# 一 Hadoop之MapReduce

## 海量数据分流处理技术思想（负载均衡）

### 1传统hash

问题：如何将大数据、流量均匀分布到N台服务器

方案：找到合理的key，hash（key）尽量分布均匀

Hash（key）mod N==0 分到第0台机器

Hash（key）mod N==1 分到第1台机器

Hash（key）mod N==N-1 分到第N-1台机器

传统hash算法存在致命问题，如果某一台机器宕机，那么应该落在该机器的请求就无法得到正确的处理，这时需要将宕掉的服务器使用算法去除，此时候会有(N-1)的服务器的缓存数据需要重新进行计算；如果新增一台机器，会有(N+1)的服务器的缓存数据需要进行重新计算。对于系统而言，这通常是不可接受的颠簸（因为这意味着大量缓存的失效或者数据需要转移）

### 2随机划分

假设有N个节点，随机划分即根据key随机生成一个数，这个数代表节点标号N

### 3一致性hash

一致性hash算法可以有效地解决分布式存储结构下动态增加和删除节点所带来的问题



首先将hash算法的值域映射成一个环形，如0~2^32-1的整数（每个整数代表一个桶空间），每个缓存key取hash转换成整数，映射到该环上，对服务器节点采用相同的hash算法（如根据ip或主机名或网卡等）转为整数映射到环上，key对应的顺时针方向最近的节点就是所属节点

增加节点

（如node4）时，有一小部分key的归属会受到影响，如key2原本属于node2，增加node4后，现在属于node4，原来node2上key2的缓存失效，缓存会在node4上重新生成

删除节点

（如node3），同样有一小部分的key归属会受到影响，如key4原本属于node3，node3被删除后，key4落到node1节点，在node3上的所有缓存key失效（这里只有key4），这些key缓存重新在node1上生成

### 4一致性hash+虚拟节点

一致性hash算法可能存在key分布不均匀的情况，如下图，所有的key都归属以一个节点node2



为了优化这种节点太少导致key分布不均匀的情况，一致性hash引入了虚拟节点的概念。

虚拟节点就是基于原来的物理节点映射成多个子节点，最后把所有子节点映射到环形空间



### 5一致性hash算法实现

## （二）MapReduce

[link\MapReduce过程详解(基于hadoop2.x架构) - py小杰 - 博客园.mhtml](link/MapReduce过程详解(基于hadoop2.x架构) - py小杰 - 博客园.mhtml)





# 二 hive

## hive简介

Hive是一个sql解析引擎，将sql转译成MR job，然后在Hadoop平台运行

### 特点：

#### 1 Hive的元数据依赖于关系型数据库，hive本身不存储数据，数据存储在hdfs中（即文件），完全依赖Hadoop和MapReduce

#### 2 hive不支持对数据的修改和删除

#### Hive元数据之各表简介：

[link\Hive 元数据表结构详解\_hive,元数据,表结构\_阳光Xiao伙的博客-CSDN博客.mhtml](link/Hive 元数据表结构详解_hive,元数据,表结构_阳光Xiao伙的博客-CSDN博客.mhtml)

##### VERSION：存储hive版本，该表中数据只有一条，如果存在多条，会造成hive启动不起来

##### Hive数据库相关的元数据表(DBS、DATABASE\_PARAMS)

DBS ：该表存储Hive中所有数据库的基本信息

DATABASE\_PARAMS ：该表存储数据库的相关参数，在CREATE DATABASE时候用WITH DBPROPERTIES(property\_name=property\_value, …)指定的参数

##### 3）Hive表和视图相关的元数据表

主要有TBLS（该表中存储Hive表，视图，索引表的基本信息）

TABLE\_PARAMS（该表存储表/视图的属性信息）

TBL\_PRIVS（该表存储表/视图的授权信息）

这三张表通过TBL\_ID关联

##### 4）Hive文件存储信息相关的元数据表

主要涉及SDS、SD\_PARAMS、SERDES、SERDE\_PARAMS，由于HDFS支持的文件格式很多，而建Hive表时候也可以指定各种文件格式，Hive在将HQL解析成MapReduce时候，需要知道去哪里，使用哪种格式去读写HDFS文件，而这些信息就保存在这几张表中

##### 5）Hive表字段相关的元数据表

主要涉及COLUMNS\_V2，该表存储表对应的字段信息

##### 6）Hive表分分区相关的元数据表

主要涉及PARTITIONS、PARTITION\_KEYS、PARTITION\_KEY\_VALS、PARTITION\_PARAMS

## Hive的sql与传统sql区别





## Hive体系架构



## Hive数据管理

### 内部表和外部表

未被external修饰的是内部表（managed table），被external修饰的为外部表（external table）；

区别：

内部表数据由Hive自身管理，外部表数据由HDFS管理；

内部表数据存储的位置是hive.metastore.warehouse.dir（默认：/user/hive/warehouse），外部表数据的存储位置由自己制定（如果没有LOCATION，Hive将在HDFS上的/user/hive/warehouse文件夹下以外部表的表名创建一个文件夹，并将属于这个表的数据存放在这里）；

删除内部表会直接删除元数据（metadata）及存储数据；删除外部表仅仅会删除元数据，HDFS上的文件并不会被删除；

对内部表的修改会将修改直接同步给元数据，而对外部表的表结构和分区进行修改，则需要修复（MSCK REPAIR TABLE table\_name;）

### Partition（分区）

### Bucket（分桶）

作用：方便抽样，提高join查询效率

### 数据类型

### Hive的窗口函数

[link\Hive 之 分析窗口函数 - happy19870612's blog - CSDN博客.mhtml](link/Hive 之 分析窗口函数 - happy19870612's blog - CSDN博客.mhtml)

#### 5.1分析函数定义

分析函数又被称为窗口函数。在联机分析(OLAP)的时候，传统是SQL可能需要进行多个自连接，从而导致查询新能很差。又或者有时候需要多维分析需要在不同层级进行聚合，并且聚合后数据行和密钥聚合的数据行都必须返回。传统是SQL都有一定的局限性。分析函数很好的可以解决这些问题，可以不使用自联结就能在一行中取出聚合和非聚合的行

#### 5.2分析函数组成

function(argument1, argument2…)over([partition-by-cluase][order-by-clasue][windowing-clause])

分析函数有三个部分组成：分区子句，排序子句，以及窗口子句。

分区子句： 其实可以理解为根据什么分组

排序子句：对每一个分区的结果集排序

窗口子句：

[ROWS|RANGE] BETWEEN <开始表达式> AND <结束表达式>

<开始表达式>:[UNBOUNDED PRECEDING| CURRENT ROW|

nRECEDING | n FOLLOWING]

<结束表达式>:[UNBOUNDED FOLLOWING|CURRENT ROW|

nRECEDING | n FOLLOWING]

UNBOUNDEDPRECEDING：针对当前分区里面的当前行而言的前面所有行

UNBOUNDEDFOLLOWING：针对当前分区里面的当前行而言的后面所有行

CURRENTROW： 当前行

nRECEDING： 针对当前行而言，前n行

nFOLLOWING：针对当前行而言，后n行

## hive的优化

[link\Hive优化（整理版） - 牧梦者 - 博客园.mhtml](link/Hive优化（整理版） - 牧梦者 - 博客园.mhtml)

1 当任务有大量小文件时，考虑合并小文件，减少map和reduce数量

2 当input的文件都很大，任务逻辑复杂，map执行非常慢的时候，可以考虑增加Map和reduce数

# 三 Hadoop1和2的区别

## （一）slot的含义

1）slot就是槽的意思，是一个资源单位，只有给task分配了一个slot之后，这个task才可以运行。slot分两种，map slot沪蓉reduce slot。另外，slot是一个逻辑概念，一个数据节点的slots数量既不是CPU的核数，也不是memory chip，一个节点的slot数量用来表示此节点的资源容量或是计算计算能力的大小，也就是说slot是hadoop的资源单位。

2）系统中map slots总数与reduce slots总数的计算方式如下:

map slots 总数=集群节点数\*mapred.tasktracker.map.tasks.maximum（默认是2）;

reduce slots 总数=集群节点数\*mapred.tasktracker.reduce.tasks.maximum（默认是2）;

# 四 yarn

# 五spark

## 简介

Spark是加州大学伯克利分校AMP实验室（Algorithms, Machines, and People Lab）开发的通用内存并行计算框架。Spark使用Scala语言进行实现，它是一种面向对象、函数式编程语言，能够像操作本地集合对象一样轻松地操作分布式数据集，具有以下特点:

1.运行速度快：Spark拥有DAG执行引擎，支持在内存中对数据进行迭代计算。官方提供的数据表明，如果数据由磁盘读取，速度是Hadoop MapReduce的10倍以上，如果数据从内存中读取，速度可以高达100多倍。

2.易用性好：Spark不仅支持Scala编写应用程序，而且支持Java和Python等语言进行编写，特别是Scala是一种高效、可拓展的语言，能够用简洁的代码处理较为复杂的处理工作。

3.通用性强：Spark生态圈即BDAS（伯克利数据分析栈）包含了Spark Core、Spark SQL、Spark Streaming、MLLib和GraphX等组件，这些组件分别处理Spark Core提供内存计算框架、SparkStreaming的实时处理应用、Spark SQL的即席查询、MLlib或MLbase的机器学习和GraphX的图处理。

4.随处运行：Spark具有很强的适应性，能够读取HDFS、Cassandra、HBase、S3和Techyon为持久层读写原生数据，能够以Mesos、YARN和自身携带的Standalone作为资源管理器调度job，来完成Spark应用程序的计算

## （一）批处理和流处理

### 1批处理系统

***批处理***在大数据世界有着悠久的历史。批处理主要操作大容量静态数据集，并在计算过程完成后返回结果。

#### 批处理模式中使用的数据集通常符合下列特征：

有界：批处理数据集代表数据的有限集合

持久：数据通常始终存储在某种类型的持久存储位置中

大量：批处理操作通常是处理极为海量数据集的唯一方法

Apache Hadoop是一种专用于批处理的处理框架，Hadoop的处理功能来自MapReduce引擎。MapReduce的处理技术符合使用键值对的map、shuffle、reduce算法要求。

#### 使用场景：

需要处理大量数据的任务通常最适合用批处理操作进行处理（准确性要求较高），大量数据的处理需要付出大量时间，因此批处理不适合实时性要求较高的场合

### 2流处理系统

流处理中的数据集是“无边界”的，流处理方式无需针对整个数据集执行操作，而是对通过系统传输的每个数据项执行操作，即处理瞬时数据，实时性要求较高

## spark之批处理和流处理模式

### 1批处理模式

与MapReduce不同，Spark的数据处理工作全部在内存中进行，只在一开始将数据读入内存，以及将最终结果持久存储时需要与存储层交互。所有中间态的处理结果均存储在内存中。

为了实现内存中批计算，Spark会使用一种名为Resilient Distributed Dataset（弹性分布式数据集），即RDD的模型来处理数据。这是一种代表数据集，只位于内存中，永恒不变的结构。针对RDD执行的操作可生成新的RDD。每个RDD可通过世系（Lineage）回溯至父级RDD，并最终回溯至磁盘上的数据。Spark可通过RDD在无需将每个操作的结果写回磁盘的前提下实现容错

### 2流处理模式

流处理能力是由Spark Streaming实现的。Spark本身在设计上主要面向批处理工作负载，为了弥补引擎设计和流处理工作负载特征方面的差异，Spark实现了一种叫做微批（Micro-batch）\*的概念

Spark Streaming会以亚秒级增量对流进行缓冲，随后这些缓冲会作为小规模的固定数据集进行批处理。这种方式的实际效果非常好，但相比真正的流处理框架在性能方面依然存在不足

### 3优势和使用场景

#### 3.1优势

1）使用Spark而非Hadoop MapReduce的主要原因是速度。在内存计算策略和先进的DAG调度等机制的帮助下，Spark可以用更快速度处理相同的数据集。

2）Spark的另一个重要优势在于多样性。该产品可作为独立集群部署，或与现有Hadoop集群集成。该产品可运行批处理和流处理，运行一个集群即可处理不同类型的任务。

3）除了引擎自身的能力外，围绕Spark还建立了包含各种库的生态系统，可为机器学习、交互式查询等任务提供更好的支持。相比MapReduce，Spark任务更是“众所周知”地易于编写，因此可大幅提高生产力。

#### 3.2使用场景

1. 为流处理系统采用批处理的方法，需要对进入系统的数据进行缓冲。缓冲机制使得该技术可以处理非常大量的传入数据，提高整体吞吐率，但等待缓冲区清空也会导致延迟增高。这意味着Spark Streaming可能不适合处理对延迟有较高要求的工作负载。

（对于重视吞吐率而非延迟的工作负载，则比较适合使用Spark Streaming作为流处理解决方案）

2）由于内存通常比磁盘空间更贵，因此相比基于磁盘的系统，Spark成本更高。然而处理速度的提升意味着可以更快速完成任务，在需要按照小时数为资源付费的环境中，这一特性通常可以抵消增加的成本。

3）Spark内存计算这一设计的另一个后果是，如果部署在共享的集群中可能会遇到资源不足的问题。相比Hadoop MapReduce，Spark的资源消耗更大，可能会对需要在同一时间使用集群的其他任务产生影响。从本质来看，Spark更不适合与Hadoop堆栈的其他组件共存一处。

## （三）spark vs MapReduce（spark计算框架为什么比MR块）

1 数据存储结构

MR中，每个map处理一个hdfs的数据切片；spark中，使用内存构建弹性分布式数据集RDD，对数据进行运算和cache

2 进行迭代计算时，MR会将结果存储到hdfs（磁盘IO），而spark中间结果存储在内存，存取速度是磁盘的多个数量级



3 MR中，一个task对应一个进程，spark中一个task对应一个线程，而计算机启动进程比启动线程慢，因为启动进程还要为其分配CPU和内存等资源