Herrmann & Lenz
Services

Parallele Programmierung in SQL und PL/SQL

Peter Bekiesch
Dierk Lenz

DOAG 2011 Konferenz und Ausstellung
17. November 2011

DIE DATENBANK-PROFIS.

Herrmann & Lenz Services GmbH
Herrmann & Lenz Solutions GmbH

- Erfolgreich seit 1996 am Markt
- Firmensitz: Burscheid (bei Leverkusen)
- Beratung, Schulung und Betrieb/Fernwartung rund um das Thema Oracle Datenbanken
- Schwerpunktthemen: Hochverfügbarkeit, Tuning, Migrationen und Troubleshooting
- Herrmann & Lenz Solutions GmbH
  - Produkte: Monitoring Module, OraInfo, REMBO
  - Stand Nr. 212 auf Ebene 2

Inhalt

- Wozu eigentlich Parallelisierung?
- Mittel und Wege in SQL
- Nutzung von PL/SQL APIs
- Parallelisierung von Außen

Wozu Parallelisierung?

- „Teile und Herrsche“ – Aufteilung einer großen Aufgabe in kleinere Unteraufgaben
- Unteraufgaben laufen gleichzeitig
- Mit n Unteraufgaben Reduzierung der Laufzeit auf $1/n$ möglich…
- Wenn alles optimal läuft
  - Unteraufgaben gleich groß
  - Ressourcen stehen zur Verfügung

Vorsicht!

- Warum soll ich mir die Mühe machen, Indizes anzulegen, wenn ein parallelisierter Full Table Scan in weniger als einer Sekunde läuft?
- Mögliche Antwort: Weil man das nicht mit vielen Benutzern gleichzeitig machen kann!

Paralleles SQL in der Datenbank

- Weitgehend automatisch
- SELECT
- DML (INSERT)
- DDL
  - CTAS
  - CREATE INDEX
  - REBUILD INDEX
  - ENABLE VALIDATE CONSTRAINT

Manchmal nur mit Tricks...

- Einschalten von Constraints kann sehr zeitintensiv sein!
- Mehrstufiges Einschalten möglich:
  1. ENABLE NOVALIDATE: überprüft keine bestehenden Daten; Constraint-Prüfung nur für alle folgenden DML-Befehle
  2. ALTER TABLE <tab> PARALLEL <n>
  3. ENABLE VALIDATE: sperrfreies, paralleles Nachvalidieren
  4. !Nicht vergessen! ALTER TABLE <tab> NOPARALLEL

Grundlagen SQL-Parallelisierung

- Prozesse müssen verfügbar sein
- `parallel_min_servers` / `parallel_max_servers`
- Enterprise Edition!

Degree of Parallelism

- DOP für DB-Objekte bei CREATE / ALTER einstellbar

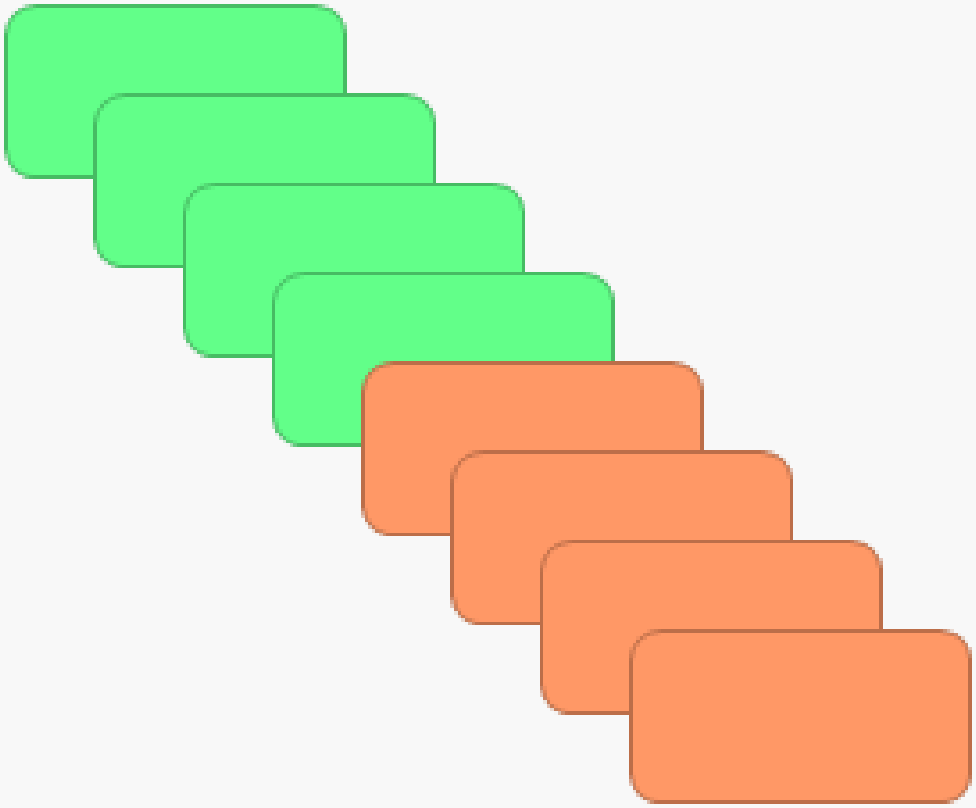
```
CREATE TABLE <tab>
PARALLEL DEGREE <n>
AS
SELECT …
```

- Bei einzelnen Befehlen mit Hint

```
SELECT /*+ PARALLEL 4 */ …
```

Wie viele Prozesse werden mit DOP n beschäftigt?

- $(2 * n) + 1$
- Maximal zwei Schritte aus dem Ausführungsplan werden gleichzeitig mit je n Prozessen ausgeführt.
- Der vorhandene Serverprozess wird zum Query Coordinator.



DOP=2

DOP=2

Neue Parameter in 11.2: Automatic Degree of Parallelism

- Auto-DOP Versuch zur Lösung des Problems

  Ausbalancierung von Parallelisierung in SQL ohne Eingriff in die Programmierung (Hints)

Auto-DOP

- Parameter `parallel_degree_policy`
  manual / limited / auto
- Parameter `parallel_min_time_threshold`
- Automatische Bestimmung des DOP
- In-Memory-Ausführung
- Parallel Statement Queueing

Neu in 11.2: dbms_parallel_execute

- API zur Steuerung von parallelen UPDATEs / DELETEs
- Ermöglicht flexible Parallelisierung in SQL und PL/SQL
- Prozeduren:
  - `create_task` erstellt eine parallele Task
  - `create_chunks_by_rowid` teilt eine Tabelle in gleich große ROWID-Bereiche
  - `run_task` startet parallele Prozesse zur Abarbeitung der Bereiche
- Übrigens: Funktioniert mit Standard Edition!

dbms_parallel_execute

- Voraussetzung: Systemprivileg CREATE JOB
- Parallelisierungsgrad durch die Anzahl gleichzeitig laufender Scheduler Jobs limitiert
- Diverse Möglichkeiten, *Chunks* zu erstellen
  - By ROWID
  - By NUMBER
  - By SQL
- Sehr flexibel durch Verwendung eines dynamischen SQL-Befehls (UPDATE, DELETE, ... PL/SQL-Block?)
- Variablen :start_id und :end_id müssen vorhanden sein

Nutzung des Schedulers

- Idee (wie `dbms_parallel_execute`): Mehrere Jobs zur gleichzeitigen Abarbeitung von Teilaufgaben aufsetzen
- Generierung von Jobs in PL/SQL automatisierbar
- Scheduler bietet exakte zeitbasierte Steuerung

Nutzung von Advanced Queueing

- Streams AQ ist Bestandteil des Oracle RDBMS
- Producer/Consumer Queues
- Idee: Beliebige Consumer nehmen sich Unteraufgaben aus einer AQ
- Besonders effizient bei unterschiedlich langen Bearbeitungszeiten
- Lose gekoppelte Systeme
  - Z.B. Producer auf DB-Server, Consumer sind MS-Windows-Server mit speziellen Grafikfunktionen

Wozu dbms_lock?

- `dbms_lock` ermöglicht, eigene datenbankweite Sperren zu erzeugen und zu nutzen
- Abbildung von Critical-Sections (Log. Transaktionsklammern)
- Einige Beispiel-Anwendungsfälle:
  - Zugriff auf Dateisystem
  - Zugriff auf nicht-threadsichere Betriebssystemprozesse
  - Synchronisation von konkurrierenden Oracle-Prozessen

Einsatz dbms_lock

- Ein Lock-Handle erzeugen
- Eine Sperre auf dem Lock-Handle anfordern (Sessionbezogen)
  - Sperranforderung erfolgreich, kein Wartezustand
  - Sperranforderung versetzt Session in Wartezustand
    - Sperranforderung bricht ab, wenn Timeout erreicht
- Erhaltene Sperranforderung wieder freigeben!
  - Kann beim commit automatisch freigegeben werden
- Tabelle `sys.dbms_lock_allocated` listet alle Lock-Handles

Einsatz dbms_lock

```
DECLARE
    v_lockname     CONSTANT VARCHAR2(10) := 'MYLOCK';
    v_lockhandle   VARCHAR2(128) := NULL;
    v_ret          INTEGER := NULL;
BEGIN
    DBMS_LOCK.ALLOCATE_UNIQUE(v_lockname, v_lockhandle);

    v_ret := DBMS_LOCK.REQUEST(v_lockhandle);

    IF (v_ret = 0) THEN
        -- usefull stuff
        NULL;
        -- more usefull stuff
        v_ret := DBMS_LOCK.RELEASE(v_lockhandle);
    ELSE
        RAISE_APPLICATION_ERROR(-20000, 'help me! '||TO_CHAR(v_ret));
    END IF;
END;
/
```

dbms_lock

```
DBMS_LOCK.ALLOCATE_UNIQUE (
   lockname          IN  VARCHAR2,
   lockhandle        OUT VARCHAR2,
   expiration_secs   IN  INTEGER   DEFAULT 864000);

DBMS_LOCK.REQUEST(
   id                  IN  INTEGER ||
   lockhandle          IN  VARCHAR2,
   lockmode            IN  INTEGER DEFAULT X_MODE,
   timeout             IN  INTEGER DEFAULT MAXWAIT,
   release_on_commit   IN  BOOLEAN DEFAULT FALSE)
  RETURN INTEGER;

DBMS_LOCK.RELEASE (
   lockhandle IN VARCHAR2)
  RETURN INTEGER;

DBMS_LOCK.SLEEP (
   seconds  IN NUMBER);
```

Parallelisierung von Außen

- Maßnahmen nur notwendig, wenn Software multi-threaded:
- SQL-Zugriff über mehrere phys. Verbindungen organisieren!
- Einsatz von Connection-Pools ratsam!
- Bei mehreren phys. Verbindungen: bei DMLs auf datenbankseitige Locks achten! Deadlock-Gefahr!

Besuchen Sie uns

- Auf unserem Stand Nr. 212 in der 2. Ebene!
- Und nach dem letzten Vortrag:

  Die Band Replay

Vielen Dank für Ihre Aufmerksamkeit!