# 关键技术研究基础与现状

## 2.1 网络爬虫技术概述

### 2.1.1 HTTP协议简介

### 2.1.2网络爬虫抓取策略

2.1.3分布式在网络爬虫技术中的应用

2.2基于分布式的大数据处理技术

### 2.2.1大数据和分布式技术

随着互联网的飞速发展，数据量的指数级增长一方面给用户带来海量的信息获取来源的同时，另一方面也给互联网产业带来了巨大的挑战，传统的单机作业无法适用于PB级别的海量数据，单纯的升级单机配置是一种性价比很低的解决方案。于此同时，横向扩展的分布式技术孕育而生。分布式技术即利用分散的多台廉价机器，在核心节点的控制调度下，完成任务的分割，分配，通过每台机器完成各自的任务并最终汇聚出正确的结果。

分布式技术包括分布式计算、分布式存储、分布式应用等方面。

### 2.2.2 MapReduce和分布式存储技术

2003年到2004年间，Google发表了MapReduce、GFS（Google File System）和BigTable三篇技术论文，提出了一套全新的分布式计算理论。

MapReduce是分布式计算框架，GFS（Google File System）是分布式文件系统，BigTable是基于Google File System的数据存储系统，这三大组件组成了Google的分布式计算模型。Google的分布式计算模型相比于传统的分布式计算模型有三大优势：首先，它简化了传统的分布式计算理论，降低了技术实现的难度，可以进行实际的应用。其次，它可以应用在廉价的计算设备上，只需增加计算设备的数量就可以提升整体的计算能力，应用成本十分低廉。最后，它被Google应用在Google的计算中心，取得了很好的效果，有了实际应用的证明。

MapReduce用于大规模数据集（大于1TB）的并行运算。概念“Map（映射）”和“Reduce（归纳）”，及他们的主要思想，都是从函数式编程语言以及矢量编程语言特性借鉴来的，暨通过用户定义的map函数将输入分割成key/value对，然后处理该数据，最终通过Reduce函数将处理完成的记过合并。

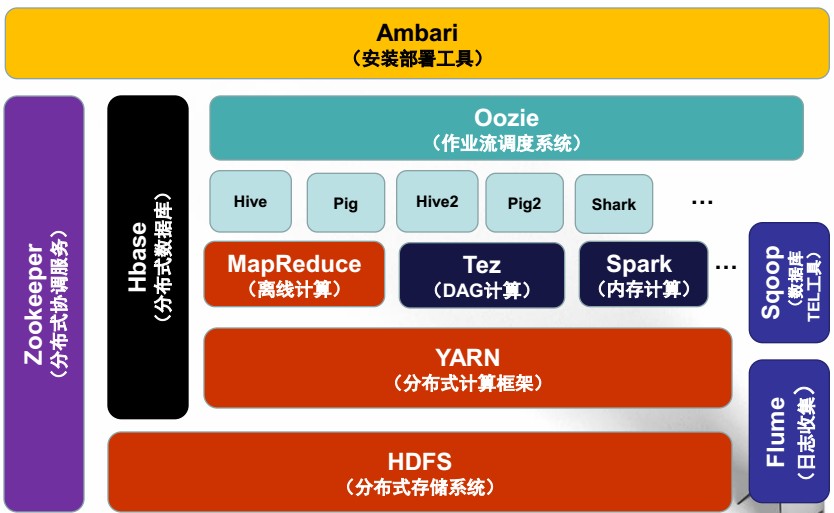
分布式文件系统在信息爆炸时代中为人们可以获取的数据成指数倍的增长的情形下提供了存储的可能，单纯通过增加硬盘个数来扩展计算机文件系统的存储容量的方式，在容量大小、容量增长速度、数据备份、数据安全等方面的表现都差强人意。分布式文件系统可以有效解决数据的存储和管理难题：将固定于某个地点的某个文件系统，扩展到任意多个地点/多个文件系统，众多的节点组成一个文件系统网络。每个节点可以分布在不同的地点，通过网络进行节点间的通信和数据传输。人们在使用分布式文件系统时，无需关心数据是存储在哪个节点上、或者是从哪个节点从获取的，只需要像使用本地文件系统一样管理和存储文件系统中的数据。

分布式存储系统通常具体良好的横向扩展能力，利用数据冗余带来的高可用性，并且可以提供弹性的存储特性。

### 2.2.3大数据处理平台Hadoop简介

Hadoop是利用MapReduce为基础的分布式计算框架，并根据GFS开发了HDFS分布式文件系统，根据BigTable开发了HBase数据存储系统。Hadoop的开源特性使其成为分布式计算系统的事实上的国际标准。Yahoo，Facebook，Amazon以及国内的百度，阿里巴巴等众多互联网公司都以Hadoop为基础搭建自己的分布式计算系统。

Hadoop采用了master/slave的分布式架构方案，其中MapReduce对应jobtracker和tasktracker，HDFS包含namenode和datanode，jobtracker和namenode对应于master节点，tasktracker和datanode对应于slave节点。以MapReduce和HDFS为核心，加上YARN的资源分配组件，Hadoop框架迅速发展并形成了庞大的技术生态圈，如下图所示：



## 2.3基于DPI数据的Web用户行为分析

### 2.3.1 Web用户行为概述

### 2.3.2 DPI简介

### 2.3.3网络访问流量分析工具WireShark介绍

### 2.3.4 Web用户行为分析与用户画像构建

## 2.4 用户兴趣推荐系统概述

## 2.4.1 用户兴趣推荐系统的意义

## 2.4.2 常见推荐模型

2.4.3 推荐系统的评价指标

# 第四章 电商用户行为分析

## 4.1

# 第五章 基于Hadoop平台的电信DPI数据处理

4.1 Hadoop平台相关组件和技术原理

4.1.1 MapReduce原理



其处理过程为：

#### MapReduce Library首先将输入文件切割成多个小片的文件pieces，然后MapReduce Library将启动复制操作，将用户程序复制到各个a cluster of machines上。

#### 在这些a cluster of machines中，其中一个比较特殊称之为master，其他的machine被称之为worker，master选择空闲的worker并将任务（map任务或者是reduce任务）分配给这些空闲的worker任务。

#### 一个worker如果被master分配了map任务的话，该worker首先读取该key/value对，然后执行用户定义的map函数，这些处理完成的key/value对被缓存到内存中。

#### 然后，将这些key/value对写入本地磁盘，然后worker通知master。

#### 如果master接收到了worker在第4步的通知之后，master将这个信息传递给reduce worker，该reduce worker通过远程系统调用的形式读取该worker磁盘上存储的处理完的数据。

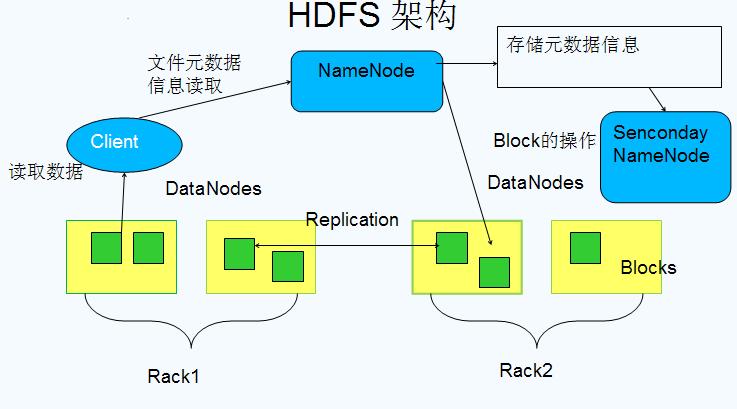
#### Reduce worker遍历已排序的数据，然后将数据传递到用户定义的Reduce函数。

#### 当所有的map和reduce完成之后，然后master唤醒用户程序。

通过MapReduce运行流程可以看出，用户程序仅仅需要编写Map函数和Reduce函数即可，MapReduce库首先通过调用用户自定义的Map函数，将输入文件分割，如果数据处理完成，将调用Reduce函数将结果合并起来。这样大大简化了用户应用程序编写的复杂度。

4.1.2 HDFS架构原理和配置优化

HDFS是Hadoop平台存储的核心技术，其架构如下图所示：



Namenode：

Namenode 上保存着 HDFS 的名字空间。对于任何对文件系统元数据产生修改的操作， Namenode 都会使用一种称为 EditLog 的事务日志记录下来。例如，在 HDFS 中创建一个文件， Namenode 就会在 Editlog 中插入一条记录来表示；同样地，修改文件的副本系数也将往 Editlog 插入一条记录。 Namenode 在本地操作系统的文件系统中存储这个 Editlog 。整个文件系统的名 字空间，包括数据块到文件的映射、文件的属性等，都存储在一个称为 FsImage 的文件中，这 个文件也是放在 Namenode 所在的本地文件系统上。

Namenode 在内存中保存着整个文件系统的名字空间和文件数据块映射 (Blockmap) 的映像 。这个关键的元数据结构设计得很紧凑，因而一个有 4G 内存的 Namenode 足够支撑大量的文件 和目录。当 Namenode 启动时，它从硬盘中读取 Editlog 和 FsImage ，将所有 Editlog 中的事务作 用在内存中的 FsImage 上，并将这个新版本的 FsImage 从内存中保存到本地磁盘上，然后删除 旧的 Editlog ，因为这个旧的 Editlog 的事务都已经作用在 FsImage 上了。这个过程称为一个检查 点 (checkpoint) 。在当前实现中，检查点只发生在 Namenode 启动时，在不久的将来将实现支持 周期性的检查点。

DataNode：

Datanode 将 HDFS 数据以文件的形式存储在本地的文件系统中，它并不知道有 关 HDFS 文件的信息。它把每个 HDFS 数据块存储在本地文件系统的一个单独的文件 中。 Datanode 并不在同一个目录创建所有的文件，实际上，它用试探的方法来确定 每个目录的最佳文件数目，并且在适当的时候创建子目录。在同一个目录中创建所 有的本地文件并不是最优的选择，这是因为本地文件系统可能无法高效地在单个目 录中支持大量的文件。

当一个 Datanode 启动时，它会扫描本地文件系统，产生一个这些本地文件对应 的所有 HDFS 数据块的列表，然后作为报告发送到 Namenode ，这个报告就是块状态 报告。

通常可以通过配置Secondary NameNode提升HDFS的高可用性，这样在Primary NameNode宕机后，依然可以正常的提供服务。配置Secondary NameNode的关键文件为：

conf/masters、conf/hdfs-site.xml、core-site.xml。

4.2.2 Hbase原理

HBase是一个构建在HDFS上的分布式列存储系统是基于Google BigTable模型开发的，典型的key/value系统； HBase是Apache Hadoop生态系统中的重要一员，主要用于海量结构化数据存储； 从逻辑上讲，HBase将数据按照表、行和列进行存储。 与hadoop一样，Hbase目标主要依靠横向扩展，通过不断增加廉价的商用服务器，来增加计算和存储能力。

Hbase表特性如下表格所示：

|  |  |
| --- | --- |
| 特性 | 描述 |
| 容量巨大 | 一个表可以有数十亿行，上百万列 |
| 无模式 | 每行都有一个可排序的主键和任意多的列 |
| 面向列 | 面向列（族）的存储和权限控制，列（族）独立检索 |
| 稀疏 | 空列并不占用存储空间，可以设计出非常稀疏的表 |
| 数据多版本 | 可维持单元中数据的多个版本，以插入时的时间戳为单位 |
| 数据类型单一 | Hbase表中数据都统一为字符串 |

Hbase中的基本概念：

## 4.1 电信DPI数据预处理

### 4.1.1电信DPI数据清洗与格式化

电信DPI原始数据中通常含有大量的非用户主动访问的流量，如包含jpg/gif图片地址的请求，js/css等脚本样式引用文件的请求，以及flash等广告插件的请求。针对这些请求流量，可以通过匹配请求后缀的方式将其清理。

另外，针对大量存在的POST方式的请求，需要根据Http报文请求头部将其过滤出。

针对DPI数据的完整性，需要将缺失字段的数据滤出。

### 4.1.2 电信DPI数据结构

电信DPI数据结构包含12个字段，每个字段由“|”分隔，字段涵义如下表所示：

//插入表格

### 4.1.3用户会话识别

### 4.1.4电商用户数据过滤与分类

针对于电商用户的过滤，根据DPI数据中domain字段进行判断，如果该字段在预先定义的电商列表中，将其放入HDFS的e-commercial文件路径下，并建立起以电商名为名称的文件；如果不在，则将其放入other目录下。

电商列表的定义为：

该过程流程图如下所示：

4.2 MapReduce处理过程原理如下：

4.2. 基于DPI数据基本统计的Hadoop编程设计

4.2.1 不同电商

4.3 基于DPI数据构建电商用户画像的Hadoop编程设计

4.4. DPI处理