

0TOB05610r回 a 10nrlr T0CU100po 考 1IOFptěj

or⁰D0m5ol

tol0ruAF040151300101010T0r0

( 和1010:G50r8100 uE01010T0701000i010

H0F0FOEeI0401 tG⁷010!010

01020POn

**第** **1** **2** **章** **Hadoop 及其生态系统**



**12.1 Hadoop简介**

Apache Hadoop是存储和处理大数据的开源软件框架，它在普通服务器 (Commodity Server) 组成的大规模集群上，对大数据进行分布式处理。

在扩展性 (Scalability) 方面，Hadoop 能够在上千台机器组成的集群上运行。大规模 集群的可靠性不能仅仅靠硬件来保证，因为节点的失败、网络的失败等状况不可避免。为 了能够在大规模集群上顺利运行，Hadoop 的所有模块的设计原则基于这样的基本假设， 那就是硬件的失败在所难免，每个节点都没有那么可靠，可能发生节点失败状况，软件框 架应该能够自动检测和处理这些失败情况。Hadoop 通过软件，在大规模集群上提供高度 的可用性 (High Availability)。

Hadoop 软件框架使用简单的编程模型 (Programming Model)MapReduce。在 Ha- doop1.0 版，用户只需以Map 函数和 Reduce 函数的形式提供数据处理逻辑，就可以在大 规模集群上对大数据进行处理。系统的可靠性、扩展性以及分布式处理等功能，由系统软 件层提供，用户无须关心。

Hadoop 项目最初由Doug Cutting和 Mike Cafarella于2005年创建，其最初的目标是 提供 Nutch 搜索引擎的分布式处理能力。目前Doug Cutting 是 Cloudera 的首席架构师， Cloudera 是一家基于Hadoop 开源软件，提供增值开发和服务的创业公司。

2013年， Hadoop 已经从1.0版演化发展到2.0版 (YARN) 。 在介绍 Hadoop 软 件 时，将首先介绍Hadoop1.0 的关键技术，然后对Hadoop2.0 的新特性做详细的介绍。

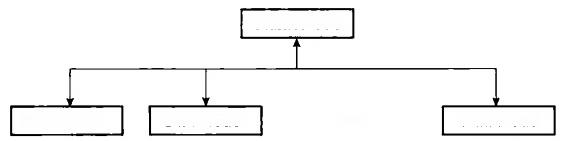
Hadoop 软件框架包含如下主要模块：(1) Hadoop Common, 这个模块包含了其他模 块需要的库函数和实用函数。(2) Hadoop Distributed File System(HDFS), 这是在由普 通服务器组成的集群上运行的分布式文件系统，支持大数据的存储，通过多个节点的并行 I/O, 提供极高的吞吐能力。(3) Hadoop MapReduce, 是一种支持大数据处理的编程模 型。(4) Hadoop YARN, 这是Hadoop2.0 的基础模块，它本质上是一个资源管理和任务 调度软件框架。它把集群的计算资源管理起来，为调度和执行用户程序提供资源的支持。

值得指出的是，Apache Hadoop的分布式文件系统 HDFS 和计算模型 MapReduce,

分别受到Google 分布式文件系统 (Google File System,GFS) 、Google MapReduce 计算 模型的启发，是对其进行模仿实现的开源软件。

**12.2** **Had** **oop分布式文件系统**

Hadoop 分布式文件系统 (Hadoop Distributed File System,HDFS) 是一个分布式 的、高度可扩展的文件系统，它是模仿GFS(Google File System) 实现的开源软件。它 使用Java 语言进行编写，具有良好的可移植性。 一个 HDFS 集群一般由一个NameNode 和若干 DataNode 组成，分别负责元信息的管理和数据块的管理(见图12-1)。

NameNode

DataNode DataNode DataNode

**图12-1** **HDFS架构**

HDFS 支持 TB 级甚至PB 级大小文件的存储.它把文件划分成数据块 (Block), 分 布到多台机器进行存储。为了保证系统的可靠性，HDFS 把数据块在多个节点上进行复制 (Replicate) 。 它并不需要在每个节点上安装RAID 存储来提供硬件级的可靠存储支持。

如果HDFS 采用的复制因子 (Replicate Factor) 为3,那么每个数据块有三个副本被 保存到三个节点上，其中的两个节点在同一个机架内，另外一个节点一般在其他机架上。 DataNode 之间可以拷贝数据副本，从而重新平衡每个节点存储的数据量，并且保证系统 的可靠性(保证每个数据块都有足够的副本)。

HDFS 一般存储不可更新的文件，只能对文件进行数据的追加。 Hadoop 大数据处理 系统一般用来支持大数据的分析型处理，数据来自各个业务系统，数据一旦装载，一般无 须进行更新。

由于 HDFS 是用Java 编写的，所以它内生地支持Java 应用程序编程接口 (API) 。 此 外 ，HDFS 还支持各种流行的编程语言，包括C++,Python,Ruby,C# 等。

Hadoop 的上层模块，比如 MapReduce 计算模型的运行时 (Runtime), 根据Name- Node 上的元信息，就可以知道每个数据块有多少副本，这些副本分别存放到哪些节点上， 于是可以把计算任务分配到这些节点执行。这样的调度策略把计算移动到数据上 (Move Computation to Data) 而不是移动数据本身，从而大大减少了大数据处理过程中的数据移 动开销，加快计算过程。

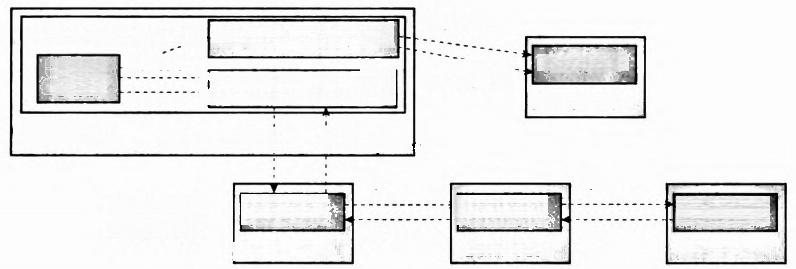
**12.2.1** **写** **文** **件**

为了进行文件数据的读写，客户端询问NameNode, 了解到它应该存取哪些 DataN- ode, 然后客户端直接和 DataNode 进行通信，数据的传输使用Data Transfer 协议，这是 一个流式数据传输协议，可以提高数据传输的效率。

所有NameNode 和 DataNode 之间的通信，包括DataNode 的注册、心跳信息、报告 数据块的元信息等，都是由DataNode 发起请求，由NameNode 来被动应答和完成管理。

当创建一个文件时，客户端把文件数据缓存在一个临时的本地文件。当本地文件累积 了超过一个数据块大小的数据时，客户端程序联系NameNode 。NameNode 更新文件系统 的命名空间 (Namespace), 并且返回新分配的 (Newly Allocated) 数据块的位置信息。 客户端程序根据这个信息把文件块数据从本地临时文件发送给 (Flush)DataNode 进 行 保存。当文件关闭 (Close) 时，剩下的最后一个数据块，传输到 DataNode 进行保存。

我们通过完整的例子展示 HDFS 如何创建一个文件，把数据写进去，最后关闭文件 (见图12-2)。整个过程涉及如下主要的步骤。



Client JVM 1.create Distributed File System

FS Data Output Stream

Client Node 4.write packet; 5.ack packt

4

Pipeline of DataNodes

DataNode

DataNode

5 DataNode

2.create

7.complete

NameNode

NameNode

DataNnde

DataNode

-3:write

6.close

HDFS client

DataNode

4

5

**图12-2** **HDFS** **文件的写入过程**

(1)客户端程序调用Distributed File System的 create() 方法。

(2)Distributed File System 向 NameNode 发起 一 个远程过程调用 (Remote Pre- ceduire Call,RPC), 由其在文件系统的命名空间里创建一个新文件。这时该文件还没有 任何数据块。

NameNode 进行一系列的检查，保证文件并未存在(才可以新建),而且客户端有权 限创建该文件。如果检查通过，那么NameNode 为新文件创建一个新的记录，保存其信 息，否则文件的创建失败。

Distributed File System返 回FS Data Output Stream 给客户端程序，以便其开始写入 数据。FS Data Output Stream 包含一个 DFS Output Stream 对象，它负责和 NameNode 以 及DataNode 的通信。

(3)当客户端开始写人数据， DFS Output Stream 把数据分解成数据包 (Packet), 并 且写入一个内部队列，称为数据队列 (Data Queue)。Data Streamer消费这个数据队列， 请求NameNode 为新的数据块分配空间，即选择一系列合适的DataNode, 用于存放各个 副本。

存放各个副本的DataNode 形成一个流水线 (Pipeline), 我们假设复制因子 (Replica Factor) 是3,于是在流水线上有三个节点。 Data Streamer 把数据包发送到第一个DataN- ode, 这个DataNode 保存数据包，并且转发给流水线上的第二个DataNode。

当写入数据超过一个数据块的大小时， Data Streamer向 NameNode 申请为新的数据 块分配空间。

(4)第二个DataNode 保存这个数据包并且转发给第三个(最后一个) DataNode。

(5)DFS Output Stream同时维护一个数据包的内部队列 (Internal Queue), 用于等 待接收DataNode 的应答信息，称为Ack Queue。当某个数据包被流水线上的所有DataN-

ode 应答以后，它才被(从Ack Queue上)删除。

(6)当客户端程序完成数据写入，它调用数据流的close() 方法。

(7)客户端把所有剩余的数据包发送到DataNode 流水线上，并且等待应答信息，最 后联系NameNode, 告诉它文件结束。

NameNode 知道文件由哪些数据块构成 (Data Streamer 请求为新的数据块分配空

间),它等待数据块的复制完成，然后返回文件创建成功。

·**12.2.2** **读文件**

在进行文件读取时，首先客户端程序使用将要读取的文件名、读取范围 (Read Range) 的开始偏移量、读取范围的长度等信息，询问NameNode 。NameNode 返回落在 读取范围内的数据块的位置 (Location) 信息。每个数据块的位置信息根据与客户端的临 近性 (Proximity) 进行排序。客户端一般选择最临近的DataNode, 向其发送读取请求。

对整个文件进行读取的过程，如图12-3所示。

Client JVM

HDFS client

Client Node

Distribued File System

ES Dat Iput Stream

1.open

3.read

6.close

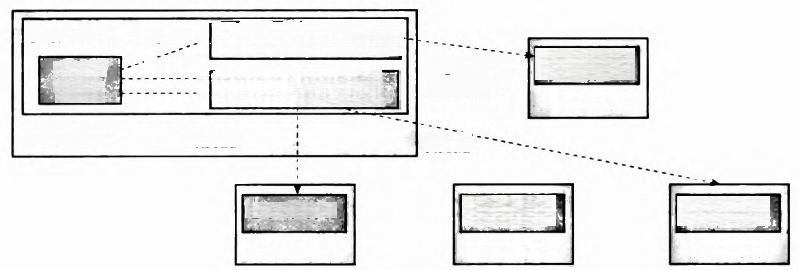
2.get block locations

NameNode

NameNode

**4.read**

5.read

DataNode

DataNode

DataNode

DataNode

**DataNode**

DataNode

**图12-3** **HDFS文件的读取过程**

客户端程序从HDFS 读取整个文件的主要步骤，包括：

(1)客户端程序通过调用File System 对象的 open() 方法，打开文件，获得 Distribu- ted File System类的一个实例。

(2)Distributed File System 通 过RPC 调 用NameNode, 获得文件开始若干数据块的 位置信息 (Locations of Blocks)。对于每个数据块来讲， NameNode 会返回拥有这个数据 块副本的所有DataNode 的地址。Distributed File System返 回FS Data Input Stream类 的 一个实例，这是一个输入流 (Input Stream), 支持文件的定位和数据的读取，使得客户端 可以读取文件数据。 FS Data Input Stream包含了一个DFS Input Stream 对象，负责管理 客户端对HDFS 的 DataNode 和 NameNode 的存取。

(3)客户端程序从输入流上调用函数read()。由 于DFS Input Stream 已经保存了文件 开始若干数据块所在的 DataNode 的 地 址 ，DFS Input Stream 连接到最近的 (Closest)

DataNode, 读取文件的第一个数据块。

(4)数据从DataNode 源源不断传送回客户端程序，客户端程序则不断地调用数据流 的 read() 方法。

(5)当到达数据块的末尾时， DFS Input Stream 将关闭到 DataNode 的连接，然后寻 找下一个数据块的最优的DataNode, 以便进行后续数据块的读取。 DataNode 的选择对客 户端程序来讲是透明的，客户端程序只是从一个连续的数据流进行读取。



客户端按照顺序读取各个数据块。当客户端不断读取数据流时，在数据块的边界， DFS Input Stream不断创建到保存其他数据块的DataNode 的连接。DFS Input Stream 同 时向NameNode 询问和提取下一批数据块的DataNode 的位置信息。

(6)当客户端完成文件的读取，它调用FS Data Input Stream实例的close() 方法。

**12.2.3 Secondary NameNode 介** **绍**

NameNode 集 中 存 储 了 HDFS 的 元 信 息 (Metadata) 。 它 负 责 执 行 命 名 空 间 (Namespace) 的一些操作，比如创建、删除、修改和列出所有文件和目录 (List) 。 它 还 执行数据块的管理操作，包括把文件映射到所有的数据块、创建和删除数据块、管理副本 的放置 (Placement) 和进行重新复制操作 (Re-Replication) 等。

除此之外， NameNode 负 责 DataNode 的成员管理 (Membership), 即接受其注册 (Registration) 和周期性的心跳信息 (Heart Beat)等。客户端和 HDFS 的数据传输是在 客户端和DataNode 之间进行的，数据传输不经过 NameNode。

为了支持高效的存取操作，NameNode 把所有的元信息保存在主内存中，包括文件和 数据块的命名空间、文件到数据块的映射、每个数据块副本的位置信息等。文件命名空间 和文件到数据块的映射信息，也持久化到 NameNode 的本地文件系统。FsImage 文件保存 整个文件系统的命名空间和文件到数据块的映射信息。 EditLog 文件则是一个事务日志 (Transaction Log) 文件，记录了对文件系统元信息的所有更新操作，比如创建文件、改 变文件的复制因子 (Replication Factor) 等。当 NameNode 启动时，它读取FsImage 和 EditLog 文件，把EditLog 的所有事务日志，应用到从FsImage 文件装载的旧版本的元信 息上，接着把新版本的Fslmage 保存到磁盘，并且截短 EditLog。

**由于NameNode 保存了文件系统的所有元信息，所以只有** **NameNode 才知道如何从** **DataNode的各个数据块重建一个文件。NameNode 宕机，将引起整个** **Hadoop 系统宕机，** 因此**，NameNode 成为Hadoop 文件系统单点失败** **(Single Point of Failure) 的瓶颈。**

**由于** **NameNode 如此重要，所以需要采取措施，保证NameNode 能够从失败中恢复。** **Hadoop 提供了两种机制来保证** **NameNode 的可恢复性。**

**第一种方式是把文件系统元信息的持久化版本备份起来** **(Backup) 。** **可以配置** **Ha-** do**op, 使得NameNode 写入元信息的持久化版本时写入到多个文件系统里。这些写入操作** **是同步的和原子化的** **(Atomic)。** **一般可以把元信息写人本地磁盘以及挂接** **(mount)** **为** **本地目录的网络文件系统** **(Network** **File** **System,NFS)。**

**第二种方式是运行一个** **Secondary NameNode, 它负责周期性地使用** **EditLog 更新** **FsImage, 保持EditLog 在一定规模之内。具体来讲，Secondary NameNode 保存了Name-** N**ode 的** **FsImage 文件和** **EditLog 文件的副本。每隔一段时间(缺省为一个小时),它从** **NameNode 拷贝元信息的映像文件，并且把映像文件和日志文件进行合并，然后把映像文** **件复制回** **NameNode,** **以便** **NameNode** **获得更新的元信息映像文件。**

假设NameNode 发生宕机状况，可以启动其他的机器，从Secondary NameNode 获得 元信息映像文件和日志文件，可以恢复发生宕机之前的最新的元信息，我们可以把这台机 器当作新的 NameNode 来使用。

**Secondary NameNode需要运行在另外一台机器上，因为它需要和主** **NameNode 一样** **规模的CPU 计算能力和内存空间，以便完成元信息的管理。**

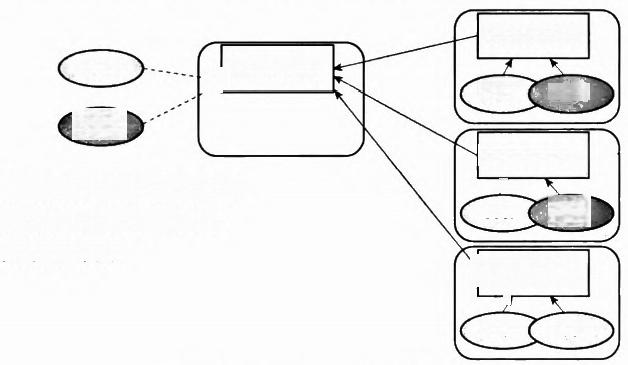


**1 2 . 3 M a p R e d u c e工** **作** **原** **理**

我们从两个方面来了解 MapReduce: 一个方面是MapReduce 作 业 (Job) 是如何运行 的；另一个方面是MapReduce 编程模型如何把一个计算任务表达成一个Map 函数和一个 Reduce 函数。

**12.3.1 MapReduce 执行引擎**

MapReduce 执行引擎运行在分布式文件系统 HDFS 之上，它包括JobTracker 和 TaskTracker 两个主要的组成部分，分别运行在NameNode 和 DataNode 上。用户提交的 数据处理请求称为一个作业 (Job), 由 JobTracker 分解为数据处理任务 (Task), 分发给 集群里的相关节点上的 TaskTracker 运行(见图12 - 4)。



TaskTracker

**client**

Task **Task**

client

TaskTracker

实线箭头：MapReduce Status

Task

圆角框：节点

TaskTracker

**Task Task**

虚线箭头：Job Submission

JobTracker

Task

**图12-4** **MapReduce 执行引擎**

客户端程序把作业提交给JobTracker 以 后 ，JobTracker 把数据处理任务发送给整个 集群各个节点的TaskTracker 。 发送任务的原则是尽量把任务推送到离数据最近的节点上 运行，甚至是推送到数据所在的节点上运行。

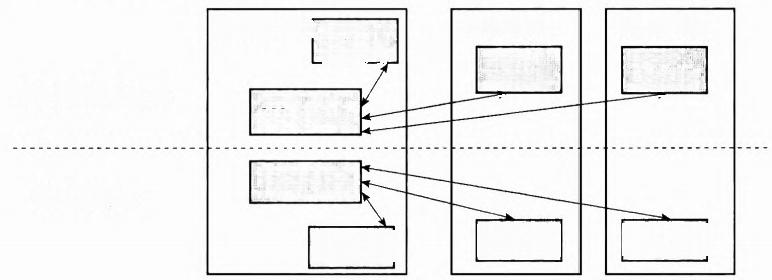
在 HDFS 里 ，JobTracker 通过HDFS NameNode, 知道哪些节点包含将要处理的各个 数据块，也就是它了解数据块的存放位置 (Location)。 如果任务 (Task) 不能发送到数 据块所在的节点，比如因为该节点目前 Task Slot (任务槽，即每个TaskTracker 可以运 行的task 数量)已经用完，那么系统优先把任务推送到同一机架里的其他节点，该节点 保留了数据块的另外一个副本 (Replica) 。 这样的任务分发策略避免或者减少了数据的传 输 (Data Transfer), 进而减少集群核心骨干网络 (Backbone Network) 上的网络流量。

如果TaskTracker 失败或者运行超时，它负责的任务被重新调度到其他的 Task- Tracker 上。在 TaskTracker 运行过程中，它向JobTracker 每隔几分钟发送一个心跳信号 (Heart Beat), 以便报告它的存活状态。 JobTracker 和 TaskTracker 的状态信息，通过内 置的一个<HTTP> 服务器Jetty 报告出来，可以通过浏览器进行查看。



在 Hadoop 0.20以前的版本， JobTracker 失败以后，所有的数据处理操作都丢失了。 从0.21版本开始，Hadoop增加了作业处理过程的检查点功能 (Checkpointing) 。Job- Tracker 在文件系统里，记录当前作业进展到什么程度。当新的JobTracker 启动以后，它 可以根据这些检查点信息，从上次检查点位置，继续数据处理工作，而不是从头开始。这 个功能改善了作业的调度效率。

图12 - 5把分布式文件系统 HDFS 和 MapReduce 执行引擎的关系清晰地展示出来。 MapReduce 和 HDFS 运行在同 一个集群上，它们是同 一个集群上运行的不同软件模块， 分别提供数据存储和数据处理功能。



Master Node[ Task

Tracker

Job

Tracker

NameNode

HDFS Layer

DataNode

Slave Node

Task

Tracker

Slave Node

Task

Tracker

MapReduce Layer

DataNode

DataNode

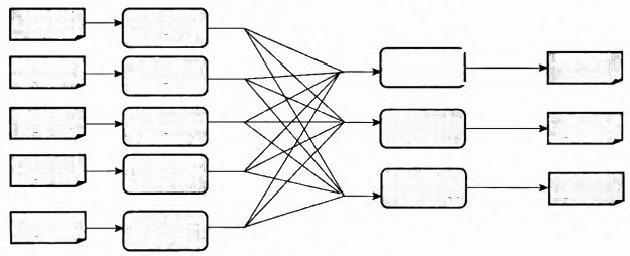
**图12-5** **HDFS与MapReduce的关系**

**12.3.2** **MapReduce** **计算模型**

在 MapReduce 计算模型中，数据以键值对(<Key,Value>) 进行建模。几乎所有的 数据都可以使用该数据模型进行建模， Key 和 Value 部分可以根据需要保存不同的数据类 型，包括字符串、整数或者更加复杂的类型。

MapReduce 并行编程模型把计算过程分解为两个主要阶段，即Map 阶 段 和Reduce 阶 段 。MapReduce 程序的具体执行过程如图12 - 6所示。首先，保存在 HDFS 里的文件即 数据源已经进行分块。这些数据块交给多个 Map 任务去执行， Map 任务执行Map 函 数 ， 根据特定规则对数据进行处理，写入本地硬盘。 Map 阶段完成后，进入 Reduce 阶 段 ，

Reduce 任务执行Reduce 输人

split0 split1

split2 split3

split4

函数，具有同样 Key

Map 任务 Shuffle

map()

map()

map()

map()

map()

值的中间结果从多个Map 任务所在的节点 Reduce任务 输出

|  |  |
| --- | --- |
| Reduce()  Reduce()  Reduce() | part0  part1  part2 |

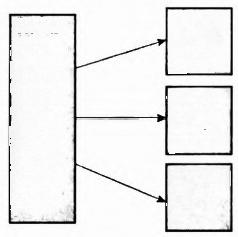
**图12-6** **MapReduce 计算过程**

被收集到一起 (Shuffle) 进行约减处理，输出结果写入本地硬盘(分布式文件系统)。程 序的最终结果可以通过合并所有 Reduce 任务的输出得到。在这里需要注意的是，输入数 据、中间结果以及最终结果，都是以〈Key,Value〉 的格式保存到分布式文件系统中，即 HDFS 中。

Map 函数处理〈Key,Value> 对，产生一系列的中间 (Key,Value> 对 。Reduce 函数用来 合并所有具有相同Key值的中间键值对，计算最终结果。MapReduce计算模型可以形式化地表 达成Map:<k1,v1>→list<k2,v2>,Reduce:<k2,list(v2)>→list(k3,v3>。

下面通过一个实例解释Map 函数和Reduce 函数如何对数据进行操作，以及MapRe- duce 程序如何对整个数据文件进行处理。

第一个实例是Word Count程序，它对整个文件里出现的不同单词进行计数。Map 函数 的功能是对文件块出现的每个单词输出〈单词，1〉的键值对，如图12-7所示。Reduce 函 数则把各个Map 函数输出的结果，按照单词进行分类，统计其出现的次数，如图12-8所示。



Smith,1 Bob,1

Brian,1

Carl,1 Carl,1 Brian,1

Smith,1 Carl,1 Bob,1

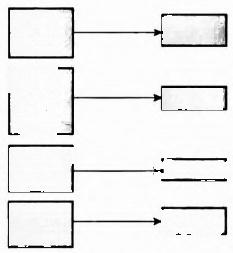
**图12** **-** **7** **Word** **Count的** **Map** **函数功能**

Smith

Bob Brian Carl Carl Brian

**Smith Carl**

**Bob**



Bob,1 Bob,1

Carl,1

Carl,1

Carl,1

Smith,1 Smith,1

Brian,1 Brian,1

**图12-** **8** **Word** **Count的** **Reduce函数功能**

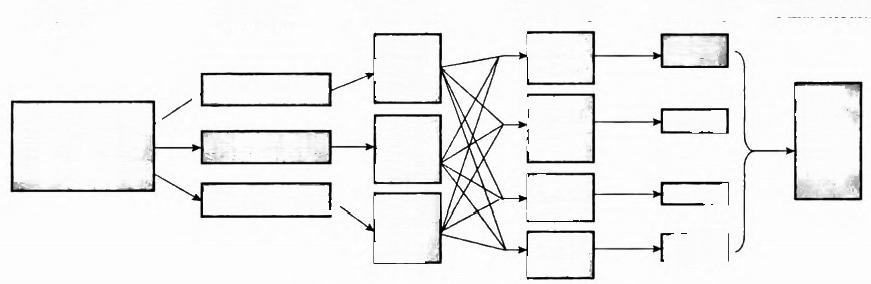
**Bob,1**

Carl,3

Smith.2

Brian.2

MapReduce 执行引擎在执行 Word Count 程序时，JobTracker 接收了Word Count 程 序以后，根据文件的数据块所在的节点，在这些节点上启动TaskTracker 运 行Map 函数， Map 函数执行完毕，结果存放在各个节点的本地文件里。接着JobTracker 在各个节点上 启动 TaskTracker 运行Reduce 函数，这些任务从各个Map 任务执行的各个节点上，把具 有相同Key 值(即相同单词)的中间结果收集到一起，就能够汇总出各个单词的计数。整 个过程如图12-9所示。



Splitting

Bob,1 Bob,1

Smit Bob Brian

Carl Carl Brian

Smith Carl Bob

Brian,2

Mapping

Smith,1 Bob,1 Brian,1

Carl,1

Carl,1

Brian,1

Smith,1 Carl,1

Bob,1

Carl,1 Carl,1 Carl,1

Smith,1 Smith,1 Brian,1 Brian,1

Smith Bob Brian Carl Carl Brian Smith Carl Bob

Bob,2

Carl,3

Smith,2 Brian,2

Reducing

**Bob,2**

Final Result

Shuffling

Smith,2

Carl,3

Input

**图12** **-** **9** **Word Count的执行过程**

我们再举一个实例。排序 (Sort) 是数据处理的一个重要操作，假设有一个大的成绩

**表.需要对其根据总分进行排序。这个表的** **Key 值包含学号，Value 值包含姓名、性别、**

出生年月、语文成绩、数学成绩、英语成绩、物理成绩、化学成绩等。

在Map 函数的设计上。我们把成绩文件块进行这样的处理，即把每个键值对，根据 其总成绩(即各科成绩的总和，是一个临时计算的值)落在某个范围，这些范围包括 (200,300),(300,400),(400,500),(500,600),(600,700)(假设成绩没有低于 200分的，也没有高于700分的),进行这样的映射，当总成绩落在200～300分时，原来 的键值对(<Key,Value>) 映射成〈Key=1,Value= 学号，姓名、性别、出生年月、语 文、数学、英语、物理、化学、总分>,当总成绩落在300～400分时，原来的键值对 (<Key,Value>) 映射成〈Key=2,Value= 学号，姓名、性别、出生年月、语文、数学、 英语、物理、化学、总分> ……。Reduce 函数设计成在一个Reduce 任务中，对某个分数 段的所有学生记录，根据总成绩排好序，然后把结果输出到文件中保存。最后，把各个 Reduce 任务的输出连起来，就是我们想要的根据总成绩排序的成绩表。

如果每个Reduce 任务接收到的学生记录太多，负载过重，还可以把总成绩的分数段 划分得更细一点，比如以每50分或者每20分作为一个分数段等，同时增加 Reduce 任务 的数量，以提高处理速度。

**12.3.3 Hadoop 1.0 的** **应** **用**

MapReduce 计算模型是由Google 提出来的，Hadoopl.0 则是对MapReduce 计算模型 的开源实现。根据上述描述，MapReduce 计算模型看起来特别简单。实际上，在这种简 单的处理之上，可以实现更加复杂的数据处理任务。

除了简单的SQL 汇总之外，研发人员已经把OLAP 、 数据挖掘、机器学习、信息检 索、多媒体数据处理、科学数据处理、图数据处理等复杂的数据处理和分析算法移植到 Hadoop 平台上(翻译成MapReduce Job) 。Hadoop 不仅仅是一个处理非结构化数据的工 具，当数据按照一定格式进行组织以后， Hadoop 平台也可以处理结构化数据。 Hadoop 平台以及 Hadoop 上的各种工具构成了一个生态系统，完成各种大数据集的处理任务。

**12.4 Hadoop生态系统**

在分布式文件系统 HDFS 和 MapReduce 计算模型之上，若干工具一起构成了整个 Hadoop 生态系统。下面我们首先对这些组件进行简单介绍，然后重点介绍 Hive。

Hive 是 Hadoop 平台上的数据仓库，用于对数据进行离线分析。它提供了一种类似于 SQL 的查询语言 HQL(Hive Query Language) 。Hive 将 SQL 转化为MapReduce 作业 (Job) 在 Hadoop 上执行。

HBase 是Google Big Table在 Hadoop 平台上的开源实现。它是一个针对结构化数据 处理的面向列分组 (Column Family) 的、可伸缩的、高度可靠的、高性能的分布式数据 库。一般用于数据服务 (Data Serving) 应用场合。

Pig 实现了数据查询脚本语言Pig Latin。用Pig Latin 脚本语言编写的应用程序，翻译 为MapReduce 作业 ·在 Hadoop 上运行。按照 MapReduce 计算模型来编写某些数据处理

任务，比如表格之间的连接操作过于烦琐。Pig Latin 提供了连接操作的原语，还提供了其 他数据操作原语，方便开发人员编写数据操作算法。像Hive 一 样，Pig 一般用于离线分 析。两者的主要区别是，Hive 使用声明性 (Declarative) 的语言HQL,Pig 使用过程性 (Procedure) 的语言Pig Latin 。Apache Hadoop生态系统如图12-10所示。

Flume bood

Log Collector Data Exchange

Zookeeper

Coordinator

Oozie

Workflow

Script

Mahout

Machine Learming

R Connectors

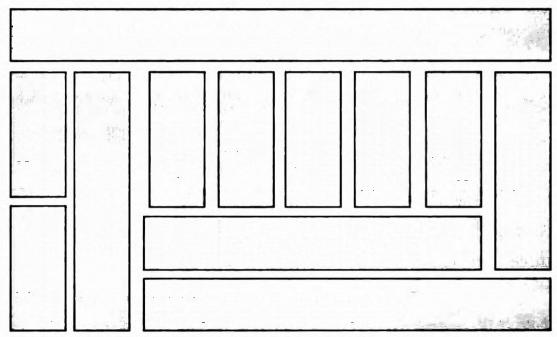
Statistics

Hive

SQL query

Base

Column Storage



Ambari

**Provisioning,Managing and Monitoring Hadoop Clusters**

MapReduce to MapReduce V2(YARN) Distributed Processing Framework

HDFS

**Hadoop Distributed File System**

**图12-10** **Apache Hadoop生态系统**

Flume 是一个可扩展的、高度可靠的、高可用的分布式海量日志收集系统， 一般用于 把众多服务器上的大量日志聚合到一个数据中心。Flume 提供对日志数据进行简单处理的 能力，比如过滤、格式转换等。同时，Flume 还具有将日志写往各种目标(本地文件、分 布式文件系统)的能力。

Sqoop 是 SQL to Hadoop的缩写，主要用于在关系数据库或者其他结构化数据源和 Hadoop 之间交换数据。比如，Sqoop 可以把MySQL 等数据库数据导入到 Hadoop 里，包 括 HDFS,HBase 以及 Hive 。反过来，它也可以将Hadoop 的数据导出到MySQL 数据库 中。数据的导入导出通过MapReduce 作业(应用程序)实现，充分利用了MapReduce 的 并行化处理能力和容错性能。

Mahout 是 Hadoop 平台上的机器学习软件包，它的主要目标是实现高度可扩展的机 器学习算法，以便帮助开发人员在大数据上运行机器学习软件。 Mahout 现在已经包含聚 类、分类、推荐引擎(协同过滤)、频繁集挖掘等经典数据挖掘和机器学习算法。

Oozie 是一个工作流调度器 (Scheduler) 。Oozie 协调运行的作业，属于一次性非循环 的作业 · 比如MapReduce 作业、Pig 脚本、Hive 查询、Sqoop 数据导入导出作业等。 Oo- zie 基于时间和数据可用性进行作业调度，根据作业间的依赖关系协调作业的运行。

Zookeeper 是模仿Google 公司的Chubby 系统的开源实现，Chubby 是一个分布式的 锁 (Lock) 服务。大部分分布式应用都需要这样一些公共服务，包括树状结构的统一命名 服务、状态同步服务(通过分布式共享锁)、配置数据的集中管理、集群管理(如集群中 节点的状态管理及状态变更通知，节点数据变更的消息通知)等。这些服务难以实现，也 难以调试。借助于Zookeeper, 人们就无须为每个应用程序实现这些功能，降低分布式软 件框架实现的复杂度，加快分布式应用的开发和部署。在由一个Master 节点和多个Slave 节点组成的分布式软件框架中，单一的Master 节点有可能导致单点失败，影响整个系统 的可靠性。如果把Master 节点用Zookeeper 管理的若干Master 节点来(其中一个节点是 Active Master) 代替，就不必担心单点失败问题了。如果Active Master 失败了，Zoo- keeper 可以挑选其他Master 来顶替它。

此外，Scribe 是 Facebook 公司开发的开源的分布式日志搜集系统，其架构简单，日 志格式灵活，并且支持异步发送消息和队列。Scribe 非常适用于收集日志数据、分析用户 行为的应用场合。DataX 是来自Taobao 的开源的数据导入导出的工具，支持 HDFS 集群 与各种关系数据库之间的数据交换。

Hadoop 及其生态系统实现了单一平台多种应用。传统的关系数据库管理系统擅长处 理关系型数据，支持单一的应用，因此是单一平台单一应用。各类 NoSQL 数据库软件， 使用不同的数据模型和存储格式，针对不同的应用场景，属于多平台多应用。 Hadoop 生 态系统，在底层利用HDFS 分布式文件系统实现各种数据的统一存储，上层有很多组件/ 工具，实现各种数据管理和分析功能，满足各种应用场景的要求。

**Hive原理**

Hive 是基于Hadoop 的一个数据仓库系统，它将结构化的数据文件映射为一张数据库 表.并提供完整的SQL 查询功能。Hive 把 Hive Query Language(HQL, 类似SQL 的查 询语言)所编写的语句转换为MapReduce 作业 (Job), 然后在Hadoop 平台上运行。使用 Hive, 用户通过类似 SQL 的查询语言，快速实现简单的数据分析和统计，不必开发专门 的 MapReduce 应用程序，学习成本低。大量熟悉SQL 语言的开发者、数据分析人员，都 可以很快地学会使用Hive进行数据分析。

相对于传统的关系数据库，Hive 具有如下特点：

**(1)Hive 使用** **Hadoop 的** **HDFS 分布式文件系统存储数据库数据，关系数据库则是** 使用服务器的本地文件系统存储数据库数据。

**(2)Hive 使用** **MapReduce计算模型实现查询处理，关系数据库则是使用自行设计的** 查询处理模型，比如 Shared Nothing无共享架构下的并行查询处理模型。

**(3)关系数据库主要为高性能查询而设计，** **Hive 的设计目标则是对海量数据进行分** **析处理，其实时性没有关系数据库好，导致Hive 和关系数据库的应用场景是不同的。**

**(4)基于Hadoop 平台，Hive 具有强大的扩展能力，可以很容易地对存储能力和计算** **能力进行扩展。关系数据库在扩展性方面则差很多，目前没有关系数据库系统能够扩展到** **1000台以上的集群规模。**

**(5)关系数据库在数据加载时，要求数据必须符合数据库表的模式，否则不能加载**， 称为“写时模式”(Writing Time Schema) 。Hive 在数据加载时，无须进行模式检查，在 读取数据时、再对数据以一定的模式进行解释。称为“读时模式” (Reading Time Sche- ma)。

(6)关系数据库可以对某行数据进行更新、删除等操作，并且支持事务处理。 Hive 不支持某个具体数据行的修改，它只支持覆盖原数据和数据的追加功能。 Hive 也不支持 事务和索引。对于更新、事务、索引等关系数据库系统的主要特征， Hive 并未支持，因 为Hive 的设计目标是对海量数据进行分析处理， 一般需要对数据进行全部扫描。对于更 新操作.需要将原来表格里的数据进行更改(转换)后，写入一个新的表格。

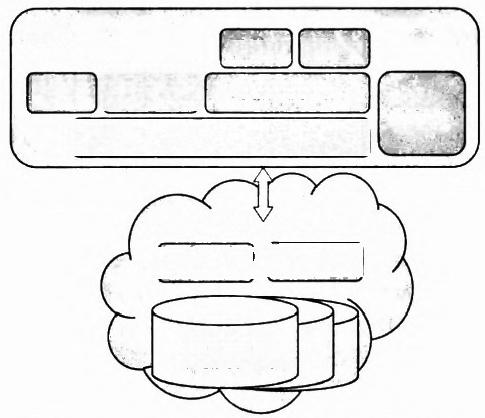
Hadoop 平台(包括HDFS 和MapReduce) 是 Hive 数据仓库的基础。Hive 系统包括如下 组件：命令行接口CLI(Command Line Interface) 、JDBC/ODBC驱动程序、Thrift Server 服 务器、WEBGUI(Graphic User Interface) 图形用户界面、元信息管理器 MetaStore 和 Driver (Complier 、Optimizer 和Executor) 等。这些组件可以分为两大类，即服务端组件和客户端

组件。

Hive 的服务端组件包括：(1) Driver 。该组件包括编译器 (Complier) 、 优化器 (Op- timizer) 和执行器 (Executor), 它的作用是将用户写的 HQL 查询语句进行解析、编译、 优化，生成执行计划，然后调用底层的MapReduce 计算模型来执行。(2)元信息管理器 MetaStore 。 是元数据管理组件，负责存储 Hive 的元数据。 Hive 的元数据存储在关系数 据库里，Hive 支持的关系数据库有 Derby,MySQL 等。元数据对于Hive 的正确运行可 谓举足轻重。 Hive 支持把MetaStore 服务独立出来，安装到远程的服务器集群里，从而解 耦 Hive 服务和 MetaStore 服务，保证 Hive 运行的健壮性。(3) Thrift Server 服务器。 Thrift Server 服务器是Facebook 开发的一个软件框架，用于开发可扩展的、跨语言的服 务接口。Hive 集成了Thrift Server 服务器，能让不同的编程语言调用Hive 的接口。

Hive 的客户端组件包括：(1)命令行接口CLI。 命令行接口允许用户交互式地使用 Hive 。(2)Thrift 客户端。包括JDBC 和 ODBC 驱动程序。(3) WEB GUI图形用户界面。 Hive 客户端提供了一种通过网页访问Hive 所提供的服务的方式。通过WEB GUI图形用 户界面访问Hive, 必须首先启动HWI(Hive Web Interface) 服务。

Hive 与 Hadoop 的关系以及Hive 的组件见图12- 11。



HIVE

JDBC ODBC

CLI weblimefiel

Daiver(Compiler,pimizer.Excutor)

Hadoop(HDFS+MapReduce)

Job Tracker NameNode

DataNode

+Task Tracker

**Meta Store**

Thrit Server

**图12-** **11** **Hive 与Hadoop的关系以及** **Hive的组件**

Hive 支持的数据类型包括整型 (Integer) 、 浮点型 (Float) 、 双精度浮点型 (Double) 以及字符串 (String) 等 。Hive 还支持更加复杂的数据类型，包括映射 (Map) 、 列 表 (List) 和结构 (Struct) 。 这些复杂类型可以通过嵌套表达更加复杂的类型。除此之外， Hive 允许用户自己定义类型 (Type) 和函数 (Function) 来扩展系统。

Hive 查询语言 (Hive Query Language,HQL) 和标准的SQL 语言非常相似，很容 易被熟悉SQL 语言的开发者和数据分析者理解和掌握。

Hive 使用传统数据库使用的表格 (Table) 、 行 (Row) 、 列 (Column) 、 分 区 (Parti- tion) 等概念，易于理解。 Hive 的数据模型包括几个主要的管理层次，分别是数据库 (Database) 、 表 格 (Table) 、 分 区 (Partition) 、 桶 (Bucket) 等。

(1)数据库 (Database) 。 相当于关系数据库里的命名空间 (Namespace) 。 它的作用 是将不同用户的数据，隔离到不同的数据库和模式中。

(2)表格 (Table) 。Hive 的表格逻辑上由存储的数据和描述数据格式的相关元数据 组成。表格的数据存放在分布式文件系统里，即HDFS, 元数据存储在关系数据库里。当创 建一张Hive 表格，还没有为表格加载数据时，该表格在分布式文件系统(即HDFS) 上就是 一个文件夹(文件目录)。Hive 里的表格分为两种类型：一种叫托管表，这种表的数据文件 存储在Hive 的数据仓库里；一种叫外部表，这种表的数据文件可以存放在Hive 数据仓库外 部的分布式文件系统上，也可以放到Hive 数据仓库里。Hive 的数据仓库就是HDFS 上的一 个目录，这个目录是Hive数据文件存储的默认路径，相关元信息会存放到元数据库里。

(3)分区 (Partition) 。Hive 里分区的概念是根据“分区列”的值，对表格的数据进 行粗略划分的机制。在 Hive 存储上，表格的主目录 (Hive 的表格对应一个文件夹)下的 一个子目录，它的名字就是我们定义的分区列的名字。分区列不是表里的某个实际字段而 是一个虚拟的属性列，根据这个列，划分和存储表格里的数据。使用分区是为了加快查询 速度而设计的，如果我们的查询仅仅需要存取部分数据，就没有必要进行全表扫描。

( 4 ) 桶 (Bucket)。Table 和 Partition都是目录级别的数据拆分。使用桶的表格，将 数据文件按一定规律拆分成多个文件，每个桶就是表目录(或者分区子目录)里的一个文 件。数据的分桶一般通过 Hash 函数实现。创建表时，用户指定需要的桶的数量，以及使 用哪个数据列进行分桶操作 (Bucket the Data)。

当用户执行一个采样 (Sample) 查询时，Hive 可以使用分桶信息，有效地裁剪数据 (Prune Data)。比如一个数据库表，每个(子)目录下有32个桶，可以通过提取每个目录 下的第一个桶文件，生成一个1/32数据量的采样。

**12.5 Hadoop 2.0版(YARN)**

**12.5.1 Hadoop 1.0 的优势和局限**

Hadoop 1.0最重要的优势是它的可扩展性 (Scalability)。在实际应用中，Hadoop 已 经 被部署到超大规模的集群上(超过3000个节点),对于传统的RDBMS来讲是无法想象的。

为什么我们需要这么大规模的集群对数据进行处理呢?因为当数据规模极大时，我们 需要考虑扩展性的代价以及I/O 瓶颈等因素。

第一个因素是我们是否很方便地对系统进行扩展 (Cost to Scale)。由 于Hadoop 能够 运行在由普通服务器构成的超大规模集群上，所以 SQL on Hadoop 系统，比起传统的 MPP 数据库比如 TeraData,Vertica,Netezza 等来讲，具有更强的扩展能力。传统的 MPP 数据库系统，需要运行在昂贵的高端服务器上，很难扩展到上千个节点。

第二个因素是处理大数据时的I/O 瓶颈。当数据的规模足够大，只有一部分数据可以 装载到内存中，剩下的数据必须保存在磁盘里，并且在处理过程中，从磁盘上不断装载到 内存中，进行后续处理。通过把I/() 分散到大规模集群的各个节点上，我们可以把数据装 载的速度大大提高，进而加快后续的处理。大规模集群把各个节点的I/O 带宽聚集起来， 获得比高端服务器大得多的I/O 带宽，可以达到几十倍甚至上百倍，这无疑是一个廉价有 效的大数据处理解决方案。

**2008年，雅虎公司使用一个拥有910个节点的Hadoop集群，在209秒内完成了1TB 数据的排序，打破了Terabyte Sort 评测基准的纪录(之前的纪录是297秒)。这件事的重** **要意义在于这是用Java 编写的开源程序首次赢得Terabyte Sort 评测基准。**

2011年3月， Media Guardian 媒体集团把年度创新奖 (Innovation Awards of the Year) 颁发给了Hadoop 项目。评审委员会认为， Hadoop 项目是“21世纪的瑞士军刀” (Swiss Army Knife of the 21st Century) 。Hadoop平台已经成为大数据处理的事实上的标 准工具，它的重要作用被越来越多的人认识。

虽然 Hadoop 已经在处理大数据方面获得了巨大的成功，但是它也有一些明显的缺 点。对于Hadoop 1.0 来讲，主要的局限包括：(1)它仅仅支持一种计算模型，即Ma- pReduce 。MapReduce 计算模型的表达能力有限。复杂的数据处理任务，比如机器学习算 法和SQL 连接查询等，很难表达为一个 MapReduce Job, 而是需要翻译成一系列的Ma- pReduce Job, 这些Job 一个接一个地执行。(2) MapReduce 作业在Map 阶段和Reduce 阶段执行过程中，需要把中间结果存盘，而且在 MapReduce 作业间，也需要通过磁盘实 现作业之间的数据交换，通过磁盘进行数据交换，效率低下，影响了查询的执行效率。在 这个计算模型上，很难再继续减小查询的响应时间。(3) Hadoop 1.0的任务调度方法远 未达到优化资源利用率的效果。在 Hadoop 1.0版中，对任务的调度方法，即如何给 TaskTracker 分配任务的过程比较简单。每个 TaskTracker 拥有 一 定数量的任务槽 (Slot), 每个活动的Map 或者Reduce 任务占用其中一个任务槽。JobTracker 把工作分配 给最靠近数据的TaskTracker, 这个TaskTracker 正好有可用的任务槽。在这个调度方法 下，并未考虑将要被分配任务的机器，其当前的系统负载是否过高。如果某个 Task- Tracker执行非常慢，它将会影响整个MapReduce 作业 (Job) 的执行，整个作业将会等 待最慢的任务完成才能结束。当然，可以通过猜测执行 (Speculative Execution) 模式， 在多个 Slave 节点上启动同一个任务 (Task), 只要其中一个任务完成即可。

**12.5.2** **业务需求推动持续创新**

Hadoop 在大数据处理领域展现了强大的威力。人们自然希望把不同来源的数据，不 管它是结构化的还是非结构化的，都整合起来，保存到一个地方。在这些数据上面，执行 各种各样的分析。

Hadoop 最初是为大数据的批处理设计的，它的关注点在于以尽量高的吞吐量处理这 些数据，但是人们希望 Hadoop 还能够支持交互式查询、数据的迭代式处理以及流数据处 理、图数据处理等。其中，迭代式处理是机器学习算法所必需的，因为机器学习算法一般 需要对数据进行多遍扫描和处理。

在这种形势下，Hadoop 2.0应运而生。由于Hadoop 2.0 的主要改变是在整个软件架构 里划分出了资源管理框架YARN(Yet Another Resource Negotiator),YARN 是 Hadoop 2.0 的主要组成部分。因此，在下文中，我们有时把YARN 和 Hadoop 2.0互换使用。

**12.5.3 YARN 原** **理**

Hadoop YARN 是 Hadoop 2.0的最重要的组成部分。 YARN 把资源管理 (Resource Management) 和作业调度/监控 (Job Scheduling/Monitoring) 模块分开。在Hadoop 1.0

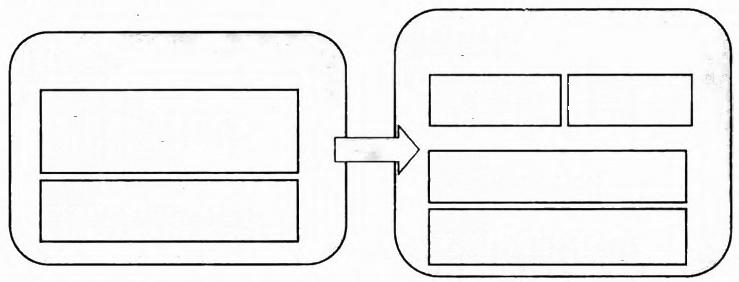


版中，这两个功能都由JobTracker

来负责。



在 Hadoop 1.0中，系统仅仅能够支持一种计算模型，即MapReduce 。 在 Hadoop 2.0 中，系统可以支持更多的计算模型，包括流数据处理、图数据处理、批处理、交互式处理 等。在Hadoop 2.0中，应用程序可以是传统的MapReduce 作 业 (Job), 或者由 一 系列任 务构成的一个有向无环图DAG(Directed Acyclic Graph) 表达的作业，能够表达更加复 杂的数据处理流程(见图12- 12)。



Hadoop 2.0

MapReduce Others

data processing data processing

YARN

resource management

HDFS

redundant,reliable storage

Hadoop 1.0

MapReduce

resource management & data processing

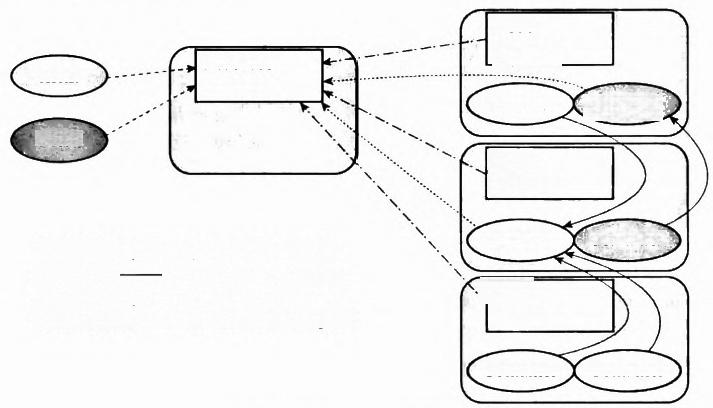
HDFS

redundant,reliable storage

**图12-** **12** **Hadoop 2.0大数据处理框架里的MapReduce**

简而言之，通过把资源管理功能单独划分出来，由Resource Manager进行管理， Ma- pReduce 在 Hadoop 2.0中，仅需要完成其擅长的工作，即批量数据处理。在Hadoop 2.0 之上，可以运行其他类型的应用，它们使用的是同一个资源管理模块。

图 1 2 - 1 3 展 示 了 Hadoop 2.0的主要组件及其关系，图12 - 14展示了作业的调度过 程。在新的架构里，包含 Resource Manager和 Node Manager 两个重要的组件。 Resource Manager 运行在Master 节点上， Node Manager运 行 在Slave 节点上， 一起负责分布式应 用程序的调度和运行。在 Hadoop 2.0平台上，应用程序包括 MapReduce 作 业 、Hive 查 询 、Pig 脚本或者Giraph 查询等。



Resource Manager

**client**

长虚线箭头(----→):Job Submission 实线箭头(— →):MapReduce Status 点划线箭头(-- →):Node Status

短虚线箭头( →): Resource Request

Node

Manager

Container

Node

Manager

App Mstr

Node

Manager

Container

**App Mst**

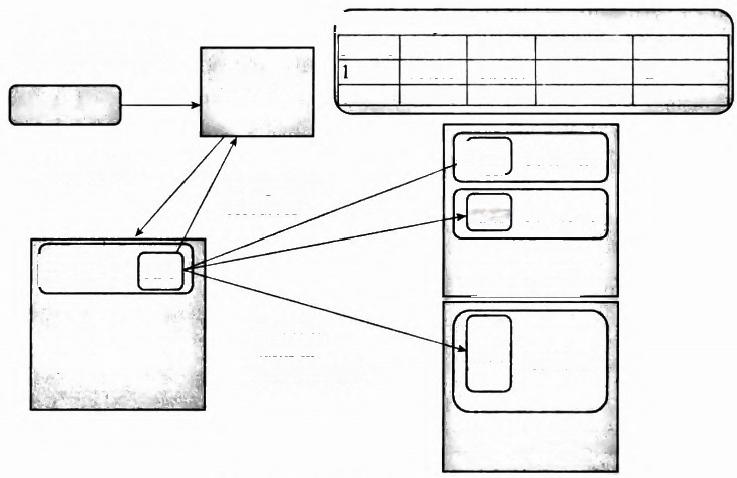
Container

Container

client

**图12-** **13** **Hadoop 2.0组件及其关系**





Resource request

priority host loc resource

Host1 Rack1 1GB+1core 2 Host2 Rack2 2GB+1core

**Client**

task Container

3.negotiate

task Container

Node Manager

(host1,rack1)

|  |  |
| --- | --- |
| 4.launch  tasks in  thecontainers | task Container  **Node Manager**  **(host2,rack2)** |

Container AM

Node Manager (hostM,rackN)

#containers 2

1

Resource Manager

1.run applications

2.start AM

resources

**图12-14** **Hadoop 2.0的作业调度**

Resource Manager 是为应用程序分配资源的最高权威。 Resource Manager 包含两个

组件，包括Scheduler 和 Application Manager 。Scheduler负责为应用程序分配资源，它根

据应用程序的资源需求以及一些限制条件，包括各个用户的限额等，完成资源的分配和调 度 。Scheduler 使用资源容器 (Container) 的概念，把 CPU 、 内存、磁盘、网络带宽等资 源整合起来。

Application Manager 接受客户端应用程序提交的作业，从Scheduler 为该应用程序申 请第一个容器，运行针对这个应用程序的Application Master, 用于执行提交的作业(应 用程序),并且在发生失败的情况下，重新启动这个应用程序的Application Master 。Ap- plication Master从 Scheduler 为应用程序申请资源，和 Node Manager 一 道在分布式环境 下执行应用程序，追踪其状态，监控作业的进展情况。

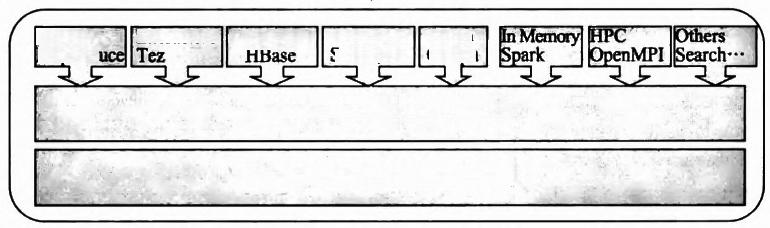
Node Manager 运 行 在Slave 节点上，它为应用程序启动容器，监控其资源使用情况 ( 包 括CPU 、 内存、磁盘、网络带宽的使用情况),并且把这些信息报告给 Resource Man- ager, 如图12 - 14所示。

**12.5.4** **YARN** **的优势**

相对于 Hadoop 1.0,Hadoop 2.0(YARN) 具有如下的主要优势：(1)扩展性。 Re- source Manager的主要功能是资源的调度工作，它能够轻松地管理更大型的集群系统，适 应了数据量增长对数据中心的扩展性提出的挑战。(2)更高的集群使用效率。 Resource Manager是一个单纯的资源管理器，它根据资源预留要求、公平性、服务水平协议 (Service Level Agreement,SLA) 等标准，优化整个集群的资源，使之得到很好的利用。

( 3 ) 兼 容Hadoop 1.0。在 Hadoop 1.0 平台上开发的 MapReduce 应用程序，无须修改， 直接在YARN 上运行。(4)支持更多的负载类型。当数据存储到 HDFS 以后，用户希望 能够对数据以不同的方式进行处理。除了MapReduce 应用程序(主要对数据进行批处

理),YARN 支持更多的编程模型，包括图数据的处理、迭代式计算模型、实时流数据处 理、交互式查询等。很多的机器学习算法需要在数据集上经过多次迭代获得最终的计算结 果。(5)灵活性。 MapReduce 等计算模型可以独立于资源管理层，单独演化和改进，使 得系统各个部件的演进和配合，更加具有灵活性(见图12-15)。

Streaming Storm, S4

Interactive

Batch

MapRed

Grapb

Online

Giraph

YARN(Resource Management)

HDFS2(Redundant,Reliable Storage)

**图12** **-** **15** **Hadoop 2.0支持更多的负载类型(应用类型)**

**12.6 Hadoop 2.0上的交互式查询引擎Hive on Tez**

**12.6.1 Tez 原** **理**

Apache Hadoop是对大数据进行批处理的事实上的标准工具，相对于传统的基于 SAN 的集群方案，Hadoop 平台提供了高度的横向扩展能力和更低的系统价格。

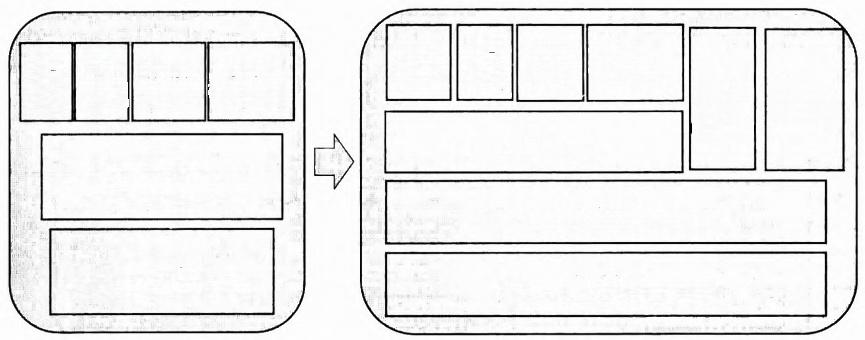
近几年来，MapReduce 计算模型作为 Hadoop 的处理引擎，提供了对大数据进行处理 的现实选择。MapReduce 的主要设计目标是对大数据进行批处理。但是，目前人们需要 在大数据上实现交互式查询，这是面向批处理的MapReduce 计算模型所不能支持的。不 仅对于交互式查询不能提供很好的支持，Hadoop 1.0的MapReduce 计算模型对于涉及迭 代 处 理 (Iterative Processing) 的机器学习算法，也不是 一 个合适的计算模型。

主要的原因是每个迭代 (Iteration) 都作为一个 MapReduce 作业实现，这个作业从 HDFS 读取数据，并且把中间结果写入HDFS 。这种执行模式对数据处理的延迟造成了不 利的 (Detrimental) 影响。

为了解决Hadoop/MapReduce计算模型执行过程中的延迟问题，HortonWorks 提出了 Tez, 作为Apache培育项目 (Incubator Project) 。Tez 项目的目标是建立一个执行框架，支持 大数据上的以DAG(Directed Acyclic Graph) 表达的作业的处理。它是Hadoop 2.0平台上可 扩展的、高效的执行引擎。 Hive,Pig 或 者Cascading 作业，通过Tez 引擎可以运行得更快。

Hadoop 2.0 的提出，使得在 Hadoop 平台上执行用 DAG 表达的作业成为可能。 YARN 把 MapReduce 数据处理模型和资源管理功能分开来，在Hadoop 1.0中这两个部 分是结合在 一 块儿的。

现在，我们可以在YARN 上运行多个数据处理模型，除了支持交互式处理的Tez, 还 包括支持迭代处理的 Spark, 支持实时流数据处理的Storm, 支持图数据处理的Gra- phLab/Giraph 等，如图12-16所示。



|  |  |
| --- | --- |
| Pig Hive (data  flow) *SQL)* | HBase Others  (Service)(cascading) |

Tez

(Execution Engine)

(Cluster resource management

YARN

(Cluster resource management)

HDFS 2

(Redundant,reliable storage)

HDFS

(Redundant,reliable storage)

Stream

(Storm) Graph

(Giraph)

Others

(Cascading)

&data processing)

Pig (Data flow)

MR

(Batch)

HBase

(services)

Hive

(SQL)

MapReduce

**图12-** **16** **Hadoop 1.0仅仅支持1种计算模型，Hadoop 2.0支持多种计算模型**

这些数据处理模型，可以共存于YARN 平台上，对保存在HDFS中的数据以不同的 方式进行处理。在这里，YARN 提供了通用的资源管理框架，MapReduce 则成为 Hadoop 集群的一种应用模型，对数据进行批处理。

Tez 处在YARN 和MapReduce/Hive/Pig 之间，提供一种通用的数据处理模型，可以 执行使用有向无环图 (DAG) 连接起来的任务 (Task) 所表达的作业。在 Hadoop 生态系 统中的Apache Hive,Apache Pig,Cascading 等工具现在运行在Tez 之上。利用Tez 的高 性能数据处理能力，这些工具能够在Petabyte 级数据上提供交互式的查询响应时间。

**12.6.2** **把数据处理逻辑建模成一个** **DAG** **连接起来的任务**

有向无环图DAG(Directed Acyclic Graph) 定义一个数据处理应用程序的数据处理 流程。其中，DAG 的顶点表示数据处理任务，它反映了一定的业务逻辑 (Business Log- ic), 即如何对数据进行转换和分析。 DAG 的边表示数据在不同顶点间的传递。

Tez 把每个顶点建模成Input,Processor 和 Output 模块的组合。 Input 和 Output 模 块 决 定了数据的格式，以及数据应该从哪里读，写入到哪里去等。 Input 代表一个顶点的输入来 源(或者管道)。通过Input,Processor 可以从数据源比如 HDFS, 或者另外 一 个 DAG 顶 点 的输出，获得输入数据。 Output 则代表 一 个顶点的输出去向(管道)。通过 Output, 一 个 Processor 可以为另外一个 DAG 顶点生成和递送数据，或者把生成的数据保存到HDFS 中 。 Processor 则包装了 (Hold) 数据转换和处理的逻辑，它从一个或者多个 Input 消费数据进行 相关处理，然后产生多个Output 。 用户把不同的Input,Processor,Output 模块组合成顶点，

在此之上，创建DAG 数据处理工作流，执行特定数据处理逻辑。

一个DAG 对应一个数据处理流程，可以表示为一张图。Tez 则自动把DAG 映射到物 理资源，在Hadoop 平台上，并行执行这些业务逻辑。Tez 运行时把DAG 的逻辑表示扩 展成物理表示。它为每个顶点增加并行性，也就是使用多个任务 (Task), 来并行执行该 顶点的计算任务。 DAG 的每条边也同步地物化为这些不同顶点的任务间的实际连接。

具体来讲，为了执行 一 个 Tez 作 业 ， 首 先 通 过 Tez 的初始化例程，把环境/配置 (Context/Configuration) 信息提供给Tez 运行时，然后，对每个顶点的每个任务(任务数 量根据并行度创建)进行初始化，再执行这些任务的Processor 的 run() 方法。当这些方



法执行结束，任务就完成了。(utput 框架把从Processor 接收到的数据，通过管道递送给紧 跟着该顶点的下游顶点的Input 模块，直到整个DAG 的所有顶点的任务执行完毕。

Tez 的一个重要和独特的特性是它可以动态地优化DAG 的执行过程，它可以根据执 行过程中获得的信息，比如数据的采样，优化 DAG 的执行计划。 Tez 支持插件式的顶点 管理模块，这些模块能够收集任务信息，在运行时改变数据处理工作流图，从而优化资源 使 用 (Resource Usage) 效率，提高性能。

Apache Pig 和 Apache Hive 把查询进行转换以后，生成一个DAG 。这 些 DAG 交 给 Tez 执行，因此， Tez 成 为Pig 和 Hive 公用的底层执行引擎。

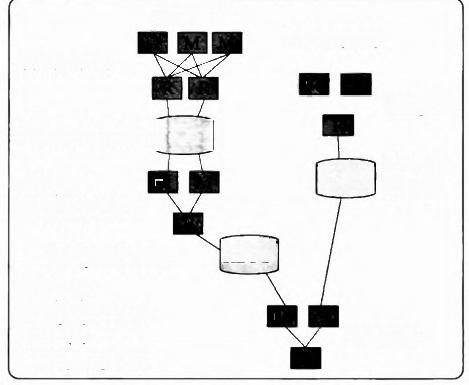
**12.6.3 Tez(DAG Job) 相** **对** **于MapReduce(Job) 的** **优** **势**

由 于DAG 的表达能力比MapReduce 计算模型要强，原先有些数据处理逻辑可能需要 用多个MapReduce 作业来表达，现在只需要一个DAG 作业来表达即可。下面通过一个具 体实例，来解释Tez 的这个优势。

下面的 Hive 语句，对三个表进行连接操作，并且计算一些聚集结果。

|  |  |
| --- | --- |
| Select a.state,Count(\*),**Average**(c.price) **From** **a**  **Join** **b** **On**(a.id=b.id)  **Join** **c** **On**(a.itemld=c.itemld) **Group** **By** a.state | 可以把b表看成客户表customer,a表看成销售表 sales,c表看成产品表product。该查询的目的是查 看各州的销售记录数量，及销售产品的平均价格。 |

在 Hive 里这个语句被转换成若干MapReduce 作业，依次运行，图12- 17 (a) 展 示 了 MapReduce 作业的执行流程。在Tez 里这个语句被转换成一个用DAG 表达的作业，如 图 1 2 - 1 7 (b) 所示。



MR jobl

Select(b.id)

Selec a.state

MR job2

1O Sync Barrier HDFS

Join(a.c)

Select c.price

IO Sync Barrier

Join(a,b)

Group by a.state Count(\*)

Average(c.price)

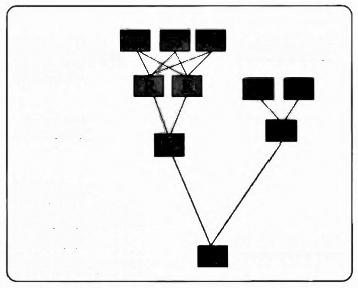
IO Sync FDFS Barrier

MR job3

HDFS



(a) 查询对应的MapReduce作业



Selec a.state,

c.itemid

Join(a.c)

Join(a,b)

Group by a.state Count(\*)

Average(c.price)

**Single Tez Job**

Select(b.id)

(b) 查询对应的Tez作业(DAG)

**图12-17** **Hive 2 MapReduce与Hive 2 Tez (来自Hortonworks)**

我们可以看到，为了执行上述连接查询， Tez 需要更少的作业 (Tez 需要1个作业，

MapReduce 需要3个作业),无须任何的同步Barrier(MapReduce 作业通过把中间结果写

入 HDFS 和从中读出来实现同步)。

Tez 通过把原先用多个MapReduce 作业表达的数据处理逻辑整合在一个作业里，消除不 必要的任务、同步Barrier、对 HDFS 的读写操作等，提高了数据处理的效率。它的查询执行 能力，对小规模、低延迟和大规模、高吞吐量的负载同样有效。Tez 把 Pig和 Hive的负载的 执行速度提高了一个数量级，Tez 已经成为Hive 1.3版本的基础(也称为Stinger)。

除了使用Tez 执行引擎，Hive 还可以通过采用列存储格式保存数据、开启向量化查 询执行模式等方式提高查询执行效率。此外，新版本的Hive 集成了基于代价的查询优化 技术 (Cost-based Optimization), 能够为SQL 查询生成更加高效的执行计划。

**12.7** **Hadoop平台上的列存储技术**

**12.7.1** **列存储的优势**

第 3 章 “OLAP 与结构化数据分析”介绍过列存储技术。在分析型应用中，列存储技术 具有若干明显的优势： (1)更少的 I/O 操作。读取数据时，可以把投影下推 (Project Pushdown), 只需读取查询需要的列，可以大大减少每次查询的I/O 数据量，甚至可以支持 谓词下推 (Predicate Pushdown), 跳过不满足条件的行。通过减少不必要的数据扫描，提高 查询性能。(2)更高的压缩比。每一列中的数据类型相同，由于相同类型的数据连续存放， 可以使用针对性的编码和压缩方法对数据进行压缩，大大降低数据存储空间。(3)由于每一 列的数据类型相同，可以使用更加适合CPU Pipeline的编码方式，减少CPU 的缓存失效 (Cache Miss) 。Hadoop生态圈涌现出一系列列存储格式，包括RC-File,ORC,Parquet 等。

**12.7.2 RCFile**

RCFile(Record Columnar File Format) 是 Facebook 公司、俄亥俄州立大学、中科院 计算所等单位合作研发的列存储文件格式。成果以论文 RCFile:A Fast and Space-effi- cient Data Placement Structure in MapReduce—Based Warehouse Systems 发表在 ICDE 2011(IEEE International Conference on Data Engineering) 会议上。

RCFile 文件格式第一次在 Hadoop 中引入了列存储格式， Hive 数据仓库因此提高了 数据分析的性能。测试显示，在多数情况下，它比其他已有的文件格式获得更好的性能。

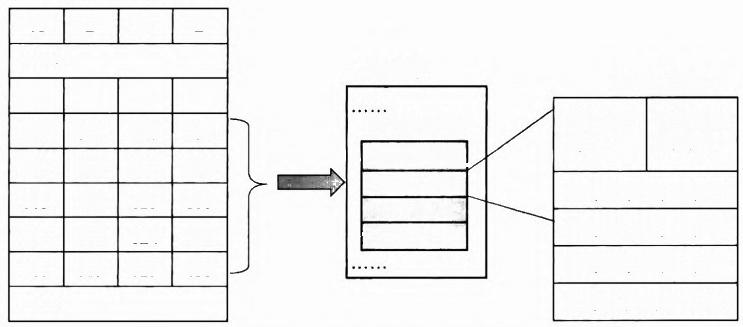
RCFile 是一种允许按行查询，同时提供了列存储的压缩效率的列存储格式，它具有 若干优势。首先，RCFile 具备相当于行存储的数据加载速度和负载适应能力。其次， RC- File的读优化可以在扫描表格时避免不必要的数据列的读取。最后，RCFile 使用列维度的 压缩，能够有效提升存储空间利用率。为了提高存储空间利用率， Facebook 各产品线产

**生的数据从2010年起均采用** **RCFile** **格式存储，原来按行存储格式保存的数据集** **(Se-** **quence** **File/Text** **File)** **也转存为RCFile** **格式。**

**为了把表格中的数据保存到文件中，RCFile 首先横向分割表格，然后纵向分割表格。** **对表格的横向分割把表格划分成多个行组** **(Row Group), 行组的大小可以由用户指定。** **在每个** **Row Group内** **部** **，RCFile 按照列存储的一般做法，把各个列数据分开，分别连续**



保存。如图12- 18所示。



Relational Table

A B C D

101 111 121 131

102 112 122 132

103 113 123 133

104 114 124 134

105 115 125 135

Row Group

16 bytes Metadata Sync header

101,102,103,104,105

111,112,113,114,115

121,122,123,124,125

131,132,133,134,135

RCFile

HDFS Block

Row Groupl Row Group2

……

Row Groupn

**图12-18** **RCFile 文件结构**

在每个Row Group内部，当进行数据写盘时， RCFile 针对每列数据使用Zlib/LZO 等 算法进行压缩，以减少空间占用。在Facebook 的数据仓库中选取部分有代表性的应用， 实验结果显示， RCFile 能够提供5:1的压缩比。

当读取数据时，RCFile 采用惰性解压策略 (Lazy Decompression), 即用户的某个查询如果 只是涉及一个表中的部分列，RCFile会跳过不需要的列的解压缩和反序列化的过程。

**12.7.3 ORC存储格式**

ORC File(Optimized Row Columnar File) 是对RCFile 做出优化的一种存储格式。ORC File 于2013年开始研发，它的设计目标是克服Hive 其他格式的缺陷，进一步提高 Hive 查询处理速 度以及Hadoop 的存储效率。运用ORC File, 可以高效地存储Hive 数据，提高Hive 的读、写 操作以及处理数据的性能。

和 RCFile 相比，ORC 文件格式具有若干优势：

(1)ORC File 支持更加丰富的数据类型，包括日期时间类型 (Date Time)、数字类型 (Decimal) 以 及Hive 的各种复杂类型 (Complex Type), 包括结构 (Struct) 、 列 表 (List) 、 映 射 (Map) 、 联合 (Union) 等。

(2)ORC File是一个自描述的 (Self Describing) 和类型感知的 (Type Aware) 列存储文件 格式。它为流式读取 (Streaming Read) 操作进行了优化，同时支持快速查找少数的数据行。 ORC File是类型感知的，它对RCFile的优化在于在文件写入时，Writer针对不同列的数据类型 使用不同的编码器进行编码，从而提高压缩的比率。比如针对整数类型，使用变长压缩方法 (Variable Length Compression), 针对字符串类型，使用词典编码 (Dictionary Encoding) 方法。

(3)ORC File引入了轻量级的索引以及基本的统计信息，包括各个数据列的最小值 (Min) 、最大值 (Max) 、 总 和 (Sum) 、 记录数 (Count) 等信息，于是在查询的处理过程中， 可以忽略大量不符合查询条件的记录。

通过谓词下推 (Predicate Pushdown), 查询处理器使用这些索引，确定哪些Stripe 需要读 取，并且利用Row Index, 进一步限定扫描的范围到最小由10000行构成的记录集合。

**ORC** **File文件结构**

一个ORC 文件由多个Stripe (可以译作条带，在这里我们直接使用Stripe) 、一个包含辅助



信息的File Footer以 及Postscript 构 成 。

( 1 ) 每 个Stripe 包括索引数据 (Index Data)、行 数 据 (Row Data) 及 一 个Stripe Footer。

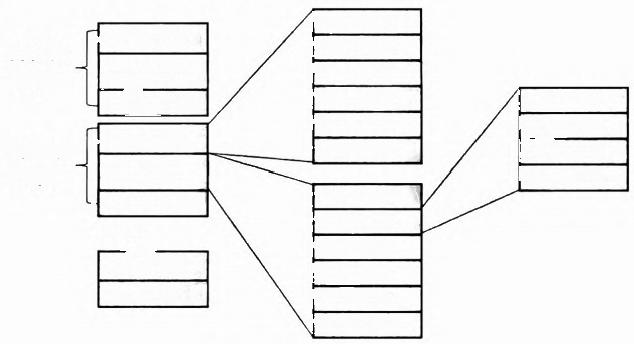
ORC File将整张表划分成一系列的行组 (Row Group),Stripe实际上就是一个Row Group。默认情况下，一个Stripe的大小为256MB, 其大小可以扩展的长度只受HDFS 的 约束。大尺寸的Stripe 针对串行I/O 做出优化，使得读取数据的吞吐量提高了，需要读取 的文件更少，同时减轻了NameNode 的负载。

Index Data 部分包含每个列的最小值与最大值。此外， 一系列的行索引条目 (Row Index Entry) 记录压缩块的偏移量，利用它可以跳到正确的压缩块位置。行索引条目使得 查询处理器可以跳过一些行，实现快速的数据读取。缺省情况下，每次最多可以跳过 10000行。Row Data部分包含每个列的数据，每一列由若干数据流构成 (Data Stream)。 Stripe Footer部分包含数据流的位置信息以及每 一 列数据的编码方式。

( 2 ) 在File Footer 里面，包含了该ORC File 文件中所有Stripe 的元信息，即每个 Stripe 的位置、每个Stripe 中有多少行以及每列的数据类型等。它里面还包含了列级别的 一些聚集结果，比如记录数 (Count) 、 最小值 (Min) 、 最大值 (Max) 、 总和 (Sum) 等 。

(3)在文件的末尾，有一个称为Postscript的结构，它用来存储压缩参数及Footer的 长度(被压缩过的Footer 的大小)。

ORC 文件结构如图12-19所示。



|  |  |  |  |
| --- | --- | --- | --- |
| 256MB Stripe  256MB Stripe | i**ndex Data**  Row Data  Stripe Footer  Index Data  Row Data  Stripe Footer  File Footer  Postscript | Column 1  Column 2  Column 3  Column 4  Column 5  Column 6  Column 1 Column 2  Column 3  Column 4  Column 5  Column 6 | Stream 2.1  Stream 2.2  Stream 2.3  Stream 2.4 |

**图12** **-** **19** **ORC文件结构**

**12.7.4 Parquet 文件格式**

Parquet 也是Hadoop 平台上的一种列存储格式。Parquet 的灵感来自2010年Google 发表 的关于Dremel 系统的论文。该论文介绍了一种支持嵌套结构的列存储格式，提升查询性能。 Parquet 为 Hadoop 生态系统中的所有项目提供支持高压缩率的列存储格式。它兼容各种数据 处理框架、对象模型，支持各种查询引擎 (Hive.Impala,Presto 等),与编程语言无关。 Parquet 最初由Twitter 和Cloudera(Impala 的开发者)合作开发完成，然后开源。2015年5 月 ，Parquet 从 Apache的孵化器里孵化完成，成为Apache 顶级项目。

**1.Parquet 在系统中的位置**

从 一 个完整的系统角度来看，能够与Parquet 配合的组件有：

(1)查询引擎： Hive,Impala,Pig,Presto,Drill,Tajo,HAWQ,IBM Big SQL等。

|  |  |  |
| --- | --- | --- |
| (2)计算框架： MapReduce,Spark,Cascading,Crunch,Scalding,Kite | | 等。 |
| (3)对象模型： Avro,Thrift,Protocol Buffers等 。 Parquet与这些组件的协作关系，如图12-20所示。  Java | | C++ |
| Avro  介  Parquet-Avro  工  Object Model  Converters  Model Agnosic  Language Agnosic | Thrift Pig Tuple Hive SeDe  介  食  Parquet-thrift Parquet-protoc Parquet-pigParquet-hive    Assembly/Striping    Column Encoding  介  Parquet File Format | Query Execution  Encoding  食 |

图12-20 Parquet与各个组件的关系

数据从内存保存到Parquet文件以及从Parquet 文件装载到内存的过程，涉及如下的组件：

(1)存储格式 (Storage Format):Parquet 文件格式 (File Format) 定义了Parquet 内部的数据类型、存储格式等。

(2)对象模型转换器 (Object Model Converter): 这部分功能由 Parquet-mr 来实现， 完成对象模型与 Parquet 数据类型的映射。

(3)对象模型 (Object Model): 对象模型可以简单理解为内存中的数据表示，包括 Avro,Thrift,Protocol Buffer,Hive SerDe,Pig Tuple,Spark SQL Internal Row 等对 象模型。

比如，Parquet-mr 项目里的Parquet-pig 子项目，负责把内存中的Pig 元组 (Tuple) 序列化并按列存储格式保存成Parquet格式的文件，以及反过来，把Parquet 格式的文件 里的数据，反序列化成Pig元组 (Tuple)。

2.Parquet 数据模型

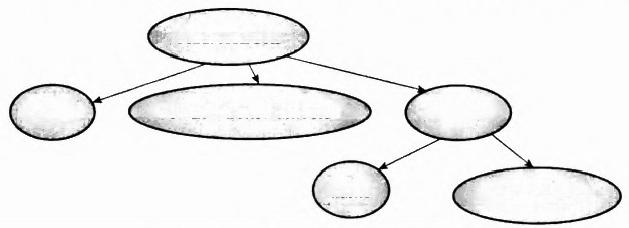
为了深入理解 Parquet 列存储格式的数据模型，我们通过如下的数据模式 (Schema) 来进行说明。

|  |
| --- |
| message AddressBook  required string owner;  repeated string ownerPhoneNumbers; repeated group contacts  {  required string name;  optional string phoneNumber;  f |

这个数据模式 (Schema) 中的每条记录，表示一个人的地址簿信息 (AddressBook)。

每个记录有且只有一个拥有者 (owner), 可以有0个或者多个电话号码 (ownerPhoneN-

umbers), 以 及 0 个 或 者 多 个 联 系 信 息 (contacts) 。 每 个 contact 有且只有 一 个名称 (name),contact 的电话号码 (phoneNumber) 则可有可无(见图12-21)。



**AddressBook**

**ownerPhoneNumbers**

owner

**name**

**phoneNumber**

**contacts**

**图12-21** **AddressBook 对应的Schema**

AddressBook 模式信息可以组织成一棵树，如图12-21所示。 一般一个Schema 用 一 棵树来表达，有一个根节点，根节点包含多个域 (Field), 即子节点，子节点还可以包含 子节点。每个域包含三个属性，分别是repetition,type,name 。repetition 描述一个 Field 出现的次数，包括required (必须出现1次)、 optional ( 出 现 0 次 或 者 1 次 ) 、repeated (出现0次或者多次)三种情况。 type 可以是一个原生类型 (primitive) 或者一个衍生类 型 (group)。

Parquet 通过把List 和 Set 表示成一个重复的域 (Repeated Field), 把 Map 表示成一 个包含有键值对的 Repeated Group(Key是必需的 (Required)), 提供对复杂数据结构的 支持，如图12-22和图12-23所示。

|  |  |
| --- | --- |
| Schema:List of Strings | Data:{“a”,“b”,“c”,… |
| message ExampleList  {repeated string list; } | {  list:“a”,  list:“b”,  list:“c”, … |

**图12-22** **List (或Set) 可以用Repeated Field 来表示**

|  |  |  |
| --- | --- | --- |
| Schema:Map of Strings to Strings | Data:{“AI”→“Alabama”,… | |
| message ExampleMap  {repeated group map  {required string key; optional string value; | {  } | map{key:“AL”  value:“Alabama”  },  map{key:“AK”  value:“Alaska”  },  … |

**图12-23** **Map** **可以用包含键值对且Key是Required的Repeated** **Group来表示**

如何把内存中的每个 AddressBook 对象按照列存储格式保存到硬盘文件中，在Par- quet 格式的存储中， 一个数据模式 (Schema) 的树结构有几个叶子节点，实际的存储中 就会有多少列 (Column) 。 于是，上述实例中的AddressBook 数据模式，在存储上共有四 个 列 (Column), 如图12 - 24所示。

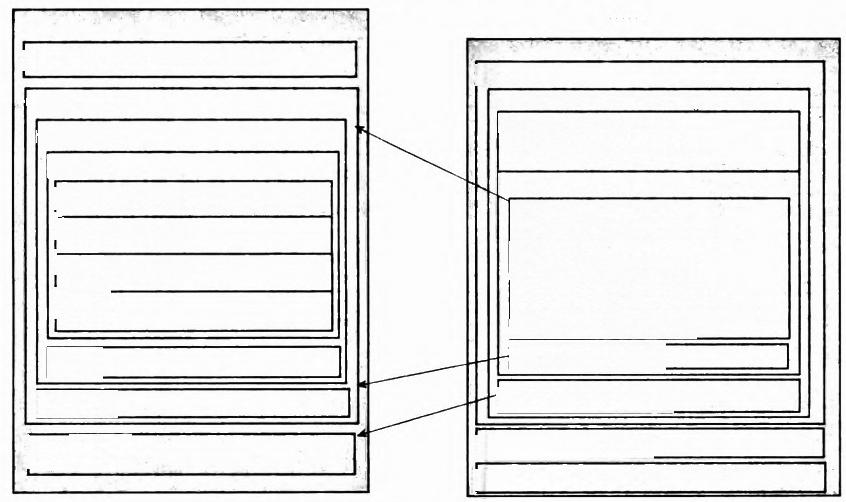
|  |  |
| --- | --- |
| Column | Type |
| **owner** | string |
| ownerPhoneNymbers | string |
| contacts.name | string |
| contacts.phoneNumber | string |

|  |  |  |  |
| --- | --- | --- | --- |
| AddressBook | | | |
| **owner** | ownerPhoneNumbers | **contacts** | |
| name | phoneNumber |
| … | … | … | … |
| … | … | … | … |
| … | … | … | … |

**图12-24** **AddressBook 实际存储的列**

**3.Parquet 文件结构**

Parquet 文件结构涉及如下几个重要的层次，如图12-25所示。



**File**

Magic Number(4bytes):“PAR1”

**Row Group 0**

Column a

Page 0

Pageheader (ThriftCompactProtocol) Repetition Levels

Definition Levels

Values

Page 1

Column b

Row Group1

Fo oter

File Meta Data(ThriftCompactProtocol)

-Version(of the format)

-Schema

-Extra<Key,Value>pairs Row group 0 meta data

Column a meta data:

-type/path/encodings/codec

-num values

-offset of first data page

-offset of first index page

-compressed/uncompressed size

-extra<Key,Value>pairs Column b meta data

Row groupl meta data

Footer Length(4bytes)

Magic Number(4butes):“PAR1”



**图12-25** **Parquet 文件结构**

(1)HDFS 文 件 (File): 一 个HDFS 文件，包括数据和元数据。数据存储在多个Block 中 。

(2)HDFS 数 据 块 (Block): 它 是HDFS 上最小的存储单位。 HDFS 把 一 个 Block 保 存到本地磁盘，并且在不同的机器上维护另外两个副本。通常情况下， 一个Block 的 大 小

为256M,512M 等。

(3)行组 (Row Group): 按照行将数据(表格)物理上划分为多个单元。每一个行 组包含一定的行数。一个行组包含该行组数据的各个列对应的列块。 一般建议采用更大的 行组大小(512MB～1GB) 。更大的行组意味着更大的列块，有利于在硬盘上做串行I/O。

由于一次可能需要读取整个行组，所以一般让一个行组刚好在一个 HDFS 数据块中， HDFS 数据块的大小需要设得更大。比如，采用1GB 的行组和1GB 的 HDFS 数据块大 小，1个 HDFS 数据块放一个行组。

(4)列块 (Column Chunk): 在一个行组中，每一列保存在一个列块中。行组中的所 有列连续地存储在这个行组中。不同的列块可以使用不同的算法进行压缩。 一个列块由多 个页组成。

( 5 ) 页 (Page): 每一个列块划分为多个页，页是压缩和编码的单元。在同一个列块 内的不同页，可以使用不同的编码方式。典型的页面大小为1MB(After Encoding)。

**4.** **数据压缩算法**

列存储给数据压缩提供了更大的发挥空间，除了我们常见的Snappy,GZIP 等通用压缩 方法以外，由于同一列的数据类型是一致的，可以使用不同的压缩算法，如表12-1所示。

**表12-1** **不同压缩算法及其应用场景**

|  |  |
| --- | --- |
| 压缩算法 | 使用场景 |
| 行程编码(Run Length Encoding) | 重复数据 |
| Delta编码(Delta Encoding) | 有序数据集，例如timestamp,自动生成的ID,以及监控的各 种指标(Metric) |
| 字典编码(Dictionary Encoding) | 小规模的数据集合，例如IP地址 |
| 前缀编码(Prefix Encoding) | Delta Encoding for Strings |

**5. 谓词下推** **(Predicate Pushdown)**

在数据库系统中，最常用的查询优化手段之一是谓词下推。通过将一些过滤条件尽可 能地尽快执行，可以减少查询计划后续需要处理的数据量，从而提升性能。

比如，对于如下的SQL 查询

|  |
| --- |
| **Select Count** (\*)  From A Join B On A.id=B.id  **Where** A.a>10 **And** and B.b<100 |

如果首先对A 和 B 执行Table Scan 操作，然后进行Join 操作，再执行过滤，最后计 算聚集函数返回，效率是不高的。但是，如果把过滤条件A.a>10 和 B.b<100 分别移到 A 表的 Table Scan 和 B 表的 Table Scan的时候执行，遴选出符合条件的记录，就可以大 大降低Join操作的输入数据，加快Join操作，并且提高整个查询的效率。

在 Parquet 中，每一个Column Chunk 在存储时，都计算并保存相应的统计信息，包 括最大值、最小值和空值个数，这些信息可以加快查询处理。我们可以针对查询字段对 数据进行排序，然后以 Parquet 格式保存，利用Parquet 为每个 Column Chunk 保存的 min/max 统计信息，把不符合查询条件的Column Chunk 忽略掉，不用装载和处理。比 如，按照成绩字段对学生成绩表进行排序，当查询成绩在90～100分的学生信息时，那些落

在这个分数段之外的Column Chunk (根据min/max 统计信息判断)就无须处理了。 Par- quet 的新版本将增加诸如Bloom Filter 等的索引结构，帮助完成数据的过滤。

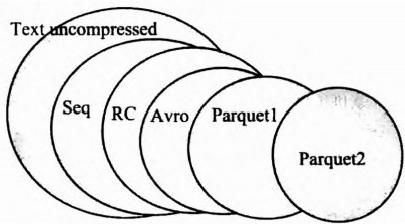
**6.** **映射下推** **(Project** **Pushdown)**

列存储格式的另一个优势是映射下推。在获取表中的数据时，只需扫描查询中需要的 列。由于避免了不必要的数据列的提取，查询的效率得以提高。

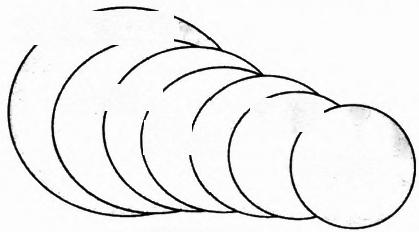
**7.Parquet** **的性能**

图12-26展示了使用不同存储格式存储TPC-H 和 TPC-DS 数据集中的Line Item表 和 Sales 表的文件的相对大小。可以看出， Parquet 较之于其他文件存储格式，能够更有效 地利用存储空间，新版本的Parquet(2.0 版)使用了更加高效的存储格式，进一步提升 了存储效率。

Size Comparison

TPC-DS 100GB scale factor(+Snappy unless otherwise specified)

Store sales



*Tey uncompressed*

Seq Avro/

Text+

LzO RC

Parquet2

Lineitem

Parqucetl

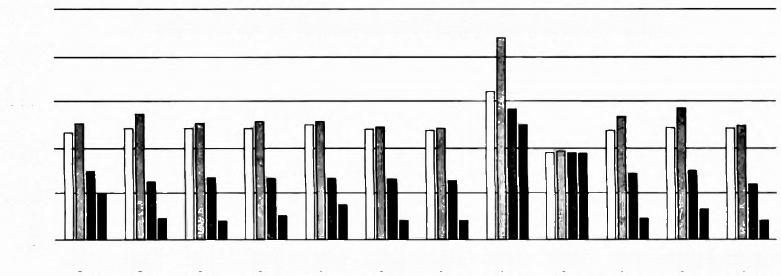
The area of the circle is proportional to the file size

**图12-26** **不同存储格式的压缩效果**

资料来源： <https://www.slideshare.net/julienledem/th-210pledem.>

图12 - 27展示了Twitter 公司在Impala 查询引擎中，使用不同文件格式，执行TPC- DS 测试基准的结果。从测试结果可以看出， Parquet 较之于其他存储格式，有较明显的性 能提升。

Impala Query Times on TPC-DS (Seconds,wall clock)



□Text ■Seq w/Snappy ■RC w/Snappy ■Parquet w/Snappy

400

300

200

100

0

Q27 Q34 Q42 Q43 Q46 Q52 Q55 Q59 Q65 Q73 Q79 Q96

500

**图12-27** **不同存储格式的查询性能**

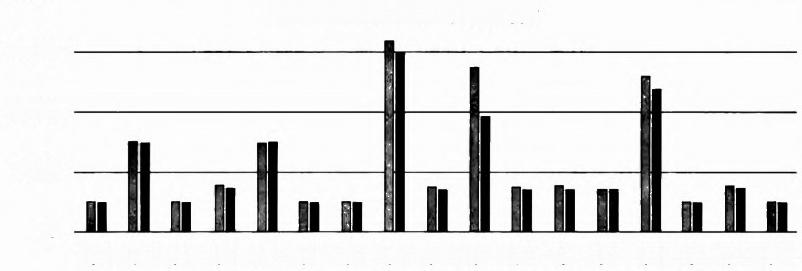
资料来源： Julien Le Dem.Parquet Strata/Hadoop World.<https://www.slideshare.net/julienledem/parquet-strat>- any-hadoopworld2013.New York,2013.

图12- 28展示了Criteo 公司在 Hive 数据仓库中，使用ORC 和 Parquet 两种列存储 格式，执行 TPC-DS 测试基准的结果。从测试结果可以看出，在数据存储方面，两种存储 格式在使用Snappy 压缩的情况下，占用的空间相差并不大，在查询性能上Parquet 格式 稍微好于ORC 格式。

在这个实验中，TPC-DS 数据集的Scale Factor为100,相对于文本文件，ORC Snap- py 把文件压缩到原来文件大小的35%,Parquet Snappy 把文件压缩到原来文件大小的 33%。所有的作业，其Mapper 设置为50。每个节点的配置为：2×6核CPU,96GB 内 存，14×3TB 硬盘。

Performance of Hive 0.11 with Parquet vs.ORC

20000



■ORC Snappy ■Parquet Snappy

15000

10000

5000

0

Q19 Q34 Q42 Q43 Q46 Q52 Q55 Q59 Q63 Q65 Q68 Q7 Q73 Q79 Q8 Q89 Q98

**图12-28** **不同存储格式的查询性能**

资料来源： Julien Le Dem.Parquet Strata/Hadoop World.<https://www.slideshare.net/julienledem/parquet-strat>- any-hadoopworld2013,New York,2013.

Parquet 格式和ORC 格式各有优缺点。Parquet 格式原生地支持嵌套式数据结构，ORC 对此支持得较差。Parquet 不支持数据的修改和ACID 事务语义，但是 ORC 对此提供支持。 在OLAP 环境下，我们很少会对单条数据进行修改，更多的是对数据进行批量导人。

**12.8** **思考题**

(1)HDFS 分布式文件系统及其读写过程。

(2)MapReduce 执行引擎，Job Tracker与 Task Tracker。

(3)MapReduce 计算模型与实例。

(4)Hadoop 的应用。

(5)Hadoop 的生态系统。

(6)Hive on MapReduce。

(7)Hadoop 1.0 的局限。

(8)Hadoop 2.0(YARN) 的原理及其优势。

(9)Hive on Tez。

(10)RCFile,ORC File,Parquet 列存储格式。