海量跨域数据交换平台①

孔晓昀[1], 王晓聪[2], 陈正奎[2]

[1](浙江省电力公司 科技信通部, 杭州 310007)
[2](浙江省地方税务局 信息化管理处, 杭州 310007)

摘 要: 不同系统之间的数据交换一般通过数据交换平台来实现, 但面对不同部门之间海量数据交换时, 由于系统的不一致, 效率成为瓶颈. 针对如何处理跨域海量数据交换和高并发请求, 笔者对构建高效率的数据交换平台进行了研究, 提出了通过三层数据处理引擎来统一调度用户请求, 解决了系统内外部数据流转的问题. 然后研究了内存块复制技术和调度算法, 结果表明此技术能大幅提高海量数据的处理速度, 并能动态响应高并发的数据请求.

关键词: 海量数据; 高并发; 数据交换平台; OCI; BIND 数算法; 动态消息调度

Data Platform of Big Cross-domain Data Interchange

KONG Xiao-Yun[1], WANG Xiao-Cong[2], CHEN Zheng-Kui[2]

[1](Department of Science and Information, State Grid Zhejiang Electric Power Company, Hangzhou 310007, China)
[2](Department of Information Technology, Zhejiang Local Taxation Bureau, Hangzhou 310007, China)

Abstract: The data platform is generally used to interchange information between different systems, but the efficiency is the bottleneck due to big data. To deal with the big data interchange and parallel requests, we design a 3-layer data engine for managing the user requests. Therefore, a state-of-art technology of memory replication and algorithm is developed to promote the data processing speed as well as quick response to parallel requests.

Key words: big data; parallel requests; data platform; OCI; BIND algorithm; dynamic message control

随着大数据时代的到来, 信息系统之间海量数据交换和高并发用户请求的处理面临新的挑战, 这里涉及到一个关键性的问题: 如何设计高效率的数据交换平台来快速处理和响应高并发请求? 这也是本文的研究目标.

笔者经过研究, 通过优化设计软件架构和算法机制, 在充分利用原有硬件的情况下, 大幅提升数据交换平台的性能是完全可行的. 从系统架构层面看, 数据交换平台承担着中间件的角色, 笔者研究的数据交换平台主要包括核心数据引擎、通讯前置程序、监控管理软件、数据交换平台接口等.

1 数据交换平台的整体设计

通常情况下, 传统的数据交换平台和数据库服务器的交互方式分为长连接和短连接, 分别负责处理批量数据和实时数据. 但是由于缺乏统一控制、调度以及架构优化, 此数据交换模式不能很好地应对海量数据处理和高并发请求. 因此, 构建分层数据引擎、优化数据处理算法才能满足需求, 这也是新型数据交换平台的特征. 针对数据交换平台的研究, 其目的是处理海量数据和快速响应用户的高并发请求

1.1 数据引擎模型

新型数据交换平台的核心是数据引擎(如图 1 所示), 主要由控制台 CONSOLE、联网调度 Scheduler、服务 TRS Service 三个模块组成. 内部用户通过客户端程序包访问 CONSOLE, 外部用户通过联网前置机访问 Scheduler 并隐性调用 CONSOLE, CONSOLE 对预设的静态路由或动态路由进行解释, 并控制整个交易

① 收稿时间:2014-04-09;收到修改稿时间:2014-05-26

流转和数据处理过程, 终通过 TRS Service 对数据库或文件操作, 通过 Scheduler 对联网单位服务调用完成相应的数据访问和操作.

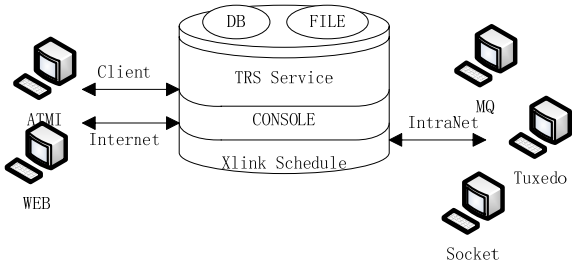

图 1　数据交换平台模型图

1.2 数据引擎架构

从数据流转的角度, 交互平台的数据引擎架构可分为数据接入层、数据控制层和数据处理层三个层次(如图 2 所示).

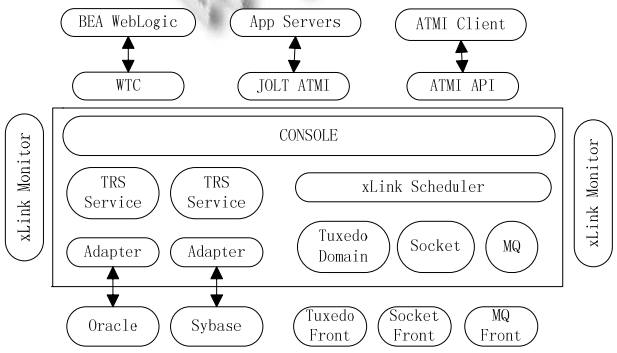

图 2　数据交换平台数据引擎架构图

数据接入层: 根据接入对象的不同可分为内部用户和外部门用户二种对象, 对于内部用户, 数据交换平台针对不同类型的应用系统提供相匹配的客户端程序包; 对于外部门用户, 数据交换平台可针对不同的联网模式或通讯方式提供不同的联网前置机服务程序.

数据控制层: 由交易控制台 CONSOLE 和联网调度服务 Scheduler 组成. 交易控制台 CONSOLE 具有交易合法性检查、数据路由计算、报文翻译等功能, 并负责交易服务的调度和管理; 联网调度服务 Scheduler 是交易控制台和联网前置机之间的桥梁, 调度和管理与外部的数据交换, 并作为特殊的数据源形式存在, 其主要职责是从外部数据源获取数据或响应外部数据请求.

数据处理层: 由交易服务 TRS Service 组成, 每一个交易服务对应一个数据源, 这些数据源既可以是本地数据源也可以是远程数据源, 既可以是关系型数据库也可以是文件, 交易服务通过调用预先设定的交易处理方法来完成数据处理. 对于从外部数据源获取数据的情形, Scheduler 作为特殊的数据源形式被调用.

在分布式运算环境下实现数据交换与信息共享目的概括起来不外乎以下二种情形: 内部不同或异构数据库之间的信息共享、与外单位或部门之间的数据交换. 针对上述二种不同情形, 新型数据交换平台总结并提炼其共性及个性差异, 既保证共性需求的充分展示, 又做到个性化需求的充分满足. 以业界成熟的消息中间件技术为基础框架, 针对分布式运算和联网业务要求, 对中间交互过程和数据对象处理进行了合理、有效的封装, 构建完成了具有高可用性、高自由度的数据交换平台产品.

新型数据交换平台具备分布式事务处理和应用通信功能, 并提供数据、报文、文件等多种数据交换服务, 能够快速建立跨多个硬件平台、数据库和操作系统的可交互操作的数据交换与信息共享平台. 新型数据交换平台提供了一个开放的环境, 支持多样的客户机、数据库、网络和通讯协议, 整个通讯交互对用户而言完全透明, 用户可与本地数据源一样调用远程数据源, 彻底摆脱通讯对用户的困扰, 真正实现对远程数据源的透明无缝调用.

目前, 数据交换平台支持 Oracle、Sybase、SQL Server 等主流数据库产品, 并通过可视化配置实现与数据库的交互, 相关业务逻辑以数据库存储过程或 SQL 语句的形式实现, 使得数据交换的共性与业务逻辑的个性有机结合, 快速响应内部数据集成和外部数据交换的需求.

2　数据交换平台的异构数据

内外部数据往往独立存在, 如何有效合理的整合数据资源是信息决策者们必将面临的问题. 新型数据交换平台提供了的解决方案, 可在不改造或极少改动原有系统的前提下, 使新的集成应用与原有系统数据源之间进行交互, 实现用户对数据的透明无缝调用.

2.1 内部数据集成

以交易和静态数据路由的形式提供用户定义和配置内部数据集成的方法, CONSOLE 为每次请求根据

数据报特征计算与之相匹配的数据路由, 并按照数据路由的定义次序依次调用 TRS Server, 从而实现不同数据库间的数据与信息共享. 对于存在二义性的数据请求, 新型数据交换平台还提供了动态路由的方法, 供请求者在特定的条件下使用.

从实际应用的角度, 还存在内部数据集成时需要利用外部数据源作为补充. Scheduler 作为特殊的 TRS Service 以外部数据源的形式供 CONSOLE 调用, 对请求者而言, 内部数据源与外部数据源是完全等同的. 当然, 如果请求者需要展示或利用上述数据, 平台将根据预设的格式以报文的形式返回请求者.

2.2 外部数据交换

外部数据交换在实际应用中, 每个单位或部门从安全的角度考虑, 都会设置相应的联网前置机作为中转, 而这些前置机需要根据双方商定的通讯方式编制相应的前置程度, 因此, 实施难度会加大, 实施进度也会拉长. 新型数据交换平台根据目前常见的通讯方式分别提供相应的通用前置程序, 以满足用户的针对性的需求.

无论是内部请求者还是外部请求者, Scheduler 是联接对方的桥梁, 负责由 CONSOLE 转入的数据交换请求的转发并接收外单位响应以及向 CONSOLE 转发来自联网前置机的数据交换请求. 从数据交换请求直到最后的处理完成, 这个过程中所涉及的 CONSOLE、Scheduler、TRS Service、前置机, 新型数据交换平台会根据系统配置库的设置, 以数据路由的方式决定调用相对应的 TRS Service 或前置机, 并对整个过程进行管理和监控. 对用户而言, 其过程是透明的, 可以将 Scheduler 作为数据源看待, 真正实现外部数据交换的“零”编程的目标.

外部数据交换必然涉及请求者和响应者二种身份, 作为请求者无论是从对方获取数据还是向对方发送数据, 一定是以交易的主动形态出现, 而作为响应者无论是接收对方数据还是返回数据给对方, 也一定是以交易的被动形态出现的.

2.3 报文处理

新型数据交换平台通过数据报文的形式传递信息, 是贯穿整个交易的主线索, 其内容由报头和报体二部分组成. 报头部分存放用户信息、路由信息、交易控制信息及出错信息等内容, 是长度为 256 字节的字符串, 对用户屏蔽. 报体部分内容用户可针对每个具体交易进行自由定义, 系统不加以限制.

报文类型支持字符串、二进制流、文件、图像、音频等内容, 在传输过程中均采用二进制传输方式, 以保证报文在传输过程中不被篡改.

报文格式支持定长报文、带分隔符的不定长报文、位图报文三种格式. 对于不定长报文, 用户可分别为每个交易指定各自的分隔符, 如不指定平台将采用默认分隔符. 分隔符分为字段分隔符、记录分隔符和结果集分隔符三种类型, 但在同一个交易中这三种分隔符不能重复, 否则平台将无法分辨和正确处理报文.

内部报文用于内部数据集成的报文被称为内部报文, 一般情况下, 存放相应存储过程的参数值或 SQL 语句的变量值. 对于外部数据交换, 一般情况下, 联网双方会预先约定相应的数据报文格式及内容, 上述报文被称为外部报文. 外部报文不能直接被数据交换平台所利用, 而是通过报文翻译器转换为内部报文后再进行数据处理; 同样的, 数据处理完成后再通过报文翻译器转换为外部报文后返回给外部请求者.

由于外部数据交换的需要, 新型数据交换平台提供了用于内部报文与外部报文相互转换的报文翻译器. 按照预先设定的报文转换规则, 在接收到外部请求或外部应答时, 将外部报文格式转换为内部报文格式; 相反的, 在向外部发出请求或应答外部请求时, 将内部报文格式转换为外部报文格式. 报文转换规则主要包含报文转换形式、字段对应关系、字段长度与格式处理三部分内容, 每一种转换规则包括内部报文转为外部报文和外部报文转为内部报文二种定义内容.

为便于跟踪和管理, 数据交换平台为每次的请求报文分配一个唯一的服务流水号作为服务线索. 服务流水号由 CONSOLE 产生, 长度为 18 个字节的数字字符串, 在 24 小时内每次请求的服务流水号不会重复. 服务器流水号由服务器号, 进程号, 时分秒毫秒和序列组成.

2.4 保持数据一致性

如何保持不同数据库之间特别是异构数据库之间的数据一致性, 到目前为止仍没有一个完美的解决方法. 但通常的做法有二种: 第一种, 采用二步提交方法, 也就是采用所有关系数据库厂商共同遵守的 XA 协议通过中间产品来实现二步提交的功能, 虽然二步提交基本可保证数据库之间的数据一致性, 但是此方法会消耗大量的系统资源, 效率较低, 同时还要求联

网单位也装配相同的中间件产品, 灵活性和可操作性较差, 由此带来的负面影响也是显而易见的; 第二种, 采用应用控制方法, 也就是通过应用程序来保证数据库之间的数据一致性, 虽然此方法具有良好的灵活性、可操作性, 同时也能提高系统的处理效率, 但是会增加应用编程量.

由此可见, 上述二种方法基本都能保证数据库之间的数据一致性, 但是在某些极端情况下均不能完全保证数据一致. 这就要求我们综合考虑数据处理性能、通用性、灵活性、可操作性等几方面因素, 既能基本保持数据一致, 又具有良好的数据处理能力和通用性. 经过长期的应用实践比较, 采用第二种方法是相对比较理想的解决方法, 数据交换平台因此也采用第二种方法来实现异构数据库之间的数据一致的要求.

数据交换平台提供了两种通过应用来保证数据一致性的解决方案: 交易冲正机制和数据同步调用. 所谓交易冲正机制, 通过编程手段并且应用系统来保证数据一致性. 也就是, 根据预先制定的数据更新逻辑, 为每个数据更新交易编制与其相对应的数据更新恢复程序, 当某个数据更新交易出现异常时, 系统启动与其相对应的数据更新恢复交易, 从而保持同一或不同数据库之间的数据一致性. 所谓数据同步调用, 就是按预定的数据处理顺序依次更新数据, 全部数据更新成功后从最后一步开始逆向提交, 也就是, 上一步数据更新提交依赖于下一步数据更新成功与否.

上述二种方案各自有其优缺点, 交易冲正机制系统资源占用少、处理效率高, 但会增加 SQL 或存储过程的编程工作量, 对某些复杂的数据处理并不一定能恢复原状; 数据同步调用不会额外增加编程工作量, 不管数据处理如何复杂均能恢复原状, 但由于是同步调用, 一次处理可能会占用很大一部分系统资源, 在某些特殊情况下会影响平台的服务性能. 在应用过程中, 交易冲正机制和数据同步调用应根据实际情况结合使用. 根据实践经验一般采用: 简单数据更新处理可采用交易冲正机制, 复杂的数据更新处理应采用数据同步调用方法.

2.5 数据交换平台的接口设计及实现

设计思路: 利用 Java 的反射机制解析对象, 获取 WEB 服务请求对象的数据, 组装成数据交换平台的数据报文; 收到数据交换平台应答后, 将数据交换平台的数据报文设置为 WEB 服务应答对象的数据. 数据报文的生成和解释过程和方法对 WEB 服务透明.

接口描述: 函数: public Object TFDInterface (Object in, Object out)

说明: 定义了返回为 Object 类型的函数 TFD Intereface (函数名暂定), 参数 in 为传入的包含数据的对象的实例, 参数 out 为最终返回的对象. 通过 Java 反射机制, 获取传入的 in 对象的成员变量的类型等, 并获取 in 对象中各个成员变量的值. 通过 Java 反射机制, 解析 out 的对象的成员变量, 将数据交换平台返回的数据为 out 对象赋值, 最终返回 out.

代码示例:

输入参数: User 对象的 Java 类, 包含了 UserName, Sex, Age 三个成员变量, 此 User 对象为传入的 in 参数.

输出参数: 返回对象 UserInfo, 包含 Address, Company, Education 三个成员变量, UserInfo 对象为 out 参数.

对 User 进行简单的实例化并赋值:

```
User user = new User();
user.setUserName("张三");
user.setSex("男");
user.setAge(20);
```

对 UserInfo 实例化:

```
UserInfo userInfo = new UserInfo();
```

调用接口函数:

```
TFDInterface((Object)user, (Object)userInfo)
```

在 TFDInterface 函数的实现方法里面就能获取到“张三”,“男”, “20”的数据, 按照约定的数据组织规则将其传入数据交换平台, 在收到应答后, 根据数据交换平台返回的列属性按照一定的规则进行分解, 逐一 Set 到(Object)userInfo 中并返回给调用者.

3 数据交换平台的数据处理

在系统一体化的背景下, 越来越多的数据集中到中央或省级部门. 譬如某省级部门的数据库, 仅核心表的数据记录就有 15 多亿条, 并且每月以数百 GB 的速度飞速增长. 为了平稳有序地管理好海量数据, 数据引擎采用了内存块复制和最优 BIND 数算法和数据库进行交互.

3.1 OCI 内存块复制

数据交换平台采用基于 OCI(Oracle Call Interface,

数据库读写调用接口)和内存块复制的大数据量调度技术, 提高数据读写处理速度(性能比常规数据库客户端工具提高 30%以上), 有效地保证了与外部门联网的海量数据处理性能.

通常情况下, 信息交换平台的数据交换会为数据包的存放而频繁地进行内存的分配释放, 时间越长, 系统内存碎片就会越多, 逐渐影响执行效率和稳定性, 同时也存在内存泄漏的隐患. 数据交换平台采用了一次性分配足够的内存, 以共享内存的形式为所有服务进程提供内存空间, 而不同进程之间的数据交换则通过系统缓冲池完成交互. 这种模式既可以避免上述的缺点, 又可以提高系统运行的稳定性和效率. 系统缓冲池(如图 3 所示)主要由三部分组成: 控制信息区、请求缓冲池、应答缓冲池.

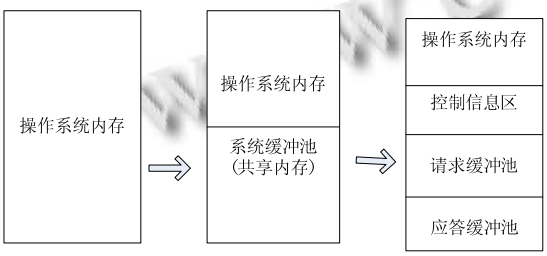

图 3 OCI 内存块复制模型图

系统缓冲池机制的核心是由缓冲阀值和数据块属性组成. 其中前者主要存放了内存页数量、大小、请求缓冲池属性、应答缓冲池属性等信息. 而后者主要存放了内存大小、数据编码、索引、状态等信息, 特定的守护进程负责管理系统缓冲池.

其中数据包以二维矩阵的形式存在于请求缓冲池中, 仅存放数据内容, 而控制信息区存放着属性信息. 进程均可以访问请求缓冲池, 而不同进程间的信息交换内容是缓冲池的编号, 这样可以提高系统的运行效率.

应答缓冲池采用的机制和请求缓冲池类似, 根据相关的规则, 应答缓冲池中的数据信息可以移动到请求缓冲池中, 再提供给下一个进程, 不同之处在于其存放的内容是应答数据包.

首个 CONSOLE 服务进程启动的时候, 会 FORK 生成守护进程(如图 4 所示), 其主要功能是负责建立、轮询检查、异常数据块清理等工作, 从而保障缓冲池的正常运行.

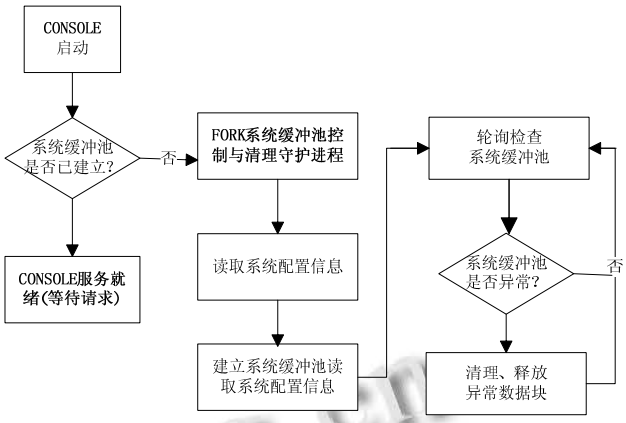

图 4 OCI 内存块复制流程图

3.2 最优 BIND 数算法

数据库一般会提供块操作机制, 外部系统以定额记录集为单位与数据库进行数据交换, 最优 BIND 数通过减少和数据库的交互次数, 控制数据交换平台以最优的方式向数据库层提交或获取数据, 从而提高数据库的处理效率(如图 5 所示). 数据交换平台一般预设了三个参数: BIND 数最小值、空间占用最大值、因子, 这些参数也可以根据实践进行二次调优.

数据处理流程原理: 当控制台接收到新的数据包, 首先启动解析程序, 随后生成请求数据矩阵, 下一个处理进程将获得数据矩阵编号. 计算最优 BIND 数将依据于该次处理请求的缓冲位置、记录长度、记录数量等信息. 最后把请求数据矩阵关联到具体的 SQL 语句或存储过程, 由客户端复制到数据库的缓冲池中. 当数据库服务器收到响应后, 根据收到的信息来计算最优 BIND 数, 并将数据结果集以矩阵的形式映射到应答缓冲池中.

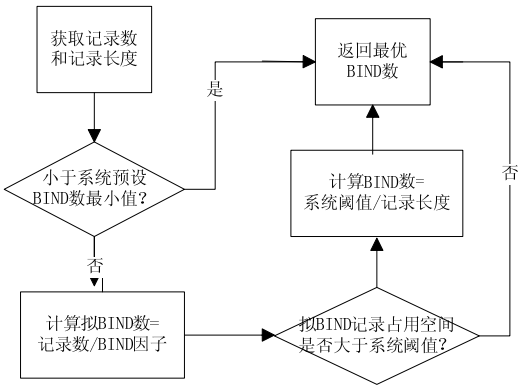

图 5 最优 BIND 数算法模型

4 数据交换平台的高并发处理

在高并发请求的信息系统中, 既有几十毫秒的实时数据处理, 也有长达几十分钟的批量数据处理, 如何保证高效有序的执行是衡量系统是否稳定可靠的重要标志.

为此, 数据交换平台借鉴了传统 CPU 的优先级调度算法和时间片轮转算法, 重新构建了先到先得算法与服务调度相结合的动态选择消息调度模式, 极大地缓解了外部高并发请求对数据库服务器的压力.

数据交换平台按照请求性质分类预设了优先级, 并根据系统实际运行的情况动态调整优先级(如图 6 所示). 当数据交换平台接收到请求后, 按从高到低的优先级顺序重组队列, 并按照时间片轮转算法, 依次轮转用户的服务请求.

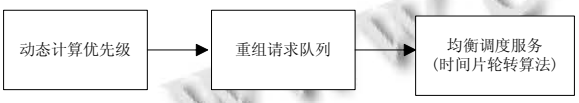

图 6　用户请求排队流程

优先级轮转算法包含了两种算法: 动态优先级算法和时间片轮转算法. 该算法的前提和基础是动态优先级算法, 而时间片轮转算法是补充, 它决定了在相同权重下哪个请求先被处理.

4.1 动态优先级算法

动态优先级算法可以避免静态优先级无法感知后台服务状况的缺点, 该算法的核心在于保证处理时间较短的请求先被执行, 也可以兼顾处理时间较长的请求. 它的算法涉及到三个决定性因素: 静态优先级、后台服务处理时间因子、请求时间因子, 其计算公式如下:

$$P = f(x) \times g(y) \times h(z) \quad (1)$$

其中, 请求时间因子 g 按照先来先到的原则进行分配, f 表示静态优先级, h 表示后台服务处理时间因子.

4.2 时间片轮转算法

由于时间片轮转算法是可以预知的, 在优先级相同的前提下, 每个请求被选择的机会均等(如图 7 所示). 当一个队列中每个成员都具有一样的权重, 时间片轮转算法会在每个成员中进行顺序选择. 而新来的请求将被轮流发给队列中的下一个副本. 在实际应用中, 此时间片轮转算法可以扩展为加权轮转算法, 它的优点在于能感知服务资源的负载变化, 然后根据服务资源的负载情况来计算权值, 起到均衡调度的作用.

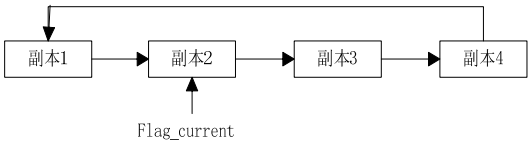

图 7　时间片轮转算法模型图

算法实现(核心伪代码如下)

输入: 副本队列

输出: 选定的副本

初始化所有副本, 令 Flag_current 指向队列头, 将所有副本的 Flagsleep 和 Flagblock 均置为 False, 将 Temp_flag 和 Sleep_flag 置为 False.

```
1 Repeat
2 If Flag_current ≠ 0
3 将 Flag_current 保存到 Temp_flag 中
4 If Flagblock = True
5 将 Flag_current 置为顺序的下一副本值
6 else
7 If Flagsleep ＝ false
8 将 Sleep_flag 标记置为 false
9 将 Flag_current 置为顺序的下一副本值
10 返回副本, 派发请求
11 else
12 Flag_current 指向副本的 Flagsleep 标记置为 false
13 将 Sleep_flag 标记置为 True
14 将 Flag_current 置为顺序的下一副本值
15 else
16 返回 member_not_found 异常
17 直到 Flag_current = Temp_flag
18 If Sleep_flag = True
19 转向 Repeat, 重复轮转
20 else
21 wait (block_time)
22 转向 Repeat, 重复轮转
```

以上描述了基本的时间片轮转算法, 以循环队列的方式组织. 通过 Flag_Current 表示本次应选择的副本, 为后端监测定义 3 种类型的消息: Messagesleep、Messageblock、Messageunblock, 并为每个副本关联 2 个阀值标记 Flagsleep 和 Flagblock. 其中 Messagesleep

消息对应 Flagsleep 标记, 表示后台服务资源处于轻微繁忙状态, 需要睡眠一段时间. Messageblock 消息对应 Flagblock 标记, 表明后台资源处于非常繁忙状态, 需要阻塞后续的请求派发. Messageunblock 表示后台服务资源的繁忙程度已经下降, 需要解除阻塞. 设定 Temp_flag 保存轮转的起点, 设定 Sleep_flag 保存 Flagsleep 标记值的修改, Block_time 记录系统派发请求的等待时间.

数据交换平台根据请求的并发情况通过优化算法动态地调整数据库有效连接数, 当连接数达到系统设定的阈值时将不再增加连接数, 后续请求将以FIFO的形式排除等待, 当排除等待的请求数超过系统设定的最高值时, 新的请求将被拒绝, 直到有空闲位置.

5 实践结果

在笔者参与的某大型软件中, 数据交换平台就在其中承担了中间件的重要角色. 其中, 底层数据库有数千张表, 数据记录达到几十亿级别, 每天和十几个外部门进行联网数据交换, 数据交换频率高且量很大.

原先在没有数据交换平台的情况下, 海量数据处理起来非常慢, 在高峰繁忙期的时候, 数据库读写经常成为系统瓶颈; 而且用户请求一旦多起来, 系统响应也非常慢, 经常遇到过了 10 多秒, 用户请求还得不到响应的情况.

在新型的数据交换平台上线并平稳运行一段时间后, 由于海量数据提交和并发请求都经过了设计和算法优化. 原来高峰期数据库繁忙的场面已经不存在了, 而且用户的请求响应基本上都能及时返回. 在主流的硬件配置下, 笔者设计的单个数据交换平台能同时处理上万的高并发请求, 并且平均处理时间控制在 1 秒以内. 如果考虑分布式部署, 数据交换平台的处理能力将会成倍提升.

6 结语

因此, 在海量数据和高并发的软件应用场景中, 采用数据交换平台作为中间件, 利用其强大的动态调度、数据处理机制以及良好的扩展性, 能够较好地满足海量数据处理和高并发的用户请求.

参考文献

1 维克托迈尔-舍恩伯格,肯尼斯库克耶.大数据时代.杭州:浙江人民出版社,2013.
2 古凌岚.一种新的信息交互平台研究与设计.计算机工程与设计,2005,26(11):3103–3105.
3 杨传辉.大规模分布式存储系统:原理解析与架构实战.北京:机械工业出版社,2013:101–129.
4 王春新,韩儒博,徐孟春.一种基于 JCA 的数据交换架构.微计算机信息,2006,22(3):136–138.
5 Cormen TH.算法导论.北京:机械工业出版社,2013.
6 李华植.海量数据解决方案.北京:电子工业出版社,2011.
7 李良,柴毅,王道斌.基于.NET 的 Oracle BLOB 数据高效存取方法.计算机工程,2008,34(20):64–65.
8 Loudon K.算法精解.北京:机械工业出版社,2012.