通讯数据处理与分析-设计报告

周彬韬 2001210564

2021年-4月-26日

1.项目背景与架构设计

1.1 项目背景

通讯记录的**元信息**蕴含着巨大的价值，我们获取不到加密后的通话内容（当然这也是敏感信息），但是可以获取号码的拨打方和接收方的所在城市，可以获取通话的时间长短、常用沟通的时间等等，这些信息都称为元数据。

对这些数据进行离线分析，可以统计月度、季度、年度话单，可以获取通话记录，从而构建一张巨大的社交网，可以洞察某个人的好友圈，可以推断、洞察从而进行精准化推荐。

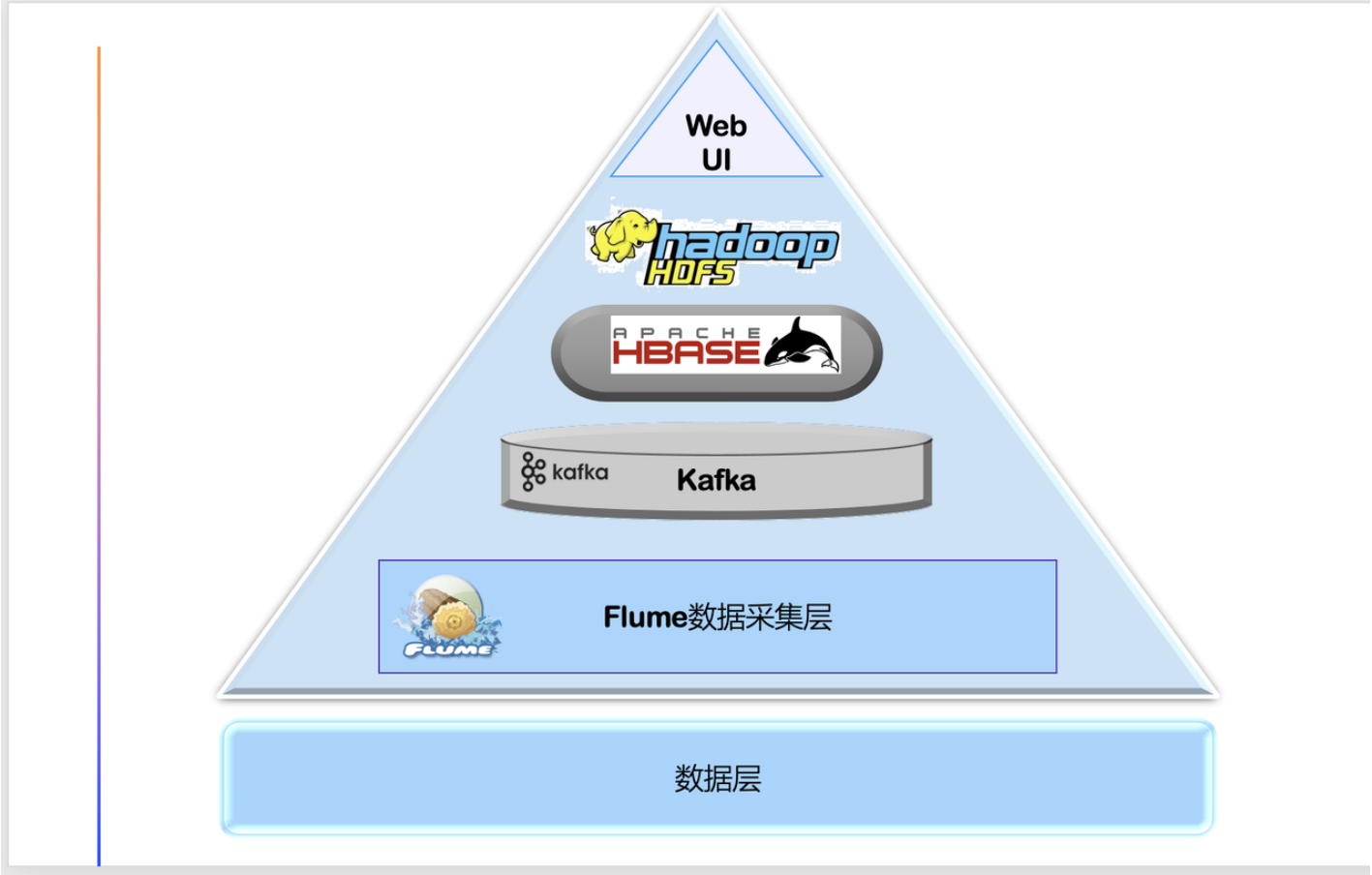
当前收集的数据：统计每天、每月以及每年的每个人的通话次数及时长以及部分个人信息。

1.2 技术选型调研

技术选型需要考虑到：数据量大小、业务需求、行业内经验、技术成熟度、开发维护成本等等因素

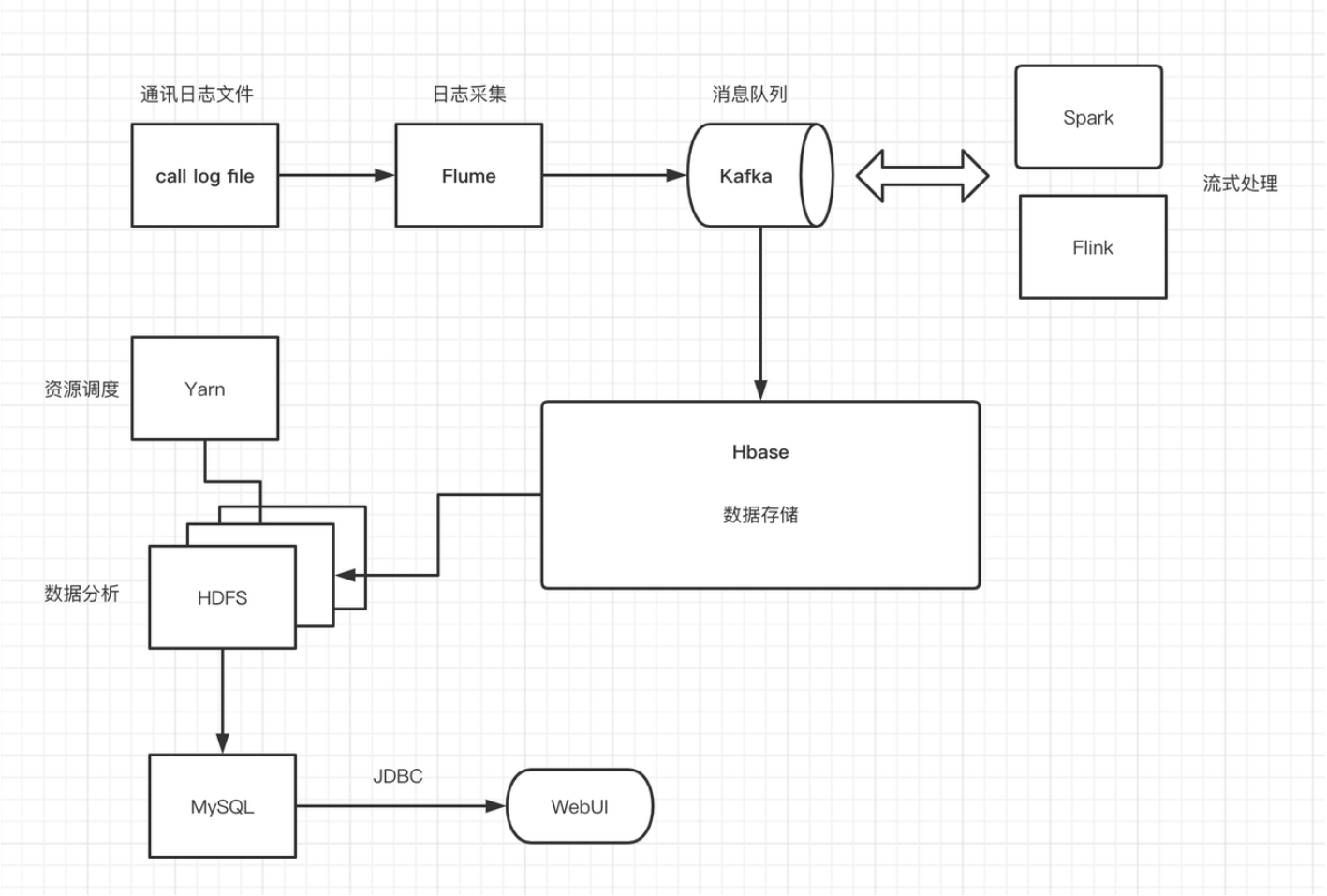
- 数据采集、传输：Flume，Kafka，Sqoop，DataX
- 数据存储：MySQL，HDFS，HBase，Redis，MongoDB
- 数据计算：Hive，Spark，Flink，Storm
- 数据查询：Presto，Kylin，Impala，Druid
- 数据可视化：Superset，QuickBI，DataV
- 任务调度：Azkaban、Oozie
- 集群监控：Zabbix
- 元数据管理：Atlas

1.3 架构设计

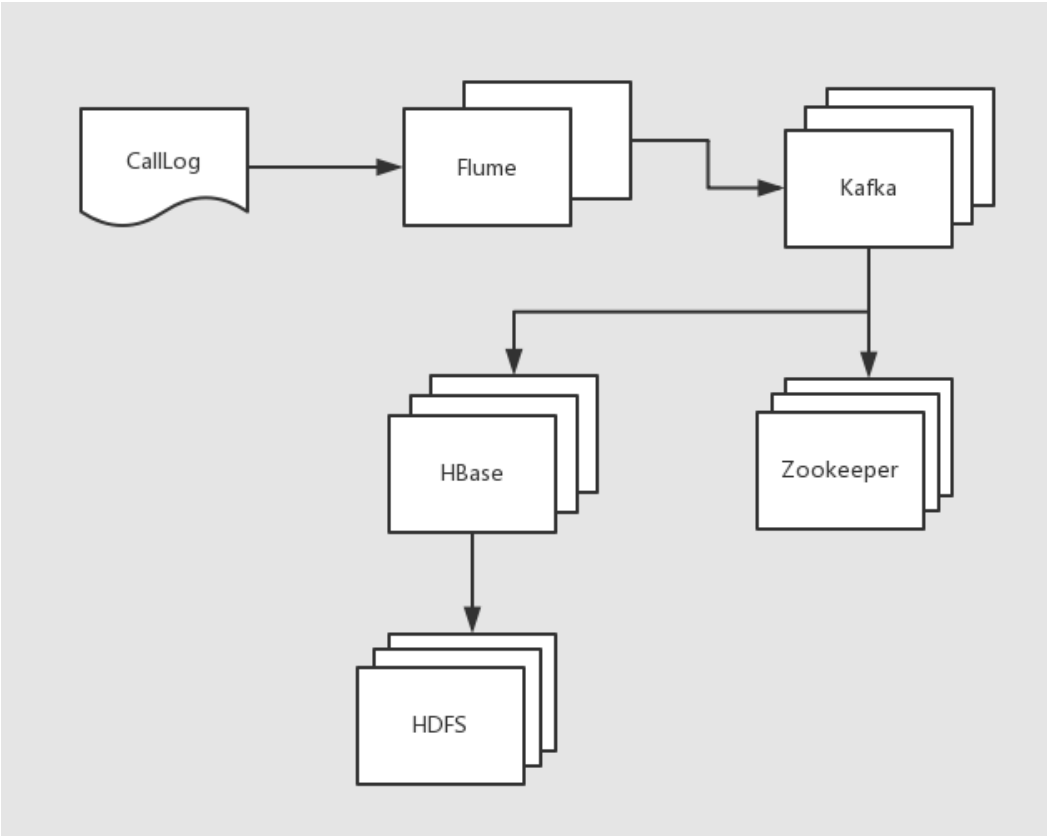


数据层是通话日志文件，由于**通话日志**是**log文件**，需要使用**海量日志采集框架Flume**进行采集，采集后的数据存储在**消息队列kafka**中，均衡生产者和消费者的生产消费速率，**kafka流出的数据输出到Hbase中**（对于kafka中流式数据分析可以采用spark和storm，本项目采用离线批处理，所以没有使用到storm）对存储的数据采用yarn-mapreduce的方式进行分析，由于HBase分析聚合后的数据量不大，所以统计分析后的结果存储在关系型数据库MySQL中，通过JDBC直接访问MySQL，方便将分析后的数据展示在前端WebUI上。

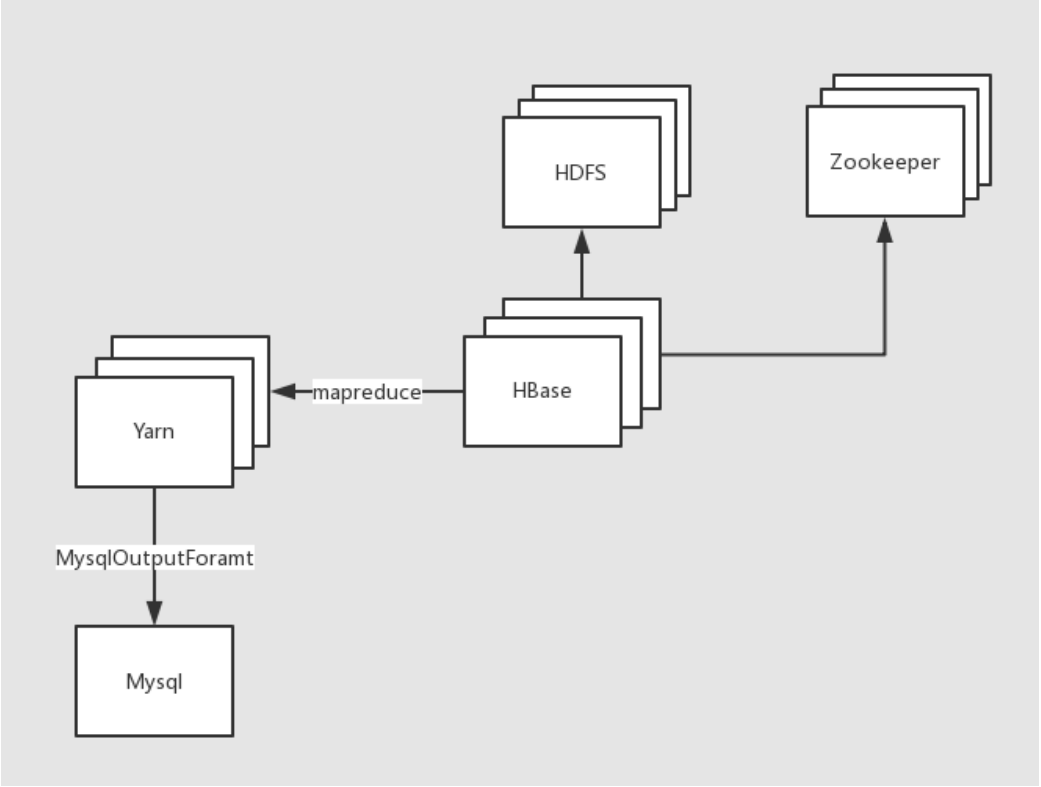
详细的架构流程图如下：


通讯日志文件
日志采集
消息队列
call log file
Flume
Kafka
Spark
Flink
流式处理
资源调度
Yarn
Hbase
数据存储
数据分析
HDFS
MySQL
JDBC
WebUI

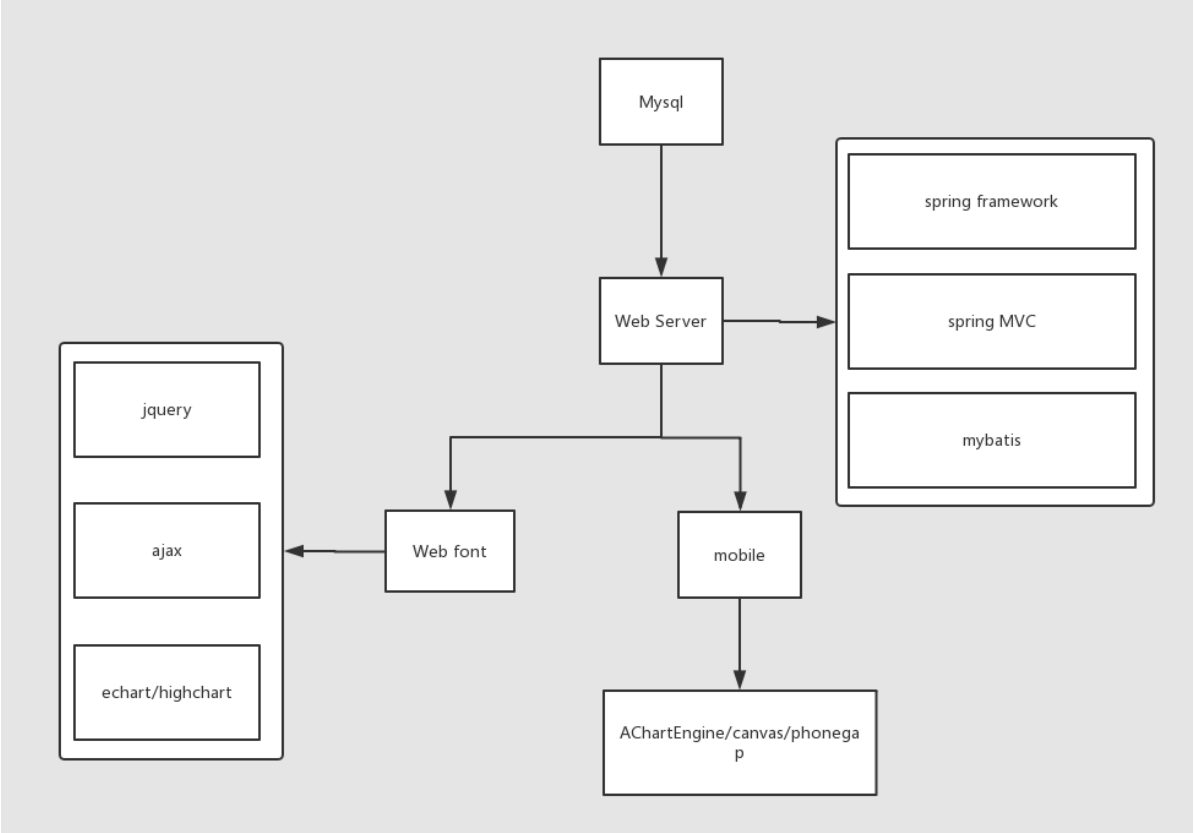
数据消费流程图如下：


CallLog
Flume
Kafka
HBase
Zookeeper
HDFS

数据处理流程图如下：



数据从MySQL到前端展示流程图如下：



2.环境搭建

2.1 系统环境

系统	版本
windows	10
Linux	CentOS7

2.2 集群搭建

框架、组件版本选择

框架	版本
Hadoop	3.1.3
Flume	1.9.0
Kafka	2.4.1
Spark	3.0.0
HBase	2.0.5
MySQL	5.7.6

集群分配

服务名称	子服务	服务器 hadoop102	服务器 hadoop103	服务器 hadoop104
HDFS	NameNode	✅		
	DataNode	✅	✅	✅

	SecondaryNameNode			✅
Yarn	NodeManager	✅	✅	✅
	Resourcemanager		✅	
Zookeeper	Zookeeper Server	✅	✅	✅
Flume				✅
Kafka		✅	✅	✅
Hbase	HMaster	✅		
	HRegionserver	✅	✅	✅

2.3硬件环境

	hadoop102	hadoop103	hadoop104
内存	4G	2G	2G
CPU	2核	1核	1核
硬盘	50G	50G	50G

3.数据生成

3.1 数据结构设计

搭建好了集群环境，接下来需要采集callLog日志文件

由于真实环境下的通信数据在尝试过后发现难以获取，爬取得到的数据杂乱不全，因此选择模拟数据产生

我们将在HBase中存储两个电话号码，以及通话建立的时间和通话持续时间，最后再加上一个flag作为判断第一个电话号码是否为主叫。姓名字段的存储我们可以放置于另外一张表做关联查询，设计数据结构和字段如下：

通话讯息表

字段名 / 列名	含义解释	举例
call1	拨打方 - 电话号码	127-1231-1231
call1_name	拨打方 - 姓名	小明
call2	接收方 - 电话号码	158-2131-7867
call2_name	接收方 - 姓名	小方
start_time	建立通话的时间	2021-04-01 18:37
start_time_ts	建立通话的时间戳	
duration	通话持续时间（秒）	600

个人基本信息表

字段名 / 列名	含义解释	举例（字段可能为空）
call	电话号码	127-1231-1231
call_name	名字	小明
flag	主动打电话还是被动接电话。 拨打方：true，接收方：false	true
age	年龄	24
address	地址	北京-海淀区

3.2 编程实现

数据模拟产生的代码逻辑：

a) 创建Java集合类存放模拟的电话号码和联系人；

b) 随机选取两个手机号码当作“主叫”与“被叫”（注意判断两个手机号不能重复），产出call1与call2字段数据；

c) 创建随机生成通话建立时间的方法，可指定随机范围，最后生成通话建立时间，产出date_time字段数据；

d) 随机一个通话时长，单位：秒，产出duration字段数据；

e) 将产出的一条数据拼接封装到一个字符串中；

f) 使用IO操作将产出的一条通话数据写入到本地文件中；

模拟生成的数据拼接成日志，并写入本地文件

Java

```java
//拼接日志
 private String productLog() {
    int call1Index = new Random().nextInt(phoneList.size());
    int call2Index = -1;
    String call1 = phoneList.get(call1Index);
    String call2 = null;
    while (true) {
        call2Index = new Random().nextInt(phoneList.size());
        call2 = phoneList.get(call2Index);
        if (!call1.equals(call2)) break;
    }
   //随机生成通话时长(30分钟内_0600)
    int duration = new Random().nextInt(60 * 30) + 1;
    //格式化通话时间，使位数一致
    String durationString = new DecimalFormat("0000").format(duration);
    //通话建立时间:yyyy-MM-dd,月份：0~11，天：1~31
    String randomDate = randomDate("2017-01-01", "2018-01-01");
    String dateString = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(Long.parseLong(randomDate));
    //拼接log日志
    StringBuilder logBuilder = new StringBuilder();
    logBuilder.append(call1).append(",").append(call2).append(",").append(dateString).append(",")
            .append(durationString);
```

```
        System.out.println(logBuilder);
        try {
            Thread.sleep(500);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return logBuilder.toString();
    }
    //将产生的日志写入到本地文件calllog中
    public void writeLog(String filePath, ProduceLog productLog) {
        OutputStreamWriter outputStreamWriter = null;
        try {
            outputStreamWriter = new OutputStreamWriter(new FileOutputStream(filePath, true), "UTF-8");
            while (true) {
                String log = productLog.productLog();
                outputStreamWriter.write(log + "\n");
                outputStreamWriter.flush();
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                assert outputStreamWriter != null;
                outputStreamWriter.flush();
                outputStreamWriter.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

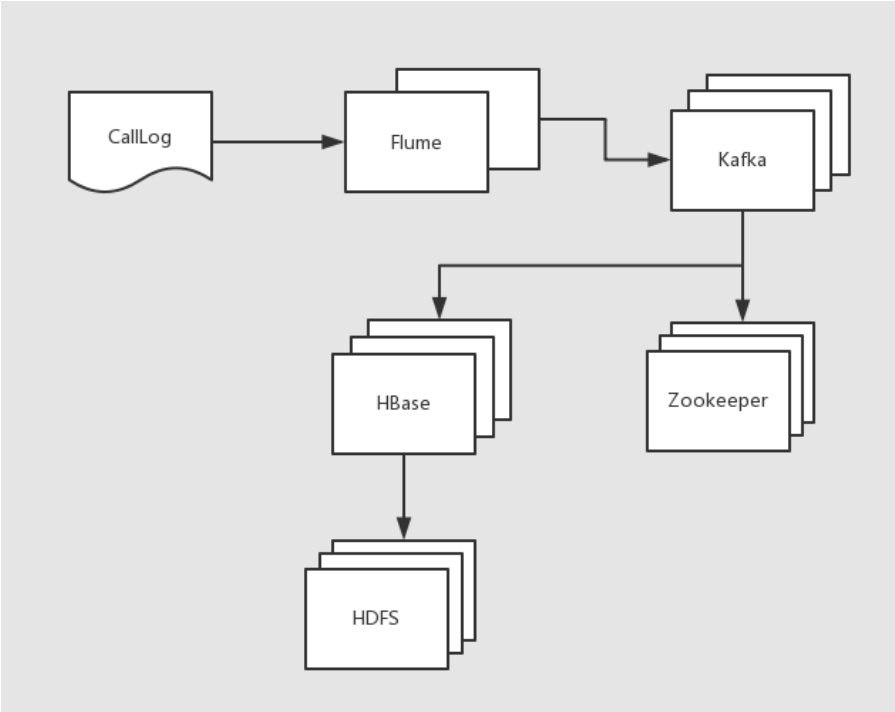
日志生成任务编写脚本，模拟源源不断产生日志数据

Bash

```
#!/bin/bash
java -cp /home/zbt/call/ producer.jar com.atguigu.producer.ProductLog /home/zbt/call/calllog.csv
```

4.数据处理

数据生成描述了如何模拟产生数据，这些数据最终是存在log文件中的，接下来需要进行真正的采集和处理，大体流程是将源源不断产生的实时数据通过flume采集到kafka然后供给hbase消费存储。



常用pipeline：线上数据 --> flume --> kafka --> HDFS

4.1 数据采集与消息队列存储过渡

思路：

a) 配置kafka，启动zookeeper和kafka集群；

b) 创建kafka主题；

c) 启动kafka控制台消费者（此消费者只用于测试使用）；

d) 配置flume，监控日志文件；

e) 启动flume监控任务；

f) 运行日志生产脚本；

g) 观察测试。

4.2 数据消费与存储

这个阶段可以引入flink和spark streaming进行流式数据实时分析，由于时间有限，没有在项目实践中加这个环节。

仅仅将消息队列kafka中读取出来的数据引入到HBase中存储

需要新建一些数据处理类，如HbaseConsumer、PropertiesUtil、HBaseUtil、HBaseDAO等等

4.3 数据查询

使用scan查看HBase中是否正确存储了数据，同时尝试使用过滤器查询扫描指定通话时间点的数据。进行该单元测试前，需要先运行数据采集任务，确保HBase中已有数据存在。

涉及到的查询代码较多，举例说明：已知要查询的手机号码以及起始时间节点和持续时间，查询该节点范围内的该手机号码的通话记录

拼装startRowKey和stopRowKey，即扫描范围， rowkey： 分区号_手机号码1_通话建立时间_手机号码2_主(被)叫标记_通话持续时间 01_15837312345_20170527081033_1_0180

比如按月查询通话记录，则startRowKey举例： regionHash_158373123456_20170501000000

stopRowKey举例： regionHash_158373123456_20170601000000

如果查找所有的，需要多次scan表，每次scan设置为下一个时间窗口即可，该操作可放置于for循环中

4.4 查询优化设计

使用HBase查找数据时，尽可能的使用rowKey去精准的定位数据位置，而非使用ColumnValueFilter或者SingleColumnValueFilter，按照单元格Cell中的Value过滤数据，这样做在数据量巨大的情况下，效率是极低的——如果要涉及到全表扫描。所以尽量不要做这样可怕的事情。注意，这并非ColumnValueFilter就无用武之地。现在，我们将使用协处理器，将数据一分为二。

思路：

a) 编写协处理器类，用于协助处理HBase的相关操作（增删改查）

b) 在协处理器中，一条主叫日志成功插入后，将该日志切换为被叫视角再次插入一次，放入到与主叫日志不同的列族中。

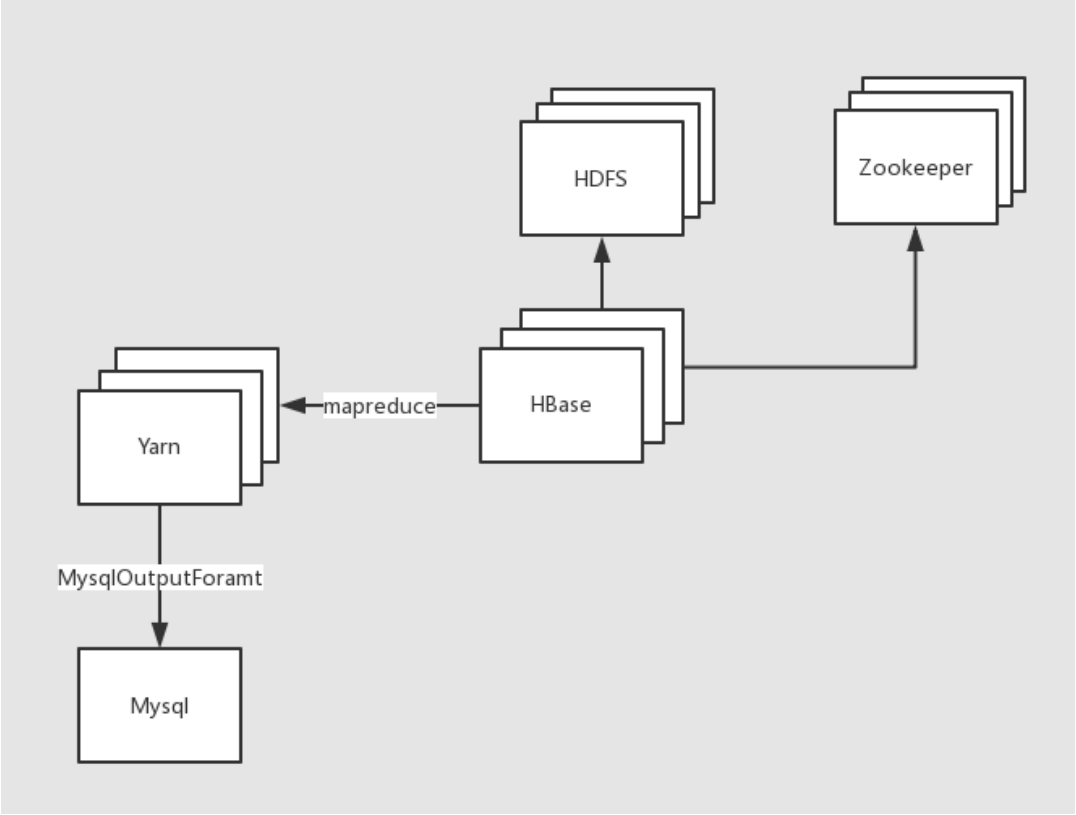
c) 重新创建hbase表，并设置为该表设置协处理器。

d) 编译项目，发布协处理器的jar包到hbase的lib目录下，并群发该jar包

e) 修改hbase-site.xml文件，设置协处理器，并群发该hbase-site.xml文件

5.数据挖掘与可视化

数据已经完整的采集到了HBase集群中，这次我们需要对采集到的数据进行分析，统计出我们想要的结果。注意，在分析的过程中，我们不一定会采取一个业务指标对应一个mapreduce-job的方式，如果情景允许，我们会采取一个mapreduce分析多个业务指标的方式来进行任务。



业务指标：

a) 用户每天主叫通话个数统计，通话时间统计。

b) 用户每月通话记录统计，通话时间统计。

c) 用户之间亲密关系统计。（通话次数与通话时间体现用户亲密关系）

5.1 需求分析

根据需求目标，设计出下述表结构。我们需要按照时间范围（年月日），结合MapReduce统计出所属时间范围内所有手机号码的通话次数总和以及通话时长总和。

思路：

a) 维度，即某个角度，某个视角，按照时间维度来统计通话，比如我想统计2021年所有月份所有日子的通话记录，那这个维度我们大概可以表述为2021年*月*日

b) 通过Mapper将数据按照不同维度聚合给Reducer

c) 通过Reducer拿到按照各个维度聚合过来的数据，进行汇总，输出

d) 根据业务需求，将Reducer的输出通过Outputformat把数据

数据输入：HBase

数据输出：Mysql

5.2 MySQL表结构设计

我们将分析的结果数据保存到Mysql中，以方便Web端进行查询展示。

1) 表：db_telecom.tb_contacts

用于存放用户手机号码与联系人姓名

	A	B	C
1	列	备注	类型
2	id	自增主键	int(11) NOT NULL
3	telephone	手机号码	varchar(255) NOT NULL
4	name	联系人姓名	varchar(255) NOT NULL

2) 表：db_telecom.tb_call

用于存放某个时间维度下通话次数与通话时长的总和。

	A	B	C
1	列	备注	类型
2	id_date_contact	复合主键（联系人维度id，时间维度id）	varchar(255) NOT NULL
3	id_date_dimension	时间维度id	int(11) NOT NULL
4	id_contact	查询人的电话号码	int(11) NOT NULL
5	call_sum	通话次数总和	int(11) NOT NULL DEFAULT 0
6	call_duration_sum	通话时长总和	int(11) NOT NULL DEFAULT 0

3) 表：db_telecom.tb_dimension_date

用于存放时间维度的相关数据

	A	B	C
1	列	备注	类型
2	id	自增主键	int(11) NOT NULL
3	year	年，当前通话信息所在年	int(11) NOT NULL
4	month	月，当前通话信息所在月，如果按照年来统计信息，则month为-1	int(11) NOT NULL
5	day	日，当前通话信息所在日，如果是按照月来统计信息，则day为-1	int(11) NOT NULL

4) 表：db_telecom.tb_intimacy

用于存放所有用户用户关系的结果数据。（作业中使用）

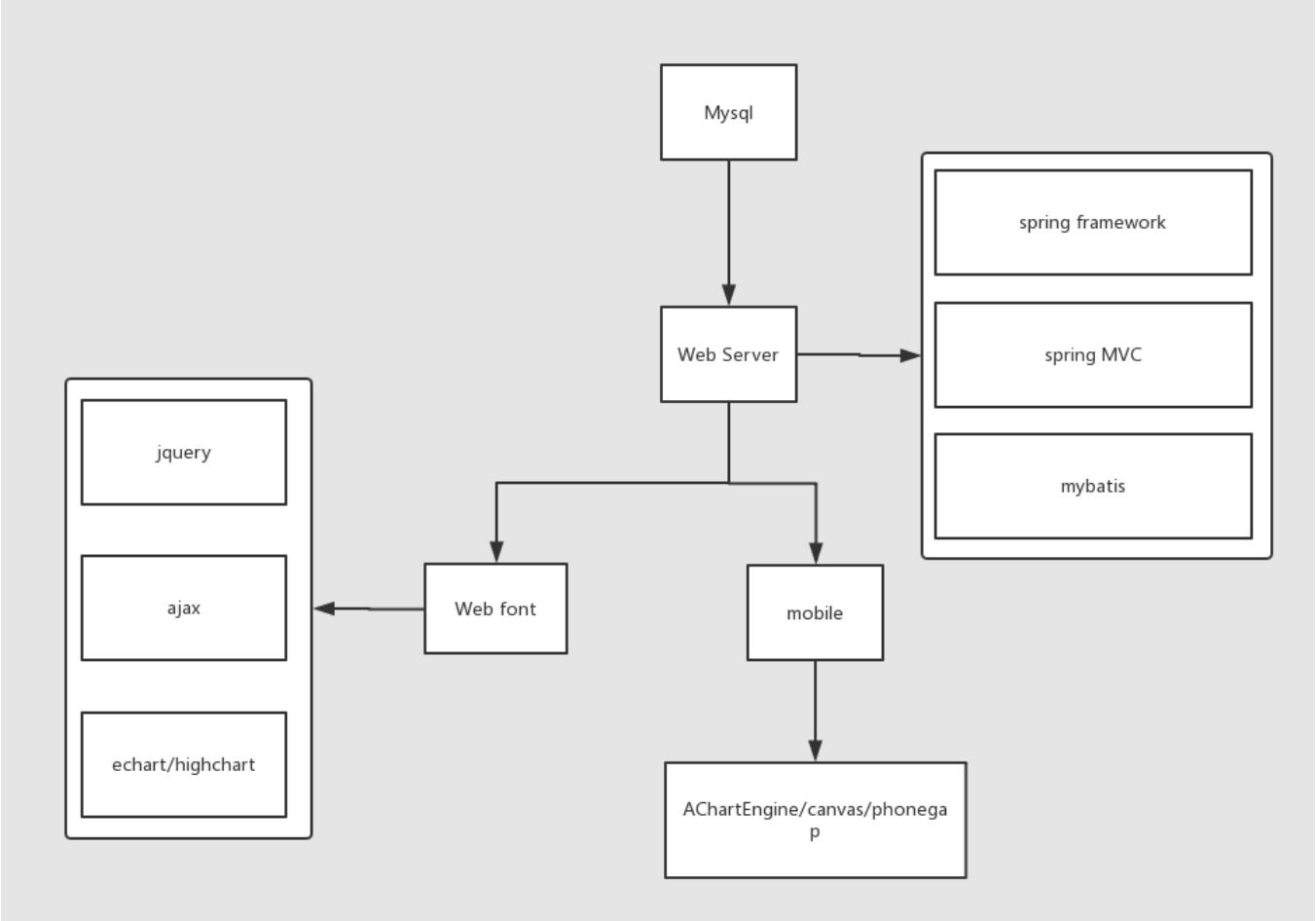
	A	B	C
1	列	备注	类型
2	id	自增主键	int(11) NOT NULL
3	intimacy_rank	好友亲密度排名	int(11) NOT NULL
4	id_contact1	联系人1，当前所查询人	int(11) NOT NULL
5	id_contact2	联系人2，与联系人为好友	int(11) NOT NULL
6	call_count	两联系人通话次数	int(11) NOT NULL DEFAULT 0
7	call_duration_count	两联系人通话持续时间	int(11) NOT NULL DEFAULT 0

5.3 类设计

	A	B
1	类名	备注
2	CountDurationMapper	数据分析的Mapper类，继承自TableMapper
3	CountDurationReducer	数据分析的Reducer类，继承自Reduccer
4	CountDurationRunner	数据分析的驱动类，组装Job
5	MySQLOutputFormat	自定义Outputformat，对接Mysql
6	BaseDimension	维度（key）基类
7	BaseValue	值（value）基类
8	ComDimension	时间维度+联系人维度的组合维度
9	ContactDimension	联系人维度
10	DateDimension	时间维度
11	CountDurationValue	通话次数与通话时长的封装
12	JDBCUtil	连接Mysql的工具类
13	JDBCCacheBean	单例JDBCConnection
14	IConverter	转化接口，用于根据传入的维度对象，得到该维度对象
15	DimensionConverter	IConverter实现类，负责实际的维度转id功能
16	LRUCache	用于缓存已知的维度id，减少对mysql的操作次数，提高
17	Constants	常量类

5.3 数据展示

接下来需要将用户按照不同维度查询出来的结果，展示到web页面上。



类表设计如下

	A	B
1	类名	备注
2	CallLog	用于封装数据分析结果的JavaBean
3	Contact	用于封装联系人的JavaBean
4	Contants	常量类
5	CallLogHandler	用于处理请求的Controller
6	CallLogDAO	查询某人某个维度通话记录的DAO
7	ContactDAO	查询联系人的DAO

5.4 定时脚本

新的数据每天都会产生，所以我们每天都需要更新离线的分析结果，所以此时我们可以用各种各样的定时任务调度工具来完成此操作。此例我们使用crontab来执行该操作。

Shell

```
# .----------------------------------------minute(0~59)
# " .---------------------------------------hours(0~23)
# " " .--------------------------------------day of month(1~31)
# " " " .-------------------------------------month(1~12)
# " " " " .------------------------------------day of week(0~6)
# " " " " " .-----------------------------------command
# " " " " " "
# " " " " " "
0 0 * * * /home/zbt/call/analysis.sh
```

6.总结

6.1 项目难点

在设计和实现过程中遇到的主要难点有：

- [ ] 各个大数据组件的兼容问题，利用三台虚拟机进行集群环境的搭建
- [ ] 数据爬取由于数据脏乱、字段缺失，爬取ip被封等等问题，尝试失败后，转换思路，设计数据结构，模拟数据生成，并写出日志到本地文件系统。
- [ ] 各个组件的环境配置部分。
- [ ] linux下一些脚本的编写，如：文件集群xsyc分发、crontab定时脚本的编写等。
- [ ] 查询过程中一行行扫描，速度太慢，从而进行查询优化的设计。
- [ ] 最终mysql表的设计与MySQL到使用superset进行前端可视化展示。

6.2 小结

本文是通讯数据处理分析的设计报告，先从宏观上介绍了项目背景，接下来介绍了技术选型与环境搭建，然后分章节介绍了数据分析处理与可视化的流程和大体设计思路，最后是本设计文档的总结部分，感谢帮助过我的小伙伴们。