Cobar的架构与实践

文 / 邱硕

Cobar是提供分布式数据库服务的中间件，是阿里巴巴B2B前台应用访问数据库的统一入口。本文讲述的就是Cobar的架构演变与应用实践。

2008年，阿里巴巴B2B成立了平台技术部，为各个业务部门的产品提供底层的基础平台。这些平台涵盖Web框架、消息通信、分布式服务、数据库中间件等多个领域的产品。它们有的源于各个产品线在长期开发过程中沉淀出来的公共框架和系统，有的源于对现有产品和运维过程中新需求的发现。数据库相关的平台就是其中之一，主要解决以下三方面的问题。

- 为海量前台数据提供高性能、大容量、高可用性的访问。
- 为数据变更的消费提供准实时的保障。
- 高效的异地数据同步。

应用层通过Cobar访问数据库（如图1所示）。对数据库的访问分为读操作（select)和写操作(update、insert和delete)。写操作会在数据库上产生变更记录，MySQL的变更记录叫binlog，Oracle的变更记录叫redolog。图1中的Erosa产品解析这些变更记录，并以统一的格式缓存至Eromanga中，后者负责管理变更数据的生产者（Erosa）和消费者之间的关系。负责跨机房数据库同步的Otter是这些变更数据的消费者之一。

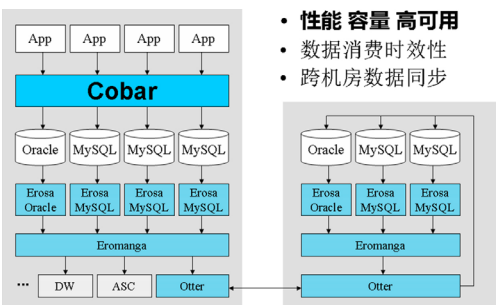

图1　Cobar是提供分布式数据库服务的中间件

截至2012年6月，Cobar的使用方已涵盖B2B中文站、国际站、国际交易、搜索、数据仓库、ITU等部门的200多个应用，其中包括中文站的Offer和国际站产品等的核心应用。

中文站Offer表的拆分和amoeba

2008年，阿里巴巴的业务基本上使用同一个Oracle+小型机做数据库，一个单点承载了多个核心业务，比如中文站的Offer应用，其数据量和访问量都很高。截至2008年，数据库的Offer表已经有1亿的数据量。在高峰时期，Oracle小型机常常load高达30，CPU利用率在90%以上。

同时，过多的应用使用同一个单点数据库，导致连接数过多，影响性能。Oracle的Standby切换在实际使用中也出现过问题，导致网站不可用这样的事故。Oracle数据库不开源，基本可以算是一个黑盒，出现的问题很难定位。随着业务量的迅速增长（2008年Offer表有1亿数据，2010年达到了3亿数量），单点的Oracle+小型机出现了瓶颈，不能满足业务需求。摆在面前的有三种选择：升级小型机的硬件；购买多台小型机以分散各个业务；以廉价PC服务器+开源数据库替代之。我们选择了第三条路，主要解决当时单点Oracle四个方面的问题。

- 数据和访问从集中式改变为分布式。

- 提供数据节点的failover。
- 解决连接数过大的问题。
- 对业务代码侵入性少。

数据分布

为了让单点数据分散到分布式的PC服务器上，基于数据表的水平拆分从一开始便被引入，成为系统的核心功能。如图2所示，MEMBER_ID字段是数据库表的拆分字段，对于某一行记录，路由算法根据拆分字段的值决定将其分布到哪个分库。

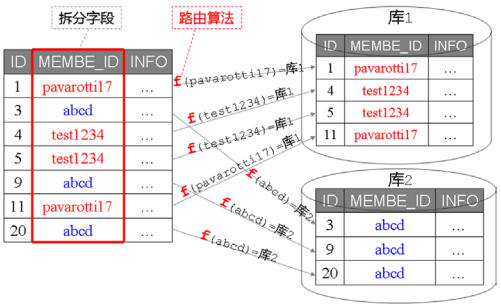

图2　数据库的水平拆分

图3描述了路由算法的过程，它使用了一致性Hash，使得扩容时数据移动较少且仅在局部移动。图3中的各个分库数据空间均匀分布（都是256），当然也是可以非均匀的，以适应不同机器的处理能力。这种水平拆分解决了第一个问题。

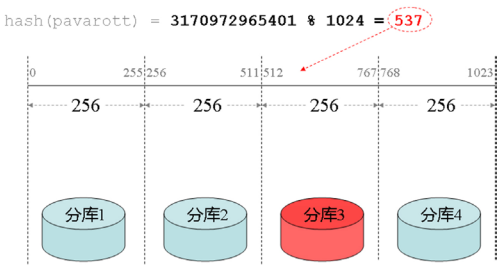

图3　路由算法的过程

第一版的系统结构

以数据水平拆分为基础，我们开发了Cobar的前身——Amoeba（下文仍旧称之为Cobar），它的结构如图4所示。

从图4中可以看出Amoeba是一个独立的进程，与应用之间通过MySQL协议进行交互，是一个proxy的结构，对外暴露jdbc:mysql://CobarIP:port/schema。对于应用而言，连接Cobar和连接

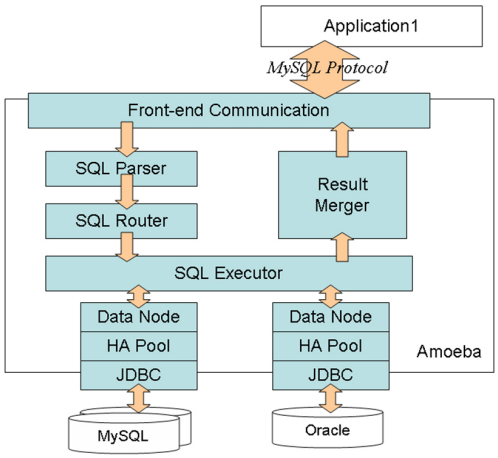

图4　Cobar的前身Amoeba

MySQL没有区别。这样做有以下几个好处。

- 无需应用引入新的jar包，从访问Oracle迁到访问Cobar可以复用原有的基于JDBC的DAO。
- 除了应用代码之外，通过普通的MySQL Command工具或者其他第三方数据库管理工具，可以像访问原有单点数据库一样访问Cobar，方便数据库的管理和问题的排查、调试。
- 数据库连接复用。Cobar使用连接池与后台真实数据库进行交互。由于Cobar的实例数量远远小于应用的实例数量，可极大程度地节约后台数据库连接（实际中，根据应用的不同，使用proxy结构后数据库连接数能够节约2~10倍不等）。虽然应用与Cobar之间的连接数和原有Oracle单点连接数相同，但Cobar本身的连接是轻量的，能够承受连接数上限远远高于后台数据库。

数据访问

Cobar通过SQL语句转发的方式实现数据访问。如图5所示，用户发来的SQL语句，Cobar解析其内容，判断该语句所涉及的数据分布在哪个分库上，再将语句转发给此分库执行（在这一版本的Cobar中，当SQL语句中涉及的拆分字段有多值，

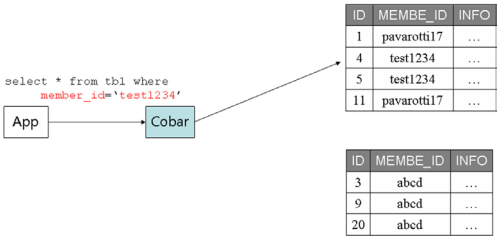

图5　通过SQL语句转发的方式实现数据库访问

如member_id IN (…)，或where条件中没有出现拆分字段时，该语句将会转发至后台所有分库执行），再将执行结果以MySQL协议包的形式送回应用端。这个过程会涉及Cobar的所有关键模块。

首先是通信模块，它负责从连续的网络数据流中识别出一个个MySQL协议包，再解析协议包识别出SQL语句输出给Parser模块。同时，把Result Merge模块输入的执行结果，编码成MySQL的协议包。它以NIO方式实现，有很高的执行效率。

这一版本的Parser负责识别出SQL语句WHERE条件中的等式（如Column1=123），以Map的形式和SQL涉及的表名一起，输出给路由模块。此时的Parser基于JavaCC实现，由文法规则自动生成（后续版本重新手写实现），文法规则涵盖了Oracle和MySQL的常用语法。

Router模块从Parser模块输出的Map中找出相应表的拆分字段的值，计算出该SQL分发到哪个分库，交由Executor负责执行。

数据源层次

水平拆分后，后台有多个数据源，对它们的管理分为两个层次，第一层是DataNode，第2层是replica。

DataNode管理拆分，一个DataNode存放一个分片的数据，彼此无数据交集。每个分片的数据存多份以保证高可用，每一份叫做一个replica，由HA层管理。每个replica表示一个具体的数据源，它是一个连接池，池内管理每一个具体的JDBC连接。路由运算只关注到DataNode层，之下的层次对其不可见。

每一份replica之间的数据复制和同步由MySQL本身的replication协议完成，同一时刻只有一个replica提供服务（称为Master，其余replica称为Slave），

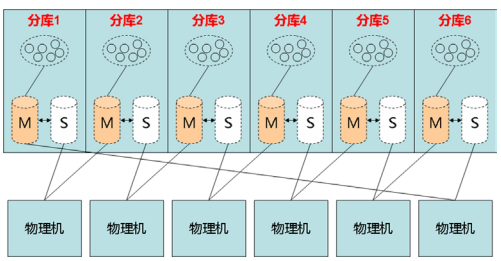

图6　物理机的部署方式

Cobar会与之保持心跳，一旦发现它不可用，会切换至另一个replica，解决Oracle单点的第二个问题。由于Slave并不对外提供服务，所以为了节约数据库的机器数量，可以采用图6中的方式部署。

MySQL采用异步的复制机制，因此多份replica间的读写分离会造成一定程度的数据不一致，因此没有被Cobar采用。

应用

这一版Cobar在2009年底开发完成，首先在规模较小的几个应用上试用。2010年，在中文站Offer应用上线。2011年下半年，包括国际站产品在内的20多个应用开始使用。这些应用原先运行在Oracle单点上，迁移至Cobar之后后端用多台MySQL提供服务，对它们进行数据迁移的过程，不能影响这些应用的在线使用，即需要进行在线迁移。在这20多个应用的实施过程中，逐步形成了一套统一的数据迁移策略，需要保证以下几点。

- 迁移过程中应用不停止或暂停服务。
- 迁移后数据一致。
- 迁移步骤中出现错误可回滚。

迁移过程涉及三个部分：应用的DAO（Data Access Objects）、Cobar和迁移任务进程，如图7所示。迁移以表为单位，过程有以下三个阶段。

阶段1：在应用的DAO代码中增加Cobar的连接，但应用的读写都在原有的Oracle上。

阶段2：应用读写仍旧在Oracle上，同时写一份数据到MySQL中。在此阶段，后台的迁移任务启动，将Oracle上相应表的全部记录读取并按照拆分规则写入MySQL实例，该过程中，前台DAO仍

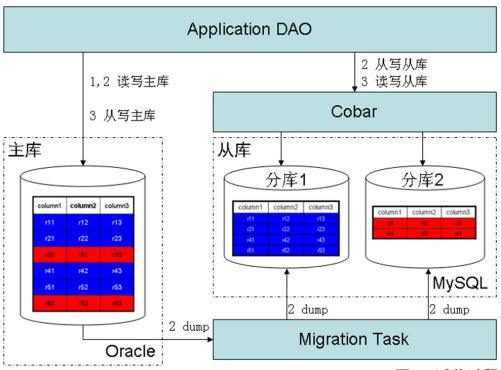

图7　迁移过程

然会对记录进行从写，可能更改或删除任务刚刚写入的数据，此时迁移任务会处理数据冲突相关逻辑。因为此时MySQL数据尚未对外提供服务，如果该阶段出现问题，可删除MySQL数据，重启一次本过程。

阶段3：阶段2结束后，主从两库的数据一致，此时，可将应用的读写迁移至Cobar，同时也从写一份数据到Oracle以便该阶段失败时回滚。

中间件的集群化

截至2011年，Cobar中间件为每个应用单独部署进程以保证应用间的隔离性，但随着应用数量的增多，Cobar服务器本身的数量也在上升。我们希望能够有一套Cobar集群服务于同一个地区的所有应用，做到跨应用的共享。为此，我们重写了Cobar，在提升性能的同时也增加一些功能。

通信模块

尽管第一版的NIO具有很高的资源利用率，但仍存在一定的优化空间，我们引入了一个ByteBuffer池，将NIO的Buffer统一管理起来，减少了NIO数据交互时的垃圾回收。重写之后的通信层，使用普通的PC服务器，Cobar实例只使用2GB的内存，单实例可以达到15万QPS。

后端去除JDBC

Cobar前端使用的是优化后的NIO通信模块，为了让该模块在后端使用，我们去除了JDBC。与后端数据库交互，Cobar直接面向协议，目前实现了基于MySQL协议的后端交互。

SQL解析和路由

原有的SQL解析器是基于JavaCC自动生成的，我们用手写的方式以LL(2)重新实现，支持MySQL 5.5的DML语法，且对它的函数和特殊语法元素，都做了很好的支持，在解析功能上做到与MySQL一致。性能比原有版本提高了10倍，解析类似这样的语句（seLEcT id, member_id, image_path, image_size, STATUS, gmt_modified from wp_image wheRe id = ? AND member_id=-123.456）只需4微秒。路由方面，实现按照多维度对表进行水平拆分，以减少语句分发的次数。

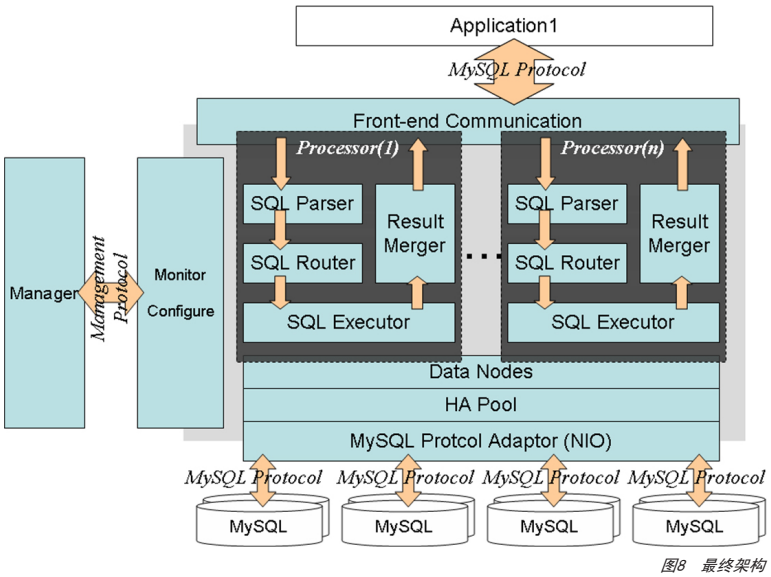

图8 最终架构

最终架构

图8是Cobar最终的架构。与第一版相比，增加了管理和监控Cobar服务器状态的Manager，同时解析路由和执行等操作被分在了单独的Processor中，以保证一定程度的隔离性。此时的Cobar只需要3-5台PC服务器，即可支撑一个机房全部应用的压力。其中一台Cobar进程的数据如下：平均QPS是3000，进程运行了9天后，只有90次FGC，而Cobar的Java进程堆内存仅1.2GB。

这一版本在2011年10月发布，至今已有200余个应用运行在它之上。

2012年6月，Cobar开源，地址为http://code.alibabatech.com/wiki/display/Cobar。7月，阿里云正在开发的火焰山项目（分布式关系型数据库的云服务）选择Cobar作为其底层中间件，实现数据分片和Proxy的需求。

邱硕

2009年加入阿里巴巴平台技术部，参与过分布式服务框架等平台产品的开发，2010年至今在Cobar团队开发分布式数据库中间件。

责任编辑：杨爽（yangshuang@csdn.net）