Parallel invocation of synchronous services in WebSphere Enterprise Service Bus

Martin Ross
Geza Geleji

This article describes the design pattern for the parallel invocation of synchronous services using the aggregation function available for WebSphere ESB mediation flows. It takes you from learning about the basic mediation primitives to optimizing ESB performance.

Introduction

IBM® WebSphere® Enterprise Service Bus (hereafter called WebSphere ESB) provides the Callout Node and Service Invoke mediation primitives to enable mediation flows to access external services. WebSphere ESB also gives you the choice of a synchronous or asynchronous invocation style. In general, synchronous service invocations are recommended, because they have less processing overhead and provide better performance. In some cases however, asynchronous invocations can reduce the overall response time of the application and are preferred, such as in the simultaneous invocation of multiple long-running services.

Aggregation blocks can be used to carry out concurrent asynchronous invocations of multiple services, and this article explains how to implement and ensure the optimal performance of this design pattern.

Asynchronous invocation of synchronous services

The choice of invocation style of the Service Invoke mediation primitives can have a profound impact on performance, so exercise care when making this choice. Asynchronous invocations have more overhead because they must be serialized, delivered through the messaging layer, and deserialized at the receiving thread. But asynchronous invocations are preferred in scenarios where the caller does not want to be blocked waiting for a request to be completed. One such scenario is the simultaneous invocation of multiple services that exhibit measurable latency in an effort to reduce the overall response time of the application. Here is a summary of the three invocation styles:

Table 1. Invocation styles used by the Service Invoke mediation primitive

<table>
<thead>
<tr>
<th>Invocation Style</th>
<th>Description</th>
</tr>
</thead>
</table>

© Copyright IBM Corporation 2013
Parallel invocation of synchronous services in WebSphere Enterprise Service Bus
This article uses the "Asynchronous with deferred response" invocation style, and in the design pattern described below, invocations from within an aggregation block must use this invocation style to achieve concurrent invocations of synchronous services. The diagram below shows the steps in an asynchronous invocation of a synchronous service from within an aggregation block:

**Figure 1. Asynchronous invocation of a synchronous service**

1. A Service Invoke mediation primitive invokes a synchronous service asynchronously. The request message is serialized and placed on an SCA queue point associated with the mediation flow. Processing of the aggregation block continues before the thread waits for responses.
2. An activation specification invokes an MDB with the request message.
3. The MDB synchronously invokes the service provider.
4. The service provider responds to the request.
5. The MDB puts the response message back to the SCA queue point associated with the mediation flow.
6. The response message is passed back to the originating mediation flow thread to continue processing.
Mediation primitives

This section describes the core mediation primitive components used in the development of aggregation design patterns to perform parallel asynchronous invocation of multiple long-running synchronous services.

Fan Out

The Fan Out mediation primitive is used at the start of an aggregation block. Two modes are supported:

- **Once mode** -- Provides the ability to branch the mediation flow
- **Iterate mode** -- Provides the ability to iterate over a repeating structure within the Service Message Object (SMO)

Use the Once mode to branch the mediation flow for aggregation design patterns where, for example, an inbound message contains a structure that needs to be forwarded on to multiple services. Use the Iterate mode for aggregation design patterns where a service needs to be invoked multiple times based upon a repeating structure within the Service Message Object (SMO).

![Figure 2. Fan Out mediation primitive](image)

Fan In

The Fan In mediation primitive is used at the end of an aggregation block, and partners a corresponding Fan Out primitive. Fan In supports three types of decision points that control when the output terminal of the Fan In primitive is fired -- when the specified decision point has been met and the last message that was received is propagated to the output terminal:

- **Simple count** -- The output terminal is fired when a specified number of messages is received at the input terminal.
- **XPath decision** -- The output terminal is fired when the XPath expression evaluates to true.
- **Iterate** -- The output terminal is fired when the input terminal has received all messages produced by the corresponding Fan Out mediation primitive.

The XPath decision and Simple count decision points may be reached multiple times in an aggregation solution, causing the output terminal to fire multiple times. The Iterate decision point causes the output terminal to fire only once.
Figure 3. Fan In mediation primitive

Service Invoke

The Service Invoke mediation primitive is used between the Fan Out and Fan In mediation primitives to invoke the services required. When you add the Service Invoke mediation primitive to the canvas in IBM Integration Developer, you select the reference and operation to be executed by the primitive.

The Async timeout property specifies the time to wait for a response when a call is asynchronous with a deferred response. Invocation style defines whether the service is invoked synchronously or asynchronously.

Figure 4. Service Invoke mediation primitive

Parallel invocation of synchronous services in aggregation design patterns

Aggregation is an important ESB pattern. It enables a single inbound request to map into multiple outbound service invocations, the responses from which can be aggregated into a single response.
back to the original request. You can implement the design pattern described in this article using either the Once or Iterate modes of operation described previously, enabling multiple asynchronous invocations of a single or multiple target services in parallel based on the same inbound request.

In the following banking scenario, two services are invoked for a single inbound request. The inbound request contains information for a specific account, and a Message Element Setter mediation primitive stores a RequestID field in the transient context of the SMO prior to the request entering the aggregation block. The Fan Out mediation primitive is configured in Once mode and the mediation flow is split into two branches: the first one invokes a service to retrieve a summary of the account information, and the second one invokes a service to retrieve the transaction history of the account. The Fan In mediation primitive is configured to fire the output terminal after the input terminal has been fired twice.

Prior to the invocation of each service, a BOMapper mediation primitive converts the message to the correct format, and a BOMapper mediation primitive immediately following each Service Invoke mediation primitive then stores the given responses in the shared context of the SMO. The output terminal of the Fan In mediation primitive is fired when the two response are received on the input terminal, and a final BOMapper mediation primitive aggregates the information stored within the shared context and transient context to create a response to the initial request:

**Figure 5. Aggregation mediation flow**

Configuring the invocation style of the Service Invoke mediation primitives to be asynchronous enables parallel invocation of the synchronous services.

**Performance considerations**

The invocation style of the Service Invoke mediation primitives can have a profound impact on performance, so exercise care when choosing the invocation style:

- **Synchronous** -- Each branch and thus each service invocation is processed sequentially, and the overall response time of the flow will be at least the sum of the response times of the services.
- **Asynchronous** -- Each branch executes in turn until encountering a Service Invoke mediation primitive, at which point the service invocation is passed on to a separate thread through the messaging layer, and processing of the mediation flow thread moves to the next branch. All service invocations are made before looking for responses from the service providers, at which point execution of the respective branch continues. The service
invocations are performed concurrently, and therefore the overall response time will be at least equal to the longest response time of the services invoked.

If the sum of expected response times for the services being invoked is not significantly greater than the longest response time of any one service, then due to the additional logic and processes that asynchronous invocation incurs, synchronous invocation may provide better performance.

When developing the mediation flow, pay attention to the order in which the branches are wired and the order in which the services are invoked in setting appropriate Async time-out values. Invoke services with largest expected latency first, so that external processing can occur while the other services are being invoked concurrently. Appropriate time-out values should also be configured, since they are used by the aggregation logic to determine which responses to look for first. Setting the time-out values appropriately for the services being invoked ensures that the aggregation logic works effectively and efficiently.

Because implementation of this design pattern results in additional processing within the SCA layer, there are additional performance tuning recommendations in addition to tuning for web services and ensuring that the server is configured correctly. The MDB that is used in the process runs on a separate thread pool that needs to be sized correctly to prevent it from becoming a bottleneck. Also, because of the increased use of the SCA queues, you need to ensure that the JMS resource artifacts and messaging engine are tuned appropriately.

**Java heap size**

An inappropriate Java heap size can have undesired effects such as increased Garbage Collection (GC) pause times, Java heap exhaustion, and even memory errors. Java provides the runtime environment for WebSphere ESB, and therefore Java configuration impacts performance and system resource consumption.

If the Java heap is too small, GC will be triggered too often and you may exhaust the available memory. Increasing the Java heap size will solve this problem, but it will increase the interval between GCs and allow more objects to accumulate on the Java heap, which means that when a GC is triggered, more time will be required to process an allocation failure and the application will be less responsive. The Java heap should be sized for between 40% and 70% occupancy, and the Generational GC policy (-Xgcpolicy:gencon) is recommended due to the typically short-lived transactional nature of objects in WebSphere ESB workloads.

Servers hosting mediation modules will be processing different workloads from the servers hosting messaging engines, and therefore each server should be tuned independently for the workloads they are processing. For more information and best practices regarding sizing and tuning the Java heap, see Related topics at the bottom of the article.

**Thread pools**

WebSphere ESB servers use thread pools to manage concurrent tasks, and the size of these thread pools affects the ability of a server to run applications concurrently.
To set the Maximum Size property of a thread pool in the Admin Console, click Servers => Application Servers and select the server name whose thread pool you want to manage. Click Additional Properties => Thread Pools and then the thread pool name.

The WebContainer thread pool is used for handling incoming HTTP and web services requests. This thread pool is shared by all applications deployed on the server, and you must tune it for the required concurrency of inbound requests to the server.

SCA Module MDBs use the Platform Messaging Component SPI Resource Adapter. It uses the default thread pool for the MDB threads, but the default thread pool is shared by many WebSphere Application Server tasks, so you should separate the execution of the MDBs to a dedicated thread pool. To change the thread pool used for the SCA Module MDBs:

1. Create a new thread pool (for example, SCAMDBThreadPool) on the server: Click Servers => Application Servers, select the server name, click Additional Properties => Thread Pools, and then click New.
2. Open the Platform Messaging Component SPI Resource Adapter from the Admin Console with server scope: Click Resources => Resource Adapters => Resource adapters. Then go to Preferences and select Show built-in resources.
3. Change the thread pool alias from Default to SCAMDBThreadPool.
4. Repeat steps 2 and 3 for the Platform Messaging Component SPI Resource Adapter at the node and cell scope.
5. Restart the server for the changes to become effective.

The SCAMDBThreadPool must then be sized for the expected concurrency of the SCA Module MDBs, which should be equal to the expected concurrency of inbound requests to the mediation module multiplied by the number of services being invoked concurrently using the described design pattern.

The com.ibm.websphere.webservices.http.maxConnection property specifies the maximum number of connections that are created in the HTTP outbound connector connection pool. You can configure the property only as a JVM custom property: From the Admin Console, click Servers => Application Servers, select the server, click Java and Process Management => Process Definition => Java Virtual Machine => Custom Properties, and then create a new property if it does not already exist. This property affects all web services HTTP connections made within one JVM. When the maximum number of connections is reached, no new connections are created, and the HTTP connector waits for a current connection to return to the connection pool. Size the maximum number of connections to be equal to the size of the SCAMDBThreadPool.

**Messaging**

Each mediation module has a corresponding Activation Specification in the Java Naming and Directory Interface (JNDI), which will contain the mediation module name with the suffix _AS. The default value for maxConcurrency of the Activation Specification is 10, which means that up to 10 business objects from the JMS queue can be delivered to the MDB threads concurrently. This value should be equal to the expected concurrency of inbound requests to the mediation
module multiplied by the number of services being invoked concurrently using the described design pattern.

The maximum batch size in the activation specification also has an impact on performance. The default value is 1. The maximum batch size determines how many messages are taken from the messaging layer and delivered to the application layer in a single step. Increasing this value slightly can improve performance in some circumstances, but response times for the first messages added to the batch may increase while waiting for the batch to complete. Therefore you should usually leave this value at 1 for this design pattern.

It is also recommended that you configure the Quality of Service (QoS) for the SCA destination appropriately. The default reliability is Assured persistent, but since this article describes processing web services and HTTP workloadadmins, you should reduce the reliability level to Express nonpersistent to improve performance. You can configure the SCA destination from the Admin Console: Click Buses => sca_system_bus_name => Destinations => destination_name, where destination_name contains the mediation module name with the prefix scala. Disable the Enable producers to override default reliability check box, and change the default and maximum reliability values to Express nonpersistent.

The sib.msgstore.cachedDataBufferSize of the messaging engine should also be sized appropriately for the workload. The cached data buffer is used to optimise the performance of the messaging engine by caching in memory data that the messaging engine might otherwise read from the data store. The size of this cache is obviously governed by memory constraints, but should be at least two times the expected concurrency of inbound requests to the mediation module multiplied by the number of services being invoked concurrently, using the described design pattern times the average message size. Set the cache size in the Admin Console: Select Service integration => Buses => bus_name => [Topology] Messaging engines => engine_name => [Additional Properties] Custom properties.

Initial Configuration
If you are unsure about how to select initial settings or the expected concurrency within the system, here are some starting points from which to tune through testing with representative workloads:

Table 2. Initial configuration settings

<table>
<thead>
<tr>
<th>Component</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java heap</td>
<td>-Xms20480m -Xmx2048m -Xgcpolicy:gencon -Xmn1024m</td>
</tr>
<tr>
<td>Thread pools</td>
<td>WebContainer: 50, SCAMDBThreadPool: 50 * Number of parallel service invocations</td>
</tr>
<tr>
<td>HTTP / web services</td>
<td>com.ibm.websphere.webservices.http.maxConnection=50* Number of parallel service invocations</td>
</tr>
<tr>
<td>Activation specification</td>
<td>maxConcurrency: 50* Number of parallel service invocations, maxBatchSize: 1</td>
</tr>
<tr>
<td>SCA queue</td>
<td>Disable the Enable producers to override default reliability check box. The default reliability can be Express nonpersistent or Maximum Reliability: Express nonpersistent</td>
</tr>
</tbody>
</table>
**Conclusions**

Implementing the parallel invocation of synchronous services in aggregation design patterns can significantly improve application response time and performance, but performance tuning is required to ensure a successful solution:

- Size the Java heap for each server in accordance with the workload for that server.
- Configure a dedicated thread pool for the Platform Messaging SPI Resource Adapter and size the thread pools relative to the expected and required concurrency within the server.
- Increase the maximum number of HTTP connections as required.
- Tune the activation specification used by the SCA Module MDB for the required concurrency.
- Appropriately define the QoS for the SCA queue definition.
- Tune `sib.msgstore.cachedDataBufferSize` for the messaging engine.
- Take care regarding the order in which the services are invoked and the async timeouts defined.
Related topic

- WebSphere ESB resources
  - WebSphere ESB documentation in IBM Knowledge Center
  - WebSphere ESB Development Guide
  - WebSphere ESB product page
  - WebSphere ESB documentation library
  - Tuning the IBM virtual machine for Java
  - HTTP transport custom properties for Web services applications
  - How to process mediations in parallel
  - Asynchronous processing in WebSphere Process Server

© Copyright IBM Corporation 2013
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)