Oracle Scheduler „Know it or leave it“

Torsten von Osten
inconso AG
Hamburg

Schlüsselworte
Oracle Scheduler, Job Chains, Oracle Advance Queuing (AQ), Scheduler Remote Agent

Einleitung
In diesem Vortrag werden Tipps, Tricks und praktischen Erfahrungen zum Einsatz des Oracle Schedulers vorgestellt. Neben den grundlegenden Scheduling-Funktionen werden insbesondere die erweiterten Konzepte, wie z.B. Job-Chains, Event-Jobs, Callback-Funktionen und der Oracle Scheduler Agent für die Remote-Ausführung von Jobs diskutiert. Dabei werden nicht nur die Möglichkeiten, die diese Features bieten und wie man diese einsetzt, aufgezeigt, sondern auch die Probleme diskutiert, die durch die Verwendung dieser Technologien herbeigeführt werden können.
In diesem Zusammenhang wird auch auf die enge Verbindung zum Oracle Advanced Queuing (AQ) hingewiesen. So werden Event-Jobs und Callback-Methoden über AQ realisiert und die Callback-Methoden werden intern wiederum durch Oracle-Scheduler-Jobs abgebildet. Ein unbedachter Einsatz kann hier schnell zu einem instabilen oder nicht performanten Job-Management führen. Aus den praktischen Erfahrungen, die wir an Hand von produktiven Systemen, die bereits mehrere Millionen Jobs verarbeitet haben, gesammelt haben, werden in dieser Präsentation Hinweise gegeben, Workarounds dargestellt und insbesondere auch die möglichen Stolpersteinen bei einem DB-Upgrade diskutiert.

Oracle Scheduler im Überblick
Der Oracle Scheduler erhielt mit Release 10gR1 Einzug in die Oracle Datenbank. Er soll das bis dato bestehende dbms_jobs-Package ablösen mit denen Pl/Sql-Programme zeitgesteuert geplant und im Hintergrund ausgeführt werden konnten. Der Großteil der API-Funktionen befindet sich nun im Package *dbms_scheduler*. Das Package *dbms_jobs* ist aus Gründen der Abwärtskompatibilität natürlich weiterhin vorhanden.
Um es schon vorweg zunehmen: hat man vor ein größeres Batch-System basierend auf den Oracle Scheduler zu implementieren, sollte man die API-Aufrufe durch ein eigenes Interface kapseln.

Über dbms_scheduler können die drei Job-Typen „*PLSQL-Block*“, „*Executable*“ (externes Programm / Shell-Script) oder „*Stored-Procedure*“ ausgeführt werden. Diese ausführbaren Elemente können auch als „*Programs*“ samt Parameter vordefiniert werden und dann von einen oder mehreren Jobs ausgeführt werden.

Den gewünschten Ausführungszeitpunkt gibt man einem Job entweder zur einmaligen Ausführung direkt als *TimeStamp* oder über einen zuvor angelegten *Schedule* mit. Über einen *Schedule* können auch komplexe Wiederholungsintervalle definiert werden. Bezüglich Intervallen ist jedoch zu beachten: wenn ein Job zu dem durch ein Intervall definierten Ausführungszeitpunkt „disabled“ wurde (z.B. wegen einer Wartung), dieser dann beim „enablen“ nicht sofort gestartet wird, sondern auf den nächsten laut Intervall berechneten Ausführungszeitpunkt terminiert wird. Bei einem 5-minütigen Intervall ist das meist zu verschmerzen, bei einem Tagesintervall z.B. für eine tägliche Datensicherung, ist dieses Verhalten schon problematischer.
Zu dem ist zu beachten, dass mehrere Jobs mit dem gleichen *Schedule* z.B. einem 5 Minuten-Intervall zur gleichen Zeit gestartet werden. Die Startzeit wird abhängig von der Startzeit des jeweiligen *Schedules* berechnet und individuell pro Job und dessen Erzeugungszeitpunkt. Um hier Lastspitzen zu

vermeiden, kann es sinnvoll sein mehrere *Schedules* mit gleichem Intervall aber unterschiedlicher Startzeit zu erstellen.

Es existiert ein Unterschied zwischen Eigentümer (Owner) und Erzeuger (Creator) eines Jobs. Ein Job läuft unter den Berechtigungen und Namen des *Owners*. Der *Owner* lässt sich beim Starten überschreiben, der *Creator* nicht. Beim Start eines *Subjobs* ist der *Creator* des aufrufenden Jobs wieder der *Owner* und *Creator* des *Subjobs*. Dies gilt es bei Verschachtelungen von Scheduler-Jobs zu beachten.

Scheduler Views

Bei Jobs, die in einer Hintergrundverarbeitung laufen, ist es natürlich wichtig dem Operator eine ausführliche Möglichkeit zu bieten, sich über den Status oder ggf. Probleme der Jobs zu informieren. Hierzu stellt Oracle mehrere Views zur Verfügung. Leider existiert kein durchgängiger Key auf die Views, so dass man für eine einzelne Information sämtliche Views zusammen-joinen muss. Sobald man jedoch mehrere zehntausend Jobs (häufig besteht die Anforderung Informationen zu gelaufenen Jobs vorzuhalten) im System vorliegen hat liegt darin jedoch ein großes Performance-Problem. Denn jede View für sich ist bereits extrem komplex – insbesondere die häufig relevante Abfrage auf den Job-Status ist sehr aufwendig. Das Joinen dieser Views kann dann schon mal einige Sekunden benötigen. Für eine Oberfläche die aktuelle Status-Informationen präsentieren soll, führt das schnell zu einem unakzeptablen trägen Verhalten.
Eine Abhilfe kann hier das Führen eigener Meta-Daten-Tabellen sein, in denen man Informationen parallel hält. Diese Meta-Daten lassen sich über das bereits eigene Interface zum *dbms_scheduler* Package und durch die später beschriebenen Callbacks befüllen.

Event-Jobs, File-Events und Callback-Routinen

Oracle Scheduler-Jobs können nicht nur Zeit sondern auch Event gesteuert gestartet werden. Dieses wird über Oracle AQ realisiert. Dazu wird dem Job bei der Erzeugung eine Message-Queue mitgegeben. Das Eintreffen einer Message startet dann diesen Job. Hierbei ist zu beachten, dass ein Job während seines Laufs per Default alle Messages/Events konsumiert. Das Auslösen von z.B. drei Events hat nicht die dreifache Ausführung des Jobs zur Folge. Bis Oracle 10g R2 konnte daran auch nichts geändert werden, welches Event-Jobs in der Praxis häufig unbenutzbar machte. Seit Oracle 11g R1 ist es aber möglich, für Event-Jobs die Abarbeitung durch parallele Lightweight-Instanzen zu aktivieren.

```
BEGIN
  DBMS_SCHEDULER.set_attribute('my_event_job','parallel_instances',TRUE);
END;
```

Neu ab 11g R2 sind die File-Events. Zur Erzeugung eines File-Watchers gibt man den Pfad und Dateinamen (ggf. mit Wildcards) der überwacht werden soll an, sowie die OS-User-Credentials für die Zugriffsberechtigung. Der Job, der mit Eintreffen einer neuen Datei ausgelöst werden soll, bekommt an Stelle einer Event-Queue den Namen des File-Watchers in seiner Definition mitgeteilt. Ein solcher auf ein File-Event reagierende Job, bzw. die Stored-Procedure dahinter, muss ein Argument vom Typ *SYS.SCHEDULER_FILEWATCHER_RESULT* besitzen über das die Informationen zur Datei übergeben werden.
Interessant ist auch die Möglichkeit den File-Watcher unter Verwendung des Remote Scheduler Agents (dazu später mehr) einen Pfad auf einem Remote-System überwachen zu lassen.

Insbesondere bei den File-Event-Jobs ist darauf zu achten das *parallel_instances* Attribut zu setzen, da ansonsten parallel eintreffende Dateien nicht abgearbeitet werden. Die Erfahrung zeigt, dass je nach Betriebssystem und File-Transfer –Tool auch die Erkennung neuer Dateien, insbesondere bei gleichen Dateinamen, oder auch die Erkennung der Vollständigkeit der Übertragung ein Problem sein kann. Oracle bietet zwar zwei Parameter zur Justierung bei der Erkennung der Vollständigkeit (*min_file_size* und *steady_state_duration*), jedoch kann es trotzdem vorkommen, dass man um eine eigene File-Watcher-Implementation nicht herum kommt, da Oracle nicht jede Konstellation sicher abdeckt.

Durch die Registrierung von eigenen Callback-Methoden gegen die Scheduler-Event-Queue, lässt sich auf Status-Änderungen der Scheduler-Jobs reagieren. Diese Callbacks werden über die normalen AQ-Mechanismen abgewickelt. Jobs lassen sich so konfigurieren, dass sie bei bestimmten Status-Änderungen eine Message in eine spezielle Scheduler-Queue einstellen. Auf diese Queue kann man dann mit üblicher AQ-Funktionalität z.B. PLSQL/Callback-Methoden registrieren. In diesen lassen sich dann z.B. auch Aktualisierungen der eigenen Meta-Daten-Tabellen abwickeln. Dabei sollte man jedoch beherzigen alle Fehler in diesen Methoden abzufangen. Des Weiteren sollte der Inhalt dieser Methoden performant sein, denn bei einem hochaktiven Batchsystem kommt es zu zahlreichen Status-Wechseln.

Callback-Routinen werden durch AQ wiederum mit internen Scheduler-Jobs abgearbeitet. Seit 11g hat Oracle die Ausführung der Callbacks etwas optimiert, so dass pro Callback nicht mehr ein neuer Job erzeugt wird, sondern mehrere Callbacks nacheinander mit dem gleichen Job abgearbeitet werden. Das spart Overhead, macht aber auch die Arbeit für den QMON-Prozess der diese internen Jobs verwaltet und die Callbacks zuweist, komplexer und damit fehleranfälliger. So kann es immer wieder vorkommen, dass ein Callback mehrfach ausgeführt wird. Daraus folgt auch, dass der Zugriff auf die Event-Queue innerhalb der Callback-Methode immer mit einem Timeout erfolgen sollte, ansonsten bleibt in solchen Fällen das ganze System hängen.
Erfahrungsgemäß ist der QMON-Prozesse, im speziellen der Koordinator, gerne einmal daran Schuld, wenn es im Event-Handling zu seltsamen Phänomenen kommt, z.B. wenn jedes Callback-Event mit einem eigen Job abgearbeitet wird. Bei einer Single-Database reicht es im Extremfall diese durchzustarten. Beim RAC muss es dann schon einmal der komplette Cluster sein, da im RAC der QMON-Koordinator ein vom Cluster verwalteter Prozess ist. Selbst beim Durchstarten einer Instanz (oder aller abwechselnd) kann das Problem bestehen bleiben, da es sozusagen mitverschoben wird.

Job-Chains

Über Job-Chains können mehrere Jobs, die dann *Steps* genannt werden, zu einem Workflow definiert werden, wobei Chains auch wiederum verschachtelt werden können. Ein *Step* kann also auch wieder eine *Chain* sein. Eine Chain Definition kann durch einen oder mehrere Jobs ausgeführt werden, wobei die *Steps* dann jeweils als *Subjobs* von Scheduler ausgeführt werden. Die *Steps* sind durch Regeln miteinander verbunden, die jeweils nach dem Start der *Chain* und nach jedem *Step* ausgewertet werden.
Der Oracle SQLDeveloper bietet neben der API auch einen grafischen Ansatz *Chains* zu definieren.

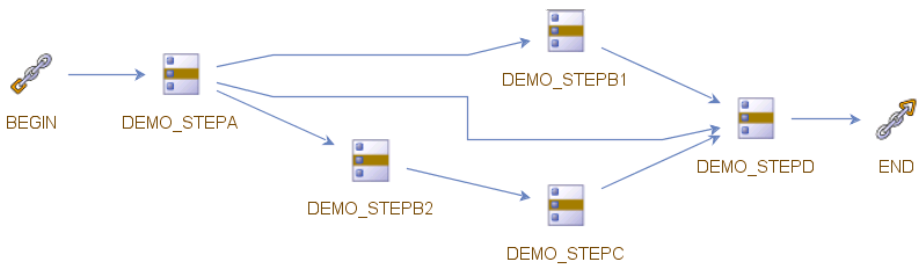

Abb. 1: Chain-Definition im Oracle SQLDeveloper

Eine grafisch aufbereitet Sicht auf die laufenden *Chains* bietet er leider nicht. Für das Operating ist dieses jedoch meistens unerlässlich und muss eigens über die Scheduler- und die speziellen Chain-Views implementiert werden. *Steps* lassen sich über die API als zu überspringen oder als zu pausieren definieren und zur Laufzeit kann zusätzlich auch der Status manipuliert werden. Die Adressierung der *Steps* ist insbesondere bei verschachtelten *Chains* gar nicht so einfach. Dieses geschieht entweder über den Subjob-Namen und den Step-Namen der Sub-Chain. Der Subjob-Name wird aus dem Namen des *Steps* abgeleitet den er ausführt. Laufen mehrere *Steps* mit gleichem Namen parallel, ergänzt Oracle die Job-Namen um eine laufende Nummer, was die Zuordnung erschwert. Einfacher ist es die *Steps* bei verschachtelten *Chains* per Punkt-Notation anzusprechen.

```
CHAIN_A_STEP_A.CHAIN_B_STEP_A
```

In einigen Fällen kann es einfacher sein einen eigenen programmatischen Ansatz für die Realisierung hierarchischer Jobs zu wählen. Über eine hierarchische Meta-Daten-Tabelle lässt sich die hierarchische Abhängigkeit leicht herstellen und visualisieren. D.h. beim Erzeugen eines neuen Jobs trägt man ggf. einfach eine Vater-Kind-Beziehung ein.

Im programmatischen Ansatz können auch einfacher komplexe Bedingungen für das Überspringen oder Starten eines Jobs/Steps geprüft werden. Jobs können dabei auch innerhalb und nicht nur am Ende eines Jobs verzweigt werden. Ein fachlicher Job muss dazu nicht technisch zerteilt werden.

Zudem lassen sich so auch Kontrollstrukturen wie Schleifen einfach realisieren. Mit Oracle-Chains wird dies schwierig, da bereits gelaufene Jobs als solche in der Kette markiert und nicht wiederholt ausgeführt werden. Dazu muss zuerst der Status des *Steps* auf *NOT_STARTED* gesetzt werden, wozu ein technischer Zwischen-*Step* notwendig wäre, der aber auch wiederum vor dem nächsten Durchlauf umgesetzt werden muss.

Dafür ist es ohne Chains schwieriger einen Job synchron abzusetzen, sprich mit der weiteren Ausführung bis zur Beendigung eines Subjobs zu warten. Denn Jobs, die man mit der *dbms_scheduler.run* -Methode synchron ausführen könnte, laufen dann auch in der gleichen Session und unter deren Transaktion. Ist dieses nicht gewünscht, lässt sich etwas aufwendiger ein Verfahren über die Callback-Events des Schedulers realisieren.

Remote Job Execution

Der Oracle Scheduler ist seit 11g R1 in der Lage Jobs auch auf Remote-Systemen auszuführen. Dazu muss dort der Remote Scheduler Agent installiert werden. Dieser ist in den Installationsquellen des Datenbank-Clients enthalten, muss aber separat in ein eigenes Oracle-Home installiert werden. Ein bereits vorhandenes Oracle-Home mit einer Client-Installation kann nicht erweitert werden.

In 11g R1 können Jobs vom Typ *Executable* remote ausgeführt werden. Ab 11g R2 kommen Remote Database Jobs hinzu, mit denen dann auch Jobs vom Typ *PL/SQL* und *Stored Procedure* ausgeführt werden können. Auf Seiten der Remote Datenbank muss allerdings zusätzlich der Remote Scheduler Agent installiert werden. Die Abwicklung erfolgt also nicht über Database Links, sondern über den gleichen Mechanismus wie bei den *Executables*.

Die Log-Dateien von Remote-Jobs, in die Standard-Out und Standard-Error der Executables umgeleitet werden, können über die *dbms_scheduler.get_file* Prozedur als Lob in die Datenbank transferiert werden. Da es in *dbms_scheduler* auch eine *put_file* Prozedur gibt, kann man mit diesen Methoden eigentlich wunderbar auch einen beliebigen File-Tranfer zwischen Datenbank und Remote-Host realisieren. Da ein File-Watcher, der gegen eine Remote-Destination arbeitet, seinem Job auch nur Informationen über die Datei, die das Event ausgelöst hat, aber nicht deren Inhalt sendet, ist dieses

dann auch die Möglichkeit auf den Datei-Inhalt zuzugreifen.
Diese Prozeduren, die man auch gegen den lokalen Datenbank-Host benutzen kann, sind, sobald sie gegen eine Remote-Destination verwendet werden, jedoch auch nicht unproblematisch. Die Datei-Operationen werfen keine Fehler, auch wenn sie missglücken. So liefert ein *get_file* immer ein leeres LOB, egal, ob die Datei nicht existiert oder ob sie leer ist.
Der Oracle Support sieht dieses jedoch nicht als Bug, sondern als Feature-Request an. Die gleiche Prozedur gegen den lokalen Host laufend liefert jedoch sehr wohl einen Fehler, wenn die Datei nicht existent ist.

Ein weiteres Problem der Remote Jobs liegt in dem manchmal inkorrekten Job-Status. In der Datenbank kann ein Job z.B. als *Failed* oder *Stopped* geführt werden, auch wenn der Prozess in Wahrheit noch auf der Remote-Seite läuft. Provozieren kann man das z.B. indem man den Remote Scheduler Agent –Dienst während der Ausführung eines Jobs durchstartet. Steht so ein Job auf automatischen Neustart oder wird er durch einen unwissenden Operator wieder aufgesetzt, kommt es zu Doppelläufern. Wo dieses ein „No-Go“ ist, muss mit mehr oder weniger Aufwand der mögliche Doppellauf in den Executables verhindert werden.

Wissenswertes zu Datenbank Migration und Updates

Es gibt diverse Gründe Datenbank-Migrationen per Import/Export durchzuführen, zum Bsp. bei einem Downgrade von Enterprise zur Standard-Edition. Dabei wird oft ein Export ohne Daten durchgeführt um erst einmal das Schema zu migrieren und später - ggf. über einen anderen Weg - die Daten nachzuholen. Oracle weist zwar ausdrücklich darauf hin, dass Scheduler-Objekte nur mit den neueren expdp und impdp exportiert werden sollen, aber damit gibt es Probleme, und zwar genau im Falle von Export ohne Daten. Für den Export schreibt Oracle die Laufzeit-Informationen der Jobs aber auch einige Definitionsdaten, wie z.B. Programm-Argument-Definitionen, in eine temporäre Tabelle, anstatt diese als Script zu exportieren. Beim Export ohne Daten wird diese interne Tabelle zwar schön aufgebaut, aber „natürlich“ nicht exportiert. Im Ziel kommen dann z.B. Programmdefinitionen ohne Argumente an. Um dem vorzubeugen sollte man möglichst alle Scheduler-Objekte und Jobs in einem eigenen Schema erstellen bzw. laufen lassen.

Datenbank-Upgrades (oder Patchset-Upgrades) sollten auf Datenbanken mit einer Standard-Konfiguration nicht an einem Samstag durchgeführt werden. Wer wissen will warum, sollte diesen Vortrag besuchen oder mal in die Oracle-Support-Note 731678.1 sehen. Auch wenn dort etwas von *fixed* in 10g R2 steht, ist das Problem zumindest bis 11.2.0.3 noch aktuell.

Kontaktadresse:
Torsten von Osten
inconso AG
Dorotheenstrasse 60
D-22301 Hamburg

Telefon: +49 (0) 40 181320 2712
Fax: +49 (0) 40 181320-278
E-Mail tvonosten@inconso.de
Internet: www.inconso.de