Oracle BPEL PM - Performance Tuning und Clustering Best Practises

Stefan Koser
Oracle Deutschland GmbH
Berlin

Schlüsselworte:

BPEL, SOA, AIA

Einleitung

Der Oracle BPEL Process Manager ist eine der zentralen SOA-Kompoenten von Oracle. Mit den zahlreichen verfügbaren Einstellungen auf Applikationsserver-, BPEL-Engine-, Domänen- und Prozess-Level ist es nicht einfach eine optimale Konfiguration für die bestmögliche Performance zu finden. Der Vortrag stellt die wichtigsten Tuning-Optionen und deren Effekt vor und gibt Erfahrungen aus dem Performance Tuning einer der weltweit größten Installationen des Oracle BPEL Process Managers.

Projektszenarien

Projekt 1: Telekommunikation – Backend Order Processing mit Oracle BPEL 10gR3

In diesem Projekt wurde Siebel CRM über eine Message Oriented Middleware (MOM) – IBM Websphere MQ) – an die Backend Order-Fulfillment Prozessierungsssysteme angebunden. Mit BPEL wurden die Geschäftsprozesse zur kaufmännischen und technischen Validierung einer Order, zum Fulfillment und der Bestandsbildung abgebildet. Eine Verarbeitung einer Order dauert im Normalfall mit allen Prüfungen, die in den Backendsystemem teilweise manuell eingegeben werden, ca. 10 Tage. In Ausnahmefällen können Prozesse jedoch verzögert oder verschoben werden, was die Laufzeit auf mehrere Monate verlängern kann.

Alle BPEL Prozess sind in 3 Ebenen strukturiert, wobei die oberen 2 Ebenen als asynchrone, langlaufende BPEL Prozesse realisiert sind und die unterste Ebene synchone Services bereitstellt, die ebenfalls in BPEL realisiert wurden.
Für 4 verschiedene Produktgrupen gibt es somit jeweils zwischen 50 und 100 BPEL Prozessinstanzen zu Laufzeit.

Vom Mengengerüst startete das Projekt zunächst mit einem Pilot mit einer Produktgruppe mit ca. 2000 Aufträgen pro Tag, wurde danach auf ca. 15000 Aufträge erweitert und verarbeitet in der nächsten Stufe 60.000-70.000 Aufrägen pro Tag.

Die Infrastruktur der Soa Suite ist mit 8 BPEL Cluster Linux-Knoten mit je 4 Cores und 2 Solaris RAC Konten mit je 20 CPU ausgestattet.

Die Last- und Performancetestergebnisse stammen auf der Tuningphase bis Ende 2009/Anfang 2010.

In dieser Zeit wurden ca. 1 Million BPEL Prozessinstanzen pro Tag neu erzeugt.

Projekt 2: Telekommunikation – Backend Order Processing mit AIA 2.4 und Oracle BPEL 10gR3

Bei diesem Kunden im Mobilfunk und DSL-Bereich wurde die Oracle Application Integration Architecture 2.4 (AIA for Comms FP) eingesetzt, um Siebel CRM 8 mit den Backend-Applicationen zu integrieren. Dabei wurden 2 „Half-PIPs“ eingesetzt:

- Order-to-Bill
- Customer Data Hub

Die gesamte Infrastruktur ging im April 2010 in Produktivbetrieb und wurde seither sukzessive um weitere Endkunden, d.h, im wesentlichen die Partnershops des Mobilfunkunternehmens ausgebaut.

Der Nachrichtenfluss erfolgt dabei typischerweise wie folgt:

1. Client (z.B. Siebel)
2. AIA Requestor ABCS – implementiert in BPEL
3. AIA Enterprise Business Service, implementiert mit Oracle ESB
4. AIA Provider ABCS, implementiert in BPEL
5. Backend System

Beispiel-Usecase ist der “Feasibility Check”, bei dem ein Auftrag auf technische Machbarkeit – u.a. des Anschlusses und der Geschwindigkeit geprüft werden.

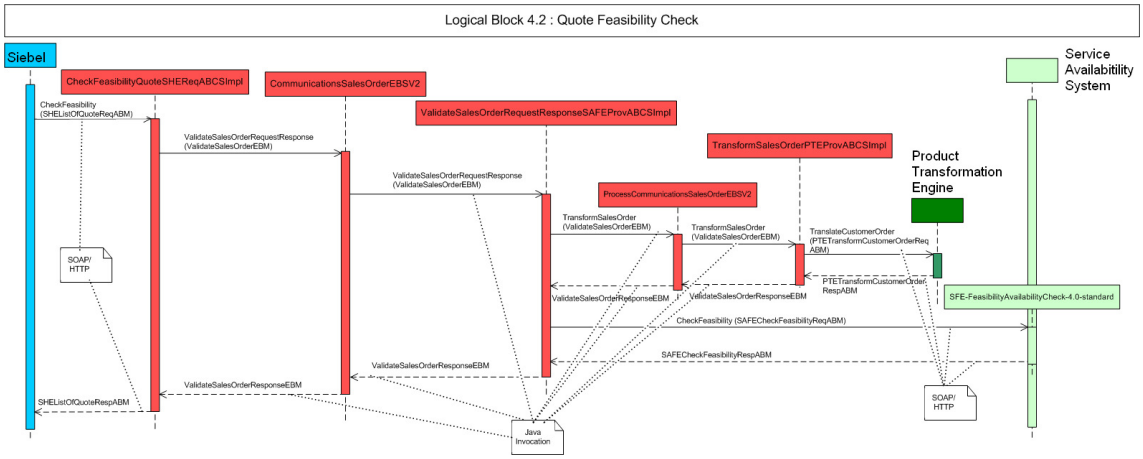


Abbildung 1: Beispiel Nachrichtenfluss Feasibility Check (rot: AIA)

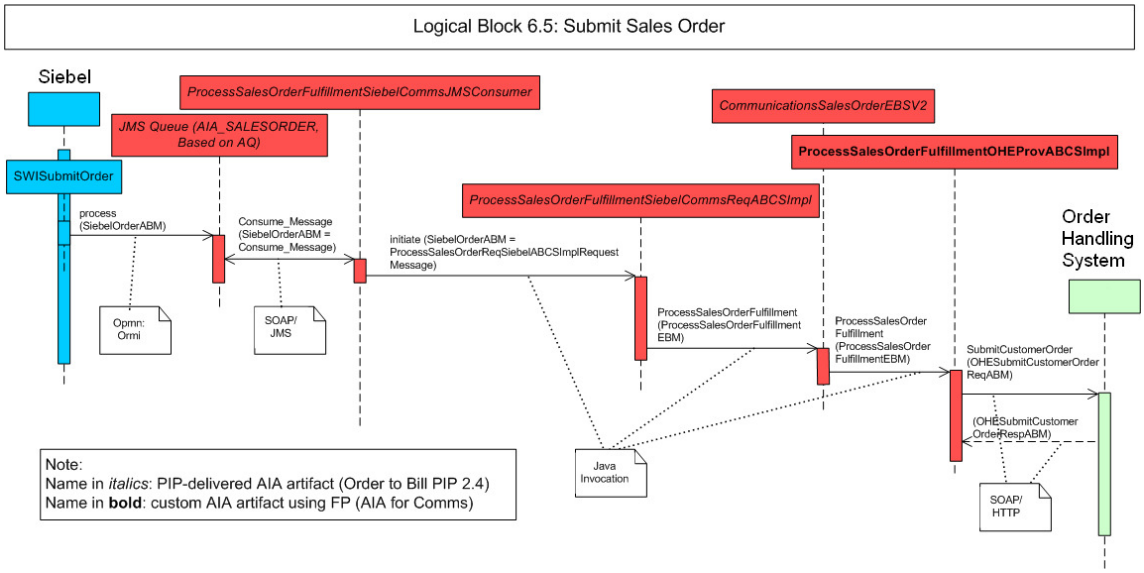


Abbildung 2: Beispiel Order Submit (rot: AIA)

Die Realisierung der AIA Prozesse erfolgt durch synchrone BPEL-Prozess mit einer Ausnahme, dem OrderSubmit Use Case, bei dem Siebel einer Order in einer JMS/AQ Queue an AIA übergibt. Nach Verarbeitung erfolgen dann ggf. Asynchrone Rückmeldungen über OrderUpdate Notifications an Siebel.

Übersicht der BPEL Tuningparameter

Das Tuning der BPEL Engine erfordert die Messung und Optimierung von sehr vielen unetrschiedlichen Parametern:

- JVM (u.a. Garbage Collection Strategie)
- Application Server (u.a. Transaction Timeout Settings)
- Data Sources (u.a. Minimum und Maximum Connections)
- Adapter (u,a, JCA Thread Pool Settings)
- JDBC (u.a. Statement Cache Size)
- BPEL Engine (Admin): u.a. soapServerURL, optSoapShotcut)
- BPEL Domain (u.a. AuditLevel, Statistiken, SyncMaxWaitTime, BPEL Thread Pools)
- BPEL Prozess (u.a. In-Memory Optimization)
- BPEL Partnerlink (u.a. skipDeepCopy, idempotent Setting)

Tuning synchroner BPEL Prozesse

Sychrone Prozessen liefern eine unmittelbare Antwort (Request-Reply) an den Aufrufer in einen zu definerenden Zeitinterval. Default für diese „syncMaxTime“ ist 45 Sekunden. In vielen Fällen ist dies – u.a.bei Verwendung von AIA zu kurz.

Der Hauptaufwand bei synchronen Prozessen unter Last ist

- Verwendung von Datanbankoperationen
- Verwendung komplexer XSLT Transformationen
- Das Mitschreiben von Log-Informationen

Daraus ergeben sich automatisch die Optimierungsmöglichkeiten.
Die BPEL-Engine kann sychrone Prozesse komplett in Memory verarbeiten, wodurch Datenbankoperationen vermieden werden. Zunächst ist jedoch aus Sicht der Requirements des Projekts zu entscheiden
Ob für fehlerfrei ausgeführte Prozesse ein AuditTrail / Flow in der BPEL Console verfügbar sein muss
Welcher Detailgrad von AuditInformationen benötigt werden

Konfigurations- und Optimierungsmöglichkeiten für synchrone Prozesse

inMemoryOptimization (default: false):

Bei nicht-persistenten BPEL-Prozesses kann dieses Property gesetzt werde, um jegliche Datenbankinteraktion zu vermeiden. Die BPEL-Instanz wird dann vollständig im Hauptspeicher gehalten. Dies funktioniert nur in Prozessen, die nicht „durable“ sind, also keine Aktivitäten wie z.B. wait, (mid-process) receive, onMessage oder onAlarm enthalten.

completionPersistPolicy (default: on):

Dieses Property kontrolliert ob bzw. wie die Meta-Daten einer BPEL-Instanz gespreichert werden:
on: die BPEL-Instanz wird normal in der Datenbank gespeichert
deferred: die BPEL-Instanz wird in einem seperaten Thread (asynchron) in der Datenbank gespeichert
faulted: nur fehlerhaft abgebrochene BPEL-Instanzen werden in der DB gespeichert
off: keine BPEL-Instanz wird in der DB gespeichert

Erfahrungen und Best Practises für synchrone Prozesse

Folgende Erfahrungswerte wurden bei Verwendung der In-Memory-Optimization erzielt:

	Medium BPEL Process with 3 subprocess rmi calls (no soap)
1	Average 60,26 ms without optimization
2	InMemoryOptimization=true: avg 24,46 ms
3	Additionally completionPersistLevel=instanceHeader: avg: 21,58 ms
4	Additionally completionPersistPolicy=faulted: 19,06 ms
5	Additionally StatsLastN=0 : avg 18,31 ms

In Summe kann die Ausführungszeit somit von ca. 60 ms auf 18 ms verringert werden.

Zu beachten ist, das bei Verwendung der inMemoryOptimization mit Versionen <10.1.3.5.1 Konflikte mit dem Fault-Policy-Framework auftreten können (siehe Bug 8825348 inMemoryOptimization=true and fault policy would work together)

Tuning asynchroner BPEL Prozesse

Erfahrungen und Best Practises für asynchrone Prozesse

Unter asynchronen Prozessen erfordern diejenigen besondere Maßnahmen, die nicht one-way, sondern auf einem Request-Callback Szenario beruhen.

Beim Tuning dieser asynchronen BPEL-Prozesse kommt es nach den Erfahrungen aus Projekt 1 insbesondere auf die Datenbank-Performance des BPEL Dehydrationstore an.

Dies liegt daran, dass insbesondere für die Zuordung von Callback-Nachrichten komplexe SQL-Queries nötig sind und Locks verwendet werden müssen, um zu vermeiden, dass parallele Threads gleichzeitigt den Status einer Callbacknachricht verändern.

Ein Beispiel einer verwendeten Query ist

```
SELECT  /*+ INDEX (ds1 ds_conversation, DS_FK) */
        ds1.conv_id,  ds1.conv_type, ds1.cikey,
        ds1.domain_ref, ds1.process_id, ds1.revision_tag,
        ds1.process_guid, ds1.operation_name, ds1.subscriber_id,
        ds1.service_name, ds1.subscription_date, ds1.state,
        ds1.properties
FROM dlv_subscription ds1, (
        SELECT DISTINCT  /*+ INDEX (ds2 ds_conversation, DS_FK) */
          ds2.cikey
        FROM  dlv_subscription ds2
        WHERE ds2.conv_id = :1  AND ds2.state = 0 AND
                 ds2.domain_ref= :2 ) unresolved_subscriptions
WHERE ds1.cikey = unresolved_subscriptions.cikey AND
      ds1.state = 0 AND NOT EXISTS  (
        SELECT  /*+ INDEX (ds3 ds_conversation, DS_FK)
                    INDEX (ds1 ds_conversation, DS_FK) */ 1
        FROM dlv_subscription ds3
        WHERE ds3.state = 1 AND ds3.cikey = ds1.cikey
                AND ds3.domain_ref = ds1.domain_ref )
                AND ds1.domain_ref = :3
        FOR UPDATE OF ds1.subscriber_id NOWAIT
```

Dieses Statement wird auf der DLC_SUBSCRIPTION Tabelle ausgeführt und ist sehr aufwändig.

Bei asynchonen Prozessen ist zudem der Effekt, dass bei nicht abgeschlossenen BPEL-Instanzen von der BPEL Engine angenommen wird, dass noch eine Callback-Nachricht eintreffen kann.
Somit werden diese Instanzen nicht vom default Purge Skript gelöscht. Brechen Subprozesse also z.B. aus Fehlergründen ab, so muss selbst dafür gesorgt werden, dass die zugehörigen BPEL-Instanzen auf Caller-Ebene ebenfalls beendet oder abgebrochen werden. Ansonsten bleiben diese Instanzen aktiv und warten „für immer“ weiter auf ihre Callback Nachricht. Dies führt dann zu stetig wachsenden Einträgen in DLV_SUBSCRIPTION und anderen Tabellen und damit potenziell zu einer konstant steigenden Verschlechterung der Datenbank-Performance.

Als zweiter Effekt wurde angesprochen, dass die Zuordnung von Callback-Nachrichten zu den auf Sie wartenden Prozessinstanten Datenbank-Locks benötigt. Dies führt bei sehr hoher Last zu einer Row Lock Contention auf der Datenbank. In Metalink sind in der Note 1071069.1 (ORA-02049 and Locks on DLV_SUBSCRIPTION Table After Applying Patch 8342907) Maßnahmen beschrieben, wie dies beseitigt bzw. vermieden werden kann.

(Voraussetzung für das performante Verabreitung in der 10.1.3.4 Version ist Verwendung von MLR > 8 und Installation von Patch 9041454: HIGH RESPONSE TIMES AND LOCKS ON DLV_SUBSCRIPTION)

Lasttestmethodik und Messverfahren

Bei der Messung der Perfomancemetriken ist zu beachten, dass die Ergebnisse aus unterschiedlichen Quellen stammen müssen, je nachdem ob In-Memory-Optimization verwendet wird oder nicht.

Wir die In-Memory-Optimization eingeschaltet und werden nur fehlerhafte sychrone Instanzen gespeichert, können keine Messwerte aus dem BPEL-Dehydrationstore per SQL gewonnen werden.

Messungen mit Auswertung der BPEL DB:

```
select CIKEY,
CREATION_DATE,MODIFY_DATE,MODIFY_DATE-CREATION_DATE,
EXTRACT(Day FROM(MODIFY_DATE-CREATION_DATE)) as Day,
EXTRACT(HOUR FROM(MODIFY_DATE-CREATION_DATE)) as Hour,
EXTRACT(Minute FROM(MODIFY_DATE-CREATION_DATE)) as Minute,
EXTRACT(SECOND FROM(MODIFY_DATE-CREATION_DATE)) as second
from orabpel.cube_instance where
process_id='CheckFeasibilityQuoteSHEReqABCSImpl' AND
CREATION_DATE > '03-SEP-10 01.00.00.000000000';
```

Messungen ohne BPEL DB : stehen aufgrund der In-Memory-Optimization keine Timestamps in der CubeInstance Tabelle zur Verfügung, belieben folgende Messmöglichkeiten:
Logging von Start- und Endezeiten eine Datei z.B. über Sensoren und anschließendes (asnychrones) Parsen der Datei zur Speicherung in eine DB
Publish der Daten in eine JMS Queue und asynchrones Verarbeiten zur Speicherung in eine DB
Verwendung eines TCP Proxies (TcpMon, SOAPUI etc.)
Verwendung von Messungen auf Client-Seite

Bei Projekt 2 wurden End-2-End Performancetests durchgeführt unter Verwendung eines Logging-Frameworks wie im Ansatz 1. Dabei ist generell zu berücksichtigen, dass die Ausführungszeiten vom Unmarshalling des Requests von SOAP bis zur Ausführung des Startsensors nicht mitgemessen werden. Dasselbe gilt für den Stopsensor bis zum Marshalling der Responsenachricht. Aus dem Vergleich von Client-seitigen Antwortzeiten und den Sensorwerten kann jedoch dieser Overhead zusammen mit dem Zeit für den reinen Netzwerktransport ermittelt werden.

Messung von End-2-End Performance ohne die Backendsysteme:

Beim Tuning der BPEL- oder AIA Infrastuktur wurden in beiden Projekten die Zeiten der Backendsysteme entweder herausgerechnet oder diese Systeme über Stubs simuliert. Ansonsten kann

es leicht passieren, dass durch ein nicht optimal skalierendes Backendsystem die eigentlichen Werte von Skalierbarkeit und Performance der BPEL-Engine nicht getroffen werden können.

Monitoring und Tuning der BPEL Thread Pools

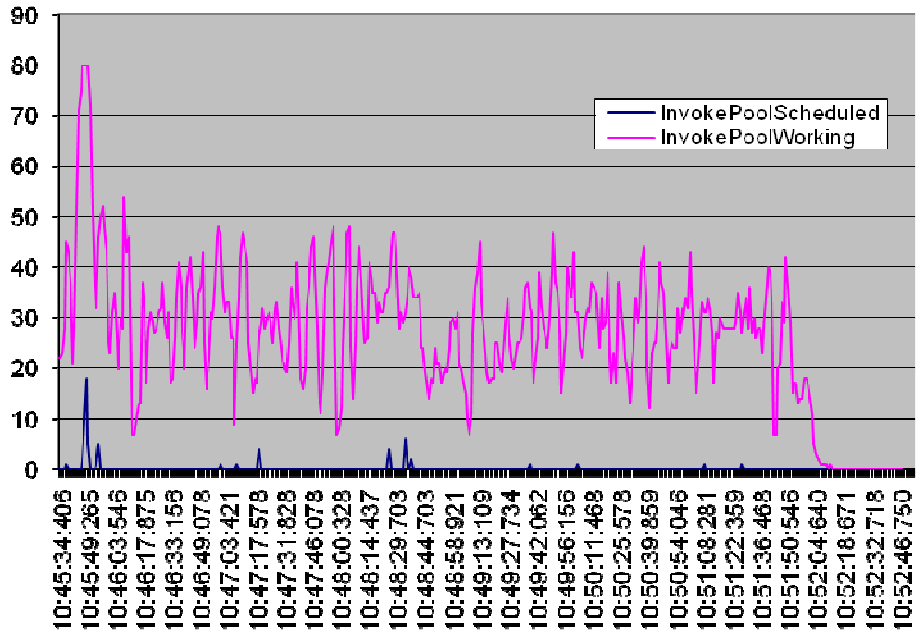
Kern der Tuningaktivitäten der BPEL Engine ist die Bestimmung der optimialen Werte für die Größe der BPEL Thread Pools.

Es gibt die 3 Pools: InvokePool, EnginePool und SystemPool sowie die zugehörgen Queues für die Nachrichten, die auf einen freien Thread auf einem dieser Pools warten.

Ein sehr wichtiges Merkmal beim Tuning ist zu bestimmen, ob die Anzahl bereitgestellter Threads für ein bestimmtes Lastprofil ausreicht. Dies kann man in der 10g über das BPEL API überwachen und sich periodisch die Anzahl der Nachrichten in den Queues ausgeben lassen. Stauen sich über eine Zeit von mehr als wenigen Sekunden viele Nachrichten in einer Queue und werden diese somit nicht zeitnah abgebaut, so verschlechtert dies die Performance erheblich und die Ausführungszeiten steigen drastisch an. Hier ist zu prüfen ob entweder

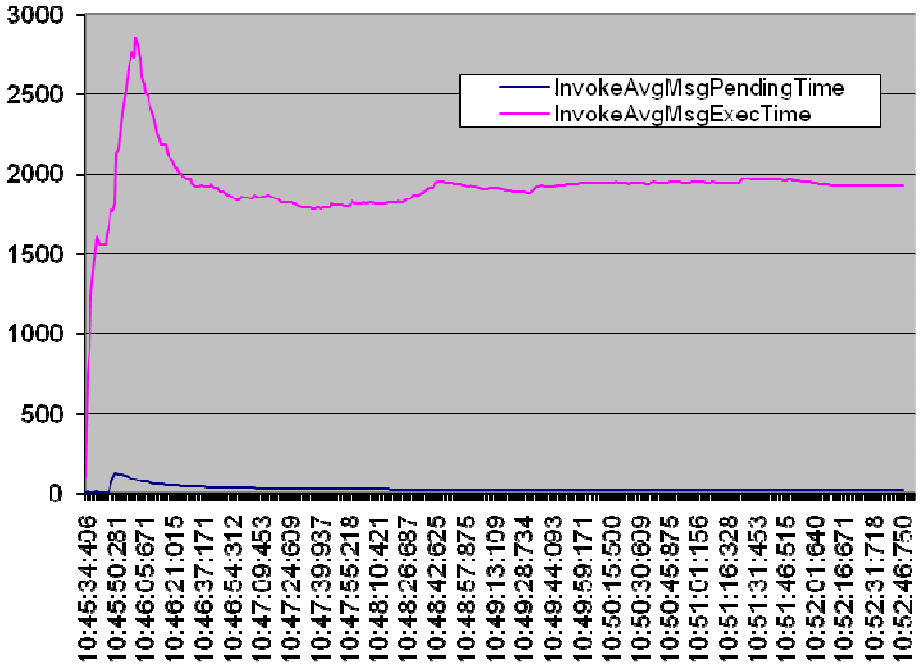
1. die Threads durch z.B. zu lange laufende SQL Statement blockiert werden oder
2. die Anzahl der Threads zu klein ist

Beispiel: ein gut getunter Invoke Pool sieht unter Last wie folgt aus:

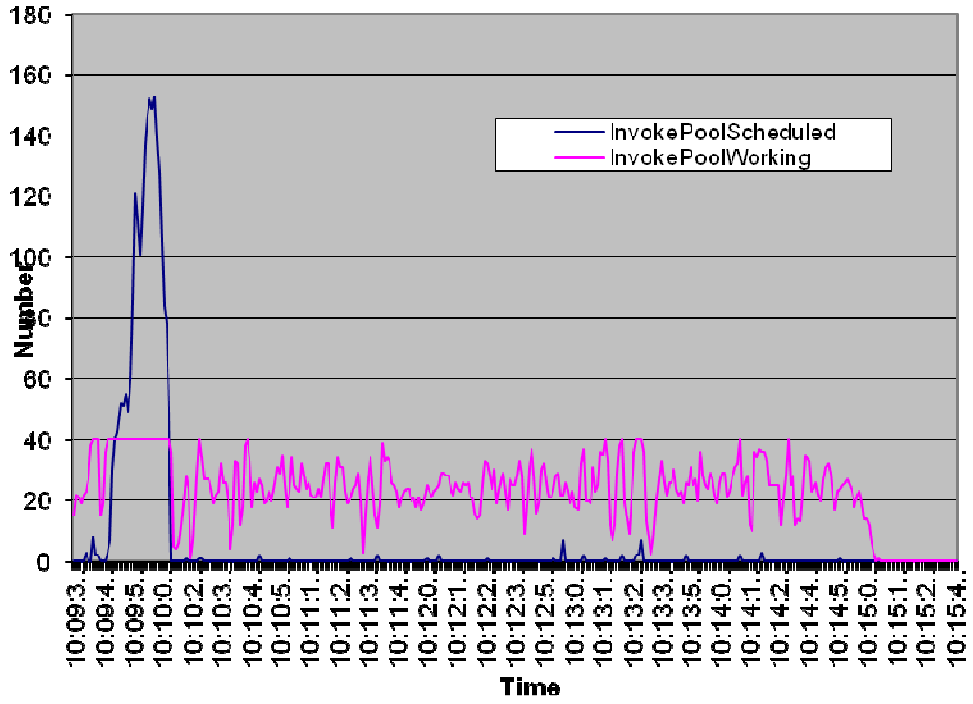


Hier werden alle Invoke-Messages durch einen freien Thread abgearbeitet. Die Invoke Queue (blau) enthält fast nie Nachrihcten.

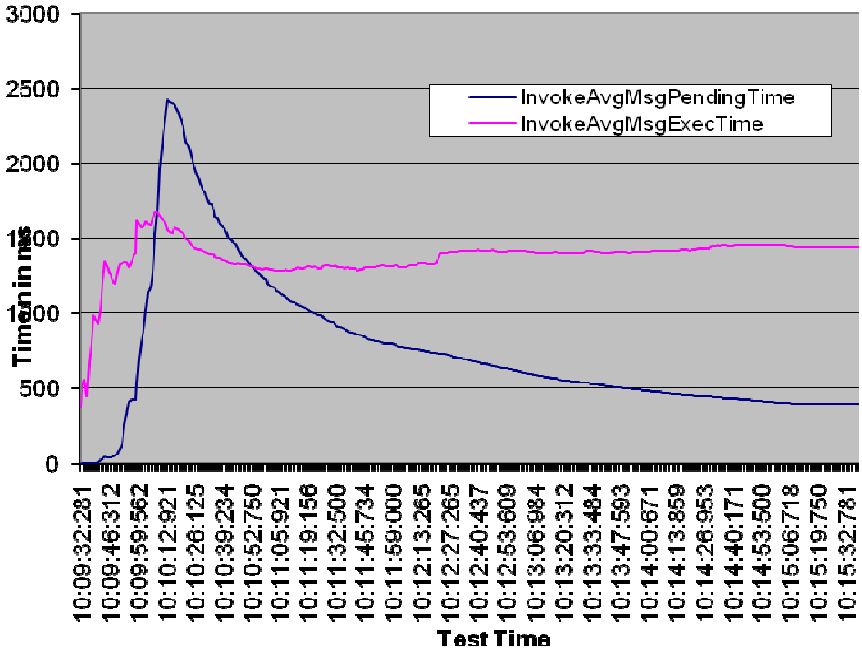
Resultat ist, dass die durchschnittliche Wartezeit auf einen freien Invoke Thread fast Null ist:



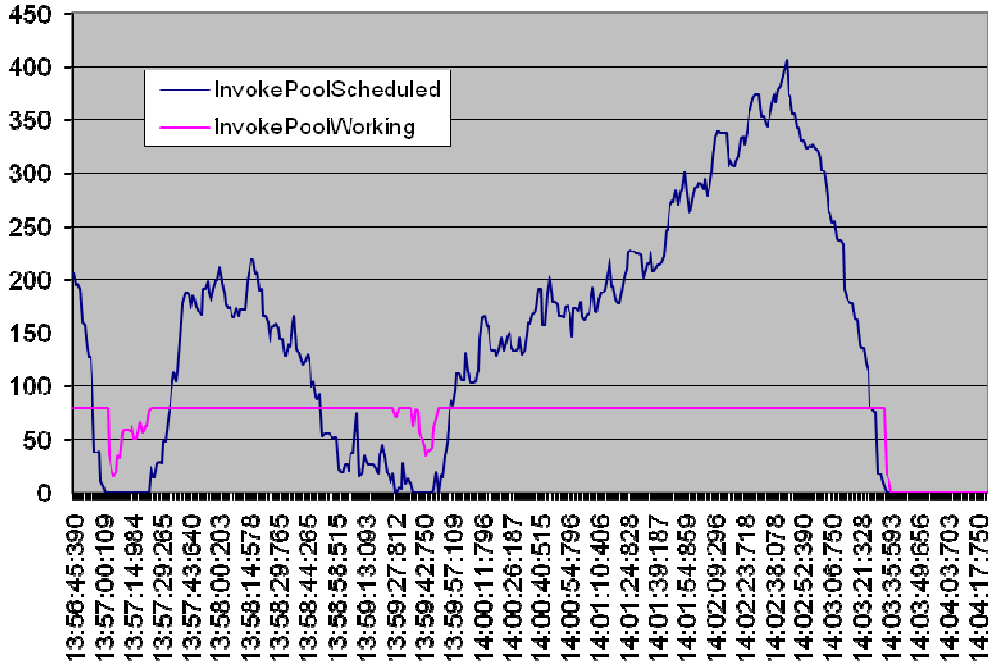
Wird der Invoke Pool dagegen zu klein dimensioniert, steigt die Anzahl der Nachrichten in der Invoke Queue schnell an:



Als Ergebnis steigt die durchschnittliche Wartezeit auf einen Thread start und baut sich nur langsam ab:



Unter sehr hoher Last kann dies dann leicht so aussehen:



DB Tuning

Durch das Erhöhen der initrans Parameter der Tabellen cube_scope und cube_instance konnten Row Lock contention erheblich verringert werden.

Bsp.:

- Tabelle CUBE_INSTANCE Parameter Initrans = 16 setzen
- Index CI_PK Parameter Initrans = 16 setzen
- Index STATE_IND Parameter Initrans = 16 setzen
- Index CI_CUSTOM2 Parameter Initrans = 16 setzen
- Index CI_CUSTOM3 Parameter Initrans = 16 setzen
- Index CI_CUSTOM4 Parameter Initrans = 16 setzen
- Tabelle CUBE_SCOPE Parameter Initrans = 16 setzen
- Index CS_PK Parameter Initrans = 16 setzen

Einfluss der Datenbankperformance auf die BPEL Gesamtperformance

Im Projekt 1 hat sich gezeigt, dass insbesondere bei Verwendung asynchroner BPEL-Prozesse die Datenbankperformance sehr kritisch für die Gesamtperformance ist.

Dabei sollte insbesonder auf folgende Messwerte geachtet werden:

- Anzahl verwendeter Datenbackconnections im Pool BPEL_PM

- Laufzeit der Top-SQL Statements für Insert/Update in CUBE_INSTANCE, DLV_MESSAGE und DLV_SUBSCRIPTION (der BPEL-Runtime)
- Laufzeit der Top-SQL Statements für Select auf CUBE_INSTANCE etc. durch die BPEL-Console

Beim Tuning der DB bei langlaufenden Prozessen sind folgende Aktivitäten sehr wichtig:

- Regelmäßiges Purging von abgeschlossenen Instanzen
- Tuning des SQL Purge Scripts
- Manuelles oder Automatisches Recovery von Invoke- und Callback Nachrichten
- Verwendung von Partitioning

Vergleich Performancetuning mit SOA Suite 11g

Die für BPEL relevanten Performancetuningparameter sind in der SOA 11g Version gleich geblieben. Insbesondere hat sich das Threading-Modell der BPEL-Engine nicht geändert.

In BPEL 11g ist der Parameter `completionPersistLevel` deprecated. Weiterhin in SOA 11g exstistieren inMemoryOptimization und completionPersistPolicy

Neu ist der Parameter oneWayDeliveryPolicy :

Dieses Property bestimmt das Verhalten der BPEL-Engine im Delivery Layer bei One-Way Calls:
`async.persist`: Nachrichten warden asnychron gespeichern in der DB, bevor sie von der Engine verarbeitet werden.
`async.cache`: Nachrichten w
erden nur in-Memory gespeichert , bevor sie von der Engine verarbeitet werden.
`sync`: Die Nachricht, die die BPEL-Instanz one-way startet wird nicht zwischengespeichert. Die Instanz wird im selben Thread gestartet.

Clustering in BPEL 10gR3

Die BPEL Versionen bis 10.1.3.4 nutzen JGroups zur Synchronisation im Cluster.

Dabie gibt es 2 generelle Konfigurationsvarianten in jgroups-protocol.xml:
- Verwendung von Multicast (UDP)
- Verwendung von TCP

Die Nutzung von Using multicasting can be an issue in the network. Multicast is based on UDP and is often blocked in the network. I encountered this issue at a few customers. The solution is rather simple, instead of using UDP/multicast we use TCP to point to the nodes in the cluster.

Beispiel aus Projekt 1:

```
<config>
  <TCP start_port="30200" loopback="true" send_buf_size="32000"
recv_buf_size="64000"/>
  <TCPPING timeout="3000" initial_hosts="10.151.129.12[30200],10.151.129.13[30200]"
port_range="2" num_initial_members="2"/>
  <FD timeout="10000" max_tries="5" shun="true" down_thread="false"
```

```
up_thread="false"/>
  <FD_SOCK down_thread="false" up_thread="false"/>
  <VERIFY_SUSPECT timeout="1500" down_thread="false" up_thread="false"/>
  <pbcast.NAKACK gc_lag="100" retransmit_timeout="600,1200,2400,4800"/>
  <pbcast.STABLE stability_delay="1000" desired_avg_gossip="20000"
down_thread="false" max_bytes="0" up_thread="false"/>
  <VIEW_SYNC avg_send_interval="60000" down_thread="false" up_thread="false" />
  <pbcast.GMS print_local_addr="true" join_timeout="5000" join_retry_timeout="2000"
shun="true"/>
</config>
```

Ab Version 10.1.3.5.1 wird kein JGroups mehr genutzt – stattdessen findet die Synchronisation über die BPEL DB statt.

Ergebnisse und Status der Projekte

Projekt 1 (BPEL)

Die Herausforderung bei Projekt 1 waren das Erreichen des geforderten Durchsatzes bei einem sehr großen Datenbestand in der BPEL DB durch die Vielzahl an langlaufenden Prozessinstanzen.

Die Perfromance konnten durch die Optimierungen so gesteigert werden, dass mehr als 1 Mio BPEL Instanzen pro Tag verarbeitet werden. Insbesondere die Optimierungen auf Datenbankebene und beim Purge-Skript haben dies ermöglicht. Als nächste Schritte sind die Nutzung von DB Partitioning sowie die Migration auf 10.1.3.5.1 geplant.

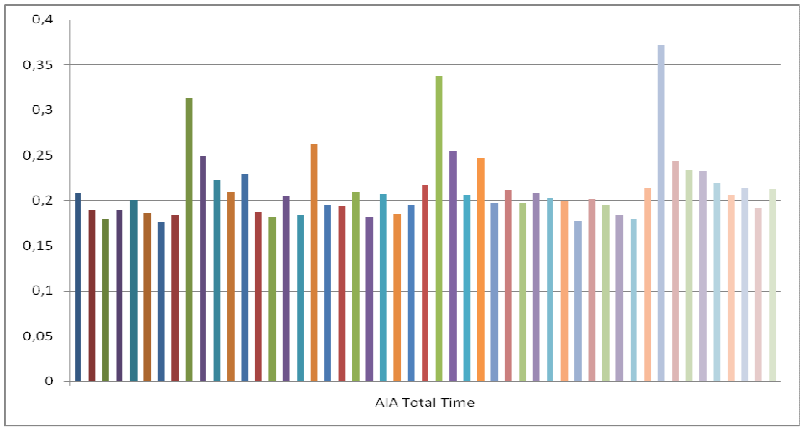
Projekt 2 (AIA+BPEL)

Die Herausforderung in Projekt 2 war nicht das Erreichen der hohen Durchsatzzahlen wie in Projekt 1, sondern der Minimierung der Ausführungszeiten innerhalb des AIA/BPEL Layers.

Alle synchronen Prozesse wurden mit der In-Memory-Optimization umgesetzt (bis auf den Use Case „Order Submit“).

Der komplexeste Use Case ist Check Feasibility, da dort der Nachrichtenfluss synchron über die Layer – Siebel → BPEL→ ESB → BPEL → ESB → BPEL → Zielsystem erfolgt.

Ergebnisse CheckFeasibility (auditLevel minimal, statsLastN =1000, ESB instacke Tracking: off)



Trotz dieser erhöhten Komplexität wurde durch die Tuningmaßnahme erreich dass die Ausführungszeit in AIA durchschnittlich nur 0,21328 s bei der zurgrundegelegten Ziellast des Systems beträgt. Bei der Gesamtausführungszeit inklusive Siebel und Backendsystemen von teilweise mehr als 30 Sekunden ist dies weniger als 1% der Gesamtzeit.

Allgemeine Lessons Learned

Bei Planung einer BPEL- basierten Architektur ist in den meisten Fällen eine Komponente zu entwerfen, die Informationen über den Status von Geschäftprozessen (im Fallen unserer beiden Kunden der „Aufträge“) liefert.

Beispielsweise soll in Siebel CRM für einen Callcenter Mitarbeiter der Status der Aufträgsdurchführung und der Plantermin sichtbar sein.

Sinnvoll ist hier, diese Abfragen nicht direkt auf den BPEL-Dehydrationstore durchzuführen, sondern diese Informationen in einer getrennten Datenbank zu speichern.

Als Varianten kommen folgende Realisierungen in Betracht:

1. asynchrones „Push“ der Status z.B. über JMS/AQ in eine DB
2. synchrone Speicherung in eine Datei und anschließendes Parse und Abspeichern in eine DB

Im Projekt 1 wurde dies anfänglich durch Sensoren mit Ansatz 1. Implementiert. Später wurde dies durch synchrone Invoke Aktivitäten ersetzt, da als fachliches Requirement keine Zeitverzögerung bei den Status-Updates toleriert werden konnte.

Das Projekt 2 nutzt Ansatz 2.

Referenzen

- Oracle® Fusion Middleware Developer's Guide for Oracle SOA Suite 11*g* Release 1 (11.1.1)
- Oracle® Application Server Performance Guide 10g Release 3 (10.1.3.1.0)
- Oracle ® SOA Suite 10.1.3.3 Best Practises Guide

Kontaktadresse:

Stefan Koser
Oracle Deutschland GmbH
Schloßstr. 2
D-13507 Berlin-Tegel

Telefon:	+49 (0) 435795-165
Fax:	+49 (0) 435795-419
E-Mail	stefan.koser@oracle.com
Internet:	www.oracle.de
Blog:	stefankoser.blogspot.com