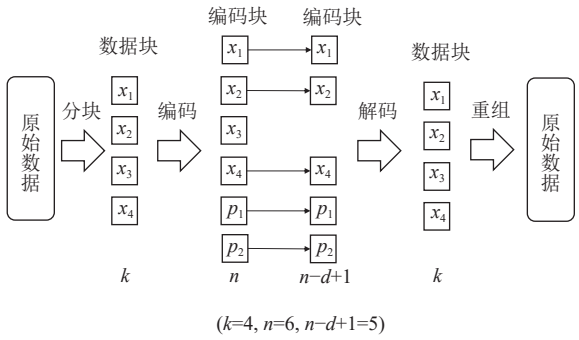


Fig. 2 Illustration of degraded data access scheme

图 2 降级数据访问方案示意图

1.2 广域纠删码数据访问技术

在广域部署的基于纠删码的存储系统(如基于纠删码的跨云存储系统)进行数据访问时，通常需要在低带宽链路上传输大量编码块，导致编码块传输速度成为了数据访问的瓶颈. 为此，研究者提出了一系列广域纠删码数据访问技术，主要分为基于数据访问方案优化的广域纠删码数据访问技术和基于缓存的广域纠删码数据访问技术.

1)基于数据访问方案优化的广域纠删码访问技术

基于数据访问方案优化的广域纠删码访问技术的基本思想是：传统分布式存储系统默认采用直接数据访问方案，因为其计算量较小. 然而，在广域环境中，不同域的节点间的数据传输速度远低于域内节点间的数据传输速度，且不同域节点的 I/O 性能差异较大. 如果数据访问节点与大量数据块存储节点属于不同的域或者大量数据块存储节点的 I/O 性能较低，则会导致必须访问数据块的直接数据访问方案的传输开销大于某些降级数据访问方案. 因此，基于数据访问方案优化的广域纠删码访问技术会综合评估各编码块存储节点和客户端间的带宽、各编码块存储节点的综合性能，求得效率较高的数据访问方案.

例如，Saeed[12] 以传输开销和服务延迟优化为目标，提出一种跨地域分布云环境下的纠删码访问方法 Sandooq. Sandooq 以编码块请求在各云数据中心的排队时延，以及以数据访问节点与存储各编码块的云数据中心的物理距离为依据建立读取优化函数，按照梯度下降的方法寻找最优数据访问方案. Zhang 等人[13] 提出基于节点性能感知和精确距离估算的纠删码降级访问方法 NADE. NADE 首先收集存储节点的容量、CPU 频率、吞吐量、响应时间、时延等指标的历史数据，并以此为依据构建各节点之间的欧氏距离. 然后，根据各节点间欧氏距离估算客户端获取每个编码块的开销，并以此为依据求出最优数据访问方案. 此外，Rashmi 等人[14] 提出了一种延迟绑定的纠删码访问方法 LBA. 采用 LBA 的数据访问节点会请求多个编码块，并选择先传输到数据访问节点的 $n-d+1$ 个编码块重构原始数据对象.

2)基于缓存的广域纠删码数据访问技术

基于缓存的广域纠删码数据访问技术的基本思想是：由于跨域传输编码块的开销较大，因而把部分访问频率高、传输速度低的编码块缓存在数据访问节点附近以提高后续数据访问速度.

例如，Zhang 等人[15] 提出了一种基于慢节点感知的编码数据缓存系统 POCache，用于提高广域纠删码的数据访问速度. POCache 将校验块缓存到数据访问节点附近，并在访问数据时优先请求数据块. 当数据块的传输速度过低时，POCache 将选择从缓存节点获取校验块，并使用缓存校验块与访问效率正常的数据块重构原始数据对象. 此外，POCache 可支持用户自行选择缓存引入算法和缓存驱逐算法. 然而，POCache 需要测出各数据块传输速度后才能确定用于重构原始数据对象的编码块，对其数据访问速度带来不利影响. Halalai 等人[16] 提出了一种基于缓存收益评估的编码数据缓存系统 Agar，将编码块的全局访问次数与各数据访问节点获取该编码块的开销的乘积作为各数据访问节点缓存该编码块的预期收益，并将预期收益较高的编码块缓存到对应的数据访问节点.

整体而言，现有基于数据访问方案优化的广域纠删码访问技术和基于缓存的广域纠删码数据访问技术均可提高广域环境(含跨云环境)下的纠删码数据访问效率，但存在 2 点不足：首先，现有工作中的缓存管理优化与编码数据访问方案优化结合不紧密，未同时做到根据缓存编码块分布情况决定数据访问方案、根据数据访问方案制定策略决定编码块缓存优先级，导致其缓存命中量和缓存命中增效仍有待提高. 此外，现有工作为优化缓存管理与编码数据访问方案，通常需要耗费较多额外的计算、网络、存储资源以采集、维护存储节点状态信息及各类节点间的网络信息.

1.3 星际文件系统

星际文件系统是一个去中心化的分布式文件系统[21-23]，基于分布式哈希表（distributed Hash table, DHT）、BitTorrent 点对点文件共享协议、Git 分布式版本控制、Merkle DAG 等已有技术，实现了数据的分片存储、内容寻址及点对点（peer-to-peer, P2P）高速传输.

1）数据分片存储

IPFS 将用户数据分为若干个大小不超过 256 KB 的数据分片，并使用 Merkle DAG 来组织各个存储节点上的数据分片. 如图 3 所示，Merkle DAG 的末端节点的内容为各个数据分片，其他节点的内容为其所有子节点的内容组合的哈希值.

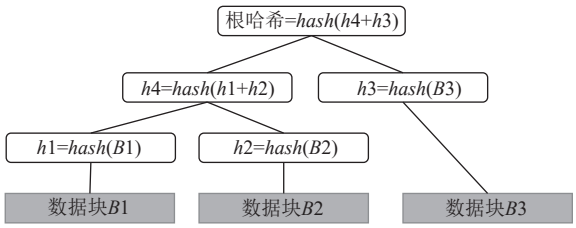


Fig. 3 Illustration of Merkle DAG

图 3 Merkle DAG 示意图

此外，Merkle DAG 的每个顶端节点对应一个数据对象，该数据对象的内容为其子孙节点中末端节点内容（数据分片）的组合.

分片存储数据并使用 Merkle DAG 来维护数据对象和数据分片间的关系的主要优点是：在同一 IPFS 存储节点上，多份用户数据对象均含有的数据分片只需要被实际存储 1 份，因而可显著降低存储开销.

2）数据内容寻址

在 IPFS 中，每份用户数据对象对应于一个 Merkle DAG 的根哈希，且每个数据分片的哈希也由 Merkle DAG 维护. 此外，IPFS 采用了分布式哈希表，支持任意 IPFS 节点通过哈希值定位到维护了指定用户数据对象的 Merkle DAG 的节点，以及存储了指定数据分片的节点.

因此，在 IPFS 中，访问数据的基本过程有 4 个：

①数据访问节点向 IPFS 集群发送需访问数据对象的根哈希，IPFS 集群通过查询分布式哈希表定位到维护了被请求数据对象的 Merkle DAG 的节点，并将相应的 Merkle DAG 传输到数据访问节点.

②数据访问节点根据接收到的 Merkle DAG 解析出被访问数据的各数据分片的哈希.

③数据访问节点向 IPFS 集群发送需访问数据对象的各数据分片的哈希，IPFS 集群通过查询分布式哈希表定位到存储了这些数据分片的节点，并将这些数据分片传输到数据访问节点.

④数据访问节点根据接收到的 Merkle DAG 和数据分片，构造出被访问的数据.

3）P2P 高速传输

在 IPFS 中，数据访问节点会同时请求多个数据分片且可能有多个存储节点上拥有部分或全部被请求的数据分片. 因此，IPFS 采用以 BitSwap[24] 数据交换协议为核心的 P2P 传输技术，根据节点信用分、性能、负载、网络状况等因素选择各数据分片的提供节点，并从多个提供节点并行地向数据访问节点发送其请求数据的各数据分片，从而显著提高数据传输速度.

此外，IPFS 具有缓存机制，各节点会将请求过的数据分片缓存在本地一段时间. 期间，如果其他节点请求这些数据分片，该节点仍能将缓存的数据分片提供出去，以提高其他节点的数据访问速度.

最后，在 IPFS 的 BitSwap 协议中，引入了信用机制，根据各节点对外提供数据的情况为其评定信用分. 对外提供数据越频繁的节点的信用分越高，而信用分越高的节点需要数据时会有更多的节点向其提供数据. 在各节点属于不同主体的场景下（如跨云存储场景），这种信用机制可以促进各主体积极对外提供数据，进而提高 IPFS 系统的整体数据访问速度.

2 基于 IPFS 的跨云存储系统框架 (IBCS)

本节提出了框架 IBCS，可基于 IPFS 的分片存储、内容寻址及 P2P 传输等技术，实现细粒度缓存管理和高速编码块传输.

2.1 体系结构

如图 4 所示，框架 IBCS 由读写控制层、存储传输层和数据服务层组成.

1）读写控制层包含协调节点和多个云代理节点，主要负责响应客户端请求，完成用户数据的写入和访问，具体工作由协调节点上的协调组件、云代理节点上的代理组件和元数据管理组件完成：

①协调组件负责解析用户命令，生成写入、访问过程的控制命令，并发往代理组件；

②代理组件负责执行协调组件发送的命令，包括调用 IPFS 组件向 IPFS 集群写入编码块或从 IPFS 集群读取编码块、对编码块进行解码操作、将解码编码块得到的数据上传到各云存储服务或传输到云外计算节点；

③元数据管理组件负责维护用户、数据对象、

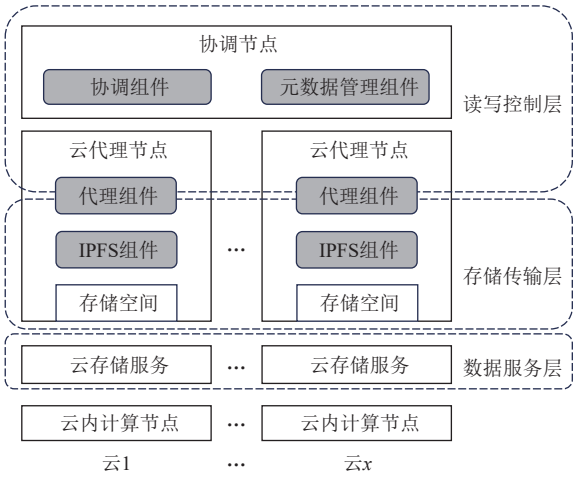


Fig. 4 Illustration of IBCS

图 4 IBCS 示意图

编码块、云代理、云存储服务等元数据.

2)存储传输层由各云代理节点构成，主要负责在各云代理节点上的存储空间中存储管理用户数据对象的编码块，并在各云代理节点之间传输这些编码块，具体工作由各云代理节点上的代理组件和 IPFS 组件完成：

①代理组件负责调用 IPFS 组件，完成本节点上持久化存储编码块及缓存编码块的读出、写入和状态变更；

②各 IPFS 组件及各云代理的存储空间构成了 IPFS 集群，负责持久化存储或缓存编码块、执行代理组件命令、从 IPFS 集群中读取编码块、向 IPFS 集群写入编码块、利用 P2P 技术在不同云的代理节点间高效传输编码块.

3)数据服务层为各个云的云存储服务(可被各云代理节点访问)，将用户数据对象存储到云存储服务后，同一云上的计算节点才能正常访问该数据对象.

2.2 工作原理

框架 IBCS 的核心工作原理包括数据写入原理、数据访问原理、存储状态管理原理和缓存实现原理.

1)数据写入

如图 5 所示，客户端首先向协调节点发起数据写入请求后，协调节点为用户数据分配云代理节点；然后，客户端会将原用户数据分割为 k 个数据块，并对这 k 个数据块进行编码计算后得到 n 个编码块；最后，客户端将 n 个编码块传输至协调节点分配的云代理节点，各云代理节点的代理组件会接收这些编码块，并调用 IPFS 组件将编码块按照 IPFS 的数据组织格式(分片后以 Merkle DAG 组织)存储到本节点的存储空间.

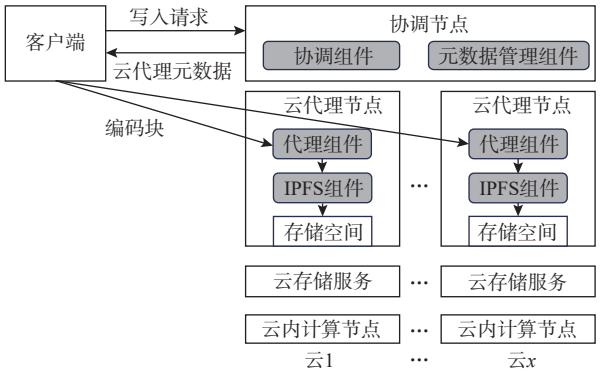


Fig. 5 IBCS data write scheme

图 5 IBCS 数据写入方案

2)数据访问

在框架 IBCS 中，有 2 种节点可能需要访问数据：云内计算节点和云外计算节点.

如图 6 所示，云内计算节点可高效访问其所在云的云存储服务中的数据. 云内计算节点需要访问数据的过程为：首先，客户端向协调节点发送数据访问请求；然后，协调节点根据客户端请求，拟制数据访问方案并生成控制命令发送云内计算节点对应的云代理节点；随后，接收到命令的云代理节点从 IPFS 集群读出待访问数据对象的编码块，解码出原始数据，并上传到对应的云存储服务.

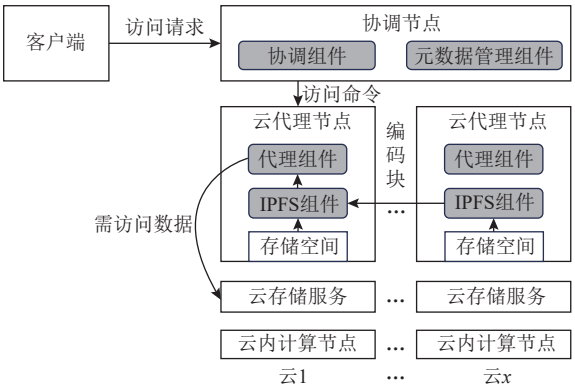


Fig. 6 IBCS data access scheme(in-cloud nodes)

图 6 IBCS 数据访问方案(云内计算节点)

云外计算节点无权限访问或不可高效访问云存储服务. 如图 7 所示，云外计算节点访问数据的过程为：首先，云外计算节点(运行着客户端)向协调节点发送请求，获取其所需数据的编码块的云代理节点信息；然后，云外计算节点拟制数据访问方案并向各代理节点请求编码块；随后，各代理节点将编码块从 IPFS 集群中读出后，发送给云外计算节点；最后，云外计算节点将收到的编码块解码为所需数据.

3)存储状态管理

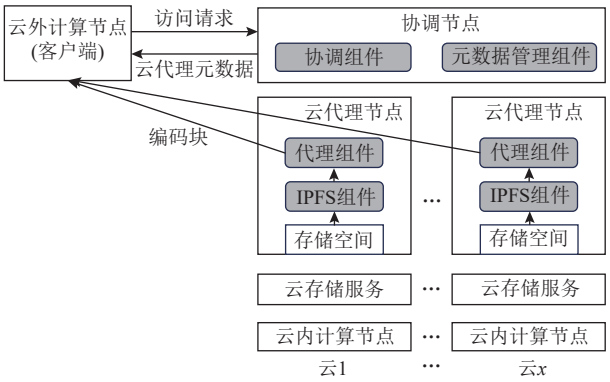


Fig. 7 IBCS data access scheme(nodes outside the cloud)

图 7 IBCS 数据访问方案（云外计算节点）

在框架 IBCS 中，各云代理节点上由 IPFS 组件维护的存储空间既是编码块的持久化存储空间，又是编码块的缓存空间.

IPFS 组件维护的存储空间中的编码块有长期存储(pinned)和临时存储(tmp)两种状态. 处于 tmp 状态的编码块会被 IPFS 组件的垃圾回收程序定时清理，处于 pinned 状态的编码块则不被垃圾回收程序清理.

代理组件向 IPFS 集群写入编码块时，为了最大化数据容错度，通常避免将一个条带上的编码块及其副本分配至同一个云代理节点上，以确保单一节点失效时同一数据对象的编码块中最多一个编码块受到影响. 图 8 说明了 IBCS 编码块存储状态的变化，写入的编码块会默认以 pinned 的状态存储在代理组件所在云代理节点的存储空间中，直到维护该存储空间的 IPFS 组件接收到改变该编码块状态的 unpin 命令. 代理节点从 IPFS 集群读取的编码块，会以 tmp 的状态存储在代理组件所在代理节点的存储空间中，直到维护该存储空间的 IPFS 组件启动垃圾回收策略. 此外，如果某云代理节点上的 IPFS 组件接收到请求某个编码块的 pin 命令，IPFS 集群使用 P2P 传输技术将该编码块传输到该云代理节点上的存储空间中并设置为 pinned 状态.

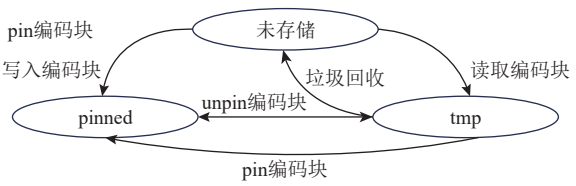


Fig. 8 Principle of coding blocks storage state management in IBCS

图 8 IBCS 编码块存储状态管理原理

因此，通过对各 IPFS 组件发送 pin 命令和 unpin 命令，可以实现编码块存储状态的灵活管理.

4)缓存实现

在框架 IBCS 中，持久化存储的编码块和缓存编码块均存储在各云代理节点的存储空间，如表 1 所示. 其中，持久化存储的编码块一直处于 pinned 状态；缓存编码块可能处于 pinned 状态，也可能处于 tmp 状态，二者间的转换由运行的代理组件控制.

Table 1 Classification of Coding Blocks in IBCS

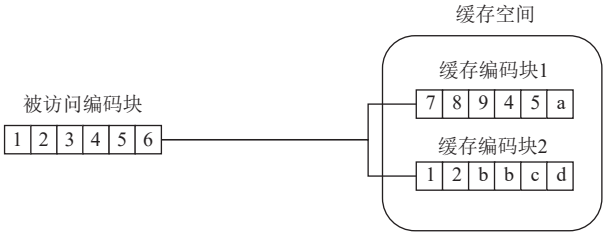
表 1 IBCS 中编码块的分类

编码块类型	存储状态	是否会被 IPFS 定时回收	是否可被访问
持久化存储的编码块	pinned	否	是
缓存编码块	pinned	否	是
	tmp	是	是

2.3 框架分析

框架 IBCS 的主要特点有 4 个方面：

1)利用 IPFS 的数据分片管理机制实现了缓存数据的细粒度管理，可从 2 个方面提高缓存命中量. 一方面，利用 IPFS 的数据分片管理机制避免在同一云代理节点上存储重复的数据分片，因而能降低存储开销，进而增加了各云代理节点可缓存的编码块数；另一方面，数据分片管理机制使得未命中缓存编码块中的命中数据分片可以被利用起来，如图 9 所示.



注：缓存管理若以编码块为粒度，则缓存命中量为 0；若以数据分片为粒度，则缓存命中量为 4 个数据分片.

Fig. 9 Illustration of fine-grained cache management in IBCS

图 9 IBCS 的细粒度缓存管理示意图

2)基于 IPFS 的 P2P 传输技术，可充分利用系统中持久化数据和缓存数据来提高编码块的传输并行度，通过减少低传输速度编码块从而有效提高编码块传输速度. 此外，IPFS 的 P2P 传输技术引入了信用机制，可以促进各云主体提升自身对外提供数据的能力，有助于不断提高跨云存储系统的编码块传输速度.

3)得益于 IPFS 的内容寻址机制，读写控制层只需要管理各个编码块的根哈希即可，无需管理各个数据分片信息，因而数据分片管理机制、P2P 传输过程对读写控制层而言是透明的，这大大降低了元数据管理开销和系统实现难度.

4)针对云内计算节点和云外计算节点,分别设计了不同的数据访问流程,适用范围较广.

3 面向存算联调的跨云纠删码自适应数据访问方法 (AECAM)

第 2 节提出了一种基于 IPFS 的跨云存储系统框架 (IBCS). 基于 IBCS, 本节提出一种面向存算联调的跨云纠删码自适应数据访问方法 (AECAM), 可通过协同优化缓存管理与数据访问方案减少数据访问过程中的编码块传输用时.

具体而言, 首先提出一种编码数据访问方案自适应制定算法 (AFERS), 可根据 IPFS 集群的各个存储节点与数据访问节点的传输延迟、各编码块在各存储节点里的存储情况、数据访问节点的类型,综合评估数据访问过程中各编码块的传输速度,并以此为依据自适应制定可避免访问传输速度评估值低的编码块的编码数据访问方案,从而缩短数据访问过程中的编码块传输用时.

然后,提出了一种基于数据访问过程感知的缓存管理算法 (CADA), 可优先在数据访问节点缓存易被编码数据访问方案自适应制定算法 AFERS 选中(传输速度预估值高)且实际传输速度相对较低的编码块,以同时提高缓存命中量及缓存命中增效,进而缩短数据访问过程中的编码块传输用时.

3.1 编码数据访问方案自适应制定算法 (AFERS)

1)核心思想

算法 AFERS 首先针对云内计算节点访问数据和云外计算节点访问数据 2 种场景,分别建模评估编码块传输速度. 然后, AFERS 可根据各编码块传输速度的评估值制定编码数据访问方案.

①编码块传输速度评估

算法 AFERS 将数据访问节点区分为云内计算节点和云外计算节点 2 种,并根据框架 IBCS 中将数据传输到 2 类数据访问节点的具体过程,使用不同的模型来评估这 2 类数据访问节点访问数据时的编码块传输速度.

(i) 云内计算节点访问数据时的编码块传输速度评估

在框架 IBCS 中,云内计算节点需要访问数据时,编码块通过 P2P 传输技术被传输到数据访问节点所对应的云代理节点,然后该云代理节点会用接收到的编码块解码出原始数据对象,并将原始数据对象上传到需访问数据的云内计算节点(数据访问节点)可访问的云存储服务.

上述过程中,影响编码块传输速度的主要因素包括 P2P 网络中存储了各编码块的云代理节点数,以及这些云代理节点与数据访问节点对应云代理节点间的实时带宽. P2P 网络中实际存储待传输编码块的云代理节点数越多,编码块传输速度越快;这些云代理节点与数据访问节点对应云代理节点间的实时带宽越高,编码块传输速度越快.

因此,若令 b 为待传输编码块, c 为数据访问节点对应的云代理节点(编码块的传输目的地节点), $E_{b,c}$ 为将编码块 b 传输到目的地节点 c 的速度, q_b 为存储编码块 b 的云代理节点数, $x_{b,i}$ $(i\in[1,q_b])$ 为第 i 个存储了编码块 b 的云代理节点的序号, $B_{c,x_{b,i}}$ 为序号为 $x_{b,i}$ 的云代理节点与传输目的地节点 c 间的实时带宽,则有

$$E_{b,c}\propto\sum_{i\in[1,q_b]}B_{c,x_{b,i}}. \tag{1}$$

此外,由于框架 IBCS 中各云代理节点上的编码块的状态包括 2 类 pinned 和 tmp, 且处于 tmp 状态的编码块有一定概率已经被回收(状态为 pinned 的编码块没被回收的概率为 1),所以,若令 Q_b 为可能存储编码块 b 的云代理节点数(存储了处于 pinned 状态或 tmp 状态的编码块 b 的云代理节点数), $y_{b,i}$ $(i\in[1,Q_b])$ 为第 i 个可能存储编码块 b 的云代理节点 $Node_{y_{b,i}}$ 的序号, $B_{c,y_{b,i}}$ 为 $Node_{y_{b,i}}$ 与传输目的地节点 c 间的实时带宽, $p_{y_{b,i}}$ 为 $Node_{y_{b,i}}$ 实际存储了编码块 b 的概率,则有

$$E_{b,c}\propto\sum_{i\in[1,Q_b]}p_{y_{b,i}}B_{c,y_{b,i}}. \tag{2}$$

由于 IPFS 会定时触发垃圾回收任务删除处于 tmp 状态的编码块,因此,若令 IPFS 触发垃圾回收任务的周期为 T, 节点 $Node_{y_{b,i}}$ 上的编码块 b 处于 tmp 状态的时长为 $t_{b,y_{b,i}}$,则该节点上编码块 b 未被回收(实际存在)的概率为

$$p_{b,y_{b,i}}=\begin{cases}0, & t\geqslant T,\\ 1-t_{b,y_{b,i}}/T, & t<T.\end{cases} \tag{3}$$

因此, $Node_{y_{b,i}}$ 中编码块 b 未被回收的概率为

$$p_{b,y_{b,i}}=\begin{cases}1, & b\text{处于pinned状态},\\ 0, & b\text{处于tmp状态且}t_{b,y_{b,i}}\geqslant T,\\ 1-t_{b,y_{b,i}}/T, & b\text{处于tmp状态且}t_{b,y_{b,i}}<T.\end{cases} \tag{4}$$

最后,由于测量节点间实时带宽的开销较大,算法 AFERS 使用节点间延迟来估算带宽,具体估算方法为:

$$B_{c,y_{b,i}} \propto 1/\ln(D_{c,y_{b,i}})+1, \tag{5}$$

其中$D_{c,y_{b,i}}$为$Node_{y_{b,i}}$与传输目的地节点c间的传输延迟. 通常情况下, 2个节点间的传输延迟越大意味着在2个节点间传输数据时的数据转发的次数越多, 而数据转发次数的增加将导致数据传输途经低带宽链路的概率增加、丢包率提升, 进而导致传输带宽降低, 因此延迟通常与带宽呈负相关. 此外, 实际测试中发现, 传输延迟少于1ms时(如2个节点处于同一云中)节点间带宽将大幅度提高(同云内节点间带宽远高于不同云节点间带宽). 式(5)符合上述2项规律. 综上, 可得

$$E_{b,c} \propto \sum_{i\in[1,Q_b]} \frac{p_{b,y_{b,i}}}{\ln(D_{c,y_{b,i}})+1}. \tag{6}$$

因此, 算法AFERS在访问数据前, 用式(7)为各编码块评分, 当传输目的地云代理节点为c时, 编码块b的传输速度评分为$S_{b,c}$:

$$S_{b,c} = \sum_{i\in[1,Q_b]} \frac{p_{b,y_{b,i}}}{\ln(D_{c,y_{b,i}})+1}, \tag{7}$$

通常而言, 评分越高, 编码块相对传输速度越高.

(ii)云外计算节点访问数据时的编码块传输速度评估

在框架IBCS中, 云外计算节点需要访问数据时, 编码块通过TCP协议直接从特定云代理节点传输到需要访问数据的云外计算节点(数据访问节点), 并由数据访问节点将编码块解码为原始数据对象.

上述过程中, 影响编码块的传输速度的主要因素为存储了编码块的云代理节点与需访问数据的计算节点间的实时带宽的最大值. 因此, 若令o为数据访问节点(编码块传输目的地节点), $E_{b,o}$为将编码块b传输到计算节点o的速度, W_b为以pinned状态存储编码块b的云代理节点数, $z_{b,i}$ ($i\in[1,w_b]$)为第i个存储编码块b的云代理节点$Node_{z_{b,i}}$的序号, $B_{o,z_{b,i}}$为$Node_{z_{b,i}}$与数据访问节点o间的实时带宽, 则有

$$E_{b,o} \propto \max_{i\in[1,W_b]} B_{o,z_{b,i}}. \tag{8}$$

此外, 算法AFERS使用节点间延迟来估算带宽, 具体估算方法为:

$$B_{o,z_{b,i}} \propto \frac{1}{\ln(D_{o,z_{b,i}})+1}, \tag{9}$$

其中$D_{o,z_{b,i}}$为$Node_{z_{b,i}}$与计算节点o间的传输延迟. 综上可得

$$E_{b,o} \propto \max_{i\in[1,W_b]} \frac{1}{\ln(D_{o,z_{b,i}})+1}. \tag{10}$$

因此, 算法AFERS在访问数据前, 用式(11)为各编码块评分, 当传输目的地计算节点为o时, 编码块b的传输速度评分为$S_{b,o}$:

$$S_{b,o} = \max_{i\in[1,W_b]} \frac{1}{\ln(D_{o,z_{b,i}})+1}. \tag{11}$$

通常而言, 评分越高, 编码块相对传输速度越高.

②编码数据访问方案制定

在跨云环境下, 数据访问过程中最耗时的步骤通常为传输编码块. 因此, 编码数据访问方案制定的主要思想是根据编码块的传输速度评分, 为不同编码数据访问方案⟨*type*,*blocks*⟩的编码块传输速度评分, 其中*type*为编码数据访问方案类型, *blocks*为访问数据时需获取的编码块的列表, 并以⟨*type*,*blocks*⟩为主要依据确定编码数据访问方案以及编码访问方案中各编码块的提供节点.

(i) 编码数据访问方案的编码块传输速度评分

通常而言, 各节点的数据接收能力远高于跨云节点间的数据传输能力, 且同一节点可并行接收不同节点发来的编码块. 因此, 编码数据访问方案⟨*type*,*blocks*⟩的编码块传输速度为*blocks*中各编码块的传输速度的最小值.

由于直接数据访问方案需要传输k个数据块, 因此该类方案编码块传输速度评分E_{dir}为k个数据块的传输速度评分的最小值; 由于降级数据访问方案需要传输任意$n-d+1$个编码块, 因此该类方案的编码块传输速度评分E_{un}为传输速度评分第$n-d+1$高的编码块的传输速度评分.

(ii) 编码数据访问方案选择

算法AFERS结合各类数据访问方案的编码块传输速度评分及计算开销确定数据访问方案类型*type*. 由于直接数据访问方案的计算开销较小, 所以将其编码块传输速度评分E_{dir}乘上一个权值s后与降级数据访问方案的编码块传输速度评分E_{un}比较. 若$sE_{\mathrm{dir}}>E_{\mathrm{un}}$, 则采用直接数据访问方案, 反之采用降级数据访问方案. 假设D_{dir}为直接访问方案中k个数据块的传输速度评分的最小值所对应的代理节点与云外计算节点间的延迟, D_{un}为降级访问方案中传输速度评分第$n-d+1$高的编码块所对应的代理节点与云外计算节点间的延迟, 将式(10)带入选择直接数据访问的条件$sE_{\mathrm{dir}}>E_{\mathrm{un}}$, 可得$D_{\mathrm{dir}} < \mathrm{e}^{s-1}D_{\mathrm{un}}^{s}$. 换言之, 当且仅当$D_{\mathrm{dir}} < \mathrm{e}^{s-1}D_{\mathrm{un}}^{s}$时, 选择直接数据访问方案, 反之选择降级数据访问方案. 因此, 当$s\leqslant 1$时, 满足$D_{\mathrm{dir}}\geqslant D_{\mathrm{un}}$即选择降级数据访问方案, 此时忽略了降级数据访问方案带来的额外计算开销, 因而$s\leqslant 1$不合理; 当$s\geqslant 2$时, 至少满足$D_{\mathrm{dir}} \geqslant \mathrm{e}D_{\mathrm{un}}^{2}$才选择降级数据访问方案,

由于满足不等式$D_{dir} \geqslant eD_{un}^2$的$D_{un}$通常远小于$D_{dir}$, 这将造成数据传输开销远低于直接访问方案的降级数据访问方案被放弃, 因而$s \geqslant 2$不合理. 因此s的合理取值范围为$1<s<2$, 本文实验中取$s=1.5$.

若采用直接数据访问方案, 需要传输的编码块*blocks*为k个数据块. 若采用降级数据访问方案, 为了尽可能提高整体编码块传输速度, 需要传输的编码块*blocks*为传输速度评分最高的$n-d+1$个编码块.

(iii) 提供编码块的云代理节点选择

当数据访问节点为云内计算节点时, 该计算节点对应的云代理节点只需要提供各需传输编码块的根哈希, 相应的编码块即可通过P2P传输技术传输到该云代理节点上, 因而此时无需选择提供各编码块的云代理节点.

当数据访问节点为云外计算节点时, 则需要从存储了需传输编码块的各云代理节点中, 选择与计算节点间的网络延迟最小的云代理节点作为提供该编码块的节点.

2)算法描述

算法AFERS的具体工作流程如算法1所示.

算法 1. 算法AFERS.

输入: 数据访问节点*accessNode*, 访问数据ID *dataID*;

输出: 编码数据访问方案⟨*type*,*blocks*⟩, 提供各编码块的云代理节点列表*nodes*.

① if *accessNode.type*==云内计算节点
② D←从元数据管理组件获得*accessNode*对应云代理节点与存储被访问数据对应编码块的云代理节点的延迟(*accessNode*, *dataID*);
③ *blockInfo*←从元数据管理组件获得被访问数据对应编码块的信息(*dataID*);
④ *blockE*←计算编码块传输速度评估值(D, *blockInfo*);
⑤ *type*←选择编码数据访问方案的类型(*blockE*, *blockInfo*);
⑥ *blocks*←选择编码块(*type*, *blockE*);
⑦ *nodes*←null;
⑧ else
⑨ *blockInfo*←从协调节点获取被访问数据对应编码块的信息(*dataID*);
⑩ if数据访问节点未初始化
⑪ D←测试*accessNode*对应云代理节点与存储了被访问数据对应编码块的云代理节点的延迟(*blockInfo*);
⑫ else
⑬ D←从本地内容获取*accessNode*对应云代理节点与存储了被访问数据对应编码块的云代理节点的延迟(*blockInfo*);
⑭ end if
⑮ end if
⑯ *blockE*←计算编码块传输速度评估值(D, *blockInfo*);
⑰ *type*←选择编码数据访问方案的类型(*blockE*, *blockInfo*);
⑱ *blocks*←选择编码块(*type*, *blockE*);
⑲ *nodes*←选择提供节点(*blocks*, D);
⑳ return⟨*type*,*blocks*⟩, *nodes*.

若数据访问节点为云内计算节点(算法1的行①), 则算法AFERS运行于协调节点上的协调组件. 首先, AFERS从协调节点上的元数据管理组件获取数据访问节点对应云代理节点与存储被访问数据的编码块的云代理节点间的延迟(各云代理节点定时检测其与其他云代理节点间的延迟, 并将检测结果发往协调节点存储), 以及被访问数据的编码块的信息, 具体包括: 存储各编码块的节点、编码块在各节点上的存储状态tmp或pinned、处于tmp状态的编码块的存入时间等(算法1的行②③). 然后, AFERS根据从元数据管理组件获取的信息, 计算各编码块的传输速度评分(如式(7)), 并根据评分选择编码数据访问方案的类型和用于重构数据的编码块(算法1的行④~⑥).

若数据访问节点为云外计算节点(算法1的行⑧), 则算法AFERS运行于数据访问节点. 首先, AFERS从协调节点的元数据管理组件获取被访问数据对应编码块的信息, 主要包括: 存储了被访问数据对应pinned状态编码块的节点ID(算法1的行⑨). 然后, 如果数据访问节点未初始化, 则数据访问节点先测试其与被访问数据对应pinned状态编码块所在云代理节点间的延迟, 并存于本地内存; 如果数据访问节点已初始化, 则直接从本地内存获得其与各pinned状态编码块所在节点间的延迟(算法1的行⑩~⑬). 接着, AFERS计算各编码块的传输速度评分(如式(11)), 并根据评分选择编码数据访问方案的类型*type*和用于重构数据的编码块*blocks*(算法1的行⑯~⑱). 最后, AFERS依据数据访问节点与被访问数据对应pinned状态编码块所在云代理节点间的延迟, 选择提供*blocks*中编码块的云代理节点(算法1

的行⑲).

3)算法分析

①算法 AFERS 针对 2 种不同类型数据访问节点访问数据的过程,分别设计了编码块传输速度评估模型,且评估各编码块传输速度时考虑了各缓存编码块的分布情况,能较为准确地评估框架 IBCS 下不同类型数据访问节点访问数据时各编码块的速度.同时,AFERS 能以编码块传输速度评估值为依据制定编码块传输速度较高的编码数据访问方案,从而能提高不同访问节点访问数据的速度.

②算法 AFERS 只需要测试各云代理节点和云外计算节点间的延迟,相较于现有的需要测试节点间实时带宽、各节点实时性能的算法,AFERS 对各云代理节点的影响较低.

3.2 基于数据访问过程感知的缓存管理算法 (CADA)

1)主要思想

算法 CADA 首先以数据对象为单位进行缓存管理,当决定缓存或驱逐某个数据对象后,再对该数据对象的各编码块进行差异化的操作,各缓存编码块被切分为若干数据分片进行管理,可节省缓存空间.

①数据对象级缓存管理

当某个云代理节点对应的云内计算节点访问数据时,算法 CADA 会将该数据对象的部分编码块加入缓存,加入缓存的编码块的初始存储状态均为 pinned.

此外,CADA 可适配任意用户自定义的数据对象级缓存驱逐算法.当触发缓存驱逐操作后,CADA 将使用用户自定义的缓存驱逐算法初步决定驱逐哪些数据对象的编码块.缓存驱逐操作一方面会定时触发,另一方面会在剩余缓存空间不足以缓存待缓存编码块时触发.

②编码块级缓存管理

缓存编码块在云代理节点的存储空间内有 pinned 和 tmp 这 2 种状态.编码块状态为 pinned 时,存在缓存优先级(取值包括 0 或 1);编码块为 tmp 状态时,没有缓存优先级,且会被 IPFS 组件的垃圾回收程序定时清理.tmp 状态的编码块被回收之前仍保留在云代理节点上,可以作为缓存读取.

当某个云代理节点对应的云内计算节点访问数据时,CADA 会选择该数据对象的部分编码块加入缓存,并为这些编码块设置缓存优先级.缓存驱逐时,首先由用户自定义的数据对象级缓存驱逐算法决定待驱逐的数据对象,CADA 会根据缓存优先级更改 pinned 缓存编码块的状态.其中缓存优先级为 1 的编码块将被设为 0,仍保留 pinned 状态;缓存优先级为 0 的编码块将被设为 tmp 状态,如图 10 所示.

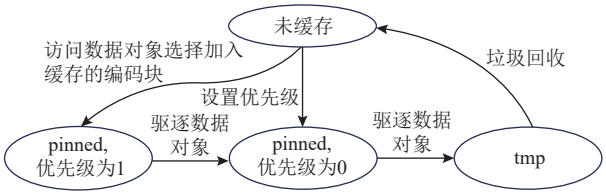


Fig. 10 Principle of coding blocks cache state management in CADA

图 10 CADA 编码块缓存状态管理原理

设置编码块缓存优先级的主要策略为:优先缓存易被算法 AFERS 选中的编码块(本次访问数据时已被 AFERS 选中的编码块)以提高缓存的命中量;优先缓存实际传输速度远低于所有被访问编码块实际传输速度平均值的编码块,使得缓存命中时可通过大幅度提高各编码块传输速度的下限而大幅度缩短编码块传输用时(即缓存命中增效较大).

因此,具体的缓存优先级设置过程为:首先,获取被 AFERS 选中的各编码块的实际传输用时,并计算出各编码块的实际传输用时的平均值$\bar{t}$和标准差σ.然后,得到实际传输用时大于$\bar{t}+j\sigma$的编码块(j 为经验常量,即使用 j-sigma 原则进行缓存决策),并将这些编码块加入缓存,其中,传输用时最长的编码块的缓存优先级将被设为 1,其他编码块的优先级将被设为 0.

③数据分片级缓存管理

加入缓存的编码块被划分为若干个大小不超过 256 KB 的数据分片并以 Merkle DAG 组织存储在云代理节点上,由 IPFS 组件维护.缓存编码块(以哈希值标识)对应 Merkle DAG 的一个根节点,内容为其叶节点所对应数据分片的组合.

算法 CADA 通过 IPFS 获取待缓存编码块哈希在 Merkle DAG 上末端节点的各数据分片哈希值,与本地已有分片的哈希进行比对,以判断数据分片的命中情况.利用 IPFS 的数据分片管理机制,在缓存空间中内容完全相同的数据分片仅存储 1 份,该数据分片在 Merkle DAG 中指向多个包含该数据分片的顶端节点,对应不同的缓存编码块.数据分片级缓存管理可节省缓存空间,增加云代理节点上可缓存的编码块数.

2)算法描述

算法 CADA 的工作流程如算法 2 所示.

算法 2. 算法 CADA.

输入：缓存量阈值 $R_{\text{threshold}}$，用户选择的数据对象级缓存驱逐算法 F.

① while(true)
② *event*←阻塞读取事件队列；
③ if(*event.type*==数据访问事件)
④ *blocks*←从 *event* 中获取被 AFERS 选中的编码块列表；
⑤ *time*←从 *event* 中获取各编码块的实际传输用时；
⑥ *slowBlocks*←计算传输用时长的编码块(*time*, *blocks*)；
⑦ 缓存编码块并设置缓存优先级(*slowBlocks*)；
⑧ *R*←获取当前缓存数据量；
⑨ if($R>R_{\text{threshold}}$)
⑩ *O*←选择拟驱逐的数据对象(*F*, *R*, $R_{\text{threshold}}$)；
⑪ 调整拟驱逐数据对象的编码块的缓存优先级并将缓存优先级为 0 的编码块设为 tmp 状态(*O*)；
⑫ end if
⑬ end if
⑭ if(*event.type*==定时缓存回收事件)
⑮ *O*←选择拟驱逐的数据对象(*F*, *R*, $R_{\text{threshold}}$)；
⑯ 调整拟驱逐数据对象的编码块的缓存优先级并将缓存优先级为 0 的编码块设为 tmp 状态(*O*).
⑰ end if
⑱ end while

算法 CADA 运行于云内计算节点对应的云代理节点上，会不停阻塞读取事件队列中的事件(算法 2 的行①②)，其中包括数据访问事件和定时缓存回收事件等.

若读取到数据访问事件，则从事件中获取被算法 AFERS 选中的编码块信息，以及各编码块的实际传输用时(算法 2 的行③~⑤). 然后，根据获得的信息得到被 AFERS 选中的编码块中传输用时较长的编码块列表 *slowBlocks*，传输用时较长的编码块指实际传输用时大于 $\bar{t}+j\sigma$ 的编码块，其中 $\bar{t}$，σ 分别为各编码块传输用时的平均值和标准差(算法 2 的行⑥)，j 为经验常量. 接着，将 *slowBlocks* 中的编码块加入缓存，并将其中传输用时最长的编码块的缓存优先级设为 1，其他编码块的缓存优先级设为 0(算法 2 的行⑦). 随后，如果当前缓存的总数据量超过阈值，则调用用户选择的数据对象级缓存驱逐算法 F 确定拟驱逐的数据对象(算法 2 的行⑧~⑩). 最后，调整拟驱逐数据对象的编码块的缓存优先级(将缓存优先级为 1 的编码块的缓存优先级调为 0)并将缓存优先级为 0 的编码块设为 tmp 状态(算法 2 的行⑪).

若读取到定时缓存回收事件，则调用用户选择的数据对象级缓存驱逐算法 F 确定拟驱逐的数据对象，并调整拟驱逐数据对象的编码块的缓存优先级(将缓存优先级为 1 的编码块的缓存优先级调为 0)并将缓存优先级为 0 的编码块设为 tmp 状态(算法 2 的行⑭~⑯).

3)算法分析

①算法 CADA 选择缓存被算法 AFERS 选中且实际传输用时较长的编码块，可以提高缓存命中量和缓存命中增效，从而缩短编码块传输用时，进而提高数据访问速度.

②算法 CADA 对实际传输用时相对较长的编码块(缓存优先级为 1)进行了保护，使其需要经历 2 次缓存驱逐操作且期间未被命中才会被清理，有利于提高缓存命中增效.

③不同数据对象级缓存驱逐算法适用于不同的场景，由于算法 CADA 可适配用户自定义的数据对象级缓存驱逐算法，故其适用场景较为丰富.

4 实验与结果

4.1 原型实现

为评估方法 AECAM 的性能，我们实现了一个面向跨云存算联调的存储系统(cross-cloud storage system for collaborative scheduling of storage and computation, C2S2)，并在该系统中实现了 AECAM.

C2S2 遵循基于 IPFS 的跨云存储系统框架 IBCS，该框架由协调组件、元数据组件、代理组件、IPFS 组件、客户端组成，如图 11 所示.

为避免单点故障，协调组件有多个，运行于多个协调节点上. 各协调组件不断阻塞读取竞争消息队列中的消息，并在成功读取消息后将其解析执行. 其中，消息由客户端写入，包括文件写入请求、访问请求等.

每个云代理节点上运行一个代理组件. 代理组件不断阻塞读取其专有消息队列中的消息，并在成功读取消息后将其解析执行. 其中，消息由协调组件写入，包括具体的编码块操作指令. 此外，代理组件

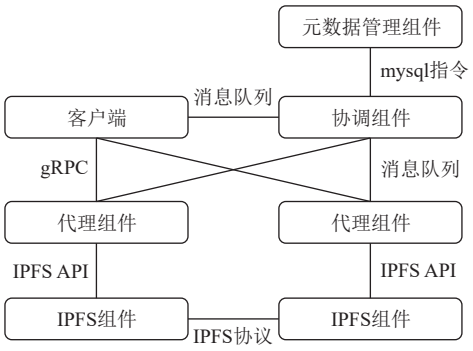


Fig. 11 Illustration of communications between C2S2 modules

图 11 C2S2 各组件间消息交互示意图

上运行着 gRPC(Google remote procedure call)服务，负责与客户端进行数据传输.

各协调节点元数据组件共同构成 mysql 集群，负责存储系统中的各类元数据：存储服务元数据、数据对象元数据、代理节点元数据、代理节点编码块元数据、云内计算节点数据对象元数据、编码块元数据. 为保证元数据安全，该 mysql 集群只能被协调组件通过 mysql 指令访问.

每个云代理节点上运行一个 IPFS 组件，各 IPFS 组件组成 IPFS 集群. 各 IPFS 组件可被其所在云代理节点上的代理组件通过 IPFS API 访问，也可与 IPFS 集群中的其他 IPFS 组件通过 IPFS 协议共同完成编码块的 P2P 传输.

4.2 实验设置

实验环境包括阿里云和华为云在北京和上海的 3 个云数据中心的 6 个节点(云主机)，各云数据中心间的带宽如图 12 所示. 华为云主机配备 2 核第 3 代 Intel 至强 3.0 GHz 处理器、4 GiB 内存和 40 GB 云硬盘；阿里云主机配备 2 核第 3 代 Intel Xeon2.7 GHz 处理器，4 GiB 内存和 40 GB 云硬盘. 其中，节点 1~5 为云代理节点，同时充当云内计算节点；节点 6 为协调节点，同时充当云外计算节点. 其中，华为云主机对应的云存储服务为 MinIO 对象存储服务，阿里云主机对应的云存储服务为 OSS 对象存储服务.

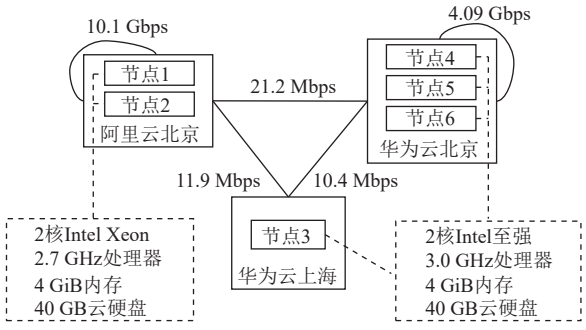


Fig. 12 Illustration of cross-cloud experimental environment

图 12 跨云实验环境示意图

为评估 C2S2 的性能，我们将其与 2 个引入缓存的基于纠删码的存储系统 Agar 和 POCache 进行了对比测试.

Agar[16] 默认采取直接数据访问方案，将编码块的全局访问次数与各数据访问节点获取该编码块开销的乘积作为各数据访问节点缓存该编码块的预期收益，并将预期收益较高的编码块缓存到对应的数据访问节点.

POCache[15] 将校验块缓存到数据访问节点附近，在访问数据时请求所有的数据块. 当访问的数据对象存在校验块缓存时，POCache 同时从缓存节点请求校验块，等待 k 个编码块到达后采取直接或降级数据访问方案解码计算重构原始数据对象.

实验采用的参数及默认值如表 2 所示.

Table 2 Parameters in Experiments

表 2 实验参数

参数	参数含义	默认值	取值范围
S_c	缓存容量/%	20	2, 5, 10, 15, 20, 25
a	Zipfian 分布参数	0.9	0.2, 0.5, 0.8, 0.9, 1.0, 1.1, 1.4
(n, k)	编码参数	(9,6)	(5,3), (9,6), (11,8), (16,10)
S_f	数据相似度/%	0	0, 30, 60, 90

缓存容量 S_c 是指缓存空间大小与待读取数据对象总大小之比.

实验中，数据对象的访问服从 Zipfian 分布[25]，少数热点数据的访问次数占总数据访问次数的比例越大，Zipfian 分布的参数 a 越大.

数据相似度 S_f 是指 2 个数据对象在内容上的相似程度. 在本次实验中，按照数据相似度将每 2 个数据对象划分为一个小组，小组内的 2 个数据对象在内容上相似，相似度为 S_f；任意小组之间的数据对象数据相似度为 0.

4.3 评价指标

我们使用 3 个指标来评价面向跨云存算联调的存储系统 C2S2 的性能.

1)缓存命中量

若存储系统以编码块为粒度进行缓存管理，则该存储系统的缓存命中量 HS 为经过多轮数据访问操作后被命中的缓存编码块的总大小 HS_{block}；若存储系统以数据分片为粒度进行缓存管理，则该存储系统的缓存命中量 HS 为经过多轮数据访问操作后被命中的缓存数据分片的总大小 HS_{slice}.

2)跨云传输量

若存储系统以编码块为粒度进行缓存管理, 则该存储系统的跨云传输量 CT 为经过多轮数据访问操作后被跨云传输的编码块的总大小 CT_{block}; 若存储系统以数据分片为粒度进行缓存管理, 则该存储系统的跨云传输量 CT 为经过多轮数据访问操作后被跨云传输的数据分片的总大小 CT_{slice}.

3)数据访问速度

若被访问数据大小为 M, 且从用户发起数据访问命令到云内计算节点对应的云代理节点将数据重构出来并上传到对应的云存储服务上所消耗的时间为 t_1, 则云内计算节点访问该数据的速度为 M/t_1; 若被访问数据大小为 M, 且从用户发起数据访问命令, 到云外计算节点对应的云代理节点上将数据重构出来所消耗的时间为 t_2, 则云外计算节点访问该数据的速度为 M/t_2.

实验中的缓存命中量、跨云传输量由仿真实验测得, 数据访问用时由真实跨云环境下的实验测得.

4.4 缓存命中量

图 13 显示了缓存命中量随编码块缓存决策所采取的 j-sigma 原则中经验常量 j 的变化而变化. 随着 j 的增加, 缓存命中量逐渐上升. 当 j 增加到 3 后, 缓存命中量基本上不再随 j 的增加而显著增加. 因此, 本文实验中取 j=3.

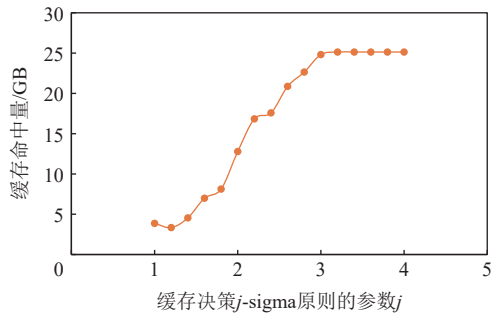


Fig. 13 Variation of cache hit volume with parameter j of the j-sigma caching decision principle

图 13 缓存命中量随缓存决策 j-sigma 原则的参数 j 的变化

图 14 显示了不同 Zipfian 分布参数 a 下 Agar, POCache, C2S2 的缓存命中量. Agar, POCache, C2S2 的缓存命中量随 a 的增大而增加. 这是由于 a 越大, 热点数据访问次数占总数据访问次数的比例越大, 因而存储系统缓存的热点数据的编码块被命中的次数越多.

图 15 显示了不同缓存容量下 Agar, POCache, C2S2 的缓存命中量. Agar, POCache, C2S2 的缓存命中量均

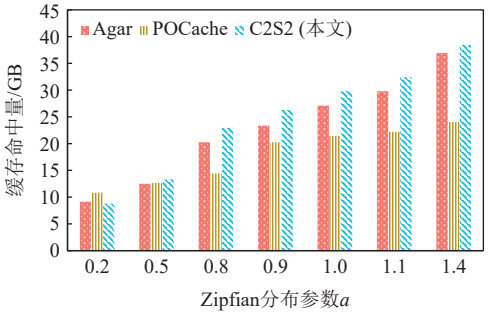


Fig. 14 Comparison of cache hit volume under different Zipfian distribution parameter a

图 14 不同 Zipfian 分布参数 a 下缓存命中量对比

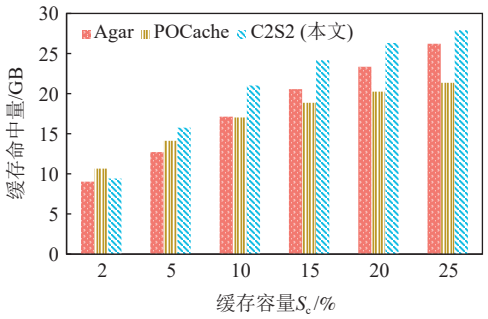


Fig. 15 Comparison of cache hit volume under different cache sizes

图 15 不同缓存容量下缓存命中量对比

随着缓存容量的增加而增加, 这是由于缓存容量越大, 可缓存的编码块越多.

图 16 显示了不同编码参数下 Agar, POCache, C2S2 的缓存命中量. Agar 和 C2S2 的缓存命中量对编码参数的变化不敏感, 而 POCache 的缓存命中量对编码参数的变化较为敏感且其缓存命中量除编码参数(5, 3)外均低于 Agar 和 C2S2. 这是因为 POCache 仅缓存校验块, 使得其对编码块中校验块的占比较为敏感. 校验块占比越少, 可缓存的编码块越少, 进而可命中的编码块越少.

图 17 显示了不同数据相似度下 Agar, POCache,

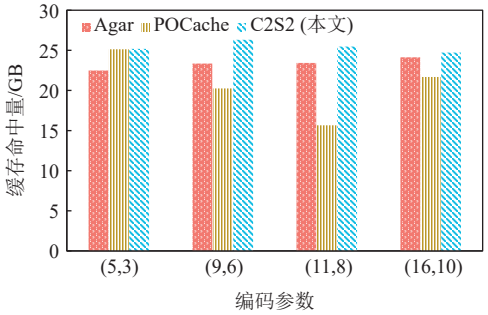


Fig. 16 Comparison of cache hit volume under different encoding parameters

图 16 不同编码参数下缓存命中量对比

C2S2 的缓存命中量. C2S2 的缓存命中量随着数据相似度的增加而增加, 而 Agar 和 POCache 的缓存命中量几乎不受数据相似度影响. 这是由于 C2S2 实现了数据分片级的缓存管理, 相似数据的编码块中的相同数据分片仅被存储了 1 次, 因而数据相似度越大, 缓存空间中能存储的编码块越多, 使得缓存命中量越大.

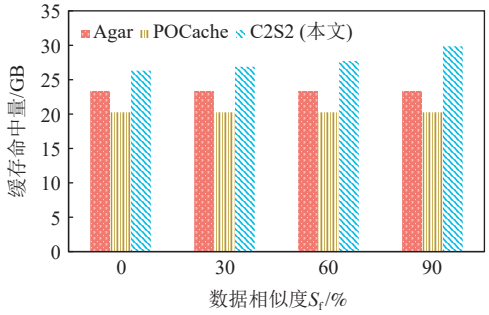


Fig. 17 Comparison of cache hit volume under different data similarity

图 17 不同数据相似度下缓存命中量对比

总体而言, C2S2 的平均缓存命中量比 Agar 和 POCache 高了 10.05% 和 65.4%, 主要原因为: 1)C2S2 实现了细粒度的缓存管理; 2)C2S2 为各已缓存的编码块设置了优先级, 可以更持久地保留数据访问方案中传输效率低的编码块, 有利于减少缓存抖动, 提高缓存命中量; 3)C2S2 可以根据实际情况缓存各类编码块, 而 POCache 只能缓存校验块且 Agar 只能缓存数据块.

4.5 跨云传输量

图 18 显示了不同 Zipfian 分布参数 a 下 Agar, POCache, C2S2 的跨云传输量. Agar 和 C2S2 的跨云传输量随 a 的增大而显著减少. 这是因为参数 a 越大, 热点数据访问次数占总数据访问次数的比例越大, 因而存储系统缓存的热点数据的编码块被命中的次数越多, 进而从缓存中直接获取的数据的总量越大. 然而, POCache 的跨云传输量受 a 的影响较小. 这是由于 POCache 在访问数据时首先访问所有的数据块, 然后舍弃最后到达的数据块, 使得其跨云传输量与缓存是否命中几乎无关.

图 19 显示了不同缓存容量 S_c 下 Agar, POCache, C2S2 的跨云传输量. 随着缓存容量的增大, C2S2 的跨云传输量有明显下降, 这是由于缓存容量越大, 可缓存的编码块越多, 缓存命中量越大. 然而, POCache 和 Agar 的跨云传输量并没有随着缓存容量的增加而明显下降. 这是因为 Agar 需要定期进行预缓存和缓存调整, 带来了额外的跨云传输量, 而 POCache 在每

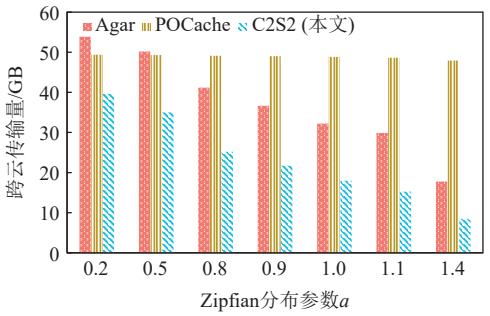


Fig. 18 Comparison of cross-cloud transfer volume under different Zipfian distribution parameter a

图 18 不同 Zipfian 分布参数 a 下跨云传输量对比

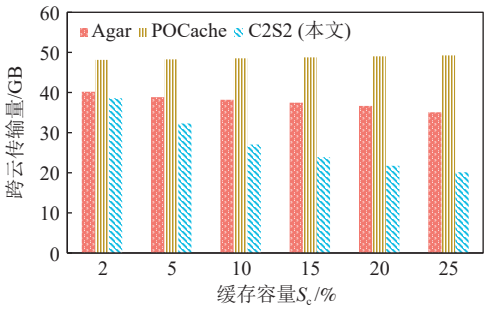


Fig. 19 Comparison of cross-cloud transfer volume under different cache sizes

图 19 不同缓存容量下跨云传输量

次访问数据时都会请求所有的数据块.

图 20 显示了不同编码参数下 Agar, POCache, C2S2 的跨云传输量. 三者跨云传输量基本上不受编码参数的影响. 这是由于 C2S2, Agar 的缓存命中量对编码参数不敏感, 且 POCache 在任何编码参数下访问数据时都会请求所有的数据块.

图 21 显示了不同数据相似度 S_f 下 Agar, POCache, C2S2 的跨云传输量. C2S2 的跨云传输量随着数据相似度的增加而减少; Agar, POCache 的跨云传输量几乎不受数据相似度影响. 这是由于 C2S2 实现了细粒度的缓存管理, 在访问数据时可以利用相似数据的

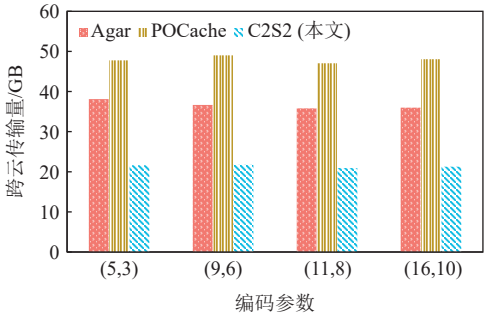


Fig. 20 Comparison of cross-cloud transfer volume under different encoding parameters

图 20 不同编码参数下跨云传输量对比

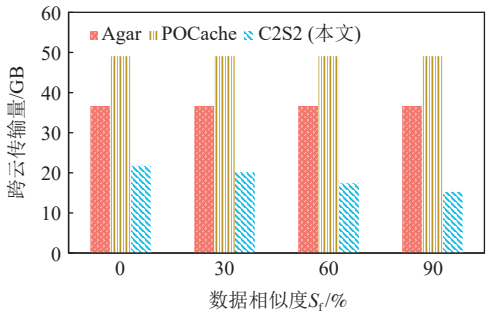


Fig. 21 Comparison of cross-cloud transfer volume under different data similarity

图 21 不同数据相似度下跨云传输量对比

编码块缓存来减少跨云传输量.

总体而言, C2S2 的平均跨云传输量比 Agar 和 POCache 分别低了 30.13% 和 53.69%, 主要原因为: 1)C2S2 只针对读取到的数据进行缓存, 不产生额外的跨云传输量.2)C2S2 利用数据内容寻址机制, 可以让相似数据对象从缓存中读取一致的数据分片, 减少跨云传输量.

4.6 数据访问速度

图 22 显示了不同数据相似度下 Agar, POCache, C2S2 的数据访问速度. 当数据相似度为 0 时, C2S2 与 POCache 和 Agar 相比, 可将数据访问速度提升 28.58%~33.04%. 这是由于 C2S2 的缓存命中量更高、缓存命中增效高, 可有效降低低带宽云间的数据传输量.

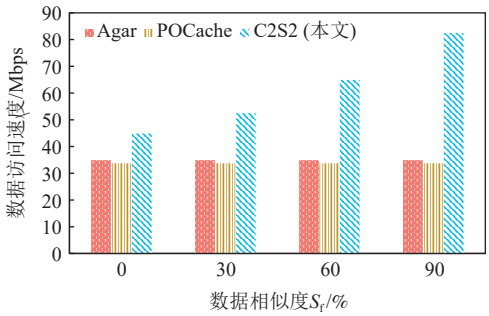


Fig. 22 Comparison of average data access speed under different data similarity

图 22 不同数据相似度下平均数据访问速度

此外, 当数据相似度为 30%, 60%, 90% 时, C2S2 的数据访问速度逐渐升高. 这是由于 C2S2 支持细粒度的缓存管理, 在访问数据时可以利用相似数据的编码块缓存来减少跨云传输量.

整体而言, C2S2 的数据访问速度平均比 POCache 和 Agar 高 75.22%~81.29%. 主要原因为:

1) C2S2 使用 IPFS 集群从多个云代理节点以数据分片为单位并发传输编码块, 能够有效提高编码块传输速度;

2) C2S2 可以评估各编码块的传输速度, 并自适应制定编码数据访问方案以规避传输速度慢的编码块;

3) C2S2 的缓存管理算法可对数据访问过程进行感知, 并通过设置优先级, 更为持久地保留数据访问方案中传输效率最低的瓶颈编码块, 提高缓存命中增效.

5 总 结

在框架创新方面, 本文提出了一种基于 IPFS 的跨云存储系统框架 (IBCS), 使用 IPFS 集群存储和缓存数据, 并有效利用 IPFS 的数据分片管理机制实现细粒度的缓存管理, 因而能提高缓存命中率.

在技术创新方面, 本文提出了一种面向存算联调的跨云纠删码自适应数据访问方法 (AECAM), 实现了纠删码编码数据缓存管理策略与编码数据访问方案选择策略深度协同优化, 在选择编码数据访问方案时以各编码块的存储和缓存情况为依据, 在设置编码数据缓存优先级时以编码数据访问方案对各编码块的评估结果和实际传输时间为依据, 可同时提高缓存命中量和命中增效.

在软件实现方面, 本文基于 IPFS 的跨云存储系统框架 (IBCS) 和面向存算联调的跨云纠删码自适应数据访问方法 (AECAM), 设计实现了一种面向跨云存算联调的存储系统 (C2S2). 与现有引入缓存的基于纠删码的存储系统 POCache 和 Agar 相比, 可将数据访问速度提高 75.22%~81.29%.

未来, 我们计划进一步研究框架 IBCS 的数据修复和数据更新性能优化问题. 具体研究如何通过合理管理缓存编码块来缩短数据修复用时, 以及如何利用框架 IBCS 的数据分片管理机制降低数据更新开销.

作者贡献声明: 张凯鑫提出主要研究思路, 完成实验并撰写论文; 王意洁提出指导意见, 修改和审核论文; 包涵提出实验方案, 参与修改论文; 阚浚晖协助完成实验.

参 考 文 献

[1] Aggarwal C, Charu C. Outlier Analysis[M]. Berlin: Springer, 2017

[2] Schmuck F, Haskin R. GPFS: A shared-disk file system for large computing clusters[C/OL] // Proc of the 1st USENIX Conf on File

and Storage Technologies. Berkeley, CA: USENIX Association, 2002 [2023-06-18].https://cse.buffalo.edu/faculty/tkosar/cse710_spring14/papers/gpfs.pdf

[3] Ghemawat S, Gobioff H B, Leung S. The Google file system[J]. ACM SIGOPS Operating Systems Review, 2003, 37(5): 29−43

[4] Halbwachs N, Caspi P, Raymond P, et al. The synchronous data flow programming language Lustre[J]. Proceedings of the IEEE, 1991, 79(9): 1305−1320

[5] Weil S A, Brandt S A, Miller E L, et al. Ceph: A scalable, high-performance distributed file system[C] // Proc of the 7th Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2006: 307-320

[6] Wang Yijie, Xu Fangliang, Pei Xiaoqiang. Research on erasure code-based fault-tolerant technology for distributed storage[J]. Chinese Journal of Computers, 2017, 40(1): 236−255(in Chinese)
(王意洁, 许方亮, 裴晓强. 分布式存储中的纠删码容错技术研究[J]. 计算机学报, 2017, 40(1): 236−255)

[7] Wang Yijie, Pei Xiaoqiang, Ma Xingkong, et al. TA-Update: An adaptive update scheme with tree-structured transmission in erasure-coded storage systems[J]. IEEE Transactions on Parallel & Distributed Systems, 2017, 29(8): 1893−1906

[8] Bao Han, Wang Yijie, Xu Fangliang. A cross- datacenter erasure code writing method based on generator matrix[J]. Journal of Computer Research and Development, 2020, 57(2): 291−305(in Chinese)
(包涵, 王意洁, 许方亮. 基于生成矩阵变换的跨数据中心纠删码写入方法[J]. 计算机研究与发展, 2020, 57(2): 291−305)

[9] Shen Zhirong, Shu Jiwu, Huang Zhijie, et al. ClusterSR: Cluster-aware scattered repair in erasure-coded storage[C]// Proc of the 34th IEEE Int Parallel and Distributed Processing Symp (IPDPS). Piscataway, NJ: IEEE, 2020: 42−51

[10] Fu Yingxun, Liu Xun, Shu Jiwu, et al. Device and placement aware framework to optimize single failure recoveries and reads for erasure coded storage system with heterogeneous storage devices[C]// Proc of the 39th Int Symp on Reliable Distributed Systems (SRDS). Piscataway, NJ: IEEE, 2020: 225−235

[11] Lakshmi J, Mohan K, Rajawat U, et al. Optimal placement for repair-efficient erasure codes in geo-diverse storage centres[J]. Journal of Parallel and Distributed Computing, 2020, 135(C): 101−113

[12] Saeed S. Sandooq: Improving the communication cost and service latency for a multi-user erasure-coded geo-distributed cloud environment[D]. Urbana, Illinois: University of Illinois at Urbana-Champaign, 2016

[13] Zhang Xingjun, Cai Yi, Liu Yunfei, et al. NADE: Nodes performance awareness and accurate distance evaluation for degraded read in heterogeneous distributed erasure code-based storage[J]. The Journal of Supercomputing, 2019, 76(6): 1−30

[14] Rashmi K V, Chowdhury M, Kosaian J, et al. EC-Cache: Load-balanced, low-latency cluster caching with online erasure coding[C]// Proc of the 12th USENIX Conf on Operating Systems Design and Implementation. Berkeley, CA: USNEX Association, 2016: 401−417

[15] Zhang Mi, Wang Qiuping, Shen Zhirong, et al. POCache: Toward robust and configurable straggler tolerance with parity-only caching[J]. Journal of Parallel and Distributed Computing, 2022, 167(C): 157−172

[16] Halalai R, Felber P, Kermarrec A M, et al. Agar: A caching system for erasure-coded data[C]// Proc of the 37th IEEE Int Conf on Distributed Computing Systems. Piscataway, NJ: IEEE, 2017: 23−33

[17] Abebe M, Daudjee K, Glasbergen B, et al. EC-Store: Bridging the gap between storage and latency in distributed erasure coded systems[C]//Proc of the 38th IEEE Int Conf on Distributed Computing Systems. Piscataway NJ: IEEE, 2018: 255−266

[18] Zhou Jiang, Xie Wei, Dai Dong, et al. PRS: A pattern-directed replication scheme for heterogeneous object-based storage[J]. IEEE Transactions on Computers, 2020, 69(4): 591−605

[19] Al-Abbasi A, Aggarwal V. TTLCache: Taming latency in erasure-coded storage through TTL caching[J]. IEEE Transactions on Network and Service Management, 2020, 17(3): 1582−1596

[20] Aggarwal V, Chen Y, Lan Tian, et al. Sprout: A functional caching approach to minimize service latency in erasure-coded storage[C]//Proc of the 36th IEEE Int Conf on Distributed Computing Systems. Piscataway NJ: IEEE, 2016: 753−754

[21] Shankar K C P, Shyry S P. A novel framework for securing ECDH encrypted DICOM pixel data stored over cloud using IPFS[J]. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 2023, 31(Supp01): 135−164

[22] Li Wenjuan, Wang Yu, Li Jin. Enhancing blockchain-based filtration mechanism via IPFS for collaborative intrusion detection in IoT networks[J]. Journal of Systems Architecture, 2022, 127(C): 102510

[23] Doan T V, Psaras Y, Ott J, et al. Towards decentralised cloud storage with IPFS: Opportunities, challenges, and future directions[J]. IEEE Internet Computing, 2022, 26(6): 7−15

[24] Liu Shuyan. Research on domain name resolution technology based on IPFS[J]. Intelligent Computer and Applications, 2020, 10(2): 365−369(in Chinese)
(刘姝言. 基于 Fabric 和 IPFS 的域名解析技术研究[J]. 智能计算机与应用, 2020, 10(2): 365−369)

[25] Szabo G, Huberman B A. Predicting the popularity of online content[J]. Communications of the ACM, 2010, 53(8): 80−88

Zhang Kaixin, born in 2002. Master candidate. His main research interests include cloud storage and erasure coding.
张凯鑫, 2002 年生. 硕士研究生. 主要研究方向为云存储、纠删码.

Wang Yijie, born in 1971. PhD. professor, PhD supervisor. Distinguished member of CCF. Her main research interests include distributed storage, big data analysis, and cloud computing.
王意洁, 1971 年生. 博士, 教授, 博士生导师. CCF 杰出会员. 主要研究方向为分布式存储、大数据分析、云计算.

Bao Han, born in 1992. PhD. His main research interests include cloud storage and erasure coding.

包　涵, 1992 年生. 博士. 主要研究方向为云存储、纠删码.

Kan Junhui, born in 2000. Master candidate. His main research interests include cloud storage and collaborative scheduling of storage and computation.

阚浚晖, 2000 年生. 硕士研究生. 主要研究方向为云存储、存算联调.