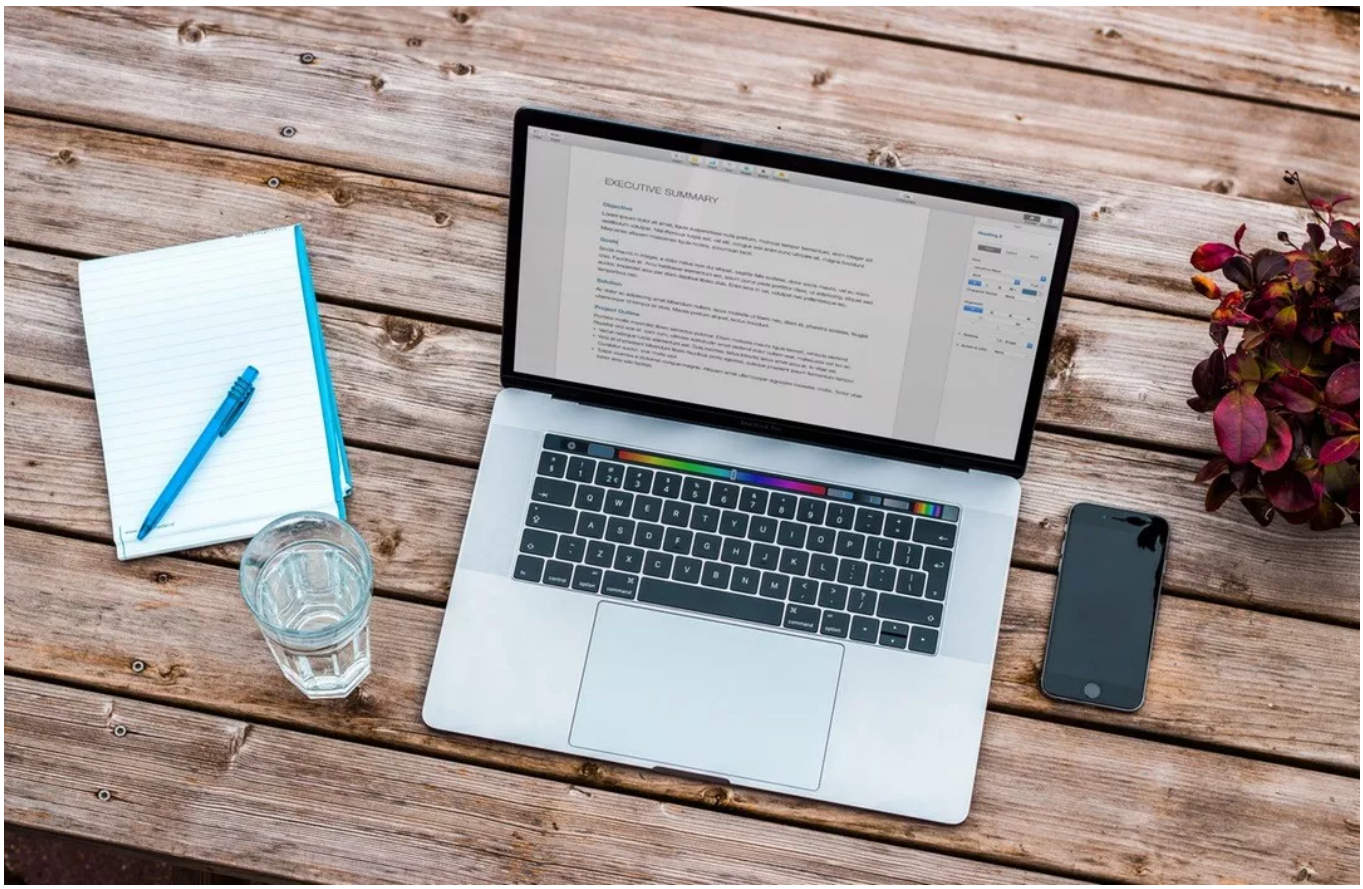
阿里如何实现秒级百万TPS？搜索离线大数据平台架构解读

原创：昆仑 阿里技术 2018-09-17

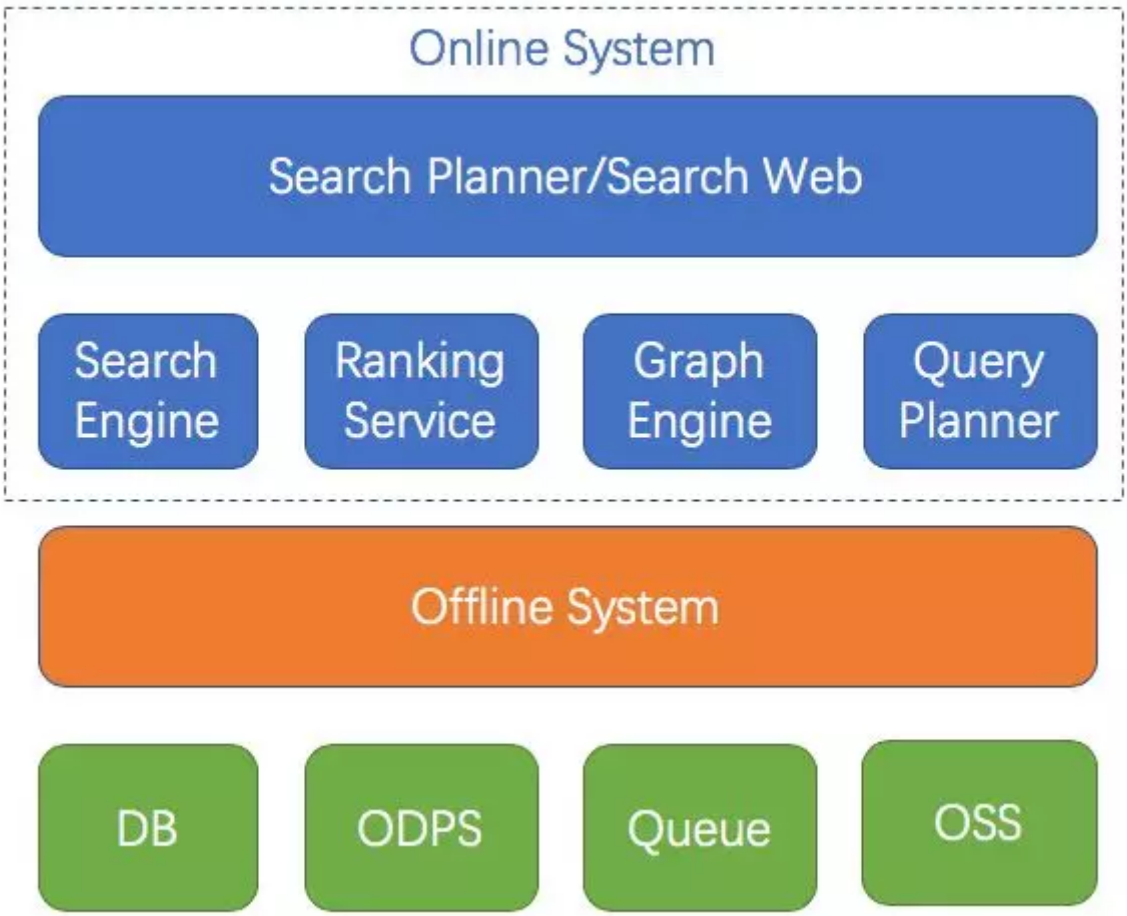


阿里妹导读：搜索离线数据处理是一个典型的海量数据批次/实时计算结合的场景，阿里搜索中台团队立足内部技术结合开源大数据存储和计算系统，针对自身业务和技术特点构建了搜索离线平台，提供复杂业务场景下单日批次处理千亿级数据，秒级实时百万TPS吞吐的计算能力。

背景

什么是搜索离线？

一个典型的商品搜索架构如下图所示，本文将要重点介绍的就是下图中的离线数据处理系统（Offline System）。



何谓离线？在阿里搜索工程体系中我们把搜索引擎、在线算分、SearchPlanner等ms级响应用户请求的服务称之为“在线”服务；与之相对应的，将各种来源数据转换处理后送入搜索引擎等“在线”服务的系统统称为“离线”系统。商品搜索的业务特性（海量数据、复杂业务）决定了离线系统从诞生伊始就是一个大数据系统，它有以下一些特点：

1. 任务模型上区分全量和增量

1）全量是指将搜索业务数据全部重新处理生成，并传送给在线引擎，一般是每天一次。这么做有两个原因：有业务数据是daily更新；引擎需要全量数据来高效的进行索引整理和预处理，提高在线服务效率。

2）增量是指将上游数据源实时发生的数据变化更新到在线引擎中。

3）性能方面有较高要求。全量需要极高吞吐能力，确保数以亿计的数据可以在数小时内完成。增量则需要支持数万TPS秒级的实时性，还需要有极高的可用性。

2. 需要支持多样化的输入和输出数据源，包括：Mysql，ODPS，TT等各种数据库和消息队列作为输入，搜索、Ranking、图、推荐等各种引擎作为输出。

3. 需要提供一定能力的数据处理能力，例如多表Join、UDTF支持等，以方便搜索业务的开发和接入。

在后续的段落中我们会看到离线系统架构围绕着这些特点，针对搜索业务的变化，做出的各种演进和发展。

发展简介

阿里商品搜索体系肇始于淘宝搜索，大约在2008年初第一代搜索系统诞生，离线系统随之上线。搜索离线系统经历多年发展，技术架构几经迭代，数据处理能力、业务支持能力不断提升。下面会分阶段介绍搜索离线的主要技术架构和特点。

★ 淘宝搜索阶段

在2008-2012这个阶段，我们重点支持淘宝搜索的业务发展，随着淘宝商品量的不断增加，逐步引入Hadoop、Hbase等开源大数据计算和存储框架，实现了搜索离线系统的分布式化，有力地支持了淘宝搜索业务的发展。但是在这个阶段，我们支持的业务线只有淘系合计不到5个业务线，为此投入了大约10名开发人员，整体效率不高。另一方面相关系统框架代码与淘系业务高度耦合，量身定制了很多特殊代码，不利于架构的推广和其它业务的支持。

★ 组件&平台化阶段

2013年底以来，特别是最近两年，随着集团技术业务线的梳理以及中台化战略的推行，搜索离线系统需要为越来越多的不同业务团队（飞猪、钉钉、1688、AE、Lazada等等）提供支持，技术框架复用、开发效率提升和平台化支持的需求越来越强烈。另一方面随着大数据计算、存储技术的发展，尤其是流计算引擎的飞速发展，离线系统技术架构上的进一步演进也具备了绝佳的土壤。

我们可以看到整个搜索离线系统的演进是沿着性能和效率两条主线，以业务和技术为双轮驱动，一步一个脚印的走到今天。这是一个技术与业务高度融合互动，互相促进发展的典型样例。

离线平台技术架构

上一节我们简要介绍了离线系统的发展历史，也简要提到技术架构的演进，下面将会把离线平台的技术架构展开介绍，主要分为平台流程以及计算和存储架构等几个方面。

平台组件和任务流程

上图描述了离线平台技术组件结构，其中部分组件的简介如下：

- Maat：分布式任务调度平台，基于Airflow发展而来，主要改进点是调度性能优化、执行器FaaS化、容器化、API及调度功能扩展等四个部分，在保持对Airflow兼容的基础上，大幅提升性能，提高了稳定性。 一个离线任务的多个Blink job会通过Maat建立依赖关系并进行调度。
- Bahamut：执行引擎，是整个离线平台的核心，负责离线任务的创建、调度、管理等各种功能，后文会详细介绍。
- Blink：Flink的阿里内部版本，在大规模分布式、SQL、TableAPI、Batch上做了大量的优化和重构。离线平台的所有计算任务都是Blink job，包括stream和batch。

- Soman：UI模块，与Bahamut后端对接，提供任务信息展示、状态管理等可视化功能，也是用户创建应用的开发业务逻辑的主要入口。
- Catalog： 存储表信息管理，提供各种数据源表的DDL能力，负责离线平台存储资源的申请、释放、变更等各种功能。
- Hippo：阿里搜索自研的分布式资源管理和任务调度服务，类似于Yarn，提供Docker管理能力，主要服务于在线系统。
- Swift：阿里搜索自研高性能分布式消息队列，支持亿级别消息吞吐能力，存储后端为HDFS，存储计算分离架构。

下图则描述了一个离线任务从数据源到产出引擎服务数据的整个过程，流程图分成三层：

- 数据同步层：将用户定义的数据源表的全量和增量数据同步到Hbase内部表，相当于源表的镜像。这个镜像中我们包含cf和d两个列族，分别存储数据库的镜像和Daily更新的数据。
- 数据关联计算层：按照数据源中定义的各种关系，将不同维度的数据关联到一起，把数据送到自定义的UDTF中进行处理，产出引擎所需的全量和增量数据。
- 数据交互层：提供全量和增量数据的存储信息，与在线服务build模块进行交互。

全增量统一的计算模型

那么如何实现对用户屏蔽离线平台内部的这些技术细节，让用户只需要关注业务实现呢？回顾第一节介绍的离线任务概念，离线任务包含全量和增量，它们业务逻辑相同，但是执行模式上有区别。为了让用户能够专注业务逻辑的开发，屏蔽离线平台技术细节实现全增量统一的计算逻辑，我们引入了Business Table（业务表）的概念。

> Business Table（业务表）：Business Table是一个抽象表，由一个全量数据表和/或一个增量流表组成，全量/增量表的Schema相同，业务含义相同。

基于业务表和数据处理组件，用户可以开发出一个描述离线处理流程的业务逻辑图，我们称之为Business Graph。下图就是一个Business Graph的样例，其中上侧红框标识的就是只包含ODPS全量数据源的Business Table，最下方红框中标识的是包含Hdfs+Swift的Business Table，除此之外我们还支持Mysql+DRC/ODPS+Swift等多种业务表的组合。图中还可以看到Join、UDTF等常用的数据处理组件，业务表与处理组件结合在一起就能够描述常见的离线业务处理逻辑。

那么如何把这个Business Graph转化为真正的离线任务呢？Bahamut作为离线平台的执行引擎，会按照Business Graph->APP Graph->Job Graph->(Blink Job/Maat Job)的顺序把一个业务描述转化为可执行的离线任务，具体如下：

1. Business Graph->APP Graph：在这个环节中我们主要有2个重要的工作：

    1）正确性校验：根据BG中的节点信息，校验节点间连接的合法性（例如两个输入源节点不能直接连接）、节点配置的正确性（数据库配置/ODPS的配置是否正确）、Schema推导是否正确。

2）任务分层优化：为了用Blink Stream模式来统一完成全量和增量的执行，我们需要将输入源数据存入内部Hbase，直接使用Blink维表Join功能来完成数据的连接。于是在节点遍历过程中遇到Join、Merge组件时，需要在AppGraph中插入一个内部的HTable节点，将Merge或者Join上游的数据同步进入Hbase。

2. APP Graph->Job Graph：JobGraph是一个Blink/Maat任务的配置DAG，其中每个节点包含配置信息，可以在后续的过程中直接转化为计算或者调度任务。

1）Blink JobGraph：从数据源业务表节点开始遍历AppGraph，每当碰到一个内部HTable节点时，会生成两个（增量/全量）同步层的Blink JobGraph。所有同步层Blink JobGraph生成后，以所有的内部HTable/queue为输入，生成两个（增量/全量）关联处理层的Blink JobGraph。

①同步层：采用Business Table中的全量/增量表配置，分别生成全量和增量的Blink任务配置，描述把数据从数据源同步到内部HTable过程。例如对于Mysql+DRC的表，全量阶段将会从mysql中拉取全表数据并转化为HFile bulkload到HTable中，增量阶段则是从DRC中拉取变化数据，直接写入HTable，并根据需求写入驱动queue。
②关联处理层：关联多个HTable，生成大宽表后调用UDTF处理，产出最终的进入引擎的数据。同样需要分别生成全量和增量任务配置

2)Maat JobGraph：基于Maat的调度任务描述DAG，主要目的是将各个层次的Blink任务按照依赖进行调度，同时执行特定的脚本完成与外部系统的交互等职责。一般来说一个离线任务会生成Build/Publish/Stop/Release等多个Maat JobGraph。

3. Job Graph->Blink/Maat Job：遍历JobGraph，调用Translator将JobGraph转换为Blink/Maat的任务代码。引入JobGraph的目的是将底层的计算引擎与计算任务描述解耦，例如：我们底层的计算引擎曾经是MapReduce +Blink-1.4-TableAPI，最近刚完成了Blink-2.1 基于SQL的升级，我们所有的工作基本上是重写了一套Translator，对上层的代码结构没有任何变动。

经过了上述的三个步骤，我们完成了BusinessGraph（业务描述）到Blink/Maat job的转化，生成了若干数据同步/处理的Blink job，以及将这些Blink job进行依赖调度的完成不同功能的Maat job。特别的针对搜索离线的场景，在调度流程中加入了大量与下游引擎交互的逻辑，包括24小时不间断增量、触发引擎消费数据、切换引擎消费增量queue等重要的业务流程。

存储与计算

★ 基于Hbase的存储架构

搜索离线大约在2012年即引入了Hbase作为数据的存储引擎，有力的支持了搜索业务从淘宝主搜到离线平台的整个发展历程，历经多次双11考验，稳定性和性能都得到明确的验证。从功能层面，搜索离线引入Hbase的原因主要是以下几点：

1. 通过Scan/Get可以批量/单条的获取数据，通过bulkload/put可以批量/单条的导入数据，这与搜索的全量/增量模型完全吻合，天然适合支持搜索离线业务。
2. 底层存储基于HDFS，LSM-Tree的的架构能够确保数据安全性，计算存储分离的架构保证了集群规模水平可扩展，易于提高整体的吞吐。通过单机性能优化（Async、BucketCache、Handler分层、Offheap）和集群的扩容，确保了业务大幅增长时，存储从来没有成为系统的瓶颈。
3. Free Schema的特性能够很好的应对业务数据频繁变化的情况，也能够方便支持一些特殊业务场景的数据逻辑。

通过引入Hbase做为离线系统的内部数据存储，我们成功解决了每天全量时对上游Mysql造成很大压力的问题，大幅度的提升了整体系统的吞吐。数据存储到Hbase也是全量任务向流式处理流程转型（MR->Stream）的基础，而这一点为后来Blink流引擎在搜索离线的孕育和发展也埋下了伏笔。

当然Hbase也不是毫无缺点，JVM内存管理的痼疾、单机Handler打满导致雪崩、缺乏容器化部署能力等也带来了不少烦恼，很快我们就会替换Hbase为阿里内部发展的另外一套存储引擎，期望能够部分的解决这些问题。

★ 基于Flink的计算架构

2016年中，搜索离线逐渐开始引入Flink作为计算引擎，重点解决搜索实时计算场景碰到的大量问题。在社区Flink版本的基础上，实时计算团队开发了Blink，增加原生yarn模式、Incremetal checkpoint等若干解决Flink大规模分布式运行问题的特性，另一方面，在DataStream/DataSet接口的基础上，进一步加强了TableAPI和SQL的功能，真正统一了Stream和Batch的调用接口，并实现计算业务逻辑的sql化开发模式。

离线平台是Blink的早期使用者和开发者，从0.8版本开始经历了多个Blink版本的升级和变迁，先后使用了DataStream、TableAPI和SQL作为任务接口，同时也开发了大量Connector以支持不同数据源之间的交互。目前离线平台已经在使用最新的Blink-2.1.1，Bahamut利用SqlTranslator直接生成SQL来描述任务逻辑，通过Bayes（Blink SQL开发平台）服务化直接提交任务到不同的Yarn集群，这样做有以下几个明显的优势：

1. 采用SQL来描述Blink任务业务逻辑非常清晰，可以直接利用Blink提供的各种Operator完成数据处理，方便任务的调试，例如：dim join、groupby，而不是在Datastream时期需要自行编写完成类似Hbase Join的Operator。
2. Blink 2.1原生支持Batch，采用Batch模式可以直接完成生成HFile的任务，下线MR任务，彻底统一计算引擎到Blink。Batch模式任务执行时采用分阶段调度可以大幅的节省计算资源，提高集群效率。Blink SQL可以通过修改提交模式，分别转化为Stream或Batch任务，在保持业务逻辑稳定的同时方便任务调试和验证。
3. 通过Bayes这样的开发平台服务化的方式提交任务到不同集群，彻底解决以前任务通过GateWay提交运维复杂的问题，添加新的Yarn集群只需要简单配置即可完成。另外在Bayes上同样会保存Bahamut自动生成提交的Sql，可以在Bayes上直接进行任务的调试和管理，方便了开发人员。

下图是一个Bahamut自动生成的Blink Sql样例，描述同步层的一个任务，任务中包含Source，Select Oper和Sink三个Operator，实现从Mysql实时变化到Hbase表的同步。

```
-- 定义数据源表，这是一个DRC（Mysql binlog流）源

CREATE TABLE DRCSource_1 (
  `tag_id`                          VARCHAR,
  `act_info_id`                      VARCHAR,
) with (
  tableFactoryClass='com.alibaba.xxx.xxx.DRCTableFactory',
  -- other config);


-- 定义输出表
CREATE TABLE HbaseSink_1 (
  `tag_id`                     VARCHAR,
  `act_info_id`                VARCHAR,
) with (
  class='com.alibaba.xxx.xxx.CombineSink',
  hbase_tableName='bahamut_search_tmall_act',
  -- other config
 );

-- 定义Blink任务的业务逻辑
INSERT INTO HbaseSink_1SELECT
  `tag_id`,
  `act_info_id`,
FROM DRCSource_1;
```

总结

搜索离线数据处理是一个典型的海量数据批次/实时计算结合的场景，搜索中台团队立足内部技术结合开源大数据存储和计算系统，针对自身业务和技术特点构建了搜索离线平台，提供复杂业务场景下单日批次处理千亿级数据，秒级实时百万TPS吞吐的计算能力。离线平台目前支持了集团内200多个不同业务线的搜索业务需求，大幅提高了业务迭代的效率，成为搜索中台的重要组成部分。很快离线平台还会在阿里云上与Opensearch/ES结合，为集团外客户提供高可用、高性能的搜索离线数据处理能力。在不远的将来离线平台将会逐渐拓展到推荐和广告的数据处理场景，有着广阔的应用场景，一个涵盖搜索/推荐/广告体系的SARO（Search Advertisment and Recommandation Offline）平台会逐步成型。

最后，搜索离线平台从0到1的建设已经走过了两年，但它离我们心目中SARO平台的愿景还离的非常远，在这个前行的道路上一定会充满挑战，有大量难题等着我们去解决。欢迎对Hadoop、Flink等大数据技术感兴趣，有Java后台开发经验的同学加盟，从阿里走向世界，让天下没有难用的搜索。点击文末“**阅读原文**”，即可查看具体岗位。

↑ 翘首以盼等你关注

你可能还喜欢

点击下方图片即可阅读

2018杭州云栖大会，有哪些科技惊喜?

如何量化考核技术人的 KPI?

看完这8本算法好书，才算真正懂了 AI

关注「阿里技术」
把握前沿技术脉搏

阅读原文