联动北方
LANDING

企业级 IT 运维宝典系列

联动北方 出品

目　录

第 10 章 Tuxedo 性能调优

10.1 问题描述

Tuxedo 管理员最核心的能力是能够对 Tuxedo 应用系统进行性能调优，当然，一些性能的提高可以通过操作系统调优和硬件升级实现，另一方面也可以从调优 Tuxedo 应用环境以及调优 Tuxedo 使用的数据库环境入手。

这里只关注 Tuxedo 环境的调优，假设 DBA 可以调优数据库。

当对 Tuxedo 系统进行性能调优时，要记住少往往更好，意思是：较少的 Tuxedo SERVER 能够比有很多没有被正确的利用的 Tuxedo SERVER 使用少的系统开支，提供更好的响应时间。

10.2 故障排查

下面每个例子提供了一个手工的方法来执行调优，也提供了一个自动的方法通过使用 Isis 工具（该工具是 Integral Technology Solutions 提供的，具体介绍：www.integral-techsolutions.com）Isis 是一个商业工具，但免费试用版可以从 Integral 网站获得。

10.3 调优独立的 Tuxedo 服务

通常单单通过调优 Tuxedo SERVICE 们就可以获得最大的调优 Tuxedo 系统的效果，查看哪些 SERVICE 需要调优，可以通过分析 SERVICE 并且确定哪些 SERVICE 消耗了最多的处理时间。

第一步要确定哪些 SERVICE 们实际上运行时间超过其他的，以及哪些 SERVICE 被调用的比其他的更频繁。SERVICE 平均响应时间，乘以被调用次数就是 SERVICE 总共的处理时间，当调优 SERVICE 时，消耗最长的处理时间的 SERVICE 是性能调优的重点。

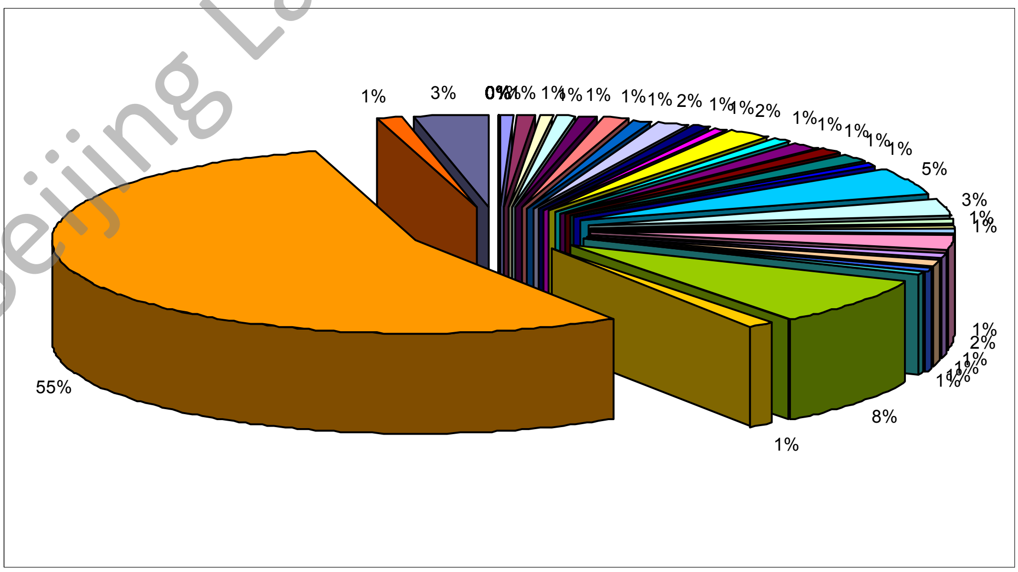

图 10-1 Isis 图显示服务响应时间

上面的图片显示了一个在 Tuxedo 上的计费系统的 SERVICE 的处理器负载情况。可以从图中清楚的看到，有个 Tuxedo SERVICE 占据了很大部分的处理器负载（图中橘黄色部分），这个 SERVICE 消耗了 55%的处理器时间，因此很显然这是一个需要着手性能调优的地方。第二的绿色显示部分的 SERVICE 也需要探查一下，因为它相对于系统中其他 SERVICE 消耗了大量的处理器时间。

这个图片是从真实的 Tuxedo 应用中获得的。

一旦 SERVICE 确定，接下来就是集中精力调优该 SERVICE。这里确定核心问题是 SERVICE 中使用了低效的 SQL 语句。该问题代码经过适当的的修改，很大程度的降低了分配到该 SERVICE 上的处理器负载，从而节省了为保障客户端性能而带来的昂贵的硬件升级。

这个图片实际上是分析原始 TxRPT 数据而获得的。

TxRPT 数据是 Tuxedo 输出的一种文件格式以提供分析服务响应时间。TxRPT 数据可以通过 Tuxedo SERVER 配置项 CLOPT 的-r 参数打开来获得。当 Tuxedo SERVER 启动并且 SERVER 的-r 打开，将输出 TxRPT 数据到 stderr 文件中，Tuxedo 允许通过 Tuxedo SERVER 配置项 CLOPT 的-e 参数来重定向 stderr 输出文件。

```
datasvr   SRVGRP=GROUP1 SRVID=10 MIN=2 MAX=2 CLOPT="-A -r -e stdout_simple"
```

示例 10-1

注意：-r 和-e 开关是在 CLOPT 的标志“--”之前

当-r 打开，Tuxedo SERVER 将打印每一次 SERVICE 被调用的情况，输出文件是-e 开关指定的，输出内容如下：

```
@CALLDB 3748        1116461950    1548917       1116461955    1553924
```

示例 10-2

其中各列分别为 SERVICE 调用的开始时间和日期以及结束的时间和日期，从这些值可以确定服务花费的时间。

Tuxedo 提供了基本的工具来分析 TxRPT 日志的结果，运行 txrpt 工具就是执行一个命令，输入数据文件以及要分析的日期，例如：

```
txrpt  -d 05/20 < stdout_simple
```

示例 10-3

将输出像下面文件中的样子：

```
SERVICE SUMMARY REPORT

SVCNAME          16p-17p          TOTALS
                 Num/Avg          Num/Avg
---------------  --------         -------
CALLDB           1081/2.12        1081/2.12
BUSINESSSVC      1082/2.53        1082/2.53
---------------  -------          -------
TOTALS           2163/2.32        2163/2.32
```

示例 10-4

在该文件中，可以看到在 Tuxedo 环境中运行着两个 SERVICE，第一个 SERVICE CALLDB 被调用了 1081 次，它的平均响应时间是 2.12 秒，第二个 SERVICE BUSUNESSSVC 被调用了 1082 次，平均响应时间是 2.53 秒。

因为 Tuxedo SERVICE 经常调用其他的 Tuxedo Service，这样就很难确定实际上到底哪个 SERVICE 是造成性能慢的根本原因。例如，像上面的 txrpt 报告中的 BUSINESSSVC 是运行最缓慢的，我们就集中精力在这个 SERVICE 上，然而，如果 BUSINESSSVC 要调用 CALLDB，这样根本原因就是 CALLDB 了，因为相差的 0.41 秒才是 BUSINESSSVC 消耗的，而剩下的时间都是 CALLDB 消耗的。

因此虽然集中注意力在运行时间长的 SERVICE 上很重要，但要注意 SERVICE 相互调用的依赖关系。

最后，当调优 Tuxedo Service 时，以下是一些容易引起性能问题的地方以及调优步骤的指标：

下表 10-1 详细的描述了这些常见问题：

常见问题	怎样解决这些问题
低效率的 SQL 访问数据库操作	同 DBA 一起测试调节 SQL 语句，或者调优数据库让数据库更有效的处理请求的 SQL，例如增加索引或者视图来优化相关的请求。
同步 Tuxedo 调用	记住，当使用 tpcall API 时，Tuxedo 将阻塞，一直等待调用的结果返回，如果所调用的是一个运行时间较长的 SERVICE，使用 tpacall 以及 tpgetrply 可能更加有效，这样程序就能够继续运行。
低效代码	有些时候，性能慢可能是由于低效的代码造成的，使用像代码审查这样基本的技术来找出问题代码，如果还不起效，那么就该应用一下高级的技术了像时间戳代码或者代码分析器。

表 10-1

为了简化确定性能瓶颈的分析过程，从(www.integral-techsolutions.com) 下载 Isis 工具。Isis 简化了导出和管理 TXRPT 文件，并且能够自动的产生帮助确定造成性能问题的根本原因。

10.4 将相似的 Tuxedo 服务分组到一个 SERVER

当开发一个 Tuxedo 系统，一个经常犯的错误就是将一些响应时间差别很大的 SERVICE 放到一个 SERVER 中

例如，假设一个 Tuxedo SERVER 有两个 SERVICE：

SERVICE A －是一个运行时间很短的 SERVICE（平均响应时间<0.1 秒），这个 SERVICE 被用来完成一个审计功能，因此它在运行期间将被调用很多次。

SERVICE B －是一个运行时间偏长的 SERVICE（平均响应时间在 15~20 秒），该 SERVICE 基于审计数据生成报告并且将报告输出到公司打印系统上。Service B 是一天被调用一次或者两次，这个依据安全部门检查频率而定。

逻辑上这两个 SERVICE 可以属于同一个 SERVER，但是放到一起会引发性能问题。由于 SERVICE A 是一个运行时间很短的 SERVICE，SERVICE B 是一个运行很少的 SERVICE，所以 Tuxedo 管理员决定只运行两个该 SERVER 实例来处理请求。

经过一天天的操作，管理员发现用户间歇性的出现超时和缓慢，这些无法解释。

进一步探查发现，在某一个性能负荷的情况下，大量的队列请求出现在等待审计 SERVICE，这个和系统性能放缓相符合。然而，管理员还是不能确定为什么会出现性能放缓。

分析 TxRPT 数据可以发现 SERVICE 的平均响应时间以及被调用的频率和时间。通过分析，管理员可以确定造成系统放缓的原因是在一分钟内出现了两个或多个报表服务。这个报表服务将两个 Tuxedo 实例都用来生成报表，不幸的是这意味着其他任何审计请求都必须加入请求队列一直等到报表完成。

审计的平均响应时间是少于 0.1 秒，然而当审计请求被放入请求队列等待报表请求完成时，这个响应时间将暴增到差不多是审计时间和报表时间的和：

总共响应时间=15~20 秒 + 0.1 秒。

另外，要考虑这点，很多审计将加入请求队列，并且要逐一运行，当审计每秒可以处理 10 个请求，报表完成要花费 20 秒并且同时来两个请求，将发生以下情况：

1. 每个审计 SERVER 实例都将执行报表服务；
2. 当报表请求出现，审计请求继续发来，在 20 秒内将收到 200 个审计请求（如果审计请求是每秒 10 个），这些请求将加入请求队列；
3. 当审计是同步运行的，那么将有 200 个客户端被阻塞，一直等待审计请求通过；
4. 最终报表服务完成了，进程开始处理审计请求，然而两个实例有 200 个请求要处理，将需要另外的 10 秒来处理完所有请求，才能使系统恢复到正常；
5. 这意味着平常 0.1 秒响应时间的审计将要花费 10-20 秒（取决于请求到来的时间），导致减少了大量的潜在的请求进入系统。

上面描述的情形能够通过更好的组织 SERVICE 来解决。很明显上面的例子要考虑将审计和报表服务分开，然而实际应用中怎样确定正确的 SERVICE 组合将会很复杂。

答案很明显，并且能够通过人工的技术或使用 Isis 自动的技术来获得。需要画一个简单的图表，该图包括四个象限，两个坐标轴，横轴是 SERVICE 调用的次数，纵轴是 SERVICE 的平均响应时间，接下来是将所有的 SERVICE 标示在图表中。

最终你将得到如以下所示的图：

图 10-2

当图表绘制完，接下来就是检查图，并确定有相似的特征的业务功能。

例如，在下面的图中标注的部分是有和用户管理相关的 SERVICE 的业务功能，这些 SERVICE 有相似的负载和响应时间并且是服务于一个相似的业务，可以将这些 SERVICE 组合成在一个 SERVER 里。

图 10-3

下面的图表显示了审计和报表服务的点，注意到他们在图表中的逻辑关系是非常的不适合的，因此如果放到一个 SERVER 中就不会很好的匹配：

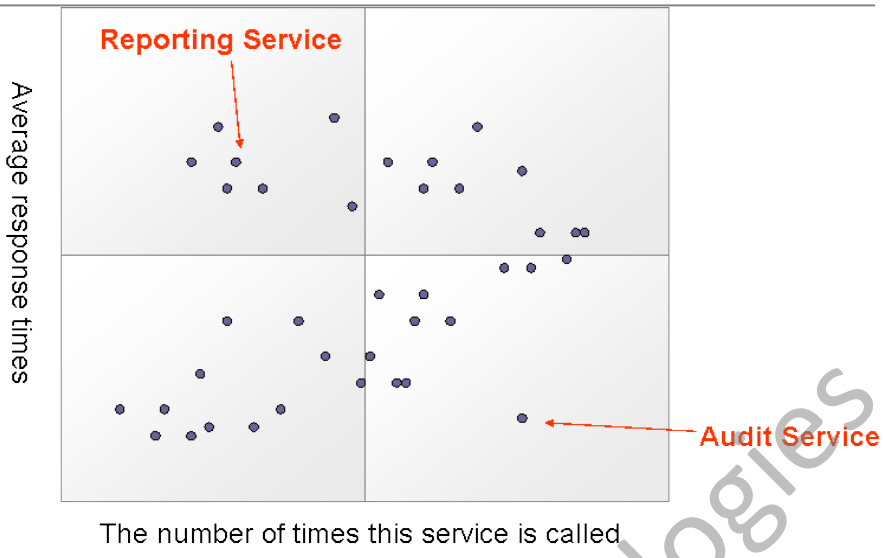

图 10-4

Isis 能够通过获得真实生产数据自动生成上面的图表来帮助 Tuxedo 性能调优。

10.5 调整 SERVER 数量

当系统中的 SERVICE 调试合适并依照负载情况和业务逻辑组合到 SERVER 中，接下来就是确定 Tuxedo SERVER 启动的合适个数和 SERVER 的合适的队列模型

首先考虑队列模型，因为是在这两个问题中比较简单的一个

Tuxedo 提供了两种可行的的队列模型，第一种是一个 SERVER 一个队列，叫做 SSSQ（Single Server / Single Queue）模型，另外一种是多 SERVER 共同监听一个队列，叫做 MSSQ（Multi Server / Single Queue）模型。

当选择是使用 SSSQ 还是 MSSQ 模型时，需要考虑：

- 在 SSSQ 模型中，Tuxedo 将会把请求放到负载最低的 SERVER 请求队列中，当请求信息在队列中时，不能够被其它 SERVER 拿出来，而要一直等待这个 SERVER 处理它。在 MSSQ 模型中，多个 SERVER 使用一个队列，其中任何一个 SERVER 空闲时，请求信息将出队列被处理。
- SSSQ 模型在 SERVER 中所有的 SERVICE 有相近的响应时间时将工作顺利。如果一个 SERVER 中 SERVICE 的的响应时间不同，使用 MSSQ 模型将更好，因为这样能够避免上面描述的响应快的 SERVICE 被响应慢的 SERVICE 阻塞的情况。
- 如果一个 MSSQ 中有大量的 SERVER（建议在一个 MSSQ 中不要超过 8 个 SERVER），而服务请求数又很少，将会造成性能放缓，这是因为当一条请求信息到达队列时，所有的 SERVER 都被告知，所有的 SERVER 竞争来获得该信息，当然只有第一个能够获得，所以当队列中有很多 SERVER 时，系统将花费大量时间来唤醒 SERVER 处理请求，而得到的结果却是发现信息已经处理完了。
- 顾名思义，在一个 MSSQ 中只有一个队列，因为分配给一个队列的内存空间由操作系统固定了，所以可用内存比 SSSQ 少很多。考虑以下情况，如果在一个 SSSQ 模型中有 10 个 SERVER，队列长度为 64K（一般队列长度），这样总共有 640K 可用来存放 SERVER 的请求信息，同样的情况，MSSQ 只有 64K，当 Tuxedo 的队列使用空间超

过 80%时，为保证可靠性，将会把请求信息存放到硬盘上，这样将造成很大的性能影响，因此如果应用的请求信息比较大，应该更多的考虑应用 SSSQ 而不是 MSSQ，因为 SSSQ 能提供更多的可用内存来存放更多的信息。

下表 10-2 总结了选择 SSSQ 或 MSSQ 应考虑的因素：

应用背景	消息偏小	消息偏大
SERVER 中的 SERVICE 有相近的负载，并且都是响应时间短以及频繁调用	MSSQ 或者 SSSQ 都能够工作的正常	考虑 SSSQ，因为大量的调用以及偏大消息将可能导致 MSSQ 队列溢出
SERVER 中的 SERVICE 有不同的负载情况或者某些 SERVICE 响应时间很长	当 SERVICE 有不同的负载情况，MSSQ 工作更好，因为如果一个 SERVER 阻塞在做某些工作，其他的能够处理	这是个棘手的情形，因为不同的负载使用 MSSQ 更好，而请求消息比较大又适合使用 SSSQ，综合两者处理可能提供最好的结果 配置多重 MSSQ 队列通过将 SERVER 分配到不同的组，例如使用三组 MSSQ，每组中 5 个 SERVER 而不是 15 个 SERVER 使用一个 MSSQ。

表 10-2

这样队列模型确定了，接下来是确定应该运行多少个 Tuxedo SERVER 实例。可以通过三个基本的步骤来达到这个目的，结合这三个步骤可以得到最佳值：

第 1 步. 使用 “-p” 开关:

Tuxedo 可以通过“-p”开关自动决定运行 SERVER 实例个数，“-p”是 CLOPT 的一个参数。

“-p”开关非常有用，它允许基于实际的负载调整 SERVER 实例个数。因为“-p”开关是被动的，经常稍微落后于真实的负载，因此有时不能及时反应当时需要的配置。

第 2 步. 使用 tmadmin 监控 SERVER 使用情况：

当运行在一个 SSSQ 配置上，即使配置了负载均衡，负载在 SERVER 实例中也并不是完全均匀地分布。当 Tuxedo 收到一个请求时，它将发送这个请求到第一个可用的 SERVER 中，如果是都不空闲，才发送给负载最少的 SERVER。

如果许多的 SERVER 可以接收同一个请求（例如在一个 SSSQ 有一个 SERVER 的 10 个实例当前都空闲），Tuxedo 将发送请求到同一个 SERVER，如果第一个 SERVER 繁忙，Tuxedo 将发送请求到第二个 SERVER，以此类推。

久而久之 Tuxedo 的这个负载均衡算法特征意味着，例如 50%负载将被第一个处理，30%将被第二个 SERVER 处理（由于只有在第一个繁忙的情况下第二个才被调用），15%将被第三个 SERVER 处理，5%被第四个处理，等等。

执行 psr（打印 SERVER）命令将显示系统中 SERVER 实例被调用的次数（当运行在一个 MP 环境中记住执行 d -m all 命令），可以给出类似以下的输出。

```
> d -m all
```

```
all> psr  -g GROUP2
Totals for all machines:
Prog Name      Queue Name  Grp Name       ID RqDone Load Done Machine
---------      ----------  --------       -- ------ --------- -------
calldb.exe     calldb      GROUP2         20     10       500 simon
calldb.exe     calldb      GROUP2         21      5       250 simon
calldb.exe     calldb      GROUP2         22      3       150 simon
calldb.exe     calldb      GROUP2         23      0         0 simon
calldb.exe     calldb      GROUP2         24      0         0 simon
```

示例 10-5

在这个例子中可以发现第一个 SERVER（实例 ID20）被调用的次数是最多的，实例 21 和 22 次之，从负载发现 SERVER 实例 23 和 24 没有被调用，这是一个很好的迹象表明这些 SERVER 实例没有必要，可以手动关闭或者配置-p 开关自动关闭。

第 3 步. 分析请求数来确定理想的 Tuxedo 配置

检查平均响应时间和调用频率，考虑 SERVER 中包含了多少 SERVICE，然后结合这些因素确定 Tuxedo 的使用率，通过这些步骤确定理想的 Tuxedo 配置。基于真实的负载来调试 Tuxedo 系统以达到最大产出。

当使用率确定，接下来给出一个公式用来确定需要多少 SERVER 实例来避免阻塞的关键问题。如果能达到最少的 SERVER 实例而保证无阻塞出现，这就是 Tuxedo 配置的理想情况。

给出一个非常简单的例子：

如果知道有一个 SERVER 中包含一个 SERVICE 将花费 1 秒响应时间，这样知道系统能够每秒钟处理一个事务。

如果知道每个小时有 8000 个服务请求（可以通过 TxRPT 文件确定），就能够计算出每秒需要处理的服务请求个数：

每秒的服务请求数= 8000 / 3600

这样计算出每秒有 2.22 个事务，所以我们可以通过 3 个 SERVER 处理这些负载。

很显然这是一个简单的例子，而真实的世界中 SERVER 包含多个 SERVICE，SERVICE 被调用的次数，以及响应时间都更加复杂，然而这个相同的准则可以用来获得理想的 Tuxedo 配置。

Isis 提供了内置的报表，该报表能够分析环境中的负载来帮助确定理想的配置，如果使用这个方法来调优 Tuxedo 环境，建议可以考虑 Isis 工具来获得帮助，因为这个工具非常有效的简化了分析。

10.6 FML 性能

如果 Tuxedo 应用使用大 FML 缓存可能会发现性能有所下降。下面的建议可以帮助你解决使用 FML 导致的性能问题。

当 FML 缓存块变得越来越大，那么将会遇到性能问题，这是由于 FML 在 Tuxedo 缓冲中包装的方式，对 FML 缓存块的包装方式的理解有利于懂得如何有效地处理 FML 结构。

每一个 FML 字段都有一个叫 FIELDID 的数字，如果检查 mkfldhdr32 创建的头文件，你会发现定义的每一个字段都会伴随着一个 FIELDID，这个值对考虑怎样包装 Tuxedo FML 缓存块非常重要。

一个 FML 中的字段是按 FIELDID 的顺序储存的，例如如果你在一个 FML 中有 3 个字段（A,B,C）并且 FIELDID 分别是 1，2 和 3，每个字段出现三次，那么实际的 FML 是：

```
[INDEX]:A1,A2,A3,B1,B2,B3,C1,C2,C3
```

示例 10-6

以这种结构形式的问题是当插入字段 A 的第四个实例时，实际上需要移动缓存块来为这个字段创建空间，例如插入 A4 的步骤是：

Step 1:为 A4 腾出空间

```
[INDEX]:A1,A2,A3,→ Clear Space for A4 → B1,B2,B3,C1,C2,C3
```

示例 10-7

Step 2: 插入 A4

```
[INDEX]:A1,A2,A3,A4,B1,B2,B3,C1,C2,C3
```

示例 10-8

当处理非常大的缓存块时，以这种方式增加字段将造成系统慢下来的性能问题，注意

这个问题有两种解决方法。第一个（也是最简单的）是使用一个字段来储存多个值，例如如果字段 A, B, C 代表了从数据库取出的列，然后可以用逗号分开 A, B, C 值，最后以一个字段返回（叫做 D），这个方法将把 FML 改成：

Step 1: 逗号隔开 A, B, C 然后创建新字段 D

Step 2: 为数据库中取到的每一条数据插入一个 D 字段，FML 如下：

```
[INDEX]:D1,D2,D3
```

示例 10-9

当取得 D 字段中的数据后，再使用分离算法返回一个字段中的多个值。

虽然这个方法简单，但是这个增加了缓存结构体的复杂性，并且需要每个收到该缓冲块的客户端懂得怎样提取这个字段中的内容。

第二个方法比第一个方法复杂，但提供了一个不需要客户端额外去解包的方法。

当插入字段到 FML 缓存块时，如果 FML 储存都按顺序插入，那么插入的性能将大幅改善，原因很简单，因为 Tuxedo 系统不需要为了插入移动字段，例如：如果存在的 FML 如下：

```
[INDEX]:A1,A2,A3
```

示例 10-10

然后增加字段 B1 将不需要移动任何字段，然而当 B1 增加了

```
[INDEX]:A1,A2,A3,B1
```

示例 10-11

开发人员决定插入 A4，这样 B1 将需要移动，也会造成性能问题。

因此，使用这种方法避免性能问题的关键是在插入字段 B 和 C 之前插入所有的 A，记住正确的字段顺序能够通过查看 FIELDID 得到，这样先插入最小 FIELDID，以升序插入剩下的字段。

不幸的是，数据通常不是以一个适当的格式返回给开发者来以此种方式插入，例如每次一个 SQL 查询返回一行数据，要实现以上方式插入 FML，经常需要缓存结果数据并要分开操作来插入 FML 缓存块。更高级的客户端有时封装这些功能到一个分开的 FML 库集合中，以此来开发更高性能的 FML。

10.7 额外的性能参数

以下一些附加的步骤，可能改善 Tuxedo 应用性能。

10.7.1 多个 WSH 连接

当一个工作站客户端连接到一个 Tuxedo 应用，工作站处理程序（WSH）作为一个本地客户端代表工作站客户端来运行。每个 WSH 能够处理很多并行的客户端连接。

WSL 配置项“-x”参数表示每个客户端处理进程能同步接入客户端连接的个数，默认的是 10，这个值必须大于 0.

依据通过 WSH 发送的负载情况，以及连接到应用的客户端个数，能够通过多路复用来调优达到应用的需求。算出复用因子是一个反复的过程，需要使用系统工具分析 WSH 进程的负载情况来确定。

10.7.2 关闭 WSL / WSH 加密

WSL 的“-z”和“-Z”参数定义了将要通过网络的从工作站客户端到工作站处理器的加密级别。

“-z”代表最小的级别，“-Z”代表了最高的加密级别，默认为 0。

如果从客户端到工作站进程不需要加密的通信链路，那么推荐检查值是否为 0 或者没有设置。

10.7.3 打开 WSL / WSH 压缩

在 WSL/WSH 上打开压缩能很大程度上改善 Tuxedo 网络传输的性能，尤其在一个广域网或者 ADSL/ISDN 方式的连接。

压缩功能的开关是通过 WSL 的“-c”参数来确定的。

“-c”参数决定了工作站客户端和处理进程之间压缩属性，任何在工作站客户端和处理进程之间的缓存块大于给定值那么就会压缩，默认的值是 2147483647，意味着当 buffer 大小在 0 和 2147483647 之间，就不会压缩。如果该值设为 0，意味着不论 buffer 大小都压缩，注意避免这种情况。

10.7.4 机器类型

当 Tuxedo 数据通过网络链路传递时，需要转换 buffer 为机器无关的格式，从而确保接收方的值和发送方的值保持一致。

例如，考虑这样的情况，当一个 Windows 机器以及一个 Solaris 机器被设置在一个网络中，由于两个环境中的数据格式不同，为了让 Tuxedo 系统能够从一个机器到另外一个机器传输一个 buffer，就必须先将数据转换为机器无关的格式，然后再通过网络传输。

这个转换花费相当大的处理时间，如果环境中的所有机器类型都一样，那么就不需要转换。决定是否要转换是通过比较远端的机器 MACHINE TYPE 以及当前机器的 MACHINE TYPE，如果两个值一样，那么 Tuxedo 就不做转换，而是简单的将该信息通过网络传输，这样节省了处理开销。

如果环境中所有的机器类型都一致，那么将*MACHINES 部分的 TYPE 设置为一致，将获得更好的性能。

10.7.5 SPINCOUNT

Tuxedo 无论客户端或服务端何时调用一个服务都要使用公告牌（BB），对于公告牌的排他性访问，Tuxedo 要通过锁机制进行保护。

SPINCOUNT 这个参数表示当锁被占用，而又有新的进程或线程要访问时，它进行多少次拿锁的重试，之后再进入睡眠状态。

在单处理器机器中重试是一个不好的选择，因为这时应该让出 CPU 给占有锁的进程，使它能尽快完成工作。在这种情况下，SPINCOUNT 应该设置为 1。

对于多处理器机器，推荐设置 SPINCOUNT 值在 5,000 到 50,000 之间。它的设置取决于处理器的类型、个数等多种因素，通常需要通过观察系统吞吐量，加以调整。

SPINCOUNT 可以用 TMIB 动态做修改。

10.7.6 去掉授权和审计安全

Tuxedo 版本 7.1 增加了 AAA（authentication, authorization, 和 auditing）安全插件功能，这样实现 AAA 插件函数将不需要基于 Tuxedo 管理的安全机制。由于 AAA 接口经常在 Tuxedo 代码中被调用，对于没有使用 AAA 安全插件的应用，就不需要付全额的开销支持 AAA 安全调用。

因此，Tuxedo 版本 8 或以上，RESOURCES 部分的 OPTIONS 参数增加了 NO_AA 选项，NO_AA 选项将避免调用审计和授权的安全函数，而对于需要认证的应用，这个特点不能关闭。

如 NO_AA 启用，以下 SECURITY 参数将受到影响：

NONE, APP_PW, 和 USER_AUTH 将继续正常工作—除非没有授权和审计要做

ACL 和 MANDATORY_ACL 参数将继续正常工作，但只使用默认的 Tuxedo 安全机制。

10.7.7 关闭多线程处理

Tuxedo 版本 7.1 中增加了线程机制，由于这种架构，所有 ATMI 调用都必须调用互斥函数以保护敏感状态的信息，此外，库中运用了分层和缓存方案导致了更多的互斥处理。如果应用中没有用到线程，关闭这些将大幅度提高性能。

为了关闭多线程处理，Tuxedo8.0 和后来的版本中实现了 TMNOTHREADS 参数，通过配置可以在不引入新 API 或标志的情况下单个进程可以打开和关闭线程。

如果 TMNOTHREADS 设置为“yes”将避免调用互斥函数。

10.7.8 关闭 XA 事务

对于不使用 XA 事务的应用，为了提高性能， Tuxedo8.0 以及更新的版本 RESOURCES 部分增加了 NO_XA 标志。

如果设置了 NO_XA，那么将不使用 XA 事务，注意如果 NO_XA 设置了，GROUPS 中设置的 TMS 将失败。