[Hadoop历史与演变` 1](#_Toc366599269)

[Hadoop HDFS 5](#_Toc366599270)

[Hadoop mapreduce 11](#_Toc366599271)

[Hadoop部署 13](#_Toc366599272)

[Hadoop应用 13](#_Toc366599273)

[学习建议 13](#_Toc366599274)

# Hadoop及相关技术简介

## Hadoop的历史

Hadoop这个名字不是一个缩写，它是一个虚构的名字。该项目的创建者，Doug Cutting如此解释Hadoop的得名："这个名字是我孩子给一头吃饱了的棕黄色大象命名的。我的命名标准就是简短，容易发音和拼写，没有太多的意义，并且不会被用于别处。小孩子是这方面的高手。Googol就是由小孩命名的。"

**2003年 Google发表描述Google分布式文件系统(简称GFS)的论文**

**2004年 Google在发表了题为"MapReduce: Simplified Data Processing on Large Clusters"的论文**

2004年-- 最初的版本(现在称为HDFS和MapReduce)由Doug Cutting和Mike Cafarella为解决大规模网页抓取、存储和处理的成本问题。

2005年12月-- Nutch移植到新的框架，Hadoop在20个节点上稳定运行。

2006年1月-- Doug Cutting加入雅虎。

**2006年2月-- Apache Hadoop项目正式启动以支持MapReduce和HDFS的独立发展。**

2006年2月-- 雅虎的网格计算团队采用Hadoop。

**08年1月 – hadoop成为apache顶级项目**

08年4月-- 赢得世界最快1 TB数据排序在900个节点上用时209秒。

09年4月-- 赢得每分钟排序，59秒内排序500 GB(在1400个节点上)和173分钟内排序100 TB数据(在3400个节点上)。

现在：hadoop技术已广泛应用与个互联网公司、政府和科研机构。

## Hadoop到底是什么？

Hadoop是一个分布式系统基础架构。用户可以在不了解分布式底层细节的情况下，开发分布式程序。充分利用集群的威力高速运算和存储。**简单地说来，Hadoop是一个易用的大规模数据存储和计算平台。**

**Hadoop实现了一个分布式文件系统（Hadoop Distributed File System），简称HDFS**

**Hadoop 实现了MapReduce 计算模型。**

下面列举hadoop主要的一些特点：

1 **扩容能力（Scalable）**：能可靠地（reliably）存储和处理千兆字节（PB）数据。

2 **成本低（Economical）**：可以通过普通机器组成的服务器群来分发以及处理数据。这些服务器群总计可达数千个节点。

3 **高效率（Efficient）**：通过分发数据，hadoop可以在数据所在的节点上并行地（parallel）处理它们，这使得处理非常的快速。

4 **可靠性（Reliable）**：hadoop能自动地维护数据的多份复制，并且在任务失败后能自动地重新部署（redeploy）计算任务。

Hadoop主要子项目

\* Hadoop Common: 在0.20及以前的版本中，包含HDFS、MapReduce和其他项目公共内容，从0.21开始HDFS和MapReduce被分离为独立的子项目，其余内容为Hadoop Common

\* HDFS: Hadoop 分佈式文件系統 (Distributed File System) － HDFS (Hadoop Distributed File System)

\* MapReduce：并行计算框架

\* HBase: 类似Google BigTable的分布式NoSQL列数据库。（HBase 和 Avro 已经于2010年5月成为顶级 Apache 项目[1]）

\* Hive：数据仓库工具，由Facebook贡献。

\* Zookeeper：分布式锁设施，提供类似Google Chubby的功能，由Facebook贡献。

\* Avro：新的数据序列化格式与传输工具，将逐步取代Hadoop原有的IPC机制。

There are numerous ASF projects included in a distribution.  Each of them has been developed to deliver an explicit function and each has it’s own community of developers and individual release cycles.  Below is an outline of the Hadoop related Apache projects.

* [**Apache Pig**](http://hortonworks.com/hadoop/pig)A platform for processing and analyzing large data sets. Pig consists on a high-level language (Pig Latin) for expressing data analysis programs paired with the MapReduce framework for processing these programs.

* [**Apache HCatalog**](http://hortonworks.com/hadoop/hcatalog)A table and metadata management service that provides a centralized way for data processing systems to understand the structure and location of the data stored within Apache Hadoop.

* [**Apache Hive**](http://hortonworks.com/hadoop/hive)Built on the MapReduce framework, Hive is a data warehouse that enables easy data summarization and ad-hoc queries via an SQL-like interface for large datasets stored in HDFS.

* [**Apache HBase**](http://hortonworks.com/hadoop/hbase)A column-oriented NoSQL data storage system that provides random real-time read/write access to big data for user applications.

* [**Apache ZooKeeper**](http://hortonworks.com/hadoop/zookeeper)  
  A highly available system for coordinating distributed processes. Distributed applications use ZooKeeper to store and mediate updates to important configuration information.

* [**Apache Ambari**](http://hortonworks.com/hadoop/ambari)An open source installation lifecycle management, administration and monitoring system for Apache Hadoop clusters.

* [**Apache Sqoop**](http://hortonworks.com/hadoop/sqoop)  
  Sqoop is a tool that speeds and eases movement of data in and out of Hadoop. i provides a reliable parallel load for various, popular enterprise data sources.

* [**Apache Oozie**](http://hortonworks.com/hadoop/oozie)  
  Oozie Java Web application used to schedule Apache Hadoop jobs. Oozie combines multiple jobs sequentially into one logical unit of work.

* **Apache Mahout**  
  Mahout provides scalable machine learning algorithms for Hadoop which aids with data science for clustering, classification and batch based collaborative filtering.

* [**Apache Flume**](http://hortonworks.com/hadoop/flume)Flume allows you to efficiently aggregate and move large amounts of log data from many different sources to to Hadoop.

**Hadoop 2.0**

The future of Hadoop is almost here. Other crucial projects to deliver Hadoop 2.0 include:

* [**Apache YARN**](http://hortonworks.com/hadoop/yarn)Part of the core Hadoop project, Apache YARN is a next-generation framework for Hadoop data processing extending MapReduce capabilities by supporting non-MapReduce workloads associated with other programming models.

* [**Apache Tez**](http://hortonworks.com/hadoop/tez)Apache Tez generalizes the MapReduce paradigm to a more powerful framework for executing a complex DAG (directed acyclic graph) of tasks for near real-time big data processing.

# Hadoop HDFS

HDFS:(Hadoop Distributed File System)，是运用MapReduce框架进行大规模分布式数据处理的高度容错性的文件系统，适合部署在廉价的机器上。HDFS能提供高吞吐量的数据访问，非常适合大规模数据集上的应用。HDFS是hadoop项目的一部分，可以说是Hadoop和Hbase的基石.

HDFS是一个主从(Master/Slave)的架构，一个HDFS集群是由一个名字节点(NameNode)，若干数据节点(DataNode)组成。

**[](http://gitsea.qiniudn.com/wp-content/uploads/2013/06/hdfsarchitecture.gif)**

**一.HDFS节本概念介绍**

* **数据块 (block)**

HDFS默认的基本存储单位是64M的数据块，和普通文件系统相同的是，HDFS中的文件是被分成64M一块的数据块存储的。 不同于普通文件系统的是，HDFS中，如果一个文件小于一个数据块的大小，并不占用整个数据块存储空间。

* **元数据节点(NameNode)**

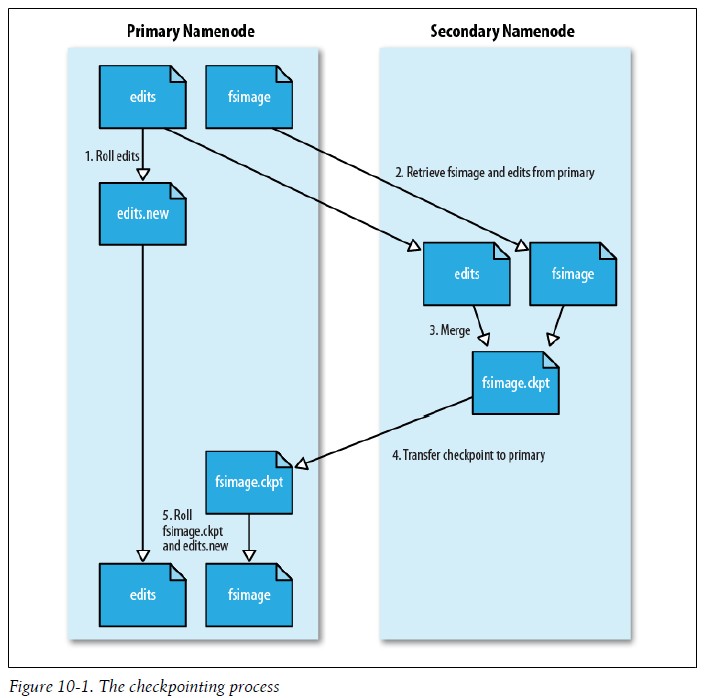
主要用来管理文件系统的命名空间，其将所有的文件和文件夹的元数据保存在一个文件系统树中。 这些信息也会在硬盘上保存成以下文件：命名空间镜像(namespace image)及修改日志(edit log),其还保存了一个文件包括哪些数据块，分布在哪些数据节点上。然而这些信息并不存储在硬盘上，而是在系统启动的时候从数据节点收集而成的。

* **数据节点(DataNode)**

真正存储数据的地方。客户端(client)或者元数据信息(namenode)可以向数据节点请求写入或者读出数据块。 其周期性的向元数据节点回报其存储的数据块信息。

* **从元数据节点(Secondary NameNode)**

从元数据节点并不是元数据节点出现问题的时候的备用节点，它和元数据节点负责不同的事情。 其主要功能就是周期性将元数据节点命名空间的镜像文件和修改日志文件合并，以防日志文件过大。     合并过后的命名空间镜像文件也在从元数据节点保存了一份，以防元数据节点失败的时候，可以恢复。secondarynamenode一般来说不应该和namenode在一起。

[](http://gitsea.qiniudn.com/wp-content/uploads/2013/05/2012081314565068.jpg)

Secondary NameNode主要是做Namespace image和Edit log合并的。

当客户端执行写操作，则NameNode会在edit log记录下来，并在内存中保存一份文件系统的元数据。

Namespace image（fsimage）文件是文件系统元数据的持久化检查点，不会在写操作后马上更新，因为fsimage写非常慢。

由于Edit log不断增长，在NameNode重启时，会造成长时间NameNode处于安全模式，即不可用状态，是非常不符合Hadoop的设计初衷。所以要周期性合并Edit log，但是这个工作如果由NameNode来完成，会占用大量资源，这样就出现了Secondary NameNode，它可以进行image检查点的处理工作。步骤如下：

* Secondary NameNode请求NameNode进行edit log的滚动（即创建一个新的edit log），将新的编辑操作记录到新生成的edit log文件；
* 通过http get方式，读取NameNode上的fsimage和edits文件，到Secondary NameNode上；
* 读取fsimage到内存中，即加载fsimage到内存，然后执行edits中所有操作并生成一个新的fsimage文件，即这个检查点被创建；
* 通过http post方式，将新的fsimage文件传送到NameNode；
* NameNode使用新的fsimage替换原来的fsimage文件，让 Secondary NameNode创建的edits替代原来的edits文件；并且更新fsimage文件的检查点时间。整个处理过程完成。

Secondary NameNode的处理，是将fsimage和edites文件周期的合并，不会造成nameNode重启时造成长时间不可访问的情况。

**二. 文件目录结构**

**1)元数据节点**

目录结构

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | [frankwong@Master name]$ ll  总用量 12  drwxrwxr-x. 2 frankwong frankwong 4096 6月   1 22:27 current  drwxrwxr-x. 2 frankwong frankwong 4096 6月   1 21:43 image  -rw-rw-r--. 1 frankwong frankwong    0 6月   1 22:27 in\_use.lock  drwxrwxr-x. 2 frankwong frankwong 4096 6月   1 22:27 previous.checkpoint    具体信息： |
| 1  2  3  4  5  6  7  8  9 | [frankwong@Master name]$ cd current/  [frankwong@Master current]$ ls  edits fsimage fstime VERSION  [frankwong@Master current]$ more VERSION  #Sat Jun 01 22:32:49 CST 2013  namespaceID=2035266975  cTime=0  storageType=NAME\_NODE  layoutVersion=-32 |

说明：

VERSION文件是java properties文件，保存了HDFS的版本号。  
layoutVersion是一个负整数，保存了HDFS的持续化在硬盘上的数据结构的格式版本号。  
namespaceID是文件系统的唯一标识符，是在文件系统初次格式化时生成的。  
cTime此处为0  
storageType表示此文件夹中保存的是元数据节点的数据结构。

**2)数据节点**

目录结构

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | [frankwong@Slave01 data]$ ll  总用量 20  drwxrwxr-x. 2 frankwong frankwong 4096 6月   1 22:27 blocksBeingWritten  drwxrwxr-x. 2 frankwong frankwong 4096 6月   1 22:27 current  drwxrwxr-x. 2 frankwong frankwong 4096 6月   1 22:27 detach  -rw-rw-r--. 1 frankwong frankwong    0 6月   1 22:27 in\_use.lock  -rw-rw-r--. 1 frankwong frankwong  157 6月   1 22:27 storage  drwxrwxr-x. 2 frankwong frankwong 4096 6月   1 22:27 tmp |

具体信息

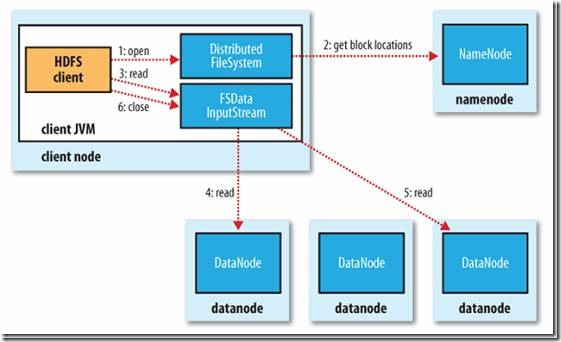
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | [frankwong@Slave01 data]$ cd current/  [frankwong@Slave01 current]$ ls -l  总用量 16  -rw-rw-r--. 1 frankwong frankwong   4 6月   1 22:27 blk\_-7914335926959921103  -rw-rw-r--. 1 frankwong frankwong  11 6月   1 22:27 blk\_-7914335926959921103\_1001.meta  -rw-rw-r--. 1 frankwong frankwong  97 6月   1 22:36 dncp\_block\_verification.log.curr  -rw-rw-r--. 1 frankwong frankwong 158 6月   1 22:27 VERSION  [frankwong@Slave01 current]$ more VERSION  #Sat Jun 01 22:27:49 CST 2013  namespaceID=2035266975  storageID=DS-663208261-192.168.1.201-50010-1370096869111  cTime=0  storageType=DATA\_NODE  layoutVersion=-32  [frankwong@Slave01 current]$ |

说明

blk\_<id>保存的是HDFS的数据块，其中保存了具体的二进制数据。  
blk\_<id>.meta保存的是数据块的属性信息：版本信息，类型信息，和checksum  
当一个目录中的数据块到达一定数量的时候，则创建子文件夹来保存数据块及数据块属性信息。

**三.文件数据流**

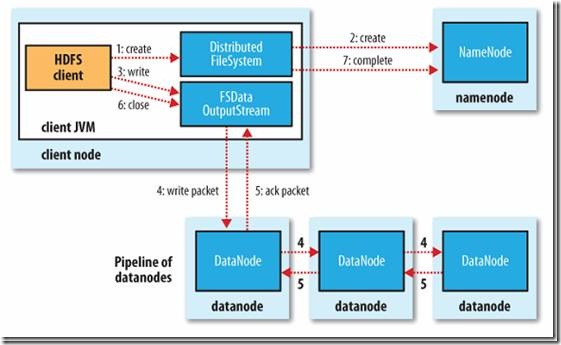
**3.1读文件过程**

[](http://gitsea.qiniudn.com/wp-content/uploads/2013/05/100926154807.png)

文件读取流程

* 客户端(client)用FileSystem的open()函数打开文件
* DistributedFileSystem用RPC调用元数据节点，得到文件的数据块信息。
* 对于每一个数据块，元数据节点返回保存数据块的数据节点的地址。
* DistributedFileSystem返回FSDataInputStream给客户端，用来读取数据。
* 客户端调用stream的read()函数开始读取数据。
* DFSInputStream连接保存此文件第一个数据块的最近的数据节点。
* Data从数据节点读到客户端(client)
* 当此数据块读取完毕时，DFSInputStream关闭和此数据节点的连接，然后连接此文件下一个数据块的最近的数据节点。
* 当客户端读取完毕数据的时候，调用FSDataInputStream的close函数。
* 在读取数据的过程中，如果客户端在与数据节点通信出现错误，则尝试连接包含此数据块的下一个数据节点。
* 失败的数据节点将被记录，以后不再连接。

**3.2 写文件过程**

[](http://gitsea.qiniudn.com/wp-content/uploads/2013/05/100926154727.png)

HDFS写文件过程

* 客户端调用create()来创建文件
* DistributedFileSystem用RPC调用元数据节点，在文件系统的命名空间中创建一个新的文件。
* 元数据节点首先确定文件原来不存在，并且客户端有创建文件的权限，然后创建新文件。
* DistributedFileSystem返回DFSOutputStream，客户端用于写数据。
* 客户端开始写入数据，DFSOutputStream将数据分成块，写入data queue。
* Data queue由Data Streamer读取，并通知元数据节点分配数据节点，用来存储数据块(每块默认复制3块)。分配的数据节点放在一个pipeline里。
* Data Streamer将数据块写入pipeline中的第一个数据节点。第一个数据节点将数据块发送给第二个数据节点。第二个数据节点将数据发送给第三个数据节点。
* DFSOutputStream为发出去的数据块保存了ack queue，等待pipeline中的数据节点告知数据已经写入成功。
* 如果数据节点在写入的过程中失败：
* 关闭pipeline，将ack queue中的数据块放入data queue的开始。
* 当前的数据块在已经写入的数据节点中被元数据节点赋予新的标示，则错误节点重启后能够察觉其数据块是过时的，会被删除。
* 失败的数据节点从pipeline中移除，另外的数据块则写入pipeline中的另外两个数据节点。
* 元数据节点则被通知此数据块是复制块数不足，将来会再创建第三份备份。
* 当客户端结束写入数据，则调用stream的close函数。此操作将所有的数据块写入pipeline中的数据节点，并等待ack queue返回成功。最后通知元数据节点写入完毕。

**链接**

# Hadoop mapreduce

MapReduce中最重要的两个词就是Map（映射）和Reduce（规约）。初看Map/Reduce这两个词，熟悉Function Language的人一定感觉很熟悉。FP把这样的函数称为”higher order function”（”High order function”被成为Function Programming的利器之一哦），也就是说，这些函数是编写来被与其它函数相结合（或者说被其它函数调用的）。如果说硬要比的化，可以把它想象成C里面的CallBack函数，或者STL里面的Functor。比如你要对一个STL的容器进行查找，需要制定每两个元素相比较的Functor（Comparator），这个Comparator在遍历容器的时候就会被调用。

拿前面说过图像处理程序来举例，其实大多数的图像处理操作都是对图像矩阵进行某种运算。这里的运算通常有两种，一种是映射，一种是规约。拿两种效果来说，”老照片”效果通常是强化照片的G/B值，然后对每个象素加一些随机的偏移，这些操作在二维矩阵上的每一个元素都是独立的，是Map操作。而”雕刻”效果需要提取图像边缘，就需要元素之间的运算了，是一种Reduce操作。再举个简单的例子，一个一维矩阵（数组）[0,1,2,3,4]可以映射为[0,2,3,6,8]（乘2），也可以映射为[1,2,3,4,5]（加1）。它可以规约为0（元素求积）也可以规约为10（元素求和）。

面对复杂问题，古人教导我们要“分而治之”，英文中对应的词是”Divide and Conquer“。Map/Reduce其实就是Divide/Conquer的过程，通过把问题Divide，使这些Divide后的Map运算高度并行，再将Map后的结果Reduce（根据某一个Key），得到最终的结果。

Googler发现这是问题的核心，其它都是共性问题。因此，他们把MapReduce抽象分离出来。这样，Google的程序员可以只关心应用逻辑，关心根据哪些Key把问题进行分解，哪些操作是Map操作，哪些操作是Reduce操作。其它并行计算中的复杂问题诸如分布、工作调度、容错、机器间通信都交给Map/Reduce Framework去做，很大程度上简化了整个编程模型。

MapReduce的另一个特点是，Map和Reduce的输入和输出都是中间临时文件（MapReduce利用Google文件系统来管理和访问这些文件），而不是不同进程间或者不同机器间的其它通信方式。我觉得，这是Google一贯的风格，化繁为简，返璞归真。

接下来就放下其它，研究一下Map/Reduce操作。（其它比如容错、备份任务也有很经典的经验和实现，论文里面都有详述）

Map的定义：

Map, written by the user, takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key I and passes them to the Reduce function.

Reduce的定义：

The Reduce function, also written by the user, accepts an intermediate key I and a set of values for that key. It merges together these values to form a possibly smaller set of values. Typically just zero or one output value is produced per Reduce invocation. The intermediate values are supplied to the user’s reduce function via an iterator. This allows us to handle lists of values that are too large to fit in memory.

MapReduce论文中给出了这样一个例子：在一个文档集合中统计每个单词出现的次数。

Map操作的输入是每一篇文档，将输入文档中每一个单词的出现输出到中间文件中去。

map(String key, String value):

// key: document name

// value: document contents

for each word w in value:

EmitIntermediate(w, “1″);

比如我们有两篇文档，内容分别是

A － “I love programming”

B － “I am a blogger, you are also a blogger”。

B文档经过Map运算后输出的中间文件将会是：

I,1

am,1

a,1

blogger,1

you,1

are,1

a,1

blogger,1

Reduce操作的输入是单词和出现次数的序列。用上面的例子来说，就是 (“I”, [1, 1]), (“love”, [1]), (“programming”, [1]), (“am”, [1]), (“a”, [1,1]) 等。然后根据每个单词，算出总的出现次数。

reduce(String key, Iterator values):

// key: a word

// values: a list of counts

int result = 0;

for each v in values:

result += ParseInt(v);

Emit(AsString(result));

最后输出的最终结果就会是：(“I”, 2″), (“a”, 2″)……

实际的执行顺序是：

MapReduce Library将Input分成M份。这里的Input Splitter也可以是多台机器并行Split。

Master将M份Job分给Idle状态的M个worker来处理；

对于输入中的每一个<key, value> pair 进行Map操作，将中间结果Buffer在Memory里；

定期的（或者根据内存状态），将Buffer中的中间信息Dump到本地磁盘上，并且把文件信息传回给Master（Master需要把这些信息发送给Reduce worker）。这里最重要的一点是，在写磁盘的时候，需要将中间文件做Partition（比如R个）。拿上面的例子来举例，如果把所有的信息存到一个文件，Reduce worker又会变成瓶颈。我们只需要保证相同Key能出现在同一个Partition里面就可以把这个问题分解。

R个Reduce worker开始工作，从不同的Map worker的Partition那里拿到数据（read the buffered data from the local disks of the map workers），用key进行排序（如果内存中放不下需要用到外部排序 – external sort）。很显然，排序（或者说Group）是Reduce函数之前必须做的一步。 这里面很关键的是，每个Reduce worker会去从很多Map worker那里拿到X(0<X<R) Partition的中间结果，这样，所有属于这个Key的信息已经都在这个worker上了。

Reduce worker遍历中间数据，对每一个唯一Key，执行Reduce函数（参数是这个key以及相对应的一系列Value）。

执行完毕后，唤醒用户程序，返回结果（最后应该有R份Output，每个Reduce Worker一个）。

可见，这里的分（Divide）体现在两步，分别是将输入分成M份，以及将Map的中间结果分成R份。将输入分开通常很简单，Map的中间结果通常用”hash(key) mod R”这个结果作为标准，保证相同的Key出现在同一个Partition里面。当然，使用者也可以指定自己的Partition Function，比如，对于Url Key，如果希望同一个Host的URL出现在同一个Partition，可以用”hash(Hostname(urlkey)) mod R”作为Partition Function。

对于上面的例子来说，每个文档中都可能会出现成千上万的 (“the”, 1)这样的中间结果，琐碎的中间文件必然导致传输上的损失。因此，MapReduce还支持用户提供Combiner Function。这个函数通常与Reduce Function有相同的实现，不同点在于Reduce函数的输出是最终结果，而Combiner函数的输出是Reduce函数的某一个输入的中间文件。

# Hadoop部署

# Hadoop应用

# 学习建议

* Hadoop权威指南/ Hadoop技术内幕
* 阅读hadoop源码，更直观，更透彻
* 使用“印象笔记”、“有道云笔记”等工具管理知识，将遇到的问题、学到的知识点记录下来
* 主动了解业内新技术：INFQ、CSDN
* <http://subject.csdn.net/hadoop/>: csdn上的hadoop专题页
* <http://hbtc.csdn.net/> ： 2012 Hadoop与大数据技术大会PPT， 2013年的会11月召开
* <http://dongxicheng.org/> ：《Hadoop技术内幕》作者的博客，详解hadoop相关技术