毕业设计（论文）报告

题目　集群缓存系统负载均衡算法的设计与实现

计算机科学与工程 院（系）计算机科学与技术 专　业

学　　号　09015131

学生姓名　郑云川

指导教师　王威

顾问老师　肖卿俊

起讫日期　2018 年 12 月—2019 年 6 月

设计地点　HKUST

东南大学毕业（设计）论文独创性声明

本人声明所呈交的毕业（设计）论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得东南大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

作者签名：_____

日 期：_____

东南大学毕业（设计）论文使用授权声明

东南大学有权保留本人所送交毕业（设计）论文的复印件和电子文档，可以采用影印、缩印或其他复制手段保存论文。本人电子文档的内容和纸质论文的内容相一致。除在保密期内的保密论文外，允许论文被查阅和借阅，可以公布（包括刊登）论文的全部或部分内容。论文的公布（包括刊登）授权东南大学教务处办理。

作者签名：_____

导师签名：_____

日 期：_____

集群缓存系统负载均衡算法的设计与实现

09015131　郑云川
指导教师　王威

摘　要

希腊字母源自腓尼基字母。腓尼基字母只有辅音，从右向左写。希腊语的元音发达，希腊人增添了元音字母。因为希腊人的书写工具是蜡板，有时前一行从右向左写完后顺势就从左向右写，变成所谓“耕地”式书写，后来逐渐演变成全部从左向右写。字母的方向也颠倒了。罗马人引进希腊字母，略微改变变为拉丁字母，在世界广为流行。希腊字母广泛应用到学术领域，如数学等。

希腊字母是希腊语所使用的字母，是世界上最早的有元音的字母，也广泛使用于数学、物理、生物、天文等学科。俄语等使用的西里尔字母也是由希腊字母演变而成。希腊字母进入了许多语言的词汇中，英语单字“alphabet”（字母表），源自拉丁语“alphabetum”，源自希腊语“αλφαβητον”，即为前两个希腊字母 α（“Alpha”）及 β（“Beta”）所合成，三角洲（“Delta”）这个词就来自希腊字母 Δ，因为 Δ 是三角形。

关键词：希腊字母，腓尼基字母，语言，深度学习

Design and Implementation of Load Balance Algorithms in Cluster Cache Systems

09015131 Yunchuan Zheng
Advisor Wei Wang

Abstract

The Greek alphabet has been used to write the Greek language since the late 9th century BC or early 8th century BC It was derived from the earlier Phoenician alphabet, and was the first alphabetic script to have distinct letters for vowels as well as consonants. It is the ancestor of the Latin and Cyrillic scripts.Apart from its use in writing the Greek language, in both its ancient and its modern forms, the Greek alphabet today also serves as a source of technical symbols and labels in many domains of mathematics, science and other fields.

In its classical and modern forms, the alphabet has 24 letters, ordered from alpha to omega. Like Latin and Cyrillic, Greek originally had only a single form of each letter; it developed the letter case distinction between upper-case and lower-case forms in parallel with Latin during the modern era.

KEY WORDS: Greek Alphabet, Phoenician Alphabet, Language, Deep Learning

目　录

第一章 前言

本章是课题的前言部分。在此章中，首先介绍了课题的实际背景，接下来是课题的目的和意义，并且对当前的研究现状做了简要调研与分析，最后介绍了课题的主要研究内容。

1.1 课题背景

1.1.1 大数据

因为技术的不断发展，包括物联网，云计算的崛起[1]、智能设备的流行等，在当今时代各种各样不同的领域（例如健康领域、政府、社交网络、营销、财务），每一天都在以前所未有的速度产生大量的数据[2]。从海量的数据中，我们能够挖掘出大量有用的规律，对人们的生活产生积极的影响。在大数据革命之前，大公司很难将他们的数据存档保存较长时间，也难以管理庞大的数据集。传统技术存储能力有限，管理工具很昂贵，它们缺乏大数据背景所需要的灵活性、可扩展性和性能。事实上，大数据管理需要大量资源，新方法和强大技术，进一步来说，大数据需要清洗，处理，分析，保护数据，并提供对大量不断发展的数据集的细粒度访问[2]。为了应对大数据带来的机遇和挑战，学界和业界开展了大量的研究与开发工作，发展出来众多技术，提供了很多成熟的模型、框架、软件。典型的互联网大数据平台（如图 1.1）从上至下大致可分为三个部分：

- 数据采集：将应用程序产生的数据和日志等同步到大数据系统，同步时数据可能还需经过清洗、转化等过程；
- 数据处理：包括大数据存储、离线计算和流式计算等；
- 数据输出与展示：大数据经过处理后将有价值的信息存入数据库，通过数据库给用户提供所需信息，或者给运营、决策人员提供所需信息。

此外，将三个部分整合起来的是大数据任务调度管理系统，它会管理数据的同步、集群资源的分配等等。

在大数据平台中（图 1.1），数据处理是非常重要的一环，实现对数据的高效清洗、存储和挖掘是这个环节的目标。对计算能力的需求随着数据量的急剧增加而增加，但是单机的处理能力和 I/O 性能并没有跟上这种增长，越来越多的企业不得不向外扩展他们的计算至集群模式[3]。集群环境对编程平台提出了更高的要求，主要有三方面，一是程序需要并行化执行，二是需要强大的容错能力，三是动态地扩展和缩减计算资源，为此，越来越多的编程模型被设计出来。在存储方面，谷歌提出了分布式文件系统 GFS（Google File System）[4]，对应的开源实现是 HDFS（Hadoop File System）[5]，它实现了对成千上万台机器上的大规模数据进

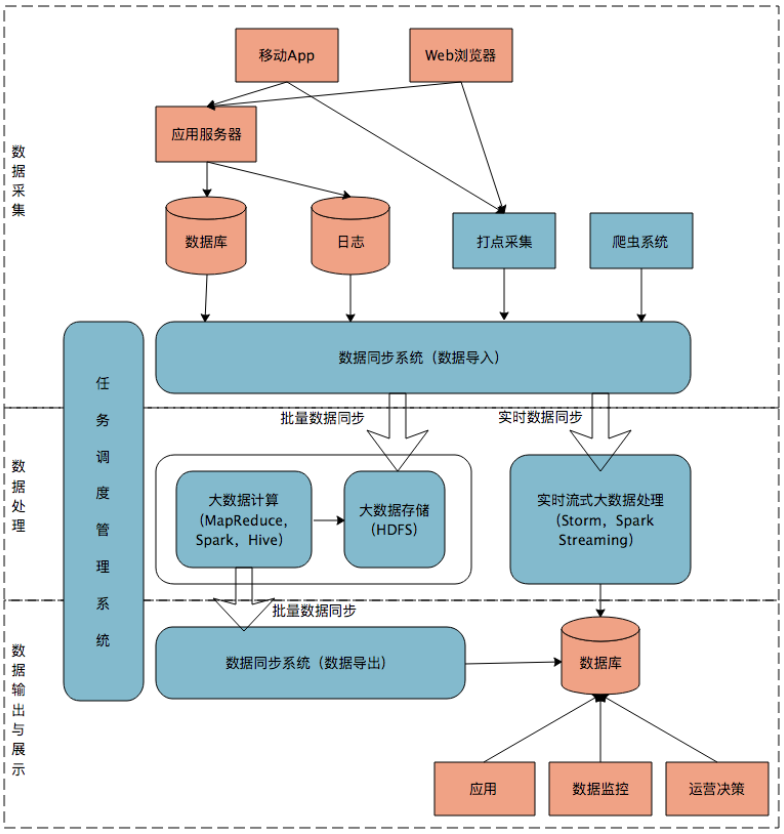

图 1.1 典型的互联网大数据平台架构。

行高效地存储、访问，并且具有高度容错能力。计算框架方面，起初，谷歌的 MapReduce[6] 提出了一种简单通用而且能够自动处理故障的批处理计算模型，但是它将中间以及最终结果保存在磁盘上，消耗大量 I/O 时间，不利于重用计算结果。Spark[3]、Pregel[7] 等采用内存计算方案来加速计算，实现数据的重复利用。

1.1.2 集群缓存

由于近来数据中心架构的改进 [8] 和高速网络设备的出现[9–11]，网络带宽和存储器 I/O 带宽之间的差距正在迅速减小 [12–15]，因此，云计算系统的性能瓶颈正迅速从网络转变为存储器 I/O。先前的工作证明从本地硬盘读取数据相比网络远程读取并没有显著的优势 [16]，这个结论同样适用于固态硬盘（SSD）。最近的一个研究 [17] 表明将数据存储在 EC2 实例的一个本地 SSD 上甚至比把数据写到 Amazon S3 [18] 上还要慢，Amazon S3 是一个远程的提供了 PUT/GET 接口的对象存储服务。当磁盘本地化变得无关紧要，云端对象存储如 Amazon S3 [18]、Windows Azure Storage [19]、和 OpenStack Swift [20] 等，逐渐取代与计算同地的存储（尤其是 HDFS [21]），作为数据密集型应用的首选存储方式。

然而，云端对象存储在磁盘 I/O 上依然是瓶颈 [22]，因为从磁盘读数据比从内存读数据慢至少两个数量级，考虑到这个问题，集群缓存系统，例如 Alluxio [23]、Memcached [24] 和 Redis [25]，被越来越多的云端对象存储系统部署来提供内存速度级别的低延迟数据访问，而集群缓存系统面对的一个很大的挑战便是如何实现负载均衡。

一个能够实现分布式内存缓存的系统是 Alluxio。Alluxio[26] 是开源的分布式内存文件系统，旨在作为上层繁多的计算框架（如 MapReduce、Apache Spark、Apache Storm、Apche

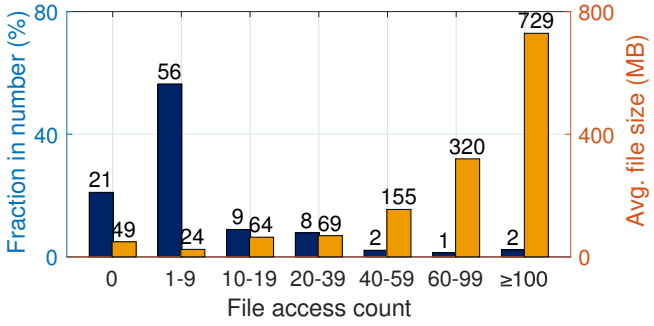

图 1.2 Yahoo! 集群上观察到的文件热门程度（蓝色）和文件大小（橙色）的分布 [29].

Mahout 等）与底层存储层（如文件系统、对象存储、键值对存储等）的中间层，提供统一的文件读写的接口，实现全局的数据访问、高效的内存数据共享、跨应用的数据管理、高效的网络带宽利用，它借助“血缘关系”、检查点机制提供强大的容错能力。鉴于这些特点，alluxio 也非常适合作为内存缓存系统，进一步加速数据分析应用。

1.1.3 负载均衡

上一小节提及的 Apache Spark 等内存计算方案的一个重要挑战是负载不均，在先前工作 [22, 27] 中，研究人员已经发现了生产集群中负载不均的两个来源：文件热门程度差别 和 网络流量不均。

在数据中心中，我们普遍观察到，文件（数据对象）热门程度差别极大，并且遵循 Zipf 分布 [22, 26–28]，也就是说，数据访问的大部分请求是由一小部分非常热门的文件贡献的。图 1.2描述了 Yahoo! 集群查询数据集 [29] 中文件的热门程度和文件大小的分布，从这个数据集可以得到某两个月内对超过四千万个文件的访问的统计数据。我们发现绝大多数文件（$\sim 78\%$）存储的是冷门数据，很少被访问（< 10 times），只有 2% 的文件有高访问量（≥ 100），这些文件通常比那些冷门的文件大很多（15-30×）。由于这些文件较大的体积和较高的访问量，缓存这些文件的服务器很容易负荷过重。

这个问题由于网络负载不均而加重，这在生产环境的数据中心非常普遍 [22, 30–32]，例如，在研究 [22] 中，研究者测量了 Facebook 一个集群所有上行和下行链路中最大利用率和平均利用率的比值，结果表明这个比值在半数以上的时间里保持在 4.5 以上，这意味着严重的负载不均。

1.2 研究现状

当今的数据并行集群依赖内存计算方案来进行高性能的数据分析工作[3, 24, 26, 33–35]，通过将数据对象缓存在内存中，I/O 密集型应用相对于传统的磁盘解决方案能够获得数量级的性能提升[3, 26, 34]

然而，内存计算方案面对的一个严峻的挑战是缓存服务器之间严重的负载不均衡。在生产集群中，数据对象有严重的热门程度差别，这意味着对一小部分非常热门的文件访问占据了总访问的很大一部分[22, 27, 28]。存储有热门文件的缓存服务器因此变为访问热点，这个问题因为网络的负载不均衡而进一步恶化。据报道，在 Facebook 的一个集群中，负载最重的链路的利用率在 50% 的时间里比平均链路利用率高出 4.5 倍[22]。访问热点和网络负载不均导致了

I/O 性能极大下降，这甚至可能会抵消内存计算带来的性能提升。

因此，保证负载均衡是提高集群缓存性能的关键，这方面的解决方案包括选择性复制[27]、纠删码[22] 和选择分割[36]，前二者是借助缓存冗余来减缓访问热点机器的负担，第三个是根据文件的热门程度将文件分割成不同份数，并随机放置在不同服务器上，分散请求负载。

1.2.1 选择复制

选择复制方案基于文件的热门程度对文件进行复制 [27, 37]，也就是说，一个文件的访问频率越高，它会越多被复制，并分散在集群中，一个文件的读请求就能随机选择一台含有这个文件副本的服务器提供服务。这样，读请求的负载就被均匀分散，提高负载均衡。

虽然选择复制已被证明对于基于磁盘的存储系统是有效的 [27]，但是因为复制带来了高额的内存开销，它在集群缓存上表现不佳 [22, 38]。研究 [36] 的实验得出，内存开销的 *线性增加*（热门文件副本数增加）换来了读延迟的 *亚线性降低*，而且热门文件的体积通常比较大（图 1.2）。

1.2.2 纠删码

有研究利用 *纠删码* [39, 40] 来实现缓存服务器之间的负载均衡且避免产生高额的内存开销。一个 (k, n) 的纠删码方案能够将一个文件均匀地切分成 k 份，然后计算同样大小的 $n-k$ 个 *奇偶校验分区*，原始文件能通过解码 n 份中的任意 k 份来获得，从而使得读请求的负载被分散到 n 台服务器上。内存的额外开销是 $(n-k)/k$，在实际设定中比选择复制低（至少 $1\times$）。

这个方法的一个有效实现是 EC-Cache [22]，它在读取文件时通过 *迟绑定* 来减轻落后机器的影响，换句话说，EC-Cache 随机读取文件分区中的 $k+1$ 份，等待其中的 k 份完成读取，而不是恰好读取 k 份。EC-Cache 在读文件的中位和尾延迟都比选择复制低很多 [22]。然而，EC-Cache 在读（写）时带来巨大的解码（编码）的额外开销，即使有高度优化的编码和实现方案，解码的开销仍会对读请求产生高达 30% [22] 延迟。

1.2.3 选择分割

研究 [36] 中提出了 SP-Cache 来实现集群缓存的负载均衡，同时避免高额的内存和计算开销。它选择性地将热门文件根据其大小和热门程度，分割成一定数目的文件分区，随机缓存到不同的缓存服务器上，这样分散了读请求的负载，同时读操作可以并行，提升性能。SP-Cache 建立了一个上限分析来量化平均延迟 [36]，并基于这个推导设计了一个高效的算法来决定每个文件的最佳分区数量，文件分割的数目太小则不足以缓解热点机器的压力，分割的数目太大则容易受到慢机器的影响。此外它采集一段时间内集群中文件的访问数据，周期性地调整各个文件的分区数目。选择分割在不产生高额内存和计算开销的情况下实现可负载均衡，但因为其分割的特性，缓存无冗余，容错性依赖底层文件系统，且读取文件必须读取所有分区，会受到慢机器的影响。

1.2.4 更细粒度负载均衡

以上方案都是针对一般意义上的文件来考虑负载均衡的，优点是非常通用，毋需考虑文件的语义，对于任何格式的文件都可以使用。它们负载均衡的粒度是文件，那么问题来了，能

否在更细的粒度进行负载均衡，提高缓存效率呢？对于语义清晰的结构化数据，比如 Parquet 文件 [41]，如果在文件的内部列与列存在热门程度差异，列之间被共同查询的概率也存在差异，那么就没有必要去分割或者复制整个文件，只要对一个文件热门的这一部分，例如其中一列或者多列进行复制或者分割就行。这样能够节约内存，提高使用效率，因为内存总是有限的，而且一部分内存需要给计算任务使用，那么留作缓存的就更少了。这个目标的挑战在于底层分布式文件系统需要了解文件的语义，与上层的应用通信来获得这部分信息，可能产生一定程度的耦合，同时我们需要明智地决定对文件的哪些列进行复制或分割，在哪些机器上进行缓存。

1.3 研究的目的与内容

当前的大数据系统主要采用复制的方式来进行容错和负载均衡，而服务器内存的容量往往有限，缓存会产生不可忽视的内存开销。根据本项目的前期调研，生产集群中结构化数据（数据表）的不同列之间，热门程度（被访问热度）存在差异，列与列之间共同被查询的概率也存在差异，我们希望复制数据表中比较热门的列，而不是全表，并基于列与列之间被共同查询的概率设计一定的放置策略，实现以更少的内存，来获得相似的负载均衡效果的目标，从而节约资源，提高缓存效率。

总的来说，本课题的主要研究内容是上文提出的针对结构化数据文件的更细粒度（列级别）的负载均衡方案，具体来说：

- 利用具有代表性的基准查询数据集，如 TPC-DS, TPC-H 等，测量数据表中列之间的查询频率，以及列与列之间被共同查询的频率，分析其中的统计及其他客观规律，为本项目的可行性奠定理论基础。
- 通过实验探究 SQL 查询过程中，数据的 shuffle 过程对任务执行时间的影响，证明数据表中相关列“捆绑放置”（bundle）的有效性，进一步强化项目的理论基础。
- 搭建 Spark SQL [42]，Alluxio [23]，HDFS [21] 为主的集群系统，探索各组件之间通信协作机制，为项目方案实现奠定基础。
- 基于已有条件，主要在 Alluxio [23] 基础上添加模块，使得 Alluxio 能够获取热门度以及关联性等信息，查阅相关文献，设计实现细粒度的负载均衡算法，并在 AWS EC2 搭建实验环境，进行测试。

1.4 论文结构

致　谢

这次的毕业论文设计总结是在我的指导老师 xxx 老师亲切关怀和悉心指导下完成的。从毕业设计选题到设计完成，x 老师给予了我耐心指导与细心关怀，有了莫老师耐心指导与细心关怀我才不会在设计的过程中迷失方向，失去前进动力。x 老师有严肃的科学态度，严谨的治学精神和精益求精的工作作风，这些都是我所需要学习的，感谢 x 老师给予了我这样一个学习机会，谢谢!

感谢与我并肩作战的舍友与同学们，感谢关心我支持我的朋友们，感谢学校领导、老师们，感谢你们给予我的帮助与关怀; 感谢肇庆学院，特别感谢计算机科学与软件学院四年来为我提供的良好学习环境，谢谢!

参考文献

[1] Botta A, De Donato W, Persico V, et al. Integration of cloud computing and internet of things: a survey[J]. Future generation computer systems, 2016, 56:684–700.

[2] Oussous A, Benjelloun F Z, Lahcen A A, et al. Big data technologies: A survey[J]. Journal of King Saud University-Computer and Information Sciences, 2018, 30(4):431–448.

[3] Zaharia M. An Architecture for Fast and General Data Processing on Large Clusters. USA: ACM Books, 2016.

[4] Ghemawat S, Gobioff H, Leung S T. The google file system. Proceedings of the 19th ACM Symposium on Operating Systems Principles, Bolton Landing, NY, 2003. 20–43.

[5] Apache hadoop. `https://hadoop.apache.org/`.

[6] Dean J, Ghemawat S. Mapreduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1):107–113.

[7] Malewicz G, Austern M H, Bik A J, et al. Pregel: a system for large-scale graph processing. Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM, 2010. 135–146.

[8] Singh A, Ong J, Agarwal A, et al. Jupiter rising: A decade of clos topologies and centralized control in Google's datacenter network. Proc. ACM SIGCOMM, 2015.

[9] Huawei. NUWA. `https://www.youtube.com/watch?v=OsmZBRB_OSw`.

[10] Asanovic K, Patterson D. FireBox: A hardware building block for 2020 warehouse-scale computers. USENIX FAST, 2014.

[11] Alistarh D, Ballani H, Costa P, et al. A high-radix, low-latency optical switch for data centers[J]. SIGCOMM Comput. Commun. Rev., 2015, 45(4):367–368.

[12] Scott C. Latency trends. `http://colin-scott.github.io/blog/2012/12/24/latency-trends/`.

[13] IEEE. Ieee p802.3ba 40 gbps and 100 gbps ethernet task force. `http://www.ieee802.org/3/ba/`.

[14] Han S, Egi N, Panda A, et al. Network support for resource disaggregation in next-generation datacenters. ACM HotNets, 2013.

[15] Gao P X, Narayan A, Karandikar S, et al. Network requirements for resource disaggregation. Proc. USENIX OSDI, 2016.

[16] Ananthanarayanan G, Ghodsi A, Shenker S, et al. Disk-locality in datacenter computing considered irrelevant. ACM HotOS, 2011.

[17] Jonas E, Venkataraman S, Stoica I, et al. Occupy the cloud: Distributed computing for the 99%. Proc. ACM SoCC, 2017.

[18] Amazon s3. https://aws.amazon.com/s3.

[19] Windows azure storage. https://goo.gl/RqVNmB.

[20] Openstack swift. https://www.swiftstack.com.

[21] Shvachko K, Kuang H, Radia S, et al. The Hadoop distributed file system. Proc. IEEE Symp. Mass Storage Syst. and Technologies, 2010, 2010.

[22] Rashmi K, Chowdhury M, Kosaian J, et al. Ec-cache: Load-balanced, low-latency cluster caching with online erasure coding. 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16), 2016. 401–417.

[23] Alluxio. http://www.alluxio.org/.

[24] Memcached. https://memcached.org/.

[25] Redis. https://redis.io/.

[26] Li H, Ghodsi A, Zaharia M, et al. Tachyon: Reliable, memory speed storage for cluster computing frameworks. Proceedings of the ACM Symposium on Cloud Computing. ACM, 2014. 1–15.

[27] Ananthanarayanan G, Agarwal S, Kandula S, et al. Scarlett: coping with skewed content popularity in mapreduce clusters. Proceedings of the sixth conference on Computer systems. ACM, 2011. 287–300.

[28] Ananthanarayanan G, Ghodsi A, Warfield A, et al. Pacman: Coordinated memory caching for parallel jobs. Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12), 2012. 267–280.

[29] Yahoo! webscope dataset. https://goo.gl/6CZZCF.

[30] Kandula S, Sengupta S, Greenberg A, et al. The nature of data center traffic: measurements & analysis. Proc. ACM IMC, 2009.

[31] Chowdhury M, Kandula S, Stoica I. Leveraging endpoint flexibility in data-intensive clusters. Proc. ACM SIGCOMM, 2013.

[32] Greenberg A, Hamilton J R, Jain N, et al. VL2: a scalable and flexible data center network. Proc. ACM SIGCOMM, 2009.

[33] Presto. https://prestodb.github.io/.

[34] Power R, Li J. Piccolo: Building fast, distributed programs with partitioned tables. OSDI, volume 10, 2010. 1–14.

[35] Memsql. https://www.memsql.com/.

[36] Yu Y, Huang R, Wang W, et al. Sp-cache: Load-balanced, redundancy-free cluster caching with selective partition. Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, Piscataway, NJ, USA: IEEE Press, 2018. 1:1–1:13.

[37] Hong Y J, Thottethodi M. Understanding and mitigating the impact of load imbalance in the memory caching tier. Proc. ACM SoCC, 2013.

[38] Huang Q, Gudmundsdottir H, Vigfusson Y, et al. Characterizing load imbalance in real-world networked caches. ACM HotNets, 2014.

[39] Huang C, Simitci H, Xu Y, et al. Erasure coding in windows azure storage. Proc. USENIX ATC, 2012.

[40] Sathiamoorthy M, Asteris M, Papailiopoulos D, et al. Xoring elephants: Novel erasure codes for big data. Proc. VLDB Endowment, volume 6, 2013. 325–336.

[41] apache parquet. `https://parquet.apache.org/`, 2019.

[42] Armbrust M, Xin R S, Lian C, et al. Spark sql: Relational data processing in spark. Proceedings of the 2015 ACM SIGMOD international conference on management of data. ACM, 2015. 1383–1394.