开源软件中的大数据管理技术

江天,乔嘉林,黄向东,王建民

清华大学软件学院,大数据研究中心;大数据系统软件国家工程实验室,北京 100084

摘要　随着谷歌文件系统和宽表结构为代表的技术打破依赖关系数据库管理海量数据的限制,以Apache Hadoop为代表的开源大数据管理系统软件新技术与系统不断涌现,并快速成熟应用。针对Apache开源社区中面向在线事务处理和在线分析处理场景的大数据管理软件,介绍了大数据管理中的数据存储、数据分区、副本机制、分布式协议等,并比较分析了分布式文件系统、键值库、时序数据库等典型分布式数据管理系统的优缺点。

关键词　大数据管理;开源软件;分布式系统

近10年来,大数据管理技术的发展日新月异,传统以本地文件系统和关系数据库进行数据管理的技术方案在性能和性价比上受到越来越严峻的挑战。2003年,谷歌公司发表谷歌文件系统[1],提出了使用廉价服务器在不可靠的硬件上构建高可用的分布式文件系统以实现海量数据管理的思想。后又于2008年发表了宽表系统[2]的技术思路。以此为代表,分布式、非关系模型的数据管理技术逐渐被广泛接受,引领了近10年来大数据管理技术的变革浪潮。2009年,非结构化查询语言NoSQL提出[3],进一步推动了基于非关系模型的、面向特定应用需求的数据管理技术的发展。

每当一个新技术被提出,总会快速地涌现出若干开源大数据管理软件将其进行实现,从而促使这些新技术快速地在世界范围内得到普及应用与验证,并进一步地进行革新。例如,Hadoop、Spark等一批开源软件已经被业界广泛使用,许多新技术、新特性都在这些软件的新版本中不断地被提出。在这些开源软件中,以Apache软件基金会旗下的大数据管理软件生态最为活跃。截至2019年11月,Apache基金会的数据管理与处理相关的软件如表1[4]所示。

通过对Apache开源社区中的开源大数据管理软件进行总结,以Hadoop、Parquet、Cassandra、IoTDB等系统为例,本文介绍存储系统中的LSM结构、列式存储,副本机制中的副本一致性、纠删码技术,数据分区技术与分布式共识算法等大数据管理技术与未来发展动向。

收稿日期:2019-11-08; 修回日期:2020-01-31

基金项目:国家重点研发计划项目(2016YFB0501504);国家自然科学基金项目(U1509213,61802224)

作者简介:江天,博士研究生,研究方向为大数据系统,电子信箱:jiangtia18@mails.tsinghua.edu.cn;黄向东,助理研究员,研究方向为分布式系统及时序数据库,电子信箱: huangxdong@tsinghua.edu.cn

引用格式:江天, 乔嘉林, 黄向东, 等. 开源软件中的大数据管理技术[J]. 科技导报, 2020, 38(3): 103-114; doi: 10.3981/j.issn.1000-7857.2020.03.007

表1 Apache软件基金会中的部分大数据管理和处理软件

序号	项目类型	项目名称	项目概述
1	数据库	Accumulo	键值数据库
2		Cassandra	键值数据库
3		HBase	键值数据库
4		CouchDB	文档数据库
5		IoTDB	时序数据库
6		Ignite	内存数据库
7	文件系统	Hadoop	分布式文件系统
8	文件格式	Avro	一种高效的文件存储格式
9		CarbonData	一种高效的文件存储格式
10		ORC	列式数据文件格式
11		Parquet	列式数据文件格式
12	查询引擎	Pig	基于MapReduce的数据查询引擎
13		Hive	基于MapReduce的数据查询引擎
14		Drill	分布式MPP架构的查询引擎
15		Kudu	基于Hadoop的列式存储引擎
16		Phoenix	基于HBase的针对Hadoop数据的OLTP和SQL查询分析
17		Trafodion	SQL-on-Hadoop
18		Lens	无缝集成Hadoop和传统数据仓库,支持OLAP
19		DataFu	孵化器项目。MapReduce库和Pig的UDF库
20		Tajo	基于Hadoop的数据仓库
21		Calcite	支持关系模型的数据查询引擎
22		MetaModel	多种数据库的统一连接器和查询API
23	其他数据管理相关系统	Lucene	全文检索系统
24		Lucy	全文检索系统
25		Helix	基于Zookeeper的集群管理框架。
26		Daffodi	孵化器项目。用于数据格式(JSON/XML等)的相互转换。
27		VXQuery	大量XML数据的查询引擎
28		Zeppelin	大数据交互式分析和可视化工具
29		Jackrabbit	Java内容仓库标准的实现,用于存储结构和非结构化数据
30		Gora	面向大数据的ORM库
31		Zookeeper	分布式协调器
32		BookKeeper	分布式日志服务,用于将单点服务扩展为分布式服务
33		Fluo	大规模数据集增量处理系统
34		Flume	收集分布式系统日志到集中节点
35		Sqoop	Hadoop和关系数据库的ETL工具

续表1　Apache软件基金会中的部分大数据管理和处理软件

序号	项目类型	项目名称	项目概述
36	数据处理框架	Tez	处理DAG图的通用应用框架
37		PredictionIO	机器学习服务
38		Samza	分布式流式处理框架
39		Spark	分布式计算引擎
40		Hama	基于BSP模型的分布式计算框架
41		Storm	分布式流式处理框架
42		Giraph	图数据处理系统
43		Flink	流式数据处理引擎
44		Edgent	孵化器项目。可运行在边缘网关的流式数据处理系统
45		Oozie	工作流调度引擎
46	其他数据处理相关系统	Knox	Hadoop的rest服务
47		Crunch	简化MapReduce编程API
48		Airavata	分布式计算任务和工作流的管理框架,采用微服务架构
49		Ambari	帮助管理Hadoop集群
50		Camel	企业集成框架,主要用于各系统之间的整合与通信
51		Beam	支持流式、批式数据处理的统一编程模型
52		Bigtop	简化Hadoop相关项目的打包和协同测试

1　大数据存储引擎关键技术

数据库的存储引擎负责缓冲区管理、磁盘数据管理数据查询处理等,是数据库的核心模块。存储引擎设计的好坏直接影响数据库的读写性能。通常来说,存储引擎的设计取决于目标负载,针对不同负载要求可以采用不同的存储管理方式。

1.1　系统负载管理

一般来说,有两种类型的负载:(1) 读多写少,例如,社交网络应用中一次写入对应多次读取;(2) 写多读少,像物联网场景中大量机器数据实时写入,但只有少量的读取请求。针对这两种负载,有不同的设计方案。

其中,读多写少是传统关系数据库的负载,通常采用B+树来管理数据,传统数据库中常用的基于B+树的存储引擎,例如InnoDB[5],虽然对于查询特别是扫描操作有着很高的性能,但是在写入时会引入大量的随机IO。

在现在的大数据应用中,尽管写入数据量在成倍增长,但查询的压力却没有以相同的速度增加。根据Yahoo公司的统计,近年来写入请求占总请求的比例在逐年上涨,逐渐占据了主导地位[6]。因此,对于写入性能的优化也受到了越来越多的关注。而基于B+树的方案在目前较为普及的基于机械硬盘的服务器上形成了严重的IO瓶颈。

1.2　LSM结构

基于日志索引结构树的LSM-Tree(log-structured merge tree)[7]方法是解决B+树在写入时引入大量随机I/O导致机械硬盘出现性能瓶颈问题的一个有效手段。Log-structured是文件系统的一种常用技术,指文件的数据在写入磁盘时将其以仅追加日志的形式写入磁盘,而非将同一个文件的数据连续地存放,通过建立索引来支持对文件的连续读取。

和B+树不同,LSM在磁盘上的数据是不可修改的,新的数据写入时会先在内存中进行缓存。当内存中的缓冲区满时,这部分数据会被排序并且整

体写入到磁盘形成一个数据块。由于这种写入操作不会修改原有的数据块,大量的随机I/O得到避免,写入效率也得到了提高。通常LSM-Tree在写入性能上相比B+树有一个数量级左右的提升[8]。

然而,LSM在查询性能上也存在着巨大的缺陷,这是因为尽管数据块在内部是有序的,但是数据块之间是无序的。在查询时,一方面可能需要对于多个数据块进行归并查询,这会导致磁盘在多个位置来回seek,从而产生大量随机I/O;另一方面,数据块之间可能包含重复的数据,但只有最新的数据块是有效的,这就使得在查询相同数据点数时,LSM-Tree需要查询更多的数据,导致性能受到影响。为了解决该问题,LSM-Tree会对磁盘上的数据进行合并整理,即将较小的、可能存在重复数据的若干数据块,通过归并排序,重新写成一个较大的,只保留最新数据的数据块,这个过程便是LSM中的合并操作(merge,compaction)。为了确定合并的策略,LSM-Tree中将数据块依据大小分为若干层级C_0,C_1,...,C_n,内存中的数据在写入到磁盘后进入C_0,同一层的数据块之间进行merge形成一个数据块,当该层的数据块大小达到一定值以后,会与下一级合并形成新的数据块。通常来说,层级的大小上限是以指数增长的。在具体的系统中,往往会根据其管理的数据特性进行相关策略上的适配。例如,Apache Cassandra中实现了4种合并策略,分别是按照文件大小和数量的合并策略、按照文件层级的合并策略、按照定长时间窗口的合并策略、按照变长时间窗口的合并策略。

合并操作提高了数据的连续性,减少了数据的冗余程度,缓解了LSM查询时的性能问题。但是合并本身既包含对老数据的读也包含对新数据的写,本身会占用大量的系统资源,从而削弱了LSM的写入优势。因此,许多工作从减少合并次数、提高合并并行度、减少单次合并的I/O开销,优化合并调度策略减少对常规业务的影响等方面着手,提出了各自对LSM-Tree的改进[6-10]。

1.3 列式文件格式

人们注意到,为了应对大数据分析,行式的文件结构存在诸多问题。首先,行式文件结构难以实现高压缩,这使得数据容量进一步膨胀。其次,行式的文件结构与多数OLTP系统类似,对于行式查询十分友好,但是对于一些面向分析的应用来说,会带来大量磁盘I/O等操作[11]。

在大数据生态发展过程中,逐渐产生了大量列式文件格式,并用于各种分析系统和应用当中。例如,目前常见的列式文件系统包括Apache Parquet、CarbonData、TsFile。其中前两者主要面向通用的关系型数据,而TsFile则主要面向时间序列这种特定数据类型。在3种列式格式文件中,均有块(block)、列(chunk)、页(page)的类似概念,这些概念代表了不同粒度大小的数据块。每个块包含了若干列数据,每个列仅包含单列数据,而这部分数据又被划分为若干页。页是这些文件系统从磁盘上做一次读取操作的最小单元。

以Parquet为例,Parquet提供用于存储嵌套型数据的结构,其设计思想来源于谷歌提出的Dremel[12]。该文件格式中块对应的概念被称作行组(row group)。每个Page内不仅存储了某列的若干个数据,还存储了两个特殊的内容:重复层次(repetition level)和定义层次(definition level)。其中,重复层次是用于判断某个数据是属于嵌套结构中的哪一层的,而定义层次则是用于判断某个字段是否为空(null)。Apache Parquet中,由于不同层次的嵌套结构数据被展开成平层的列,两个字段重复层次和定义层次用来将各个Column中的数据重组成原来的一行数据。Parquet文件尾部存储了元数据信息Metadata,元数据和数据是对应的,相当于数据的索引信息。例如,每一个列组的元数据记录了对应列组的列数、在文件中的偏移量等。

虽然通用的列式存储文件格式能够很好地应对多数应用场景,但是面对特定数据类型时,这些通用文件格式难以做到针对这些数据类型的高压缩、高读写。为此,针对特定数据类型的文件存储格式也应用而生。例如,Apache TsFile是一种针对时间序列存储优化的文件格式。TsFile针对工业物联网领域的时序数据管理特点,采用了“设备-传感器”两层数据模型。TsFile的数据组织层次分为列组(chunk group)、列与页3层,其中每个列组

属于一个特定的设备,一个列属于一个传感器。文件尾部对所有列组按照设备号进行了索引。使得TsFile可以高效地管理多个设备的数据。每个页内部包括时间列和值列两列数据,采用列式存储,具有高压缩比和读写性能。

除了提出新的文件结构外,目前针对存储结构的研究主要关注如何面对动态变化的负载,寻找最佳的存储结构或索引结构,从而避免系统管理员手动调整存储结构[13-14]。

2　数据分区技术分析

数据分区作为分布式存储系统的基本功能,其分区结果好坏决定了系统中是否存在热点节点,并直接制约了系统的读写性能。数据分区主要包括两项任务:一是将数据分成不重复的子集,每个子集是数据复制和分配的基本单元;二是将各个子集划分到集群中不同的节点。传统的关系数据的分区策略包括行式分区和列式分区,分布式数据库中通常采用分布式哈希表(DHT)或查找表的方式进行分区。

一致性哈希是分布式哈希表的一种实现。一致性哈希首先要确定哈希函数,哈希函数的值域构成一个环,接下来将各个节点进行哈希,哈希成环上的一个节点。P个节点会将环分割成P段。每个节点负责操作逆时针方向最近段的数据。当要操作一个数据点时,先将数据进行哈希,当数据点的哈希值落在某节点负责的段上,这个数据就归该节点管理。基于一致性哈希的副本策略为:顺时针在环上寻找相邻节点进行存储。假定有5个节点——A、B、C、D、E,数据项a被哈希到了B所负责的段上。当副本数为3时,a的副本就分配在B及其顺时针方向最近的2个节点C和D上。一致性哈希的缺点是无法根据动态变化的读写负载进行调整。因此,一致性哈希发展出了虚拟节点的概念,一个物理节点对应多个虚拟节点,虚拟节点会将一致性哈希环分成更多的段,一定程度避免了数据不均衡的问题。开源系统ApacheCassandra就采用了一致性哈希的分区方式。一致性哈希的优势是各个节点不需要维护分区信息,只需要记录哈希函数就可以定位数据所在节点,实现较为简单。

在一致性哈希中,节点到地址空间的映射决定了数据的分区结果,而映射结果(即数据分区结果)对系统的性能影响很大。其主要原因是当节点在地址空间的映射不均时,会使得某些节点负责大量的数据,从而成为性能瓶颈[15]。该问题在数据存在副本、不同节点性能不同(即异构性能的集群)的应用场景中更为突出。

一致性哈希本质上是记录了哈希函数,而查找表是将数据的分区和对应的节点信息全部记录在分区信息表中,当读写操作到来时,先到分区表中查找所在节点,再进行请求路由。Apache Hbase就采用查找表的方式管理分区信息,并且将分区信息注册到Zookeeper中。查找表的优势在于分区信息不受限于某个哈希函数,可根据负载动态调整,但查找表的更新需要额外处理。

针对数据分区的研究中,基于DHT的负载均衡问题提出了多个维度上的度量方法:分别基于节点和数据分布的维度、基于查询分布的维度、基于流量的维度进行度量[15]。后两者随着副本机制的兴起,已可以通过副本机制实现。前两种度量方法在目前的系统中主要采用生成随机数的方法[16]或手动配置的方法[17]进行数据分区,其缺点是很容易导致次优解出现[18],而后两种方法则过于烦琐。此外,为了更好地做到负载均衡,虚拟节点技术[19-21]及一些混合分区方法[22]也被引入某些合适的应用场景下,但是不太适用于像Cassandra这种虚拟节点数量有上限的系统。

分布式系统具有较好的水平扩展能力,因此集群扩展时的数据分区修改也是值得研究的主题之一。已有研究者对Cassandra和HBase系统的扩展性进行了实验[23],但实验仅涉及系统默认的数据分区方法。还有人提出基于节点性能的贪心算法寻找新增节点时的最优设置[24],以及类似于混合分区的方法改变每个节点的虚拟节点数量,以达到更好的分区效果[25]。随着实际应用中集群规模的不断扩大,以及云原生数据库的兴起,数据分区与数据迁移的优化依然是分布式数据管理的研究热点。

3 副本机制设计

副本是指一份数据在系统中重复存储多份,以降低对某一节点的访问负载,确保数据的高可用性与高可靠性。例如,为了保证系统的可用性和数据的安全性,HDFS、HBase、Cassandra、MongoDB等多数分布式系统均采用了副本机制,实现了在大量不可靠的服务器上可靠地存储海量数据。

3.1 副本一致性问题

副本机制带来的一个重要问题是副本一致性问题。一致性是对系统中的数据状态的一种描述,并且在不同场景中有不同的含义:在传统数据库的事务领域,是指在事务发生前后,数据的约束是一致的;在其他领域中,尤其是在分布式存储系统中,指的是数据的多个拷贝(即副本)在物理上或者被用户认为是一致的,此时称之为副本一致性。

对于副本一致性这一特定优化问题的讨论可以追溯至30年前[26],且时至今日仍是热点问题[27]。副本一致性与传统关系数据库中所说的ACID中的"C"所代表的一致性侧重点不同。对于ACID中的C代表的一致性主要指数据的约束状态是正确的[28],而副本一致性则是指一个数据在多个副本之间是一致的,这种一致性又被称作相互一致性(mutual consistency)[29]。副本一致性的研究工作包括副本一致性的度量、模型定义和改进。

针对副本不一致问题,研究界和工业界做了大量的对实际系统的一致性度量。例如,Facebook自称在系统中存在0.0004%的读操作会返回不同的值[27];有研究者分析Amazon SimpleDB系统中在0~450 ms内读取到最新值的概率是33%[30];还有研究者展示了在Yammer公司的Riak集群中,读取最新数据最大的延迟为5 s[31]。

人们意识到,在实际应用中对系统的一致性的评价不是一个二值问题,并且实际应用中并不要求副本绝对的强一致性。例如,在谷歌日历的应用中,当用户修改共享权限时,虽然能立刻保存成功,但是可能需要经过24 h才能生效。尽管最坏需要24 h才能达到数据的一致,但是人们认为对于谷歌日历这个应用是足够的。又如在电子商务应用中,卖家对商品价格的修改必须立刻可以被其他买家读取到,否则就可能出现交易纠纷。在第一个应用中,虽然用户在24 h内可能察觉不到权限修改的生效,但是对应用来说,这仍然是"对"的,而对于第二个应用来说,即使买家晚1 min看到最新价格,也可能是"错"的。因此,在设计系统时,首先要明确什么是"对",什么是"错",即人们所说的一致性模型定义。

一致性模型通常定义为两大类:第一种是从系统提供者的角度观察,关注点为副本间的进程同步和操作顺序,即以数据为中心的一致性;第二种是从客户端、使用者的角度观察,这时系统是一个黑盒,这种角度称为以用户为中心的一致性模型。两种一致性模型又可以进一步细分为强一致性[32]、弱一致性[32]、最终一致性[32]、因果一致性[33]、单调读一致性[34]。

一致性的改进方法[35-41]往往与一致性模型相对应。也就是说,一种一致性的改进方法仅适用于特定的一致性模型。此外,一致性的改进工作主要基于CAP理论[42]和PACELC理论[43]展开[44-46]。虽然对副本一致性有着如此大量的研究工作,但对于副本一致性和延迟或者其他系统特性的权衡、对副本一致性的度量和提升工作有迫切需求。

在现有系统中,不同的系统采用了不同处理方法应对副本一致性问题。

在HDFS中,副本以"基于管道复制"的方式进行拷贝[47]:当客户端写数据到HDFS时,它会首先将数据块写给第1个存储节点,当第1个存储节点接收到数据后,该节点会链式地将收到的数据传递给第2个存储节点用于存储第2个副本,以此类推,直到最后一个存储节点也接收到一个副本为止。

为了保证副本一致性,HDFS采用了一种悲观策略:给定一个数据块,当所有副本都写成功时,主节点才会标记该数据块写成功。这种简单的机制确保了从用户角度看到的所有数据都是一致的(但在早期Hadoop的发布版本中,由于实现问题,存在数据不一致的现象)。而事实上,该策略之所以能够在HDFS上有效运行、不影响系统的可用性(例如某个节点宕机时系统仍然支持写入数据),是由

于数据块的副本放置位置是由主节点决定的,这与其他系统有明显不同。

针对HDFS存在的副本问题,现有研究工作围绕副本放置和传输展开,确保各节点的负载均衡[48-49]、提升HDFS的读写效率[50]、集群变化时的数据恢复[51]、减少集群负载[52]、支持更好的分布式数据分析。在Cassandra系统中,副本以法团机制进行管理,以时间戳的方式对每个数据添加版本信息,并采用Merkle Tree的数据结构进行副本对比。Cassandra采用了可调的一致性,支持最终一致性到强一致性,但要牺牲可用性。

3.2　副本与纠删码

在HDFS2.0及之前的版本中,虽然通过副本机制确保了系统的可靠性,然而这种机制带来了较大的磁盘空间占用,并且这个问题随着数据量的增大显得愈发严重。例如,一般而言HDFS中的副本数量往往大于等于3,这就意味着每存储1 TB数据,需要占用3 TB的存储空间,这给存储空间和网络带宽带来了很大的压力。

为此,从HDFS3.0起,纠删码机制被引入系统。纠删码最初被应用于通信传输过程中的错误检查和恢复。对于存储系统而言,由于存储系统中数据是要不断被写入、被更新,在使用纠删码过程中不仅要考虑数据恢复能力,还要考虑数据的更新效率。存储系统中常用的纠删码技术,包括RS编码、横式阵列码、纵式阵列码、LDPC编码等[53],纠删码的评价指标包括容错能力、空间利用率、编码效率、更新效率、重构效率5种。

总体而言,如果一个存储系统用于频繁地更新数据,则采用副本机制更加合适;如果一个存储系统主要用于归档数据,则采用纠删码更加合适。由于纠删码在恢复数据时不可避免地涉及到较多计算,最近也有一些工作通过使用GPU或者TPU加速纠删码计算[54-55]。

4　分布式协议

为使集群中的各节点准确有效地协同工作,多种分布式协议被提出,现选择其中最有代表性的法团协议和Paxos等共识算法进行介绍。

4.1　法团(Quorum)协议

法团协议是一种相对简单的副本控制协议,主要通过控制读写请求需要完成的副本数实现对副本一致性的控制。假设系统采用N=3副本,写一致性W的值代表一个写操作返回成功前需要有多少个副本成功执行了这次写操作;读一致性R值的含义类似。调整W和R的值就会得到不同的系统一致性。如果$W+R\geqslant N$,则系统可以保证强一致,根据鸽巢原理,读操作总能读到一个最新版本的数据。Apache Cassandra等系统就采用了法团协议。

此外,法团协议可以作为其他复杂协议的基础。后面提到的Paxos、Raft等算法均采用了“如果超过半数以上参与者同意,则提案生效”的策略,本质上就是一种法团协议。

4.2　共识算法

分布式数据库通常有两种架构,第一种是主从架构,系统集群中通常有一个主节点,负责所有请求的路由和分配、资源调度等,其余节点为从节点,接受主节点的请求。主从架构的优点是实现简单,但是主节点会成为系统的性能瓶颈。此外,主节点宕机后集群无法提供服务。Hadoop、Spark就采用了这种架构。另一种是分布式的P2P架构,系统中所有节点角色相同,均可接受写入和查询请求。接受请求的节点都会充当协调者,负责将请求路由给多个副本节点,Cassandra就采用了这种架构。

共识(consensus),有时也被意译为“一致性”,认为“共识”能更好地表达其完成的功能。共识算法最常用于支持分布式事务应用场景。主从架构的系统中,当主节点宕机时,需要从剩余的从节点中选出一个新的主节点,这就要求各个从节点能够“达成共识”。P2P架构中,由于系统中没有特殊地位的节点,因此当多个节点产生冲突时,也需要一种算法来使得各个节点“达成共识”,因此需要共识算法。

针对共识问题,最简单的想法是各节点产生冲突时,以第一个提出操作的节点为准。然而,由于分布式系统的各节点不存在严格对齐的物理时钟,因此这种方案是不可行的。常用的两种分布式共

识算法为Paxos[56]和Raft[57],在Paxos算法中,节点角色较多,算法比较复杂,因而没有被工业界广泛使用,仅有一些大型系统使用了Paxos协议,例如Zookeeper、OceanBase。Raft则是一种比Paxos更易懂的共识协议,从提出之日起,就有各种语言的实现版本,被广泛应用于分布式系统中。目前,针对Paxos和Raft都存在大量的改进工作[58-62]。

随着硬件技术的发展,不存在严格对齐的物理时钟的问题逐步解决。例如,谷歌提出的Spanner系统就通过采用原子钟,达成了各节点统一时钟的状态,从而大幅度简化了分布式事务的处理。

5 Apache社区中的大数据管理系统

5.1 分布式文件系统Apache Hadoop

Apache Hadoop[4]是Doug Cutting等基于谷歌提出的GFS、MapReduce技术研发的开源大数据管理和处理软件。Hadoop分布式文件系统(Hadoop distributed file system,HDFS)是其中一个模块,是谷歌GFS的开源实现。HDFS适合存储大文件,其采用分布式的思想,一个HDFS集群由主节点和多个存储节点组成,将文件切分成若干个数据块,每个数据块存储在不同的节点上,主节点负责存储各数据块与节点的对应关系。在HDFS中,各数据块应该被存储在哪些节点上是由主节点根据各存储节点的负载决定的,并且这些数据块-存储节点的对应关系会被被逐一记录到磁盘上。所有对于数据块的访问都经过主节点来分配协调。为了保证数据的可靠性和服务高可用,HDFS采用副本机制和纠删码保证数据可靠性。

目前尚无证据能够证明在HDFS系统中副本是否严格强一致性。HDFS系统问题记录显示,在特定场景下,HDFS会产生副本不一致的情况。Hadoop虽然提供了分布式文件存储的能力,但是并不限制用户存储何种类型的数据。Hadoop HDFS将这些文件统一当作二进制文件黑盒化处理。而Hadoop MapReduce组件为了进行数据处理,原生地支持了文本文件、CSV文件格式、压缩文件等格式。通过自定义文件InputFormat等接口,MapReduce可以支持任意格式的文件。

5.2 键值数据管理系统

键值数据模型简单易用,一个数据项由键和值两部分组成,写入接口为write(键)=值。查询接口为get(键)。目前流行键值数据库包括Apache Accumulo[63]、Apache Cassandra、LevelDB[64]、RocksDB[65]等。键值数据库的模型简单,支持的查询也仅限于单点查询、范围查询。键值数据库的存储引擎通常都是基于LSM的。Apache Accumulo有自己的服务进程,而LevelDB、RocksDB常用作其他系统的内置存储模块。

除了在工业界广泛应用以外,LevelDB和RocksDB因其开源特性,也常常在学术中被用作基础系统来实现研究者所提出的算法,或作为与研究者的比较基准。有研究者提出将LevelDB的键和值分离的存储方式,减少了合并时不必要的数据重写[66]。也有人为LSM定制了优化的文件系统,并在其上运行LevelDB,获得了显著性能提升[67]。国内也有学者开展关于LevelDB的研究,对LevelDB中LSM-Tree的持久化缓存机制进行改进,取得不错的效果[68]。

5.3 时序数据管理系统

物联网时序数据是工业大数据的主要管理对象。典型的大数据应用场景可能包含数万个联网设备,每台设备以秒级的频率采集几十上百个测点,最终达到每秒数百万数据点的吞吐率要求。时序数据管理系统主要分为3种架构:基于关系数据库的,例如TimescaleDB[69];基于键值数据库的,例如OpenTSDB[70]、KairosDB[71];原生的时序数据库,例如InfluxDB[72]、Apache IoTDB[73]、Little Table[74]。其中IoTDB是中国高校目前唯一进入Apache的项目,也是目前Apache社区唯一一个时序数据库项目。目前IoTDB可以实现单节点每秒数百万甚至千万数据点写入、TB级数据毫秒级查询、支持查询分析一体化应用、"云-端"一体化部署等功能。InfluxDB在时序数据库中起步较早、功能丰富、用户基础好,是国内外目前应用最广的项目,主要被应用在工业数据库的设计中[75],尤其是环境监控应用场景中[76]。

由于时间序列数据本质上是一个时间函数的部分采样结果,而现实中的函数多数是连续、有一定规律的,这使得对时间序列进行压缩切实可行,一般时间序列数据库系统也都内置有压缩算法。时间序列数据总量庞大,压缩代价自然也不小,对于数据库系统会产生较大的负担,因此有学者提出利用GPU加速压缩[77],以不同的分辨率(小时、天、周、月)来查看同一条时间序列是最常见的分析需求,为避免在切换分辨率时进行重复或者非必要的计算,有研究者提出支持对时间序列进行多分辨率的存储与高效查询[78]。时间序列数据同时也是一种流式数据,因时间戳的存在为其带来了新的机遇与挑战,因此有的研究者尝试在时间序列上运用改良后的一些流数据处理技术[79],并取得了很好的效果。

6　结论

对现有开源软件的大数据管理系统,以及其中使用的典型数据管理技术进行了综述,内容涵盖数据存储、数据分区、副本机制、一致性协议等。在大数据管理系统中,存储、分区、副本与一致性协议技术相互关联交织在一起,在引入一种技术时往往需要同时考虑其他部分的依赖。

大数据技术研究领域还有非常多的开放问题有待解决,尤其是在面向特定领域数据管理,例如时序数据管理、图数据管理及根据负载动态优化数据库等问题的解决方面,有广泛的研究空间。

参考文献(References)

[1] Ghemawat S, Gobioff H, Leung S-T. The Google file system[C]//Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles. New York: ACM, 2003: 29-43.

[2] Chang F, Dean J, Ghemawat S, et al. Bigtable: A distributed storage system for structured data[J]. ACM Transactions on Computer Systems, 2008, 26(2): 1-26.

[3] Wu C, Huang Y F, Lee J. Comparisons between MongoDB and MS-SQL databases on the TWC website[J]. Journal of Software Engineering and Applications, 2015, 4: 35-41.

[4] Apache Projects List[EB/OL]. [2019-11-17]. https://projects.apache.org/projects.html?category.

[5] Kofler M. InnoDB Tables and Transactions[M]//The Definitive Guide to MySQL. Apress, Berkeley, CA, 2004: 239-259.

[6] Sears R, Ramakrishnan R. bLSM: A general purpose log structured merge tree[C]//Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. New York: ACM, 2012: 217-28.

[7] O'Neil P, Cheng E, Gawlick D, et al. The log-structured merge-tree (LSM-tree)[J]. Acta Informatica, 33(4): 351-385.

[8] Shetty P, Spillane R, Malpani R, et al. Building workload-independent storage with VT-trees[C]//Usenix Conference on File & Storage Technologies. New York: ACM, 2013: 13-17.

[9] Zhang Z, Yue Y, He B, et al. Pipelined compaction for the LSM-tree[C]//2014 IEEE 28th International Parallel and Distributed Processing Symposium. Piscataway N J: IEEE, 2014: 777-786.

[10] Pan F, Yue Y, Xiong J. dCompaction: Delayed compaction for the LSM-tree[J]. International Journal of Parallel Programming, 2017, 45(6): 1310-1325.

[11] Abadi D. The design and implementation of modern column-oriented database systems[J]. Foundations and Trends in Databases, 2012, 5(3): 197-280.

[12] Melnik S, Gubarev A, Long J J, et al. Dremel: Interactive analysis of web-scale datasets[C]//Proceedings of the 36th International Conference on Very Large Data Bases. Piscataway N J: IEEE, 2010: 330-339.

[13] Bian H, Yan Y, Tao W, et al. Wide table layout optimization based on column ordering and duplication[C]//Proceedings of the 2017 ACM International Conference on Management of Data. New York: ACM, 2017: 299-314.

[14] Schlosser R, Kossmann J, Boissier M. Efficient scalable multi-attribute index selection using recursive strategies [C]//2019 IEEE 35th International Conference on Data Engineering. Piscataway N J: IEEE, 2019: 209-220.

[15] Felber P, Kropf P, Schiller E, et al. Survey on load balancing in peer-to-peer distributed hash tables[J]. IEEE Communications Surveys & Tutorials, 2013, 16(1): 473-492.

[16] Lakshman A, Malik P. Cassandra: a decentralized structured storage system[J]. ACM SIGOPS Operating Systems Review, 2010, 44(2): 35-40.

[17] Black B. Cassandra Troubleshooting[Z]. Cassandra Sum-

mit 2010, 2010.

[18] Paiva J, Ruivo P, Romano P, et al. Auto placer: Scalable self-tuning data placement in distributed key-value stores[J]. ACM Transactions on Autonomous and Adaptive Systems (TAAS), 2015, 9(4): 19.

[19] Virtual nodes/balance[EB/OL]. [2019-11-17]. http://wiki.apache.org/cassandra/VirtualNodes/Balance.

[20] Dabek F, Kaashoek M F, Karger D, et al. Wide-area cooperative storage with CFS[C]//ACM SIGOPS Operating Systems Review. New York: ACM, 2001, 35(5): 202-215.

[21] Stoica I, Morris R, Karger D, et al. Chord: A scalable peer-to-peer lookup service for internet applications[J]. ACM SIGCOMM Computer Communication Review, 2001, 31(4): 149-160.

[22] Chen Z, Yang S, Tan S, et al. Hybrid range consistent hash partitioning strategy—A new data partition strategy for NoSQL Database[C]//2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications. Piscataway N J: IEEE, 2013: 1161-1169.

[23] Kuhlenkamp J, Klems M, Röss O. Benchmarking scalability and elasticity of distributed database systems[J]. Proceedings of the VLDB Endowment, 2014, 7(12): 1219-1230.

[24] Wang X, Loguinov D. Load-balancing performance of consistent hashing: asymptotic analysis of random node join[J]. IEEE/ACM Transactions on Networking, 2007, 15(4): 892-905.

[25] Hsiao H C, Chung H Y, Shen H, et al. Load rebalancing for distributed file systems in clouds[J]. IEEE Transactions on Parallel and Distributed Systems, 2012, 24(5): 951-962.

[26] Parker D S, Popek G J, Rudisin G, et al. Detection of mutual inconsistency in distributed systems[J]. IEEE Transactions on Software Engineering, 1983(3): 240-247.

[27] Lu H, Veeraraghavan K, Ajoux P, et al. Existential consistency: Measuring and understanding consistency at Facebook[C]//Proceedings of the 25th Symposium on Operating Systems Principles. New York: ACM, 2015: 295-310.

[28] Agrawal D, El Abbadi A, Singh A K. Consistency and orderability: Semantics-based correctness criteria for databases[J]. ACM Transactions on Database Systems, 1993, 18(3): 460-486.

[29] Thomas R H. A majority consensus approach to concurrency control for multiple copy databases[J]. ACM Transactions on Database Systems, 1979, 4(2): 180-209.

[30] Wada H, Fekete A, Zhao L, et al. Data consistency properties and the trade-offs in commercial cloud storage: the consumers´ perspective[C]. 5th The biennial Conference on Innovative Data Systems Research, Asilomar, CA, January 9-12, 2011.

[31] Bailis P, Venkataraman S, Franklin M J, et al. Probabilistically bounded staleness for practical partial quorums [J]. Proceedings of the VLDB Endowment, 2012, 5(8): 776-787.

[32] Vogels W. Eventually consistent[J]. Communications of the ACM, 2009, 52(1): 40-44.

[33] Bailis P, Ghodsi A, Hellerstein J M, et al. Bolt-on causal consistency[C]//Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. New York: ACM, 2013: 761-772.

[34] Terry D B, Demers A J, Petersen K, et al. Session guarantees for weakly consistent replicated data[C]//Proceedings of the Third International Conference on Parallel and Distributed Information Systems, 1994. Piscataway N J: IEEE, 1994: 140-149.

[35] Terry D B, Demers A J, Petersen K, et al. Session guarantees for weakly consistent replicated data[C]//Proceedings of 3rd International Conference on Parallel and Distributed Information Systems. Piscataway N J: IEEE, 1994: 140-149.

[36] Nayate A, Dahlin M, Iyengar A. Transparent information dissemination[C]//ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing. Berlin: Springer, 2004: 212-231.

[37] Bourne S. A conversation with Bruce Lindsay[J]. Queue, 2004, 2(8): 22-33.

[38] Urgaonkar B, Ninan A G, Raunak M S, et al. Maintaining mutual consistency for cached web objects[C]//Proceedings 21st International Conference on Distributed Computing Systems. Piscataway N J: IEEE, 2001: 371-380.

[39] Theel O, Raynal M. Static and dynamic adaptation of transactional consistency[C]//Proceedings of the Thirtieth Hawaii International Conference on System Sciences. Piscataway N J: IEEE, 1997, 1: 533-542.

[40] Perrin M, Petrolia M, Mostefaoui A, et al. Consistent shared data types: Beyond memory[D]. Nantes: Université de Nantes, 2014.

[41] Bailis P, Ghodsi A, Hellerstein J M, et al. Bolt-on caus-

al consistency[C]//Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. New York: ACM, 2013: 761-772.

[42] Brewer E A. Lessons from giant-scale services[J]. IEEE Internet Computing, 2001, 5(4): 46-55.

[43] Abadi D. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story[J]. Computer, 2012, 45(2): 37-42.

[44] Davidson S B, Garcia-Molina H, Skeen D. Consistency in a partitioned network: A survey[J]. ACM Computing Surveys (CSUR), 1985, 17(3): 341-370.

[45] Tanenbaum A S, Van Steen M. Distributed systems: principles and paradigms[M]. Upper Saddle River: Prentice-Hall Inc., 2007.

[46] Bermbach D, Kuhlenkamp J. Consistency in distributed storage systems[C]//International Conference on NetworkedSystems. Berlin: Springer, 2013: 175-189.

[47] HDFS architecture guide[EB/OL]. [2019-10-07]. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Replication+Pipelining.

[48] Dai W, Ibrahim I, Bassiouni M. A new replica placement policy for hadoop distributed file system[C]//2016 IEEE 2nd International Conference on Big Data Security on Cloud (Big Data Security), IEEE International Conference on High Performance and Smart Computing (HPSC), IEEE International Summit (Confluence). Piscataway N J: IEEE, 2014: 36-39.

[49] Ye X, Huang M, Zhu D, et al. A novel blocks placement strategy for Hadoop[C]//2012 IEEE/ACIS 11th International Conference on Computer and Information Science. Piscataway N J: IEEE, 2012: 3-7.

[50] Patel N M, Patel N M, Hasan M I, et al. Improving HDFS write performance using efficient replica placement[C]//5th International Conference—Confluence The Next Generation Information Technology Conference on Intelligent Data and Security (IDS). Piscataway N J: IEEE, 2016: 262-267.

[51] Higai A, Takefusa A, Nakada H, et al. A study of effective replica reconstruction schemes at node deletion for HDFS[C]//2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. Piscataway N J: IEEE, 2014: 512-521.

[52] Qu K, Meng L, Yang Y. A dynamic replica strategy based on Markov model for hadoop distributed file system (HDFS) [C]//2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS). Piscataway N J: IEEE, 2016: 337-342.

[53] 舒继武, 罗象宏, 存储系统中的纠删码研究综述[J]. 计算机研究与发展, 2012, 49(1): 1-11.

[54] Haddock W, Curry M L, Bangalore P V, et al. GPU erasure coding for campaign storage[M]//Kunkel J M, Yokota R, Taufer M, et al. High Performance Computing. Cham: Springer International Publishing, 2017: 145-159.

[55] Chen H, Fu S. Parallel erasure coding: Exploring task parallelism in erasure coding for enhanced bandwidth and energy efficiency[C]//2016 IEEE International Conference on Networking, Architecture and Storage (NAS). Piscataway N J: IEEE, 2016: 1-4.

[56] Lamport L. Paxos made simple[J]. ACM Sigact News, 2001, 32(4): 18-25.

[57] Ongaro D, Ousterhout J. Ousterhout. In Search of an Understandable Consensus Algorithm[C]//Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference. Berkeley: USENIX, 2014: 305-320.

[58] Lamport L. Fast paxos[J]. Distributed Computing, 2006, 19(2): 79-103.

[59] Lamport L, Massa M. Cheap paxos[C]//International Conference on Dependable Systems and Networks, 2004. Piscataway N J: IEEE, 2004: 307-314.

[60] Gafni E, Lamport L. Disk paxos[J]. Distributed Computing, 2003, 16(1): 1-20.

[61] Copeland C, Zhong H. Tangaroa: A byzantine fault tolerant raft[J/OL]. [2019-09-30]. http://www.scs.stanford.edu/14au-cs244b/labs/projects/copeland_zhong.pdf.

[62] Arora V, Mittal T, Agrawal D, et al. Leader or majority: Why have one when you can have both? improving read scalability in raft-like consensus protocols[C]//9th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 17). Berkeley: USENIX, 2017: 46-51.

[63] Apache Accumulo[EB/OL]. [2019-11-17]. https://accumulo.apache.org.

[64] LevelDB[EB/OL]. [2019-11-17]. https://github.com/google/leveldb.

[65] RocksDB[EB/OL]. [2019-11-17]. https://rocksdb.org.

[66] Wang L, Ding G, Zhao Y, et al. Optimization of leveldb by separating key and value[C]//2017 18th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT). Piscataway N J: IEEE, 2017: 421-428.

[67] Mei F, Cao Q, Jiang H, et al. LSM-tree managed storage for large-scale key-value store[J]. IEEE Transactions on Parallel and Distributed Systems, 2018, 30(2):

400-414.
[68] 饶毓琳. 基于LSM-Tree 的持久化缓存机制的优化研究[D]. 武汉: 华中科技大学, 2016.
[69] TimescaleDB[EB/OL]. [2019-11-17]. https://www.timescale.com.
[70] OpenTSDB[EB/OL]. [2019-11-17]. http://opentsdb.net.
[71] KairosDB[EB/OL]. [2019-11-17]. http://kairosdb.github.io.
[72] InfluxDB[EB/OL]. [2019-11-17]. https://www.influxdata.com/products/influxdb-overview.
[73] Apache IoTDB[EB/OL]. [2019-11-17]. https://iotdb.apache.org.
[74] Rhea S, Wang E, Wong E, et al. Littletable: A time-series database and its uses[C]//Proceedings of the 2017 ACM International Conference on Management of Data. New York: ACM, 2017: 125-138.
[75] 徐化岩, 初彦龙. 基于 influxDB 的工业时序数据库引擎设计[J]. 计算机应用与软件, 2019, 36: 9.
[76] Balis B, Bubak M, Harezlak D, et al. Towards an operational database for real-time environmental monitoring and early warning systems[J]. Procedia Computer Science, 2017, 108: 2250-2259.
[77] Przymus P, Kaczmarski K. Dynamic compression strategy for time series database using GPU[M]//New Trends in Databases and Information Systems. Berlin: Springer, 2014: 235-244.
[78] Llusa S A, Vila-Marta S, Escobet C T. Formalism for a multiresolution time series database model[J]. Information Systems, 2016, 56: 19-35.
[79] Sevcech J, Bielikova M. Symbolic time series representation for stream data processing[C]//2015 IEEE Trustcom/BigDataSE/ISPA. Piscataway N J: IEEE, 2015, 2: 217-222.

Big data technologies in open source software: A survey

JIANG Tian, QIAO Jialin, HUANG Xiangdong, WANG Jianmin

School of Software, Research Center for Big Data, Tsinghua University; National Engineering Laboratory for Big Data Software, Beijing 100084, China

Abstract The Google's GFS and Big Table have broken the limitations of the technology of having to use the relational databases to manage the big data in the past decade. A number of open source big data management systems, such as the Apache Hadoop, carry the technology further by developing more matured technologies and applications. This paper reviews the big data management systems in addressing the usage scenarios of the OLTP and the OLAP based on the Apache software, and the state of art of the data storage engine, the data partition, the data replication, the distributed system protocol, together with a comparison of the pros and the cons of the current distributed file system, the key value store, and the time series database.

Keywords big data management; open source software; distributed system

（责任编辑　刘志远）