InfoQ–揭秘Apache Spark 3.0 新特性在FreeWheel核心业务数据团队的应用与实战

新特性

Hanke

2021-01-04

## 引言

相信作为Spark的粉丝或者平时工作与Spark相关的同学大多知道，Spark 3.0在2020年6月官方重磅发布，并于9月发布稳定线上版本，这是Spark有史以来最大的一次release，共包含了3400多个patches，而且恰逢Spark发布的第十年，具有非常重大的意义。

团队在Spark发布后，快速动手搭好Spark 3.0的裸机集群并在其上进行了初步的调研，发现相较于Spark 2.x 确实有性能上的提升。于是跟AWS EMR和Support团队进行了多次沟通表达我们的迫切需求后，EMR团队给予了快速的响应，在11月底发布了内测版本。作为第一批内测用户，我们做了Data Pipelines上各个模块的升级，测试和数据验证。团队通过高效的敏捷开发赶在2020年圣诞广告季之前在生产环境顺利发布上线，整体**性能提升高达40%**（对于大batch）的数据，**AWS Cost平均节省25%~30%之间**，大约每年至少能为公司节省百万成本。目前线上稳定运行，预期借助此次升级能够更从容为FreeWheel高速增长业务量和数据分析需求保驾护航。

在这次Spark 3.0的升级中，其实并不是一个简简单单的版本更换，因为团队的Data Pipelines所依赖的生态圈本质上其实也发生了一个很大的变化。比如EMR有一个大版本的升级，从5.26升级到最新版6.2.0，底层的Hadoop也从2.x升级到3.2.1，Scala只能支持2.12等等。本篇文章主要是想和大家分享一下Spark 3.0在FreeWheel大数据团队升级背后的故(xuè )事(lèi)和相关的实战经验，希望能对大家以后的使用Spark 3.0特别是基于AWS EMR上开发有所帮助，可以在Spark升级的道路上走的更顺一些。

## 团队介绍

FreeWheel核心业务数据团队的主要工作是通过收集，分析来自用户的视频广告数据，来帮助客户更好地制定广告计划，满足客户不断增长的业务需求，最终帮助客户实现业务的增长。其中最主要的两类数据分别是预测数据和历史数据: \* **预测数据**会根据用户历史广告投放情况进行算法分析和学习来得到未来预测情况，在此基础上向客户提供有价值的数据分析结果，比如广告投放是否健康，广告位是否足够，当前的广告售卖是否合理等等信息。通过这些数据分析的反馈可以帮助用户更好地在广告定价、售期等方面做出正确的决定，最终达到自己的销售目标。 \* **历史数据**主要是提供用户业务场景数据分析所需要的功能，比如数据查询，Billing账单，广告投放情况，市场策略等，并且通过大量的历史数据从多维度多指标的角度提供强有力的BI分析能力进而帮助用户洞察数据发生的变化，发现潜在的问题和市场机会。

作为核心业务数据团队里重要的成员，**Transformer**团队的主要负责: \* **基于大数据平台技术建立Data Pipelines** \* 负责将交易级别的数据转化为分析级别的数据，服务下游所有的数据产品 \* **构建统一的数据仓库** \* 通过分层业务模型来构建所有数据产品不同场景下（历史或者预测）使用一致的业务视图和指标 \* 提供不同粒度或者维度的聚合事实数据 \* 提供基于特定场景的数据集市 \* **提供统一的数据发布服务和接口**

### 数据建模和Data Pipelines架构

当交易级别的广告（历史或者预测）数据进入系统后，会通过数据建模和Data Pipelines进行统一的建模或者分析，视业务需要更进一步构建数据集市，生成的聚合事实数据会被发布到数据仓库Hive和Clickhouse里供下游数据产品通过Presto或者Clickhouse查询引擎来消费。如下是整体建模和Data Pipelines的架构图： Team Intro

其中主要模块包括： \* **Optimus**  
\* 正如它的名字一样，Optimus同样是Transformer团队的模块中的领袖人物，肩负业务数据团队最重要的数据建模部分。通过分层数据建模的方式来构建统一的基于上下文的数据模型，保障所有下游产品在不同的应用和业务场景下的计算指标，计算逻辑一致，且避免来回重复计算扫描数据。比如预测数据和历史数据同样的指标含义，就使得提供给客户的数据对比更有说服力和决策指导意义。目前它会产生将近四十张左右的小时粒度的历史事实表和预测事实表。目前每天处理的数据在TB级别，会根据每个小时的数据量自动进行扩或者缩集群，保证任务的高性能同时达到资源的高效利用目标。 \* **JetFire**  
\* JetFire是一个基于Spark的通用ETL框架，支持用户通过SQL或者Code的方式灵活的定制ETL任务和分析数据任务。目前主要用于Post-Optimus的场景，生成基于特定业务场景更高聚合粒度的数据集市上。比如生成todate(迄今为止)的统计指标，像每个客户截止到目前或者过去18个月的广告投放总数。这样就可以避免每次查询对底层数据或者Optimus生成的聚合数据进行全扫。生成一次供多次查询，可以极大提高查询效率，降低成本。 \* **Publisher** \* 基于Spark的数据发布模块，负责将数据发布到数据仓库里。由于数据建模产生的数据按日期进行分区，当存在Late Data的时候，很容易生成碎小文件，Publisher通过发布数据前合并碎小文件的功能来提升下游的查询效率。 \* **Bumblebee** \* 主要是为数据建模和Data Pipelines的各个模块提供模块测试和集成测试环境，供业务开发的同学使用。此外，基于此提供所有Data Pipelines的整体一致的CD和灾备方案，保障在极端场景下系统的快速启动和恢复。 \* **Data Restatement** \* 除了日常的Data Pipelines，在客户数据投放出现问题或者数据仓库数据出现偏差遗漏时，需要自动修数据的Pipelines来支持大范围的数据修正和补偿。整体的作业调度需要保证日常工作正常完成的情况下，尽快完成数据修正工作。目前提供整个batch或者delta两种方式修数据，来满足不同的应用场景。 \* **Data Publish API** \* 负责为下游提供数据发布信息，来触发一些订阅的报表或者产品发布。

除了Data Publish API服务部署在EKS上，其他相关模块目前都运行在AWS EMR上，灵活使用Spot Instance和On Demand混合模式，高效利用资源。团队基于以上的模块为公司的业务发展提供有力的数据和技术保障。

## 实践成果

这次升级主要的实践成果如下： ### 性能提升明显 \* **历史数据**Pipeline对于大batch的数据（200~400G/每小时）性能提升高达40%， 对于小batch（小于100G/每小时）提升效果没有大batch提升的那么明显，每天所有batches平均提升水平27.5%左右。 \* **预测数据**性能平均提升30% \* 由于数据输入源不一样，目前是分别两个pipelines在跑历史和预测数据，产生的表的数目也不太一样，因此做了分别的评估。

以历史数据上线后的端到端到运行时间为例（如下图），肉眼可见上线后整体pipeline的运行时间有了明显的下降，能够更快的输出数据供下游使用。  
runtime

### 集群内存使用降低

集群内存使用对于大batch达降低30%左右，每天平均平均节省25%左右。  
以历史数据上线后的运行时集群的memory在ganglia上的截图为例（如下图），整体集群的内存使用从41.2T降到30.1T，这意味着我们可以用更少的机器花更少的钱来跑同样的Spark任务。  
memory

### AWS Cost降低

Pipelines做了自动的Scale In/Scale Out策略: 在需要资源的时候扩集群的Task结点，在任务结束后自动去缩集群的Task结点，且会根据每次batch数据的大小通过算法学习得到最佳的机器数。通过升级到Spark 3.0后，由于现在任务跑的更快并且需要的机器更少，上线后统计AWS Cost每天节省30%左右，一年能为公司百万成本。

如下是历史数据Pipeline上线后，通过AWS Billing得到的账单Cost数据，可以看到在使用 Spot Instance 情况下(花费柱状图较短的情况下)从上线前(蓝色线)到上线后(红色线)每天有显著的30%左右的成本下降，如果使用AWS On Demand的Instance的话那么节省就更可观了。 aws-cost

### 其他

* Data Pipelines里的所有的相关模块都完成了Spark 3.0的升级，享受最新技术栈和优化带来的收益。
* 由于任务运行时间和需要的机器数明显下降，整体的Spot Instance被中断的概率也大大降低，任务稳定性得到加强。
* 发布了自动化数据验证工具进行端到端的数据验证。
* 统一并升级了所有模块的CD Pipelines。

接下来我们具体看看我们做了什么，又踩了什么样的坑，以及背后有什么魔法帮助达到**既让任务跑得快又能为公司省钱的效果**。对Spark 3.0新特性感兴趣的同学可以参考我的另外一篇文章–关于[Spark 3.0的关键新特性回顾](https://xie.infoq.cn/draft/40812)。

## 我们做了什么？遇到什么坑？

Data Pipelines和相关的回归测试框架都进行相关依赖生态圈的统一升级，接下来会跟大家详细分享细节部分。

### Spark升级到最新稳定版3.0.1

Spark 3.0.1是社区目前推荐使用的最新的稳定版本，于2020年九月正式发布，其中解决了3.0版本里的一些潜在bug。

#### 主要的改动

* **打开Spark 3.0 AQE的新特性**  
  主要配置如下：

"spark.sql.adaptive.enabled": true,  
 "spark.sql.adaptive.coalescePartitions.enabled": true,  
 "spark.sql.adaptive.coalescePartitions.minPartitionNum": 1,  
 "spark.sql.adaptive.advisoryPartitionSizeInBytes": "128MB"

需要注意的是，AQE特性只是在reducer阶段不用指定reducer的个数，但**并不代表你不再需要指定任务的并行度了**。因为map阶段仍然需要将数据划分为合适的分区进行处理，如果没有指定并行度会使用默认的200，当数据量过大时，很容易出现OOM。建议还是按照任务之前的并行度设置来配置参数spark.sql.shuffle.partitions和spark.default.parallelism。

* **升级HyperLogLog相关的UDAF到新接口**  
  Spark 3.0提供了通过用户定制实现的Aggregator来注册实现UDAF，可以避免对每一行的数据反复进行序列化和反序列化来进行聚合，而只需在整个分区里序列化一次 ，缓解了对cpu的压力，提升性能。假如一个DataFrame有100万行数据共10个paritions，那么旧的UDAF方式的序列化反序列化需要至少100万+10次(合并分区里的结果)。 UDAF 而新的函数只需要10次即可，大大减少整体的序列化操作。
* **依赖Hadoop版本升级**  
  依赖的Hadoop根据Spark和EMR支持的版本升级到3.2.1

ext {  
 hadoopVersion = "3.2.1"  
}  
compile group: "org.apache.hadoop", name: "hadoop-client", version: "${hadoopVersion}"

* **打开History Server Event Logs滚动功能**  
  Spark 3.0提供了类似Log4j那样对于长时间运行的日志按照时间或者文件的大小进行切割，这样对于Streaming长期运行的任务和大任务来说比较友好。

"spark.eventLog.rolling.enabled": true,  
 "spark.eventLog.rolling.maxFileSize": "1024m",  
 "spark.eventLog.buffer.kb": "10m"

#### 遇到的坑

* **读Parquet文件失败**  
  升级到Spark 3.0后，读源数据Parquet文件会出现一些莫名的问题，有些文件可以正常解析，而有些文件则会抛出失败的异常错误，这个错误是整个升级的Blocker，非常令人苦恼。
  + **具体的错误信息**
* org.apache.spark.sql.execution.QueryExecutionException: Encounter error while reading parquet files. One possible cause: Parquet column cannot be converted in the corresponding files.
  + **原因**
    - 在仔细调试和阅读源码后发现，Spark 3.0在Parquet的嵌套schema的逻辑上做了修改，主要是关于使用的优化特性spark.sql.optimizer.nestedSchemaPruning.enabled时的变化，具体可以进一步阅读相关的[ticket](https://github.com/apache/spark/pull/24307)。
    - 而产生的影响就是当在有嵌套schema的Parquet文件上去读取不存在的field时，会抛出错误。而在2.4以前的版本是，是允许访问不存在的field并返回none，并不会中断整个程序。
  + **解决办法**
    - 由于我们数据建模和上游开发模式就是面向接口编程，为了不和schema严格绑定，是会存在提前读取一些暂时还没有上线的field并暂时存放空值。因此，新的逻辑修改直接就break了原来的开发模式， 而且代码里也要加入各种兼容老的schema逻辑。
    - 于是我们将优化spark.sql.optimizer.nestedSchemaPruning.enabled会关掉后，再进行性能的测试，发现性能的影响几乎可以忽略。
    - 鉴于上面的影响太大和性能测试结果，最终选择设置spark.sql.optimizer.nestedSchemaPruning.enabled = false。后续会进一步研究是否有更优雅的解决方式。
* **History Server的Connection Refused**  
  Spark 3.0里History Server在解析日志文件由于内存问题失败时， History Server会重启，随后会出现Connection Refused的错误信息，而在2.x里，并不会导致整个History Server的重启。
  + **解决方案**  
    增加History Server的内存。
    - 在Master结点, Spark配置文件里修改：
  + export SPARK\_DAEMON\_MEMORY=12g
    - 然后重启History Server即可 sudo systemctl restart spark-history-server
* **History UI显示任务无法结束**
  + **原因**
    - 打开AQE后由于会对整个查询进行再次切分，加上3.0也会增加很多相关Observable的指标，比如Shuffle，所以整体的History Logs会变的相对较大，目前对于某些batch的任务产生的logs无法及时同步到History Server里，导致从History UI去看任务执行进度时会存在一直在in progress状态，但实际上任务已经执行完毕。
    - 在阅读源码和相关Log后，比较怀疑是Spark Driver在eventLoggingListerner向升级后的HDFS(Hadoop 3.2.1)写eventlogs时出了什么问题，比如丢了对应事件结束的通知信息。由于源码里这部分debugging相关的Log信息相对有限，还不能完全确定根本原因，后续会再继续跟进这个问题。 > 其实类似的问题在Spark 2.4也偶有发生，但升级到3.0后似乎问题变得频率高了一些。遇到类似问题的同学可以注意一下，虽然Logs信息不全，但**任务的执行和最终产生的数据都是正确的**。
* **HDFS升级后端口发生变化**  
  端口号变化列表:
  + Namenode 端口: 50470 –> 9871, 50070 –> 9870, 8020 –> 9820
  + Secondary NN 端口: 50091 –> 9869, 50090 –> 9868
  + Datanode 端口: 50020 –> 9867, 50010 –> 9866, 50475 –> 9865, 50075 –> 9864

### EMR升级到最新版6.2.0

* **系统升级**  
  EMR 6.2.0使用的操作系统是更好Amazon Linux2，整体系统的服务安装和控制从直接调用各个服务自己的起停命令(原有的操作系统版本过低)更换为统一的Systemd。
* **启用Yarn的结点标签**  
  在EMR的6.x的发布里，禁用了Yarn的结点标签功能，相较于原来Driver强制只能跑在Core结点上，新的EMR里Driver可以跑在做任意结点，细节可以参考[文档](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-master-core-task-nodes.html)。  
  而由于我们的Data Pipelines需要EMR的Task节点按需进行扩或者缩，而且用的还是Spot Instance。因此这种场景下Driver更适合跑在常驻的(On Demand)的Core结点上，而不是随时面临收回的Task节点上。对应的EMR集群改动：

yarn.node-labels.enabled: true  
yarn.node-labels.am.default-node-label-expression: 'CORE'

* **Spark Submit 命令的修改**  
  在EMR新的版本里用extraJavaOptions会报错，这个和EMR内部的设置有关系，具体详情可以参考[EMR配置](https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-spark-configure.html) ，修改如下:  
  spark.executor.extraJavaOptions=-XX –> spark.executor.defaultJavaOptions=-XX:+UseG1GC

#### 遇到的坑

* **Hive Metastore冲突**
  + **原因**  
    EMR 6.2.0里内置的Hive Metastore版本是2.3.7，而公司内部系统使用的目前版本是1.2.1，因此在使用新版EMR的时候会报莫名的各种包问题，根本原因就是使用的Metastore版本冲突问题。
  + **错误信息示例**：
* User class threw exception: java.lang.RuntimeException: [download failed: net.minidev#accessors-smart;1.2!accessors-smart.jar(bundle), download failed: org.ow2.asm#asm;5.0.4!asm.jar, download failed: org.apache.kerby#kerb-core;1.0.1!kerb-core.jar, download failed: org.apache.kerby#kerb-server;1.0.1!kerb-server.jar, download failed: org.apache.htrace#htrace-core4;4.1.0-incubating!htrace-core4.jar, download failed: com.fasterxml.jackson.core#jackson-databind;2.7.8!jackson-databind.jar(bundle), download failed: com.fasterxml.jackson.core#jackson-core;2.7.8!jackson-core.jar(bundle), download failed: javax.xml.bind#jaxb-api;2.2.11!jaxb-api.jar, download failed: org.eclipse.jetty#jetty-util;9.3.19.v20170502!jetty-util.jar, download failed: com.google.inject#guice;4.0!guice.jar, download failed: com.sun.jersey#jersey-server;1.19!jersey-server.jar]
  + **解决方案**
    - 初始方案:
  + "spark.sql.hive.metastore.version": "1.2.1",  
    "spark.sql.hive.metastore.jars": "maven"
    - 但初始方案每次任务运行时都需要去maven库里下载，比较影响性能而且浪费资源，当多任务并发去下载的时候会出问题，并且[官方文档](https://spark.apache.org/docs/2.4.0/sql-data-sources-hive-tables.html#interacting-with-different-versions-of-hive-metastore)不建议在生产环境下使用。因此将lib包的下载直接打入镜像里，然后启动EMR集群的时候加载一次到/dependency\_libs/hive/\*即可，完善后方案为:
  + "spark.sql.hive.metastore.version": "1.2.1",  
    "spark.sql.hive.metastore.jars": "/dependency\_libs/hive/\*"
* **Hive Server连接失败**
  + **错误信息**
* Caused by: org.apache.thrift.TApplicationException: Required field 'client\_protocol' is unset! Struct:TOpenSessionReq(client\_protocol:null, configuration:{set:hiveconf:hive.server2.thrift.resultset.default.fetch.size=1000, use:database=default})
  + **原因**
    - 和Hive metastore包冲突类似的问题，由于Spark 3.0 里用的hive-jdbc.jar包版本过高。
  + **解决方案**
    - 下载可用的对应的lib包，将Spark 3.0里自带的hive-jdbc.jar包进行替换。
  + wget -P ./ https://github.com/timveil/hive-jdbc-uber-jar/releases/download/v1.8-2.6.3/hive-jdbc-uber-2.6.3.0-235.jar
* **写HDFS数据偶尔会失败**  
  在最新版的EMR集群上跑时，经常会出现写HDFS数据阶段失败的情况。查看Log上的error信息:
  + Spark Log
* Spark Log:  
  Caused by: org.apache.hadoop.ipc.RemoteException(java.io.IOException): File /user/hadoop/output/20201023040000/tablename/normal/\_temporary/0/\_temporary/attempt\_20201103002533\_0146\_m\_001328\_760289/event\_date=2020-10-22 03%3A00%3A00/part-01328-7c2e85a0-dfc8-4d4d-8d49-ed9b6aca06f6.c000.zlib.orc could only be written to 0 of the 1 minReplication nodes. There are 1 datanode(s) running and 1 node(s) are excluded in this operation.
  + HDFS Data Node Log
* Data Node Log:  
  365050 java.io.IOException: Xceiver count 4097 exceeds the limit of concurrent xcievers: 4096  
  365051 at org.apache.hadoop.hdfs.server.datanode.DataXceiverServer.run(DataXceiverServer.java:150)  
  365052 at java.lang.Thread.run(Thread.java:748)
  + **解决方案**：  
    调大对应的HDFS连接数。
* dfs.datanode.max.transfer.threads = 16384
  + 不确定EMR集群在升级的过程中是否修改过HDFS连接数的默认参数。

### Scala 升级到 2.12

由于Spark 3.0不再支持Scala 2.11版本，需要将所有的代码升级到2.12的版本。更多Scala 2.12的新的发布内容可以参考[文档](https://www.scala-lang.org/news/2.12.0/#library-improvements)。  
\* **语法升级** \* JavaConversions被deprecated了，需要用JavaConverters并且显示调用.asJava或者.asScala的转化 \* 并发开发相关接口发生变化Scala.concurrent.Future \* **周边相关依赖包升级** \* 包括但不限于 scalstest, scalacheck, scalaxml升级到2.12对应的版本

### 其他相关调整

* **集群资源分配算法调整**  
  整体使用的集群内存在升级3.0后有明显的降低，Data Pipelines根据新的资源需用量重新调整了根据文件大小计算集群资源大小的算法。
* **Python升级到3.x**

## 为什么既能提升性能又能省钱？

我们来仔细看一下为什么升级到3.0以后可以减少运行时间，又能节省集群的成本。 以Optimus数据建模里的一张表的运行情况为例： \* 在reduce阶段从没有AQE的40320个tasks锐减到4580个tasks，减少了一个数量级。 \* 下图里下半部分是没有AQE的Spark 2.x的task情况，上半部分是打开AQE特性后的Spark 3.x的情况。 tasks \* 从更详细的运行时间图来看，shuffler reader后同样的aggregate的操作等时间也从4.44h到2.56h，节省将近一半。 \* 左边是spark 2.x的运行指标明细，右边是打开AQE后通过custom shuffler reader后的运行指标情况。 tasks

**原因分析**： \* AQE特性: \* [AQE](https://xie.infoq.cn/draft/40812)对于整体的Spark SQL的执行过程做了相应的调整和优化(如下图)，它最大的亮点是可以根据已经完成的计划结点真实且精确的执行统计结果来不停的反馈并重新优化剩下的执行计划。 AQE \* **AQE自动调整reducer的数量，减小partition数量**。Spark任务的并行度一直是让用户比较困扰的地方。如果并行度太大的话，会导致task过多，overhead比较大，整体拉慢任务的运行。而如果并行度太小的，数据分区会比较大，容易出现OOM的问题，并且资源也得不到合理的利用，并行运行任务优势得不到最大的发挥。而且由于Spark Context整个任务的并行度，需要一开始设定好且没法动态修改，这就很容易出现任务刚开始的时候数据量大需要大的并行度，而运行的过程中通过转化过滤可能最终的数据集已经变得很小，最初设定的分区数就显得过大了。  
AQE能够很好的解决这个问题，在reducer去读取数据时，会根据用户设定的分区数据的大小(spark.sql.adaptive.advisoryPartitionSizeInBytes)来自动调整和合并(Coalesce)小的partition，自适应地减小partition的数量，以减少资源浪费和overhead，提升任务的性能。 \* 由上面单张表可以看到，打开AQE的时候极大的降低了task的数量，除了减轻了Driver的负担，也减少启动task带来的schedule，memory，启动管理等overhead，减少cpu的占用，提升的I/O性能。 \* 拿历史Data Pipelines为例，同时会并行有三十多张表在Spark里运行，每张表都有极大的性能提升，那么也使得其他的表能够获得资源更早更多，互相受益，那么最终整个的数据建模过程会自然而然有一个加速的[结果](#性能提升明显)。 \* 大batch（>200G）相对小batch（<100G）有比较大的提升，有高达40%提升，主要是因为大batch本身数据量大，需要机器数多，设置并发度也更大，那么AQE展现特性的时刻会更多更明显。而小batch并发度相对较低，那么提升也就相对会少一些，不过也是有27.5%左右的加速。 \* 内存优化 \* 除了因为AQE的打开，减少过碎的task对于memory的占用外，Spark 3.0也在其他地方做了很多内存方面的优化，比如Aggregate部分指标瘦身（[Ticket](https://issues.apache.org/jira/browse/SPARK-29562)）、Netty的共享内存Pool功能([Ticket](https://issues.apache.org/jira/browse/SPARK-24920))、Task Manager死锁问题([Ticket](https://issues.apache.org/jira/browse/SPARK-27338))、避免某些场景下从网络读取shuffle block([Ticket](https://issues.apache.org/jira/browse/SPARK-27651))等等，来减少内存的压力。一系列内存的优化加上AQE特性叠加从[前文](#集群内存使用降低)的图中可以看到集群的内存使用同时有30%左右的下降。 \* Data Pipelines里端到端的每个模块都升级到Spark 3.0，充分获得新技术栈带来的好处。

综上所述，Spark任务得到端到端的加速 + 集群资源使用降低 = 提升性能且省钱。

## 未来展望

接下来，团队会继续紧跟技术栈的更新，并持续对Data Pipelines上做代码层次和技术栈方面的调优和贡献，另外会引入更多的监控指标来更好的解决业务建模中可能出现的数据倾斜问题，以更强力的技术支持和保障FreeWheel正在蓬勃发展的业务。

最后特别感谢AWS EMR和Support团队在升级的过程中给予的快速响应和支持。

## Reference

* [Spark 3.0关键新特性回顾](https://xie.infoq.cn/draft/40812)
* [Spark 3.0 Release Notes](https://spark.apache.org/releases/spark-release-3-0-0.html)
* [AQE](https://databricks.com/blog/2020/05/29/adaptive-query-execution-speeding-up-spark-sql-at-runtime.html)
* [DPP](https://blog.knoldus.com/dynamic-partition-pruning-in-spark-3-0/)
* [UDAF](https://issues.apache.org/jira/browse/SPARK-27296)
* [CBO](https://databricks.com/blog/2017/08/31/cost-based-optimizer-in-apache-spark-2-2.html)
* [Spark SQL语法](https://github.com/apache/spark/blob/master/docs/sql-ref-syntax-qry-select.md)
* [Spark SQL配置](https://spark.apache.org/docs/latest/configuration.html#spark-sql)
* [Spark Web UI使用](https://github.com/apache/spark/blob/master/docs/web-ui.md)
* [Spark Event Logs滚动](https://issues.apache.org/jira/browse/SPARK-29779)
* [EMR Spark配置](https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-spark-configure.html)
* [Parquet嵌套schema问题](https://github.com/apache/spark/pull/24307)
* [Spark Hive Metastore配置](https://spark.apache.org/docs/2.4.0/sql-data-sources-hive-tables.html#interacting-with-different-versions-of-hive-metastore)
* [EMR 结点标签配置](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-master-core-task-nodes.html)
* [Scala 2.12改进](https://www.scala-lang.org/news/2.12.0/#library-improvements)
* [Spark Aggregation指标改进](https://issues.apache.org/jira/browse/SPARK-29562)
* [Spark Netty 共享内存Pool](https://issues.apache.org/jira/browse/SPARK-24920)
* [Spark Task Manager 死锁问题](https://issues.apache.org/jira/browse/SPARK-27338)
* [Spark Shuffle Block避免网络读取](https://issues.apache.org/jira/browse/SPARK-27651)