阿里巴巴大数据实践（数据管理篇）

天线嘟嘟茄 IP属地: 广东
2020.10.07 01:10:08 字数 7,767 阅读 2,477

第12章 元数据

12.1 元数据概述

12.1.1 元数据定义

按照传统的定义，元数据（Metadata）是关于数据的数据。元数据打通了源数据、数据仓库、数据应用，记录了数据从产生到消费的全过程。元数据主要记录数据仓库中模型的定义、各层级间的映射关系、监控数据仓库的数据状态及ETL的任务运行状态。

在数据仓库系统中，元数据可以帮助数据仓库管理员和开发人员非常方便地找到他们所关心的数据，用于指导其进行数据管理和开发工作，提高工作效率。

将元数据按用途的不同分为两类：

- 技术元数据（Technical Metadata)

存储关于数据仓库系统技术细节的数据，是用于开发和管理数据仓库使用的数据。

（包括分布式计算系统存储元数据/运行元数据、数据开发平台中数据同步/计算任务/任务调度等数据、数据质量和运维相关元数据等）

- 业务元数据（Business Metadata）

从业务角度描述了数据仓库中的数据，它提供了介于使用者和实际系统之间的语义层，使得不懂计算机技术的业务人员也能够“读懂”数据仓库中的数据。

（包括OneData元数据和数据应用元数据等，如数据报表、数据产品等的配置和运行元数据）

12.1.2 元数据价值

元数据有重要的应用价值，是数据管理、数据内容、数据应用的基础，在数据管理方面为集团数据提供在计算、存储、成本、质量、安全、模型等治理领域上的数据支持。

例如在计算上可以利用元数据查找超长运行节点，对这些节点进行专项治理，保障基线产出时间。

在数据内容方面为集团数据进行数据域、数据主题、业务属性等的提取和分析提供数据素材。例如可以利用元数据构建知识图谱，给数据打标签，清楚地知道现在有哪些数据。

天线嘟嘟茄
总资产1

12.1.3 统一元数据体系建设

元数据的质量直接影响到数据管理的准确性，如何把元数据建设好将起到至关重要的作用。

元数据建设的目标是打通数据接入到加工，再到数据消费整个链路，规范元数据体系与模型，提供统一的元数据服务出口，保障元数据产出的稳定性和质量。

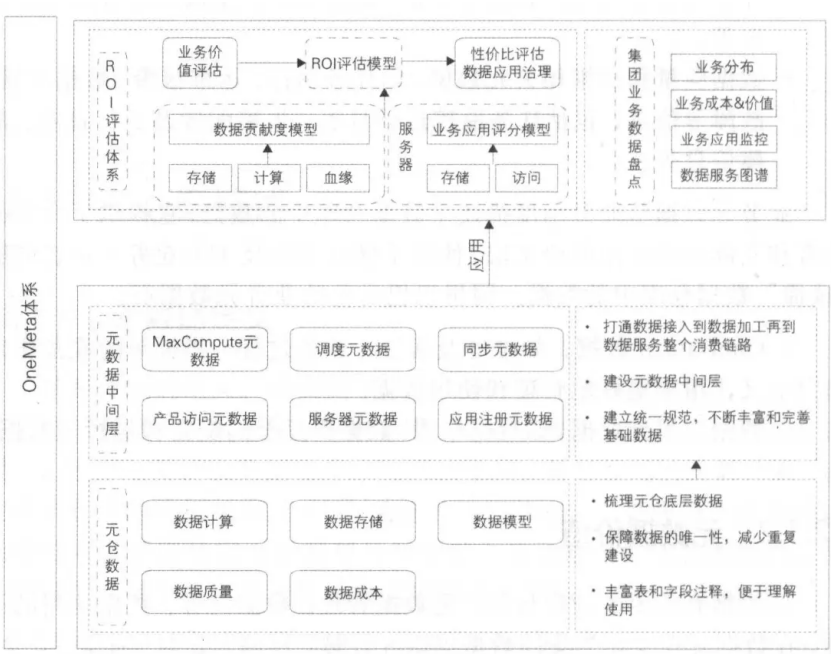

图 12.1 统一元数据体系建设思路图

图12.1 统一元数据体系建设思路图.png

12.2 元数据应用

数据的真正价值在于数据驱动决策，通过数据指导运营。通过数据驱动的方法，我们能够判断趋势，从而展开有效行动，帮助自己发现问题，推动创新或解决方案的产生。这就是数据化运营。同样，对于元数据，可以用于指导数据相关人员进行日常工作，实现数据化“运营”。

比如对于数据使用者，可以通过元数据让其快速找到所需要的数据；对于ETL工程师，可以通过元数据指导其进行模型设计、任务优化和任务下线等各种日常ETL工作；对于运维工程师，可以通过元数据指导其进行整个集群的存储、计算和系统优化等运维工作。

12.2.1 Data Profile

核心思路是为纷繁复杂的数据建立一个脉络清晰的血缘图谱。通过图计算、标签传播算法等技术，系统化、自动化地对计算与存储平台上的数据进行打标、整理、归档。形象地说，DataProfile实际承担的是为元数据“画像”的任务。

- 基础标签

针对数据的存储情况、访问情况、安全等级等进行打标。

- 数仓标签

针对数据是增量还是全量、是否可再生、数据的生命周期来进行标签化处理。

- 潜在标签

这类标签主要是为了说明数据潜在的应用场景，比如社交、媒体、广告、电商、金融等。

12.2.2 元数据门户

元数据门户致力打造一站式的数据管理平台、高效的一体化数据市场。包括“前台”和“后台”，“前台”产品为数据地图，定位消费市场，实现检索数据、理解数据等“找数据”需求；“后台”产品为数据管理，定位于一站式数据管理，实现成本管理、安全管理、质量管理等。

- 数据地图

围绕数据搜索，服务于数据分析、数据开发、数据挖掘、算法工程师、数据运营等数据表的使用者和拥有者，提供方便快捷的数据搜索服务，拥有功能强大的血缘信息及影响分析，利用表使用说明、评价反馈、表收藏及精品表机制，为用户浮现高质量、高保障的目标数据。

比如在进行数据分析前，使用数据地图进行关键词搜索，帮助快速缩小范围，找到对应的数据；比如使用数据地图根据表名直接查看表详情，快速查阅明细信息，掌握使用规则：比如通过数据地图的血缘分析可以查看每个数据表的来源、去向，并查看每个表及字段的加工逻辑。

- 数据管理平台

围绕数据管理，服务于个人开发者、BU管理者、系统管理员等用户，提供个人和BU全局资产管理、成本管理和质量管理等。针对个人开发者，主要包括计算费用和健康分管理、存储费用和健康分管理，并提供优化建议和优化接口；针对BU管理者和管理员，主要提供BU、应用、集群等全局资产消耗概览、分析和预测。

12.2.3 应用链路分析

对于某个数据计算任务或表，其重要程度如何，是否还有下游在使用，是否可以下线；阿里巴巴有这么多数据产品，都依赖哪些MaxCompute表，对这些MaxCompute表是否需要根据应用的重要程度进行资源、运维保障。

对于这些问题，我们都可以通过元数据血缘来分析产品及应用的链路，通过血缘链路可以清楚地统计到某个产品所用到的数据在计算、存储、质量上存在哪些问题，通过治理优化保障产品数据的稳定性。

通过应用链路分析，产出表级血缘、字段血缘和表的应用血缘。其中表级血缘主要有两种计算方式：一种是通过MaxCompute任务日志进行解析；一种是根据任务依赖进行解析。

常见的应用链路分析应用主要有影响分析、重要性分析、下线分析、链路分析、寻根溯源、故障排查等。

12.2.4 数据建模

传统的数据仓库建模一般采用经验建模的方式，效率较低且不准确。基于现有底层数据已经有下游使用的情况，我们可以通过下游所使用的元数据指导数据参考建模。通过元数据驱动的数据仓库模型建设，可以在一定程度上解决此问题，提高数据仓库建模的数据化指导，提升建模效率。

- 数据模型建设

基于下游使用中关联次数大于某个阈值的表或查询次数大于某个阈值的表等元数据信息，筛选用于数据模型建设的表。

基于表的字段元数据，如字段中的时间字段、字段在下游使用中的过滤次数等，选择业务过程标识字段。

- 关联从表

基于主从表的关联关系、关联次数，确定和主表关联的从表。

- 目标模型

基于主从表的字段使用情况，如字段的查询次数、过滤次数、关联次数、聚合次数等，确定哪些字段进入目标模型。

12.2.5 驱动ETL开发

可以通过DataProfile得到数据的下游任务依赖情况、最近被读写的次数、数据是否可再生、每天消耗的存储计算等，这些信息足以让我们判断数据是否可以下线；如果根据一些规则判断可以下线，则会通过OneClick触发一个数据下线的工作任务流，数据Owner可能只需要点击提交按钮，删除数据、删除元数据、下线调度任务、下线DQC监控等一系列操作就会自动在后台执行完成。

第13章 计算管理

目前内部MaxCompute集群上有200多万个任务，每天存储资源、计算资源消耗都很大。如何降低计算资源的消耗，提高任务执行的性能，提升任务产出的时间，是计算平台和ETL开发工程师孜孜追求的目标。

13.1 系统优化

Hadoop等分布式计算系统评估资源的方式，一般是根据输入数据量进行静态评估，Map任务用于处理输入，对于普通的Map任务，评估一般符合预期：而对于Reduce任务，其输入来自于Map的输出，但一般只能根据Map任务的输入进行评估，经常和实际需要的资源数相差很大，所以在任务稳定的情况下，可以考虑基于任务的历史执行情况进行资源评估，即采用HBO(History-Based Optimizer，基于历史的优化器）。

13.2 任务优化

主要从数据倾斜方面进行数据优化。

13.2.1 Map倾斜

Map端是MR任务的起始阶段，Map端的主要功能是从磁盘中将数据读入内存，在Map端读数据时，由于读入数据的文件大小分布不均匀，因此会导致有些MapInstance读取并且处理的数据特别多，而有些MapInstance处理的数据特别少，造成Map端长尾。

以下两种情况可能会导致Map端长尾：

- 上游表文件的大小特别不均匀，并且小文件特别多，导致当前表Map端读取的数据分布不均匀，引起长尾。

可通过对上游合并小文件，同时调节本节点的小文件的参数来进行优化。

- Map端做聚合时，由于某些MapInstance读取文件的某个值特别多而引起长尾，主要是指CountDistinct操作。

表的分发和笛卡儿积操作。使用随机分布函数后，Map端只负责数据的分发，不再有复杂的聚合或者笛卡儿积操作，因此不会导致Map端长尾。

（代码略）

Map端长尾的根本原因是由于读入的文件块的数据分布不均匀，再加上UDF函数性能、Join、聚合操作等，导致读人数据量大的MapInstance耗时较长。在开发过程中如果遇到Map端长尾的情况，首先考虑如何让MapInstance读取的数据量足够均匀，然后判断是哪些操作导致MapInstance比较慢，最后考虑这些操作是否必须在Map端完成，在其他阶段是否会做得更好。

13.2.2 Join倾斜

MaxComputeSQL在Join执行阶段会将JoinKey相同的数据分发到同一个执行Instance上处理。如果某个Key上的数据量比较大，则会导致该Instance执行时间较长。其表现为：在执行日志中该JoinTask的大部分Instance都已执行完成，但少数几个Instance一直处于执行中（这种现象称之为长尾）。

因为数据倾斜导致长尾的现象比较普遍，严重影响任务的执行时间，尤其是在“双11”等大型活动期间，长尾程度比平时更严重。比如某些大型店铺的PV远远超过一般店铺的PV，当用浏览日志数据和卖家维表关联时，会按照卖家ID进行分发，导致某些大卖家所在Instance处理的数据量远远超过其他Instance，而整个任务会因为这个长尾的Instance迟迟无法结束。

三种常见的倾斜场景：

- Join的某路输入比较小

可以采用MapJoin，避免分发引起长尾。

- Join的每路输入都较大，且长尾是空值导致的

可以将空值处理成随机值，避免聚集。

- Join的每路输入都较大，且长尾是热点值导致的

可以对热点值和非热点值分别进行处理，再合并数据。

13.2.3 Reduce倾斜

Reduce端负责的是对Map端梳理后的有序key-value键值对进行聚合，即进行Count、Sum、Avg等聚合操作，得到最终聚合的结果。

Distinct是MaxComputeSQL中支持的语法，用于对字段去重。比如计算在某个时间段内支付买家数、访问UV等，都是需要用Distinct进行去重的。

MaxCompute中Distinct的执行原理是将需要去重的字段以及GroupBy字段联合作为key将数据分发到Reduce端。

因为Distinct操作，数据无法在Map端的Shuffle阶段根据GroupBy先做一次聚合操作，以减少传输的数据量，而是将所有的数据都传输到Reduce端，当key的数据分发不均匀时，就会导致Reduce端长尾。

Reduce端产生长尾的主要原因就是key的数据分布不均匀。比如有些Reduce任务Instance处理的数据记录多，有些处理的数据记录少，造成Reduce端长尾。

- 对同一个表按照维度对不同的列进行Count Distinct操作，造成Map端数据膨胀，从而使得下游的Join和Reduce出现链路上的长尾。

- Map端直接做聚合时出现key值分布不均匀，造成Reduce端长尾。

可以对热点key进行单独处理，然后通过“Union All”合并。

- 动态分区数过多时可能造成小文件过多，从而引起Reduce端长尾。

可以把符合不同条件的数据放到不同的分区，避免通过多次“InsertOverwrite”写人表中，特别是分区数比较多时，能够很好地简化代码。

但是动态分区也有可能会带来小文件过多的困扰。

MaxCompute对动态分区的处理是引人额外一级的Reduce Task，把相同的目标分区交由同一个（或少量几个）Reduce Instance来写入，避免小文件过多，并且这个Reduce肯定是最后一个ReduceTask操作。

- 多个Distinct同时出现在一段SQL代码中时，数据会被分发多次，不仅会造成数据膨胀N倍，还会把长尾现象放大N倍。

在7天、30天等时间范围内，分PC端、无线端、所有终端，计算支付买家数和支付商品数，其中支付买家数和支付商品数指标需要去重。因为需要根据日期、终端等多种条件组合对买家和商品进行去重计算，因此有12个Count Distinct计算。在计算过程中会根据12个组合key分发数据来统计支付买家数和支付商品数。这样做使得节点运行效率变低。

针对上面的问题，可以先分别进行查询，执行GroupBy原表粒度+ buyer_id，计算出PC端、无线端、所有终端以及在7天、30天等统计口径下的buyer_id（这里可以理解为买家支付的次数），然后在子查询外GroupBy原表粒度，当上一步的Count值大于0时，说明这一买家在这个统计口径下有过支付，计人支付买家数，否则不计入。计算支付商品数采用同样的处理方式。最后对支付商品数和支付买家数进行Join操作。

（代码略）

对MultiDistinct的思考如下：

- 上述方案中如果出现多个需要去重的指标，那么在把不同指标Join在一起之前，一定要确保指标的粒度是原始表的数据粒度。

比如支付买家数和支付商品数，在子查询中指标粒度分别是：原始表的数据粒度＋buyer_id和原始表的数据粒度＋item_id，这时两个指标不是同一数据粒度，所以不能Join，需要再套一层代码，分别把指标GroupBy到“原始表的数据粒度”，然后再进行Join操作。

- 修改前的MultiDistinct代码的可读性比较强，代码简洁，便于维护；修改后的代码较为复杂。

当出现的Distinct个数不多、表的数据量也不是很大、表的数据分布较均匀时，不使用MultiDistinct的计算效果也是可以接受的。

所以，在性能和代码简洁、可维护之间需要根据具体情况进行权衡。另外，这种代码改动还是比较大的，需要投入一定的时间成本，因此可以考虑做成自动化，通过检测代码、优化代码自动生成将会更加方便。

目前Reduce端数据倾斜很多是由Count Distinct问题引起的，因此在ETL开发工作中应该予以重视Count Distinct问题，避免数据膨胀。对于一些表的Join阶段的Null值问题，应该对表的数据分布要有清楚的认识，在开发时解决这个问题。

第14章 存储和成本管理

如何有效地降低存储资源的消耗，节省存储成本，将是存储管理孜孜追求的目标。

14.1 数据压缩

14.2 数据重分布

在MaxCompute中主要采用基于列存储的方式，由于每个表的数据分布不同，插人数据的顺序不一样，会导致压缩效果有很大的差异，因此通过修改表的数据重分布，避免列热点，将会节省一定的存储空间。

目前我们主要通过修改distributeby和sortby字段的方法进行数据重分布。

14.3 存储治理项优化

阿里巴巴数据仓库在资源管理的过程中，经过不断地实践，慢慢摸索出一套适合大数据的存储优化方法，在元数据的基础上，诊断、加工成多个存储治理优化项。

目前已有的存储治理优化项有未管理表、空表、最近62天未访问表、数据无更新无任务表、数据无更新有任务表、开发库数据大于1OOGB且无访问表、长周期表等。

通过对该优化项的数据诊断，形成治理项，治理项通过流程的方式进行运转、管理，最终推动各个ETL开发人员进行操作，优化存储管理，并及时回收优化的存储效果。在这个体系下，形成现状分析、问题诊断、管理优化、效果反馈的存储治理项优化的闭环。通过这个闭环，可以有效地推进数据存储的优化，降低存储管理的成本。

14.4 生命周期管理

生命周期管理的根本目的就是用最少的存储成本来满足最大的业务需求，使数据价值最大化。

管理策略：

- 周期性删除策略

所存储的数据都有一定的有效期，从数据创建开始到过时，可以周期性删除X天前的数据。

- 彻底删除策略

无用表数据或者ETL过程产生的临时数据，以及不需要保留的数据，可以进行及时删除，包括删除元数据。

- 永久保留策略

重要且不可恢复的底层数据和应用数据需要永久保留。

比如底层交易的增量数据，出于存储成本与数据价值平衡的考虑，需要永久保留，用于历史数据的恢复与核查。

- 极限存储策略

户维表）就使用极限存储。

- 冷数据管理策略

冷数据管理是永久保留策略的扩展。永久保留的数据需要迁移到冷数据中心进行永久保存，同时将MaxCompute中对应的数据删除。

一般将重要且不可恢复的、占用存储空间大于IOOTB，且访问频次较低的数据进行冷备，例如3年以上的日志数据。

- 增量表merge全量表策略

对于某些特定的数据，极限存储在使用性与存储成本方面的优势不是很明显，需要改成增量同步与全量merge的方式，对于对应的delta增量表的保留策略，目前默认保留93天。

例如，交易增量数据，使用订单创建日期或者订单结束日期作为分区，同时将未完结订单放在最大分区中，对于存储，一个订单在表里只保留一份；对于用户使用，通过分区条件就能查询某一段时间的数据。

管理矩阵：

- 历史数据等级划分

对历史数据进行重要等级的划分。

- 表类型划分

划分为事件型流水表（增量表）、事件型镜像表（增量表）、维表、Merge全量表、ETL临时表、TT临时数据、普通全量表等。

14.5 数据成本计量

在计量数据表的成本时，除考虑数据表本身的计算成本、存储成本外，还要考虑对上游数据表的扫描带来的扫描成本。

我们将数据成本定义为存储成本、计算成本和扫描成本三个部分。

通过在数据成本计量中引人扫描成本的概念，可以避免仅仅将表自身硬件资源的消耗作为数据表的成本，以及对数据表成本进行分析时，孤立地分析单独的一个数据表，能够很好地体现出数据在加工链路中的上下游依赖关系，使得成本的评估尽量准确、公平、合理。

14.6 数据使用计费

对于数据表的使用计费，分别依据这三部分成本进行收费，称为：计算付费、存储付费和扫描付费。

把数据资产的成本管理分为数据成本计量和数据使用计费两个步骤。通过成本计量，可以比较合理地评估出数据加工链路中的成本，从成本的角度反映出在数据加工链路中是否存在加工复杂、链路过长、依赖不合理等问题，间接辅助数据模型优化，提升数据整合效率；通过数据使用计费，可以规范下游用户的数据使用方法，提升数据使用效率，从而为业务提供优质的数据服务。

第15章 数据质量

- 完整性

指数据的记录和信息是否完整，是否存在缺失的情况。

数据的缺失主要包括记录的缺失和记录中某个字段信息的缺失，两者都会造成统计结果不准确，所以说完整性是数据质量最基础的保障。

- 准确性

准确性是指数据中记录的信息和数据是否准确，是否存在异常或者错误的信息。

- 一致性

一般体现在跨度很大的数据仓库体系中，比如阿里巴巴数据仓库，内部有很多业务数据仓库分支，对于同一份数据，必须保证一致性。（必须都是同一种类型，长度也需要保持一致）

所以，才有了公共层的加工，以确保数据的一致性。

- 及时性

在确保数据的完整性、准确性和一致性后，接下来就要保障数据能够及时产出，这样才能体现数据的价值。

一般决策支持分析师都希望当天就能够看到前一天的数据，而不是等三五天才能看到某一个数据分析结果；否则就已经失去了数据及时性的价值，分析工作变得毫无意义。现在对时间要求更高了，越来越多的应用都希望数据是小时级别或者实时级别的。

15.2 数据质量万法概述

- 消费场景知晓

通过数据资产等级和基于元数据的应用链路分析解决消费场景知晓的问题。

根据应用的影响程度，确定资产等级；根据数据链路血缘，将资产等级上推至各数据生产加工的各个环节，确定链路上所涉及数据的资产等级和在各加工环节上根据资产等级的不同所采取的不同处理方式。

- 数据生产加工各个环节卡点校验

主要包括在线系统（OLTP）和离线系统（OLAP）数据生产加工各个环节的卡点校验。

- 风险点监控

针对在数据日常运行过程中可能出现的数据质量和时效等问题进行监控，并设置报警机制，包括在线数据和离线数据的风险点监控两个方面。

- 质量衡量

对质量的衡量既有事前的衡量，如DQC覆盖率；又有事后的衡量，主要用于跟进质量问题，确定质量问题原因、责任人、解决情况等，并用于数据质量的复盘，避免类似事件再次发生。

根据质量问题对不同等级资产的影响程度，确定其是属于低影响的事件还是具有较大影响的故障。质量分则是综合事前和事后的衡量数据进行打分。

- 质量配套工具