Parallelisierung in Datenbank-Batchverarbeitungen

Dr. Kurt Franke
Cellent Finance Solutions AG
Stuttgart

Schlüsselworte

Parallele Abfragen, Parallel-Degree, Parallelverarbeitung

Einleitung

In Datenbank-Batchjobs gibt es immer die Anforderung, dass diejenigen Batchverarbeitungen, die nicht gleichzeitig mit dem Onlinebetrieb stattfinden können, innerhalb eines vorgegebenen Zeitfensters abgeschlossen sein müssen. Andererseits sollen auch die Investitions-Kosten für die Hardware in einem angemessenen Rahmen bleiben. Das führt zwangsläufig auch zu der Forderung, dass die Hardware möglichst optimal genutzt wird. Es sollen eben nicht größere Teile der Austattung wie z. B. Prozessorkerne oder IO-Bandbreite längere Zeit unbenutzt bleiben. Beide Anforderungen führen dazu, bei Batchverarbeitungen soweit wie möglich mit Parallelisierung gearbeitet wird, idealerweise so, dass alle derartigen Verarbeitungen während des Offline-Zeitfensters abgearbeitet werden können und dabei die Hardware optimal ausgelastet wird. Wenn noch Batchverarbeitungen während des Online-Zeitfensters erforderlich sind, z. B. wegen umgehend benötigter Ergebnisse, oder weil das vorhergehende Zeitfenster nicht ausreichend war, so stehen dafür weniger Rechen- und IO-Resourcen zur Verfügung, weil die Online-Perfomance dadurch nicht vermindert werden darf.

Welche Möglichkeiten zur Parallelisierung gibt es in der Datenbank ?

Oracle unterstützt die Parallelisierung von SQL-Statements, sofern man eine Enterprise-Edition einsetzt. In DDL-Statements ist oft eine PARALLEL DEGREE Klausel möglich, in welcher der Parallelisierungsgrad bei einer Objects-Erzeugung oder Änderung sowie nachfolgend als Object-Eigenschaft für Queuries spezifiziert werden kann. In Queuries besteht neben der zuvor erfolgten Definition des Parallellisierungsgrads auf Objectebene noch die Möglichkeit der Verwendung von SQL-Hints für einzelne darin selectierte Objecte sowie Unterabfragen. Sinnvoll ist diese Form der Parallelisierung nur für Statements zur Massendatenverarbeitung. Bei Einzelsatzverarbeitung würde das System zwar besser ausgelastet, ohne jedoch dadurch eine bessere Verarbeitungs-Performance zu erreichen – im Gegenteil wird hier eher eine Verschlechterung eintreten, weil die Kommunikation zwischen dem Queury-Koordinator und den Query-Ausführungs-Prozessen hinzukommt – das ganze natürlich nur dann, wenn sich der Queury-Optimizer überhaupt zu einem parallelen Ausführungsplan überreden läßt.

Gerade in Batchverarbeitungen gibt es noch eine weitere Parallelisierungsmöglichkeit. Im Allgemeinen gibt es sowieso mehr Daten zu verarbeiten, als bei vorhandener Hardware im verfügbaren Zeitraum durchgeführt werden kann. Die Verarbeitungen müssen dann zumindest teilweise gleichzeitig laufen. Dabei muss sichergestellt werden, dass bei voneinander abhängigen Verarbeitungen die korrekte Reihenfolge eingehalten wird und dass keine Verarbeitungen gleichzeitig laufen, die durch indirekte Wechselwirkung über die Daten das Ergebnis verfälschen würden. Weiterhin sollte dabei sichergestellt werden, dass die vorhandenen Resourcen optimal genutzt, aber nicht überlastet werden, weil das letztendlich wieder zu einer Verminderung der Performance führt.

SQL Parallelisierung

Die parallele oder nicht parallele Ausführung von SQL-Statements oder Sub-Statements wird vom SQL-Optimizer bei der Erstellung des Ausführungsplans festgelegt. Die Parallel-Query-Sessions werden von der Datenbank bereitgestellt und verwaltet, ohne dass der Ersteller der Statements dazu etwas tun muss und dabei auch gar keine Eingriffsmöglichkeiten hat. Das ist ja durchaus positiv, weil der Entwickler dafür keinerlei Aufwände hat.

Der Optimizer bestimmt den Parallelisierungsgrad eines Zugriffs aus den beim Object gespeicherten Degree-Wert und aus Parallel-SQL-Hints im auszuführenden Statement, soweit nicht andere Bedingungen dies verhindern. Eine solche Bedingung ist beispielsweise die bereits komplette Verwendung der konfigurierten Parallel-Query-Server (Initialisierungs-Parameter parallel_max_servers). Hints haben dabei Vorrang vor den beim Object hinterlegten Werten. Hints sind dabei als optimalere Festlegung des Parallelisierungsgrads zu sehen, weil diese auf Statement- oder sogar auf Sub-Statement-Ebene definiert werden und damit genau passend gewählt werden können. Eine Festlegung auf Objectebene wirkt hingegen auf verschiedene Statements mit dem jeweiligen Object. Dass je nach Statement unterschiedliche Parallelisierungsgrade für ein Object sinnvoll sein können, kann mit einer Festlegung auf Objectebene nicht abgebildet werden, es sei denn, man greift in solchen Fällen doch auf SQL-Hints zurück, um den auf Objectebene definierten Wert zu überschreiben.

Definition des Parallelisierungsgrad eines Objects:

```
ALTER TABLE mytable PARALLEL 8;
```

Definition des Parallelisierungsgrad in einem Statement:

```
SELECT /*+ parallel(t,16) */ FROM mytable t ...
```

Diese Form der Parallelisierung bewirkt nur dann eine erhöhte Verarbeitungsperformance, wenn möglichst alle größeren Datenmengen mit Massen_SQL-Statements bearbeitet werden. Gerade hier kann Oracle die Daten in einzelne Bereiche aufteilen, die jeweils einem einzelnen Parallel-Query-Prozess zur Verarbeitung zugeteilt werden. Bei Einzelsatzverarbeitungen in einer Cursor-Loop würde – falls sich der Optimizer dann überhaupt zu einem parallelen Executionplan bewegen läßt – die Performance wegen des zusätzlichen Overheads sogar abnehmen.

Wenn eine Datenbank-Applikation für verschiedene Kunden mit um Größenordnungen unterschiedlichen Datenmengen entwickelt wird, ist es von Vorteil, ein Möglichkeit zu haben, die Parallelisierungsgrade für den jeweiligen Kunden über Einstellungsparameter geeignet anzupassen, so dass sie optimal zur Datenmenge und zur Hardware passen. Dies ist möglich, soweit mit dynmischen SQL-Statements gearbeitet wird, und natürlich bei Index- oder Statistik-Erzeugung während der Verarbeitung. Bei statischen SQL-Statements funktioniert dies nicht, weil zum Zeitpunkt des SQL-Parsing die SQL-Hints als Literale im Statement eingebettet sein müssen. Gerade bei Verarbeitungen unterschiedlicher Subsets einer Entität, die nach diesen Subsets in Partitionen gegliedert sind, werden oft Arbeitstabellen eingesetzt, deren Name erst zur Laufzeit berechnet wird, um eine Einschränkung der Subsets zur Programmierzeit zu vermeiden. Schon dadurch wird der Einsatz dynmischer SQL-Statements notwendig und eine Parametrisierung via Hints stellt keinen wesentlichen Mehraufwand dar. Bei dynamischem SQL werden die aus einer Parametertabelle gelesenen Wert via concat in den SQL-String eingebaut. Sogar verschiedene Multiplikatoren auf einen Parameterwert sind hier für verschiedene Statements machbar. Es ist natürlich auch möglich, mehrere solcher Parameter einzusetzen, um eine in manchen Fällen sinnvolle unabhängige Einstellungsmöglichkeit für verschiedene Statements zu erhalten.

Parametrisierte Definition des Parallelisierungsgrads in einem Statement:

```
DECLARE
  stmnt VARCHAR2(32767);
  p_degree BINARY_INTEGER;
BEGIN
  SELECT p_value INTO p_degree FROM param_table WHERE p_name = 'DEGREE';
  stmnt := 'CREATE TABLE mytable2 AS' || chr(10) ||
    ' SELECT /*+ parallel(t,' || to_char(p_degree) || ') */' || chr(10) ||
    ' FROM mytable t1';
  EXECUTE IMMEDIATE stmnt;
END;
```

Parallel laufende Verarbeitungen

Bei parallel laufenden Verarbeitungen werden nicht nur SQL-Statements, sondern auch die gesamte darum herum ausgeführte Logik – sei es in PL/SQL oder in Java – unabhängig voneinander in verschiedenen Sessions ausgeführt. Als Voraussetzung für gleichzeitiges Laufen gilt hier, das keine gegenseitige Abhängigkeit besteht, d. h. dass nicht eine Verarbeitung die verarbeiteten Daten der anderen Verarbeitung als Input benötigt, also eine Folgeverarbeitung der ersten ist. Insbesondere bei Verarbeitungen, die auch DDL wie Exchange Partition einsetzen, dürfen gleichzeitig laufende Verarbeitungen auch nicht die gleichen Daten bearbeiten, z. B. um verschiedene Attribute zu berechnen.

Anders als bei der SQL-Parallelisierung gibt es hier keinen Komplettservice der Oracle-Datenbank. Erforderliche Sessions müssen entwicklerseitig bereitgestellt und verwaltet werden. Seit Oracle 10 kann das mit dem dbms_scheduler Package mit bis zu 999 Prozessen durchgeführt (zuvor mit dbms_job maximal 36), so dass eine separate Session-Erzeugung durch Connect von außerhalb der Datenbank nicht mehr erforderlich ist. Es ist jedoch immer erforderlich, eine Kommunikation zwischen der/den Sessions, die eine Verarbeitung einer separaten Session zur gleichzeitigen Ausführung zuteilen, und den Sessions, die diese Verarbeitungen durchführen, zu implementieren.

Im einfachsten Falle wird das durch die Hinterlegung eines Statuswerten, ggf. mit Oracle-Fehler, in einer Steuertabelle erledigt. Für eine korrekte Steuerung ist es notwendig, sicher zu wissen, ob die eine Session, der eine Verarbeitung zugeteilt wurde, noch existiert oder abgebrochen ist. Sofern die Steuerungssession keine Rechte auf v$session etc. hat, bleibt nur die Möglichkeit, dass eine Regel zur Berechnung eines Locknamens für alle Verarbeitungen aufgestellt wird und jede Session, der eine Verarbeitung zugeteilt wurde, den entsprechenden Lock via dbms_lock Package allokiert (Rechte auf dbms_lock müssen zugeteilt worden sein). Dabei muss sichergestellt sein, dass diese Locks die Lebendauer einer Session haben, solange sie nicht wieder freigegeben werden. Damit ist es jederzeit möglich, festzustellen, ob eine Verarbeitung noch aktiv ist und eben auch, ob sie abgebrochen ist, auch wenn sie keine Möglichkeit mehr hatte, diesen Zustand zurückzumelden. Es läßt sich nun sicherstellen, das eine Verarbeitung nur dann angestartet wird, wenn sie nicht bereits läuft. Ebenso kann sichergestellt werden, dass eine notwendige Vorverarbeitung erfolgreich abgeschlossen wurde.

Da diese Parallelisierungsmethode die kompletten Verarbeitungen betrifft, ist es damit möglich, mehrere mit Cursor-Loops durchgeführte Verarbeitungen parallel durchzuführen. Damit kann auch bei solchen Verarbeitungen eine bessere und gleichmäßigere Auslastung der Hardware erreicht werden, vorausgesetzt, es stehen genügend gleichzeitig ausführbare Verarbeitungen zu Verfügung. Dies ist jedoch eine Frage des Designs der einzelnen Verarbeitungen. Idealerweise erfolgt dies so, das Verarbeitungen immer dort unterteilt werden, wo Serialisierungspunkte mit anderen Verarbeitungen erforderlich sind. Dann sind alle Serialisierungspunkte im Bereich der Steuerung verfügbar und

können über UND-verknüpfte Abhängigkeiten einfach gehandelt werden. Bei einem derartigen Design werden die Parallelisierungsmöglichkeiten von Verarbeitungen in separaten Sessions optimal genutzt.

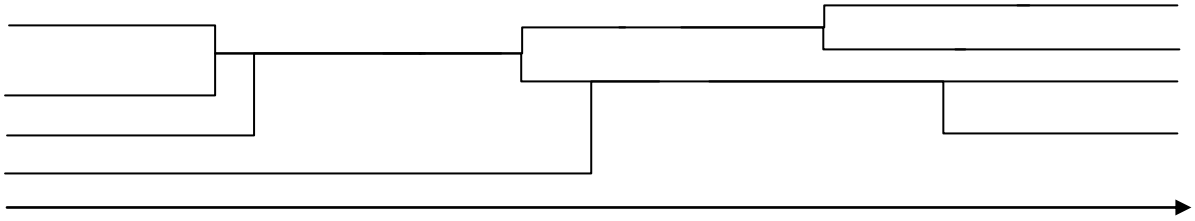

Abb. 1: Jeder Serialisierungspunkt erfordert eine Zusammenführung vorher paralleler Verarbeitungen

Wenn sich hingegen Serialisierungspunkte innerhalb einer geschlossenen Verarbeitung befinden, müssen bis zum letzten Serialisierungspunkt der betrachteten Vearbeitung alle benötigten Daten aus anderen Verarbeitungen bereitstehen, bevor die betrachtete Verarbeitung gestartetet werden darf, oder, falls dies gar nicht möglich ist, die anderen Verarbeitungen seriell in die betrachtete Verarbeitung integriert werden.

Alternativ wäre es noch möglich, in die einzelnen Verarbeitungen Mechanismen zu integrieren, die es erlauben, Informationen über den Stand der jeweiligen Verarbeitung auszutauschen. Dies kann über IPC erfolgen – z. B. via dbms_pipe Package – oder einfach durch Schreiben von Zwischenzuständen in eine Tabelle, die von anderen Verarbeitungen jeweils vor einem eigenen weiteren Zwischenschritt überprüft wird. Bezüglich der Entwicklungsresourcen ist dieses Verfahren jedoch aufwändiger, weil Steuerlogik in den einzelnen Verarbeitungen implementiert werden muss und diese dadurch auch unübersichtlicher und weniger einfach wartbar werden.

Zusammenfassung

Die Parallelisierungsgrade bei gleichzeitig laufenden Verarbeitungen werden hauptsächlich vom Design dieser Verarbeitungen bestimmt und sind dadurch kaum nach einer Auslieferung an einen Kunden beeinflußbar. Bei der SQL-Parallelisierung hingegen kann bei dynamischem SQL ein einstellbarer Parameterwert als Basis zur Bestimmung eines Parallelisierungsgrads via SQL-Hint verwendet werden. Es trifft sich gut, dass die dynamischen SQL-Statements vorwiegend bei Tabellen mit großen Datenmengen eingesetzt werden, bei denen mit Exchange Partition und dynamisch erzeugten Arbeitstabellen gearbeitet wird, um eben auch gleichzeitig laufende Verarbeitungen mit andern Subsets der Daten, aber eben mit gleichartigen Arbeitstabellen, zu ermöglichen. Da stellt eine Einbindung eines Parameterwertes in einen schon vorhandenenen SQL-String via String Concatenation keinen wirklich signifikanten Mehraufwand dar.

Kontaktadresse:

Dr. Kurt Franke
Cellent Finance Solutions AG
Calwer Straße 33
D-70173 Stuttgart

Telefon: +49 (0) 711-222992-676
Fax: +49 (0) 711-222992-899
E-Mail Kurt.Franke@cellent-fs.de
Internet: www.cellent-fs.de