Scalable CXF applications using JMS transport

Java Message Service (JMS) is wide spread and popular messaging API. As JMS is standardized, the same application code can successfully work with different JMS implementations: WS MQ, Active MQ, Tibco, Joram, BEA WebLogic, OpenJMS.
CXF provides a transport that enables endpoints to use JMS queues and topics.

DEFAULT CXF CONSUMER AND PRODUCER USING JMS

Implementing CXF client and service using JMS transport is trivial.
Basically, it is enough to configure two things in WSDL:
a) specify jms transport URI in binding element;
b) define jms address in port element.

WSDL binding and port should look like:

```
<wsdl:definitions
    xmlns:jms="http://cxf.apache.org/transports/jms"
...
  <wsdl:binding name="Greeter_SOAPBinding" type="tns:Greeter">
        <soap:binding style="document" transport="http://cxf.apache.org
/transports/jms"/>
...
 </wsdl:binding>

  <wsdl:service name="JMSGreeterService">
        <wsdl:port binding="tns:JMSGreeterPortBinding" name="GreeterPort">
            <jms:address
                destinationStyle="queue"
                jndiConnectionFactoryName="ConnectionFactory"
jndiDestinationName="dynamicQueues/test.cxf.jmstransport.queue">
               <jms:JMSNamingProperty name="java.naming.factory.initial"
value="org.apache.activemq.jndi.ActiveMQInitialContextFactory"/>
                  <jms:JMSNamingProperty name="java.naming.provider.url"
value="tcp://localhost:61616"/>
           </jms:address>
        </wsdl:port>
   </wsdl:service>
```

CXF clients and servers implemented in java or using Spring configuration magically work for this WSDL (under the hood CXF selects correct JMS Conduit and Destination based on address URL).
Details are described in http://cxf.apache.org/docs/jms-transport.html.
CXF also delivers jms_queue and jms_pubsub examples illustrating using JMS transport with default settings for ActiveMQ.

SCALABILITY PROBLEMS

Unfortunately there are two main scalability drawbacks when using default JMS configuration:

1. It doesn't provide sessions pooling and consumers/producers cache ⭐.
2. Default JMS message consumer is single threaded. It means that only one thread will get messages from the queue or topic and pass them to further processing.

Both aspects are critical for enterprise applications and their implementation is not an easy task. Is there any solution? Yes: Spring JMS functionality and CXF Features. Let discuss them in detail.

⭐ - Some JMS vendors provide integrated session pooling and consumers/producers cache in ConnectionFactory. In this case using Spring CachingConnectionFactory is not necessary. Please refer vendor documentation to clear it.

SPRING JMS FUNCTIONALITY

Spring provides a number of useful classes that helps to implement scalable JMS application. Important for us are: org.springframework.jms.connection.CachingConnectionFactory
org.springframework.jms.listener.DefaultMessageListenerContainer

CachingConnectionFactory

CachingConnectionFactory provides session pooling, consumers and producers cache. Bellow is a sample configuration of CachingConnectionFactory:

```
<bean id="cachingConnectionFactory" class="org.springframework.jms.connection.
CachingConnectionFactory">
        <property name="targetConnectionFactory">
                <bean class="org.apache.activemq.ActiveMQConnectionFactory">
                        <property name="brokerURL" value="tcp://localhost:
61616" />
                </bean>
        </property>
        <property name="sessionCacheSize" value="20"/>
        <property name="cacheProducers" value="true"/>
        <property name="cacheConsumers" value="true"/>
</bean>
```

As you can see it is possible to set the size of the session pool and switch on producers and consumers caching.

DefaultMessageListenerContainer

DefaultMessageListenerContainer enables getting messages from the destination in parallel, using multiple threads.
Configuration of DefaultMessageListenerContainer looks like:

```
<bean id="queueContainerListener"
        class="org.springframework.jms.listener.
DefaultMessageListenerContainer">
                <property name="connectionFactory" ref="connectionFactory" />
                <property name="destinationName" value="Q_WM_OUT" />
                <property name="messageListener" ref="simpleListener" />
                <property name="cacheLevel" value="3" />
                <property name="concurrentConsumers" value="10" />
                <property name="maxConcurrentConsumers" value="50" />
</bean>
```

It is possible to define here:

1. Initial and maximal number of concurrent consumers. This tells the Spring to always start up a initial number of consumers (concurrentConsumers). When a new message has been received, if the maxConcurrentConsumers has not been reached, then a new consumer is created to process the message.
2. Cache level (3- cache connections, sessions and consumers; 2 – cache connections and sessions, 1 – cache connections only)
3. Specify message listener class (implementing MessageListener interface) and connection factory.

It is important to be aware of following things related to consumers caching:

- it doesn't make sense to increase the number of concurrent consumers for a JMS topic. This just leads to concurrent consumption of the same message.
- the concurrentConsumers property and the maxConcurrentConsumers property can be modified at runtime, for example, via JMX.
  For the details you can refer following Bruce Snider's blog.

You can see that Spring provides solution for both mentioned scalability aspects. But how we can use it in CXF?

CXF FEATURES

As the CXF JMS implementation is based the Spring JMS classes the user can benefit from described Spring JMS functionality.
CXF allows to configure details of the JMS transport using the JmsConfigFeature. A Feature is something that is able to customize a Server, Client, or Bus, typically adding capabilities. In our case we will add a feature in jaxws:endpoint and jaxws:client to tune the JMS transport.

Server configuration

```
<bean id="cachingConnectionFactory" class="org.springframework.jms.connection.
CachingConnectionFactory">
        <property name="targetConnectionFactory">
                <bean class="org.apache.activemq.ActiveMQConnectionFactory">
                        <property name="brokerURL" value="tcp://localhost:
61616" />
                </bean>
        </property>
        <property name="sessionCacheSize" value="20"/>
        <property name="cacheConsumers" value="true"/>
</bean>

<bean id="jmsConfig" class="org.apache.cxf.transport.jms.JMSConfiguration"
                p:connectionFactory-ref="cachingConnectionFactory"
                p:cacheLevel="3"
                p:concurrentConsumers="16"
                p:maxConcurrentConsumers="16"
                p:targetDestination="Q_HSC"
                p:wrapInSingleConnectionFactory="false"
        />

<jaxws:endpoint id=" JMSGreeterService" address="jms://"
                implementor="#JMSGreeterServiceImpl">
                <jaxws:features>
                        <bean class="org.apache.cxf.transport.jms.
JMSConfigFeature">
                                <p:jmsConfig-ref="jmsConfig">
                        </bean>
                </jaxws:features>
</jaxws:endpoint>
```

You can see that the endpoint configuration contains the JMSConfigFeature that has a JMSConfiguration property.
JMSConfiguration supports all settings that we have seen in Spring DefaultMessageListenerContainer: cached connection factory with session pool size, number of concurrent consumers, cache level. All settings of JMSConfiguration are described in details in http://cxf.apache.org/docs/using-the-jmsconfigfeature.html.
Using this configuration the server application can be tuned to achieve optimal performance in our target environment.

Client configuration

```
<bean id="cachingConnectionFactory" class="org.springframework.jms.connection.
CachingConnectionFactory">
        <property name="targetConnectionFactory">
                <bean class="org.apache.activemq.ActiveMQConnectionFactory">
                        <property name="brokerURL" value="tcp://localhost:
61616" />
                </bean>
        </property>
        <property name="sessionCacheSize" value="20"/>
        <property name="cacheProducers" value="true"/>
</bean>

<bean id="jmsConfig" class="org.apache.cxf.transport.jms.JMSConfiguration"
p:connectionFactory-ref="connectionFactory" p:targetDestination="Q_HSC"
                p:wrapInSingleConnectionFactory="false" />

<jaxws:client id="JMSGreeterService" address="jms://"
        serviceClass="com.sopera.services.tpoc.eventgenerator.EventGenerator”>
                <jaxws:features>
                        <bean class="org.apache.cxf.transport.jms.
JMSConfigFeature">
                                <property name="jmsConfig" ref="jmsConfig"/>
                        </bean>
                </jaxws:features>
        </jaxws:client>
```

Client configuration looks very similar to the server one except two things:

1. CachingConnectionFactory activates producers caching instead consumers caching;
2. JMSConfiguration hasn’t concurrent consumers settings: client concurrency is under application control and can be implemented using standard Java concurrency API.

CONCLUSION

It is possible to achieve scalability of a CXF client and service using Spring JMS functionality and the CXF JMS Configuration Feature.
It is not necessary to write any line of code, just configure and leverage already existing stuff.
Using this feature can have essential invluence on the performance for some environments: in one Prove Of Concept I have improved CXF service throughput on 360% (from 500 to 1800 msg/sec) just using session pool and multithread JMS consumer!
Reference performance numbers for SOAP over JMS are represented in Christian Schneider's article, so you can easily compare it with own results and make appropriate tunning if necessary.

REFERENCES

1. CXF JMS Transport: http://cxf.apache.org/docs/jms-transport.html
2. CXF Features: http://cxf.apache.org/docs/features.html

3. Spring JMS functionality: http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference /html/jms.html
4. Spring JMS performance tunning: http://bsnyderblog.blogspot.com/2010/05/tuning-jms-message-consumption-in.html
5. CXF performance: http://www.liquid-reality.de/pages/viewpage.action?pageId=5865562