

公开课主题： 京东云海开放数据服务架构 及 基于Spark-Streaming的实时计算服务平台

主讲人： 廖晓辉

京东云平台-开放云部

Agenda



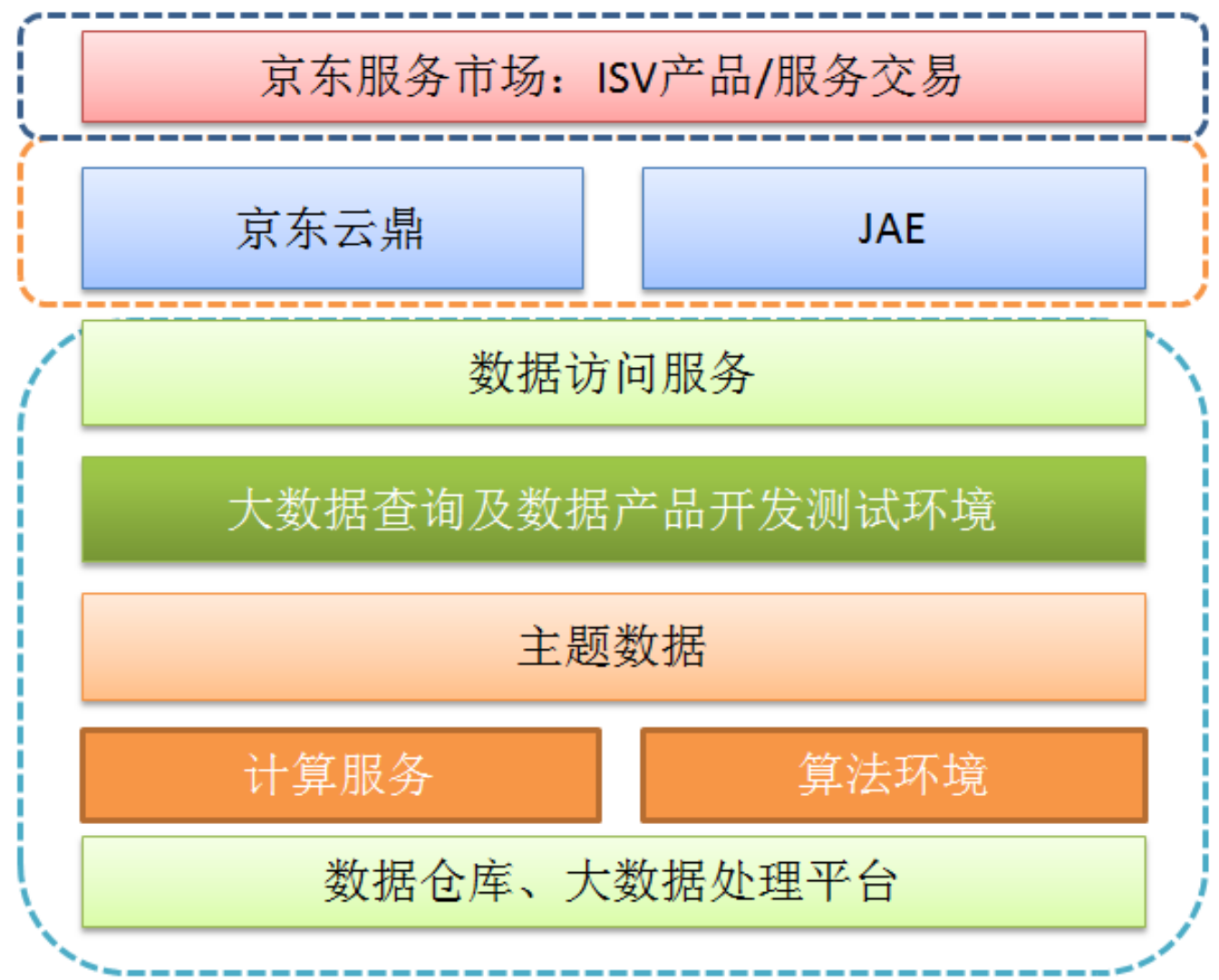
- 京东云海数据开放服务架构
 - 云海简介
 - 底层技术平台架构
- 实时计算服务平台介绍
 - 技术要点
 - 运行和监控
 - Logging
 - 示例展示
 - 典型问题
 - 性能调优实践

云海简介



- 目前开放京东商品、商家、客服绩效、品牌等几大主题数据，并开放几十个实时指标订阅；为ISV提供海量数据分布式存储计算平台，提供完整的云端数据仓库解决方案。
- DaaS
 - Data warehouse as a Service
 - Data development platform as a Service
- 自助计算，云端工作台
- 灵活数据接口
 - 使用JOS API规范，支持根据需求自定义API，供自己的APP调用

对接商家的需求：数据化运营，提高消费者满意度



云海前端功能展示



产品导航 | 客户服务

Hi, 欢迎来到京东云

- 首页
- 数据仓库
- 帮助文档
- 数据标准
- 开发者论坛
- 控制台

选择appkey:

查询内容: 表名

查询

0

去提交申请单

来源: 京东

获取类型: 计算 订阅

计算类型: 离线 实时

计算周期: 每小时 每天 每月

一级主题: 商家主题 商品主题 店铺基础 公共维表 客服绩效 行业主题 数据订阅

查看数据

表名:	dwb.dwb_itm_prod_class				
来源:	京东	等级:	L1	类型:	非卖家表
获取方式:	计算	计算类型:	离线	计算周期:	每天
描述:	产品分类表				
一级主题:	商品主题				
二级主题:	其他				

取消授权

表名:	dwb.dwb_loc_region				
来源:	京东	等级:	L1	类型:	非卖家表
获取方式:	计算	计算类型:	离线	计算周期:	每天
描述:	地区维表				
一级主题:	公共维表				
二级主题:	地域维表				

取消授权

云海前端功能展示



云海 Beta

数据仓库

数据管理

数据开发

生产部署

我的程序

基础数据

个人数据

新建程序

运行

停止

保存

部署任务

输入筛选表名



- 基础数据
 - 店铺基础
 - 店铺PC流量
 - 店铺收藏
 - 店铺订单
 - 商家主题
 - 商家买家分析
 - 商家关注分析
 - 商家流量分析
 - 商家销售分析
 - 商品主题
 - 其他
 - 商品关注分析
 - 商品流量分析
 - 商品销售分析

新建程序_1

数据开发

运行记录

结果数据

小象科技

让你的数据产生价值



京东云
JCloud.com

云海 Beta

数据仓库

数据管理

数据开发

生产部署



程序管理

程序列表



任务管理

任务部署

任务列表

运行记录



接口发布

数据接口发布



接口列表

数据接口列表

生产部署 > 任务管理 > 任务部署

任务类型: ☒ 计算 ☐ 导出 ☐ 抽取

任务名称: 请填写, 不超过50字

任务ID: 审批通过后可见

运行周期: ☒ 每天 (09:00前产出数据)

程序类型: ☒ HIVE程序 ☐ MR程序

部署程序:

生产部署

依赖数据表:	表名	类型	所属任务ID	任务状态

产出结果表:	表名	类型

任务描述:

提交部署



程序管理

程序列表



任务管理

任务部署

任务列表

运行记录



接口发布

数据接口发布



接口列表

数据接口列表

生产部署 > 接口发布 > 数据接口发布

数据访问接口定义

接口名称: 请填写, 不超过50字

接口ID:

接口描述:

请输入接口描述

接口SQL:

请输入SQL

变量	默认值	描述	操作
变量	默认值	描述	操作
<input type="text"/>	<input type="text"/>	<input type="text"/>	添加

快速测试SQL

点击后自动保存, 并进行测试, 在下面返回结果

测试结果:

保存

提交发布

M/R开发支持

Create命令创建模板工程

IDE导入模板工程

IDE下进行业务逻辑开发

本地调试M/R

云海web部署上线

upload命令上传至云海运行

JDDP SDK目前仅支持如下几个命令:

```
create      create 创建程序, 会从京东云海服务器下载MR的naven模板工程到该命令参数指定的路径.
            使用格式为: create <progran path>
            如: create d:\var\test

debug      试运行指定的MR程序.
            使用格式为: debug [options] <progranName ! progranId> [runday]
            如: debug -a 123 -s 456 test
            或者: debug -a 123 -s 456 test 20140504

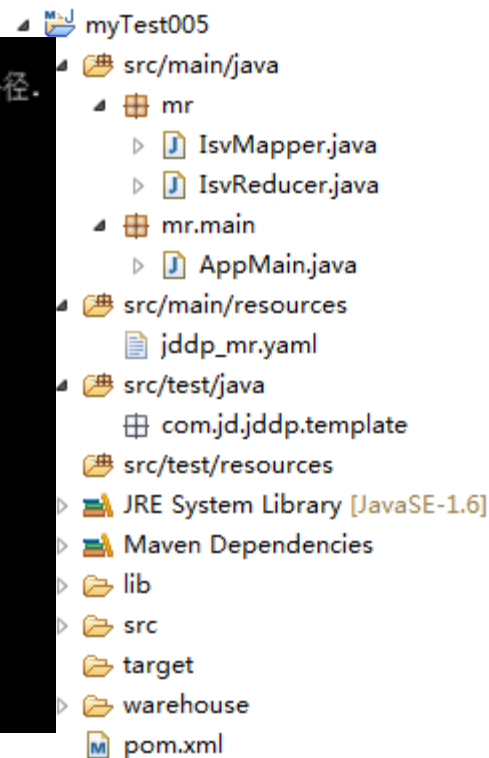
download    下载程序到当前路径中.
            使用格式为: download [options] <progranName !progranId>
            如: download -a 123 -s 456 test 或者: download -a 123 -s 456 100

kill        终止执行指定MR程序.
            使用格式为: kill [options] <progranName !progranId>
            如: kill -a 123 -s 456 test 或者: kill -a 123 -s 456 100

list        列出ISV用户下所有的MR程序列表.
            使用格式为: list [options]
            如: list -a 123 -s 456

show        获取并显示指定MR程序的配置文件内容.
            使用格式为: show [options] <progranName !progranId>
            如: show -a 123 -s 456 test 或者: show -a 123 -s 456 100

upload      上传程序到京东云海服务器.
            使用格式为: upload [options] <progranName !progranId>
            如: upload -a 123 -s 456 test 或者: upload -a 123 -s 456 100
```



技术平台架构介绍



数据同步

- 数据抽取和数据推送
- 关系库<->数据仓库
- HBase <->关系库



元数据管理

- 元数据管理
- 模型支持
- 血缘分析



基础平台

- 集群运维管理
- Hadoop/HBase/Spark/Storm/Kafka



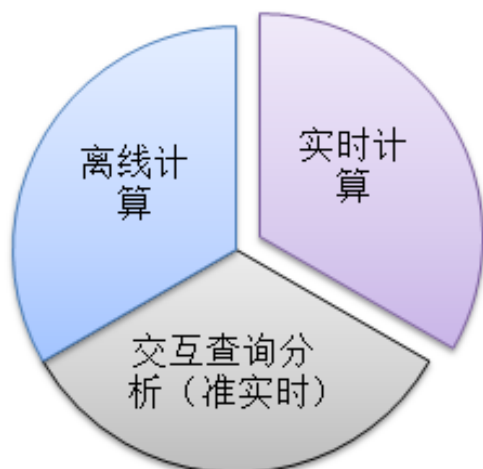
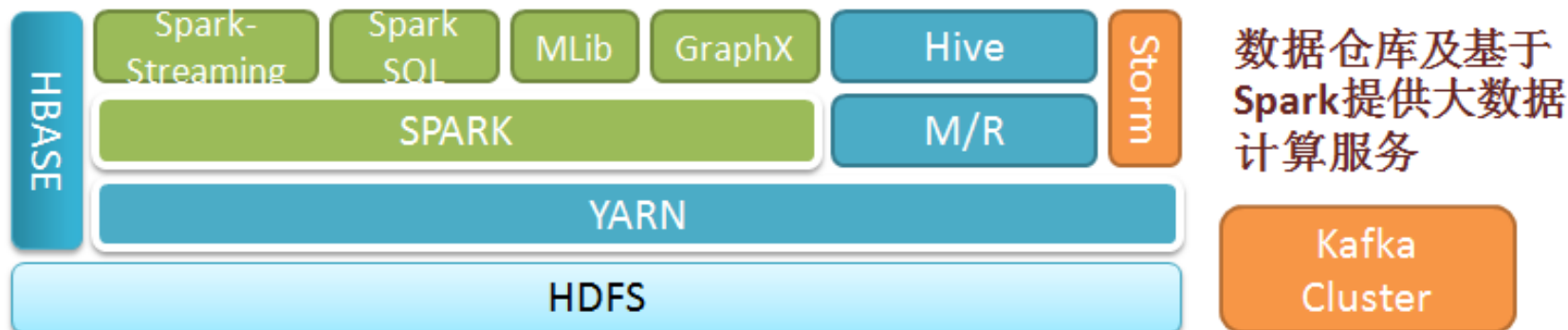
IDE

- | | |
|---------------|---------------------|
| • 数据集成开发环境 | • 任务部署（发布、配置） |
| • 权限管理 | • 任务管理 |
| • 测试集 | • workflow引擎 |
| • 任务部署（发布、配置） | • 数据展示（可视化）[长期计划提供] |

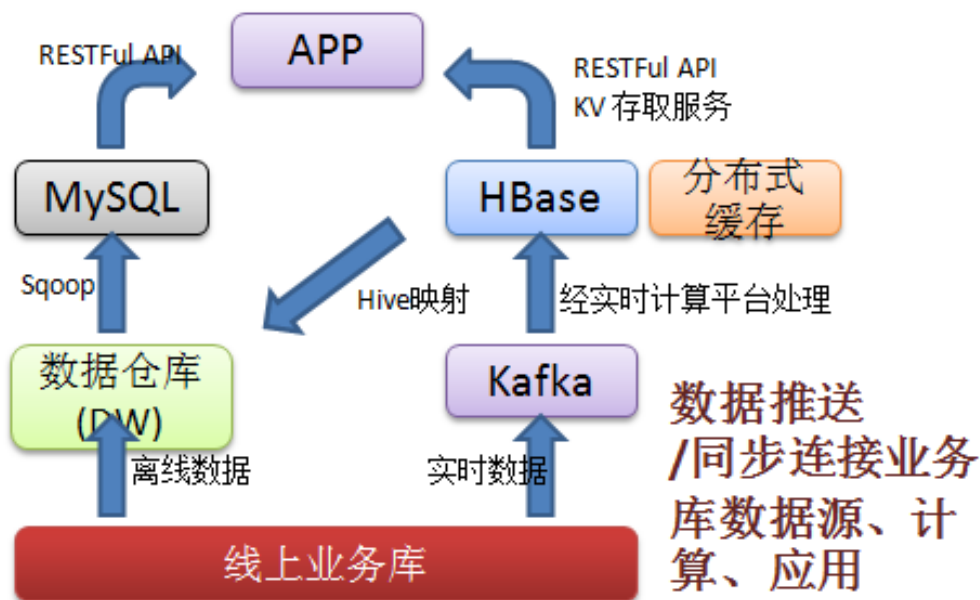
任务调度及 workflow



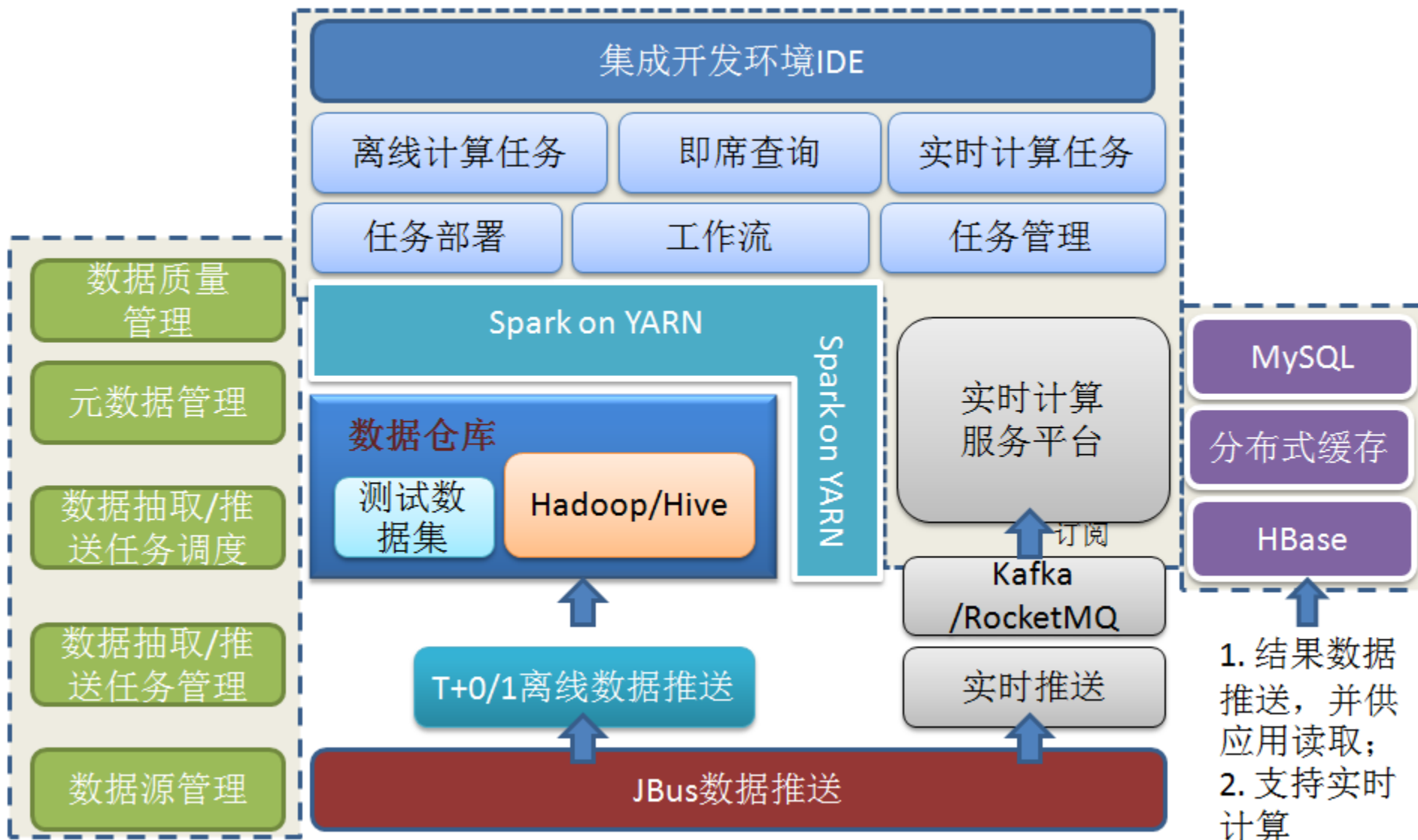
数据仓库及基础计算服务构建



支持业务的不同大数据处理需求



底层基础数据平台系统架构



1. 结果数据推送，并供应用读取；
2. 支持实时计算

- 基于Spark大数据处理平台，实际应用到生产环境
 - Spark/Spark-Streaming应用/运维，调优，及根据业务的源代码级别的扩展
- 实时计算服务平台
 - 基于Spark-Streaming开发，新的实时指标计算需求经简单配制或写SQL就可测试上线，大大降低开发成本。
 - 可关联实时增量流式数据和离线数据，实时增量计算模式。秒级计算延迟。
- 向实时数据仓库演进
 - 准实时同步线上业务库相关数据，并可被实时快速查询
 - 解决分库分表，数据量大且有更新场景的业务数据通常只能离线同步，延迟较长时间才能查询/分析的问题。
- HBase二级索引

实时计算服务平台

实时计算服务是？



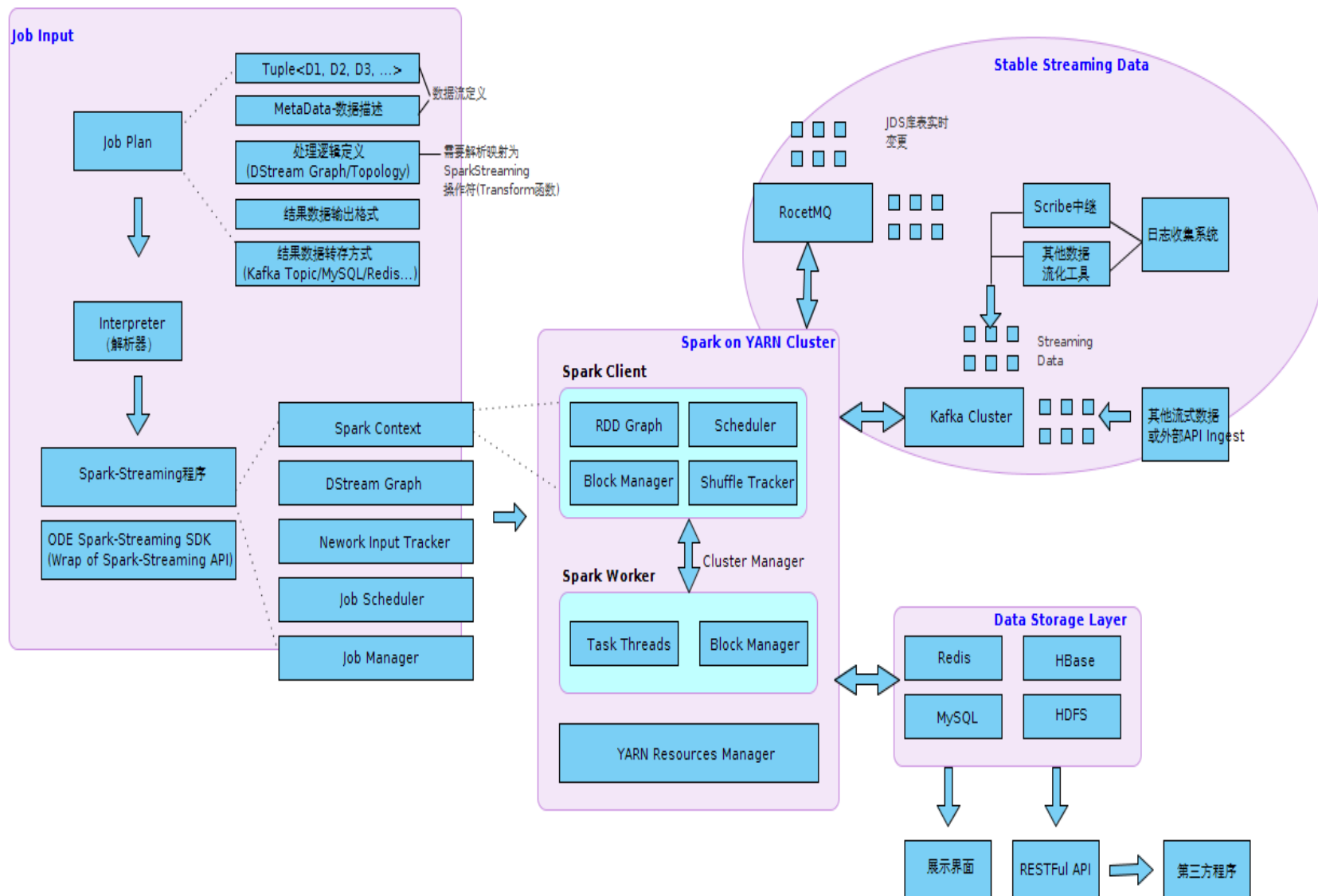
- 目的：
 - 将通用的功能或可公用的资源平台化服务化，让用户专注实时业务处理逻辑的开发，用SQL或SDK，要简单高效。
- 相关的计算环境：
 - 便捷的数据导入/流式数据接入：
 - 实时事件信息/流式数据
 - 导入前预处理：清洗/过滤...，SDK？
 - 高吞吐量流式数据支持。
 - 元数据管理
 - 大数据量（如全量数据）的高效导入。
 - 稳定可靠的流数据
 - 流数据HA，多份拷贝，保留一定时期历史数据，可重置消费点。
 - 多租户资源隔离，处理能力弹性扩容(吞吐量及响应时间)，容错机制(考虑恢复成本和运行时成本)，数据的严格有序到达和无序，及消费的有序无序.....
 - 结果数据获取：
 - 配合用户前端展示的相关数据获取API。
 - Pull历史数据；Push实时结果。

Why Spark-Streaming?



- 用Spark Stack解决Batch以及Streaming处理问题，不需维护多套Framework。
- YARN integration
- 长期演进考虑： Spark-Streaming有丰富的数据转换API，现团队在基于StreamSQL做大数据的增量计算

实时计算服务平台



实现技术要点



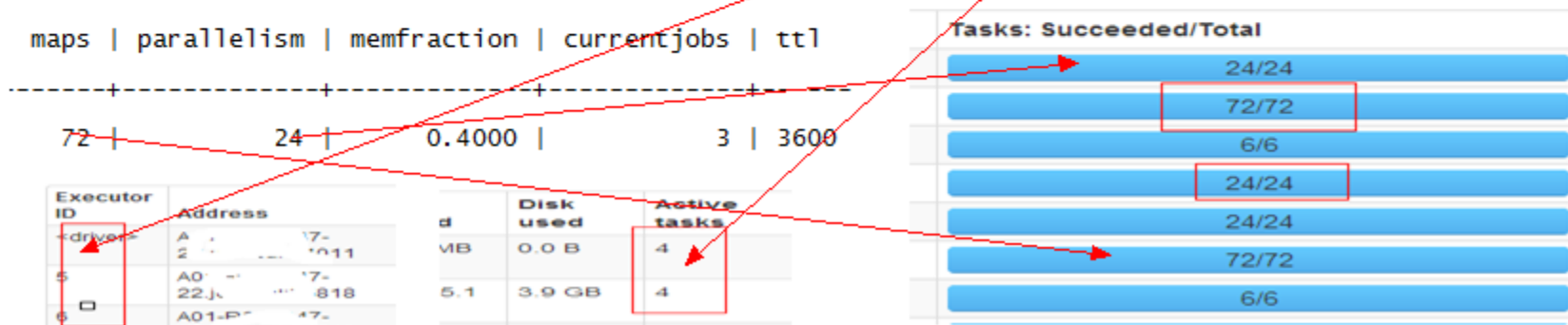
- 24/7 Streaming App
 - Continuous Running
 - Failover处理
 - 每天处理~10B消息
 - 秒级延迟
- 实时Metric，异常监控报警
 - 上下游监控
- 限流及异常恢复
 - 位点重置，故障预案
- 完善日志
 - 帮助尽快TroubleShooting，缩短异常到恢复时间
- 实时应用开发配置化、SQL化

Launch App on Yarn

1、启动命令 (spark on yarn)

```
SPARK_JAR=./assembly/target/scala-2.10/spark-assembly-0.9.0-incubating-hadoop2.3.0.jar ./bin/spark-class  
org.apache.spark.deploy.yarn.Client --jar  
/home/hadoop/running/kafkawithdiff.jar --class  
com.ode.realtime.app.AppKafkaTest --args 10010 --num-workers 6 --  
master-memory 3g --worker-memory 2g --worker-cores 4 --name  
10010_limitspeedworking
```

2、报表级的参数配置



App Configuration Sample



```
System.setProperty("spark.shuffle consolidateFiles", "****");
System.setProperty("spark.streaming.blockInterval", "****");
System.setProperty("spark.serializer",
"org.apache.spark.serializer.KryoSerializer");
System.setProperty("spark.default.parallelism", "****");
System.setProperty("spark.storage.memoryFraction", "****");
System.setProperty("spark.streaming.concurrentJobs", "****");
System.setProperty("spark.cleaner.ttl", "****");
System.setProperty("spark.shuffle.compress", "****");
System.setProperty("spark.shuffle,spill.compress", "****");
```

在创建JavaStreamingContext之前 进行属性配置

```
JavaStreamingContext ssc = new JavaStreamingContext(runmode,
appName,new Duration(3000), System.getenv("SPARK_HOME"),
JavaStreamingContext.jarOfClass(App.class));
```


[Stages](#)
[Storage](#)
[Environment](#)
[Executors](#)

Spark Stages

Total Duration: 21.73 h

Scheduling Mode: FIFO

Active Stages: 2

Completed Stages: 939


Failed Stages: 0

Active Stages (2)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total
156444	combineByKey at ShuffledDStream.scala:42	2014/09/04 15:49:16	56 ms	0/2
14	runJob at NetworkInputTracker.scala:182	2014/09/03 18:05:41	21.73 h	0/1

Completed Stages (939)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total
156441	collect at AppMQ4rowkey.java:147	2014/09/04 15:49:15	20 ms	32/32
156442	combineByKey at ShuffledDStream.scala:42	2014/09/04 15:49:15	8 ms	1/1
156439	collect at AppMQ4rowkey.java:147	2014/09/04 15:49:14	24 ms	32/32
156440	combineByKey at ShuffledDStream.scala:42	2014/09/04 15:49:14	104 ms	1/1
156437	collect at AppMQ4rowkey.java:147	2014/09/04 15:49:13	21 ms	32/32
156438	combineByKey at ShuffledDStream.scala:42	2014/09/04 15:49:13	12 ms	1/1


Stages Storage **Environment** Executors

Environment

Runtime Information

Name	Value
Java Home	/export/App/jdk1.6.0_25/jre
Java Version	1.6.0_25 (Sun Microsystems Inc.)
Scala Home	
Scala Version	version 2.10.3

Spark Properties

Name	Value
spark.app.name	realtime_100021409133553907
spark.cleaner.tti	3600
spark.default.parallelism	24
spark.driver.host	A01-10-147-100-100
spark.driver.port	39637
spark.filesystem.uri	hdfs://10.147.100.100:8020
spark.home	/export/App/spark-0.9.0-incubating
spark.httpBroadcast.uri	http://10.147.100.100:304
spark.jars	/data1/hadoop/hadoop-tmp/nm-local-dir/usercache/hadoop/filecache1
spark.local.dir	/tmp/hadoop-tmp/nm-local-dir/usercache/hadoop/appcache/appl
spark.master	yarn-standalone
spark.serializer	org.apache.spark.serializer.KryoSerializer
spark.shuffle.spill.compress	false
spark.shuffle.compress	false
spark.shuffle consolidateFiles	false
spark.storage.memoryFraction	0.4


[Stages](#)
[Storage](#)
[Environment](#)
[Executors](#)

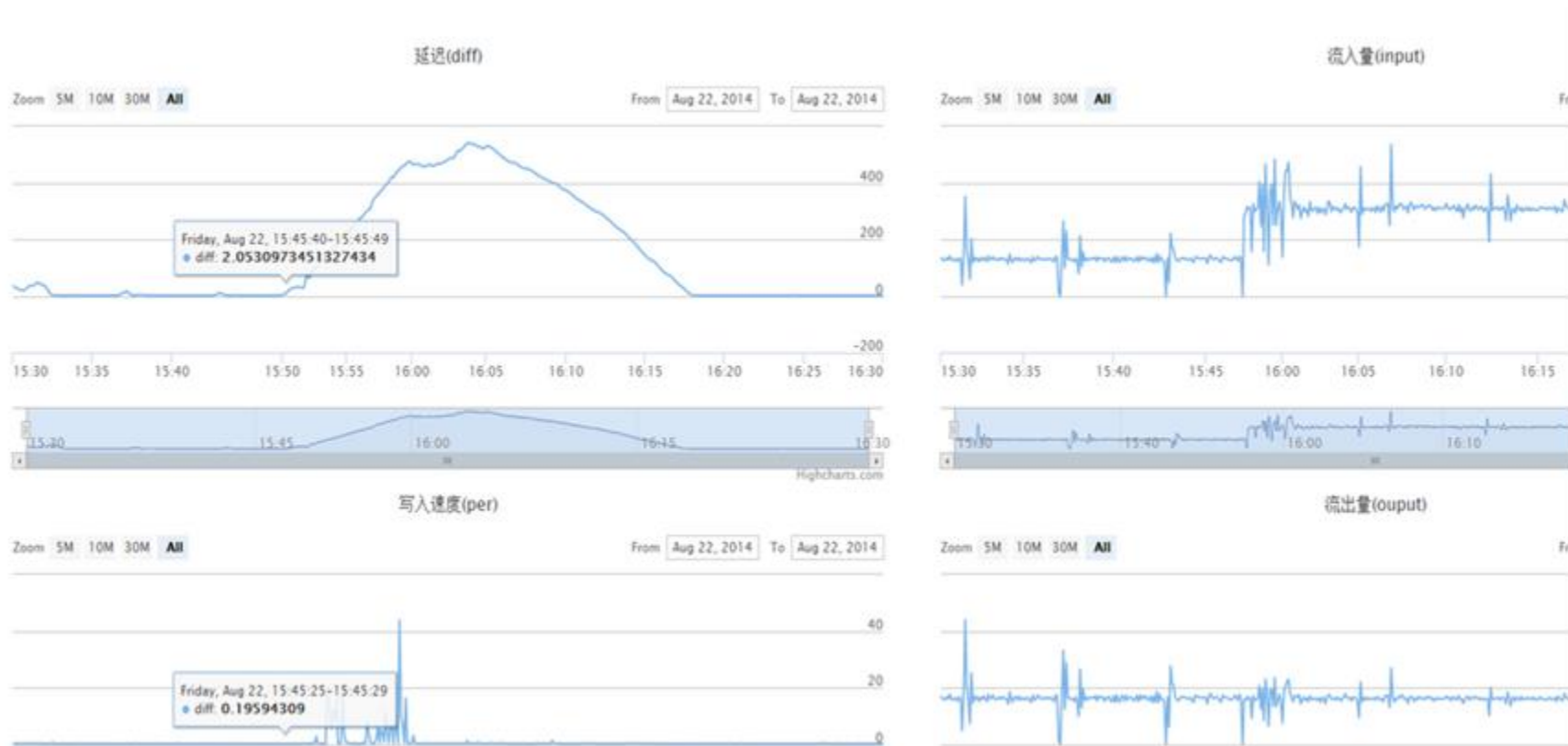
Executors (7)

Memory: 1569.4 MB Used (5.7 GB Total)

Disk: 6.9 GB Used

Executor ID	Address	RDD blocks	Memory used	Disk used	Active tasks	Failed tasks	Complete tasks	Total tasks	Task Time	Shuffle Read	Shuffle Write
3	A01-P00G-H47-27.jdl.local:36647	0	0.0 B / 785.1 MB	0.0 B	2	0	3002241	3002243	230.94 h	42.8 GB	111.9 GB
6	A01-P00G-H47-27.jdl.local:36418	7170	784.6 MB / 785.1 MB	3.5 GB	3	0	2428557	2428560	174.69 h	34.7 GB	109.9 GB
4	A01-P00G-H47-28.jdl.local:36175	7175	784.8 MB / 785.1 MB	3.5 GB	0	0	3227664	3227664	219.03 h	41.6 GB	135.4 GB
1	A01-P00G-H47-24.jdl.local:36963	0	0.0 B / 785.1 MB	0.0 B	0	0	2778004	2778004	214.15 h	62.9 GB	4.2 GB
2	A01-P00G-H47-38.jdl.local:37094	0	0.0 B / 785.1 MB	0.0 B	0	0	2855738	2855738	211.95 h	64.7 GB	4.5 GB
5	A01-P00G-H47-24.jdl.local:41438	0	0.0 B / 785.1 MB	0.0 B	0	0	2765404	2765404	214.29 h	62.7 GB	4.2 GB
<driver>	A01-P00G-H47-28.jdl.local:36607	0	0.0 B / 1092.3 MB	0.0 B	0	0	0	0	0 ms	0.0 B	0.0 B

监控示例



实时应用效果展示

实时概览 (数据更新时间: 2014-09-04 15:51:23)



浏览量: 7114

访客数: 2520



下单人数: 111

成交人数: 267



下单单数: 111

成交单数: 267



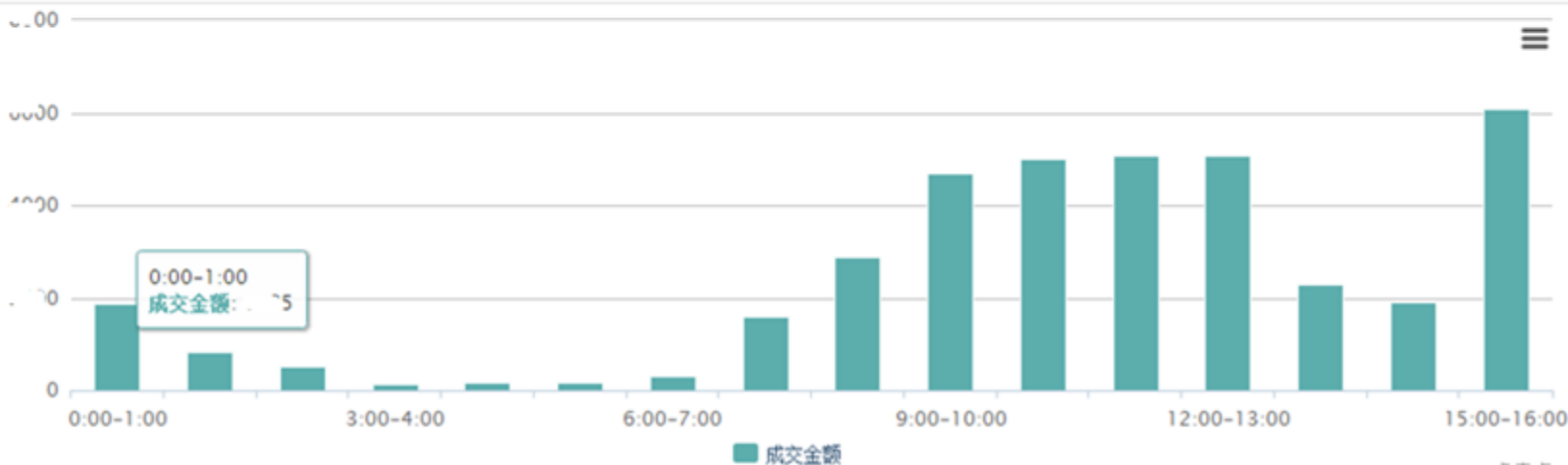
下单金额: 10000

成交金额: 63705

实时趋势

成交金额 成交人数 访客数 浏览量

各时段的数据



APP运行日志

- 1) 运行结束后可导出日志，但运行中的不能导出

```
yarn logs -applicationId application_14#####_0### > log###
```

- 2) 运行时日志查看

- 1 界面 不能看到worker的输出

ID	Logs
application 1407837983011 0287	logs

stderr : Total file length is 964059366 bytes.
stdout : Total file length is 53 bytes.

- 2 后台 可看到所有的输出，日志所在路径见hadoop的配置中的log4j

Spark

Stages

Storage

Environment

Executors

Executor ID	Address	RDD blocks
3	191.106.1147-221010000002	0

```
[root@A01 ~]# cd /export/Logs/hadoop/yarn/application_1407837983011_0287/container_1407837983011_0287_01_000002/
total 1868688
-rw-rw-r-- 1 hadoop hadoop 1913529019 Sep  3 09:20 stderr
-rw-rw-r-- 1 hadoop hadoop 0 Sep  2 19:20 stdout
[root@A01 ~]# pwd
/export/Logs/hadoop/yarn/application_1407837983011_0287/container_1407837983011_0287_01_000002
[root@A01 ~]#
```

典型问题

- Block还未进行计算就因ttl的设置而删除，导致not found错误

`System.setProperty("spark.cleaner.ttl", "3600");` 单位:秒

以上指自动清理一小时以前的RDD，如果你的计算延迟了一小时，就会报以下的错，并导致应用失败

解决办法：1) 提高性能，以减少延迟的时间，使延迟一直在安全范围内。

2) 对输入流进行**限速，如每秒只允许多少条**，这样，在生产者进行大量数据发送时，延迟还在可控范围之内

```
14/09/01 09:15:38 WARN TaskSetManager: Lost TID 15131 (task 452.0:0)
14/09/01 09:15:38 WARN TaskSetManager: Loss was due to java.lang.Exception
java.lang.Exception: Could not compute split, block input-0-1409534137000 not found
    at org.apache.spark.rdd.BlockRDD.compute(BlockRDD.scala:45)
    at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:241)
    at org.apache.spark.rdd.RDD.iterator(RDD.scala:232)
    at org.apache.spark.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:34)
    at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:241)
    at org.apache.spark.rdd.RDD.iterator(RDD.scala:232)
    at org.apache.spark.rdd.FilteredRDD.compute(FilteredRDD.scala:33)
    at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:241)
    at org.apache.spark.rdd.RDD.iterator(RDD.scala:232)
    at org.apache.spark.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:34)
    at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:241)
    at org.apache.spark.rdd.RDD.iterator(RDD.scala:232)
    at org.apache.spark.scheduler.ShuffleMapTask.runTask(ShuffleMapTask.scala:161)
    at org.apache.spark.scheduler.ShuffleMapTask.runTask(ShuffleMapTask.scala:102)
    at org.apache.spark.scheduler.Task.run(Task.scala:53)
    at org.apache.spark.executor.Executor$TaskRunner$anonfun$run$1.apply$mcV$sp(Executor.scala:213)
    at org.apache.spark.deploy.SparkHadoopUtil.runAsUser(SparkHadoopUtil.scala:49)
    at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:178)
    at java.util.concurrent.ThreadPoolExecutor$Worker.runTask(ThreadPoolExecutor.java:886)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:908)
    at java.lang.Thread.run(Thread.java:662)
14/09/01 09:15:38 WARN TaskSetManager: Lost TID 15130 (task 452.0:0)
```

典型问题Cont.



- HBase读写速度优化

- 解决办法:

- RowKey设计
 - 并发

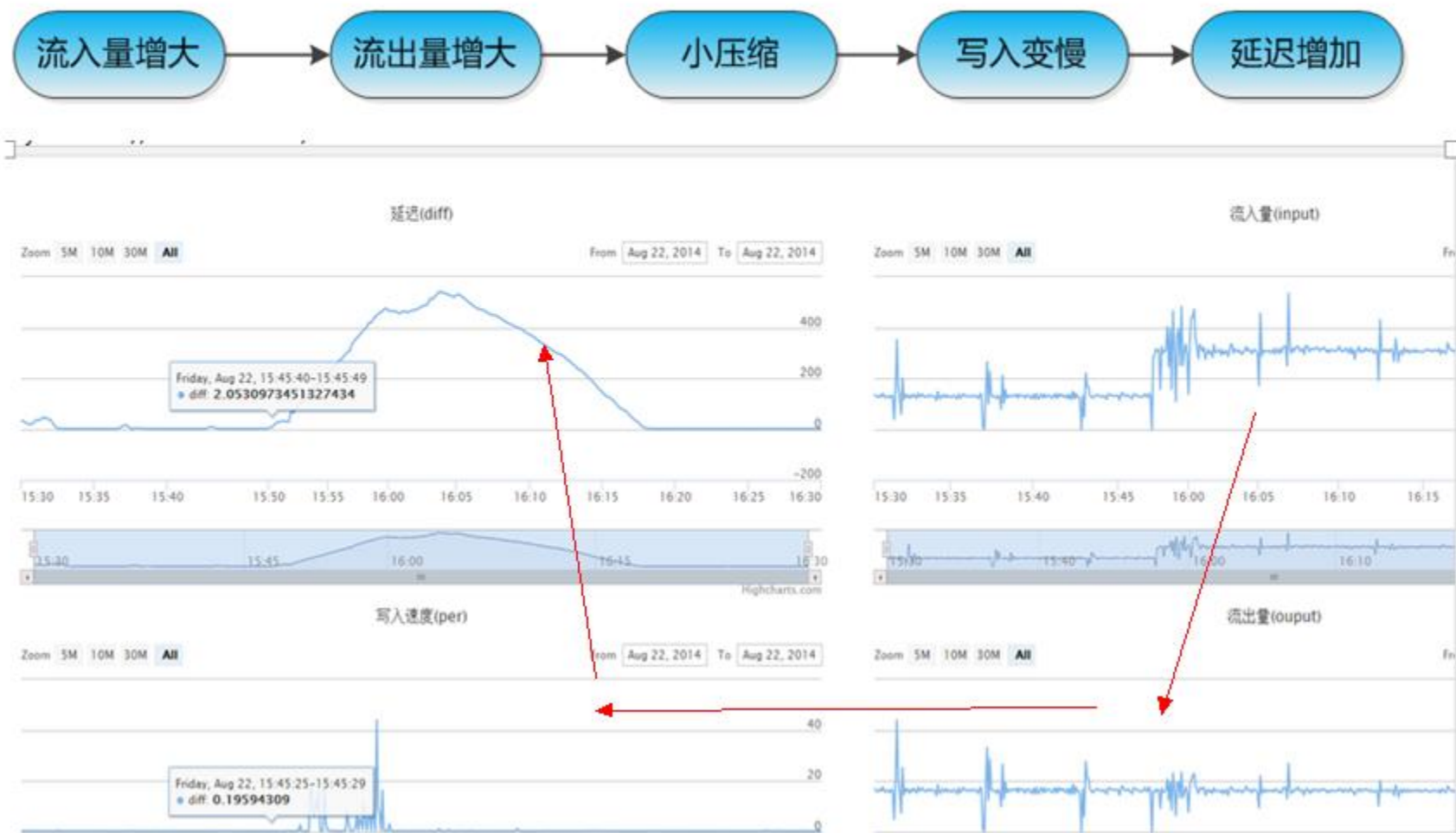
附：用HBase Increment的原子性做汇总计算，做到一个指标可拆解为多个App，结果叠加，简化计算逻辑，提高并发。

- INFO AMRMClientImpl: Waiting for application to be successfully unregistered.

- 一般是由于配置文件不正确引起，比如线上环境与配置环境打包是混淆。

典型问题Cont.

- HBase Minor Compact影响 恶性循环



- 问题1：建立一个数据接收流进行数据接收，数据会存放于接收work及备份work上。在任务调度时考虑到数据本地性，task调度到接收流和副本所在两个work上，其他work空闲，形成计算不均

解决：1) 设置多个接收流 2) 通过repartition调节RDD中partition数量，将task调度分散，使得计算均匀。

- 问题2：reduce task数目不合适

解决：默认为8，需根据实际情况进行调节。可调节参数spark.default.parallelism。通常，reduce数目设置为core数目的2---3倍。数量太大，造成很多小任务，增加启动任务的开销；数目太少，任务运行缓慢。

性能调优-Cont.



- 问题3: shuffle磁盘IO时间长
解决: 可以设置spark.local.dir为一组磁盘, 并尽量设置磁盘为IO速度快的磁盘。通过增加IO来优化shuffle性能。
- 问题4: map/reduce数量大, 造成shuffle小文件数目多, default: shuffle文件数目为map tasks * reduce tasks
解决: 通过设置spark.shuffle consolidateFiles为true, 来合并shuffle中间文件, 文件数为Cores * Reduce tasks数目。

性能调优-Cont.



- 问题5：GC或OOM问题严重

解决：调整spark.storage.memoryFraction。Default:0.6。Further，观察app运行过程中的GC实际情况，进行其他调节。

- 问题6：block not found

解决：调整spark.cleaner.ttl。RDD及元数据的过期时间。

性能调优-Cont.



- 问题7：序列化时间长或结果大。
解决：通过设置spark.serializer为org.apache.spark.serializer.KryoSerializer。
使用广播变量。
- 问题8：系统吞吐量不高
解决：设置spark.streaming.concurrentJobs.
- 问题9：单条记录处理时间长
解决：使用mapPartition替换map，提高Dstream RDD处理的并行度.

谢谢

We are hiring!
liaoxiaohui@jd.com

联系我们：

- 新浪微博：ChinaHadoop
- 微信公号：ChinaHadoop



让你的数据产生价值！

Backup Slides

1、MQ

```
JavaDStream<byte[]> messagesmq= RocketMQUtils.createStream(  
    ssc  
    , mqServer , groupName,consumerid , topic  
    ,tag , startStr , stopStr  
    , StorageLevel.MEMORY_AND_DISK_2()  
    , new DelayAndLimitCallback(consumerDelay, rateLimit)  
);
```

2、KAFKA

```
JavaPairDStream<String, String> messages = KafkaUtils.createStream(ssc ,  
    topicMap , kafkaParams);
```

```
messages.mapPartitions(new PairFlatMapFunction<Iterator<byte[]>, String, Double>() {  
    @Override  
    public Iterable<Tuple2<String, Double>> call(Iterator<byte[]> iterator) throws Exception {  
        List<Tuple2<String, Double>> list = new ArrayList<Tuple2<String, Double>>();  
        todo();  
        list.add(new Tuple2<String, Double>("received", Double.valueOf(data.size())));  
        return list;  
    } }).filter(new Function<Tuple2<String, Double>, Boolean>() {  
    @Override  
    public Boolean call(Tuple2<String, Double> stringDoubleTuple2) {  
        if (stringDoubleTuple2 == null) {  
            return false;  
        }  
        return true;  
    }  
}).reduceByKey(new Function2<Double, Double, Double>() { @Override  
    public Double call(Double i1, Double i2) {  
        return i1 + i2;  
    }  
});
```



```
javaPairDStream.foreachRDD(new Function2<JavaPairRDD<String,  
    List<Double>>, Time, Void>() {  
    @Override  
    public Void call(JavaPairRDD<String, List<Double>>  
        stringIntegerJavaPairRDD, Time time) {  
        List<Tuple2<String, List<Double>>> list = stringIntegerJavaPairRDD.collect();  
        for (Tuple2 t : list) {  
            todo();  
        }  
        return null;  
    }  
});
```