Komfortables Job-Scheduling in einer hochverfügbaren RAC-Umgebung

Peter Hawelka, pdv Technische Automation + Systeme GmbH

Trotz umfangreicher online-Verarbeitung innerhalb von Benutzerdialogen bleibt die Job-Verarbeitung in datenintensiven Applikationen eine wichtige Komponente auch in modernen komplexen IT-Systemen. Die Job-Ablaufsteuerung ist dabei das Instrument zum kontrollierten Ablauf der Hintergrundprozesse - einzeln oder logisch verkettet. Eine besondere Bedeutung kommt dabei der Fehlerbehandlung und dem Wiederaufsetzen von Jobs zu. Die abzubildenden Arbeitsabläufe ergeben sich hier aus den Geschäftsprozessen, die zum Teil aus speziellen Anforderungen an die Job-Scheduler-Funktionen resultieren.

Die Integration des Job-Scheduling in eine hochverfügbare IT-Infrastruktur auf Basis des Oracle Real Application Clusters (RAC) stellt dabei eine besondere Herausforderung dar. Mittels Job-Klassen kann das Job-Scheduling zu einer effektiven Ressourcenauslastung beitragen. Zur Kontrolle des weitgehend automatisch- und ereignisgesteuerten Ablaufs von mehreren Hundert Jobs täglich wird ein leistungsfähiger Job-Browser benötigt, der sich vorzugsweise als Web-Anwendung in eine moderne IT-Infrastruktur einpasst. Hauptmerkmal des Job-Browsers ist die Darstellung von hierarchischen Job-Ablaufketten. Bearbeitungsfunktionen wie „drill-down“ ermöglichen dabei eine komfortable Navigation innerhalb der Ablaufketten. Des Weiteren müssen Störungen im Ablauf mit dem Job-Browser bearbeitet werden können. Wichtige Werkzeuge hierfür sind die Darstellung von Ablaufprotokollen, die Filterung von Jobs mittels vordefinierter und dynamischer Job-Filter sowie das manuelle Wiederaufsetzen von gesamten Abläufen oder Teilablaufketten.

Einfache administrativen Steuerung mittels Job Queues

Job Queues regeln den Ablauf einer Menge von Jobs. So kann beispielsweise die Anzahl gleichzeitig laufender Jobs einer Job Queue einfach administrativ gesteuert und kontrolliert werden. In Verbindung mit den Job-Klassen lässt sich die Auslastung eines RAC-Systems zusätzlich optimieren. Job Queues ermöglichen dem Administrator darüber hinaus, die Kontrolle über das Geschehen in der Datenbank zu behalten. Beispielsweise lassen sich Wartungsfenster auch kurzfristig leicht umsetzen, ohne dass der Betriebsablauf wesentlich gestört wird.

Mit der Datenbank 10*g* wird eine weitgehend überarbeitete API für die in die Datenbank eingebauten Scheduling-Funktionen seitens Oracle bereitgestellt. Diese API, das „dbms_scheduler“-Package, ersetzt das alte „dbms_jobs“-Package und kann aufgrund seiner vielen Erweiterungen ein stabiles Fundament für den Aufbau eines Job-Schedulers in der Oracle-Datenbank bieten. Allerdings ist die API aufgrund ihrer Verallgemeinerung und Vielfältigkeit nicht ganz einfach zu benutzen. Eine Implementierung kann daher – je nach Anforderung an den Job-Scheduler – relativ aufwendig sein. Zudem fehlen Funktionen, die eine einfache Hierarchisierung der Jobabläufe ermöglichen. Die Umsetzung von Job-Queues, kann nicht mit der zur Verfügung stehenden API alleine erreicht werden.

Man kann sich die grundlegende Struktur des Oracle-„dbms_scheduler“-Package vereinfacht so vorstellen: Aufgaben werden in Form von Programmen als Datenbank-Objekt gespeichert und dem Programm als „Schedule“ zugeordnet, der ebenfalls in der Datenbank gespeichert wird. Ein „Schedule“ enthält die Information, wann und wie ein Job die im Programm definierten Aufgaben ausführen soll. Dieser Job kann auch als Instanz eines Programms angesehen werden. Für jedes Programm lassen sich Argumente definieren, über die dem Job Informationen zur Laufzeit mitgegeben werden. Ein weiteres Attribut eines Jobs ist die Job-Klasse. Mit deren Hilfe lassen sich Job-Gruppen bilden und gemeinsamen Ressourcen zuordnen. In einer hochverfügbaren RAC-Umgebung lässt sich so eine effektive Lastverteilung der RAC-Knoten realisieren.

Eine Aufgabe wird durch einen Job entsprechend den Ablaufbedingungen ausgeführt. Wie diese Aufgabe formuliert werden muss, wird durch den Parameter „program_type“ im Befehl „create_program“ definiert. Mögliche Programm-Typen sind beispielsweise „STORED_PROCEDURE“ oder „PLSQL_BLOCK“. Ein einfacher „program_type“, ist die „STORED_PROCEDURE“. Diese kann beliebigen PLSQL-Code enthalten, der die auszuführende Aufgabe hinreichend beschreibt. Ein Job-Scheduling-System lässt sich damit sehr einfach durch Prozeduren aufbauen, die die Referenzen auf die auszuführenden externen Programme oder Programmketten enthalten, die dann selbst wieder Jobs erzeugen können. Hierdurch errreicht man eine große Flexibilität der Job-Abläufe. Oracle verwaltet die Jobs mit ihren Laufzeit-Informationen in zentralen System-Tabellen und stellt mit den „ALL_SCHEDULER“-Views die Sichten auf die Job-Informationen zur Verfügung. Diese Views sind teilweise recht komplex und bedürfen einer exakten Analyse, um die gewünschten Job-Laufzeit-Informationen zu erhalten. In der Praxis zeigt sich jedoch, dass für den Aufbau eines Job-Scheduling wesentliche Mechanismen fehlen. Ins-

besondere lassen sich hierarchische Beziehungen von Jobs (dynamische Aufrufgeschichte), wie sie zur Laufzeit ausgeführt wurde, nicht einfach abbilden. Eine mögliche Realisierung kann durch die Speicherung von Zusatzinformationen in einer erweiterten Job-Tabelle erfolgen. Diese Daten werden dann durch den eindeutigen Job-Namen referenziert. Es ergibt sich daraus ein recht aufwendiges Datenschema, das durch eine übergeordnete View gekapselt werden kann und ein einfacher Zugriff auf die Job-Informationen zur Verfügung steht. Aufgrund der komplexen Darstellung der Job-Zustands-Informationen innerhalb der Oracle-System-Views muss jedoch ein besonderes Augenmerk auf die Performance dieser übergeordneten View gelegt werden. Insbesondere der Zugriff auf die Job-Statusinformationen erweist sich als laufzeitkritisch und bedarf besonderer Maßnahmen.

Es ergibt sich eine Software-Struktur, die einerseits das Oracle „dbms_Scheduler"-Package mit den Job-Daten und dem Streams AQ als Basis-Funktionalität nutzt. Andererseits werden die notwendigen Erweiterungen an Metadaten und Funktionen durch eine übergeordnete Job-View und einer eigenen Job-Scheduler-API zur Verfügung gestellt.

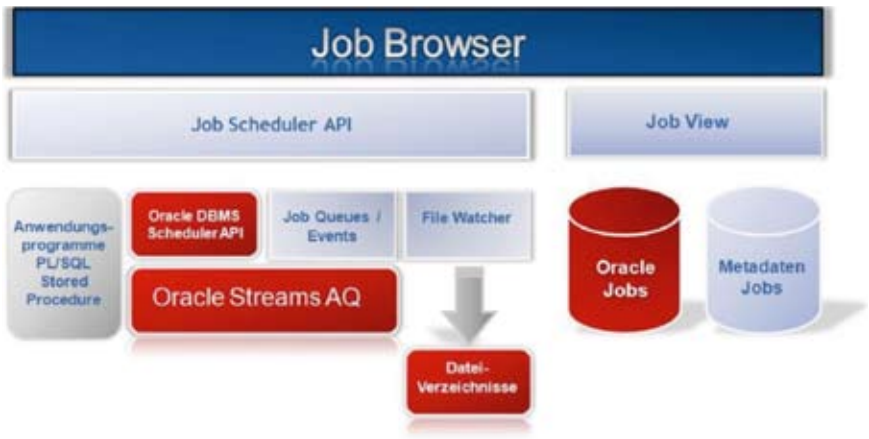

Abbildung 1: Schematische Darstellung der Software-Schichten

Job Queues mit Streams Advanced Queuing

Oracle Streams Advanced Queuing (AQ) ist ein in die Datenbank eingebetteter asynchroner Mechanismus zur Nachrichtenübermittlung, mit dem über die Datenbank Informationen von einem Producer gesendet und von einem oder mehreren Consumern gelesen werden. Für die Nachricht steht eine frei definierbare Nachrichtenstruktur zur Verfügung. Dieser Mechanismus wird zur Interprozess-Kommunikation vom Oracle Scheduler genutzt. Es liegt daher nahe, diesen Mechanismus auch für die Implementierung der Job-Queues zu nutzen. In unserem Beispiel geschieht dieses im Prinzip in folgender Weise: Basis aller Job-Queues ist eine Queue-Table, die durch folgenden Aufruf erzeugt wird:

```
Dbms_aqadm.create_queue_
table(queue_table_
name,payload,..)
```

Dabei definiert „payload" den Datentyp der Nachricht. Es kann ein DB-interner Typ, aber auch ein anwendungsspezifischer Typ sein. Anschließend wird eine Message-Queue definiert und angegeben, wer Nachrichten daraus konsumieren darf.

```
Dbms_aqadm.create_queue(queue_
name,queue_table_name,..)
Dbms_aqadm.add_
subscriber(queue_name,consumer)
```

Um die Anzahl der Message-Queues gering zu halten, sollte nicht für jede Job-Queue eine eigene Queue eingerichtet werden, sondern es wird eine Queue verwendet, in der die Information über alle Job-Queues gehalten werden. Die Zugehörigkeit zur Job-Queue ist dann in der Nachricht selbst abgelegt. Eine Nachricht in der Message-Queue repräsentiert ein „Runtoken" für eine Job-Queue. Die Anzahl der Runtokens einer Job-Queue definiert, wie viele Jobs gleichzeitig ausgeführt werden können.

Ein Job fordert nach dem Start zunächst ein Runtoken der jeweiligen Job-Queue an. Ist eines vorhanden, wird es konsumiert und am Job-Ende wieder eingestellt. Ist keines vorhanden, wartet der Job, bis ein Runtoken von einem anderen Job zurückgestellt wird, bevor seine Ausführung beginnen kann.

Das Anfordern des Tokens erfolgt über ein PLSQL-Codestück, das im Wesentlichen ein „Dbms_aq.dequeue (WAIT)" vor der eigentlichen Verarbeitungsroutine beinhaltet. Aber wie stellt man sicher, dass der Job nach seinem Ende, also auch im Fehlerfall oder Abbruch durch Runterfahren der Datenbank, sein Token zurückgibt?

Abbildung 2: Schematische Darstellung einer Job-Queue

Es kann ausgenutzt werden, dass der Oracle Scheduler die Job-Events wie „JOB_SUCCEEDED“, „JOB_FAILED“ etc. auch über eine Message-Queue signalisiert. Über eine Callback-Prozedur auf dieser System-Event-Queue lässt sich in jedem Fall das Job-Ende erkennen und das vorher konsumierte „Runtoken“ wieder einstellen.

```
Dbms_scheduler.add_event_queue_
subscriber(..)
Dbms_aq.register( sys.schedu-
ler$_event_queue,
                 ,plsql://pdv.
notify?pr=1‘,..)
```

Was passiert aber, wenn sehr viele Jobs in einer Job-Queue gleichzeitig gestartet werden, die Anzahl gleichzeitig laufender Jobs aber begrenzt ist? In einem solchen Fall werden erst mal alle Jobs gestartet, das heißt, auch jeweils einen Prozess in der Datenbank erzeugt, und auf ein Runtoken ihrer Job-Queue gewartet. Das Ergebnis wäre ein mit wartenden Jobs ausgelasteter Server. Das könnte dazu führen, dass in der Oracle-Datenbank selbst die eigenen System-Prozesse nicht mehr erzeugt werden können und es damit zu einem Datenbank-Stillstand kommt. Dieser Zustand ist in jedem Fall zu verhindern.

Um die Anzahl der wartenden Jobs und damit auch der laufenden Prozesse zu minimieren, muss sichergestellt sein, dass ein startender Job, der ein Runtoken anfordern will, auch eines erhält. Jobs werden vom Oracle-Scheduler nur gestartet, wenn sie auch aktiviert sind. Das bedeutet, wenn man den Job-Zustand „Enabled/Disabled“ in Verbindung zur Anforderung eines Runtokens setzt, kann erreicht werden, dass nur die in einer Job-Queue zulässige Anzahl Jobs auch tatsächlich aktiv sind und einen Prozess erzeugen können.

Eine weitere Herausforderung besteht in der Realisierung eines zuverlässigen „File-Watchers“ zur Überwachung der externen Verzeichnisse. Die Anforderung besteht darin, übertragene Dateien auch zu erfassen, deren Zeitstempel weit in der Vergangenheit liegen. Hier bietet sich eine Lösung mittels Advanced Queuing an. Dabei wird das Auftreten eines Events über die Message-Queues signalisiert. Über eine Callback-Prozedur der Message-Queue wird dann ein Verarbeitungsjob gestartet, der seinerseits das Programm zur Verarbeitung ausführt. Die Informationen aus dem File-Event können in Form von Parametern an das Verarbeitungsprogramm übergeben werden.

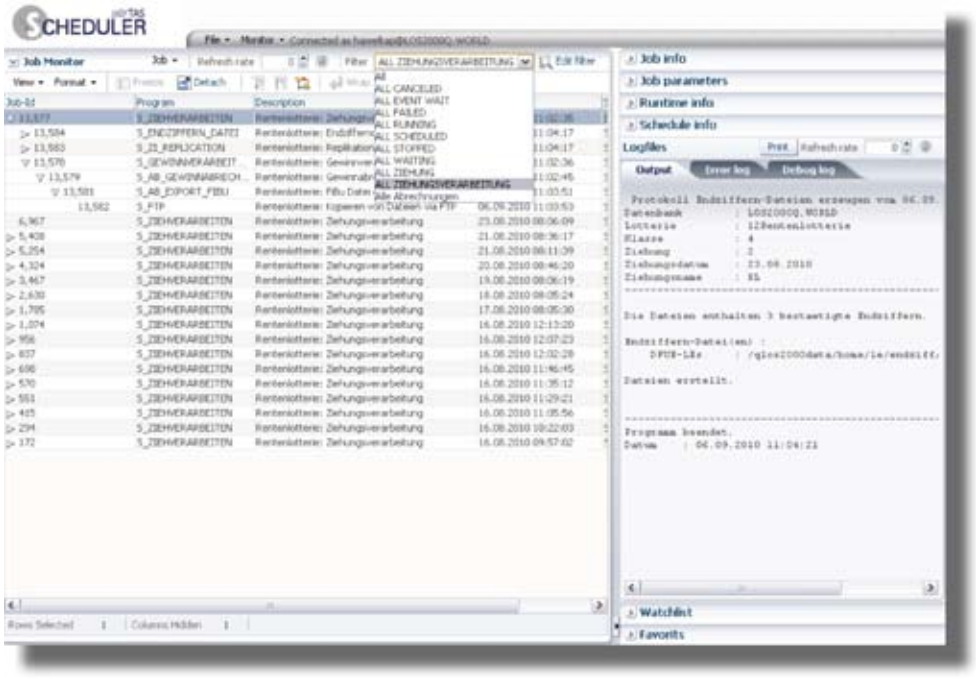

Abbildung 3: Job-Monitor der geplanten und ausgeführten Jobs in einer hierarchischen Darstellung

Die Verarbeitung von Ereignissen der AQ bietet eine Vielzahl von Möglichkeiten. Es lässt sich hiermit auch die Synchronisierung von komplexen Job-Ablauffolgen umsetzen.

In einem einfachen Beispiel wird durch einen laufenden Job ein weiterer Job gestartet. Der aufrufende Job soll auf die Ausführung des gestarteten Jobs warten und beispielsweise das Ergebnis auswerten, bevor er seine Ausführung fortsetzt. Damit er auf das Job-Ende reagieren kann, gibt der aufrufende Job dem aufgerufenen Job einen Synchronisations-Event-Name mit. Dieser aufgerufene Job erzeugt dann am Ende seiner Ausführung eine entsprechende Synchronisations-Nachricht, die dem aufrufenden Job sein Ausführungsende signalisiert.

Wird der Job-Scheduler zur Prozesssteuerung in einem IV-System eingesetzt, müssen Sichten auf die geplanten und bereits ausgeführten Jobs dem Benutzer zur Verfügung gestellt werden. Als primäre Entwicklungsplattform hat sich der Autor für das Oracle Application Developer Framework (ADF) 11*g* entschieden (Rich Faces + Business Components), mit dem die Anforderungen, insbesondere an die hierarchische Darstellung der Jobs, am produktivsten umsetzbar sind. Dabei wird von den zahlreichen Built-in-Funktionalitäten des ADF profitiert. Eine der wesentlichen Komponenten ist der sogenannte „Tree Table“, mit dem sich die Jobs hierarchisch dargestellen lassen. Diese Sicht stellt zumeist den Mittelpunkt der Benutzer-Interaktionen dar.

Als Schnittstelle zur Datenbank fungiert ein View-Objekt, das auf der übergeordneten Scheduler-View basiert. Ein weiteres Merkmal für eine effiziente Job-Überwachung und Bearbeitung stellen Filterfunktionen auf die Schnittstellen-View dar. Die Filter werden als einfacher String in einer Datenbank-Tabelle abgelegt und können über einen Funktionsaufruf als Where-Condition vom Client-Modul direkt angewendet werden. Hierdurch lässt sich eine weitgehende Unabhängigkeit des Job-Client von komplexen Datenbank-Funktionen erreichen.

ADF bietet samt der Datenbankanbindung eine weitgehende Unterstüt-

zung und moderne Umsetzung. So ist es möglich, ein realistisches Layout-Modell zu erstellen, anhand dessen die Einzelheiten mit den Anwendern leicht abgestimmt werden können. Hierdurch erreicht man von Anfang an eine hohe Akzeptanz. Des Weiteren lässt sich die „User Customization“ von ADF nutzen, mit der es ohne großen Programmieraufwand möglich ist, dass der Endanwender die Oberfläche nach seinen Bedürfnissen – etwa welche Spalten angezeigt werden sollen – konfigurieren kann. Durch Einschalten des MDS-Repository wird diese dann automatisch für jeden Benutzer gespeichert.

Der Einsatz von „Regions“ und „Task-Flows“ erlaubt die Umsetzung eines Single-Window-Konzepts sowie modifizierbarer Wizards für das Einplanen, Ändern und Wiederaufsetzen von Jobs. Die PL/SQL-API wurde über den im JDeveloper enthaltenden JPublisher in Java-Klassen gewandelt und in das Application-Model importiert. Die Besonderheit liegt darin, dass vor dem Aufruf der API-Funktionen eine Proxy-Connection an die Datenbank durchgeführt wird, um die PL/SQL API unter individueller Benutzerkennung ausführen zu können. Dieses Vorgehen ist insbesondere beim Starten und Wiederaufsetzen von Jobs notwendig, damit sie unter der eigenen oder sogar unter der Benutzerkennung eines anderen Benutzers ablaufen kann. Das Berechtigungskonzept der Anwendung wird hierdurch in keinem Fall durch die Jobs aufgebrochen.

Neben vielen Vorteilen zeigt ADF wie jedes andere Framework auch seine Schwächen. So verlässt es sich bei der Authentifikation auf den J2EE-Container, den Oracle Weblogic-Server. Dieser bietet leider keine Möglichkeit, eine Authentifikation gegen die Datenbank-User durchzuführen. So muss man hier einen eigenen Login-Provider schreiben, was schlecht dokumentiert und bei Weitem komplizierter ist, als der einfache JAAS Provider, der im OC4J noch ausreichte.

Fazit

Der Oracle-Scheduler und die in der Datenbank integrierten Streams-Advanced-Queuing-Methoden eigenen sich gut für den Aufbau eines Job-Scheduling-Systems, da es auch die Möglichkeiten der Lastverteilungen in einer RAC-Umgebung nutzen kann. Mittels individueller Erweiterungen lassen sich Job-Queues und ein effektiver File-Watcher implementieren. Die Bereitstellung von Programmierschnittstellen und Daten-Views kapselt viele Komplexitäten, was die Anwendungsentwicklung wesentlich vereinfacht und erleichtert.

Mit der Erstellung eines individuellen ADF-Job-Browsers wird dem Anwender ein performantes Arbeitswerkzeug bereitgestellt, mit dem ein schnelles und übersichtliches Arbeiten ermöglicht wird. Die Nutzung der modernen Komponenten im ADF erlauben eine hohe Benutzer-Akzeptanz. Darüber hinaus bietet ein sauberes Schnittstellen-Konzept auch eine Erstellung des Jobs-Browsers auf einer anderen technologischen Basis, wie etwa Apex oder .Net.

Kontakt:
Dr. Peter Hawelka
hawelka@pdv-tas.de

PL/SQL Challenge: Die DOAG ist mittlerweile auf Platz 2

Seit fast einem Jahr führt der PL/SQL-Guru Steven Feuerstein, hierzulande bestens bekannt als Keynote-Speaker der DOAG 2010 Konferenz + Ausstellung und Dozent des erfolgreichen Berliner Expertenseminars Best of Oracle PL/SQL, einen Wettbewerb für PL/SQL-Programmierer durch. Als Preise winken Bargeld, Bücher sowie auch „unbezahlbare Gewinne“ wie Consulting durch Steven Feuerstein persönlich. Inzwischen hat es die DOAG auf Rang 2 der Weltrangliste gebracht!

1. OTN – Oracle Technology Network
2. DOAG – Deutsche ORACLE-Anwendergruppe e.V.
3. ODTUG – Oracle Developer Tools User Group
4. OGH – Oracle Gebruikersclub Holland
5. IOUG – International Oracle User Group
6. UKOUG – United Kingdom Oracle Users Group
7. AIOUG – All India Oracle User Group
8. AUSOUG – Australian Oracle User Group
9. OBUG – Oracle Benelux User Group
10. OAUG – Oracle Applications Users Group

Infos unter: http://plsqlchallenge.com/