某电商的年销售数据分析

Document for Cloud Computing Project

云计算课程项目文档

CLOUD COMPUTING, AUTUMN 2017

BY

1552672 吴可菲
1552673 陈明曦

Tongji University
School of Software Engineering

Contents

1 项目简介

本项目模拟了某电商商家的年终销售数据分析，商品数的数量级为百万，订单数的数量级为亿，在这种情况下，传统单机式数据库已远远不能满足需求，适合开发和运行处理大规模数据的平台，如hadoop应运而生。本项目的重点在于要求实现hadoop高可用性，因此我们选择利用Docker容器中hadoop镜像搭建伪分布式集群，用zookeeper管理集群实现集群的高可用性，再用基于Hadoop的一个数据仓库工具hive，可以将结构化的数据文件映射为一张数据库表，最后通过JDBC进行的sql语句转换为MapReduce任务进行运行。

2 功能介绍

该项目面向商家，给商家查询分析过去一年的销售数据，包括销量排行、销量变动、价格变动、商品订单等信息；同时，商家可以通过获得的结果，选择下架（删除）商品、修改商品价格、名称等信息，也可以新增商品。

2.1 数据分析

- 展示数据库中所有商品数、类别数、订单数，直观感受数据规模。
- 展示每个类别每一季度销量柱状图，便于商家分析不同类别、不同季度之间的销售情况。
- 按年度销量从高到底，展示商品类别销量排行。

2.2 查询

2.2.1 查询类别信息

- 输入商品类别名称，得到该类别的商品的销量排行，包括商品名称、编号、年度销量；
- 选择日、月、季度，得到该类别2017年不同时间维度的销量走势图。

2.2.2 查询商品信息

- 输入商品编号查询，得到该商品的所有订单信息，包括订单编号、订单时间、订单价格、买家名称、买家地址、买家电话；
- 根据订单价格和订单时间，绘制一年内商品价格走势图；
- 选择日、月、季度，得到该商品2017年不同时间维度的销量走势图。

2.3 增删改

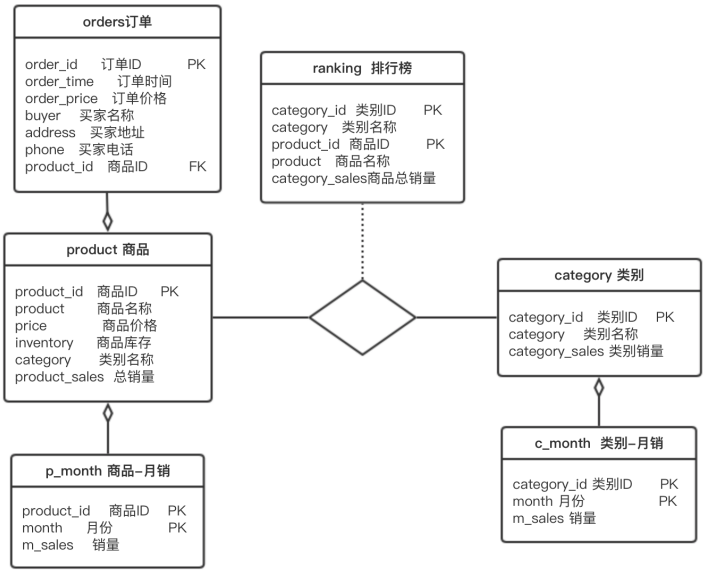
- 输入商品编号、名称、类别、价格、库存，可新增商品；
- 输入商品编号得到该条商品信息，可对商品名称、类别、价格、库存的信息进行修改；
- 输入商品编号得到该条商品信息，可删除该商品。

3 数据规模

- 总商品数：4，534，783
- 总类别数：55
- 总订单数：1，143，365，549

4 数据存储

4.1 表的设计



4.2 Hive的选择

Hive是分布式的关系型数据库，主要用来并行分布式处理大量数据。

- 数据存储位置：Hive建立在Hadoop之上的，所有的数据都是存储在HDFS中的。
- 索引：Hive在加载数据的过程中不会对数据进行任何处理，甚至不会对数据进行扫描，因此也没有对数据中的某些Key建立索引。Hive要访问数据中满足条件的特定值时，需要暴力扫描整个数据，因此访问延迟较高。由于 MapReduce 的引入， Hive 可以并行访问数据，因此即使没有索引，对于大数据量的访问，Hive 仍然可以体现出优势。
- 数据更新：由于Hive是针对数据仓库应用设计的，而数据仓库的内容是读多写少的。因此，Hive中不支持对数据的改写和添加，所有的数据都是在加载的时候中确定好的。
- 执行延迟：Hive中大多数查询的执行是通过 Hadoop 提供的 MapReduce 来实现的（类似 select * from tbl的查询不需要MapReduce）。由于要走MapReduce，即使一个只有1行1列的表，如果不是通过select * from table;方式来查询的，可能也需要8、9秒。另外一个导致 Hive 执行延迟高的因素是 MapReduce框架。由于MapReduce 本身具有较高的延迟，因此在利用MapReduce 执行Hive查询时，也会有较高的延迟。
- 可扩展性：由于Hive是建立在Hadoop之上的，因此Hive的可扩展性是和Hadoop的可扩展性是一致的。

综上所述，Hive是高延迟、结构化和面向分析的。但Hive比较擅长处理大量数据。当要处理的数据很多，并且Hadoop集群有足够的规模，才能体现出它的优势。

在我们的项目中，查询信息、刷新页面的响应都比较缓慢，倘若在查询数据时用到表的链接或count、order by等操作时，查询速度会慢到令人发指，因此我们选择以数据的冗余换取数据的存储时间，且在读取数据之后再对数据做处理，如ranking表实际是product表和category表的join，但是为了方便读取数据我们单独建了这张关系表，并且在后端处理时先调取数据，再对数据进行处理，如按销量排名、计算个数之后返回给前端展示。

4.3 Mysql的选择

由于Hive不支持对数据的改写和添加，因此在进行商品的增删改操作时，我们使用了传统关系型数据库MySQL，并使用Sqoop，依据Mysql表每条记录中的修改时间戳字段，进行增量导入。 由于我们的项目是一个面向商家的年终销售分析系统，因此对于数据的实时更新要求较低，因此我们通过oozie定时调用Sqoop从MySQL中增量导入数据至Hive表。

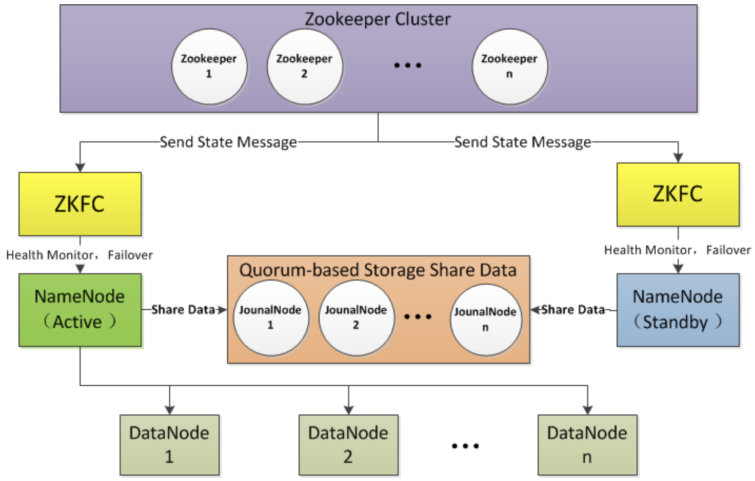
5 高可用性分析

5.1 高可用性的概念及作用

HA(High Available),高可用性群集，是保证业务连续性的有效解决方案，一般有两个或两个以上的节点，且分为活动节点及备用节点。通常把正在执行业务的称为活动节点，而作为活动节点的一个备份的则称为备用节点。当活动节点出现问题，导致正在运行的业务（任务）不能正常运行时，备用节点此时就会侦测到，并立即接续活动节点来执行业务。从而实现业务的不中断或短暂中断。

5.2 高可用性的方案

5.2.1 NameNode+HA架构



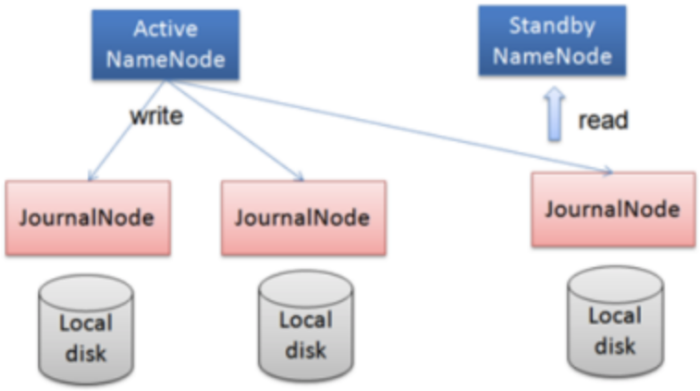
从架构图可以看出，使用Active NameNode，Standby NameNode 两个节点可以解决单点问题，两个节点通过JounalNode共享状态，通过ZKFC 选举Active ，监控状态，自动备份。

- Active NameNode:接受client的RPC请求并处理，同时写自己的Editlog和共享存储上的Editlog，接收DataNode的Block report, block location updates和heartbeat。
- Standby NameNode:同样会接到来自DataNode的Block report, block location updates和heartbeat，同时会从共享存储的Editlog上读取并执行这些log操作，保持自己NameNode中的元数据（Namespcae information + Block locations map）和Active NameNode中的元数据是同步的。所以说Standby模式的NameNode是一个热备（Hot Standby NameNode），一旦切换成Active模式，马上就可以提供NameNode服务。
- JounalNode:用于Active NameNode ， Standby NameNode同步数据，本身由一组JounnalNode节点组成，该组节点奇数个。
- ZKFC:监控NameNode进程，自动备份。

5.2.2 HA的工作原理

HA的基本原理就是用2N+1台JounalNode存储EditLog，每次写数据操作有大多数（¿=N+1）返回成功时即认为该次写成功，数据不会丢失了。这个原理是基于Paxos算法,所能容忍的是最多有N台机器挂掉，如果多于N台挂掉，这个算法就失效了。

为了保持standby NameNode实时的与主Active NameNode的元数据保持一致，他们之间交互通过一系列守护的轻量级进程JournalNode。任何修改操作在 Active NameNode上执行时，JounalNode进程同时也会记录修改log到至少半数以上的JounalNode中，这时 Standby NameNode 监测到JounalNode 里面的同步log发生变化了会读取 JounalNode 里面的修改log，然后同步到自己的的目录镜像树里面，如下图： 当发生故障时，Active的 NameNode 挂掉后，Standby



NameNode 会在它成为Active NameNode 前，读取所有的JounalNode里面的修改日志，这样就能高可靠的保证与挂掉的NameNode的目录镜像树一致，然后无缝的接替它的职责，维护来自客户端请求，从而达到一个高可用的目的。

5.3 Zookeeper的用处

5.3.1 Zookeeper的工作原理

ZooKeeper是一个开源的分布式协调服务。在ZooKeeper中，有三种角色：Leader、Follower、Observer。一个ZooKeeper集群同一时刻只会有一个Leader，其他都是Follower或Observer。ZooKeeper默认只有Leader和Follower两种角色，没有Observer角色。ZooKeeper集群的所有机器通过一个Leader选举过程来选定一台被称为『Leader』的机器，Leader服务器为客户端提供读和写服务。

5.3.2 Zookeeper在master选举的应用

ZooKeeper是一个高可用的分布式数据管理与协调框架。基于对ZAB算法的实现，该框架能够很好地保证分布式环境中数据的一致性。也是基于这样的特性，使得ZooKeeper成为了解决分布式一致性问题的利器。

因此，HDFS中Active NameNode的选举，即Master选举可以说是ZooKeeper最典型的应用场景了。

针对Master选举的需求，通常情况下，我们可以选择常见的关系型数据库中的主键特性来实现：希望成为Master的机器都向数据库中插入一条相同主键ID的记录，数据库会帮我们进行主键冲突检查，也就是说，只有一台机器能插入成功——那么，我们就认为向数据库中成功插入数据的客户端机器成为Master。

依靠关系型数据库的主键特性确实能够很好地保证在集群中选举出唯一的一个Master。但是，如果当前选举出的Master挂了，显然，关系型数据库无法通知我们这个事件，而ZooKeeper可以做到。

利用ZooKeepr的强一致性，能够很好地保证在分布式高并发情况下节点的创建一定能够保证全局唯一性，即ZooKeeper将会保证客户端无法创建一个已经存在的ZNode。也就是说，如果同时

有多个客户端请求创建同一个临时节点，那么最终一定只有一个客户端请求能够创建成功。利用这个特性，就能很容易地在分布式环境中进行Master选举了。

成功创建该节点的客户端所在的机器就成为了Master。同时，其他没有成功创建该节点的客户端，都会在该节点上注册一个子节点变更的Watcher，用于监控当前Master机器是否存活，一旦发现当前的Master挂了，那么其他客户端将会重新进行Master选举。这样就实现了Master的动态选举。

6 工作步骤

6.1 基于hadoop的HA集群搭建

- Zookeeper：3个
- JournalNode：3个
- NameNode：3个
- DataNode：3个

6.1.1 搭建hadoop分布式集群

- 先构建一个具备ssh功能的镜像
- 基于这个镜像再构建一个带有jdk的镜像
- 基于这个jdk镜像再构建一个带有hadoop的镜像
- 修改配置文件core-site.xml、 hdfs-site.xml、mapred-site.xml、 yarn-site.xml到$HADOOP_HOME/etc/hadoop目录下，制作成一个docker镜像，然后用同一个镜像运行多个容器
- 格式化namenode并启动hadoop伪分布式

6.1.2 配置hadoop的高可用

- 在分布式集群的基础上修改配置文件，增加三个JournalNode、一个NameNode节点
- 配置zookeeper
- hadoop的启动：先启动JournalNode后再格式化NameNode节点，此时启动NameNode节点，只在当前格式化后的节点上成功启动了NameNode进程，其余NameNode节点则启动失败。这时需要在启动NameNode的节点上执行命令完成格式化，再执行start-dfs.sh命令，成功启动其余NameNode节点。此时三个NameNode节点的状态都为standby。
- 启动zookeeper服务，设置其中一个NameNode为active状态

最终，下图为最终搭建完成的NameNode节点状态图。

Overview 'namenode3:9820' (active)

Namespace:	cluster
Namenode ID:	nn3
Started:	Thu Jan 11 17:43:46 +0800 2018
Version:	3.0.0, rc25427cecx461ee979d30edd7a4b0f50718e6533
Compiled:	Sat Dec 09 03:16:00 +0800 2017 by andrew from branch-3.0.0
Cluster ID:	CID-d2ab46f4-726a-40ce-b10e-75f259eb8545
Block Pool ID:	BP-1689186383-172.18.0.2-1515663800650

Summary

Security is off.

Safemode is off.

1 files and directories, 0 blocks = 1 total filesystem object(s).

Heap Memory used 112.42 MB of 319 MB Heap Memory. Max Heap Memory is 1.73 GB.

Non Heap Memory used 48.84 MB of 49.75 MB Commited Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	439.21 GB
DFS Used:	168 KB (0%)
Non DFS Used:	28.2 GB
DFS Remaining:	388.5 GB (88.45%)
Block Pool Used:	168 KB (0%)
DataNodes usages% (Min/Median/Max/stdDev):	0.00% / 0.00% / 0.00% / 0.00%
Live Nodes	7 (Decommissioned: 0, In Maintenance: 0)

Overview 'namenode1:9820' (standby)

Namespace:	cluster
Namenode ID:	nn1
Started:	Thu Jan 11 17:43:46 +0800 2018
Version:	3.0.0, rc25427cecx461ee979d30edd7a4b0f50718e6533
Compiled:	Sat Dec 09 03:16:00 +0800 2017 by andrew from branch-3.0.0
Cluster ID:	CID-d2ab46f4-726a-40ce-b10e-75f259eb8545
Block Pool ID:	BP-1689186383-172.18.0.2-1515663800650

Summary

Security is off.

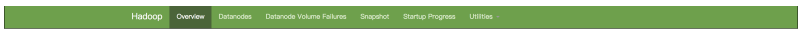
Safemode is off.

1 files and directories, 0 blocks = 1 total filesystem object(s).

Heap Memory used 162.62 MB of 254.5 MB Heap Memory. Max Heap Memory is 1.73 GB.

Non Heap Memory used 48.84 MB of 50.13 MB Commited Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	439.21 GB
DFS Used:	168 KB (0%)
Non DFS Used:	28.2 GB
DFS Remaining:	388.5 GB (88.45%)
Block Pool Used:	168 KB (0%)
DataNodes usages% (Min/Median/Max/stdDev):	0.00% / 0.00% / 0.00% / 0.00%
Live Nodes	7 (Decommissioned: 0, In Maintenance: 0)



Overview 'namenode2:9820' (standby)

Namespace:	cluster
Namenode ID:	nn2
Started:	Thu Jan 11 17:17:59 +0800 2018
Version:	3.0.0, rc25427cecx461ee979d30edd7a4b0f50718e6533
Compiled:	Sat Dec 09 03:16:00 +0800 2017 by andrew from branch-3.0.0
Cluster ID:	CID-cfcc069c-8a4b-47ae-a4f5-02f64f712331
Block Pool ID:	BP-1940538564-172.18.0.2-1515662253960

Summary

Security is off.

Safemode is off.

1 files and directories, 0 blocks = 1 total filesystem object(s).

Heap Memory used 101.38 MB of 319 MB Heap Memory. Max Heap Memory is 1.73 GB.

Non Heap Memory used 48.51 MB of 49.63 MB Commited Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	439.21 GB
DFS Used:	168 KB (0%)
Non DFS Used:	28.2 GB
DFS Remaining:	388.5 GB (88.45%)
Block Pool Used:	168 KB (0%)
DataNodes usages% (Min/Median/Max/stdDev):	0.00% / 0.00% / 0.00% / 0.00%
Live Nodes	7 (Decommissioned: 0, In Maintenance: 0)

6.2 数据处理

- 数据来源：网络销售数据，选择其中需要的字段如商品编号、商品名称、类别名称、订单信息、买家信息等等。
- 数据处理：通过KETTLE进行预处理，但是由于一些数据过于奇怪因此我们通过脚本编造了一些数据，本项目的旨在考量数据的量，而不在于数据的真实性；通过python脚本计算商品销量、类别销量、月销量等数据存入csv表中，最终得到所有的可用数据。
- 建表：在hadoop中搭建hive，向hive中导入数据；在MySQL中建表，导入数据。

6.3 网页构建

- 数据库与后端：用JDBC从Hive中读取数据，或对MySQL进行增删改的操作;
- 后端：用Spring Boot做后端，处理从数据库得到的数据，如计算数量、排序后返回给前端;
- 前端与后端：使用AngularJS实现数据传输，做到前后端分离;
- 前端：使用bootstrap的框架构建网站，走纯白极简风，将数据多以图片的形式展现如多段柱状图、折线图等，方便用户分析销售趋势，增强用户体验。

7 环境与架构

7.1 分布式文件型数据库

- OS：Ubuntu 14.04.5 LTS (GNU/Linux 4.4.0-31-generic i686)
- Hardware：CPU：Core i5 RAM:4Gb
- Database：Hadoop 2.8.2 Hive 2.1.1
- JournalNode*3, Zookeeper*3, NameNode*3, DataNode*3;
- Interface：JDBC

7.2 关系型数据库

- OS：Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-32-generic x86_64)
- Hardware：CPU：Core i5 RAM：2Gb
- Database：MySQL 5.6.25
- Interface： JDBC

7.3 查询、展示程序

- Database Driver: JDBC
- Backend： Spring Boot
- Frontend：Angular.js HTML5

8 参考资料

1.http://blog.csdn.net/sikongyuanruo/article/details/53643404
2.http://blog.csdn.net/x1066988452/article/details/51940353
3.http://blog.csdn.net/vipyeshuai/article/details/50847281
4.https://www.cnblogs.com/felixzh/p/5869212.html
5.https://www.cnblogs.com/tgzhu/p/5790565.html