Introduction

The purpose of this document is to describe the internal threading model utilized by Oracle Service Bus (OSB), with a focus on the HTTP Transport, and its implications with regards to performance and server stability. There are three key points to comprehend:

- Proxy service request and response pipelines always execute in separate threads.
- When invoking an external service, threads may be blocking or non-blocking depending on the pipeline action, the Quality of Service option and the transport used.
- When using blocking calls, a work manager having a Min Thread Constraint must be associated with the response to prevent server deadlock.

Threading

As mentioned, the request and response pipelines of a proxy service are executed by different threads. While the request thread originates from the transport of the Proxy service, the response thread originates with the business service transport. This is important to remember when assigning work managers to services, which is discussed in more detail below.

OSB optimizes invocations between proxy services. When one HTTP proxy service calls a second HTTP proxy service, the transport layer is bypassed. The request message is not sent via a network socket, so the transport overhead is eliminated. Instead, the thread processing the initial proxy service will continue processing the request pipeline of the called service. Similarly, when invoking a business service, the proxy service thread is also used to perform the send of the request. As the HTTP Transport is implemented utilizing asynchronous capabilites of WebLogic Server (WLS), the response of the business service is processed by a different thread.

Pipeline Actions

Route Action

By default, the HTTP transport utilizes asynchronous features of WebLogic Server to prevent thread blocking while waiting for a business service response. For the execution of a route action, once the thread finishes sending the request it returns to the pool where it is then utilized to process other work. When a response is returned from the external service, a second thread is scheduled to process it. This behavior can be modified by using a Route Options action and setting the Quality of Service to Exactly Once.

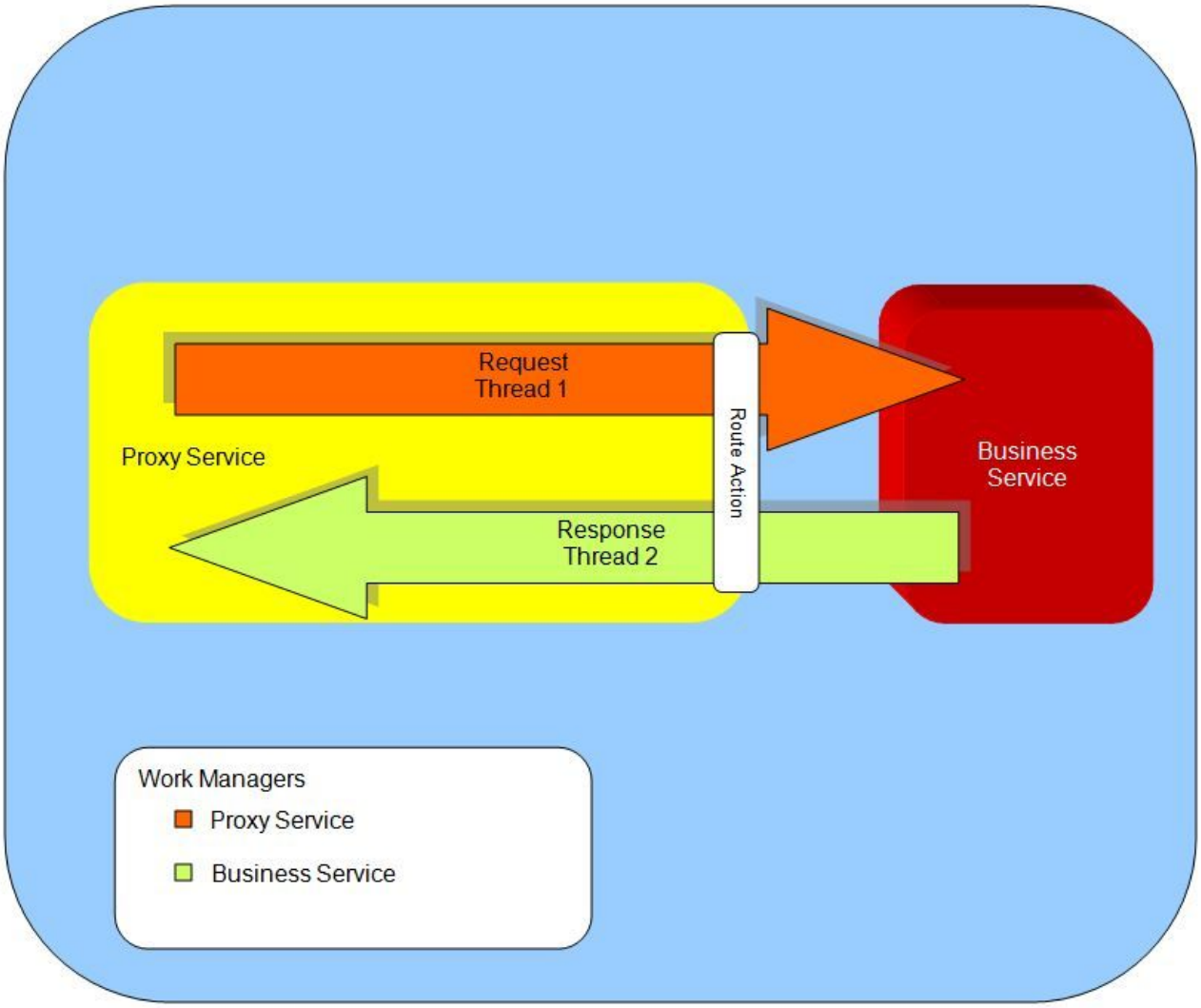

Figure 1. Threads and work managers used by a proxy service routing to a business service

Publish Action

A publish is a one-way send. It provides the means of invoking an external service but without receiving a response. This is often used to provide notification of an event, such as for logging or auditing. By default, no feedback of whether the call was successful or not is returned to the pipeline thread.

For both of the above actions, the use of Exactly Once forces the request thread to block until a response is received. This allows the request thread to notify the caller of an error immediately, without callback. This behavior is also useful when attempting to throttle the number of threads simultaneously processing a given proxy service.

Service Callout Action

A service callout is implemented as a synchronous blocking call. Its design intention is to allow the service designer the ability to invoke an external service to enrich a request message prior to routing the request to the target service. While the callout is awaiting a response, the request pipeline thread blocks until a response thread notifies it that the response is ready and processing can continue.

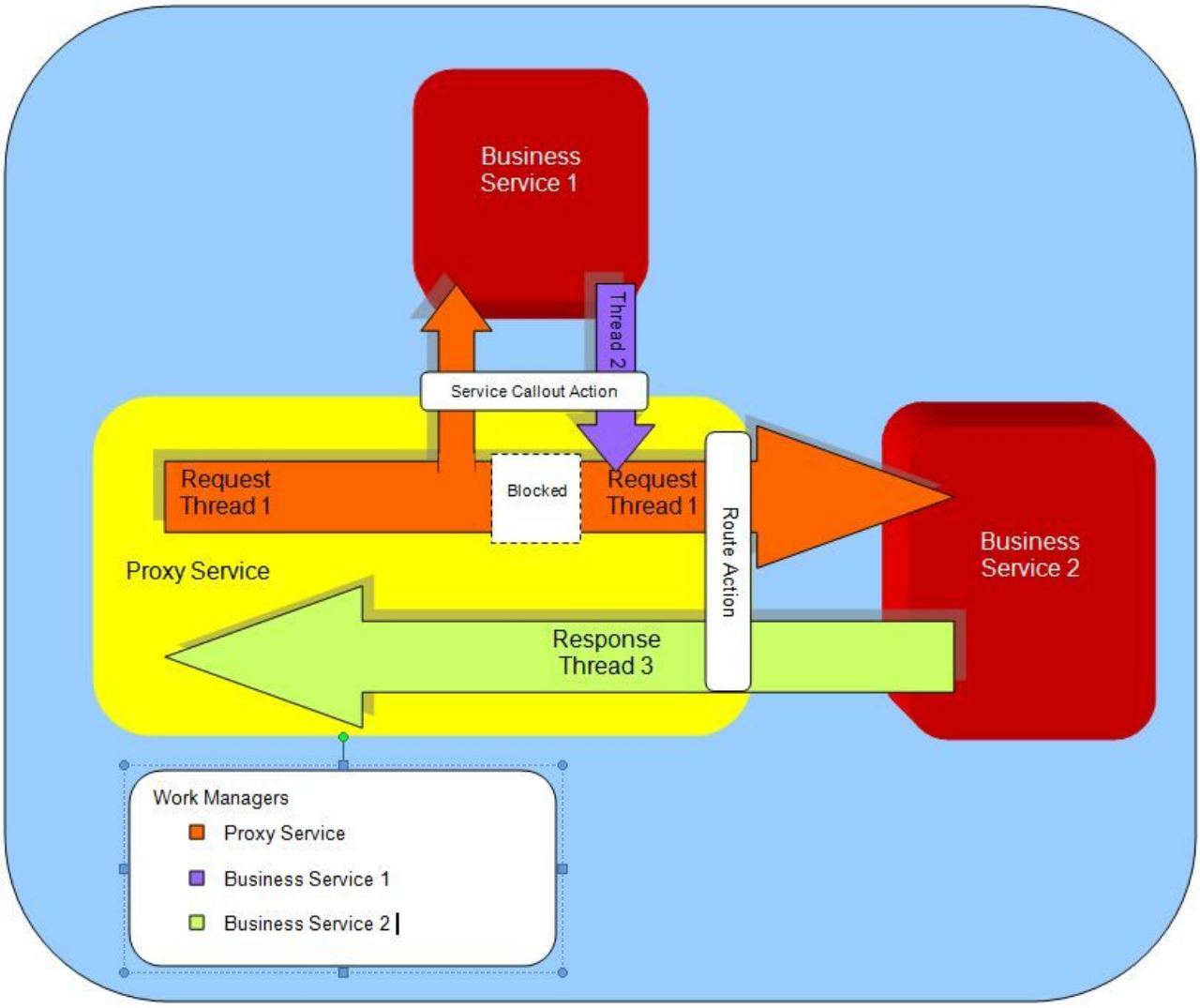

Figure 2. Threads and work manangers used for a proxy service utilizing a service callout

Work Managers

WebLogic Server utilizes a self-tuning thread pool for executing all application related work. The pool size is managed by an increment manger which adds or removes threads to the pool when it deems it necessary. Complex algorithms are used to determine when to increase pool size but the size of Active threads will never exceed a maximum of 400. As requests enter the server, a scheduler manages the order in which these requests are executed. When the number of requests exceed the number of available threads, they are queued and executed as threads return to the pool and become available. Work Managers are a means of indicating the type of work and priority of a request to the scheduler.

There are several aspects of configuring a Work Manager (Please refer to the WebLogic Server documentation for complete details), but the two most common facets are Max Thread Constraints and Min Thread Constraints. A Max Thread Constraint is a means of limiting the number of concurrent threads executing a type of request. The number of threads configured restricts the scheduler from executing more than that number at a time. However, its important to note that because the thread pool is shared amongst all Work Managers, it does not guarantee that number of threads will be available for processing at any given time. Some users confuse the notion of constraints with the idea of establishing dedicated thread pools for a service.

A Min Thread Constraint guarantees a minimum number of threads for processing. If there aren't sufficient threads available in the thread pool to process up to the minimum number, the scheduler will use Standby threads to satify the minimum. The significance of this is that Standby threads are not counted as part of the maximum number of 400 threads in the pool. When a thread is executing a request associated with a work manager containing a Min Thread Constraint, rather than returning to the free pool it first checks the queue for another request associated with the same constraint and executes it. For this reason the use of Min Thread Constraints must be done judiciously. Over using Min Thread Constraints can cause resource starvation of the Default work manager, leading to unpredicatble results.

Work managers are defined through the WLS console. By default, they are “fair share” work managers with a share value of 50. What this means is that when two work managers are created, they both have equal priority with the scheduler. When attempting to execute waiting requests, the scheduler ensures that requests associated with each work manager are given an equal number of thread resources (assuming an equal number of waiting requests).

From within OSB, the user associates a work manager to a service by configuring a dispatch policy, when supported by the transport used for that service. OSB's HTTP Transport supports the use of Work Managers. Given a simple proxy service that routes to a business service, assigning different work managers to each service allows the scheduler to recognize that new requests are different work from responses received. Subsequently, each work manager is scheduled evenly when work requests (either a new incoming request message for a proxy or a response message for a business service) become queued up. This becomes vital when a business service is invoked with a blocking call, such as with a service callout.

Another key point to remember is that once a thread is processing under the designation of a given work manager, it will continue to do so until it is returned to the pool. Therefore, when invoking a business service or a proxy service from within a pipeline of another proxy, that currently executing thread will continue processing under the work manager of the initial proxy service. Any work manager configured on the service being called is ignored in this case.