干货 | 携程机票大数据架构最佳实践

2017-08-29 许鹏 Hadoop技术博文

本文转载自 **携程技术中心（ctriptech）** 公众号，本文PPT请点击下面 **阅读原文** 获取

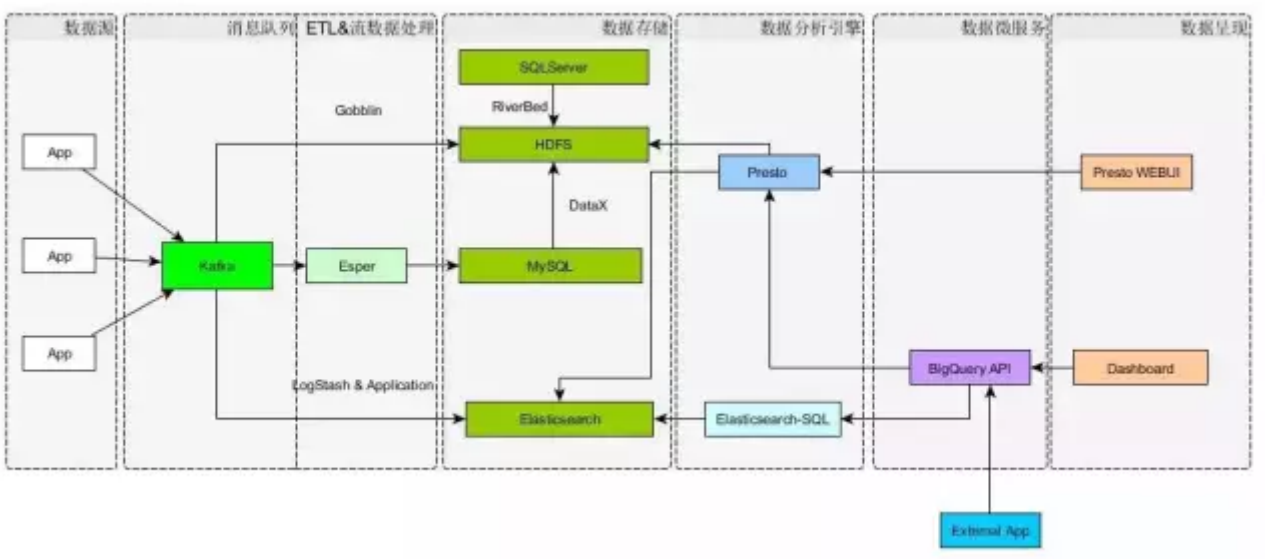
作者简介

许鹏，携程机票大数据基础平台Leader，负责平台的构建和运维。深度掌握各种大数据开源产品，如Spark、Presto及Elasticsearch。著有《Apache Spark源码剖析》一书。本文来自许鹏在〖DAMS 2017中国数据资产管理峰会〗上的分享，首发DBAplus社群（ID：dbaplus）。

现如今大数据一块有很多的开源项目，因此首先搭建平台的难点其实在于如何选择一个合适的技术来做整个平台的架构，第二，因为有业务数据，用了平台之后的话，如何用平台把数据分析出来让用户有很好的交互性的体验。第三个层面就是理工科喜欢建模，而在这整个过程当中，我们会形成一种非数据建模，而主要是我们如何分不同层面的人员搭配，进而做成这样一个大数据团队。

一、数据平台技术选型

1、整体框架



这个框架应该是一种大路货，或者更认为是一种比较常见的架构。前面也就是从数据源到消息队列到数据的清理、数据呈现等这些大家容易想到的东西，而在这样一个大帽子下面，所不一样的东西是具体选用什么样的组件来填这个空，在不同的场景下，每个人的选择是不大相同的。

像消息队列这一层，我们选用了Kafka，这是目前大家普遍用到的，因为它有高吞吐量，采用Push和Pull结合的方式，消费端主动拉取数据。ETL这块，目前大家都希望采用一种可以自定义的方式，一般来说比较流行的是用LinkedIn提供的Camus来做从Kafka到HDFS的数据同步。这应该是一种较为流行的架构。

那么放到HDFS上面的数据，基本上是为了批处理做准备的，那么在批处理分析的时候，我们选择一个什么样的分析引擎，可能就是一个值得争议的焦点，也就是说，也许在这个分析引擎的下面，有Hive，有Spark，有Presto，有Impala，还有其它的东西。 在这些引擎当中的选择或者实践，需要结合具体使用场景。

下面讲讲为什么会选择Presto而不是其它。假设在座的各位有Presto使用经验的话，会发觉Presto它是一个CLI的用户界面，并没有好的一种Web UI，对一般用户来说，CLI的使用会有难度，不管这是感觉上的还是实际上的，所以需要有个好的Web UI来增加易用性。

当前在GitHub上面能找到的Presto webui的就是Airbnb提供的AirPal，但根据我们的使用经验，不怎么友好，特别在UTC的时间设置上，同时它的社区维护已停滞在两年前，这一块我们做了适配，然后用Presto的StatementClient做了Web UI。前端采用的是jquery的easyui, 像刚才讲的批处理这一条线，就是用在了批处理这一块上。下面这一条线就是说有些数据可能是希望立马存储，立即被搜索到，或者做简要的分析。

作为搜索引擎，社区这一块，大家耳熟能详的应该是Elasticsearch，Elasticsearch的社区非常活跃，而且它的推广速度，应用型上面易都很好。但是Elasticsearch的难点在于如何对它进行好的维护，后面我会讲到它可能存在的维护痛点。

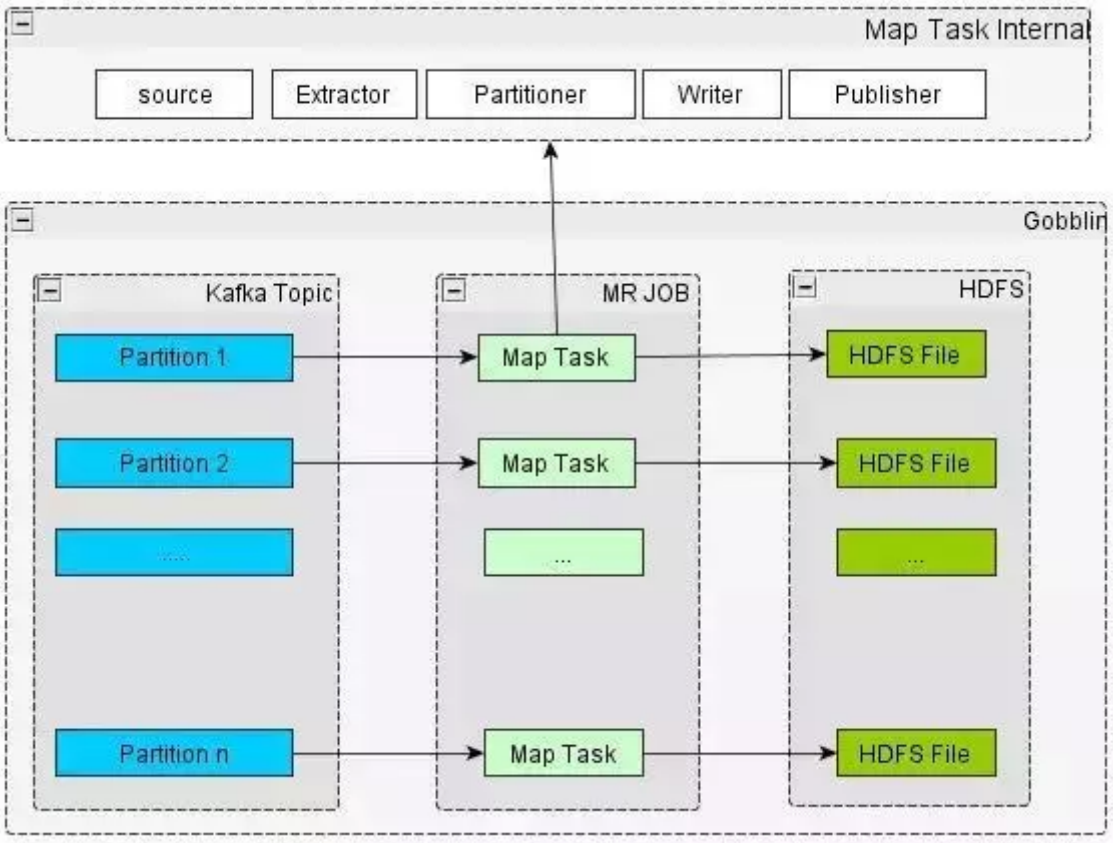
那么，Elasticsearch有非常强大的搜索能力，响应时间也是非常快的，但是它的用户接口，有自己的一套基于Lucene的搜索语法，当然Lucene的这一套语法本身是非常极客的，很简洁，但是一般的人不愿意去学这个东西，因为对于分析师来讲去学，就意味着以前的武功，几十年功夫白费了。

于是我们就采用了一个插件Elastisearch-SQL，这样就可以采用SQL语句对Elasticsearch进行点查询或者范围查询。而且在Elasticsearch的演进路径当中，也会支持SQL，按照之前看到的ES roadmap, 应该在17年最迟不超过18年发布6.×，重要的特性之一是对SQL的支持，大家可以看到如果不支持SQL，就等于是自废武功，或者拒客户于千里之外。

WebUI是人机交互的部分，我们会进行Ad-hoc查询，但在整个部门当中有不少程序希望调用查询，也就是应用的接口，采用SOA的架构，我们自己开发实现了 BigQuery API，可以通过这种调Restful 接口方式，进行取数或者分析。那么我们会自动判别到底是到ES这一侧还是到Presto进行取数。

在很多公司的使用当中，数据分析这一块是需要报表的，就是要有很好的Dashboard。

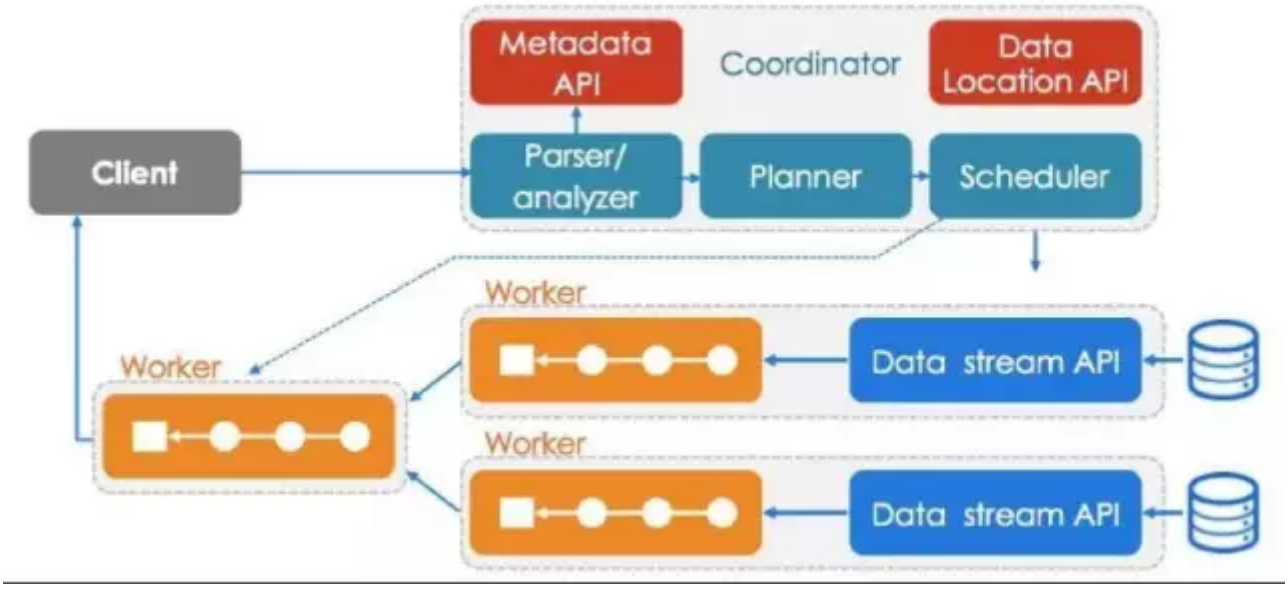
2、ETLPipeLine -- Gobblin



这个是ETL相对比较细节的一些东西。快速过一下这个图。在ETL的时间当中，比如说为什么不直接用像Spark或者流的方式，最常见的问题就是小文件的问题，到时候需要清理合并小文件，这很麻烦。如果采用Zeus去调度，然后设定一定数目的Partition，就有一个Map Task对应，尽可能的写满一个Block，以64M或者128M为主。在存储的时候我们除了考虑它的大小之外，存储格式的选择也应该是必须考量的范围。

从我们当前的选择来看，建议使用ORC这样的文件格式，采用这个文件格式是由于它已经内嵌了一定级别的索引，尽管索引不是非常细粒度，但是在某些层面是能够急速地提高检索，跳过不符合条件的数据块，避免不必要的数据传输。目前相对比较有希望的，或者大力推广的一个格式就是华为公司在推的CarbonData，它含有的索引粒度，索引信息比ORC更加细致。他们目前也出了1.×的版本，是相对来讲较为成熟一个版本。

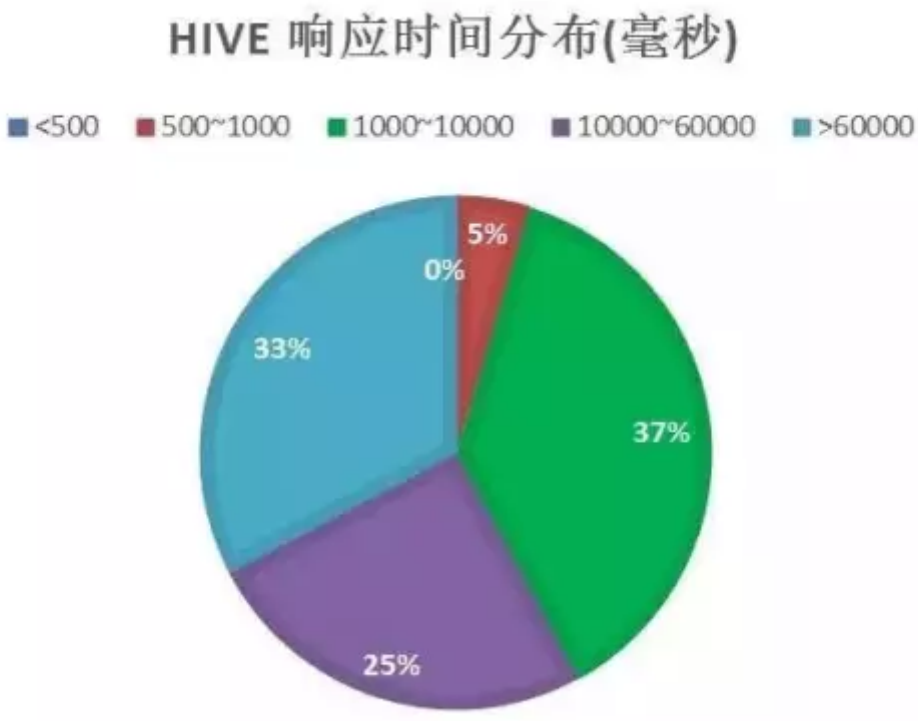
3、分析引擎 -Presto



这里讲的是Presto的内部机理。为什么不用Hive和Spark？Hive相当于是俄国的武器，特点就是傻大黑粗，绝对的稳定，稳定到什么程度？稳定到就是它是最慢的一个，有一个笑话就是我的成绩一直很稳定，因为老考倒数第一，没人可以比过，所以一直很稳定，而正数第一不见得很稳定，Hive就是这个特点，绝对可以出来结果，但是会让你觉得人生没有指望。

Spark的特点就是它名头绝对的够响，但是会发觉Spark具体的使用过程当中有些问题？资源共享是一个问题，如果说你光用Spark，肯定Concurrent Query出现问题的，要前置一个东西，比如Livy或者什么东西来解决掉你的资源共享问题。而且Spark的雄心很大，几乎想把所有东西都吃下去，所有东西都吃，就很难，因为你要涉及很多的领域。

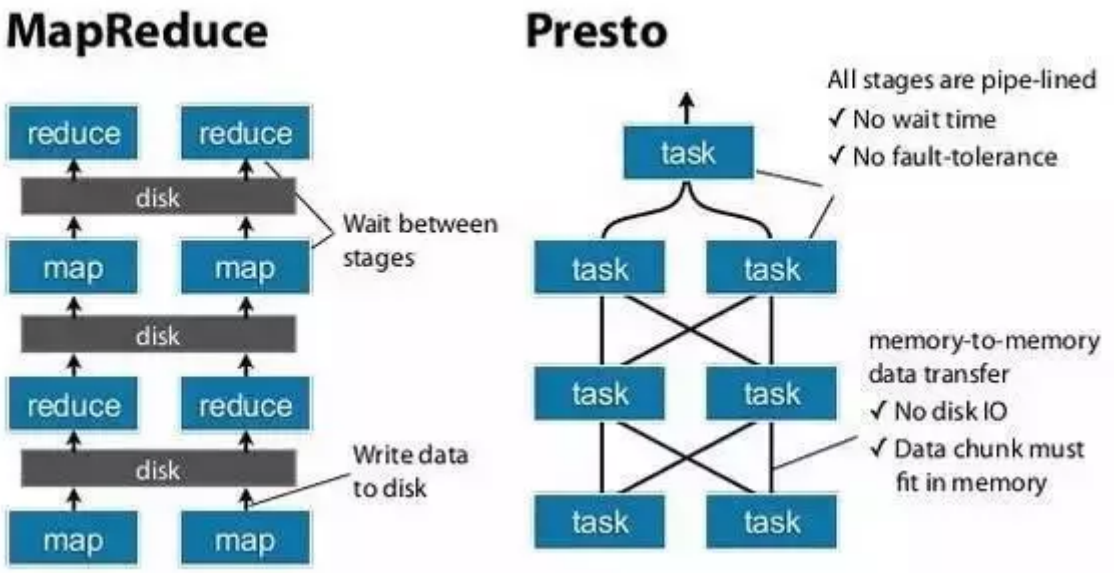
Presto只专注于数据的分析，只关注SQL查询层面，只做一件事，这个充分体现了Unix的哲学，遵循只干一件活，不同的活通过Pipeline的方式串起来。而且Presto是基于流水线的，只要有一个块当中结果出来了，然后比如说我们最典型的就是后面加一个后置的条件，然后limit 10或者Limit 1，你会发觉很快出来结果，用Spark会发现它Where条件的搜索会经历多个Stage，必须到前面的Stage都完成了才可以跑下一个Stage， 那个Limit 1的结果要到后面才过滤。



从Presto后面给出的这些数据可以看到，这种层面上的一个提升。基于ORC的文件存储，它的提升应该是5倍或者10倍，10倍到20倍的提升。它的架构简单来说是有一个Client，然后这个Client提交SQL语句过来，前面有一个Planner和Scheduler，会把相应的SQL的东西分层，分成不同的Stage，每一个Stage有多个Task，这些真正的Task是运行在不同的Workers上面，利用这些Workers去数据源读取数据。

也就是说Presto是专注于在数据分析这侧，具体数据的存储在外面，这个当中肯定要去解决哪些东西是值得去拉取的，有哪些东西可以直接推到数据源侧去搞定，不需要傻乎乎地把很多东西拉上来。

分析引擎比较——Presto与MapReduce



大家可以看到我刚才提到一个基于Stage的方式，一个基于Pipeline的方式，Pipeline的方式就是整个过程中，处理没有停顿，整个是交叉的，它不会等上一个Stage完成后再进行下一个Stage，Spark的特点就是等到一个Stage结束了，数据吐到Disk中，下一个Stage再去拉数据，然后再进行下一个。Pipeline就是说我有一个Task处理完，直接将数据吐到下一个Task，直到Aggregator节点。

那么在这个过程当中，你也会看到Presto的一个最大特点就在于所有的计算就在内存当中，你会想到人的大脑，机器的内存都是有限的，会崩掉了，崩掉就崩掉了，早死早超生，大不了再跑一趟，这就是Presto的一个基本原则。

MapReduce会重启，如果成功了还好，重启很多次崩掉是不是三观尽毁？通过这种特点也表明Presto适用的场景，适用于交互式查询，如果是批量的，你晚上要做那种定期报表的话，把整个交给Presto是不负责任的表现，因为有大量的时间，应该给Hive比较好。

4、近实时搜索 –Elasticsearch

下面讲讲ES层面的东西，也就是近实时的搜索引擎，它所有的东西都是基于Lucene上面进行一个包裹，对JSON支持的非常好。同时Elasticsearch支持横向、水平扩展，高可用，易于管理，社区很活跃，背后有专门的商业公司。它的竞品就是Solr，Solr的Cloud，SolrCloud安装较为复杂，引入了独立的第三方东西，对ZooKeeper集群有极大的依赖，这样使得Solr Cloud的管理变得复杂。

SolrCloud的发展也很活跃，现在是到了6.×，后续就是到7.×，而且SolrCloud的6.×当中引入了对SQL的支持，ES和SolrCloud是同门师兄弟，通过同门师兄弟的相互竞争可以看到发展的趋势——SQL一定是会支持的。

如果大家做搜索这一块东西的话，上面这张图其实是很常见的，它肯定会在某一个节点上面有相应的一个主分区，有一个Primary partition，而在另外一个节点上面它有一个Replicas，而且Replica可能不只一个，如果这些没有，这张图就没有太多好讲的。问题是该分几个Replica，在每台机器上分几个不同的partition，如果在从事维护工作的话，上述问题是值得去分析和考究的。

ES调优和运维

OS Level
- File Handler
  - ulimit
- Memory
  - turn off swappiness
- I/O Scheduler
  - CFQ
  - vm.dirty_ratio
  - vm.dirty_background_ratio

Elasticsearch
- Cluster Level
  - Shard allocation
  - Concurrent recovery
- Index Level
  - Shard
  - Replica
  - Refresh
  - Merge
  - Schema Design
  - Balance indexing and query

下面讲ES的调优和运维，从两个层面出发。

第一个层面就是OS， 讲到Linux， 调优过程中自然会考虑到它的文件句柄数，然后它的Memory，它的I/O的调度，I/O的调度线如果在座各位对内核比较感兴趣的话，你会发现基本使用CFQ，因为在生产环节上大多会采用Redhat或者CentOS来部署，不会部署到像自己玩的Archlinux或者Gentoo上面，不可能这样做的。

还有就是它的Virtual memory DirtyRatio，这个东西是会极大地影响响应时间，或者说有时你会发觉I/O操作，而且CPU一直比较高，因为有文件缓存，缓存足够多的话就一直往磁盘去写，所以我们的办法就是把原来设置比较高的vm.dirty_ratio，由默认20%调小到10%。意思就是说缓存内

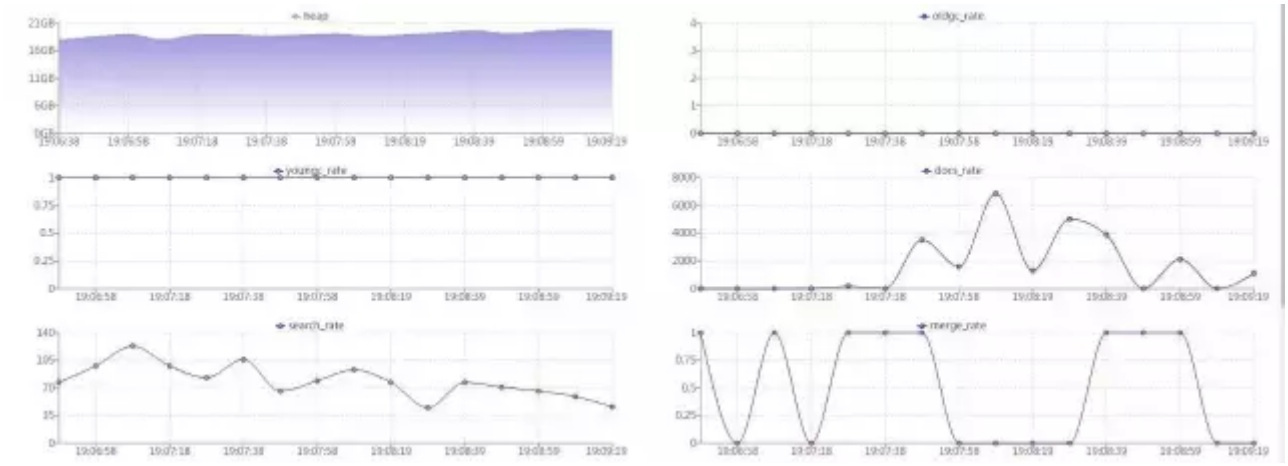
容一旦超过系统内存的10%其它活不要干了，专心致志吐这个缓存内容。 Vm.dirty_background_ratio是说如果达到这个阀值，就开始将文件缓存内容写入到磁盘。OS层面的调优和数据库的系统调优有相似性。

另一个层面的调优是ES本身，首先就是说我在一个Cluster上，Shard的数目要均匀分布。

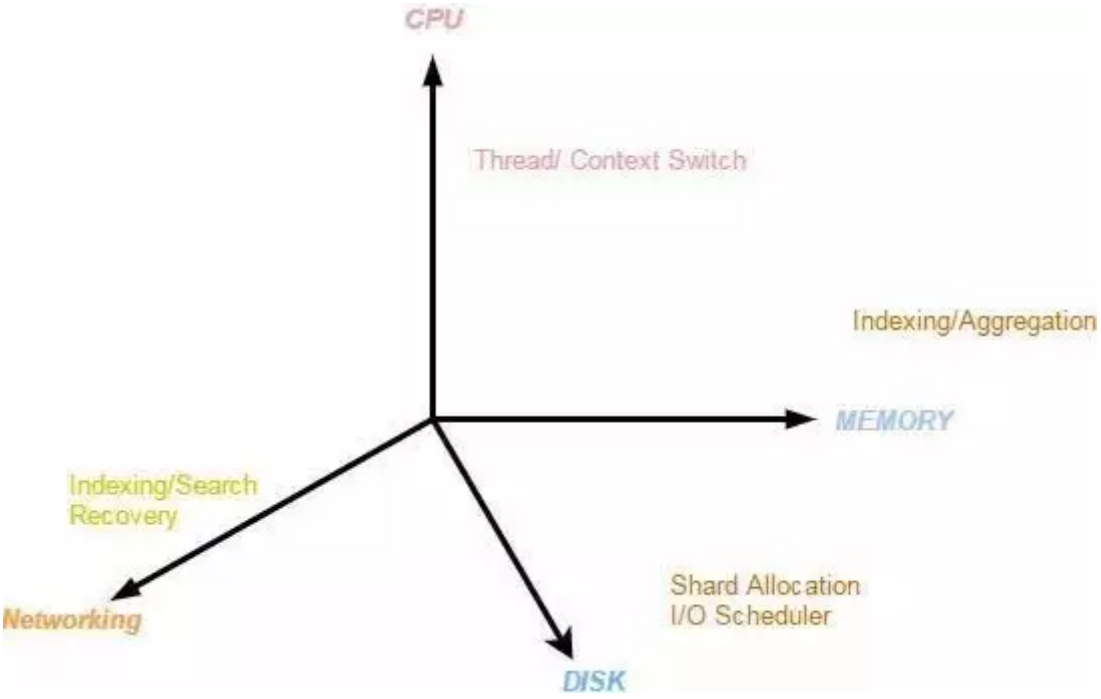
我这里放了一张截图，这个截图大家可以看到所有的节点上面，Shard数目上来讲是非常均匀的。有相应的参数调整可以达到这样的效果。第二个就是会有一个Replica的过程，比如新加一台机器或者说我是减少一台机器，要做相应的维护，机器的集群会做动态的扩容和缩减。那么这时如果都来做Shard的转移，整个集群的写入和查询会受很大影响，所以做一定的均衡，两者之间要有一定的Balance。这些讲的都是集群级别，下面讲索引级别的优化。

索引级别的优化就是我要对Shard的数目，到底是这个Index是分十个Shard存还是5个来存，refresh的频率，Refresh就是说这个数据写入多久之后可以被搜索到。Refresh时间拉得越长，数据吞吐量越大，但是可以被搜索到的时间越滞后。还有Merge的过程，因为分片，为了减少对文件句柄使用， 所以需要进行Merge。有人讲就是因为ES支持Schemaless了，所以不需要fixed的Schema。但在实际的使用过程中发觉，如果不做一定限制的话，每个人都认为是自由的，就会出现一个Field的急速膨胀，在某个索引下面成千上万的字段， 这样一来索引的写入速度就下来了。

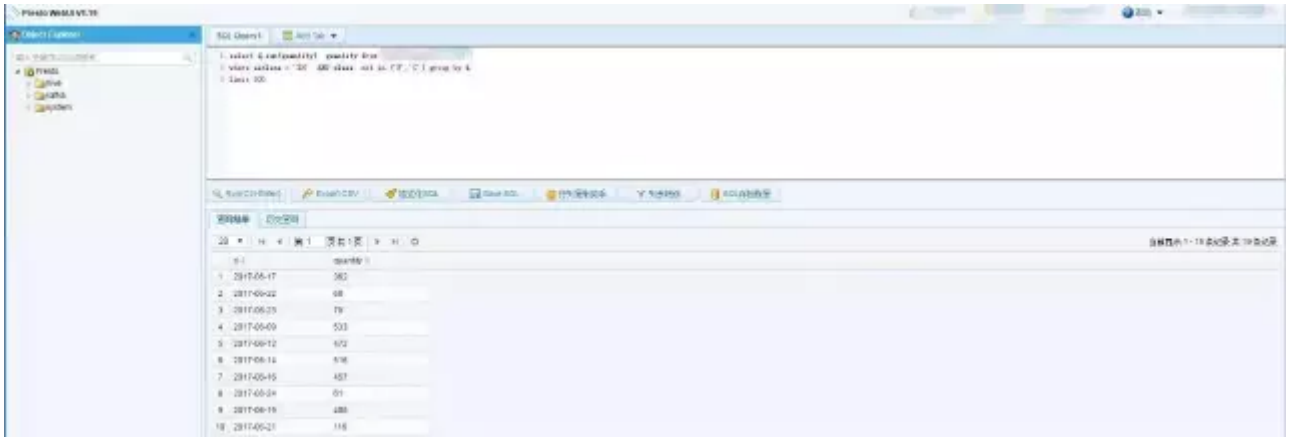
下图是我们自己写的Dashboard，说到ES，可能在座的也有不少在用，如果说你们升级到5.×后发现一点，1.×比较好的插件Marvel，5.×里面就没有，提供的就是X-pack，X-pack是要收钱的，那么它同时提供了一个所谓的basic版本，Free的东西大家都知道，便宜无好货，就是说它的功能是对比了1.×的版本，很多信息都是没有的。



我们的话就是自力更生，因为你所有的内容都是可以通过Rest API读取到，只不过是需要在前端可视化一下。那么这张图就是我们做的工作，可以很方便地看到当前节点的写入量、查询量，当前节点的索引，Shard数目还有当前集群的状态，如果一旦状态变为red，可以邮件通知，在页面上还可以进一步点下去了解每一个节点和索引的详细信息。

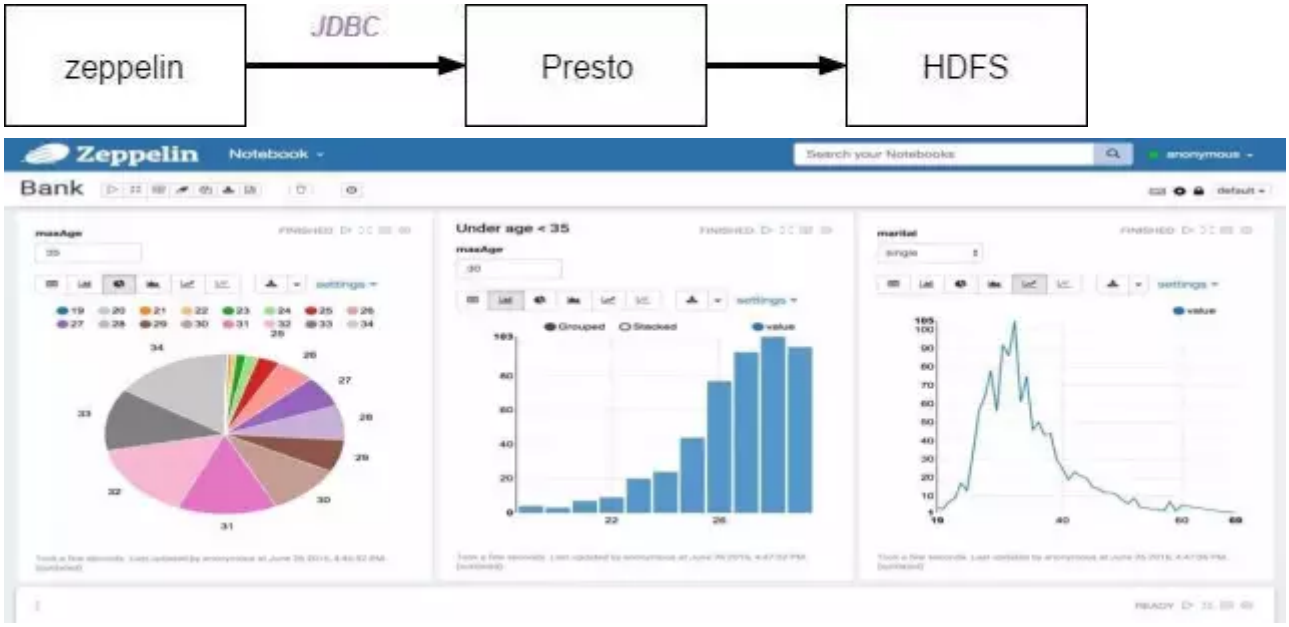


稍微总结一下，一般来说在调优上考量的不外乎四个维度：一个CPU的维度，一个Memory的角度，还有就是Disk的I/O角度，另外一个是网络。 比如从原来的百M网卡升级到千M网卡，从千M到万M，查询的响应速度会有很大提升。



这是前面提到我们统一的一个SQL查询的接口，大家可以看到这挺简陋的，很傻很天真的样子，我就是上面输入一个SQL，下面很快就出来一个结果。但就是因为采用了这种方式，因为后面是它采用了Presto这个引擎，在部门内部，我们有不少同事都在使用这个进行数据查询，目前的日常使用量应该是在近8K的样子，因为最近还升级了一下网卡，升级到万M网卡，使得速度更加快。多余的时间喝喝咖啡抽抽烟生活多美好，比等在那里焦虑有意思多了。

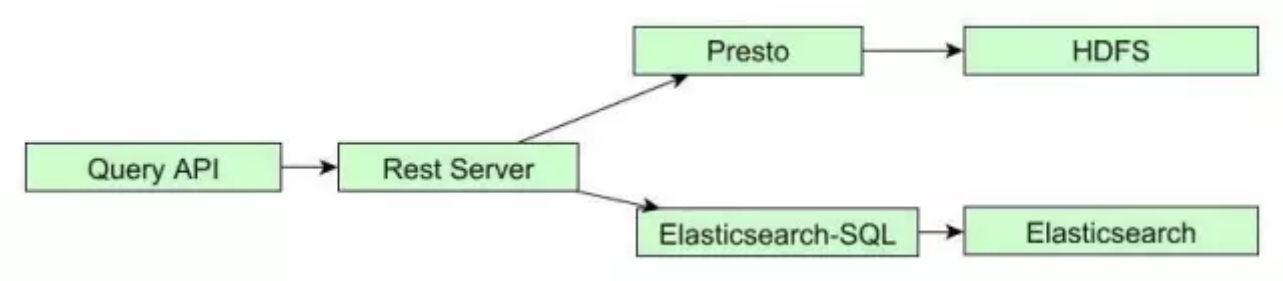
5、数据可视化——Zeppelin



在做数据可视化这一块时，可以借鉴竞争对手或者竞品，看看别人在做什么，如果说大家去看Hue， Hue的话，其实就是上面输入查询语句之后，后续就把结果很好地显示出来。我们目前所考虑的就是说如何做到Data visualize的，目前尝试用Zeppelin，这个可以通过JDBC接口对接Presto，把数据查询出来，通过简单的拖拽，直接把报表以图形化的方式展现出来。

补充一下，Zeppelin这个如果要对接Spark，如果只是一个Spark集群，直接对接这个Spark集群，资源利用率是非常非常低的，但是你在前置一个Livy Server的话，通过Livy来进行资源调度，资源共享会比较好。目前有两个这一方面的竞品，一个Livy，另外一个就是Oyala它提供的SparkJob ServerS，干的活其实都是一样。Zeppelin是对Livy Server做了整合。

6、数据微服务 –Rest查询接口

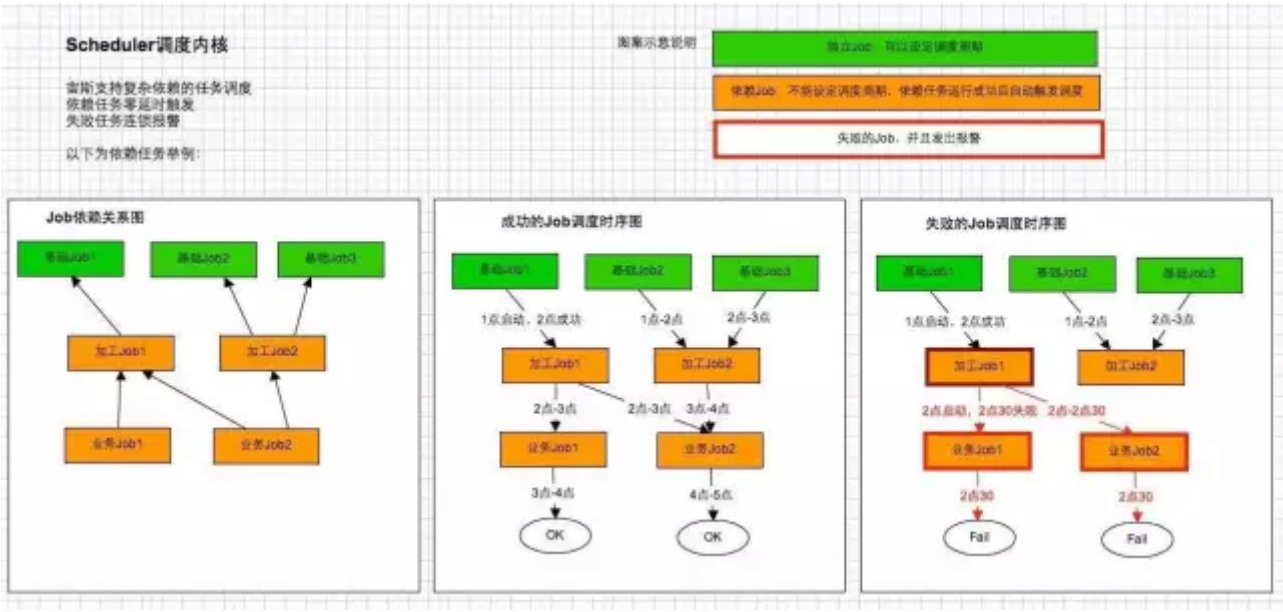


微服务这一块，我们提供了一个BigQuery API，这样的好处是有一个统一的查询入口，有统一的权限管理。因为查询时不是所有人都应该看到所有的数据，这很容易出问题，可能有比较实实在在的数据，它不像一般的日志数据，特别像机票或者我们这边的酒店，它的数据有不少的一些敏感信息，这需要做相应的权限管理。

这个入口统一之后，做权限管理就比较方便了，出问题的话只要查相应的日志就OK了。而且使用的是统一的查询语言，都用的是大家比较熟知的这种SQL语句，不是说用了一个新的东西就要学习一套新的知识，这样子的话原有知识不容易得到传承，这是大家都应尽量去避免的。

7、任务调度器 –Job Scheduler

- Zeus-https://github.com/ctripcorp/dataworks-zeus



其实在做一套大数据的平台时，少不了任务调度这一块。任务调度这一块我们使用的是Zeus系统，携程在这一块开源出来，由我们公司Ops的团队专门来负责开发和维护个平台。但是你想，通过这个平台递交的任务包括，ETL和定时任务，可以实现将数据从Kafka放入到HDFS或者是把SQL Server和MySQLDB里面的数据同步到HDFS。调度这一块目前市面上的竞品还有AirFlow和其它。

二、数据团队能力建设

	后台技术	前端技术和框架	Linux知识
引擎开发	√		
交互界面设计		√	
平台运维	√		√
平台规划	√	√	√
开发语言	Java/Scala/Python	Javascript, React.js	

这部分讲的是我们团队的建设。目前我把它分成五个不同的角度，第一个是引擎的开发，这一块是相对较难的，它对后台的技术要求比较高。

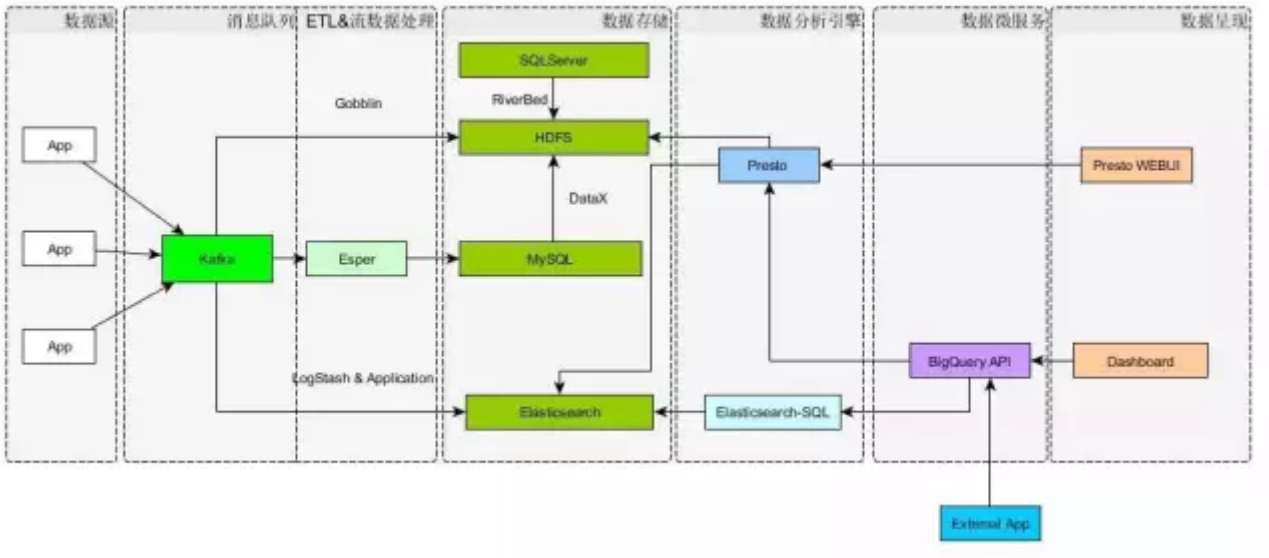
第二是交互界面设计，整个东西做出来，如果只是做了引擎，或者对引擎做了，但是没有实际的人用，老板肯定也会叫滚蛋的，肯定要一环套一环，形成有效的传动，不是单点，只讲发动机没有任何意义的，要讲整车。所以有引擎，引擎的要求也比较高，会有一个交互界面的设计，就是我如何用这些引擎的东西。

把这些东西都弄上后，可以转起来了，整个可以转起来之后，我们还有个运维，其实大家可以逐步发现一个趋势，就是无论大数据也好，云平台也好，对运维的要求都是比较高的，一个好的运维不仅要掌握一个基础的OS层面的东西，对后台也得有一个较好的概念或者好的研究。无论是从后台服务开发转到运维还是从运维转后台服务器开发，两者都需要去交叉学习。

那么，一个平台规划相对来说就是一个架构师或相对更高层一点人员的工作范畴，视野可以更高一点，这样的角色肩负了架构和产品经理这两个概念的东西，因为像这种东西最主要是内部使用，比较难以独立出来。

语言这一块就是见仁见智，我只是把我们现在采用到的，使用到的东西列了一下，有上述这么多。

大体我们的实践的就是这些。我们所有的部分应该就在这一张图里，这张图的内容看起来比较平淡，但是如果需要把这张图弄好，确实花了不少时间。


数据源
消息队列
ETL&流数据处理
数据存储
数据分析引擎
数据微服务
数据呈现
SQLServer
RiverBed
Gobblin
HDFS
App
App
App
Kafka
Esper
MySQL
DataX
Presto
Presto WEBUI
BigQuery API
Dashboard
LogStash & Application
Elasticsearch
Elasticsearch-SQL
External App

阅读原文