微店大数据开发平台架构演进

原创：郝伟臣 微店技术团队 2016-11-28

《论语·卫灵公》有云：“工欲善其事，必先利其器。“ 意欲警示世人：要做好一件事，准备工作非常重要。简单来说，与其着急忙慌的开始做一件事，不如沉下心来仔细思考下如何做这件事，毕竟从上海到北京，就算晚出发一天，一个开汽车的也比一个徒步的要早到目的地。今天跟大家分享微店大数据平台诞生的背景以及含有的功能特性和架构设计。

一、为什么需要大数据开发平台

微店在16年4月份之前，数据开发流程基本是这样的：

1. 开发人员通过公共账号登录安装了Hive、Hadoop客户端的gateway机器；
2. 编写自己的脚本，调试代码，完成后通过crontab配置脚本定时执行；
3. 为了防止脚本被其他同事修改，一些谨慎的同事会在每次开发完自己的脚本后同步一份到本机，后面为了实现版本控制，把脚本同步到了git；

可以说这样的开发方式连“小米加步枪”都算不上，而且存在诸多问题：

1. 效率低下。
2. 脚本或代码没有版本控制，开发人员想回滚到以前的版本很不方便。
3. 若开发人员疏忽，添加新的需求后未经过调试，将可能会影响生成的数据，进而影响线上业务。
4. 任务缺乏权限控制，可登陆gateway的任何人都可修改、运行脚本。
5. 对于脚本中依赖的表，只能预估它每天产生的时间，一旦它产出延迟，将影响数据的产出。
6. 任务失败无任何报警，只能依靠人工发现。
7. 任务失败重新恢复后无法自动通知依赖下游重新生成。
8. 任务失败要逐层向上游查找最源头的任务失败原因，排查异常繁琐。
9. 一旦gateway机器故障，所有的任务都将灰飞烟灭，毫无疑问这将是一场灾难。

为此，开发一个大数据开发平台，提高大数据开发的效率，为线上每天调度的任务保驾护航已迫在眉睫。

在16年4月份，我们研究并部署了zeus，公司数据开发效率得到了显著提高，解决了开发调试脚本，同步脚本，失败报警等问题，但同时zeus也存在一些问题，zeus使用的GWT的前端框架，开发新的需求比较繁琐，开发、发布任务的流程很不规范，改动很容易就影响线上，带来线上服务的不稳定。基于这些考虑我们启动了二期的开发，并且将二期大数据开发平台定名为Mars。Mars底层复用zeus，主要规范数据开发、调试、发布流程，更换zeus前端框架，改为使用extjs，并实现脚本版本控制，回滚历史脚本等一系列功能。

二、大数据开发平台应该具备的功能特性

Mars具备的功能特性：

1. 引入版本控制，方便开发人员回滚到之前版本，快速恢复线上调度的任务。
2. 规范大数据开发、测试、上线的流程。
3. 权限控制，任务的所有人、管理员才可以操作任务。
4. 依赖调度，所有依赖的任务执行成功，自动触发自身执行。
5. 任务执行失败，发送执行失败消息给任务所有人，人工介入。
6. 手动恢复任务，恢复成功后，自动通知下游的任务重新执行。

7. 任务依赖图谱，成功失败用不同颜色区分，失败源头一目了然。
8. 任务信息存储在数据库，Mars机器采用分布式系统架构，即使单台机器故障也不会影响使用。
9. 输入输出检测，判断输入表是否准备好，检测输出表数据是否完整。
10. 合理使用Hadoop资源。用户只能使用所属团队指定的hadoop队列。

三、Mars大数据平台构成

大数据开发平台建立在HDFS、YARN、HiveMeta的基础服务之上，目前支持通过Hive、Kylin查找数据，后面所有的数据查询入口将都集成在这里，包括：ES、Redis、Hades等，大数据平台目前支持Shell、Hive、MR、Spark四种任务类型。

四、Mars系统架构设计

Mars系统架构图：

Mars大数据开发平台依托于Hadoop集群，所有的mars机器都必须安装hadoop、Hive客户端。Mars机器有两个身份：master&worker。master负责管理Job、给worker分配Job；worker负责执行Job，提交Job到Hadoop集群，接收Job执行的日志信息。

五、分布式系统架构

- **谁是master，谁是worker**

应用启动时，机器通过抢占的方式争当master，通过在数据库中插入一条记录的方式标识当前谁是master，master每隔一定时间去更新数据库，通过维护一个更新时间间接告知worker它还活着，worker同样每隔一定时间去查询数据库，倘若master的更新时间超过一定时间间隔，worker则认为master故障，第一个发现的worker将当仁不让的成为新的master。

- **master、worker容灾模式**

master与worker之间使用基于protobuf的netty进行通信，master会每隔一段时间去检测与worker的连接，若发现worker故障，master将断开与worker的连接，把分配到该故障worker上的任务重新分配到其他worker上去执行；若master故障，worker将使用抢占的方式争当新的master，新的mater将中止所有正在运行的Job，重新分配。

六、定时、依赖调度

- **定时调度**

Mars使用Quartz来实现定时调度。Quartz是一个完全由java编写的开源作业调度框架。简单地创建一个实现org.quartz.Job接口的java类。Job接口包含唯一的方法：public void execute(JobExecutionContext context) throws JobExecutionException，在你的Job接口实现类里面，添加业务逻辑到execute()方法。一旦配置好Job实现类并设定好调度时间，Quartz将密切注意剩余时间。当调度程序确定该是通知你作业执行的时候，Quartz框架将调用你的Job实现类（作业类）的execute()方法，去做它该做的事情。

- **依赖调度**

依赖调度是指该Job所有依赖的Job全部成功运行完毕时需要被触发的任务类型。当且仅当其所有依赖的任务在当天全部成功运行完毕后，依赖任务将会被触发执行。那么依赖调度是如何实现的?

例如，任务110、112为定时任务，任务119为依赖任务，任务119依赖任务110、112。起初119的状态为：依赖JobId：110、112，已就绪JobId：空。110、112未就绪，119不执行；凌晨2点任务110执行成功，任务119收到110执行成功事件，状态更新为：依赖JobId：110、112，已就绪JobId：110、112未就绪，119此时仍然不执行；凌晨3点任务112执行成功，任务119收到112执行成功事件，状态再次被更新为：依赖JobId：110、112，已就绪JobId：110、112。110、112均已就绪，触发119开始执行。

七、执行任务都做了哪些工作

- 后端

1. 用户在开发中心运行、选中运行或在调度中心手动执行、手动恢复。
2. Mars worker机器进行预判断，包括权限判断、脚本中是否有参数为替换等。
3. worker机器向master机器发送执行任务请求。
4. master机器收到执行任务请求，把任务加入执行队列。
5. master机器定时扫描执行队列，选择合适的worker（负载均衡）执行此任务。
6. 前置准备。包括：资源文件下载，数据表数据监测等。
7. 执行任务。
8. 后置处理。包括数据浮动检查、旧分区清理、添加执行成功标志等。
9. 如果生成了结果文件，还要上传结果文件到Hadoop，方便以后下载。

- 前端

1. 用户触发执行任务。
2. 每隔一段时间请求日志，刷新到前端页面。
3. 任务执行完毕。
4. 如果是开发中心有结果数据，则请求Hadoop上的结果数据。
5. 进行数据展示。

八、遗留问题及后续发展方向

遗留问题

- 检测未跑任务。master挂掉或部署过程中的定时任务不会被触发，需要有机制发现这种任务。
- 重新部署后，正在运行的任务会重新跑。正在运行的任务会被master取消掉，重新分配执行，如果任务执行需要较长的时间，这样做就是无法接受的。
- 检测数据质量。目前输出表仅简单的检测了数据浮动（即数据大小），对于表中的数据内容需要进一步检测，以保证数据产出的合法性。

后续发展方向

- 资源账单。规范用户Hadoop资源使用。
- 数据地图。方便用户找数据。
- 血缘关系。方便用户追溯数据来源。

- 数据流动。方便数据互通。

欢迎有志之士，投身微店团队，杭州join-hz@weidian.com、北京join-bj@weidian.com