# 数据库系统应用分片中间件①

王亚玲[1], 杨 超[1], 章名尚[2]

[1](国网信息通信产业有限公司, 北京 100031)
[2](天津市普迅电力信息技术有限公司, 天津 300384)

**摘 要**: 随着国家电网信息化系统一级部署的陆续实施, 业务数据库的表越来越多, 表中的数据量越来越大. 海量数据的存储与访问对数据库造成了相当大的负载, 数据处理能力和访问能力均遭遇瓶颈. 本文针对国家电网一级部署数据量快速增长的特点, 提出了数据分片中间件 UAPDS(UAP DB Sharding). UAPDS 以集群、分区、分片三层拆分模式, 通过读写分离、单库分表、多库分表等多种解决方案将数据按照一定的规则切分到不同的库或表中, 有效的缓解了对数据库的压力, 提高了数据访问效率, 在实际应用中取得了较好的效果.

**关键词**: 数据分片; 海量数据; 读写分离; 分表;

## Middleware of Database System Fragmentation

WANG Ya-Ling[1], YANG Chao[1], ZHANG Ming-Shang[2]

[1](State Grid Information Communication Industry Group Co. Ltd., Beijing 100031, China)
[2](Tianjin Richsoft Electric Power Information Technology Co. Ltd., Tianjin 300384, China)

**Abstract**: As the successively implementation of State Grid's information system first-degree deployment, the tables in database become much more than before, the datas in table become very large. Massive data storage and access caused considerable load on the database, data processing capacity and access encounter bottlenecks. The amount of data is growing rapidly based on State Grid first-degree deployment, this paper proposed data fragmentation middleware UAPDS (UAP DB Sharding) technology. UAPDS provides multiple split solutions, such as separated of reading and writing, divided into multiple tables in a database or in multiple databases, by router, partition, shard three layer split mode, the data cut into different databases or tables according to certain rules, effectively relieve the pressure on the database, improved the efficiency of data access, achieved better in the practical application effect.

**Key words**: data fragmentation; massive data; read and write separation; points table

国家电网公司"十二五"信息化建设要求在两级部署的 SG186 信息平台基础上, 建设以支撑智能电网发展, 实现业务集中和数据共享, 逐步建成各业务应用的一级部署信息平台. 随着国家电网公司信息化项目一级部署应用范围的不断推进, 业务数据库的表越来越多, 单表的数据量越来越大, 海量数据的存储与访问对数据库造成了相当大的负载, 数据处理能力和访问性能下降, 集中式数据库已经无法满足企业对海量数据高可用性、高可靠性的需求, 已成为系统发展的瓶颈. 为了缓解数据量的不断增长对数据库的压力, 这就需要使用数据分片技术.

目前数据分片解决方案主要有垂直拆分和水平拆分. 垂直拆分是将表按照功能模块、关系密切程度切分到不同的库中. 水平拆分是将表的数据按某种特定规范进行散列划分, 然后存储到多个结构相同的表上[1]. 数据分片技术出现了不少的优秀产品, 如 Hibernate Shards、Cobar、数据库自行分区等, 但这些产品都有一定的限制. Hibernate Shards 实现机制与 ORM 框架绑定, 扩展比较困难, 如果 ORM 框架发生变动, 会带来很大的维护成本; Cobar 是在数据访问代

① 收稿时间:2015-01-29;收到修改稿时间:2015-04-02

理层实现分库分表逻辑, 因为增加了代理, 需要额外的消息通讯, 性能方面不太理想, 且 Cobar 仅支持 MySQL 数据库,具有一定的局限性; 主流的关系型数据库, 如商用数据库 Oracle、DB2 有成熟的数据库分区解决方案, 但价格昂贵, 而开源数据库 MySQL 和 PostgreSQL 虽分别在 5.0 和 8.0 版本中增加了数据库分区特性, 但功能不够强大. 针对上述产品存在的问题, 提出了自行设计的数据分片中间件 UAPDS, 并且在国网业务应用系统中取得了良好的效果.

## 1 UAPDS简介

UAPDS 是一种数据库水平扩展解决方案. 针对业务系统的需求多样化, 形成了扩展性强、对上层应用影响小以及对各种数据库有更好更通用支持的具有业务普适性的数据分片中间件, 对整个应用程序完全透明, 部署拓扑图见图 1.

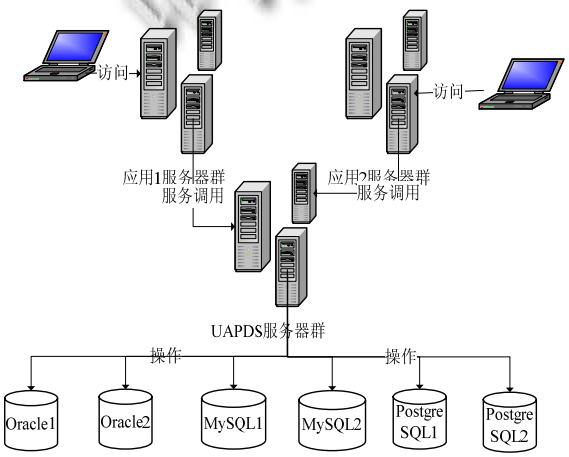


图 1 UAPDS 部署拓扑图

UAPDS 是一个基于 java 开发的独立部署模式的中间件, 应用请求访问中间件后, 经由中间件的解析、路由、执行操作,按照一定的业务规则将数据拆分到数据库中. UAPDS 支持同构数据库数据拆分, 也支持异构数据库数据拆分, 而且还支持读写分离、单库分表、多库分表等多种拆分解决方案. 为了达到高可用性和高可靠性, UAPDS 支持集群配置和负载均衡策略.

## 2 UAPDS设计与实现

UAPDS 引入了集群(router)、分区(partition)和分片(shard)的概念, 集群对应一个常规数据库, 是统一的管理入口, 一个集群包含多个分区; 分区对应一个逻辑分区表, 一个分区包含多个分片; 分片对应一张数据表. 架构图见图 2.

从图中可以看出 UAPDS 的设计原理如下:

(1)传入的 SQL 语句通过语法解释器 jsqlparser 进行语法解析, 最终生成 java 类层次结构.

(2)集群获取解析后的 SQL, 然后调用分区的 SQL 路由规则确定是在哪个分区上执行; 接下来 SQL 转移到分区后, 调用分片的 SQL 路由规则确定在哪些分片上执行 SQL.

(3)调用 JDBC 接口执行 SQL, 根据各分片执行的结果进行归并处理.

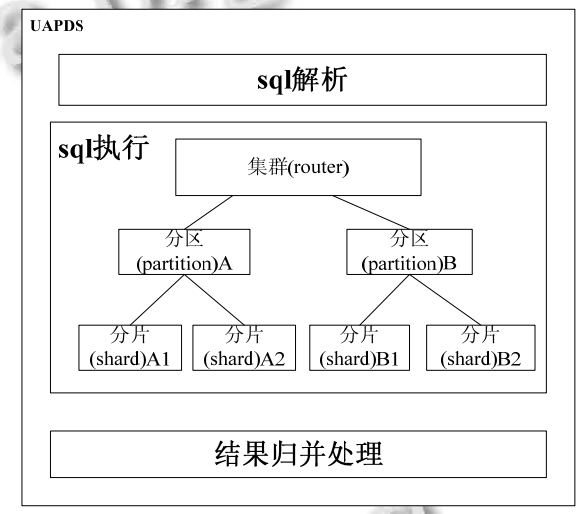


图 2 UAPDS 架构图

### 2.1 拆分方案

UAPDS 支持主从模式和分片模式. 主从模式即读写分离, 分片后的数据完全一致; 分片模式又分为单库分表和多库分表, 分片后的数据是不一致的, 各分片的数据合集才是一个完整的记录.

#### 2.1.1 读写分离策略

读写分离是将数据库的写操作集中到一个数据库中, 而一些读操作则分解到其它数据库中. 为了确保数据库的稳定性以及对于某些业务是读多写少的情况, 则需对数据库进行读写分离. 通过读写分离就可以将数据库的处理压力分解到多个数据库上, 从而提升数据的处理能力.

鉴于服务器的处理能力存在一定的差异, 负载均衡策略采取加权最小连接算法. 假设有一组服务器 $S=\{S_1,S_2,\ldots,S_n\}$, $W(S_k)$为分配给服务器 $S_k$ 的权值, $C(S_k)$为服务器的当前连接数, 当前新连接请求发送到服务

器 m 由如下条件决定 $C(S_m)/W(S_m)=\min\{C(S_k)/W(S_k)\}(k=1,2,\ldots,n)$.

读写分离执行流程如下:

(1)应用发起请求后, 首先获取分区模式, 当 Model=1 时进行读写分离模式.

(2)写操作时, 中间件找到配置为 writeable=true 对应的数据源, 然后将数据插入到数据库中.

(3)将写库中的数据复制到读库中. 考虑到数据复制的数据一致性, 引入了 Quorum 算法. 假设 N 为数据复制的节点数量, R 为成功读操作的最小节点数, W 为成功写操作的最小节点数, 如果 W+R>N,写操作和读操作请求节点集会存在重叠, 这样可以保证强一致性[2].

(4)读操作时, writeable=false 对应为读数据库, 读库对应的服务器配置可能存在一定的差异性, 通过参数 readWeight 设置读权重值, 数值越大默认被分配的读取任务越大.

(5)为了合理分配读操作, 通过负载均衡策略算法重新命中读数据库.

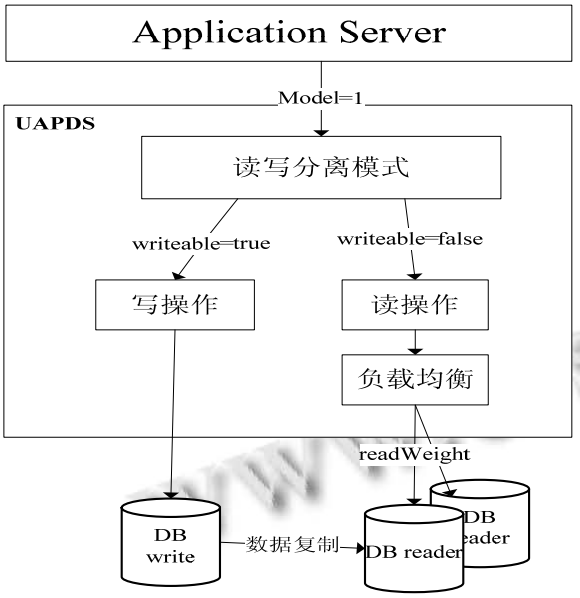


图 3　读写分离流程图

2.1.2 单库分表策略

单库分表是将原本存储于一个数据表按照一定的规则拆分成不同的子表存储到同一个数据库中. 一张表的数据达到一定量时, 数据操作性能急剧下降, 通过单库分表大大提高了数据查询速率.

单库分表执行流程如下:

(1)应用发起请求后, 首先获取分区模式, 当 Model=2 时进行单库分表模式.

(2)分区规则负责建立分区逻辑表, 定义规则实现类和表名, 确定分区范围, 所有包含逻辑表的 SQL 语句都会路由到本分区.

(3)在划定的分区范围中, 分片规则负责定义分片绑定的数据源, 建立分区逻辑表与物理表之间的关联关系, 确定分片范围.

(4)建立数据连接, 对数据库的不同表进行读写操作.

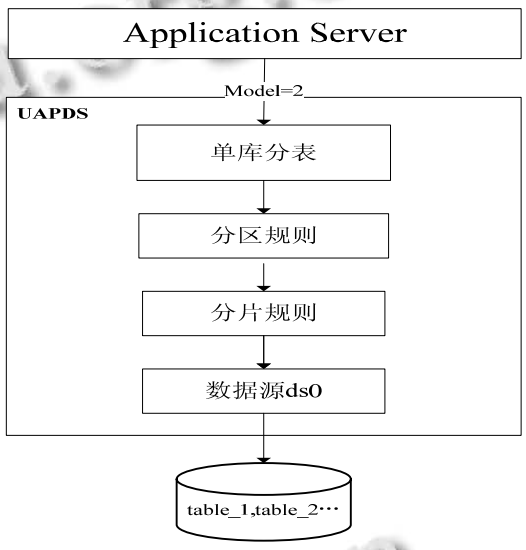


图 4　单库分表流程图

2.1.3 多库分表策略

多库分表是将原本存储于一个数据库的数据表按照一定的规则拆分到不同的数据库中. 单库分表虽然解决了数据访问性能问题, 但数据仍然在一个数据库中, 不管怎样升级硬件资源, 单台服务器的资源如 CPU、内存、网络 IO 资源、连接数等总是有限的. 通过多库分表有效的解决了库中表与表之间的 IO 争夺.

多库分表与单库分表流程基本相同, 如图 5 所示, 只是分片规则分别绑定到不同数据库的数据源, 读写操作均在不同的数据库中完成.

集群中 router 可以嵌套, 因此再复杂的集群模式都可以构建出来. 在实际场景中, 依据项目的复杂程度通过读写分离、单库分表和多库分表各种组合拆分方案进行拆分, 使系统性能得到最大提升.

2.2 主键生成策略

数据拆分到不同的库或不同的表以后, 数据库自

身的主键生成策略已经无法确保主键的全局唯一性. UAPDS 中间件提供了两种生成策略: 第一种是采用 UUID 生成 32 位的唯一主键, 值得一提的是该策略无法保证主键的连续性. 第二种是通过维护中间表 key_table 的主键生成策略, key_table 表结构中维护两个字段 tableName 和 startId, tableName 表示对数据拆分的逻辑表名, startId 表示对应表的起始值, 每次需要为表生成主键 ID 时就从 key_table 中获取对应表的 startId 值, 并且将 startId 的值与步长值(默认为 1, 可以自行制定)求和后更新 key_table 中以便下次使用.

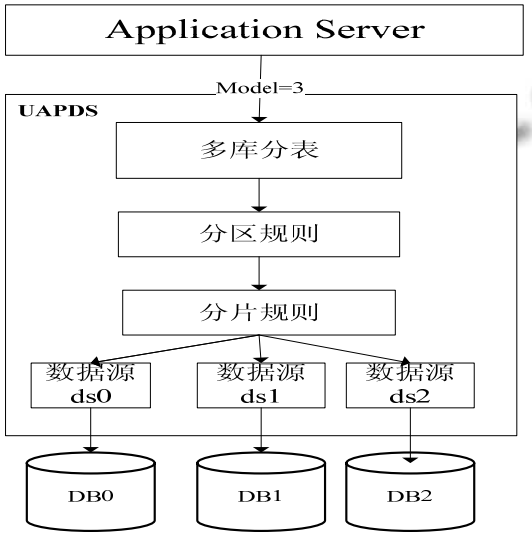


图 5 多库分表流程图

### 2.3 路由归并策略

数据拆分到不同的数据库或不同的数据表中, 查询时就会涉及到多个库或表,此时不能简单的对每个数据库执行查询, 还需要对所有查询结果路由归并. 下面以一个典型的例子说明路由归并策略.

以单库分表的聚合函数 avg 为例, 假设存在三个结构相同的物理表, 表名分别为 tableName_1, tableName_2,tableName_3, 同时假设对表的 score 字段求平均值, sql 语句为 select avg(score) from tableName. 路由归并策略如下:

(1)获取到查询 sql 后, 进行 sql 语法解析替换操作, 替换后对应的 sql 语句为 select count(score), sum(score) from tableName_x(x 分别为 1,2,3).

(2)分别执行子查询语句, 得到子查询结果 rs_1,rs_2,rs_3.

(3)归并查询结果, 获取 rs_1,rs_2,rs_3 中的 score 的总和和总记录数, 然后求平均数.

UAPDS 中间件可以实现正确的路由归并, 并对聚合统计等函数有很好的支持, 对用户是完全透明的, 无需关心细节, 用户只需查询逻辑表信息便可得到正确的归并结果.

### 2.4 分布式事务策略

按照规则分库分表后要保证在不同的库或不同的表的数据的完整性和一致性, 需要采用分布式事务机制.

#### 2.4.1 事务处理

事务是并发控制的基本逻辑单位. 所谓“事务”是一系列有单个用户或应用程序提交的数据库操作, 这些操作是一个不可分隔的整体. 为了保证数据库的完整性(正确性), 数据库系统必须维护事务的以下特性(简称 ACID): 原子性(Atomicity), 一致性(Consistency), 隔离性(Isolation)和持久性(Durability)[3].

#### 2.4.2 分布式事务

分布式事务是指事务的参与者、支持事务的服务器、资源服务器以及事务管理器分别位于不同的分布式系统的不同节点之上. 对于传统的单点上事务, 所有的事情都在一台机器上完成, 而在分布式事务中, 会有多个节点参与[4].

JOTM (Java Open Transaction Manager) 是 ObjectWeb 的一个开源 JTA(Java Transaction API)实现, 它本身也是开源应用程序服务器 JOnAS(Java Open Application Server)的一部分, 为其提供 JTA 分布式事务的功能.

中间件通过 JTA 完成分布式事务,使用 JOTM 框架作为底层事务实现. 执行事务流程如下:

(1)通过扩展构建 StandardXADataSource 分布式事务的数据源.

(2)依据 StandardXADataSource 数据源构件分布式事务连接 XAConnection.

(3)使用 JOTM 事务管理器管理事务并启动事务.

(4)执行数据库操作, 成功则提交事务, 否则回滚事务.

### 2.5 规则策略及二次扩展

数据拆分和归并需依托于规则才能实现分片效果, UAPDS 中间件提供了分区规则、分表规则和主键生成器的默认实现. 分区实现了按 sql 语句参数值进行分区匹配的规则; 分片实现了按参数的值进行分片、单库

分表的按主键字段取余和多库分表的按主键字段取余的规则; 主键生成器实现了自增的 int 型主键、自增的 long 型主键和以 UUID 生成字符串型的主键.

规则的默认实现已经能够满足大部分应用, 对于不能满足的业务需求, UAPDS 以接口的形式提供了分区规则接口 PartitionRule, 分片规则接口 ShardRule 和分布式主键获取接口 KeyGenerator, 方便进行二次扩展开发.

### 2.6 非功能特性及限制

(1)数据操作的松耦合性: 并不是所有的表都需要数据切分, 需要切分的数据使用 UAPDS JDBC 进行操作, 不需要切分的数据则使用普通 JDBC 进行操作. 数据库操作架构图如图 6.

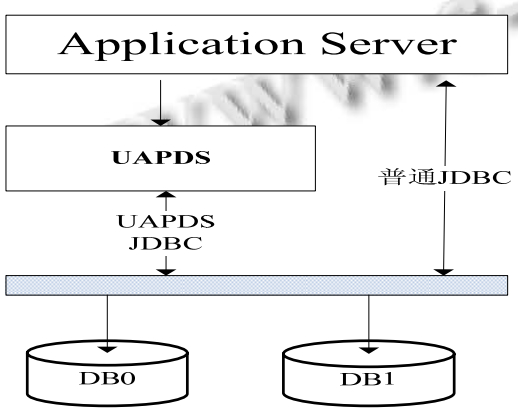


图 6　数据库操作架构图

(2)心跳检测机制: 集群配置时向主机发送 sql 语句, 通过语句的返回结果来判断主库的运行情况, 当检测到主机异常, 则自动切换到备机.

(3)限制: 为了获得对不同数据库切分的通用性, UAPDS 做了如下限制.

①不支持跨分区关联(Join)查询. 由于不同分区的数据库进行 Join, 只能通过内存 Join 或临时表 Join 的方案, 但是不管采用哪种都会带来大量的网络开销、内存开销、CPU 开销、数据一致性等问题. 同时在分库分表前, 需要对数据进行分析, 合理分区, 把需要 Join 的表放在不同的分区中本身就是不合理的. 因此我们在设计时就可以将有主从关联的表放到一个库中或者采取小表复制方案, 通过建立冗余表达到 join 的目的.

②为了体现 UAPDS 对多数据库的支持, 分表后只支持光标分页, 不支持 sql 分页. 不同的数据库采用 sql 分页的语法不一致, 比如 Oracle 的 rownum, mySQL 的 limit, sql server 的 top 函数.

③做分表的字段, 其值不允许被修改. 分表规则字段决定了一个数据的流向, 当分表规则字段的值被修改之后, 就会导致分表规则无法定位此数据.

## 3　应用案例

UAPDS 已成功应用于我司的经济法律管理业务系统(简称: 经法系统)当中. 经法系统以合同管理为核心, 在系统一级部署后, 总部、网省和地市的合同数据都存放在一个数据库中, 合同表 LAW_T_CONTRACT_INFO 每天以十几万的数量增长, 系统处理能力遭遇瓶颈. 在实际应用中通过 UAPDS 中间件将合同表拆分到不同的数据库中.

第一步按照多库分表规则策略以单位信息 ORG_ID 进行水平分割配置, 分割后表名相同为 LAW_T_CONTRACT_INFO, 表结构均相同, 配置规则为合同_n= 合同基本信息{单位 ID: ORG_IDn}(0<n<=count(ORG_ID)).

第二步按照单库分表规则策略以合同创建日期 CREATE_TIME 进行水平分割配置, 分割后表名分别为 LAW_T_CONTRACT_INFO_m(0<m<=12), 表结构相同, 配置规则为合同_n_m=合同基本信息{创建时间: CREATE_TIMEm}.

根据实际情况两次水平扩展以后, 中间件依据单位和月份规则完成了合同信息分放到不同数据库的不同表中, 达到了预期的效果.

通过对合同表 LAW_T_CONTRACT_INFO900 万条数据拆分前后进行性能分析, 拆分之前查询全网每个月合同金额汇总查询响应平均时间为 10.717 秒, 采用 UAPDS 后查询响应品均时间为 4.584 秒. 显著提高了查询效率.

## 4　结语

本文研究并设计了数据分片 UAPDS 中间件, 简要地介绍了 UAPDS 中间件将逻辑数据库表按照读写分离、单库分表、多库分表不同拆分方案拆分到多个物理数据库或物理数据库的多张表中, 实现了数据库的水平扩展. 同时提供分布式主键、分区规则、分片规则、函数统计、合并查询、分布式事务和二次扩展等功能. 从而有效的解决了数据量大、并发访问效率

低的问题.

## 参考文献


1 李文杰,周剑华.分布式应用层中间件的设计[学位论文].武汉:武汉科技大学,2011.

2 Kroenke D.数据库原理.北京:清华大学出版社,2008:2–3.

3 曾宪杰.大型网站系统与 Java 中间件实践.北京:电子工业出版社,2014.

4 邵佩英.分布式数据库系统及其应用.科学出版社,2000

5 侯佳佳,乔云华,卜建国,王海丹.基于分布式数据库数据处理的研究[学位论文].北京:北京机械工业自动化研究所,2013.

6 黄涛.基于 J2EE 的分布式事务处理的研究.浙江海洋学院学报(自然科学版), 2004,(9):261–264.

7 Adabala S. Interfacing wide-area network computing and cluster management software: DQS and PBS via PUNCH. Proc. of the 9th IEEE International on High Performance Distributed Computing. 2000.

8 Kadamuddi D. Clustering algorithm for parallelizing software systems in multi processors environment. IEEE Trans. on Software Engineering, 2000.

9 Buyya R. High performance cluster computing. Architectures and Systems, 1999.

10 Zou YQ, Liu J, Wang SC, Zha L, Xu ZW. CCIndex: A complemental clustering index on distributed ordered tables for multi– dimensional range queries. In: Ding C, Shao ZY, Zheng R, eds. Proc. of the NPC. Zhengzhou. Springer-Verlag. 2010. 247–261.