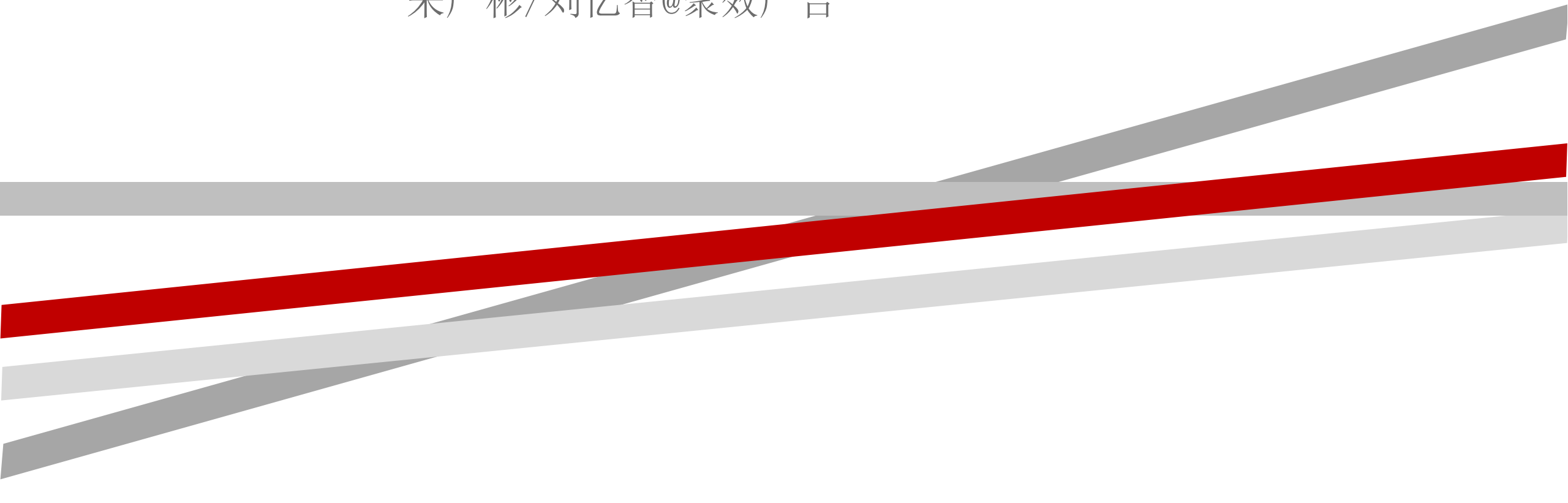
Spark在计算广告领域的应用实践

朱广彬/刘忆智@聚效广告



目录

计算广告简介

◇计算广告学，顾名思义是计算驱动广告的学科，这是相对传统的广告而言的

◇计算广告学是一门以计算技术驱动的广告营销科学

◇传统广告

- 纸媒广告、电视广告、墙体广告……
- 受众不够精准，投放效果差

◇计算广告

- 网络受众体量大，能够做到精准投放
- 更加关注效果

◇计算广告面临的挑战：

- 在特定语境下特定用户和相应的广告之间找到"最佳匹配"

计算广告与大数据

- 计算广告之所以能够兴起，主要原因也来自于互联网公司的大数据能力
  - 庞大的数据量：上百亿/天，TB~PB级数据规模
  - 数据复杂度：非结构化、零散稀疏
  - 对实时性的苛刻要求：线上竞价 200ms以内，离线学习模型最好尽快反哺线上
- 涉及的大数据处理技术：
  - 大规模搜索和文本分析
  - 数据处理与ETL
  - 统计模型
  - 机器学习
  - 实时流计算
  - ……

聚效广告平台

◈成立时间

- 2009年

◈关键词

- DSP 移动 跨屏 自助式 效果营销 一站式平台

◈研发中心

- 上海 北京

◈员工情况

- 180余名精英团队，60%以上为技术研发中坚力量

◈客户数量

- 服务超过40000家广告主，覆盖众多一线知名品牌

我们的数据

我们的数据

- 全网各大媒体，上亿广告位
- 每天 100亿+次请求，20TB+ 日志量
- 集群规模：500+，存储容量：16PB+，已用11PB+
- 日均Job数：3500+，日处理数据量：250TB+

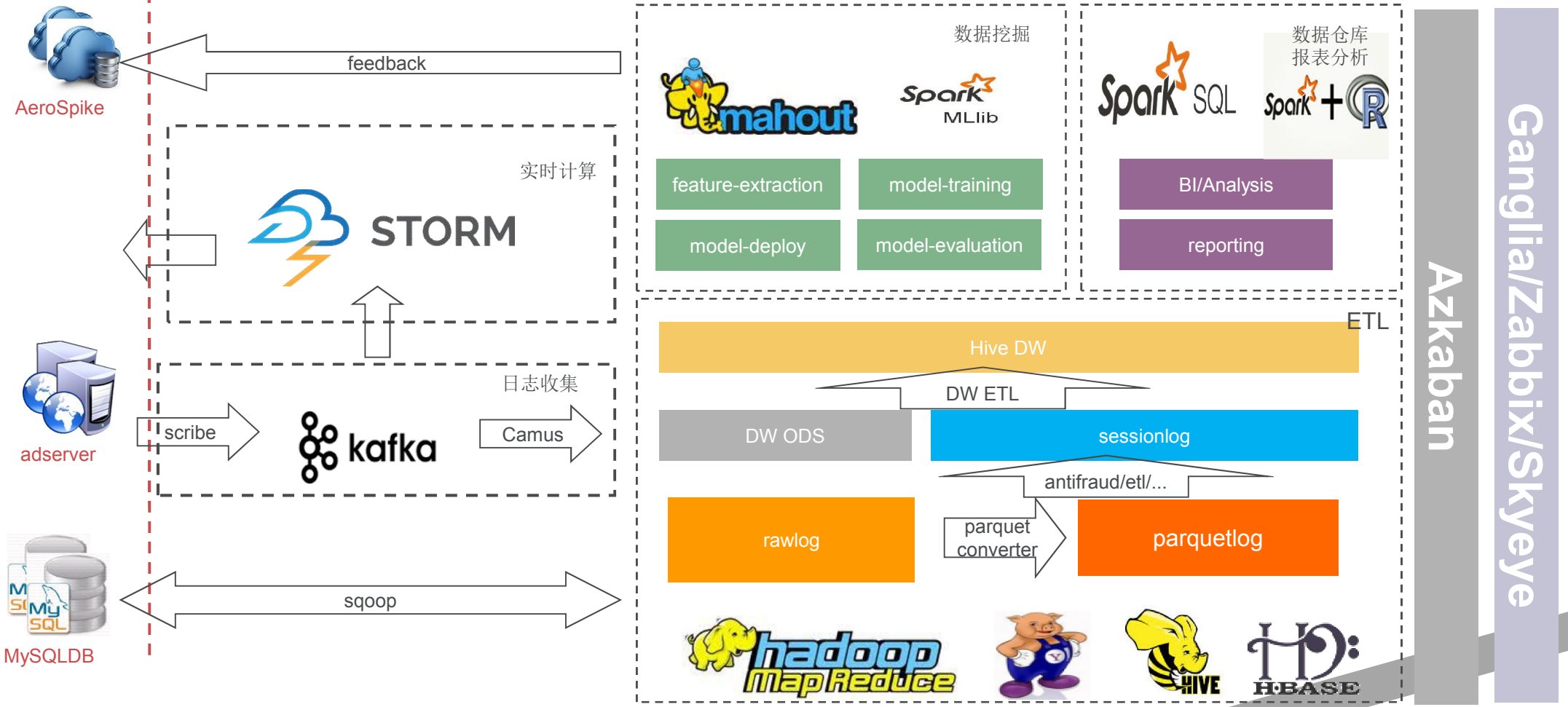
我们如何用数据

- 统计：日志收集与处理，数据仓库
- 预测：机器学习与数据挖掘，CTR预估，人群分类，个性化定向等等

我们的数据处理架构

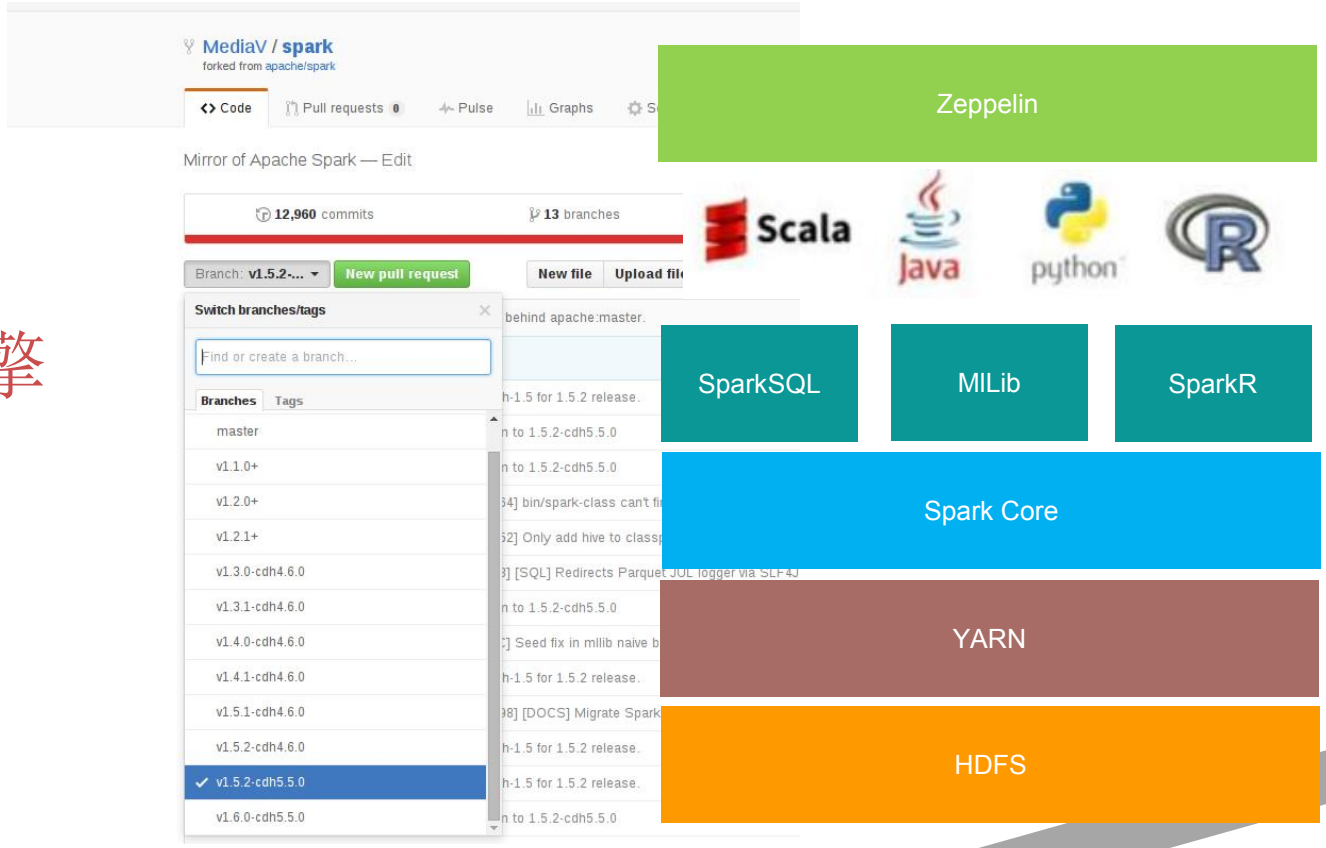
- 以Hadoop为核心，Kafka、HBase、Hive、Spark、Storm

Data Platform Architecture

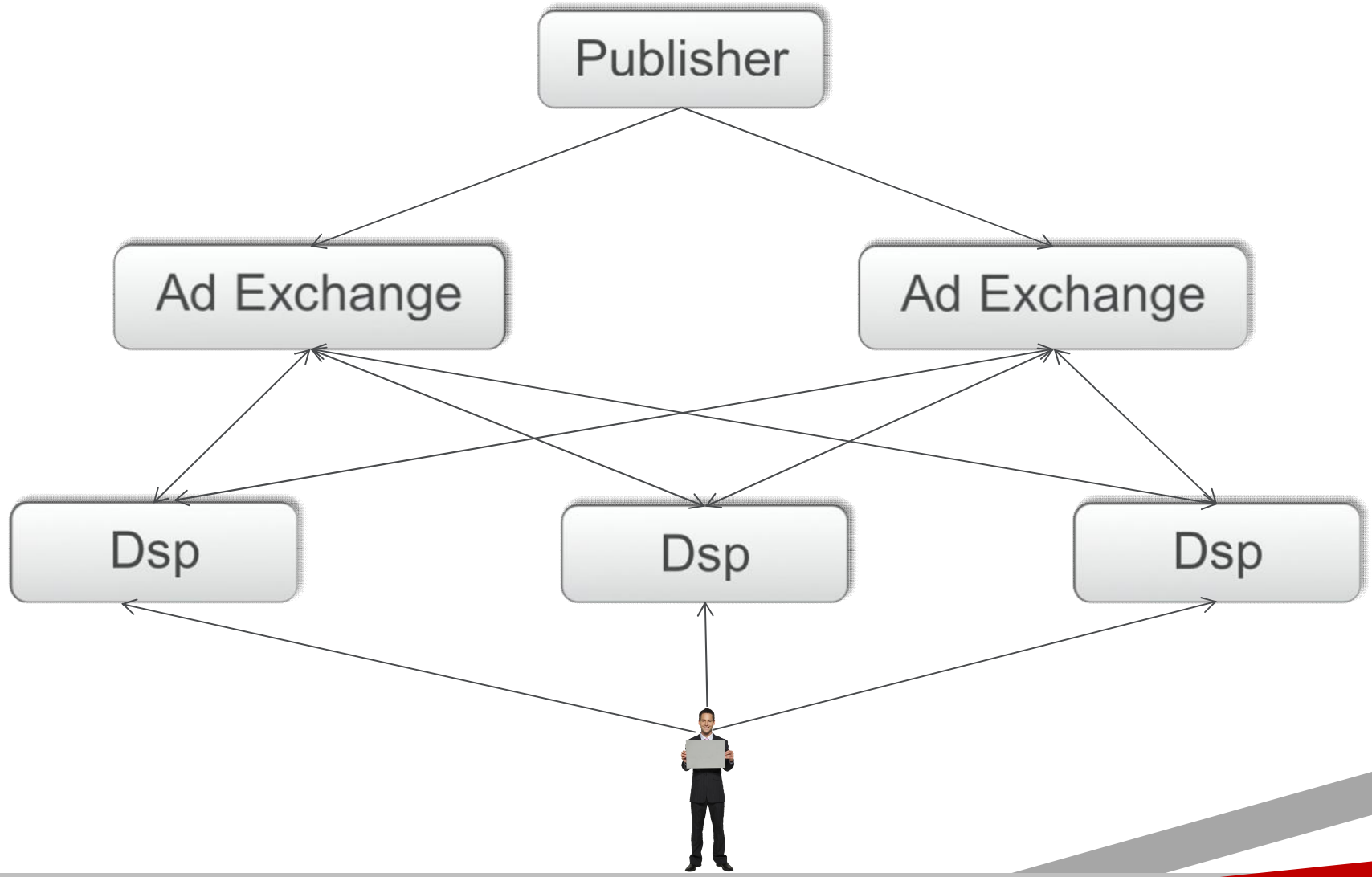


Spark@MVAD

- CDH + MV-Spark 1.5.2
  - 紧跟Apache社区, 0.9~1.6
  - on Yarn部署
- MLlib 机器学习与模型训练
- SparkSQL 数据仓库查询引擎



在线广告程序化交易



Spark MLLib在聚效的应用

◈ 广告点击率预估

- Logistic Regression On Spark
- 数据规模
- 评价指标
- Spark v.s. Hadoop MR

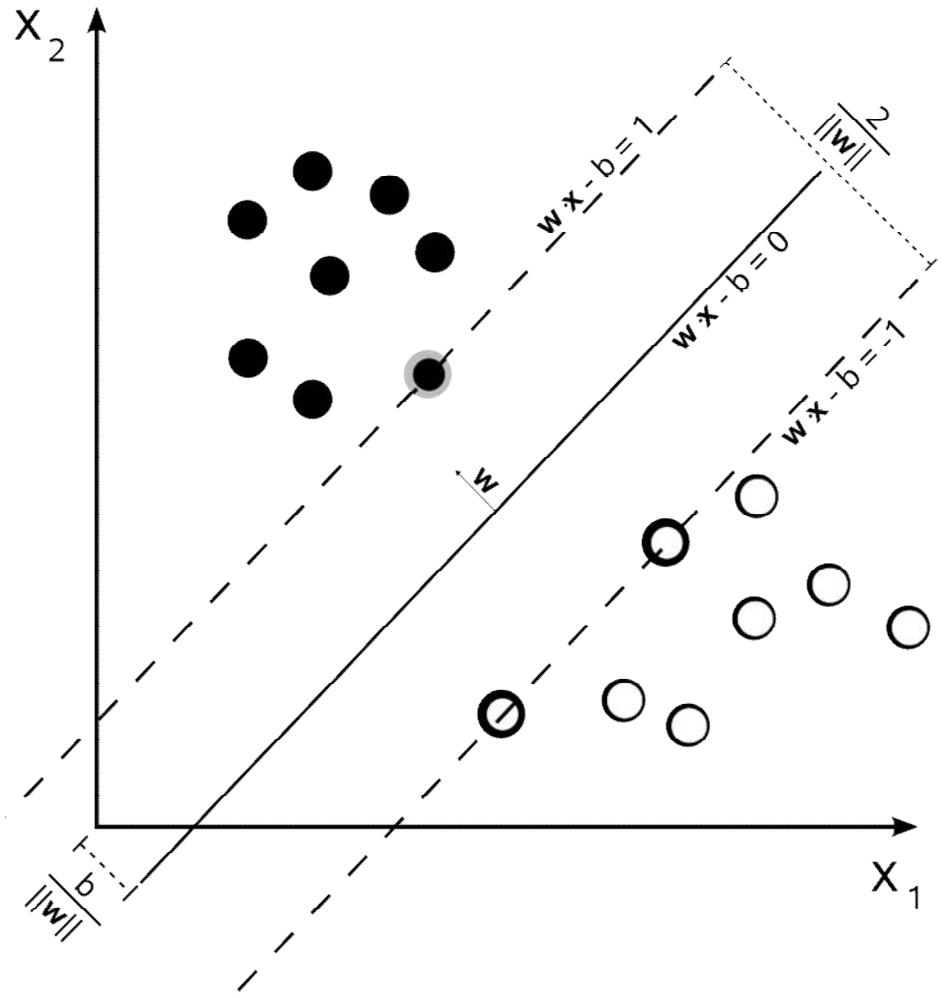
◈ 受众定向和筛选

- 个性化推荐：CF, ALS, etc.
- 人群转化率预估：Decision Tree, Random Forest, etc.

◈ 商品库和商品分类

- E.g. 海尔（Haier）BCD-190TMPK 190升 两门冰箱 -- 家用电器/大家电/冰箱
- 6000w+ 商品
- 2000+ 类别
- Train SVM on Spark

SVM的并行实现



$$\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{N}\xi_i$$

$$s.t. \quad \xi_i \geq 0, y_i f(\mathbf{x}_i) \geq 1 - \xi_i$$

SVM的并行实现

原问题

$$\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{N}\xi_i$$
$$s.t. \quad \xi_i \geq 0, y_i f(\mathbf{x}_i) \geq 1 - \xi_i$$

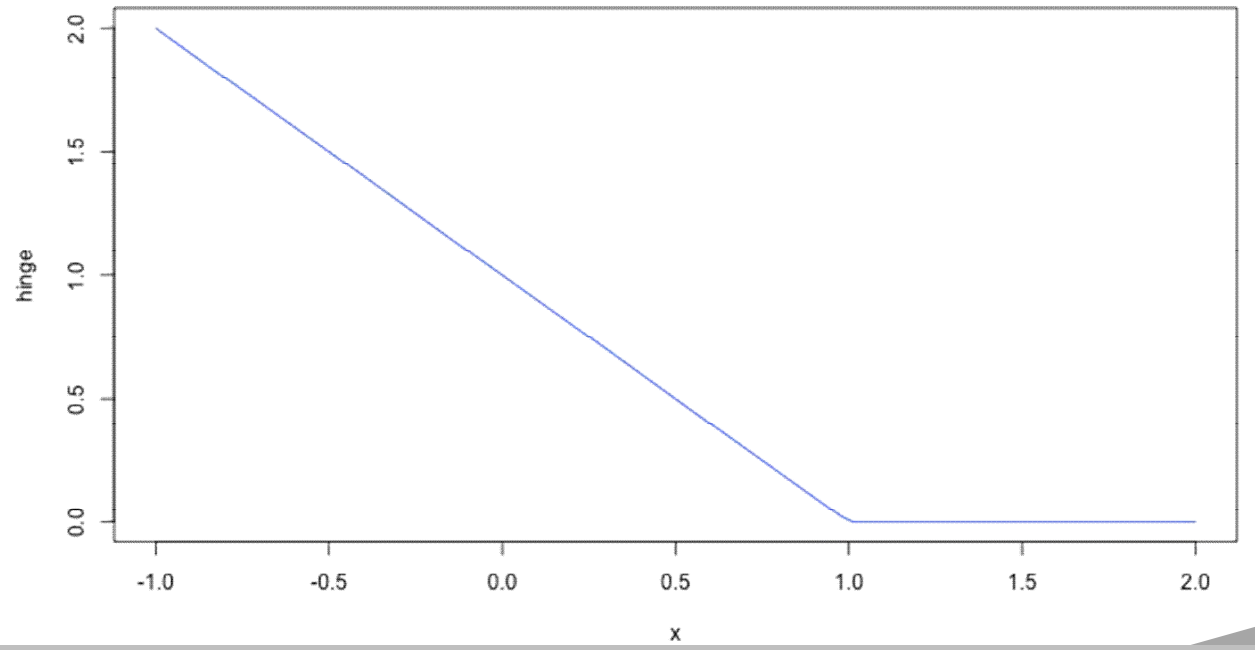
Hinge Loss

$$\min_{\mathbf{w}} \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{N}\max[0, 1 - y_i f(\mathbf{x}_i)]$$

SVM的并行实现

Hinge Loss

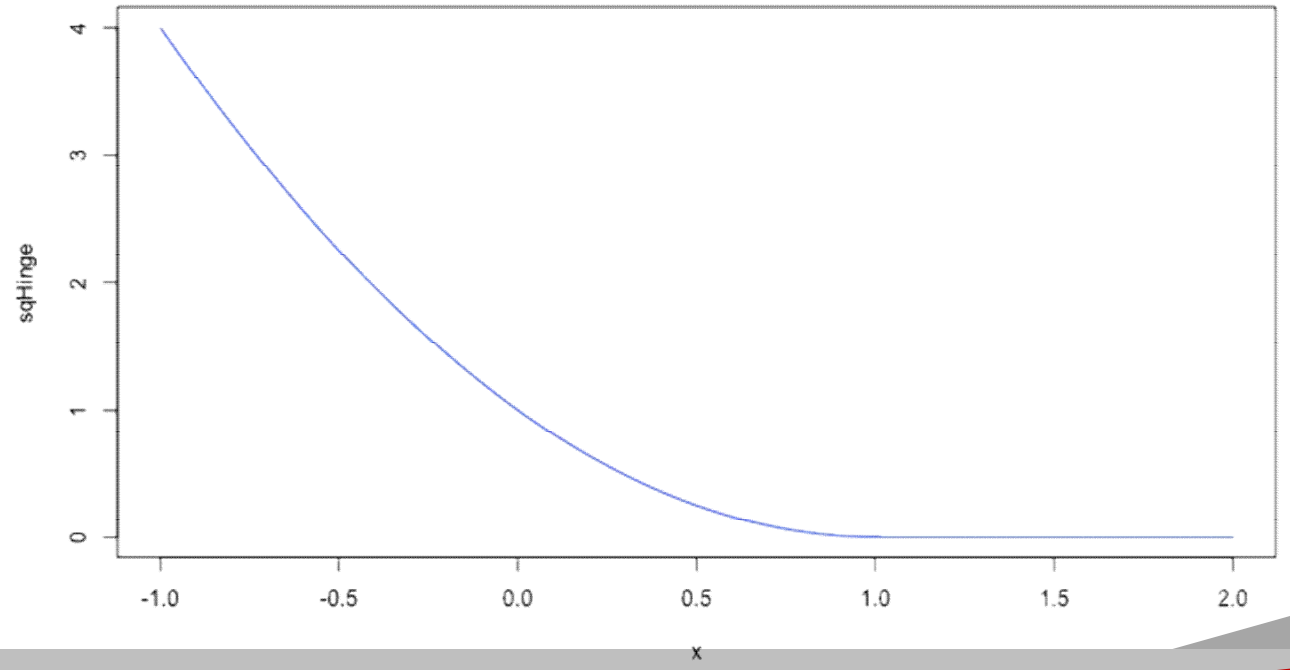
$$\min_{\mathbf{w}} \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{N} \max[0, 1 - y_i f(\mathbf{x}_i)]$$



SVM的并行实现

Squared Hinge Loss

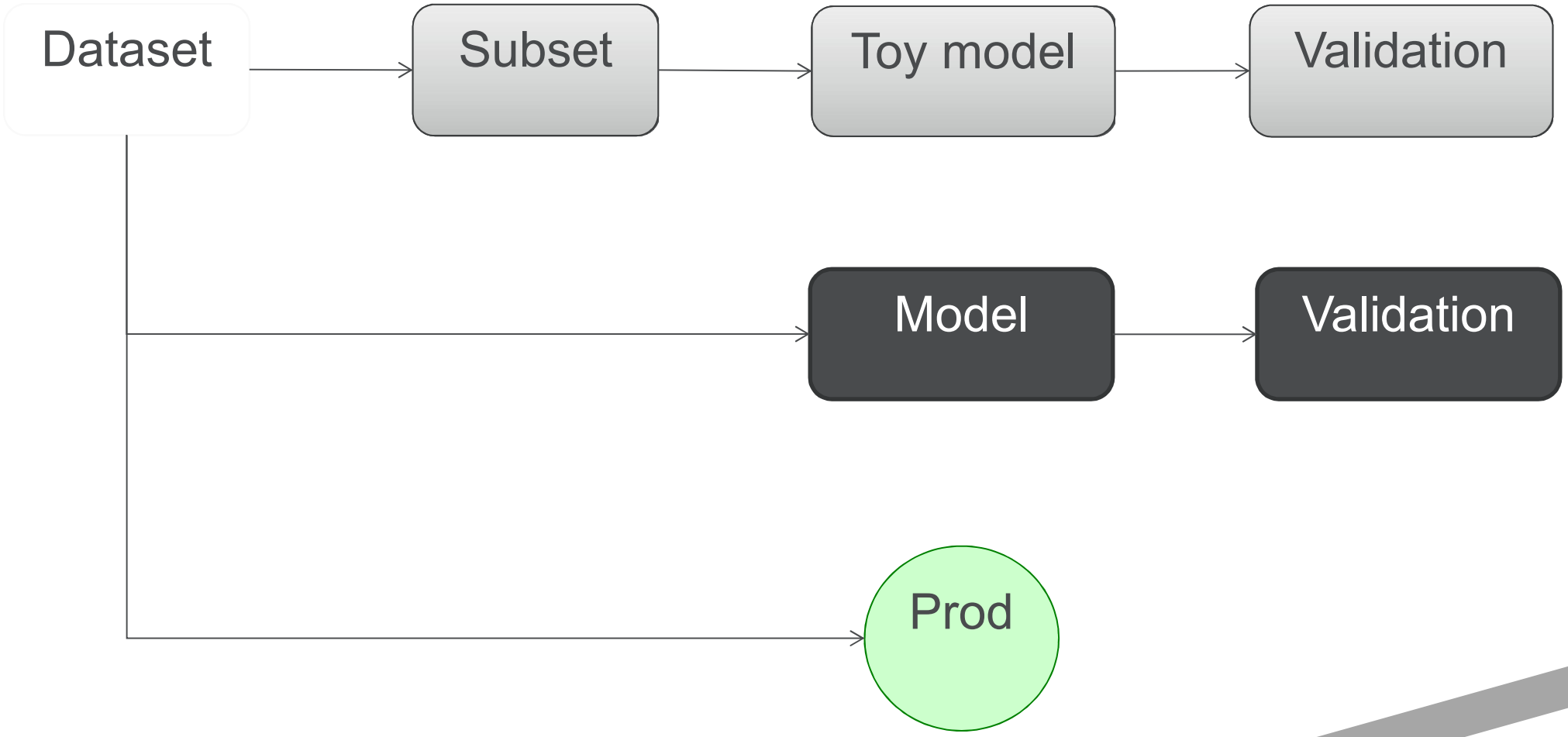
$$\min_{\mathbf{w}} \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{N} \max[0, 1 - y_i f(\mathbf{x}_i)]^2$$



SVM的并行实现

```
val scaler = (new StandardScaler).fit(input.map(x => x.features))
val data = input.map(labeledPoint =>
            (labeledPoint.label,
            appendBias(scaler.transform(labeledPoint.features))))

var weights = LBFGS.runLBFGS(
            data,
            new SquaredHingeLoss(),
            new SquaredL2Updater(),
            numCorrections,
            convergenceTol,
            maxIter,
            regParam,
            initialWeightsWithIntercept)

weights = scaler.transform(weights)
```

ML Pipeline


Dataset
Subset
Toy model
Validation
Model
Validation
Prod

ML Pipeline – Scikit Learn

```
from sklearn.pipeline import Pipeline
text_clf = Pipeline([
                        ('vect', CountVectorizer()),
                        ('tfidf', TfidfTransformer()),
                        ('clf', MultinomialNB())])

from sklearn import cross_validation
scores = cross_validation.cross_val_score(text_clf, X, y, cv=5)
```

ML Pipeline – Spark ML

```
val tokenizer = new Tokenizer()
val hashingTF = new HashingTF()
val lr = new LogisticRegression()

val pipeline = new Pipeline()
  .setStages(Array(tokenizer, hashingTF, lr))
```

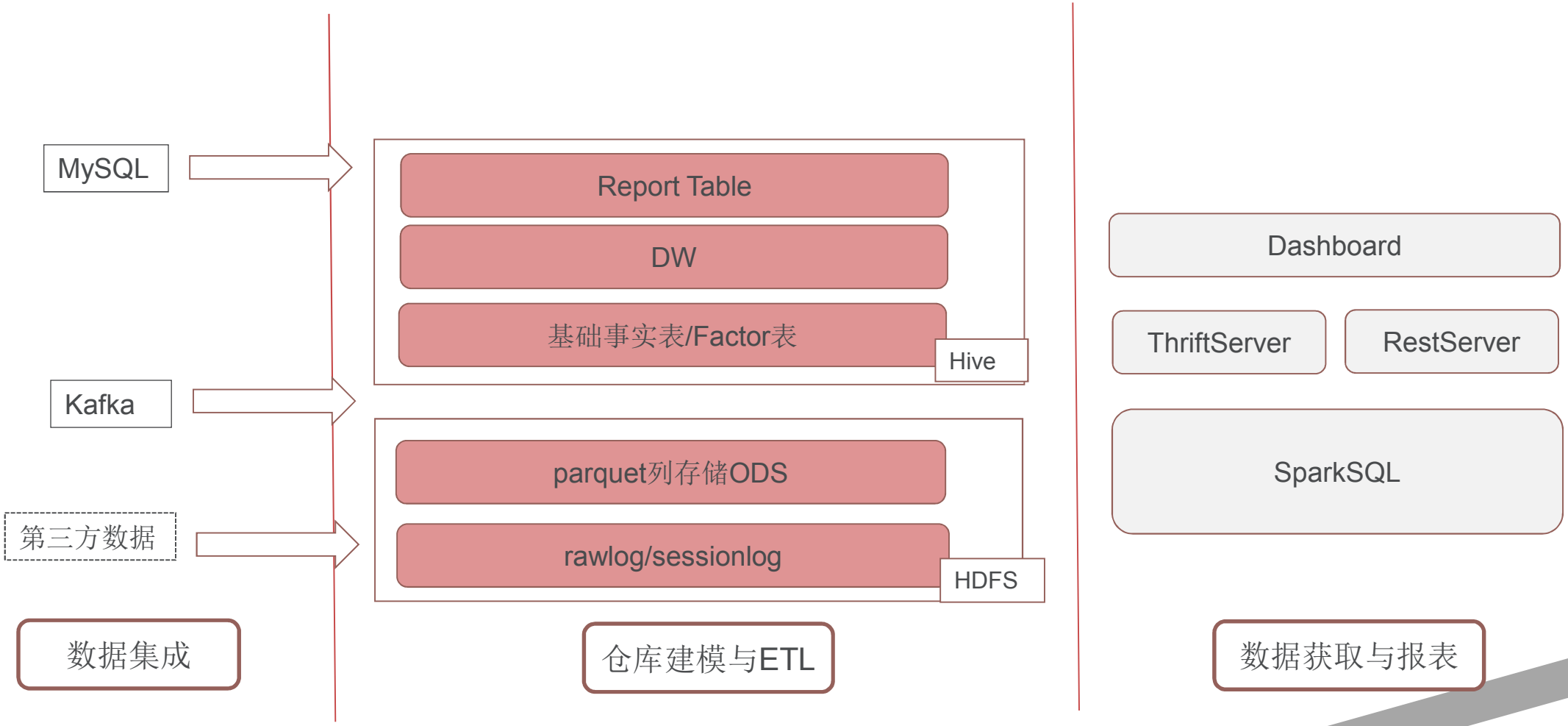
ML Pipeline – Spark ML

```
val paramGrid = new ParamGridBuilder()
  .addGrid(hashingTF.numFeatures, Array(10, 20, 40))
  .addGrid(lr.regParam, Array(0.01, 0.1, 1.0))
  .build()

val cv = new CrossValidator()
  .setNumFolds(3)
  .setEstimator(pipeline)
  .setEstimatorParamMaps(paramGrid)
  .setEvaluator(new BinaryClassificationEvaluator)

val cvModel = cv.fit(trainingDataset)
```

ML On Spark

- 满足常用的算法需求
- 快速迭代
  - 内存计算
  - 多样灵活的语言前端
- 端到端的解决方案
  - 和其他机器学习框架的整合
- 模型可用只是成功的第一步
  - 线上线下交互
  - 模型 / 业务指标

MVAD数据仓库架构


MySQL
Kafka
第三方数据
Report Table
DW
基础事实表/Factor表
Hive
parquet列存储ODS
rawlog/sessionlog
HDFS
Dashboard
ThriftServer
RestServer
SparkSQL
数据集成
仓库建模与ETL
数据获取与报表

Why SparkSQL?

◈ 最原始诉求:

- 希望能找到一个比Hive更快的查询引擎（结构化仓库数据）
- 简单高效的方式处理我们的ODS日志（半结构化日志）
- 兼容目前的处理方式, 减少系统迁移成本

◈ 可选:

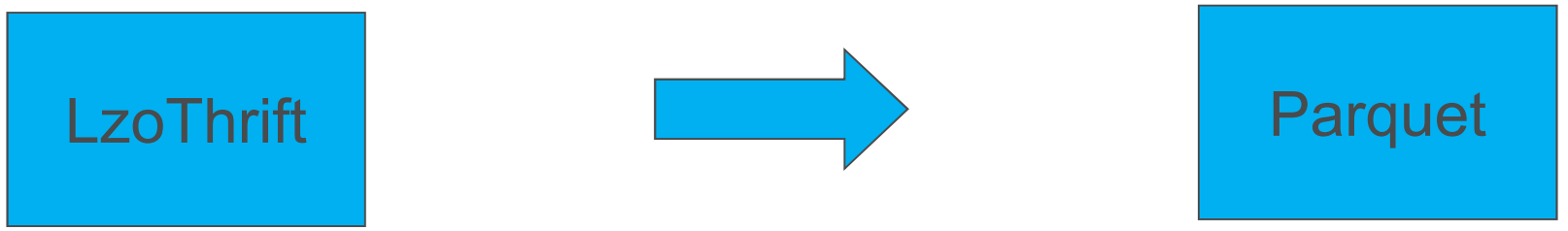
- impala
- presto
- hive on tez
- hive on spark
- SparkSQL

只能处理结构化仓库数据,
需要所有数据做结构化仓库建模

Why SparkSQL?

- 计算效率:
  - 天然DAG, 减少Job间启动的开销
  - 内存, Cache
- 兼容现有的Hadoop生态系统
  - HDFS、Hive、HBase、Cassandra .....
- 统一的数据处理栈
  - SQL、Streaming、ML、ETL
- on Yarn水平扩展, 部署升级简单
  - yarn-client、yarn-cluster mode
  - only a jar
- 活跃的社区:
  - Spark社区非常活跃, 享受社区"红利"

ODS: 半结构化日志Parquet化



- Thrift做Schema约束并序列化
- Lzo行级别压缩

- 自描述, Schema保持一致
- 列存储并Lzo做Chunk级别压缩
- 层次结构,利用Partition Discovery

```
MVAD: [zhugb@adm1ss] ~$ hadoop fs -ls /mvad/warehouse/ods/dsp/date=2016-03-30/hour=09/type=d.b
Found 2 items
drwxr-xr-x   - hadoop yarn          0 2016-03-30 10:44 /mvad/warehouse/ods/dsp/date=2016-03-30/hour=09/type=d.b/device=mobile
drwxr-xr-x   - hadoop yarn          0 2016-03-30 10:43 /mvad/warehouse/ods/dsp/date=2016-03-30/hour=09/type=d.b/device=pc
```

ODS: Pig -> DataFrame

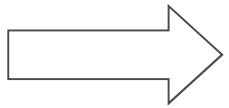
```
dsplog = LOAD '/mvad/warehouse/ods/dsp/date=2015-09-01/hour=00/type=*/*' USING
parquet.pig.ParquetLoader();

A= FOREACH dsplog GENERATE request.geo.province, request.userAgentInfo.os,
request.userAgentInfo.browser;

B = GROUP A BY province, os, browser;

C = FOREACH B GENERATE group, COUNT(A);

STORE C INTO '/tmp/xxx';
```

```
// Create a DataFrame from Parquet files
val df = sqlContext.read.parquet("/mvad/warehouse/ods/dsp/date=2015-
09-01/hour=00/type=*")

// Using DataFrame API
val result =
df.select("request.geo.province","request.userAgentInfo.os","request.user
AgentInfo.browser").groupBy("province", "os", "browser").count()

result.show()

result.write.format("parquet").mode("overwrite").save("/tmp/dsprequest")

---------------------------------------------------------------------------------------

// Using SparkSQL
val table = df.registerTempTable("dsplog")
val sql = " select count(1) from dsplog group by request.geo.province,
request.userAgentInfo.os, request.userAgentInfo.browser"
val result1 = sqlContext.sql(sql)
result1.show()
```

Hive: HQL -> SparkSQL

- hive -f xxx.hql

```
MVAD: [zhugb@az4ss] ~$ hive
16/03/21 14:07:44 WARN conf.HiveConf: HiveConf of name h

Logging initialized using configuration in file:/etc/hiv
WARNING: Hive CLI is deprecated and migration to Beeline
hive> show databases;
OK
bi
default
mediav_base
mvdw
```



- spark-sql -f xxx.hql

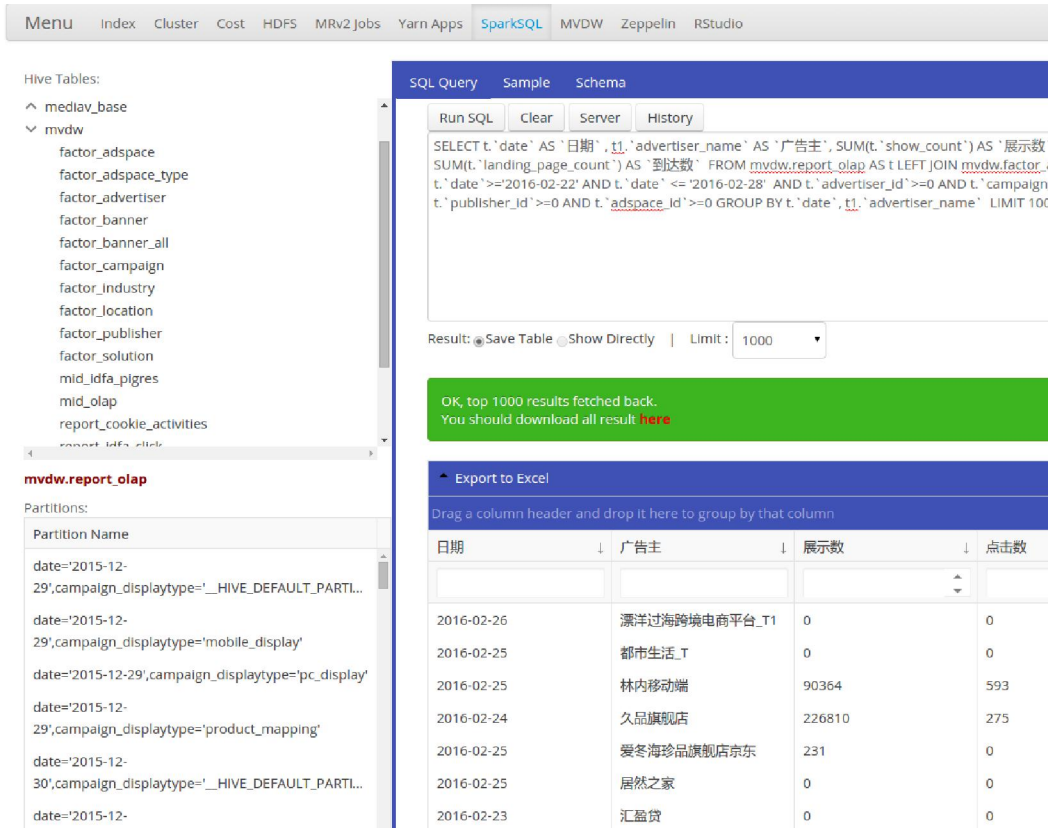
```
MVAD: [zhugb@az4ss] ~$ spark-sql
SET spark.sql.hive.metastore.version=1.1.0
SET spark.sql.shuffle.partitions=200
SET spark.sql.parquet.compression.codec=lzo
SET spark.sql.autoBroadcastJoinThreshold=104857600
SET spark.sql.parquet.cacheMetadata=true
SET spark.sql.parquet.useDataSourceApi=true
SET spark.sql.codegen=true
SET spark.sql.tungsten.enabled=true
SET spark.sql.parquet.filterPushdown=true
SET spark.sql.hive.convertMetastoreParquet=true
SET spark.sql.inMemoryColumnarStorage.batchSize=1000
SET spark.sql.inMemoryColumnarStorage.compressed=true
SET spark.sql.hive.metastore.jars=/etc/hive/conf:/etc/hadoop/
r/lib/hadoop-hdfs/.//*:/usr/lib/hadoop-yarn/lib/*:/usr/lib/ha
SET spark.sql.parquet.binaryAsString=true
SET hive.support.sql11.reserved.keywords=false
SET spark.sql.hive.version=1.2.1
SET spark.sql.hive.version=1.2.1
spark-sql> show databases;
OK
bi
default
mediav_base
mvdw
```

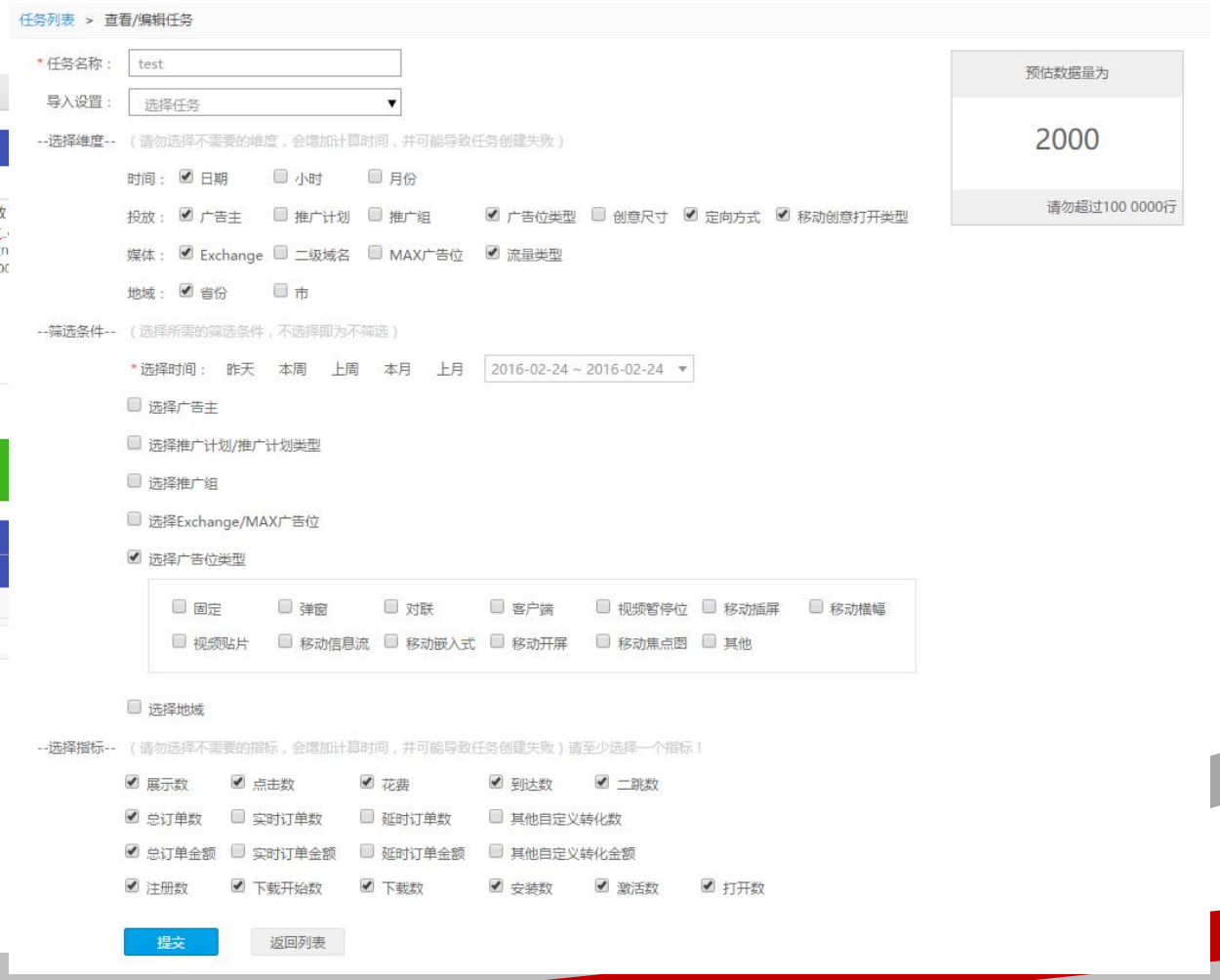
- HiveServer
- SparkSQLThriftServer

Web化简单易用

多维分析报表：

SparkSQLWeb：





成效与反馈

◈ 平台与系统不断优化

- 日志列存储化(Parquet)与压缩，节省存储资源，提高计算效率(2~3X)
- Hive升级至0.13，存储与计算性能提升(1Hour -> 20 Min)
- SparkSQL追随社区，享用最新的优化(1.5.2)(<5 Min)

◈ 降低数据获取成本

- SparkSQLWeb
- 多维分析

极大方便了产品、运营、优化、BI同学

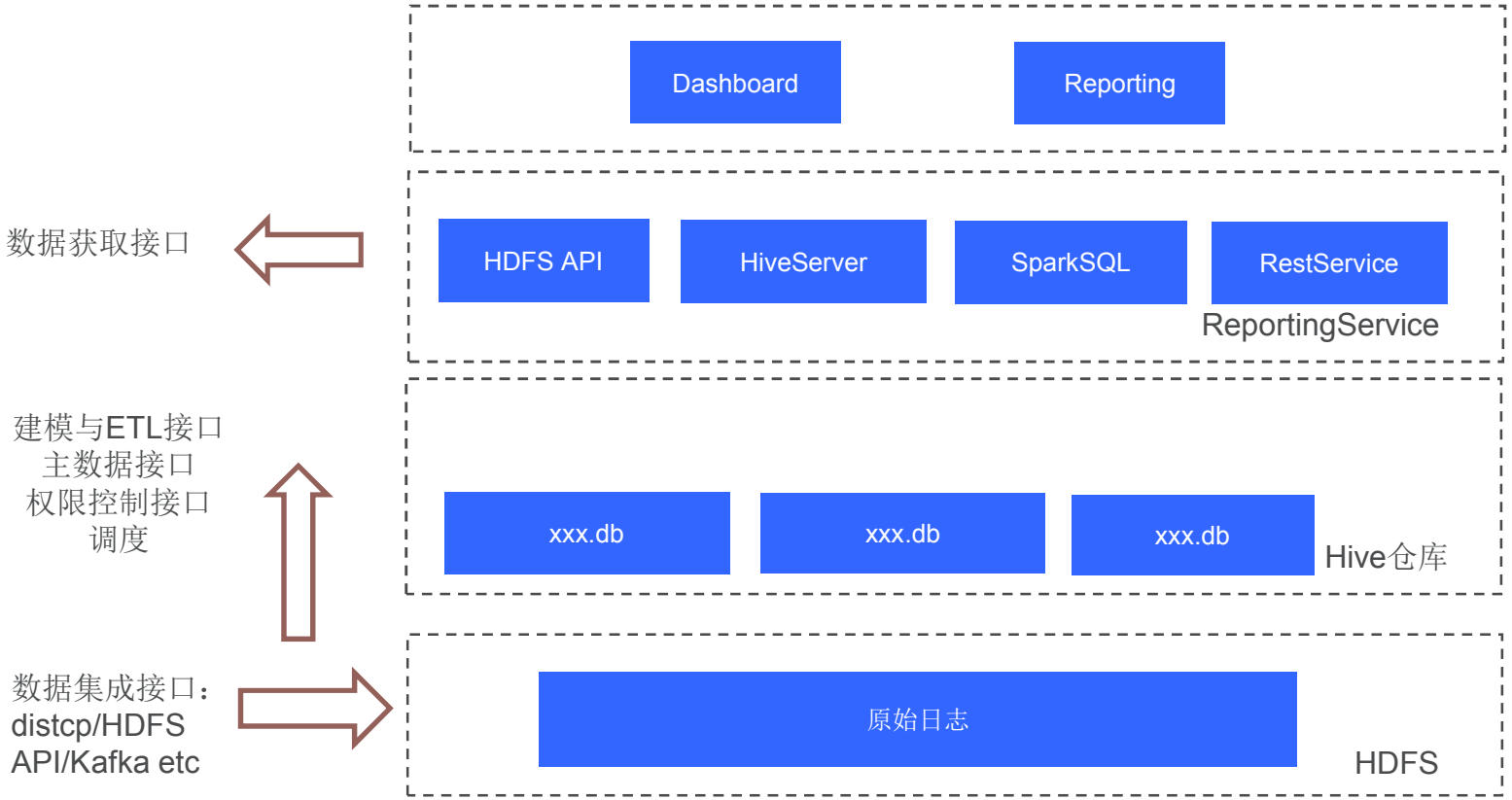
后续规划

◈ 平台化

- 统一所有数据入仓库
- 统一与规范Hive ETL、SparkSQL报表使用流程
- 避免繁冗复杂的ETL计算，统一平台优化，使得处理和获取数据更简单

◈ 服务化

- 每一个步骤与模块都提供外部接口
- 在支撑自由业务的基础上，提供对外服务：数据集成接口，ETL流程，报表数据获取API等


Dashboard
Reporting
数据获取接口
HDFS API
HiveServer
SparkSQL
RestService
ReportingService
建模与ETL接口
主数据接口
权限控制接口
调度
xxx.db
xxx.db
xxx.db
Hive仓库
数据集成接口：
distcp/HDFS
API/Kafka etc
原始日志
HDFS

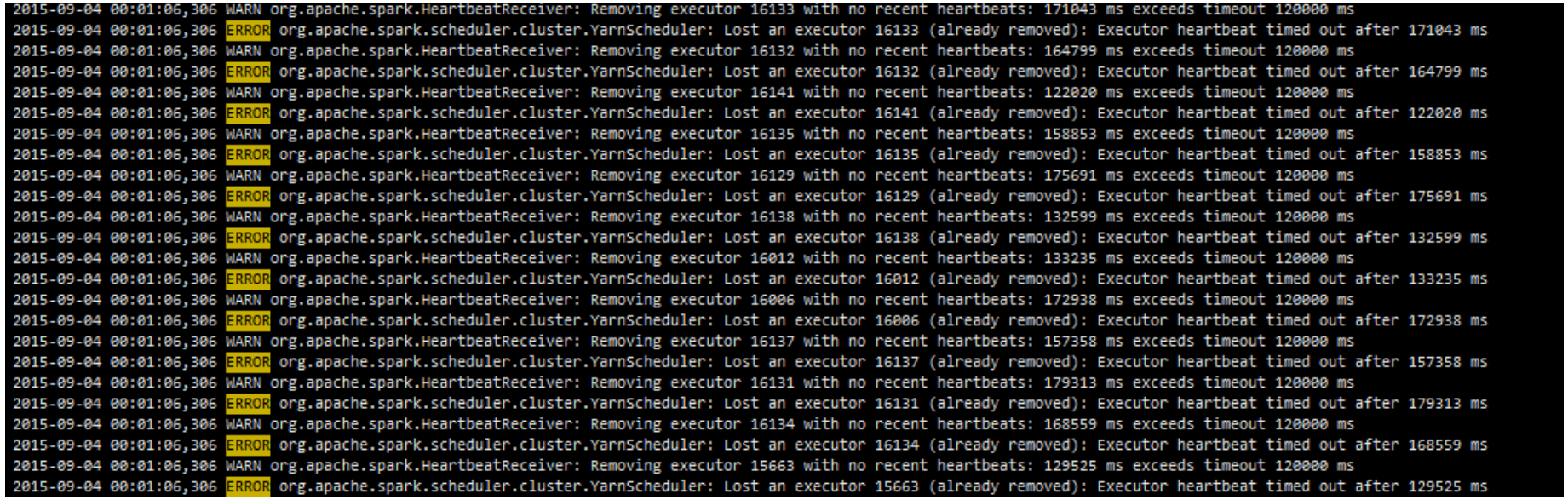
Spark 调优

- Based on 1.5.2
- 调优的前提是监控(发现问题)
- 对源码和原理有一定的了解(解决问题)
- 跟进社区,了解社区进展

如何监控Spark

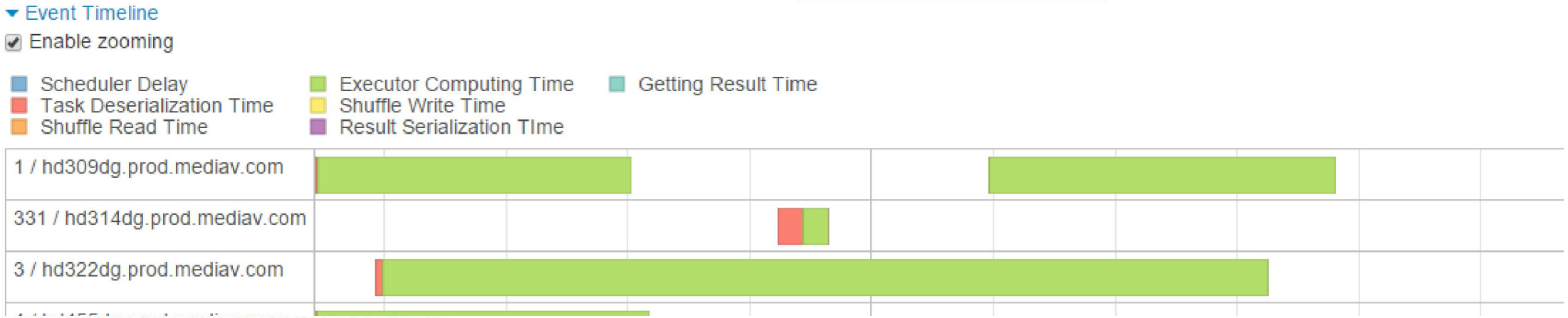
◈ Log

- Driver Log
- Appmaster Log
- Executor Log



如何监控Spark

◈ WebUI



Summary Metrics for 555 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	0 ms	21 s	26 s	34 s	1.3 min
Scheduler Delay	8 ms	19 ms	35 ms	50 ms	1.5 min
Task Deserialization Time	0 ms	26 ms	91 ms	0.2 s	5 s
GC Time	0 ms	0 ms	0 ms	0.1 s	2 s
Result Serialization Time	0 ms	0 ms	0 ms	0 ms	3 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Input Size / Records	0.0 B / 22	50.6 MB / 10529027	61.0 MB / 11163578	63.1 MB / 11617720	68.8 MB / 17947649
Shuffle Write Size / Records	0.0 B / 0	14.0 B / 1	14.0 B / 1	14.0 B / 1	14.0 B / 1

GC is always a pain

- Monitor GC

Tasks

Index	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	Scheduler Delay	Task Deserialization Time	GC Time	Result Serialization Time	Getting Result Time	Input Size / Records	Write Time	Shuffle Write Size / Records
254	1375	0	SUCCESS	NODE_LOCAL	1 / hd309dg.prod.mediav.com	2015/09/04 12:32:14	26 s	37 ms	0.2 s	0.1 s	0 ms	0 ms	63.6 MB (hadoop) / 11704623		14.0 B / 1
300	1374	0	SUCCESS	NODE_LOCAL	181 / hd210dg.prod.mediav.com	2015/09/04 12:32:14	6 s	37 ms	65 ms		0 ms	0 ms	13.9 MB (hadoop) / 2416074		14.0 B / 1
40	1376	0	SUCCESS	NODE_LOCAL	136 / hd471dg.prod.mediav.com	2015/09/04 12:32:14	26 s	38 ms	0.1 s	0.2 s	0 ms	0 ms	61.3 MB (hadoop) / 11077822		14.0 B / 1
129	1378	0	SUCCESS	NODE_LOCAL	280 / hd463dg.prod.mediav.com	2015/09/04 12:32:14	0.5 s	40 ms	0.2 s		0 ms	0 ms	158.3 KB (hadoop) / 13737	1 ms	14.0 B / 1
321	1381	0	SUCCESS	NODE_LOCAL	105 / hd341dg.prod.mediav.com	2015/09/04 12:32:14	21 s	39 ms	94 ms		0 ms	0 ms	60.6 MB (hadoop) / 10969869		14.0 B / 1
358	1380	0	SUCCESS	NODE_LOCAL	297 / hd463dg.prod.mediav.com	2015/09/04 12:32:14	34 s	36 ms	0.1 s	26 ms	0 ms	0 ms	60.7 MB (hadoop) / 17824209		14.0 B / 1

GC is always a pain

◈ JVM & GC Tuning

```
spark.yarn.am.memory                   3G
spark.yarn.am.memoryOverhead           1024
spark.yarn.am.extraJavaOptions         -Xmn1G -XX:+UseG1GC
spark.yarn.am.extraLibraryPath         /usr/lib/hadoop/lib/native
spark.driver.memory                    3G
spark.driver.extraJavaOptions          -Xmn1G -XX:+UseG1GC -XX:+PrintGC -XX:+PrintGCDetails -
Xloggc:/tmp/spark.driver.gc.log -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/tmp
spark.driver.extraLibraryPath          /usr/lib/hadoop/lib/native
spark.executor.memory                  3G
spark.yarn.executor.memoryOverhead   1024
spark.executor.extraJavaOptions      -Xmn1G -XX:+UseG1GC
spark.executor.extraLibraryPath        /usr/lib/hadoop/lib/native
spark.executor.extraJavaOptions      -Xmn1G -XX:+UseG1GC
```

控制Task并行度

◈ parallel more. then faster

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
27	default	select advertiser_id,solution_id,dmplabels as dp_label,sum(valid_show_count) as daily_numofshow,sum(valid_clic... Spark JDBC Server Query +details	2015/09/16 10:03:27	10 s	200/200			98.8 KB	
26	default	select advertiser_id,solution_id,dmplabels as dp_label,sum(valid_show_count) as daily_numofshow,sum(valid_clic... Spark JDBC Server Query +details	2015/09/16 10:00:15	3.2 min	1385/1385	341.9 GB			92.0 KB

◈ 两种Task：

ShuffleMapTask并行度：

- 数据层优化：降低BlockSize，降低Parquet/ORC StripeSize
- 应用层优化：

```
val rdd = sc.textFile("hdfs://...",minPartitions)
val rdd = sc.hadoopFile[keyClass, valueClass, InputFormat]("hdfs://...", minPartitions)
rdd.repartition(numPartitions)
rdd.coalesce(numPartitions, true)
```

ResultTask并行度：

- spark.default.parallelism
- spark.sql.shuffle.partitions

```
override def defaultParallelism(): Int = {
  conf.getInt("spark.default.parallelism", math.max(totalCoreCount.get(), 2))
}
```

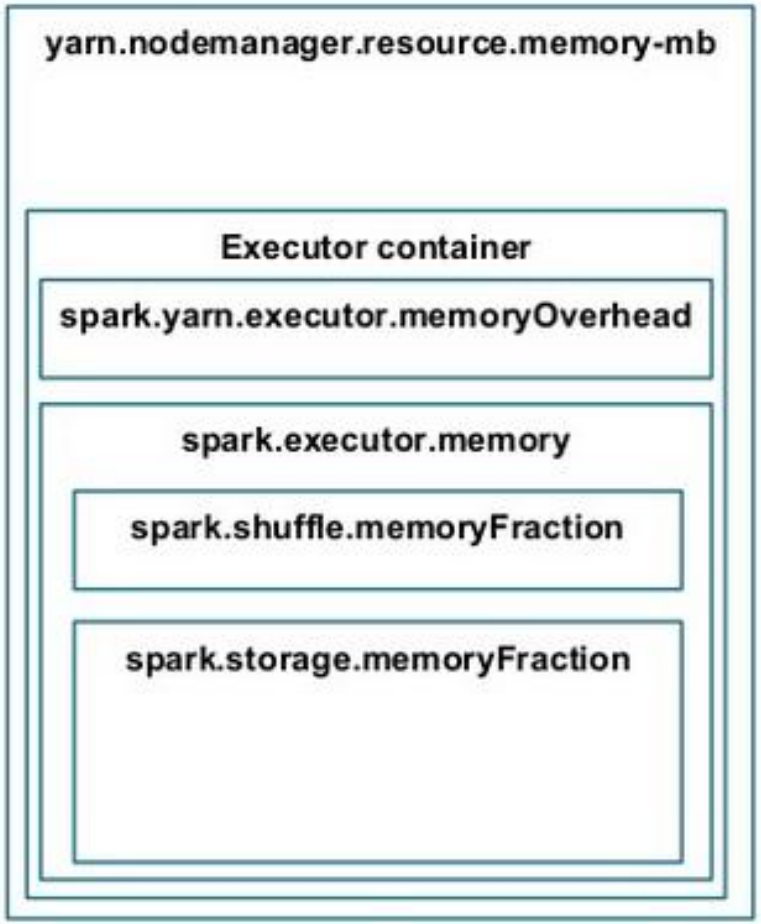
```
/** Number of partitions to use for shuffle operators. */
private[spark] def numShufflePartitions: Int = getConf(SHUFFLE_PARTITIONS, "200").toInt
```

```
rdd.groupByKey(numPartitions)
rdd.reduceByKey(_func, numPartitions)
rdd.join(other, numPartitions)
```

配置参数优化

◈ 资源和内存分配

```
spark-submit --num-executors 10 \
--executor-cores 2 \
--executor-memory 3G \
--driver-cores  2  \
--driver-memory  3G
```



配置参数优化

◈ 序列化

spark.closure.serializer	org.apache.spark.serializer.JavaSerializer
spark.serializer	org.apache.spark.serializer.KryoSerializer

◈ 压缩

spark.broadcast.compress	true
spark.rdd.compress	true
spark.io.compression.codec	org.apache.spark.io.LZFCompressionCodec

◈ Speculation & FairScheudling

spark.scheduler.mode	FAIR
spark.speculation	true
spark.task.cpus	1
spark.task.maxFailures	8
spark.yarn.report.interval	1000
spark.scheduler.maxRegisteredResourcesWaitingTime	3s

Dynamic Allocation

- allow Spark application to scale the number of executors up and down based on workload
- if executors are idle, remove them.
- if we need more executors, request them.

```
spark.shuffle.service.enabled                   true
spark.dynamicAllocation.enabled                 true
spark.dynamicAllocation.initialExecutors        10
spark.dynamicAllocation.minExecutors            10
spark.dynamicAllocation.maxExecutors            500
spark.dynamicAllocation.schedulerBacklogTimeout  5
spark.dynamicAllocation.sustainedSchedulerBacklogTimeout 5
spark.dynamicAllocation.executorIdleTimeout     120
```

automatically determine executors based on workload

spark-submit --master yarn-client --num-executors 100 xxx

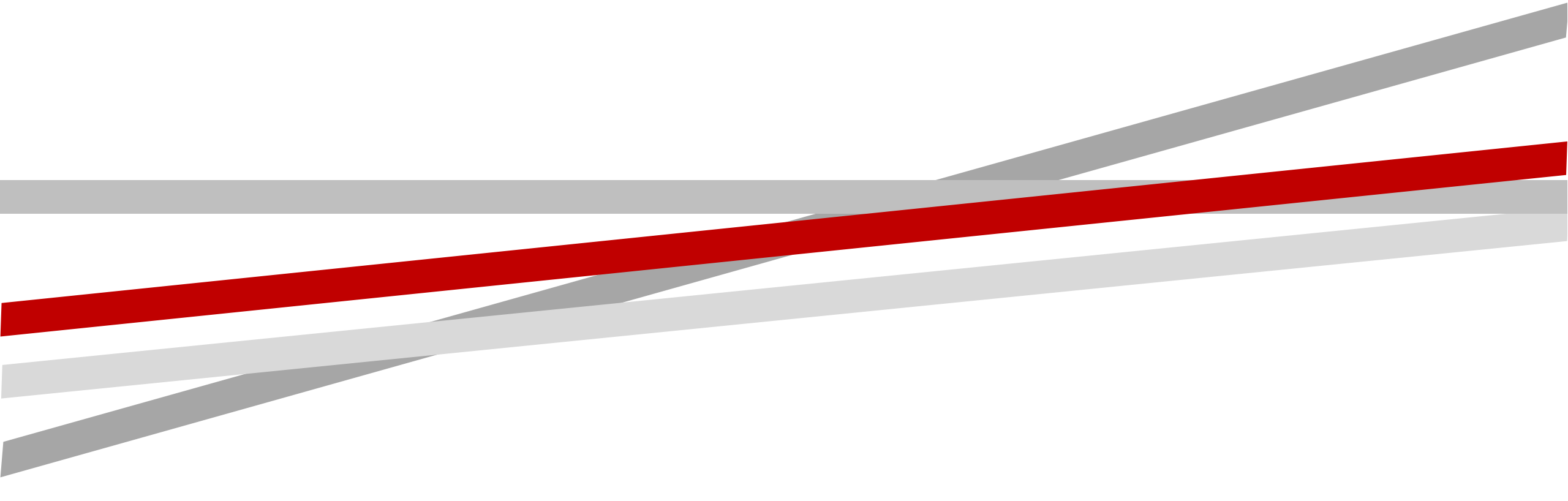
spark-submit --master yarn-client xxx

其他配置

- shuffle
- networking
- ……

- Spark is complicated
  - https://spark.apache.org/docs/latest/configuration.html

We are hiring ...



Q&A Time

Thanks