Multiprocessing in PL/SQL

Der Weg aus dem Single Thread

Christian Wille
Pitss GmbH
Stuttgart

Schlüsselworte
PL/SQL, DBMS_SCHEDULER, Multithreading, Multiprocessing.

Einleitung

In vielen Verarbeitungen ist eine parallele Abarbeitung der Aufgaben ratsam oder gar notwendig, um die Ausführungsdauer in einem akzeptablen Rahmen zu halten. Die beiden grundsätzlichen Ansätze in diesen Bereich sind Multithreading und Multiprocessing (auch Multitasking genannt). Der Unterschied zwischen den beiden Verfahren ist die Ebene auf der die Nebenläufigkeit erzeugt wird. Einerseits werden beim Multithreading mehrere Ausführungsfäden (Threads) in einem abgeschotteten Prozeß der Software gestartet, so daß die Software die Synchronisation und Fehlerbehandlung der Threads verantwortlich ist. Andererseits werden beim Multiprocessing mehrere abgeschottete Prozesse gestartet, deren Ausführung das Betriebssystem übernimmt.

Im folgenden werden die verbreiteten JAVA Technoligen Thread und Process mit den Möglichkeiten des PL/SQL verglichen. Abschließend wird ein erfolgreiches Beispiel aus der Praxis vorgestellt.

Vergleich Java - PL/SQL

PL/SQL wird ausschließlich in einem Prozeß ausgeführt und bietet keine Möglichkeiten des Multithreadings außer durch Java Packages. Allerdings bietet das Package DBMS_SCHEDULER die Möglichkeit Jobs aus PL/SQL Code zu planen und zu starten. Diese Jobs werden gewöhnlich in der DB Console gepflegt. Ein Job ist ein losgelöster separater Prozess im Betriebssystem. Damit kann ein Multitasking umgesetzt werden.

Folgende Tabelle vergleicht die Möglichkeiten der drei Varianten losgelöste Verarbeitungsstränge zu bearbeiten und zu kontrollieren.

Anforderung eines parallelen Verarbeitungs-stranges	Java Thread	Java Process	PL/SQL
Starten	`Thread p = new Thread(); p.start();`	`Process p = Runtime.getRuntime() .exec( "edit.exe");`	`DBMS_SCHEDULER.CREATE_JOB ( job_name => 'JOB101', job_type => 'PLSQL_BLOCK', job_action => 'begin worker(101); end;', start_date => SYSDATE, enabled => TRUE);`
Beenden	`p.interrupt ();`	`p.destroy();`	`DBMS_SCHEDULER.STOP_JOB ( job_name => 'JOB101', force => true);`
Ausführung abwarten	`p.join();`	`p.waitFor();`	X
Ausführung planen	X	X	`DBMS_SCHEDULER.CREATE_JOB (... start_date => SYSDATE+1,...);`
Ausführung pausieren extern	`p.yield();`	X	X
Ausführung pausieren intern	`p.sleep (100);`	X	`DBMS_LOCK.sleep(100);`
Priorität setzen	`p.setPriori ty (1);`	X	`DBMS_SCHEDULER.SET_ATTRIBUTE ( Name => 'JOB101', attribute=>'job_priority', value => 1); //Gilt nur innerhalb der Job Klasse`
Eigenschaften abfragen	`Id, Name, Priority, State, Alive, Interrupted`	X	`Über die Tabelle USER_SCHEDULER_JOBS: OWNER,JOB_NAME,JOB_SUBNAME,JOB_CREATOR, CLIENT_ID,GLOBAL_UID,PROGRAM_OWNER, PROGRAM_NAME,JOB_TYPE,JOB_ACTION, SCHEDULE_OWNER,SCHEDULE_NAME, SCHEDULE_TYPE,START_DATE, REPEAT_INTERVAL,JOB_CLASS,ENABLED, AUTO_DROP,STATE,JOB_PRIORITY, MAX_RUN_DURATION,LOGGING_LEVEL,SYSTEM, NLS_ENV,SOURCE,DESTINATION,COMMENTS`

Tab. 1: Tabelle Vergleich Java PL/SQL

Anhand dieser Tabelle wird schnell ersichtlich, daß parallele Verarbeitung in PL/SQL durchaus eine ernste zu nehmende Alternative zu den klassischen Multithreaded oder Multiprocessing Anwendung darstellen kann.

Bedingung für den Einsatz ist, daß nicht bis zum Ende eines einzelnen Prozesses gewartet oder die Ausführung pausiert werden muß. Natürlich müssen sich die weiteren Anforderungen an die Software im Bereich des PL/SQL umsetzen lassen.

Beispiel aus der Praxis

Als Beispiel aus der Praxis soll die Kassenverabeitung eines Handelsunternehmen dienen. Dort werden jeden abend über 1400 Textdateien ausgelesen und die Daten in entsprechenden Tabellen des Warenwirtschaftssystem, der Kassenverarbeitung und der Revision gespeichert. Im Abschluß der Verarbeitung wird die Geschäftsführung per SMS über die eingearbeiteten Umsätze informiert.

Im Ursprungssystem wurden die Dateien auf den Kassen der Filiale generiert. Ein Abrufserver verschob die Dateien auf einen File-Server. Eine dotNet Anwendung auf den Verarbeitungsserver startete bis zu 6 gleichzeitige Threads, wenn genügend Dateien auf den File-Server bereitstanden. Jeder dieser Threads laß die Daten aus einer Datei aus und trug diese in der Datenbank ein.

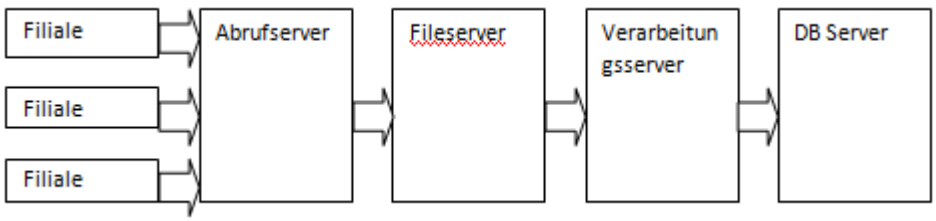

Abb. 1: ursprünglicher Aufbauder Verarbeitung

Im Rahmen der Umstellung wurde der File-Server auf den Datenbank-Server verschoben und die dotNet Anwendung wurde durch PL/SQL Prozeduren ersetzt.

Eine PL/SQL Prozedur übernahm die Steuerung der Verarbeitung und startet innerhalb eines vorgegebenen Zeitfenster mittels des Package DBMS_SCHEDULE in Abhängigkeit der Anzahl vorhandener Dateien und der aktuell aktiven Sessions auf der Datenbank mehrere Tasks der PL/SQL Prozedur, die die Kassendaten analog zur alten dotNet Anwendung in die Datenbank einträgt.

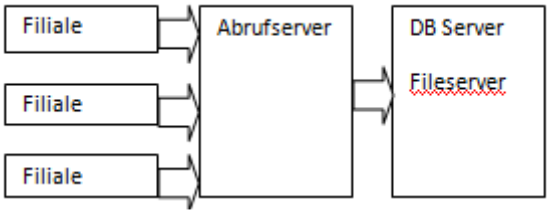

Abb. 1: neuer Aufbau der Verarbeitung

Hierdurch ergab sich eine Verkürzung der durchschnittlichen Verarbeitungsdauer einer einzelnen Datei von 26 auf 20 Sekunden. Mit den Faktor 1400 Dateien ergibt sich hieraus eine Einsparung von 2,3 Stunden.

Weiter wurde die Anzahl der parallel laufenden Prozesse an die CPU Auslastung der Datenbank gekoppelt, so daß eine eine annähernd gleichmäßige Auslastung des Datenbank Servers erreicht wird und Prozesse, die nicht der Kassenverarbeitung angehören nicht beeinträchtigt werden.

Diese Umstellung hat zur Folge, daß die Geschäftsführung nicht mehr um ca. 23:30 Uhr sondern bereits um ca. 21:45 Uhr ihre Umsatzzahlen zugestellt bekommt.

Zusätzlich wurden einerseits mögliche Fehlerquellen des Gesamtprozesses reduziert, so daß der Gesamtprozeß nicht nur schneller sondern auch stabiler geworden ist. Andererseits wurde die Anzahl der Systeme verkleinert, so daß Wartungs- und Monitoringaufwände ebenfalls reduziert wurden.

Kontaktadresse:
Christian Wille
Pitss GmbH
Zettachring, 2
D-70567 Stuttgart

Telefon: +49 (711) 72-8752 05
Fax: +49 (711) 72-8752 01
E-Mail cWille@pitss.de
Internet: www.pitss.com