ORACLE
FUSION MIDDLEWARE

An Oracle White Paper
July 2012

Load Balancing in Oracle Tuxedo ATMI Applications

ORACLE

Introduction

This paper is intended for the following audiences:

- Designers and programmers knowledgeable about Oracle Tuxedo Application-to-Transaction Monitor Interface (ATMI) and who want to write Tuxedo ATMI applications.
- Designers and programmers knowledgeable about Oracle Tuxedo CORBA and who want to write Tuxedo CORBA applications.
- Tuxedo system administrators who deploy, troubleshoot and diagnose Tuxedo Applications.

Both Oracle Tuxedo CORBA and Oracle Tuxedo ATMI inherently provide transparent load balancing, although Oracle Tuxedo CORBA and Oracle Tuxedo ATMI achieve routing and load balancing differently due to the fundamental differences between objects and procedures. Meanwhile, since Oracle Tuxedo CORBA is built on top of Oracle Tuxedo ATMI, they share the same concept and implementation at ATMI service level load balancing. So, although this paper is focusing on the load balancing in Oracle Tuxedo ATMI application, it also helps for Oracle Tuxedo CORBA programmers to understand the load balancing in Oracle Tuxedo CORBA applications. There is a dedicated white paper “Load Balancing in Oracle Tuxedo CORBA Applications” describing the load balancing in Oracle Tuxedo CORBA applications.

The first part of this paper describes the fundamentals needed to understand load balancing in Oracle Tuxedo ATMI. It describes the load balancing algorithm and factors affecting load balancing. The second part of this paper provides several server configuration patterns which play roles in load balancing in the application system.

Tuxedo Routing

How Requests Are Routed

Oracle Tuxedo ATMI services are stateless, so requests may be routed to any available server.

- Workstation client

Workstation clients join Tuxedo application through workstation Listener/Handler (WSL/WSH). Environment variable WSNADDR=”//host:port” is used to locate the WSL/WSH. Accordingly, workstation client’s service request routing consists of two stages:

- Service request is sent from workstation client to WSH.
- WSH routes the service request to appropriate server on behalf of the workstation client. WSH acts as a native client at this stage.

- Native client

If Data Dependent Routing (DDR) is specified, candidate server groups are screened out first; then the best server is chosen within the candidate server groups. Otherwise, the service request will be routed to the best server selected from all the server groups where the qualified servers are configured.

Goal of Load Balancing

The goal of Tuxedo Load Balancing is to perform a quick set of calculations that will give a good distribution of workload between servers in OLTP applications, which typically require short response time and high throughput. In the situation that multiple servers on a machine are idle, any of these servers can be chosen without causing a request to wait. In fact, it is better to choose the same servers over and over in such a situation rather than cycling between all such servers, since this will minimize the amount of OS process switching that must occur. These goal and principle will be reflected in the load balancing algorithm design as described in following paragraphs.

Where Load Balancing Takes Place

The objective of load balancing is to choose the least busy server from all available servers so as to distribute the service requests evenly in the whole system to gain best system response time and throughput. In Oracle Tuxedo, load balancing takes place on the client side. For request/response service, tpcall/tpacall invokes load balancing algorithm to choose the least busy server; for conversational service, tpconnect invokes load balancing algorithm to choose the least busy server. Whereas for /WS clients, the tpcall/tpacall/tpconnect just send the service request to WSH and do not perform load balancing in /WS clients. WSH calls native client routine to achieve the load balancing task on behalf of /WS clients. To achieve the load balancing between /WS clients and WSL servers, multiple WSL access points can be configured by WSNADDR. This feature can assign the /WS clients evenly to different WSL servers to balance the work load between WSL/WSH in the system.

Native clients and WSH are able to get the full knowledge to route the service request by retrieving the local Bulletin Board (BB). From the request service name, client process can retrieve the service entry from the BB, and then the servers and their request queues. Each request queue is associated with a parameter *wkqueued*, which indicates the current work load in the queue. For Single Server Single Queue (SSSQ) configuration, *wkqueued* is used; for Multiple Server Single Queue (MSSQ) configuration, *wkqueued*/n (n is the number of servers on the queue) is used by client process to make choice between candidate servers. The higher the value, the heavier the server's work load. Tuxedo load balancing choose the server with lowest *wkqueued* value as the best server if they are both idle or both busy.

By default, load balancing takes place locally. It means in MP configuration, the load balancing criteria (*wkqueued* and server state) are updated locally on each node, and the nodes do not exchange the

wkqueued and the server state information. The client processes make choice of best server only based on the local BB information. For instance, figure-1 describes the following configuration:

- machines MACH_1 and MACH_2 are configured in the MP mode application
- simpserv_1 runs on MACH_1 and simpserv_2 runs on MACH_2, they both provide the service "TOUPPER"

Consider the following executing scenario (To simplify the case, we suppose the net load is small and can be omitted):

- simpcl_1 call TOUPPER on MACH_1 and choose simpserv1 as the best server and update the *wkqueued* to 50 in local BB.
- At the same time, simpcl_2 call TOUPPER on MACH_2 and also choose simpserv_1 because current *wkqueued* of simpserv_2 is 50 and it is higher than that of simpserv_1 with value 0. simpcl_2 is not aware of the fact simpcl_1 on MACH_1 is also invoking simpserv_1. Hence it will update the *wkqueued* of simpserv_1 to 50 in local BB, although the actual work load of simpserv_1 is actually growing up to 100.

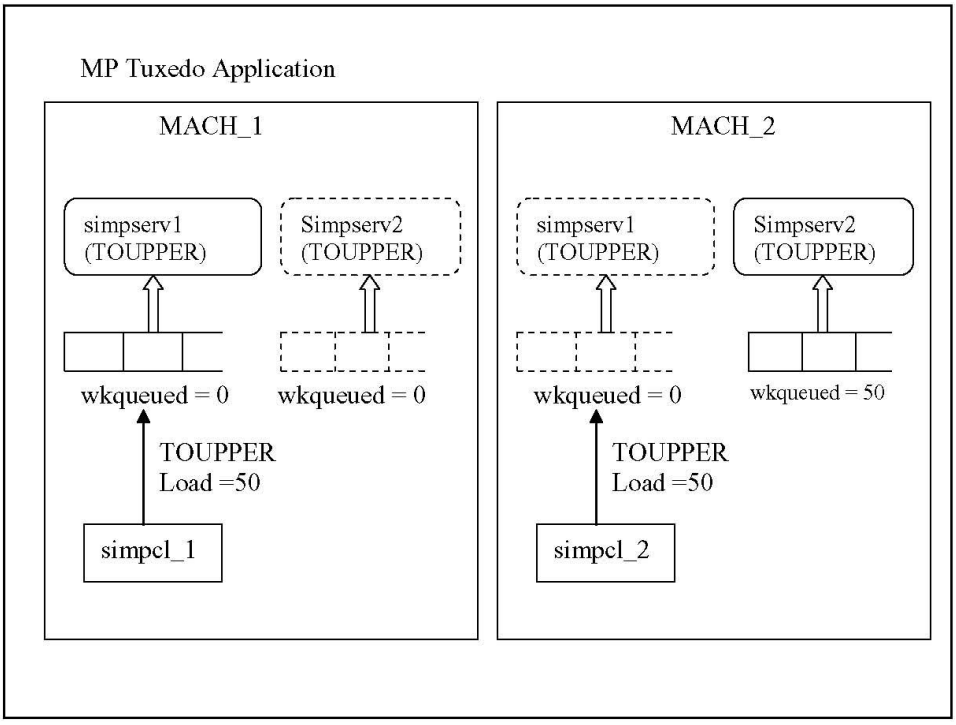

Figure-1 Load Balancing Take Place on Client Side Locally

In order to solve this issue and achieve more accurate load balancing in MP mode, TSAM 12*c* enhances the load balancing by broadcasting the *wkqueued* information within the whole domain. Refer to *Enhanced Tuxedo Load Balancing* for details.

Load Balancing Algorithm

The load balancing algorithm implemented in Tuxedo use the bubble sort approach to choose the best server among the set of candidate servers. The winner of each round of comparison survives and continues to next round of comparison. The rule of winner choosing between a pair of candidates can be summarized as following points:

- Local idle server first
- Remote servers are always treated as busy server
- If both servers are busy or idle, compare their *wkqueued* value, choose the server with lower value.
- If both servers are the same in term of above criteria, choose the first one as the winner of this round of comparison.

To understand above algorithm, the involved concepts should be understood firstly:

Server Location

LOCAL – the server is on the same machine as the native client process.

REMOTE – the server is on the different machine from the native client process.

```
MACH_2
MACH_1
wkqueued = 50
Simpserv2
(TOUPPER)
TOUPPER
Load =50
wkqueued = 0
simpcl_2
simpserv1
(TOUPPER)
TOUPPER
Load =50
wkqueued = 0
simpcl_1
simpserv1
(TOUPPER)
MP Tuxedo Application
wkqueued = 0
```

```
Simpserv2
(TOUPPER)
```

Server State

IDLE – the state that a server stays between two successive executions of service routine. Please note that a server stays in IDLE state does not imply the request queue of this server is empty. There is a tiny time window between when server finishes current service request and when it fetches the next request from the queue. The server stays in IDLE state within the tiny time window although there is still backlog in its request queue.

BUSY – the state that a server stays when it is executing the service routine, i.e. serving the client.

Work Queued Updating

In Oracle Tuxedo, there are two different strategies of calculating work queued of server according to different models of application and the load balancing parameters setting:

- Periodic Accumulative Updating

wkqueued is increased by LOAD value of the service when the service request is added to the server's request queue; it's not decreased when the request is completed by the server. The *wkqueued* value grows up linearly during the sanity scan period until BBL reset it to 0 at sanity scan. The resetting is necessary because it prevents the load balancing heuristic from being affected by the stale historical work queued information.

This updating method will see the service requests to be spread to the servers evenly in round-robin style. So we also refer to load balancing algorithm which applies this approach of updating *wkqueued* as Round-Robin (RR) algorithm.

- Real-Time Updating

wkqueued is increased by LOAD value of the service when service request is added to the request queue; it's decreased by LOAD value of the service upon the request is completed by the server.

This updating method maintains *wkqueued* in a real-time mode and reflects the current work queued more accurate than prior approach.

Specially, in an unloaded system, the values of *wkqueued* of queues keep 0 during most of the running time. In this situation, you may observe one or two servers do most of work in an unloaded system, rather than an even distribution of work among all servers. This is because load balancing algorithm always chooses the first server when two servers are equal in work load. Therefore, the second server will not be chosen until a new request arrives while the first server is busy. The third server will not be chosen until a new request arrives while the first and second servers are both busy, and so on. This kind of behavior is reasonable because it optimize the chance that the server's working set is already in the cache.

We also refer to the load balancing algorithm which applies this approach of updating *wkqueued* as Real-Time (RT) algorithm.

Figure-2 illustrates the difference between these two updating approaches.

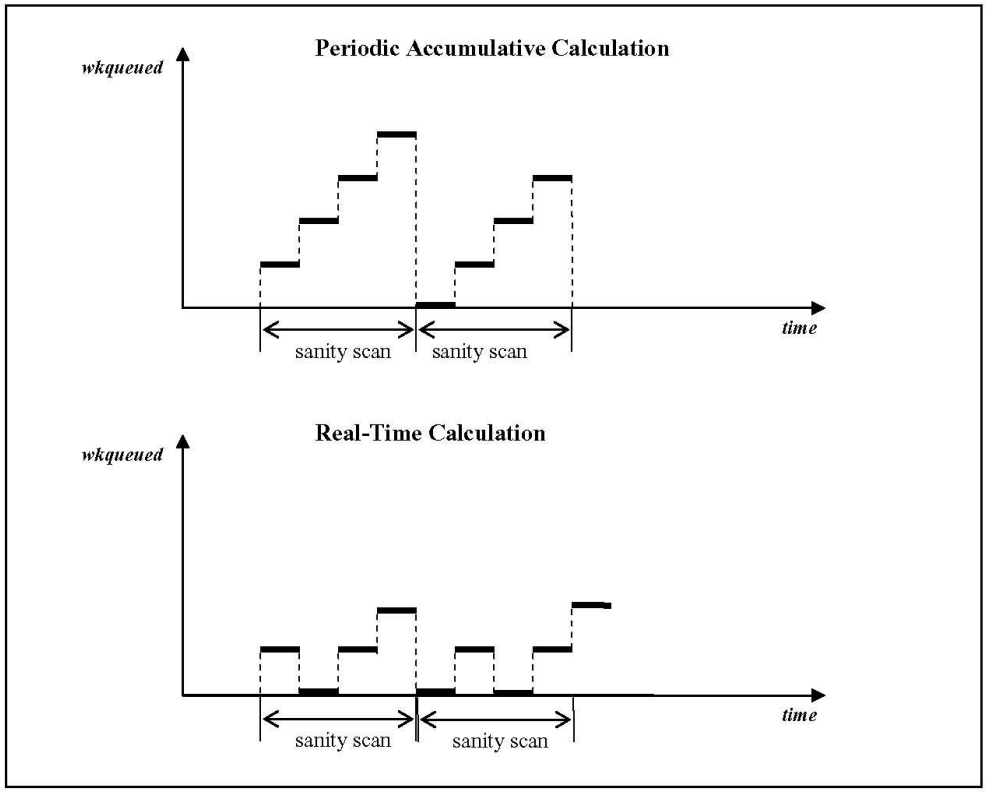

Figure-2 Two Approaches of Updating Work Queued

Role of BBL in Load Balancing

As mentioned in previous chapter, BBL is responsible for resetting the *wkqueued* parameter of all the server request queues in the system at interval of sanity scan. This operation is necessary because of the potential inaccuracy of *wkqueued* due to the race condition in updating *wkqueued* in BB by multiple processes, and the sync operation is able to eliminate the negative influence of the stale work queued information on the load balancing.

The operation is performed in a little different way for Round-Robin and Real-Time load balancing algorithm:

- Round-Robin

```
If CORBA stateless service
If servers on the request queue are all idle and the queue is empty
Then
    Reset wkqueued to 0
Else
    Do nothing;
```

```
Else
    Reset wkqueued to 0.
```

- Real-Time

```
If servers on the request queue are all idle and the queue is empty
Then
    Reset the wkqueued to 0
Else
    Do nothing.
```

Known Issues in Tuxedo Load Balancing Algorithm

Load balancing algorithm does its best efforts to distribute the service request evenly among the available servers, but due to following potential problems it may not work as perfectly as you expected:

- Doubtable IDLE server state

As explained in previous "Server State" section, server in idle state does not necessarily mean its request queue is empty. Idle state is just a transient state server stays between it finish current service request and pick up next request. Since Tuxedo applies local-idle-first strategy in choosing the best server, it may inappropriately choose an idle server with high backlog in its request queue, rather than a currently busy server but without backlog in its request queue.

- BBL sync *wkqueued*

BBL help to eliminate stale statistic information by sync *wkqueued* parameter of each request queue periodically. On the other side, the sync operation may do harm to load balancing. Suppose Real-Time load balancing algorithm is used, there are two servers with *wkqueued* 100 and 50 separately and they are both in idle state when BBL sync *wkqueued* for them. If a client issues the service request at the very moment, the higher work load server may be mistakenly chosen to serve the new request because the sync by BBL removes the load difference of these servers.

- Inaccuracy of *wkqueued*

If service cache is enabled, update of work load counters in BB will not be protected by lock. Even if service cache is disabled, decreasing of *wkqueued* (by servers upon service is done) is still not protected by BB lock unless ACCSTATS option is specified in SHM model. Slight inaccuracies in load balancing can occur in high load application systems when load balancing statistics are updated without a lock. For most applications, the additional overhead incurred in waiting for a lock to update statistics would slow the system more than any load balancing inaccuracies that are likely to result from not waiting for the lock.

Parameters Affecting Load Balancing

Several UBBCONFIG parameters affect load balancing within an Oracle Tuxedo ATMI application. They are:

- LDBAL

- MODEL
- LOAD
- NETLOAD
- SCANUNIT
- SANITYSCAN
- SICACHEENTRIESMAX
- ACCSTATS

LDBAL is specified in the RESOURCE section of Tuxedo configuration file. Its value can be Y or N. From literal understanding, it specifies whether or not load balancing is performed. However, it is not true in real application - Tuxedo always do load balancing for each service request whatever LDBAL is set, but its value determines the way load balancing works. Table-1 illustrates how LDBAL together with MODEL affects the load balancing. If LDBAL is not specified, the default is Y.

MODEL is specified in the RESOURCE section of Tuxedo configuration file. Its value can be SHM or MP, indicating single machine or multiple-machine application. Its setting together with LDBAL determines how load balancing works:

TABLE 1. LDBAL CONFIGURATION

	LDBAL = Y	LDBAL = N
SHM	Real-Time, BBL sync wkqueued periodically	Real-Time, BBL does not sync
MP	Round-Robin, BBL sync wkqueued periodically	Real-Time, BBL does not sync

As above table illustrates, BBL does not play its role of sync *wkqueued* if LDBAL is set to N. In a SHM model application, this should not be a problem. Because all the servers are on the same machine and there is only one BB in the system, real-time update of *wkqueued* can reflect servers' work load correctly. So in SHM model, setting LDBAL to Y or N does not affect the load balancing much.

However, in MP model application, lake of sync *wkqueued* operation implies load balancing between servers across multiple machines will not work. From this point of view, the statement "No load balance when LDBAL is set to N" makes sense to some extent. The reason is that Tuxedo takes the local-BB-only strategy to update work load counters. Using figure-1 as an example, consider TOUPPER service requests are initiated from MACH_1, if remote simpserv_2 is chosen as the best server by load balancing algorithm, its *wkqueued* is increased locally in MACH_1 BB , after the service is done by simpserv_2, the *wkqueued* is decreased by simpserv_2 locally on MACH_2 BB. Therefore,

wkqueued of simpserv_2 in MACH_1 BB will never get decreased and it will eventually grow up to a large enough value to make it lose the load competition against local server simpserv_1. Hence the load balancing between the local server and remote server will not work.

LOAD is specified in the SERVICES section of Tuxedo configuration file and it specifies a relative load factor associated with a service instance. If not specified, the default is 50. A greater number indicates higher load. Load balancing algorithm uses this value to update the *wkqueued* parameter of the request queues.

NETLOAD is specified in the MACHINES section of Tuxedo configuration file and it specifies the additional load to be added when computing the cost of sending a service request from this machine to another machine. TMNETLOAD is the environment variable counterpart of the parameter and its setting takes precedence over the UBBCONFIG setting. Load balancing take this value into account when determine if route a service request to a remote server. If you prefer local server to handle the request, you may set this parameter to a very high value.

SCANUNIT and **SANITYSCAN** are specified in the RESOURCES section of Tuxedo configuration file. They multiplied together determine the interval BBL sync the work load parameters for each queue in the system.

SICACHEENTRIESMAX is specified in the MACHINES section of Tuxedo configuration file and it specifies the maximum number of service cache entries any process is to hold on this machine. If it is not specified, the default value is 500. If it is set to 0, no service caching will be performed by any process on this machine. SICACHEENTRIESMAX can also be specified in SERVERS section for individual servers. It takes precedence over the setting in MACHINES section. TMSICACHEENTRIESMAX is the environment variable counterpart of the parameter and its setting takes precedence over the UBBCONFIG setting. From load balancing point of view, service cache on or off determine if the calculation of *wkqueued* and candidate server choosing procedure is protected by BB lock or not. If service cache is on, the procedure will not be protected by BB lock and the load balancing will be less accurate, but the system gain more performance; otherwise, it's protected by BB lock and the load balancing will be more accurate, but cost the system performance.

ACCSTATS means accurate statistics on work load counters. It only works in SHM model. This feature can be enabled by setting attribute TA_OPTIONS in class T_DOMAIN to ACCSTATS in TMIB, or via `tmadmin` subcommand `shmstats ex`. But ACCSTATS alone can't guarantee the accuracy, only when it's enabled together with service cache disabled, can the accuracy be assured. On the other side, since the accuracy is assured by exclusive access to BB, performance is lost at the same time, especially in a high load system.

Enhanced Tuxedo Load Balancing

TSAM 12*c* enhances the load balancing algorithm of Tuxedo by leveraging the real-time performance metrics collected by TSAM. It can solve the following limitations in transitional Tuxedo load balancing algorithm:

- The LOAD and NETLOAD values must be pre-configured in UBBCONFIG as a constant value.

- Client cannot get the accurate *wkqueued* value of remote servers

TSAM collects the following performance metrics which are used for enhanced load balancing:

- **Network time**: time takes to transmit the message from the requester machine to the service provider machine and the corresponding reply transmitted back to the originating machine, including the possible delay on the BRIDGE.
- **Queue wait time**: time waiting in the server queue before the request gets served.
- **Service execution time**: time the server takes to serve the request.

All these performance metrics are broadcasted within the whole domain.

In the enhanced load balancing algorithm:

- The service response time is the sum of the **Network time**, **Queue wait time**, and the **Service execution time**.
- The **Average service execution time** replaces the constant service LOAD.
- The **Average network time** replaces the constant NETLOAD.
- The **Queue wait time** replaces the *wkqueued*.

Server Configuration Patterns

As aforementioned, Tuxedo load balancing algorithm is performed at the client side to distribute the system load evenly among the service providers. Once the least busy server is chosen and the request is routed the request queue of the server, server side configuration can also play important role in balancing the system load.

Single thread or Multi-thread

- Single thread server

Single thread server cannot handle the next request until it finishes current one. In other words, single thread server can't handle two or more service request simultaneously. For typical OLTP applications, since the execution time of the service function is supposed to be very short, single threaded server can fulfill most OLTP application requirement with short response time.

A single threaded server is run if the `buildserver -t` option is not specified or MAXDISPATCHTHREADS is not specified or is 1 for a multi-thread server built with –t option.

- Multi-thread server

Tuxedo will create a pool of working threads for the multi-thread server when booted. The size of the thread pool is determined by server configuration parameter MINDISPATCHTHREADS in UBBCONFIG. There is a dedicated dispatcher thread within the server responsible for dispatching the service requests to the working threads. It's the load balancing within the multi-thread server. The dispatcher thread always picks up an idle thread to handle the incoming request. If all the working

threads are busy, new working threads are spawned to handle the request until reach the maximum thread pools size specified by MAXDISPATCHTHREADS. If all working threads are still busy under this condition, new service requests will be queued in the request queue.

Since multi-thread server can handle multiple service requests simultaneously, it helps improve the system response time if the service function take relatively long time to complete and multiple concurrent service requests are dispatched to the same server.

A multi-thread server is run if the server is built with "-t" option and MAXDISPATCHTHREADS is greater than 1.

SSSQ or MSSQ

- Single Server Single Queue (SSSQ)

This is the default setting of application servers. Each server has its own request queue. Or in other words, the request queue has only one server to serve it. This configuration totally depends on the Tuxedo load balancing to distribute the service requests evenly to servers in the system. The accuracy of load balancing affects the system response time and throughput significantly.

- Multiple Sever Single Queue (MSSQ)

This configuration enables two or more servers to service the same request queue. When multiple idle servers contending on the same request queue to get the next request, which server will get the next message depends on the operating system. In other words, there are two stages in load balancing in this configuration:

- Tuxedo load balancing algorithm chooses the set of MSSQ servers as the targets and request is queued into the shared request queue
- OS distributes the request messages to the servers.

MSSQ can help resolve the potential inaccuracy of Tuxedo load balancing and enhance the response time and throughput of the application system.

Conclusion

In a distributed computing environment, decision on the load balancing strategy is a tradeoff between performance and accuracy. Higher accuracy means more exchange of load information, more critical sections, hence lower performance and vice versa. There is no perfect solution from the view of both angles. Tuxedo load balancing do its best effort to distribute the work load evenly among the system, but it does not guarantee the 100% accuracy. Application designers should not make such assumption that Tuxedo can do load balancing perfectly and put the stake on the improper assumption.

TSAM 12*c* enhances the load balancing algorithm of Tuxedo by leveraging the real-time performance metrics collected by TSAM. Even though it may introduce slight performance impact (which is imperceptible in most situations) because of the monitoring overhead, it is a good choice for Tuxedo applications which appreciate the accuracy of load balancing.

Multi-thread server and MSSQ configuration on the server side can help resolve the side effect of the potential inaccuracy of load balancing in certain situations.

Load Balancing in Oracle Tuxedo ATMI Applications
July 2012

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200

oracle.com

Oracle is committed to developing practices and products that help protect the environment

Hardware and Software, Engineered to Work Together