**Scenario**

This sample demonstrates how to create a stateful actor service that supports long running computations in Azure Service Fabric. In some situations, worker actors receive messages from a client application that need to be processed by a complex, multistage elaboration that takes a relatively long time to complete. While processing a message, the worker actor should be able to receive and elaborate more messages in parallel, or, when messages need to be processed in a strict chronological order, the worker actor should be able to store incoming messages in a persistent queue waiting to be processed. Think about an IoT scenario where a Service Fabric application implements a command and control pattern to ingest, process and react to telemetry events generated by thousands of devices. In this context, a separate actor represents and handles a single device, receives and processes its events, generates and transmits commands to it. When message elaboration takes a long time to complete, the actor needs to be able to receive messages from the associated device while processing. This is particularly important when the application is subject to a sudden traffic burst. The Service Fabric actor model supports a strict, turn-based model for invoking actor methods. This means that no more than one thread can be active inside the actor code at any time. A turn consists of the complete execution of an actor method in response to a request from other actors or clients, or the complete execution of a timer/reminder callback. Even though these methods and callbacks are asynchronous, the Actors runtime does not interleave them. A turn must be fully finished before a new turn is allowed. In other words, an actor method or timer/reminder callback that is currently executing must be fully finished before a new call to a method or callback is allowed. A method or callback is considered to have finished if the execution has returned from the method or callback and the task returned by the method or callback has finished. It is worth emphasizing that turn-based concurrency is respected even across different methods, timers, and callbacks. The Actors runtime enforces turn-based concurrency by acquiring a per-actor lock at the beginning of a turn and releasing the lock at the end of the turn. Thus, turn-based concurrency is enforced on a per-actor basis and not across actors. Actor methods and timer/reminder callbacks can execute simultaneously on behalf of different actors.

In our scenario, the turn-based model for accessing actor methods could be an issue: while an actor is executing a long-running method, it cannot receive and process other requests. This sample demonstrates how to solve this problem. In particular, the proposed solution shows how to use stateful actors to:

* Parallel execution:
  + Receive and process multiple messages in parallel
  + Monitor the completion of individual message processing.
  + Stop the long running elaboration of a given message.
  + Collect and aggregate results in the internal state of the actor.
* Sequential processing:
  + Receive and process multiple messages in a rigorous chronological order
  + Monitor the completion of individual message processing.
  + Stop the long running elaboration of a given message.
  + Collect and aggregate results in the internal state of the actor

This sample also demonstrates how to:

* Create a stateful actor class that inherits from the [StatefulActor](https://msdn.microsoft.com/en-us/library/microsoft.servicefabric.actors.statefulactor.aspx" \o "StatefulActor) class and manually handles the internal state using its[StateManager](https://msdn.microsoft.com/en-us/library/microsoft.servicefabric.actors.iactorstatemanager.aspx) property.
* How to inherit a stateful actor from an abstract base class.
* How to create a custom collection using a stateful actor class that inherits from the [StatefulActor](https://msdn.microsoft.com/en-us/library/microsoft.servicefabric.actors.statefulactor.aspx" \o "StatefulActor) class and manually handles the internal state using its [StateManager](https://msdn.microsoft.com/en-us/library/microsoft.servicefabric.actors.iactorstatemanager.aspx" \o "StateManager) property.
* How to run multiple actor services within the same process.

For more information on the turn-based model for accessing actor methods, see the following resources:

* [Introduction to Service Fabric Reliable Actors](https://azure.microsoft.com/en-gb/documentation/articles/service-fabric-reliable-actors-introduction/)
* [Reliable Actors reentrancy](https://azure.microsoft.com/en-gb/documentation/articles/service-fabric-reliable-actors-reentrancy/)
* [Actor Timers and Actor Reminders](https://azure.microsoft.com/en-gb/documentation/articles/service-fabric-reliable-actors-timers-reminders/)

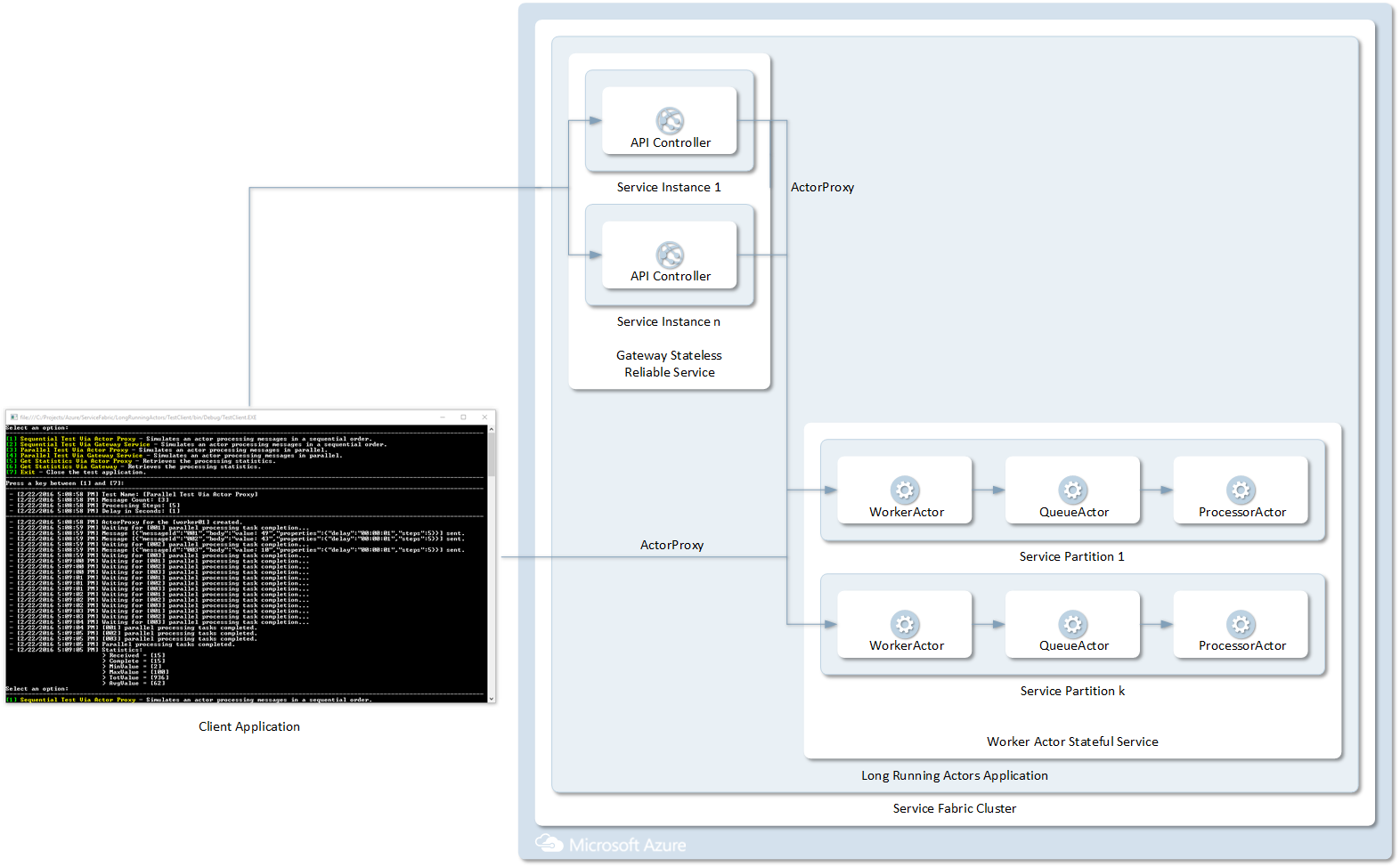
**License**

Microsoft Corporation (“Microsoft”) grants you a nonexclusive, perpetual, royalty-free right to use and modify the software code provided by us for the purposes of illustration  ("Sample Code") and to reproduce and distribute the object code form of the Sample Code, provided that you agree: (i) to not use our name, logo, or trademarks to market your software product in which the Sample Code is embedded; (ii) to include a valid copyright notice on your software product in which the Sample Code is embedded; and (iii) to indemnify, hold harmless, and defend us and our suppliers from and against any claims or lawsuits, whether in an action of contract, tort or otherwise, including attorneys’ fees, that arise or result from the use or distribution of the Sample Code or the use or other dealings in the Sample Code. Unless applicable law gives you more rights, Microsoft reserves all other rights not expressly granted herein, whether by implication, estoppel or otherwise.

THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL MICROSOFT OR ITS LICENSORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THE SAMPLE CODE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

**Architecture Design**

The following picture shows the architecture design of the application.

[](https://raw.githubusercontent.com/paolosalvatori/LongRunningActors/master/Images/Architecture.png?token=ABlOM_daQKkKUL5DdzgsQv97-FqnRp-zks5W4pFiwA%3D%3D)

**Message Flow**

1. A console application can be used to send a configurable amount of messages to a worker actor using one of the following options:
   * Directly via an instance of the [ActorProxy](https://msdn.microsoft.com/en-us/library/microsoft.servicefabric.actors.actorproxy.aspx) class. This option can be used to debug and test the application on the local cluster or to simulate a context in which the worker actor is invoked by another service running on the same Service Fabric cluster.
   * Via a gateway service implemented by an ASP.NET Web API REST service running in a stateless reliable service. The service in question uses an OWIN listener to host the service. For more information, see [Get started: Service Fabric Web API services with OWIN self-hosting](https://azure.microsoft.com/en-us/documentation/articles/service-fabric-reliable-services-communication-webapi/).

See below for more details on how to configure and use the client application.

1. Incoming messages are processed by a stateful Worker Actor. This actor allows to process incoming messages in a parallel or in strict chronological order, to monitor or stop the elaboration, collect and aggregate results in the internal state. See below for more details.

In the demo, the client application can be used to send one or more messages to the worker actor and then it starts checking the completion state of message processing by invoking a method exposed by the actor every second. This mechanism is not necessary as the client could send the messages to process to the worker actor using a **fire-and-forget** approach without checking for the completion of their processing. For completeness, the sample implements also a mechanism to monitor the processing state of both parallel and sequential processing tasks as well as a mechanism to eventually stop them, but using this pattern is optional. The worker actor is used to accumulate the results produced by the elaboration of individual messages. In the demo, the elaboration of each message produces a random number between 1 and 100 to simulate a real computation. Both processing and sequential processing tasks invoke the worker actor to notify their completion and transmit the return value. The worker actor changes its persistent state to update the completion state of the processing tasks and updates the following statistics:

* number of received messages
* number of complete messages
* number of stopped messages
* minimum value
* maximum value
* total value
* average value
* latest N messages received and corresponding result, where N is configurable

The sample can easily be changed to replace the code that emulates message elaboration with a real processing code. In addition, the processing logic can be changed to write or send the return value of each message processing to an external repository or service. For example, in a IoT scenario, the worker actor could eventually send a command to an external device, directly or indirectly via an outbound actor or by sending a C2D message to an IoT Hub.

**Service Fabric Application**

The Service Fabric application is composed of three services:

* **GatewayService**: this is a stateless reliable service running an **OWIN** listener and exposing an **ASP.NET Web API REST** service that is used as a gateway in front of the stateful actor service.
* **WorkerActorService**: this is a stateful actor service hosting three actor services:
  + **WorkerActorService**: this service is responsible to handle the interactions with external services, start message processing, monitor or stop the message elaboration, collect and aggregate results.
  + **QueueActor**: this actor inherits from the **CircularQueueActor** abstract class which implements a circular queue. The reason why the queue has been implemented with a separate queue rather than with one of the collections provided by the .NET framework is twofold:
    - Minimize the footprint of the state portion to write or read whenever an enqueue or dequeue operation is executed.
    - Extend the intrisic semantics of enqueue and dequeue operations with custom logic.
  + **ProcessorActor**: this actor is responsible for processing messages in a chronological order.

**Note**: one of the advantages of stateless services over stateful services is that by specifying **InstanceCount="-1"** in the**ApplicationManifest.xml**, you can create an instance of the service on each node of the Service Fabric cluster. When the cluster uses [Virtual Machine Scale Sets](https://azure.microsoft.com/en-gb/documentation/articles/virtual-machines-vmss-overview/) to to scale up and down the number of cluster nodes, this allows to automatically scale up and scale down the number of instances of a stateless service based on the autoscaling rules and traffic conditions.

**The Worker Actor**

The following table contains the actor interface implemented by the **WorkerActor** class.

namespace Microsoft.AzureCat.Samples.WorkerActorService.Interfaces

{

/// <summary>

/// This interface represents the actions a client app can perform on an actor.

/// It MUST derive from IActor and all methods MUST return a Task.

/// </summary>

public interface IWorkerActor : IActor

{

/// <summary>

/// Starts processing a message in sequential order.

/// </summary>

/// <param name="message">The message to process.</param>

/// <returns>True if the operation completes successfully, false otherwise.</returns>

Task<bool> StartSequentialProcessingAsync(Message message);

/// <summary>

/// Starts processing a message on a separate task.

/// </summary>

/// <param name="message">The message to process.</param>

/// <returns>True if the operation completes successfully, false otherwise.</returns>

Task<bool> StartParallelProcessingAsync(Message message);

/// <summary>

/// Stops the sequential processing task.

/// </summary>

/// <returns>True if the operation completes successfully, false otherwise.</returns>

Task<bool> StopSequentialProcessingAsync();

/// <summary>

/// Stops the elaboration of a specific message identified by its id.

/// </summary>

/// <param name="messageId">The message id.</param>

/// <returns>True if the operation completes successfully, false otherwise.</returns>

Task<bool> StopParallelProcessingAsync(string messageId);

/// <summary>

/// Used by the sequential processing task to signal the completion

/// of a message processing and return computed results.

/// </summary>

/// <param name="messageId">The message id.</param>

/// <param name="returnValue">The message processing result.</param>

/// <returns>True if the operation completes successfully, false otherwise.</returns>

Task<bool> ReturnSequentialProcessingAsync(string messageId, long returnValue);

/// <summary>

/// Used by the parallel processing task to signal the completion

/// of a message processing and return computed results.

/// </summary>

/// <param name="messageId">The message id.</param>

/// <param name="returnValue">The message processing result.</param>

/// <returns>True if the operation completes successfully, false otherwise.</returns>

Task<bool> ReturnParallelProcessingAsync(string messageId, long returnValue);

/// <summary>

/// Checks if the sequential processing task is running.

/// </summary>

/// <returns>True if sequential processing task is still running, false otherwise.</returns>

Task<bool> IsSequentialProcessingRunningAsync();

/// <summary>

/// Checks if the elaboration of a given message is running.

/// </summary>

/// <param name="messageId">The message id.</param>

/// <returns>True if the elaboration of the message is still running, false otherwise.</returns>

Task<bool> IsParallelProcessingRunningAsync(string messageId);

/// <summary>

/// Sets sequential processing state.

/// </summary>

/// <param name="runningState">True if the sequential processing task is still running, false otherwise.</param>

/// <returns>True if the operation completes successfully, false otherwise.</returns>

Task<bool> SetSequentialProcessingStateAsync(bool runningState);

/// <summary>

/// Gets the worker actor statistics from its internal state.

/// </summary>

/// <returns>The worker actor statistics.</returns>

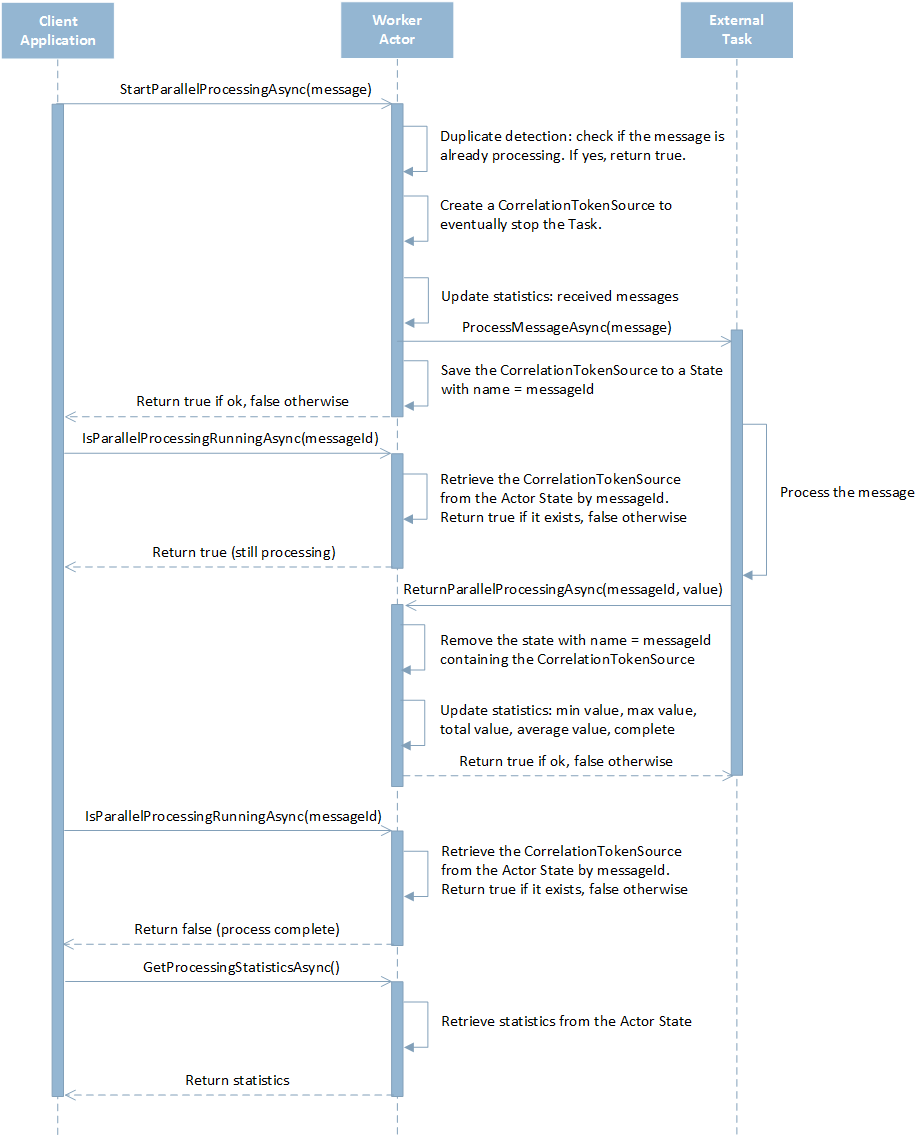
Task<Statistics> GetProcessingStatisticsAsync();

}

}

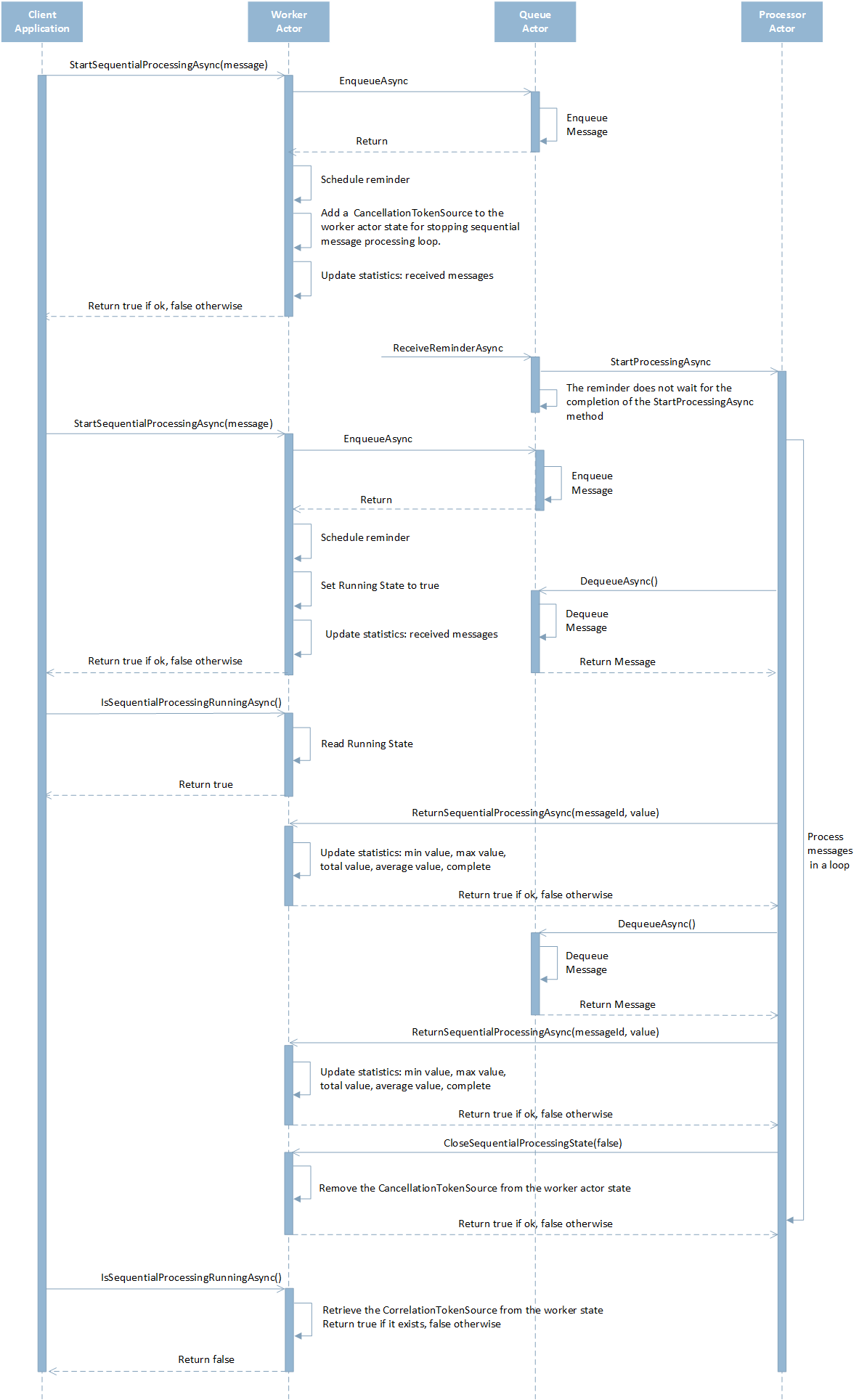
**Parallel Processing**

The following diagram shows the sequence diagram for the parallel message processing pattern.

[](https://raw.githubusercontent.com/paolosalvatori/LongRunningActors/master/Images/Parallel.png?token=ABlOM2Ho7g__EIWBE5BSB4PZxocyG45Xks5W4pGrwA%3D%3D)

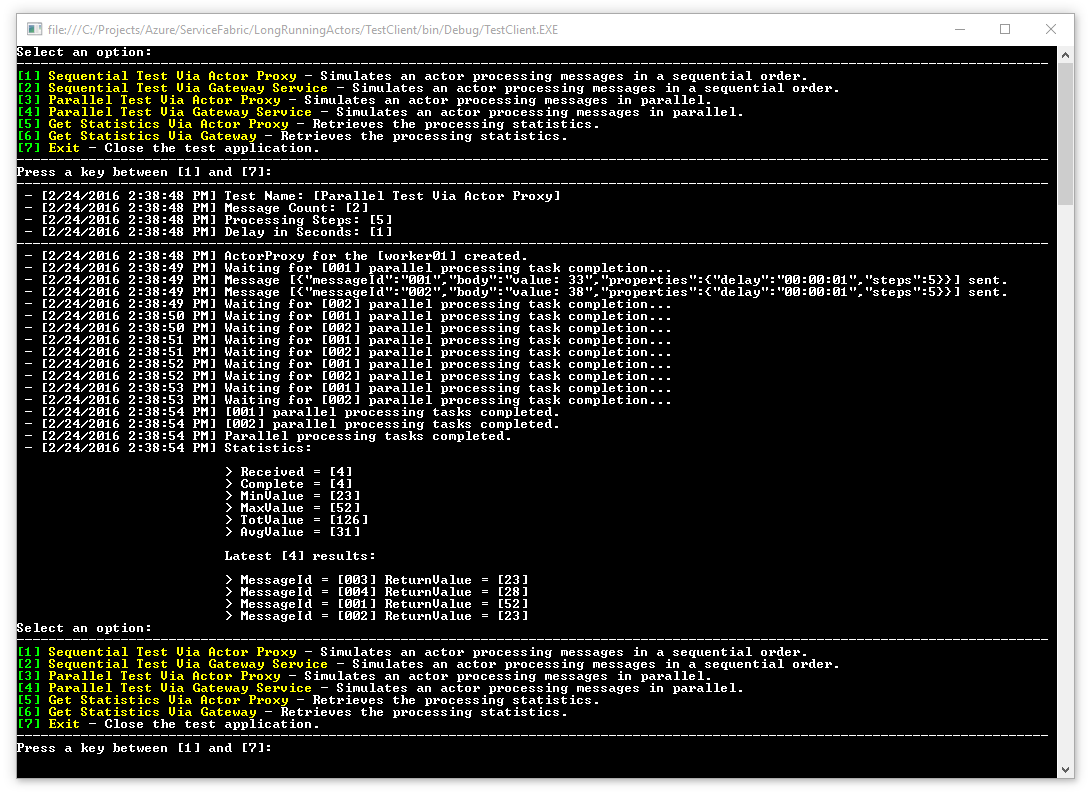
**Sequential Processing**

The following diagram shows the sequence diagram for the sequential message processing pattern.

[](https://raw.githubusercontent.com/paolosalvatori/LongRunningActors/master/Images/Sequential.png?token=ABlOMxbyCRUPV2AoHfdEHdPURSTDc7auks5W4pHCwA%3D%3D)

**Client Application**

The image belowshows the options offered by the client application to test the parallel and seuqntial message processing patterns directly via an **ActorProxy** object or via the **GatewayService**.

[](https://raw.githubusercontent.com/paolosalvatori/LongRunningActors/master/Images/Client.png?token=ABlOM_CV4k3GqlIYr90x90gkekWXEzjJks5W4pHkwA%3D%3D)

The following table contains the settings defined in the configuration file of the client application:

<?xml version="1.0" encoding="utf-8"?>

<configuration>

<startup>

<supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.1"/>

</startup>

<appSettings>

<!--The Gateway URL. Use:

- http://localhost:8082/worker when testing the application on the local Service Fabric cluster

- http://<SF\_CLUSTER\_NAME>.<REGION>.cloudapp.azure.com:8082/worker

when testing the application on Service Fabric cluster on Azure

-->

<add key="gatewayUrl" value="http://localhost:8082/worker"/>

<!-- The count of messages to send to the worker actor -->

<add key="messageCount" value="3"/>

<!-- the number of steps emulated by the worker or processor actor for each message -->

<add key="steps" value="5"/>

<!-- the time spent by each step in seconds -->

<add key="delay" value="1"/>

</appSettings>

</configuration>

**Service Fabric Configuration Files**

**ApplicationParameters\Local.xml** file in the **LongRunningActors** project:

<?xml version="1.0" encoding="utf-8"?>

<Application xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

Name="fabric:/LongRunningActors"

xmlns="http://schemas.microsoft.com/2011/01/fabric">

<Parameters>

<Parameter Name="GatewayService\_InstanceCount" Value="1" />

<Parameter Name="GatewayService\_ServiceRelativePath" Value="worker" />

<Parameter Name="GatewayService\_MaxRetryCount" Value="3" />

<Parameter Name="GatewayService\_BackoffDelay" Value="1" />

<Parameter Name="WorkerActorService\_PartitionCount" Value="1" />

<Parameter Name="WorkerActorService\_QueueLength" Value="5" />

<Parameter Name="QueueActorService\_PartitionCount" Value="1" />

<Parameter Name="ProcessorActorService\_PartitionCount" Value="1" />

</Parameters>

</Application>

**ApplicationParameters\Cloud.xml** file in the **LongRunningActors** project:

<?xml version="1.0" encoding="utf-8"?>

<Application xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

Name="fabric:/LongRunningActors"

xmlns="http://schemas.microsoft.com/2011/01/fabric">

<Parameters>

<Parameter Name="GatewayService\_InstanceCount" Value="-1" />

<Parameter Name="GatewayService\_ServiceRelativePath" Value="worker" />

<Parameter Name="GatewayService\_MaxRetryCount" Value="3" />

<Parameter Name="GatewayService\_BackoffDelay" Value="1" />

<Parameter Name="WorkerActorService\_PartitionCount" Value="5" />

<Parameter Name="WorkerActorService\_QueueLength" Value="5" />

<Parameter Name="QueueActorService\_PartitionCount" Value="1" />

<Parameter Name="ProcessorActorService\_PartitionCount" Value="1" />

</Parameters>

</Application>

**ApplicationManifest.xml** file in the **LongRunningActors** project:

<?xml version="1.0" encoding="utf-8"?>

<ApplicationManifest xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

ApplicationTypeName="LongRunningActorsType"

ApplicationTypeVersion="1.0.0"

xmlns="http://schemas.microsoft.com/2011/01/fabric">

<Parameters>

<Parameter Name="QueueActorService\_PartitionCount" DefaultValue="5" />

<Parameter Name="QueueActorService\_MinReplicaSetSize" DefaultValue="2" />

<Parameter Name="QueueActorService\_TargetReplicaSetSize" DefaultValue="3" />

<Parameter Name="ProcessorActorService\_PartitionCount" DefaultValue="5" />

<Parameter Name="ProcessorActorService\_MinReplicaSetSize" DefaultValue="2" />

<Parameter Name="ProcessorActorService\_TargetReplicaSetSize" DefaultValue="3" />

<Parameter Name="WorkerActorService\_PartitionCount" DefaultValue="5" />

<Parameter Name="WorkerActorService\_MinReplicaSetSize" DefaultValue="2" />

<Parameter Name="WorkerActorService\_TargetReplicaSetSize" DefaultValue="3" />

<Parameter Name="WorkerActorService\_QueueLength" DefaultValue="10" />

<Parameter Name="GatewayService\_InstanceCount" DefaultValue="-1" />

<Parameter Name="GatewayService\_ServiceRelativePath" DefaultValue="worker" />

<Parameter Name="GatewayService\_MaxRetryCount" DefaultValue="3" />

<Parameter Name="GatewayService\_BackoffDelay" DefaultValue="1" />

</Parameters>

<ServiceManifestImport>

<ServiceManifestRef ServiceManifestName="WorkerActorServicePkg" ServiceManifestVersion="1.0.0" />

<ConfigOverrides>

<ConfigOverride Name="Config">

<Settings>

<Section Name="WorkerActorCustomConfig">

<Parameter Name="QueueLength" Value="[WorkerActorService\_QueueLength]" />

</Section>

</Settings>

</ConfigOverride>

</ConfigOverrides>

</ServiceManifestImport>

<ServiceManifestImport>

<ServiceManifestRef ServiceManifestName="GatewayServicePkg" ServiceManifestVersion="1.0.0" />

<ConfigOverrides>

<ConfigOverride Name="Config">

<Settings>

<Section Name="GatewayServiceConfig">

<Parameter Name="ServiceRelativePath" Value="[GatewayService\_ServiceRelativePath]" />

<Parameter Name="MaxRetryCount" Value="[GatewayService\_MaxRetryCount]" />

<Parameter Name="BackoffDelay" Value="[GatewayService\_BackoffDelay]" />

</Section>

</Settings>

</ConfigOverride>

</ConfigOverrides>

</ServiceManifestImport>

<DefaultServices>

<Service Name="QueueActorService" GeneratedIdRef="0ee85dce-1c0a-4a09-82c2-d80d8a1ee2e8">

<StatefulService ServiceTypeName="QueueActorServiceType"

TargetReplicaSetSize="[QueueActorService\_TargetReplicaSetSize]"

MinReplicaSetSize="[QueueActorService\_MinReplicaSetSize]">

<UniformInt64Partition PartitionCount="[QueueActorService\_PartitionCount]"

LowKey="-9223372036854775808"

HighKey="9223372036854775807" />

</StatefulService>

</Service>

<Service Name="ProcessorActorService" GeneratedIdRef="ea86c350-5862-4840-a99c-552d11dbeaf5">

<StatefulService ServiceTypeName="ProcessorActorServiceType"

TargetReplicaSetSize="[ProcessorActorService\_TargetReplicaSetSize]"

MinReplicaSetSize="[ProcessorActorService\_MinReplicaSetSize]">

<UniformInt64Partition PartitionCount="[ProcessorActorService\_PartitionCount]"

LowKey="-9223372036854775808"

HighKey="9223372036854775807" />

</StatefulService>

</Service>

<Service Name="WorkerActorService" GeneratedIdRef="e0a44d7b-779e-4e80-a31c-4760b81e9891">

<StatefulService ServiceTypeName="WorkerActorServiceType"

TargetReplicaSetSize="[WorkerActorService\_TargetReplicaSetSize]"

MinReplicaSetSize="[WorkerActorService\_MinReplicaSetSize]">

<UniformInt64Partition PartitionCount="[WorkerActorService\_PartitionCount]"

LowKey="-9223372036854775808"

HighKey="9223372036854775807" />

</StatefulService>

</Service>

<Service Name="GatewayService">

<StatelessService ServiceTypeName="GatewayServiceType" InstanceCount="[GatewayService\_InstanceCount]">

<SingletonPartition />

</StatelessService>

</Service>

</DefaultServices>

</ApplicationManifest>