# 分布式系统的三种优化

陈倩[1)]

1) 华中科技大学计算机科学与技术学院

**摘 要** 本文从分布式系统的历史与发展着手，介绍一些经典的分布式系统以及近三年来分布式系统领域的一些新的研究。本文从分布式系统领域具有划时代意义的系统——GFS[1]开始，介绍一些经典的分布式系统，包括 HDFS[2]、Bigtable[3]、Haystack[4]等。然后，本文选取了三个近三年的研究论文进行分析：一个是 Facebook 公司的 Tectonic[5]文件系统，它是一个 EB 级别的大规模文件存储系统，Tectonic 的出现是为了将公司中分布式存储的一些通用问题集中在一个系统中解决，降低研发成本。Tectonic 将以前使用特定于服务而设计的系统的大型租户整合到通用的多租户文件系统中，与单独使用特定于服务而设计的系统相比，Tectonic 具有于他们相当的性能，而且，Tectonic 具有更高的资源利用率、更简单的操作复杂性、更简单的服务和更好的伸缩性。第二个是 INSTalytics[6]，INSTalytics 被用于在大型数据中心进行高效的大数据分析，它使数据能够以相同的存储成本同时在四个不同的维度上进行分区，从而使更多的查询在没有网络洗牌的情况下能够受益于分区过滤和连接。第三个是 Hu[7]等提出的使用结合奇偶校验局部性和拓扑局部性来解决宽条带修复问题的方法，与现有技术相比，用这种方法可以将单个块修复时间减少 90.5%。

关键词 分布式存储系统；Tectonic；INSTalytics；擦除编码；云存储

## Three Optimizations of Distributed Systems

CHEN Qian[1)]

[1)]Department of Computer Science and Technology, Huazhong University of Science and Technology, WuHan, China

**Abstract** Starting from the history and development of distributed systems, this article introduces some classic distributed systems and some new research in the field of distributed systems in the past three years. This article starts with GFS [1], an epoch-making system in the field of distributed systems, and introduces some classic distributed systems, including HDFS [2], Bigtable [3], Haystack [4], etc. Then, this article selects three research papers in the past three years for analysis: one is Facebook's Tectonic [5] file system, which is an Exabytes large-scale file storage system. The emergence of Tectonic is to distribute the company. Some common storage problems are solved in one system to reduce R&D costs. Tectonic integrates large tenants who previously used a service-specific design system into a common multi-tenant file system. Compared with the service-specific design system alone, Tectonic has comparable performance to them, and Tectonic has better performance. High resource utilization, simpler operation complexity, simpler service and better scalability. The second is INSTalytics[6]. INSTalytics is used for efficient big data analysis in large data centers. It enables data to be partitioned on four different dimensions at the same time at the same storage cost, thereby enabling more queries In the absence of network shuffling, it can benefit from partition filtering and connection. The third is the method proposed by Hu [7] to use the combination of parity check locality and topological locality to solve the problem of wide stripe repair. Compared with the prior art, this method can reduce the repair time of a single block. 90.5%.

**Key words** Distributed storage system; Tectonic; INSTalytics; Erasure coding; Cloud storage

# 1 引言

随着计算机系统规模变得越来越大，将所有的业务单元集中部署在一个或若干个大型机上的体系结构，已经越来越不能满足当今计算机系统，尤其是大型互联网系统的快速发展，各种灵活多变的系统架构模型层出不穷。分布式的处理方式越来越受到业界的青睐——计算机系统正在经历一场前所未有的从集中式向分布式架构的变革。

所谓的集中式系统就是指由一台或多台主计算机组成中心节点，数据集中存储于这个中心节点中，并且整个系统的所有业务单元都集中部署在这个中心节点上，系统的所有功能均由其集中处理。

集中式系统的最大的特点就是部署结构非常简单，底层一般采用从 IBM、HP 等厂商购买到的昂贵的大型主机。因此无需考虑如何对服务进行多节点的部署，也就不用考虑各节点之间的分布式协作问题。但是，由于采用单机部署，很可能带来系统大而复杂、难于维护、发生单点故障（单个点发生故障的时候会波及到整个系统或者网络，从而导致整个系统或者网络的瘫痪）、扩展性差等问题。

分布式系统是一个硬件或软件组件分布在不同的网络计算机上，彼此之间仅仅通过消息传递进行通信和协调的系统。简单来说就是一群独立计算机集合共同对外提供服务，但是对于系统的用户来说，就像是一台计算机在提供服务一样。分布式意味着可以采用更多的普通计算机（相对于昂贵的大型机）组成分布式集群对外提供服务。计算机越多，CPU、内存、存储资源等也就越多，能够处理的并发访问量也就越大。

分布式系统是由一组通过网络进行通信、为了完成共同的任务而协调工作的计算机节点组成的系统。分布式系统的出现是为了用廉价的、普通的机器完成单个计算机无法完成的计算、存储任务。其目的是利用更多的机器，处理更多的数据。

分布式系统分为分布式计算与分布式存储。一个标准的分布式系统应该具有以下几个主要特征：分布性、对等性、并发性、缺乏全局时钟以及故障总是发生。分布式系统还面临着通信异常、节点故障等问题。

随着信息化程度的不断提高，全球数据日益膨胀。面对当前 PB 级的海量数据存储需求，传统的存储系统在容量和性能的扩展上存在瓶颈。云存储以其扩展性强、性价比高、容错性好等优势得到了业界的广泛认同。由于其前瞻性，众多企业都将其作为进军云计算的第一步。分布式系统和分布式块存储作为云存储中重要的技术，成为奠定云存储发展的重要基石。

同时，由于全球数据量的日益增加，各种应用对数据读取速度的要求越来越高，以及企业出于对研发成本的考虑，对分布式系统进行各方面的优化是必要的。

# 2 发展背景

本节主要介绍几个经典的分布式系统。包括 Google 的三驾马车：GFS(Google File System)[1]、MapReduce[8] 和 Bigtable[3] 以及 HDFS(Hadoop Distributed File System)[2]。

第一个是 Google 的分布式文件系统 GFS[1]，GFS 被人誉为划时代的设计，打开了大数据时代。并不是因为它采用了多么令人惊讶的新技术，而在于它采用了廉价的商用计算机集群构建分布式文件系统，在降低成本的同时能够高效可靠地存储大规模的数据，并且将这一技术免费公开了出来。

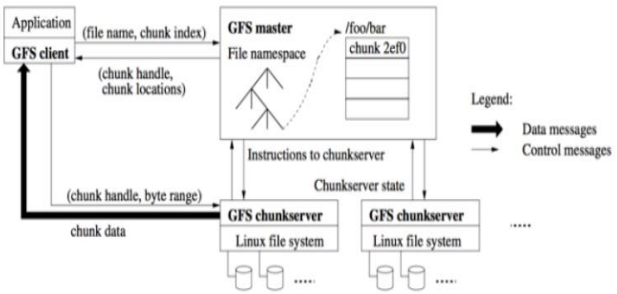


图 1 GFS 架构

GFS 系统中主要存储文件元数据和文件数据，文件元数据中存储着文件系统命名空间、文件名与文件存储位置信息之间的映射关系等。GFS 将文件切割为固定大小的 Chunk 然后分散存储，经典的 Chunk 大小设置为 64MB，Chunk 是数据复制的基本单位，每个 Chunk 默认复制三份分散存储（可能存储在不同的磁盘、机器、机架甚至数据中心），这为 GFS 提供了很好的容错性。

GFS 包含两类节点，一类是 Master 节点，存储元数据，一类是 Slave 节点，存储 Chunk。Master 节点是 GFS 的中心节点，所有的客户端都需要先访

问元数据节点以获取 Chunk 的具体位置，然后再与最近的存储有所需 Chunk 的 Slave 节点通信。

另外，GFS 还设计了一种租约机制来保持 Chunk 副本之间的一致性。租约（Lease）是由 GFS 中心节点 Master 分配给 chunk 的某个副本的锁。持有租约的副本方可处理客户端的更新请求，客户端更新数据前会从 Master 获取该 chunk 持有租约的副本并向该副本发送更新请求。GFS 在 chunk 多副本之间选择出一个主副本，由主副本来协调客户端的写入，保证多副本之间维持一个全局统一的更新顺序。

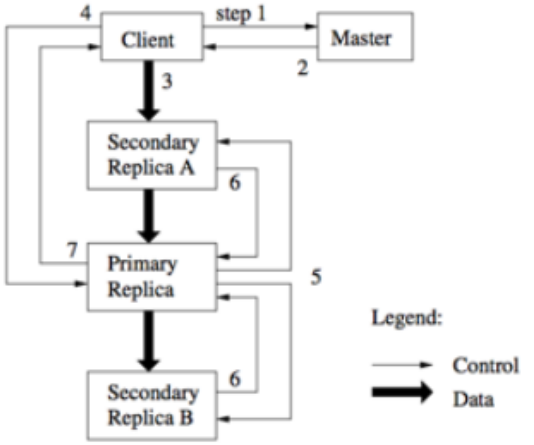


图 2 GFS 数据写入

第二个是 MapReduce[8]，MapReduce 是一种分布式计算框架，以一种可靠的、有容错能力的方式并行地处理 TB 级别的海量数据集。主要用于搜索领域，解决海量数据的计算问题。MapReduce 由两个阶段组成：Map 和 Reduce，用户只需实现 map() 和 reduce()两个函数，即可实现分布式计算。这大大简化了分布式计算的复杂度，使得任何需要的程序员都可以编写分布式计算程序。

第三个是 Bigtable[3]，Bigtable 是 Google 在 NoSQL 领域的分布式表格系统。由于 Google 需要存储的数据种类繁多且每秒需要处理海量的服务请求，商用数据库无法满足需求，因此 Bigtable 应运而生。Bigtable 是一个分布式的、多维的映射表。表中的数据通过一个行关键字（Row Key）、一个列关键字（Column Key）以及一个时间戳（Time Stamp）进行索引。在 Bigtable 中一共有三级索引：行关键字为第一级索引，列关键字为第二级索引，时间戳为第三级索引。

下图中的列族 anchor 保存了该网页的引用站点（比如引用了 CNN 主页的站点），qualifier 是引用站点的名称，而数据是链接文本；列族 contents 保存的是网页的内容，这个列族只有一个空列“contents:”，“contents”列下保存了网页的三个版本，我们可以用（“com.cnn.www”, “contents:”, t5) 来找到 CNN 主页在 t5 时刻的内容。

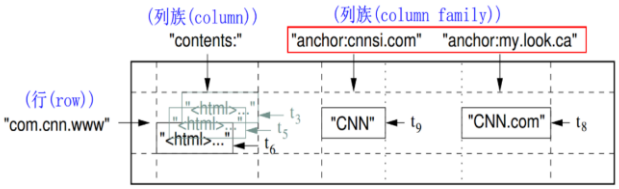


图 3 Bigtable 存储结构举例

Bigtable 系统的内部采用的是一种类似 B+树的三层查询体系。首先是第一层 Chubby file，这一层是一个 Chubby 文件，它保存着 root tablet 的位置。这个 Chubby 文件属于 Chubby 服务的一部分，一旦 Chubby 不可用，就意味着丢失了 root tablet 的位置，整个 Bigtable 也就不可用了。第二层是 root tablet。root tablet 其实是元数据表的第一个分片，它保存着元数据表其它子表的位置。root tablet 很特别，为了保证树的深度不变，root tablet 从不分裂。第三层是其它的元数据子表，它们和 root tablet 一起组成完整的元数据表。每个元数据片都包含了许多用户子表的位置信息。

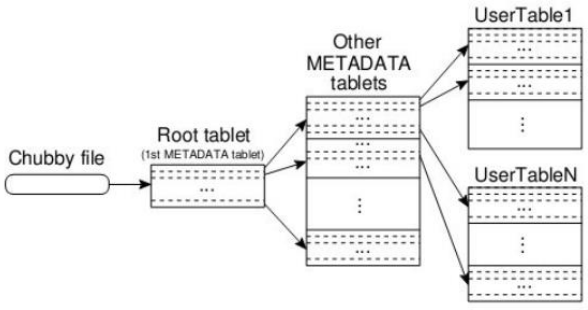


图 4 BigTable 的三层寻址

最后是 HDFS[2]，它是 GFS 的具体实现，是 Hadoop 项目的核心子项目，是分布式计算中数据存储管理的基础，是基于流数据模式访问和处理超大文件的需求而开发的。HDFS 可以运行于廉价的商用服务器上。它所具有的高容错、高可靠性、高可

扩展性、高获得性、高吞吐率等特征为海量数据提供了具有一定容错性的存储，为超大数据集的应用处理带来了很多便利。

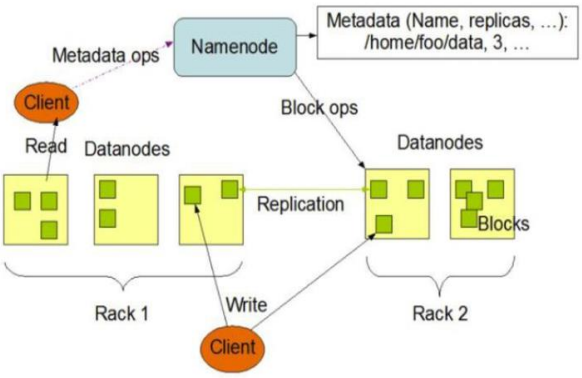


图 5 HDFS 的架构

其中，Namenode 即为 GFS 中的 Master，Datanodes 即为 GFS 中的 Slaves。

## 3 研究进展

### 3.1 Facebook的文件系统：Tectonic[5]

Tectonic 是 Facebook 公司现在正在使用的底层分布式文件系统。Tectonic 这个项目起始时间大约在 2012 年，今年，Facebook 在 FAST21 公开介绍了 Tectonic 的架构以及他们如何实现真正的大规模存储（即 EB 级存储）、如何为多租户提供性能之间的隔离、如何实现面向不同租户的特定优化能力等问题。

3.1.1 Tectonic 之前的系统

Tectonic 的出现是为了将 Facebook 公司中分布式文件存储的一些通用问题集中在一个系统中解决，以此降低研发成本。一般拥有分布式文件系统的大公司往往都是将一个分布式文件系统集群服务于一类业务，比如某个集群只给对象存储用，或者只给块存储用，但是在 Tectonic 系统上，为了减少运维成本、增加集群的资源利用率，在同一个大的 Tectonic 集群上同时支持了诸多业务，在论文[5]中主要介绍了数据仓库和对象存储。对象存储，也叫做基于对象的存储，是用来描述解决和处理离散单元的方法的通用术语，这些离散单元被称作为对象。就像文件一样，对象包含数据，但是和文件不同的是，对象在一个层结构中不会再有层级结构。每个对象都在一个被称作存储池的扁平地址空间的同一级别里，一个对象不会属于另一个对象的下一级。数据仓库是为了企业所有级别的决策制定计划过程，提供所有数据类型的战略集合。它出于分析性报告和决策支持的目的而创建。为需要业务智能的企业 ,为需要指导业务流程改进、监视时间，成本，质量以及控制等。

在设计 Tectonic 之前，Facebook 采用将一个分布式文件系统集群服务于一类业务的模式。比如，在 Tectonic 之前，Facebook 有三大在离线存储系统 Haystack[4]、F4[9]和 HDFS[2]。其中 Haystack 系统存储热对象,，F4 系统存储温对象、数据仓库由多个 HDFS 集群构建。所谓热对象就是高访问频率的对象，热对象老化、访问频率降低后就变成温对象。

Haystack 用来存储请求具有长尾效应的海量小文件（图片）。在传统的文件系统中，元数据存在磁盘上，每访问一个文件就要去磁盘上查元数据，然后才能知道文件的具体位置，这导致元数据查找成为瓶颈。另外，传统的文件系统设计中，图片被放在 CDN（Content Delivery Network，内容分发网络）上，浏览器通过和 CDN 交互拿到图片，对于热点图片，可以直接从 CDN 缓存中返回，而不在 CDN 缓存中的图片则需要去存储系统中查询。因此，CDN 只能解决热点图片的请求。Haystack 通过压缩元数据让所有的元数据被加载到内存，以此减少磁盘操作，这样，上述的两个传统文件系统的问题就都解决了。Haystack 通过将很多张图片存在同一个文件中，将小文件变成大文件来进行元数据压缩。

Facebook 的照片、视频和其他二进制大对象(BLOB)需要可靠存储和快速访问的语料库非常庞大，并且还在不断增长。随着 BLOB 占用空间的增加，将它们存储在 Haystack 中变得越来越低效。为了提高存储效率，Facebook 设计了一个名为 F4 的新系统，用来专门存储温对象。F4 中存储的温对象都是由热对象冷却下来后形成的，这些对象在处于热对象阶段时存储在 Haystack 中，所以 F4 的对象都是由 Haystack 转移过来的，因此，它们不需要插入等操作，F4 只支持读和删除操作。Haystack 中存储热对象，因此使用 3 备份模式，这样设计的好处在于性能良好，缺点是成本高。F4 中存储温对象，访问的频繁度比热对象低，因此 F4 使用 RS 编码，在实现高可靠的同时将副本数量降低。

Facebook 在 ICDE2010 上介绍了基于 Hadoop 的数据仓库 Hlive，Hive 存储海量数据在 Hadoop 系统中，提供了一套类数据库的数据存储和处理机

制。它采用类 SQL 语言对数据进行自动化管理和处理，经过语句解析和转换，最终生成基于 Hadoop 的 MapReduce 任务，通过执行这些任务完成数据处理。Hadoop 系统中常用的文件存储格式有支持文本的 TextFile 和支持二进制的 SequenceFile 等，它们都属于行存储方式。Facebook 发表的文章[10]，介绍了一种高效的数据存储结构——RCFile（Record Columnar File），其应用于 Facebook 的数据仓库 Hive 中。对比传统数据库中三种常见的数据存储结构，行存储、列存储和混合存储，RCFile 更有效地满足了基于 MapReduce 的数据仓库的 4 个关键需求，即快速的数据加载、快速的查询处理、高存储空间利用率和对高动态工作负载模式的强适应性。

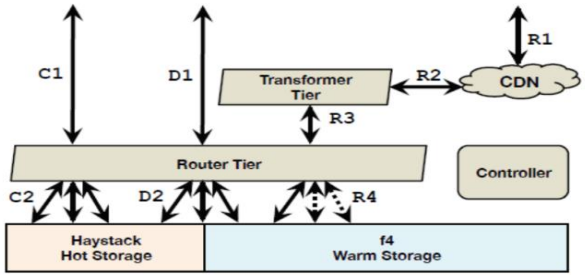


图 6 加入 F4 后 Facebook 的对象存储架构

### 3.1.2 Tectonic 架构

从模块的角度看，一个 Tectonic 集群由四大模块组成：元数据服务 Metadata Store、后台服务 Background Service、客户端库 Client Library 和数据存储服务 Chunk Store。

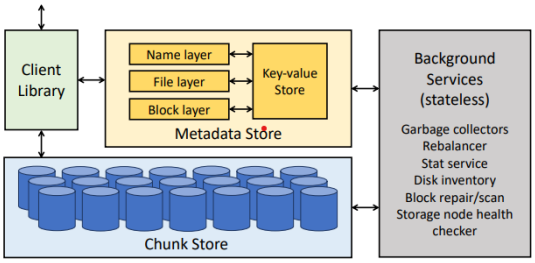


图 7 Tectonic 架构

Tectonic 文件系统的所有元数据都通过 Metadata Store 维护，元数据被分为了三层，第一层叫 Name layer，主要存储目录和目录元素（子目录或文件）之间的映射关系；第二层叫 File layer，主要存储文件和 Block 之间的映射关系；第三层叫 Block layer，主要存储 Block 和 Chunk 之间的映射关系，Block layer 还维护 Disk 和 Block 的反向映射，便于在盘或机器下线时快速定位它上面的 block 信息，从而做快速数据修复。这三层元数据都以 K-V 的形式存储到 key-value store 中，这个 key-value store 是一个理论上可以无限扩展的分布式键值系统。Tectonic 使用了 ZippyDB[11]作为元数据的 key-value store。用这种方法，可以实现一个可真正意义上水平扩展的元数据服务，从而达到 EB 级的存储。

| Layer | Key | Value | Sharded by | Mapping |
|---|---|---|---|---|
| Name | (dir_id, *subdirname*) | subdir_info, subdir_id | dir_id | dir → list of subdirs (expanded) |
| | (dir_id, *filename*) | file_info, file_id | dir_id | dir → list of files (expanded) |
| File | (file_id, blk_id) | blk_info | file_id | file → list of blocks (expanded) |
| Block | blk_id | list<disk_id> | blk_id | block → list of disks (i.e., chunks) |
| | (disk_id, blk_id) | chunk_info | blk_id | disk → list of blocks (expanded) |

**Table 1: Tectonic's layered metadata schema. *dirname* and *filename* are application-exposed strings. `dir_id`, `file_id`, and `block_id` are internal object references. Most mappings are expanded for efficient updating.**

图 8 Metadata 的三层划分

将目录树以键值对的形式存储到分布式 key-value 系统中是一个比较朴素的分布式目录树的技术方案，Tectonic 有两个较为显著的特点：（1）mapping 关系是 expanded 的。它的具体含义是：如果 value 是一个列表，并不是将整个列表作为 value，而是将列表中的每个元素作为一个单独的 key，同时将真正的 key 作为当前元素的 key 的前缀。这样做的好处是如果 key 对应的内容被修改时，并不需要将整个 value 都读出来，例如一个文件系统中，某个目录下面可能会有百万个文件，如果通过非 expanded 的方式，那么需要将巨大的一个 value 读出来、修改、写入，这会显著降低文件系统的性能。在 expanded 模式下，只需要更改某个特定的键值对即可。（2）分布式系统做 Shard（Shard 是 ZippyDB 操作的基本单位）操作的基本单位并不是整个 key，而是 key 的部分前缀。例如 Name layer 中，并不是按照<dir_id, subdirname>整个 key 做 Shard，而是根据 dir_id 做 Shard，这样一个目录下的所有元素都位于同一个 Shard 下面，便于操作。相应地，同一个文件下面所有的 Block 的元数据也都位于同一个 Shard 下面。

热点规避是设计一款分布式文件系统必须要面临的问题之一，在 Tectonic 中，大约有 2/3 的元数据操作都是针对文件的，Block layer 使用 Hash-partition 的方式来划分 Shard 可以使得热点均

匀分散在 ZippyDB 的所有 Shards 中。另外，Tectonic 中的目录和文件都可以被封装。一个目录被封装是指它的当前孩子元素列表以及元数据无法被修改，并不影响它的孙子元素；另外文件也可以被封装，封装后无法再追加写。Tectonic 可以将封装状态的目录和文件做一层缓存服务，这样不仅可以做热点规避，也可以提高读性能和元数据服务失效后的可用性。

3.1.3 Tectonic 对多租户的支持

Tectonic 将资源分为两类：固定资源和临时资源。固定资源更新很慢，且无法在租户之间共享，不能同时分配给多个租户。比如磁盘容量，租户的配额是预分配好的，可以在不影响服务的情况下手动更新。临时资源需求经常变动，分配也会实时变化。磁盘的 IOPS 和 Metadata 查询能力就是两个临时资源。临时资源需要细粒度的自动化动态管理，来做到资源的共享、隔离和高效利用。

Tectonic 通过以下手段完成对临时资源的公平的共享、隔离和利用：

1) Tectonic 在租户内使用了应用分组的模式来管理临时资源，每个分组称为一个 TrafficGroup，一个 TrafficGroup 中的应用拥有类似的流量模式和延迟需求。每个租户可以拥有若干个 TrafficGroup，这样的划分解决了资源管理空间复杂度的问题，否则如果按照应用划分去做资源管理会带来极大的额外计算成本。

2) 每个 TrafficGroup 通过 TrafficClass 来管理资源分配，分为 3 类， Gold，Silver 和 Bronze，分别对应于延迟敏感性，普通和后台应用三大类应用。稀有的资源通过 TrafficClass 来进行优先级控制和分配。一个 TrafficGroup 富裕的资源会优先在租户内分配给其它 TrafficGroup 共享，一个租户富裕的资源会被分配给其它租户的 TrafficGroup 共享。

3) 为了达到对临时资源共享和隔离的弹性管理目标，Client Library 有一个限速算法，它使用近实时的分布式计数器来跟踪每个租户和 TrafficGroup 在最近时间窗口内的资源需求，采用的是修改过的漏桶算法。到来的请求会增加漏桶的需求计数。 Client Library 会基于 TrafficClass 依次检查本 TrafficGroup，本租户，乃至其它租户的空闲资源。

4) 存储节点采用 3 种策略来优化 Gold 组的请求延时：一是使用 WRR(weighted round-robin)来保证在低 TrafficClass 请求有足够时间的情况下，优先让高 TrafficClass 的请求先执行；二是在每个磁盘上，限制来自非 Gold 的 TrafficClass 的请求数。避免磁盘 IO 被低优先级流量占满；三是磁盘本身也可能会对 IO 做排序，比如优先执行一个非 Gold 的请求。为了避免这种情况，当 Gold 请求被挂起在磁盘上超过一定时间时，Tectonic 会停止下发非 Gold 的请求。

Tectonic 还支持租户特定场景的优化，论文主要讨论了数据仓库写优化和对象存储优化。Tectonic 引入了 Full-block 的 RS 写入接口来提高数据仓库写优化，另外，Tectonic 使用仲裁写入来解决写长尾延迟问题。对于对象存储，Tectonic 使用 Quorum 写机制来达到低延时。

## 3.2 INSTalytics[6]

3.2.1 INSTalytics 介绍

INSTalytics 是在分布式文件系统和计算层协调工作的堆栈，用于在大型数据中心进行高效的大数据分析。INSTalytics 放大了数据分析系统中数据分区的优点，与传统的一维分区不同，INSTalytics 使数据能够以相同的存储成本同时在四个不同的维度上进行分区，从而使更多的查询在无需网络洗牌的条件下能够受益于分区过滤和联接。

为了在数据中心出现故障的情况下确保高可用性，分布式文件系统通常会设计一些冗余机制来确保能从其他节点恢复故障节点的数据。一种常见的冗余机制是在不同的服务器上保留多个（通常为三个）数据副本。虽然冗余提高了可用性，但它带来了巨大的存储和写放大开销，通常被视为可用性的成本。这样的冗余机制给大数据分析处理带来了日渐严重的工作负载。数据分析系统为有效执行查询而采用的一种流行技术是数据分区，其中输入文件在特定列上进行排序或分区，从而使具有特定列值范围的记录在文件中进行物理聚集。使用分区布局，只对特定范围的列值（例如 1%）感兴趣的查询可以只扫描文件的相关分区，而不是扫描整个文件。

INSTalytics 允许数据沿四个不同的维度同时进行分区，而不是现在的单一维度，从而允许大部分查询实现高效分区过滤和高效连接，而无需网络洗牌。在 INSTalytics 中实现此类改进的关键方法是使分布式文件系统具有计算感知能力；INSTalytics 通过定制分布式文件系统的三向复制，实现了这种异构分区。这种逻辑复制的明显挑战是确保与物理复制相同的可用性和恢复性能；简单的布局需要以其

他分区顺序扫描整个文件，以恢复单个失败的块。INSTalytics 使用一种基于超级数据块和数据块内循环存储桶的新颖布局技术，以实现与物理复制一样高效的恢复。它还确保了与实际故障场景下的物理复制相同的可用性和故障隔离保证。INSTalytics 中的布局技术还支持除了三个逻辑副本之外的第四个分区维度。

### 3.2.2 INSTalytics 实现

INSTalytics 在微软内部的分析堆栈的代码库中实现。分析堆栈中的文件系统架构与标准的 appendonly 分布式文件系统非常相似。以 GFS 为例，其中 Master 与 GFS 的 Master 功能类似，Storage Nodes 与 GFS 的 Slaves 类似。

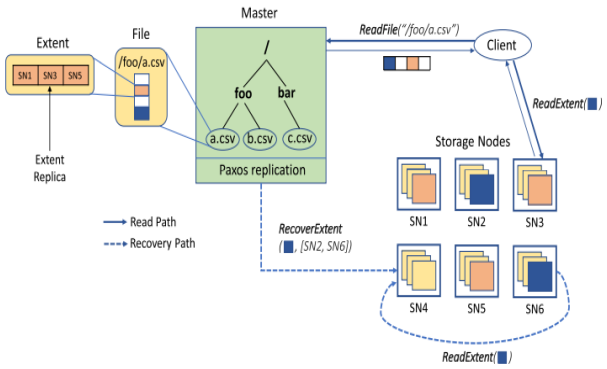


图 9 文件系统架构

INSTalytics 在堆栈中增加了一些全新的 API，作为前提，需要对 Master 和 SN 中的元数据进行更改。这里对 INSTalytics 的三个关键路径做一些简要介绍：

（1）创建路径：这是将逻辑复制文件引入系统的路径。操作流程如图 10 所示。在主机上添加了一个新的 API LogicallyReplicate，使用户能够将现有的物理复制文件转换为逻辑复制文件。通过这种方式，用户可以选择需要逻辑复制的文件子集，并且这些文件可以与系统中的其他物理复制文件共存。LogicallyReplicate API 将目标文件的 URL 和其他与逻辑复制相关的参数（如超级扩展数据块大小）以及需要使用的适配器（取决于文件格式）作为参数。

（2）恢复路径：逻辑复制文件中的扩展数据块的恢复路径中的高级操作流（如图 11 所示）与物理复制文件中的扩展数据块非常相似：主机向存储节点发送请求以触发单个扩展数据块的恢复，存储节点负责生成所需数据块的本地副本所需的处理。这满足了主节点和存储节点之间没有额外交互的约束。目前，主机中触发扩展数据块恢复的条件是：扩展数据块的正常副本少于三个。此条件对于逻辑复制文件中的扩展数据块是不够的。例如，这样的扩展数据块可以在分区维度 0 中有两个健康的副本，在分区维度 1 中有一个，在分区维度 2 中有 0 个。即使它有三个健康副本，我们仍然需要触发恢复，因为其中一个分区维度有 0 个健康副本。从本质上讲，恢复触发条件需要知道分区维度。对于逻辑复制文件中的数据块，我们将条件更改为：至少有一个分区维度没有正常的副本。一旦主机决定恢复一个扩展数据块，它将选择一个目标存储节点来容纳该扩展数据块的新副本，并向其发送一个 CreateLogicalExtent 请求，其中 mode 设置为 mode_recover。在做出此选择时，它强制执行超级数据块级故障隔离。与物理数据块恢复不同，在物理数据块恢复中，请求中只传递要恢复的数据块的现有副本列表，这里我们传递有关该数据块所属的相应超级数据块中所有数据块副本的信息。

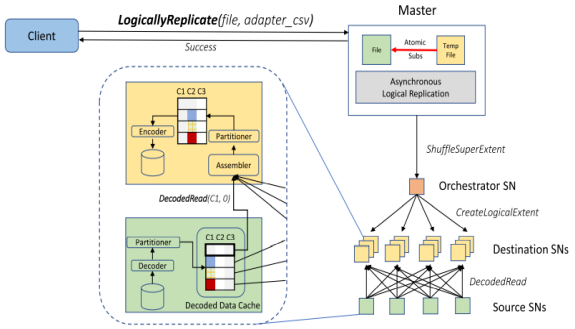


图 10 创建路径操作流程

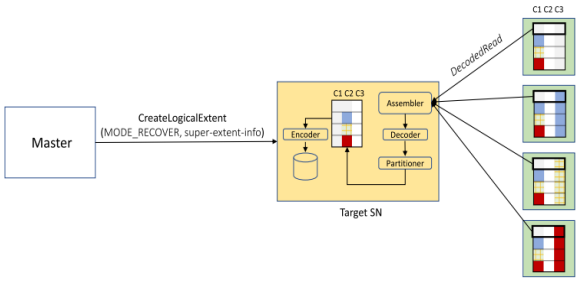


图 11 恢复路径操作流程

（3）读取路径：读取路径包括范围读取、过滤读取和切片读取。范围读取：由于存储节点上基于适配器的设计，客户端可以通过正常的基于块的

接口读取数据块；过滤读取：为了增加布局带来的好处，在计算层中运行的过滤查询需要能够根据过滤条件仅读取文件中数据块的特定子集。为此，INSTalytics 在存储层提供过滤读取功能；端到端读取路径包括对文件进行过滤读取，以仅获取所需扩展数据块子集的信息，然后是一系列 ReadExtent 调用。这类似于物理复制文件的读取路径；切片读取：INSTalytics 在存储节点上引入了一个新的 API。此 API 所需的处理类似于创建路径中源存储节点上的解码处理。

### 3.3 分布式存储系统中的宽条带擦除编码

擦除编码存储数据条带和奇偶校验块，是分布式存储系统的一种低成本冗余机制。最近有研究人员提出使用宽条带来抑制条带中奇偶校验块的比例，以节省存储空间。然而，宽条纹会加重修复惩罚，而现有的高效修复擦除编码方法无法有效地处理宽条纹。论文[7]提出了一种组合局部性，这是第一种通过结合奇偶校验局部性和拓扑局部性来系统地解决宽条带修复问题的机制。[7]在 Amazon EC2 上的实验表明，与基于局部性的现有技术相比，组合局部性将单个块修复时间减少了 90.5%，冗余度仅为 1.063×。

擦除编码是一种已建立的低成本冗余机制，用于在现代分布式存储系统中防止数据存储故障，特别是 ReedSolomon（RS）码在当今的擦除编码部署中被广泛采用。在擦除编码能有效降低存储冗余的基础上，宽条带在存储领域引起了研究人员的注意，它提供了一个机会来实现接近最优的冗余，并尽可能最大限度地节省存储。虽然宽条带极大地节省了存储空间，但随着带宽修复，进一步加重了修复惩罚。

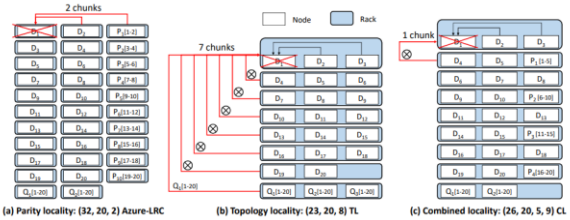


图 12 三个基于位置的方案的示例

为了实现组合局部性的目标，从图 12 中可以观察到，组合局部性通过下载 r-1 个数据块加上一个本地奇偶校验块（即，修复带宽为 r 个块）去修复一个数据块。由于组合局部性将一些 r 块放置在相同的机架中，因此可以对每个机架中的块应用局部修复，从而减少跨机架修复带宽。直观地说，如果驻留在每个机架中的条带的最大块数 c 增加，本地修复可以包括更多块，从而进一步减少跨机架修复带宽。因此，我们的目标是找到最大可能的 c≤f (f 为条带的可容忍节点故障数)。如果 c=f，则可以最小化跨机架修复带宽。因此，构造组合局部性机制是为了确保 c=f。然而，(n,k,r) LRC 有不同的构造，提供不同级别的容错性。因此，[7]选择具有最高容错能力的 LRC 结构。

[7]是第一篇系统解决宽条带修复问题的论文，并且，[7]中设计的 ECWide 通过实现局部组合性，解决了单块修复和全节点修复两种修复类型，在超低存储冗余下减轻了跨机架修复带宽。[7]中还设计了一种高效的编码方案，允许宽条带的奇偶校验块在多个节点之间并行编码，以及一种机架内奇偶校验更新方案，允许奇偶校验块在机架内本地更新，以减少跨机架传输。

## 4 总结与展望

本文从几种经典的分布式文件系统开始，介绍了分布式系统的发展历史。另外，本文从近三年的分布式系统顶级会议 FAST 中挑选了三篇文章，进行了一个大致的解读。Facebook 公司的 Tectonic[5]文件系统是一个 EB 级别的大规模文件存储系统，Tectonic 的出现是为了将公司中分布式存储的一些通用问题集中在一个系统中解决，降低研发成本。Tectonic 将以前使用特定于服务而设计的系统的大型租户整合到通用的多租户文件系统中，与单独使用特定于服务而设计的系统相比，Tectonic 具有于他们相当的性能，而且，Tectonic 具有更高的资源利用率、更简单的操作复杂性、更简单的服务和更好的伸缩性；INSTalytics 被用于在大型数据中心进行高效的大数据分析，它使数据能够以相同的存储成本同时在四个不同的维度上进行分区，从而使更多的查询在没有网络洗牌的情况下能够受益于分区过滤和连接；Hu[7]等提出的使用结合奇偶校验局部性和拓扑局部性来解决宽条带修复问题的方法，与现有技术相比，用这种方法可以将单个块修复时间减少 90.5%。随着全球数据量的剧增，超级公司也越来越意识到存储形态多样化后，构建一个统一的底层文件系统的必要性。并且，如何高效的利用存储资源也越来越引起了企业和研究人员的注意。

# 参 考 文 献


[1] Ghemawat S, Gobioff H, Leung S T. The Google file system[J]. A Sigops Operating Systems Review, 2003, 37(5):29-43.

[2] HDFS erasure coding. https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HDFSErasureCoding.html, Retrieved in Jan 2021.

[3] Chang F， Dean J， Ghemawat S，et al. Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: a distributed storage system for structured data. In Proc. USENIX Symposium on Operating System[J]. 2010.

[4] Beaver D, Kumar S, Li H C. Finding a needle in haystack: Facebookars photo storage. 2010.

[5] Pan S, Stavrinos T, Zhang Y, et al. Facebook's Tectonic Filesystem: Efficiency from Exascale[C]//19th {USENIX} Conference on File and Storage Technologies ({FAST} 21). 2021: 217-231.

[6] Sivathanu M, Vuppalapati M, Gulavani B S, et al. Instalytics: Cluster filesystem co-design for big-data analytics[C]//17th {USENIX} Conference on File and Storage Technologies ({FAST} 19). 2019: 235-248.

[7] Hu Y, Cheng L, Yao Q, et al. Exploiting Combined Locality for Wide-Stripe Erasure Coding in Distributed Storage[C]//19th {USENIX} Conference on File and Storage Technologies ({FAST} 21). 2021: 233-248.

[8] Condie T, Conway N, Alvaro P, et al. MapReduce online[C]//Nsdi. 2010, 10(4): 20.

[9] Muralidhar S, Lloyd W, Roy S, et al. f4: Facebook's Warm {BLOB} Storage System[C]//11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14). 2014: 383-398.

[10] He Y, Lee R, Huai Y, et al. RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems[C]//2011 IEEE 27th International Conference on Data Engineering. IEEE, 2011: 1199-1208.

[11] Cao Z, Dong S, Vemuri S, et al. Characterizing, modeling, and benchmarking rocksdb key-value workloads at facebook[C]//18th {USENIX} Conference on File and Storage Technologies ({FAST} 20). 2020: 209-223.