中小型企业基于大数据技术的项目实践

前言

我们这次 Chat 主要交流的主题是：中小型企业基于大数据技术的项目实践,笔者将就大数据技术栈开始说起，同时，在后面的内容中，将涉及笔者在工程实践中的一些具体经验。

下面，我们将从大数据技术的干货介绍开始，这部分内容对于有基础的童鞋来说，可以快速略过。

大数据技术初探

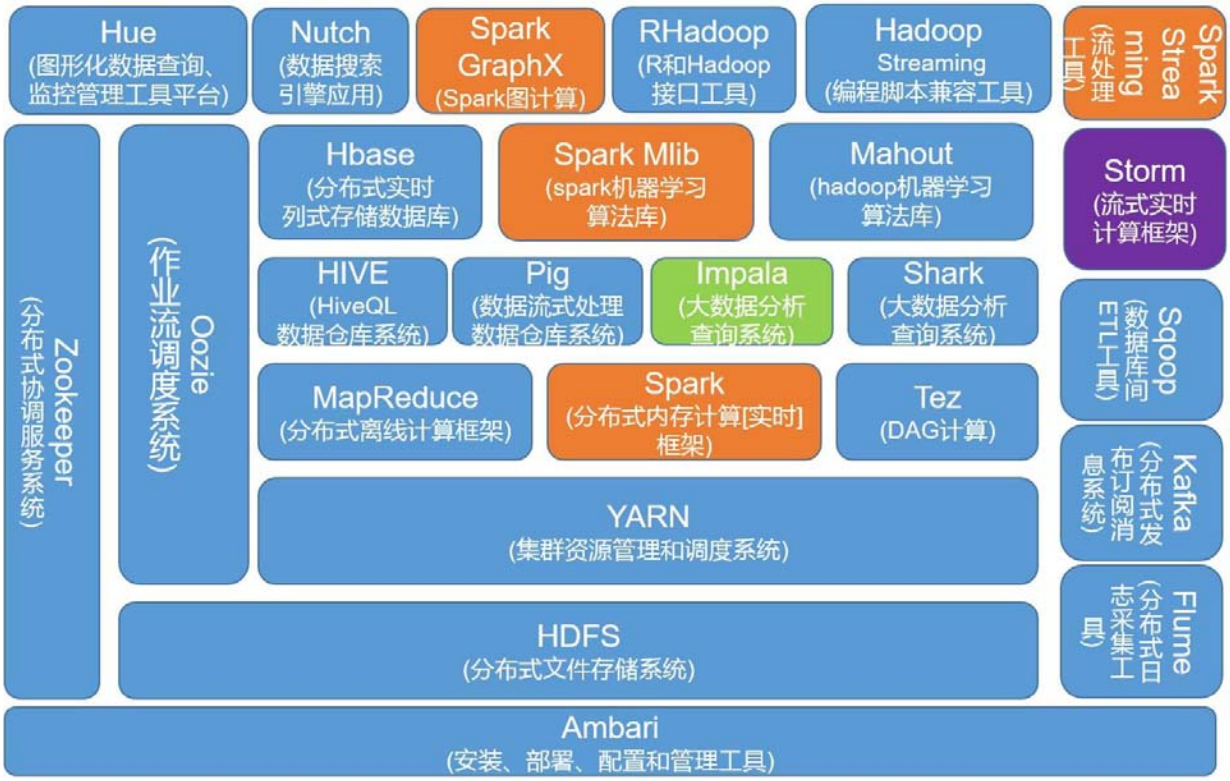
准确来说“大数据”这个概念并不存在，其就是在曾经我们提到过的“海量数据”的基础上，数据量级再一次增大，导致传统的处理手段无法进行及时、有效的处理。为了表征与传统数据处理手段的区别，表明技术的先进性，提出来了一个新词——”大数据”。

GitChat

作为 DT 时代的代表技术之一，大数据紧紧地与人工智能，云计算技术相结合，三者相辅相成，共同促进产业变革，技术进步。无论在学术界还是工业界，这“三驾马车”无疑都是最热门和前沿的。

大数据技术是近几年火起来的一项技术，主要应用场景是日志收集与处理，数据分析，机器学习模型的训练等。基于这些，我们可以实现商业智能（BI）、科学决策等。

我们所谓的大数据技术栈，无外乎也就是 Hadoop 生态系统，Hadoop 生态系统如下图所示:



上图所示，是 Hadoop 生态系统的一个示意图。那么，作为一个大数据工程师，是否有必要都要掌握上述内容呢？答案是否定的！

大数据技术主要表现在：

GitChat

1. 大规模数据存储
2. 弹性计算
3. 集群资源调度
4. 数据收集
5. 集群一致性保证

笔者，将针对上面的内容，逐步展开探讨。

大规模数据存储

网盘就是一个典型的大数据存储应用。毫无疑问，网盘上存储的数据量是海量的，这需要一个集群去存储，也就是我们说的云存储。

类似地，我们在工业实践中，也会遇到各种各样数据，这些数据有些是冷数据，也有的是热数据。但是，无论是冷的还是热的，只要是存储意义的数据我们必然要给他存储起来，以便后续使用。

举个例子，一个访问量大的网站，每天产生的日质量是很大的，这些数据我们可以存储起来，以便后续使用。

Hadoop 的 HDFS 可以认为是实际上的工业标准，其存储模式是文件分块存储，多机备份（冗余），通过 standby 节点来进行心跳探测，保证可用性。

除了 HDFS，我们使用云产品的时候，可能也会用亚马逊的公有云产品，也即是 AWS 的 S3 存储系统。

由于笔者所在公司的业务是面向海外市场的，云服务选择的是 AWS，用的云存储是亚马逊的 S3，免去了自己部署 Hadoop HDFS 的过程。Hadoop 的HDFS是自带读取AWS S3的API的。

但是，值得说明的是，Hadoop 的 HDFS 并不太适合频繁更改，或者是海量的小文件存储，毕竟一个文件块就很大了，有的版本默认是 128M，有的是64M，海量小文件，一般使用的是 FastDFS 或者淘宝开源的 TFS。

弹性计算

所谓弹性计算，也就是之前学术界所说的网格计算，现在很流行的分布式计算。我们知道，单节点的算力是有限的，包括超级计算机的架构也是上千个 CPU 和 GPU 们组成的。

我们在平时使用的时候，自然不会设计出超级计算机这样复杂的硬件基础设施，我们通过 TCP/IP 协议来传送数据，在不同的节点上进行并行计算，最后再讲结果汇总，这种算法我们叫做 map/reduce 算法。

这种理念是 Google 提出来的，有兴趣的大家可以去 **Google 学术**下载一下 Google 大数据三篇论文。其是大数据技术的一个奠基。

GitChat

Hadoop 有三个组件，用于大规模数据存储的 HDFS，用于分布式计算的 Map/Reduce 引擎，和资源调度 Yarn。

只不过 Hadoop 的同名计算引擎 MapReduce 在涉及到中间数据缓存的时候，要写入 HDFS 上，我们知道 HDFS 本身就是建立在外存上的，而且还要有冗余备份，整个读取和写入速度都比较慢，所以，现在真正使用的就是 Spark 计算引擎，MR（MapReduce）引擎都快被废掉了。

Spark 是一个通用的计算引擎，其除了核心 Core，为应用层封装了机器学习，图计算，流式计算框架和 SparkSQL 即席查询四个模块，用起来很是方便，我们在实际工程中，用的最多的也就是 Spark 了。

Spark 与 Hadoop 的 MR 引擎不同的是，Spark 的中间数据存储在内存中，所以速度特别快。但是，Spark 的内存要求比较大，不过，内存毕竟也不算太贵嘛。

集群资源调度

所谓的资源调度，主要指的就是 CPU 和内存资源的调度，集群中哪台节点比较闲，就给他多点任务，这样，可以使整体的集群负载均衡，这对于分布式集群来说是十分重要的，直接影响了集群的计算性能。

Hadoop 自带的模块是 Yarn，Spark 也自带一个，叫做 mesos，不过，我们说，Spark 是 Hadoop 生态系统中的成员，自然而然 Spark 也可以使用 Hadoop 的 Yarn 资源调度引擎，

避免了部署上的麻烦。

数据收集

数据分为流式数据和批处理数据。

所谓的流式数据是像流水一样的数据，通常用的计算引擎是Spark Streaming和Storm，我们公司主要用到的是Spark Streaming。

二者的区别就是，Spark Streaming不是严格意义的实时，是一种准实时，每隔一段时间来对收集到的数据运算一次，这样达到一种流式计算的效果，而Storm是严格意义的实时，来一条数据处理一条。

对于我们公司来讲，不需要这么实时的效果，同时Spark streaming直接就用Spark框架编写就ok了，团队成员的技术栈比较吻合，避免了再次学习Storm的成本，也减少了版本发布和维护上的苦难。

但是具体的选型，还要结合公司的实际情况。

说到流式数据的收集，我们不得不提到 Kafka 这个消息中间件。其是发布/订阅模式的，可以用来做流式数据收集的消息队列，起到缓存与缓冲的作用，详细介绍请单击这里。

这是一整套流式数据处理的架构，在网上找到这几篇博文，感觉还可以，推荐给大家：

> http://shiyanjun.cn/archives/1097.html
>
> http://spark.apache.org/docs/latest/streaming-kafka-integration.html

GitChat

除此之外，再介绍一个叫做flume的东西，他的官方介绍是：

> Apache Flume is a distributed, reliable, and available system for efficiently collecting, aggregating and moving large amounts of log data from many different sources to a centralized data store.
>
> The use of Apache Flume is not only restricted to log data aggregation. Since data sources are customizable, Flume can be used to transport massive quantities of event data including but not limited to network traffic data, social-media-generated data, email messages and pretty much any data source possible.

flume多用作日志的收集，常用来收集诸如nginx日志等等，其配合kafka使用，可以做到数据的流式收集。

具体的架构使用，请单击这里参见博文介绍。

集群一致性保证

我们知道作为一个集群，一致性是应高考虑的一个重要因素。

例如，我们在一个集群上两个不同节点读取到的数据不一样，那么我们是相信谁的？很容易就无法做出下一步的处理。

所以，我们在上面的Hadoop生态系统的图示中可以看到一个贯穿始终的叫做zookeeper的东西，这个东西就是用来保障集群一致性的。

Zookeeper主要提供的是Java API，他是通过观察者模式来实现的，不同节点注册一个watcher，来监听事件。

它实现了paxos算法，paxos算法是一个比较复杂的算法，整个算法的推倒与证明过程一页A4纸都写不下。

Zookeeper实现的paxos算法也是fast paxos，或者说是paxos算法的精简版本。通过zookeeper我们可以保证整个集群的一致性，也就为后来基于zookeeper的应用提供了高可用（HA）的基础。

大数据技术工程实践

笔者以大数据技术使用的一个典型场景为例，展开探讨，场景描述：

GitChat

> 应用场景是针对一款app的日志分析，该app的架构方式是基于HTTP的微服务，app算是典型的社交软件。包括聊天，更新状态，群组讨论，更新个人信息等都是通过调用HTTP接口来实现的，当然，这些内容都是加密过的，包括服务器之间的通讯也都是通过证书来验证的。

这样的微服务架构就为我们的日志分析提供了方便，可以认为，日志上的url路径包含了很多的信息，基于不同的url我们可以发现用户的行为，并针对用户的行为进行数据分析。

数据的收集

如果是做离线计算的，可以直接把日志下载到本机，然后再对本机上的所有日志进行统一的计算；

Spark是支持AWS S3的，不过这得基于Hadoop来实现，还得安装Hadoop,在实际使用中坑很多。Spark 读取S3数据可以使用亚马逊官方的Java driver来做，相对来说坑比较少。

不过，Spark直接读取HDFS上的数据相对容易很多，坑也没有多少；
在实际使用的时候，可以尝试用流式日志下载的方式，在下载的同时，进行数据的分析，实际上还是比较高效的。

数据的 ETL

ETL（ Extract-Transform-Load ）用来描述将数据从来源端经过抽取（ extract ）、转换（ transform ）、加载（ load ）至目的端的过程。

ETL的方式有很多，有基于现有用具进行ETL的，也有自己编写代码进行ETL的。

笔者所采用的ETL方式是基于Spark的ETL，基于Spark的ETL有诸如灵活快速等特点，这里有几篇博文，介绍了Spark的ETL，总的来说，用Spark来做ETL还是比较高大上的。

> http://blog.csdn.net/u011204847/article/details/51247306
>
> http://blog.csdn.net/zbc1090549839/article/details/54407876

上面说到，笔者的日志数据存储在AWS 的S3上，故而介绍写AWS S3的日志格式:
原文链接请单击这里。

> s3文件的路径格式：
>
> bucket[/prefix]/AWSLogs/aws-account-id/elasticloadbalancing/region/yyyy/mm/dd/aws-account-id_elasticloadbalancing_region_load-balancer-name_end-time_ip-address_random-string.log
>
> 日志的存储格式：
>
> timestamp elb client:port backend:port request_processing_time backend_processing_time response_processing_time elb_status_code backend_status_code received_bytes sent_bytes “request” “user_agent” ssl_cipher ssl_protocol

GitChat

总之，就是包括了用户的请求IP，请求设备，时间，请求方法，请求路径和服务器的相应和处理时间等等。

这里居然还有专门针对AWS 日志的分析系统的博文，详见这里。

我们的目标是利用spark将这种存储于亚马逊S3的原始日志格式进行转换，存储在数据仓库中。

对于数据仓库，比较著名的应该是HBase了，HBase是基于HDFS的一个NoSQL列式数据库，存储容量大。

不过，对我我的业务场景来说，选用HBase并不太适合，因为很多数据存储很长时间并没有必要，最多只需要存储最近一个月的经过ETL后的数据就可以了，没有必要存储那么多冷数据，所以，我选择了MongoDB进行数据的存储。

那么我们就明确了ETL的目标，将来自于AWS S3的原始数据（raw log）经过ETL，存储在MongoDB中，MongoDB中存储的格式类似于:

```
{
    "time":"2017-2-1-26 UTC xx:xx:xx",
    "url":"http://foo.com/ab?c=d&e=f",
    "uri":"ab",
    "uid":"10000"
}
```

MongoSpark

MongoDB和Spark之间是可以用来做高速地数据传输的，我们使用MongoDB来作为Spark的数据持久层，MongoDB的Spark driver名称就叫做MonogSpark。

这有一篇文章，非常详细地介绍了MongoDB和Spark用于大数据解决方案的架构设计，很推荐，详见这里。

原文部分内容摘要

HDFS VS MongoDB

GitChat

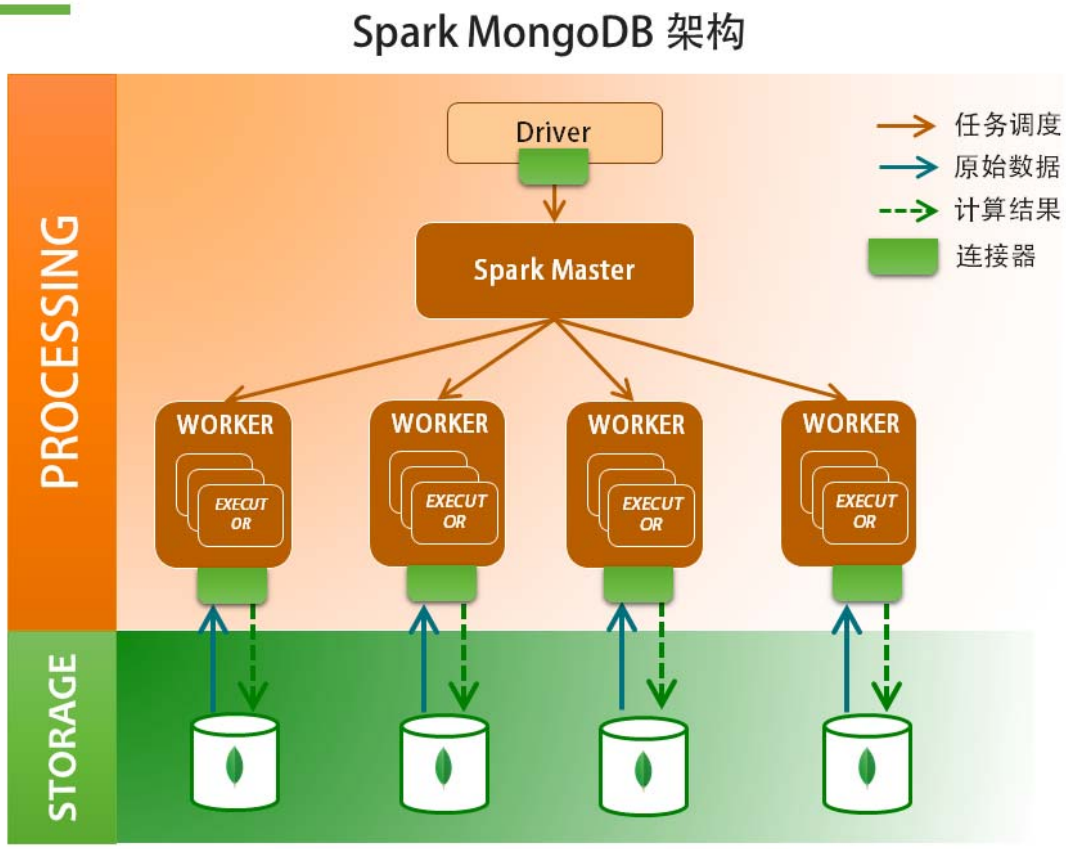
既然我们说MongoDB可以用在HDFS的地方，那我们来详细看看两者之间的差异性。

在说区别之前，其实我们可以先来注意一下两者的共同点。HDFS和MongoDB都是基于廉价x86服务器的横向扩展架构，都能支持到TB到PB级的数据量。数据会在多节点自动备份，来保证数据的高可用和冗余。两者都支持非结构化数据的存储，等等。

HDFS和MongoDB的区别

- 如在存储方式上 HDFS的存储是以文件为单位，每个文件64MB到128MB不等。而MongoDB则是细颗粒化的、以文档为单位的存储。
- HDFS不支持索引的概念，对数据的操作局限于扫描性质的读，MongoDB则支持基于二级索引的快速检索。
- MongoDB可以支持常见的增删改查场景，而HDFS一般只是一次写入后就很难进行修改。
- 从响应时间上来说，HDFS一般是分钟级别而MongoDB对手请求的响应时间通常以毫秒作为单位。

MongoDB-Spark架构



GitChat

什么时候选用MongoDB

- 涉及到快速读取数据
- 建立索引
- 对数据的存储粒度要求较细（文档形式）
- 能够对数据进行修改的场合。

什么时候选用HDFS

- HDFS数据存储节点不要求就有较大的内存，而MongoDB要想保证读写迅速的前提是要占据较大的内存空间；
- 对数据修改的要求不高，例如图片，音视频文件，一般写入后不需要再次修改；
- HDFS被设计部署在低廉的硬件设备上，对硬件的要求不苛刻，能够保证高可用性，集群的数据吞吐量也很高；
  相比之下，MongoDB对CPU和内存的要求要高得多。

MongoDB的地理位置搜索

MongoDB具有很多高级搜索功能，譬如微信搜索附近的人，我们可以通过MongoDB的GEO搜索来完成，这是MongoDB的又一大好处，有关地理位置搜索，推荐这篇博文：

> http://blog.csdn.net/wang7807564/article/details/78863591

数据的分析

我们首先来回顾一下，日志中主要包括的内容有：

在我们的日志url中记录了用户的id，用户的行为，用户的行为属性，用户的设备，用户的IP，用户访问时间，服务器处理时间，服务器响应时间等等。

上述数据是来自日志的原始数据，经过ETL后，被存储到MongoDB的raw数据库中，以K-V对文档的形式存储起来，下面，我们将要对存储到MongoDB中，经过整理后的数据进行分析。

宏观分析

宏观分析是最基础也是最简单的，例如：

1. 我们可以统计一天24Hour，那个小时用户的活跃量最多；
2. 我们可以根据用户的IP来判断哪个区域的用户最多；
3. 我们可以根据使用设备，来判断使用什么终端的用户最多；
4. 同样，我们也可以用服务器的响应时间来判断服务器的运转情况。

宏观分析，在用Spark进行编程的时候，首先经过map过程，转换成我们想要的形式，例如：我们要统计24小时，分时统计用户活跃量。
这样，我们经过map后，就可以形成这样的一个形式：

GitChat

```
//我们假设,rdd的存储格式是一个Document,Document是MongoDB driver的存储格式，它实现了Map接口。

val rdd = MongoSpark.load(...)
//从MongoDB中直接加载某个table，也就是说，rdd的类型是 RDD[Document].这里用到的是scala编程，与Java类似
val count =
rdd.map(x=>{
    (parse2Hour(x.getString("time")),1)
}).reduceByKey(_+_)
//得到了分时统计结果，与写wordcount是类似的。
//parse2Hour()是一个函数，实现了将存储的UTC 格式的time提取出小时，这个其实自己实现一个简单的文本分割就搞定了。
count.foreach(println)
//打印出统计的结果
```

微观分析

所谓微观分析，就是粒度更细致的分析了。

我们在上面只是分析出所有的用户群体，在那个时间段更加活跃。
现在，我们再看另外一个例子：

我们想要分析uid为1000的用户，在一天24小时中，哪个小时活动最频繁。统计出来的结果，可以直接用做给他推送消息的推送时间点来使用。

其实，这个编程与上面的宏观统计类似，只不过，我们要将所有的rdd进行一个group分组，把所有uid相同的全都放到一起去。

之后，再在这个子rdd中分析该用户在哪个时间段最活跃即可。

示例代码如下：

```
val rdd = MongoSpark.load(...)
//从MongoDB中直接加载某个table
val user = rdd.groupBy(_.getString("uid"))
//通过用户的uid不同，来划分为不同的子rdd
val count = user.map(x=>{
    //每个划分出来的子rdd的格式是这样的：
    // ("uid",[Document1,Document2,...])
    /*
    我们可以看出来，划分出来的结果实际上是一个元组，元组的第一个元素就是我们划分的依据，元组的第二个元素就是一个List,这个List把所有属于这个元组的Document都包括进去了。
    */
    //后面，我们再对这个List进行一个暴力扫描，扫描出其中我们想要的结果就ok了,这里根据业务不同，代码省略，如果不会分布式并行编程，就给collect()到本地，编写相关的业务代码也Ok.
    ...
    //最后返回结果：
    (uid,某个小时)
})
```

GitChat

机器学习

其实，在我们实践当中，最常用到的机器学习算法恐怕就是聚类算法了。

聚类是一种无监督学习，我们最常用到的聚类算法就是kmeans算法，Spark的MLlib库为我们实现了kmeans算法，我们直接调用就OK了。

通过聚类算法，我们可以实现：

因为我们在日志中是包含用户的行为特征的，根据这些行为特征，我们可以通过聚类算法来实现用户的分群。

这里简单介绍下kmeans算法的原理：

> kmeans算法需要指定参数 k ，用来告诉算法需要分成几个类别；
> 然后将事先输入的n个数据对象划分为 k个聚类以便使得所获得的聚类满足以下条件：
> 同一聚类中的对象相似度较高；

> 不同聚类中的对象相似度较小。
> 聚类相似度是利用各聚类中对象的均值所获得一个“中心对象”来进行计算的。
> 聚类算法是一种迭代算法，通过反复迭代，来使得结果趋向于最优。这个迭代次数也是可以指定的，不过也不是越多越好，因为越往后改变就越小，效果不理想，反而浪费时间，这个需要具体去调试。

那么，我们在进行聚类的时候，我们可以统计某个用户，我们就叫他小明吧，下面我举个例子，假设下面的数据都是针对小明童鞋行为产生的日志情况，进行统计分析的结果：

小明的基本用户信息：

{
“name”:”小明”，
“age”:”18”,
“gender”:”male”,
“country”:”china”,
…
}

日志统计信息：

GitChat

{
“发送聊天记录”:250,
“陌生人聊天”:200,
“好友聊天”:25,
“群组聊天”:25,
“给别人照片点赞”:100,
“浏览别人发的说说”：100，
“给别人说说点赞”，52,
“搜索附近的人”,100,
“勾搭过几个陌生人”:50,
“阅读推荐文章”:0,
…
}

当然了，上面的日志统计结果我只是举个例子，我们可以选择其中的某几个具有代表性的作为特征向量，根据这些特征向量来对用户进行聚类。

譬如，我们可以选择：

聊天记录，陌生人聊天比例，搜索陌生人次数，勾搭过几个陌生人等等
来衡量某些人对陌生人交友的喜好程度。

这里，顺便说一下归一化的问题。

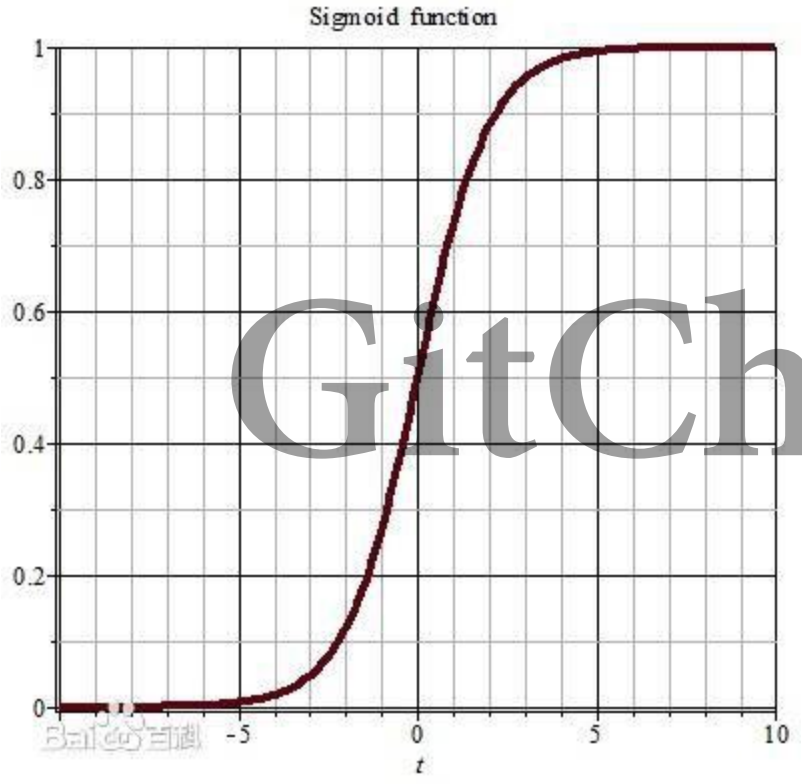
归一化

在上面的例子中，我们可以看到，如果某个人搜索附近的人频次特别高，而且只有这个人的水平特别高，可能达到了100000000这个量级，而除他之外的所有人可能都是200一下的量级。

这样，在进行数据计算的时候，直接用100000000这个数字带进去算很容易对结果造成干扰，甚至数字还有溢出的可能。

我们想办法，将这些数字映射到[0,1]的区间中，用小数来表示，这样，我们叫做归一化。

比较简单的归一化可以是用某个用户的值除以全体的总数；也可以是，用某个用户的值处理这个群体中最大的那个值；这样都可以保证结果是在[0，1]之间的，当然了，对于某些特别“奇葩”的用户，我们也可以用sigmoid函数来进行映射，sigmoid函数是一种S型曲线函数，他的图像是:



GitChat

这个当做了解就行了，实际上在一些分工明确的公司里，会有专门的算法组来进行优化和设计的。

不用参考wiki百科了，这个百度百科虽然Low一点，但是说得已经够用了，详见这里。

通过Kmeans算法，我们可以对用户进行聚类，相同类型的人，会被聚类到一起，可以供我们进行统计分析，科学决策和相似用户推荐等等。

推荐系统

诸如涉及到评分相关内容的，都可以用作推荐系统。推荐系统，只要保证能够维护好这几个数据表就可以做了:

用户信息表，产品信息表，用户对产品的评分表。

现在在工业界最常用的推荐系统算法是协同过滤相关算法，Spark 的MLlib库为我们实现了推荐系统的算法。

算法比较常用的一个是基于产品信息的（ ItemCF ），一个是基于用户的（ UserCF ），这里有一篇博文，介绍了上面两种算法。

在实际应用场景中，可能并非具有具体评分值，那么就需要我们根据用户的具体行为来为其指定具体的分数，譬如一张图片，衡量用户对其的喜欢程度：

浏览图片算作1，评论算作2（举个例子，这个有歧义，也可能是差评），点击大图观看算作3，点赞算作4，分享算作5，等等。

任务调度系统

大数据的任务调度系统主要有hadoop 的oozie，不过相对而言，笔者更喜欢用领英开源的任务调度系统——azkaban，azkaban的官方简介是：

> azkaban was implemented at LinkedIn to solve the problem of Hadoop job dependencies.We had jobs that needed to run in order, from ETL jobs to data analytics products.
>
> Initially a single server solution, with the increased number of Hadoop users over the years, Azkaban has evolved to be a more robust solution.

GitChat

可以看到，领英官方就用它来做大数据相关的任务调度使用，这里推荐一篇博文，详细介绍了 azkaban 用作大数据领域任务调度系统的配置和应用方法。

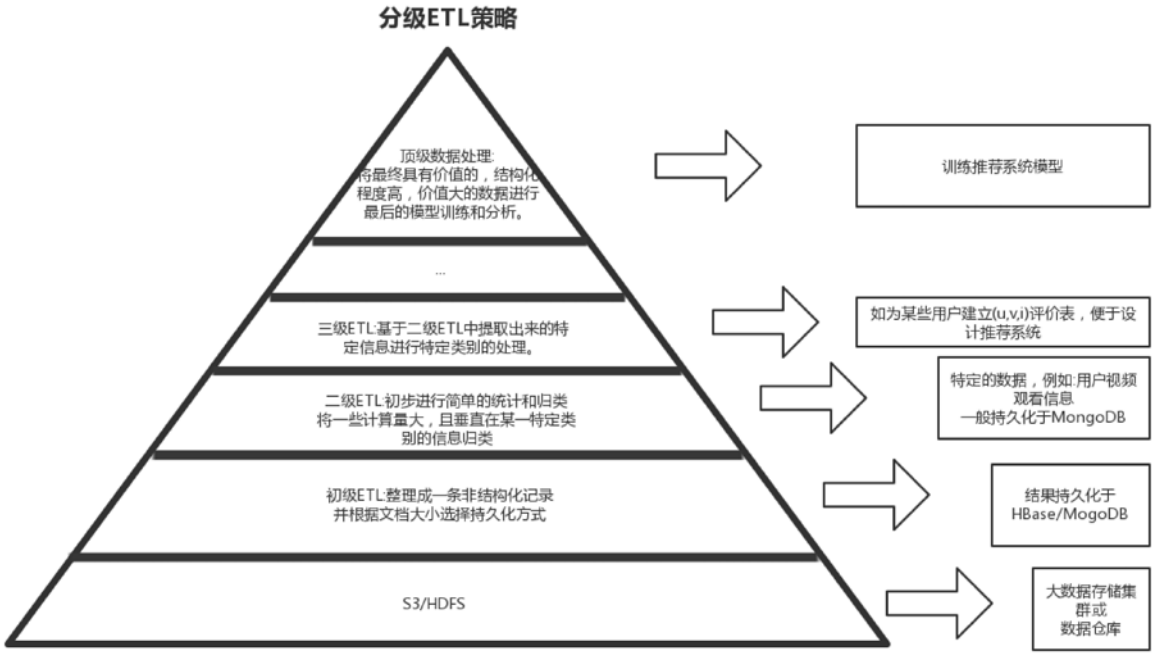
通过azkaban就可以做到解放人力：任务的自动调用和执行，而且可以指定调用顺序，定时触发还有报错功能，的确是件神器。

经验之谈

合理架构

在考虑实现大数据平台的时候，要对需要实现的产品做一个全方位的衡量，选择适合自己业务需要的方式针对性地架构，不应直接从网上copy一种方案便开始实施。

举一个例子，某种场合下，我们可以提出多级ETL的方式，来实现数据的复用，这些数据之间的关系呈现出金字塔状，如图所示：



越在金字塔上部分的数据量越小，经过ETL也变得更加细粒度，这部分数据的冗余部分相对较少，越在下面的数据冗余越大，越是冷数据。假设这样不同层的数据，我们可以对其进行复用，那么我们就有必要进行多级的ETL，如果这种复用情况很没有必要，我们也没有必要进行多级的ETL.具体是否适合我们的应用场景，要依据我们具体的业务情况来进行分析，不能按图索骥。

GitChat

保证任务调度顺序

任务调度系统我们使用Azkaban而不使用croncat（Linux自带的工具），是因为azkaban可以让我们自行指定任务之间的依赖关系。这些依赖是一个DAG，我们在azkaban中配置任务之间顺序的时候，一定要把握好任务之间的关系，当涉及到并行事务的时候，要考虑到二者之间的执行顺序和耦合关系，否则将会造成任务的失败。

保证集群的高负载

一个计算集群不能浪费掉，集群的价格比较昂贵，我们往往都是使用的云服务。对于不是按量付费的云服务，我们要保证集群的高负载。也就是让集群始终处于一种工作状态，不要将集群空着，这样比较浪费资源。对于流式数据处理来讲，集群自然是保证一直在工作。但是，对于离线计算来讲，可能当我们提交完一个作业之后，很快任务就执行结束，如果确定没有什么额外的计算任务，请选择按量付费，这样能节约很大一笔开销。

对于很多云服务商来讲，他们往往提供了MapReduce的云服务，在有条件的情况下，也可以购买这种云服务，避免配置的繁琐，也能够合理地按量付费。

充分挖掘节点算力

spark的默认设置，每个节点都有内存使用上的限制，我们可以通过修改conf目录中的配置文件，来修改spark使用的内存量。譬如spark-env.sh文件中的参数 `SPARK_WORKER_MEMORY` 可以设置工作节点的内存使用，这个使用值尽可能设的大一些，可以提高集群性能。

考虑批处理调用HTTP API

由于spark是一种并行编程思想，在某些调用上是并行地取执行。例如我们通过HTTP微服务的方式，查询一个用户的性别：

> http://foo.com/getGender/10001

每一个并行的执行操作都会去调用一次HTTP请求，来查询某个用户的性别。实际上，对于查询这种操作，远程的服务器是通过扫描数据库中的内容来完成的，多次反复扫描和一次批量地扫描效率相比是要差很多的。以MongoDB为例，执行两次findOne()和执行一次findMany()相比，开销可能要达到1.8倍左右，这还不算远程服务器响应并发时的性能消耗。对于这些操作，可以合并执行，将HTTP API改成：

> http://foo.com/getGender/100001,100002,1111,112333

GitChat

降低耦合

通过分析日志中的URL请求来完成大数据分析，避免修改现有的代码，可以实现大数据平台与现有平台之间的分离，实现松耦合。大数据平台的数据源来源于日志文件，避免对现有的业务代码侵犯，可以对现有数据采用读取的方式丰富数据来源，但是，尽量不要取修改业务系统中的数据。这样，把大数据平台作为一个单独的系统来实现，可以避免修改现有的业务系统。

总结

在本次的Chat中，我们谈到了中小型企业基于大数据技术的项目实践，其实，对于中小型企业来讲，可能数据量并没有大型公司相向得那么多，一般一天产生的日志条数几千万到一亿的居多，对于这种离线计算场景，其实并不一定就非得用分布式集群去消费数据，如果公司尚有闲置的单节点内存容量达到16G，双核心及以上的一台机器，实际上在做离线计算的时候，也够用了。

囿于时间仓促，笔者水平有限，如有疏漏在所难免，敬请不吝指教。
同时，有关本次文本没有涉及到的内容，或者不懂之处，我们将在后续的交流中展开讨论。具体交流形式是微信群交流，错过群讨论的用户也可以下载交流记录文档，或者添

加笔者的微信号**wotchin**，或者在笔者的微信公众号（cn92geek）后台留言，咨询本此chat的相关内容或者进行业务和技术上的探讨。

GitChat