Research on the Causes and Positioning of Message Queue Blocking in Transaction Middleware

Meimei Wu
School of Science
Communication University of China
Beijing, China

Yingkai Wang
Beijing Data Center
China Construction Bank
Beijing, China

***Abstract*—Transaction middleware is widely used in large-scale and key transaction processing area, such as banking, telecommunication and finance. The important feature of the trading system is high real-time, high availability and the ability of rapid recovery when system failure. When the trading system's exceptions which use transaction middleware occur, the common and easy-to-find exception should be the message queue blocking. This article researches the cause of message queue blocking ,combined with the ORACLE TUXEDO, on the base of the basic principles of the transaction middleware, the way of request call and the relationship with the operating system message queue etc,and then proposes an effective method of abnormal server positioning.**

Keywords-transaction middleware; message queue; blocking; positioning

交易中间件消息队列堵塞成因及定位方法的研究

吴梅梅[1]，王英凯[2]
1. 中国传媒大学理学院，北京，中国，100024
2. 中国建设银行北京数据中心，北京，中国，100068

【摘要】交易中间件在银行、电信、金融等大规模关键事务领域中的应用非常广泛，交易系统的重要特点就是实时性高、系统可用性要求高，一旦出现问题要求迅速恢复。使用交易中间件的交易系统出现异常时，比较常见和易于发现的异常即为消息队列阻塞。本文结合在业界占有重要地位的 ORACLE TUXEDO，基于交易中间件的体系结构、请求调用方式以及与操作系统消息队列的关系等基本原理，对 TUXEDO 消息队列阻塞的成因进行了研究，并给出了一种行之有效的异常 server 定位方法。

【关键词】交易中间件；消息队列；阻塞；定位

1 引言

交易中间件提供了一个基础的框架来帮助建立、运行和管理一个 C/S 或多层 C/S 模式的应用。交易中间件的主要标准是 X/OPEN 组织定义的分布式交易处理参考模型。交易中间件理论上相对成熟，功能和性能界定清晰，在银行、电信、金融等大规模关键事务领域中的具有广泛的应用。交易系统的一个重要特点就是实时性高，一旦出现问题要求系统服务迅速恢复。而其常见的异常即为中间件所使用消息队列阻塞引起的交易处理停滞。因此对消息队列阻塞的问题快速定位，对交易系统高可用性实现、缩短故障处理时间具有重要意义。本文基于交易中间件的基本原理，对交易中间件消息队列阻塞的成因进行研究，以期给出一种有效的异常 server 定位方法。

2 交易中间件基本原理

2.1 交易中间件的常见架构

X/Open DTP 模型(1994) [1]包括应用程序(AP)、事务管理器(TM)、资源管理器(RM)、通信资源管理器(CRM)四部分。一般来说，常见的事务管理器(TM)是交易中间件，常见的资源管理器(RM)是数据库，常见的通信资源管理器(CRM)是消息中间件。遵循 DTP 模型对交易中间件的功能定位和接口定义[2]，交易中间件的典型体系结构[3]如图 1。

978-988-19116-9-8/10/$25.00 ©2011 IIEC
FCSA 2011

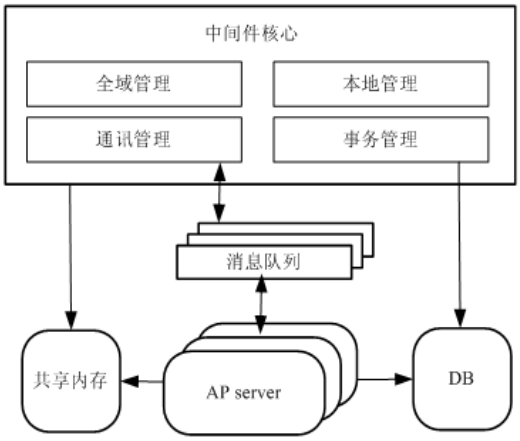

Figure 1. typical architecture of middleware
图 1. 中间件的典型体系结构

其中与本文探讨问题相关的两个部分：

（1）通信管理： 在 TUXEDO 中由 GWTDOMAIN[4]承担此角色，负责在不同的主机与应用进程间进行通讯包的传递和调度。

（2）共享内存和消息队列：作为核心之间、核心与应用进程之间的通信机制，登记核心及应用的运行控制信息，在应用与核心之间传递应用数据。

2.2 交易系统常用的 TUXEDO 通信方式

TUXEDO 提供了丰富的通讯方式，在交易系统中常用的与交易相关的方式为：

(1) 同步调用/应答（tpcall 调用）

Client 将调用的 service name 和交易请求数据作为参数调用 tpcall，TUXEDO 负责将请求数据传递至相应的 service 进行处理。Service 处理完毕，将应答数据传递给 TUXEDO，TUXEDO 将应答数据返回给 tpcall 的调用者。在 service 的处理过程中，tpcall 的调用者处于等待状态。

(2) 异步调用/应答（tpacall/tpgetreply 调用）

Client 将调用的 service name 和交易请求数据作为参数调用 tpacall，TUXEDO 负责将请求数据传递至相应的 service 进行处理。Service 处理完毕，将应答数据传递给 TUXEDO。tpacall 的调用者在调用 tpacall 后不必等待 tpacall 的返回，也就是说在 service 的处理过程中，tpacall 的调用者不处于等待状态。Tpacall 的调用者稍后使用 tpgetreply 取得交易处理结果。

(3) 转发（tpforward 调用）

转发发生在两个 service 间，为异步调用方式。当一个 service A 在处理过程中需调用另外一个 servcie B 时，可采用 tpcall 的方式，此方式需等待 service B 的返回。也可采用转发方式，以 tpforward 方式调用 service B，此时不需等待 service B 的返回，对原始请求的应答也不再由 service A 负责。

2.3 TUXEDO service 调用原理

TUXEDO 将交易请求以消息的形式在消息队列中传递。当 TUXEDO 收到交易请求，由 TUXEDO 的系统进程查询 BBL 中相应 server 对应的消息队列 ID，将消息放入消息队列。TUXEDO server 会不断地从其对应的消息队列中取出消息调用 service 处理函数进行处理。

这个过程是 TUXEDO 基本的请求流转过程，如 service 处理过程中出现问题，使消息不能及时被取出处理，则出现消息队列阻塞现象，使请求流转过程终止或变得缓慢，从而使应用系统对外服务暂停或变慢。

TUXEDO 有 WSL、域通讯等方式，交易请求可由 TUXEDO 在不同的主机间转发。对交易的处理，最终还是由 TUXEDO 通过将请求以消息的形式放入消息队列，由 service 所在的 server 进程从队列中读取消息来实现[5]。

2.3.1 消息队列

操作系统提供了一系列的进程间通信机制，来实现不同进程间的信息传递，消息队列就是其中之一[7]。消息队列创建后由操作系统来维护，所有消息队列中的消息都放在系统内核当中,并且它们都有一个相应的消息队列标识符，作为消息读写时消息队列的唯一标识。进程可读写任意队列中特定的消息，其次序是先进先出(FIFO)[8]，核心负责维护这一适当的次序。另外同一消息队列中的消息，具有消息类型的属性，进程可根据消息类型分别读出各自需要的消息。

2.3.2 Service 调用过程的实现

当 TUXEDO 收到交易请求，由 TUXEDO 的系统进程查询 BBL 中相应 server 对应的消息队列 ID，将消息放入消息队列。

TUXEDO Server 会不停的从自己负责的消息队列中读取消息，进行处理，返回应答。这一循环过程实现了 TUXEDO 交易调用的基本流程。

在编写 TUXEDO 应用的时候，开发者并不需要编写消息队列操作的功能，这一功能由 TUXEDO 来提供。编译 TUXEDO 应用 server 的时候，buildserver[5]会为应用 server 链接 TUXEDO 的库函数，并提供 main 函数在内的消息处理的框架，在消息处理框架的控制下，TUXEDO server 会不断地从消息队列中取出消息，将消息中包括的

交易请求数据取出，并调用应用的 service 函数进行处理，当 service 处理完毕返回时，TUXEDO 的消息处理框架再次从消息队列中读取下一个消息，进入下一个交易处理周期[6]。由此可见，在 service 处理过程中，TUXEDO 的消息处理框架是挂起状态，直到 service 处理完毕，才重新获得进程的控制权。

这个过程是 TUXEDO 基本的请求流转过程，如 service 处理过程中出现问题，使 servcie 返回缓慢或迟迟不能返回，那么消息处理框架迟迟得不到进程的控制权，也就不能及时从消息队列中取出下一个消息进行处理。这种情况下此进程处于忙碌状态，当包含同一 service 的所有进程均处于忙碌状态，调用此 service 的交易请求即无 server 从消息队列中取出，而调用此 service 的请求不断的由 TUXEDO 写入对应消息队列，又没有进程取出，一定时间后，消息队列中的消息越积越多，形成消息队列堵塞。之后对这个 service 的调用因得不到处理，造成交易超时而失败。这个情况使请求流转过程终止或变得缓慢，从而使应用系统对外服务暂停或变慢。

3 故障进程定位

当出现消息队列阻塞时，为定位问题，首要也是关键步骤是根据阻塞的消息队列来分析定位出异常 TUXEDO server。

3.1 Server 进程与消息队列的对应关系

TUXEDO 的 server 与消息队列间有不同的对应关系，常见的有 SSSQ(Single Server Single Queue)和 MSSQ（Multi Server, Single Queue）[9]。

顾名思义，SSSQ 即一个 server 对应一个 Queue，一个消息队列中的消息仅由单一进程负责处理。这种模式的缺点是，当交易量比较大时，server 处理请求过程中，再到达此消息队列的请求得不到及时处理，必须等此 server 处理完当前请求。此过程中即使有空闲 server 也不可以越俎代庖。为了从一定程度上解决此问题，对于 SSSQ，TUXEDO 采取均衡负载的方式，即对同一个 service 所在的多个 server，向其消息队列写如消息时采用一定的均衡负载算法，使消息放入忙碌 server 对应消息队列的可能性减小。

交易系统由于处理的交易量较大，通常采用 TUXEDO 负载均衡或者 MSSQ（Multi Server, Single Queue）的方式。MSSQ（Multi Server, Single Queue），就是有多 server 对应于一个消息队列。请求放入队列时不用做选择，这些 server 中的空闲 server 负责取出消息进行处理。

消息队列在操作系统中由 msgid 唯一标识[8]，TUXEDO server 由进程号 pid 唯一标示。消息队列与 TUXEDO server 的对应关系即 msgid 和 pid 的对应关系。此对应关系有两个方法可以找到：

(1) 通过 TUXEDO 的管理命令查找消息队列对应的 server 名字。在 tmadmin 中使用 PQ[10] 命令，Queue 为队列中阻塞的消息数，progname 为对应的 server 名。但此方法在消息队列发生阻塞的情况下，PQ 命令时常出现不能及时返回的情况，所以在消息队列出现阻塞时，建议使用第二种方法。

(2) 通过对消息队列最后读写进程的进程号来确定对应 TUXEDO server 的 pid。使用操作系统命令 ipcs -qa 可以看到阻塞队列的最后一个消息 send 和 receive 的进程号，根据进程号使用操作系统命令得到进程名即 TUXEDO server 的名字。在 ipcs －qa 的输出结果中，QNUM>1 时说明此消息队列出现了阻塞。LSPID 和 LRPID 为此消息队列最后一次写入和读出消息的进程。因应用 server 负责从消息队列读出消息进行处理，所以根据 LRPID 找到进程名即为消息队列对应的 server 名字。

3.2 无嵌套调用的情况

Service 在处理请求时，应用处理均在此 service 完成，不需要调用其它 service。这种情况下，只要根据 3.1 所述方法根据阻塞的消息队列找到对应 server 的名字即可。于简单的公式，可以直接以文本方式输入；对于复杂的公式，可以考虑使用公式编辑器，或者将公式制作成图片后插入文中。编辑公式的过程中要特别注意减号与连字符的区别，前者较长，后者较短。

3.3 有嵌套调用的情况

对 service 在处理请求的时候，应用处理过程中还需要调用其它的 service 来完成处理。这种情况下则不能单纯根据堵塞的消息队列定位到异常 server 进程。

对于嵌套调用的 service，根据同步调用和异步调用的差别，出现同样消息队列阻塞情况时，可能的异常进程也不相同。

3.3.1 同步调用

在 service A 调用 service B 为同步调用（tpcall）的情况下，如图 2。

service A 调用 service B 时必须等待 service B 处理完毕返回结果。在 servcie B 的处理过程中，service A 一直

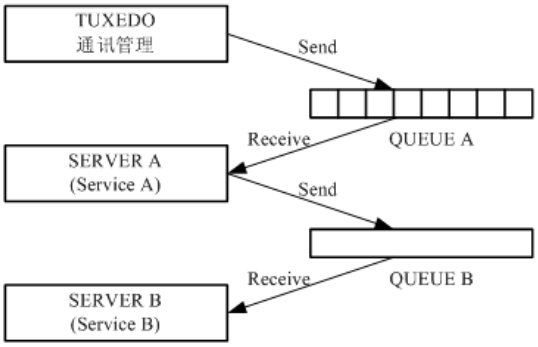

Figure 2. Queue A blocking in synchronization call
图 2. 同步调用 Queue A 阻塞

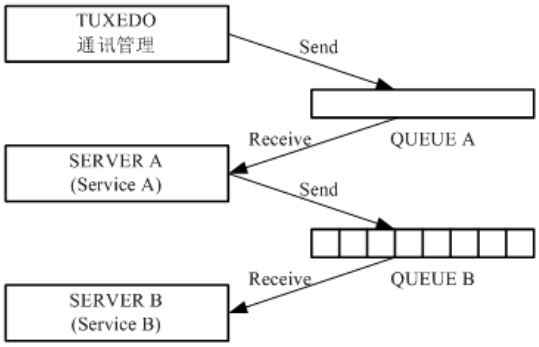

Figure 3. Queue B blocking in asynchronization call
图 3. 异步调用 Queue B 阻塞

处于等待状态，不能再接收新的交易请求，也不会再对 service B 产生新的调用。如果 service B 异常为繁忙状态，servcie A 因等待 service Bj 将也处于繁忙状态，此时，service A 不会再对 service B 有新的调用，Queue B 不会有消息阻塞，因 TUXEDO 通讯管理进程仍会不断向 Queue A 中写入消息，故 Queue A 产生消息队列阻塞。

那么在同步调用的情况下，是否 Queue A 阻塞就说明是 Service B 所在的 server B 异常呢，也未必尽然。Service A 除了对 service B 的调用外，本身也进行应用处理，如果是 service A 异常，Queue B 因得不到 service A 传来的消息，会保持空状态，Queue A 因 TUXEDO 调动进程的不断写入，会产生阻塞。为区分同步调用下 Queue A 阻塞时异常进程为 server A 还是 server B，只需检查当时 server B 的空闲状态即可，如 server B 空闲，则说明异常进程为 server A，反之为 server B。此检查可用 TUXEDO 管理工具 tmadmin 中的 psr[10]实现。

3.3.2 异步调用

在 service A 调用 service B 为异步调用（tpacall、tpforward）的情况下，如图 3。

service A 调用 service B 时不等 service B 处理完毕。在 servcie B 的处理过程中，service A 可以不断处理新的请求，并将对 service B 调用的消息写入 Queue B。如果 service B 异常为繁忙状态，Queue B 中因 servcie A 不断写入新的消息产生阻塞。因 service A 对新消息的处理没有停顿，TUXEDO 调动进程写入 Queue A 的消息被及时读取，故 Queue A 不产生阻塞。

与同步调用不同，在异步调用的情况下，如出现 Queue B 阻塞即说明是 service B 所在的 server B 异常。

因为如果 service A 出现异常，则消息不能源源不断的写入 Queue B，Queue B 就不会阻塞了。

4 结语

本文基于交易中间件的基本原理，探讨了交易中间件消息队列阻塞的成因，并结合 TUXEDO 给出了一种行之有效的异常 server 定位方法。在此基础上，可通过进程调试工具以及数据库 session、sql 的实时状态等进一步分析异常 server 的运行情况，对于进行系统故障的快速排除，恢复系统服务具有重要意义。

致谢

感谢中国传媒大学青年理科基金（YNG0709）的资助。

References (参考文献)

[1] The Open Group. Distributed Transaction Processing: The TX(Transaction Demarcation) Specification[S/OL]. :The Open Group Standards, 1995.
[2] CHEN Heping, YAN Yufeng， FANG Hongping. Design and Implementation of Message-Oriented Middleware Based on the DTP Model[J]. *Computer Systems & Applications*, 2003, 12(3)，P40-42.
[3] LI Zhitong, LV Guoying. Basic Realizing Principle of the Transaction Middleware[J]. *Computer Engineering*, 2003, 29(6): P32-33.
[4] Oracle Corporation. Using the Oracle TUXEDO Domains Component 10g Release 3 (10.3)[M/OL].: Oracle Press, 2009.
[5] YANG Min, DING Yuehua, WEN Guihua. Development and Application of Three-layer Model based on Middleware TUXEDO[J]. *Computer Development & Applications*, 2005, 18(2):P11-13.
[6] DING Xiaojin, XU Jiangyan. The application of middleware technology - BEA TUXEDO to inter-bank commercial real-time business[J]. *Joumal of Hubei Nomal University(Natural Science)*, 2008, 28(4):P54-57.
[7] GUO Shengxing, WANG Jing, LIAO Jianxin. Design of Persistence Message Queues Based on COPART-MACO[J]. *Joumal of Beijing Gechnology and Business University(Natural Science Edition)*, 2010, 28(1):P69-72.
[8] LI Gang, WU Chunqing. Digital UNIX Realtime M essage Queue Passing[J]. *Computer Engineering & Science*, 2000, 22(2):P87-90.
[9] Oracle Corporation. Setting Up an Oracle TUXEDO Application 10g Release 3 (10.3)[M/OL].: Oracle Press, 2009.
[10] Oracle Corporation. Administering an Oracle TUXEDO Application at Run Time 10g Release 3 (10.3)[M/OL].: Oracle Press, 2009.