CPU, IO und Parallelität verwalten mit dem Database Resource Manager

Reinhold Boettcher
arvato Systems GmbH
Gütersloh

Schlüsselworte

Database Resource Manager, Inter-Database RM, Intra-Database RM, Resource Plans, Directives, Consumer Groups, IORM, Cell server metrics, Exadata

Einleitung

Es werden eine Produktions-, zwei Integrations- und eine Testumgebung auf einer Exadataumgebung betrieben. Aufgabe war es, die Ressourcen nach Instanzen, nach Tageszeiten und Nutzergruppen innerhalb der Datenbanken zu verteilen. Die Ressourcen Memory, CPU, parallele Prozesse und vor allem IO sollten aufgeteilt und auch kontrolliert werden.

Das Monitoring der CPU und Parallelität konnte relativ einfach über Testkonstellationen überprüft werden, beim Thema IO Resource Management war der Nachweis nicht einfach. Durch parallele Ausführung von IO-lastigen SQL-Statements und gleichzeitiger Überwachung der Cell Server Metriken konnte dann an Hand von Zahlen nachgewiesen werden, dass die Exadata auch die IO-Ressourcen nach den eingestellten Vorgaben steuert.Nicht zu unterschätzen ist dabei der Fakt, dass erst bei Volllast des IO-Subsystems die Funktionsweise des IO Resource Managers (kurz IORM) nachweisbar ist..

Eine wichtige Erkenntnis nebenbei: auf einer Exadata habe ich zum ersten Mal einen tiefen Einblick in mein angeschlossenes SANerhalten können. Ähnlich wie bei den v$-Tabellen innerhalb der Datenbank bekommt man eine Fülle von Informationen, nicht nur bezogen auf die Hardware sondern auch für die im Resource Manager (kurz RM) eingestellten Ressourcen bezogen auf Datenbanken und darin enthaltene Consumergruppen.

An dieser Stelle gilt es jetzt, „die Spreu vom Weizen“ zu trennen und die richtigen Zahlen zu kontrollieren.

Ausgangssituation

In dem Projekt wird ein Exadata Halfrack (mit aktuell 2 aktiven Datenbankknoten) mit einem Storage Expansion Rack betrieben. Während zu Beginn Test- und eine Integrationsumgebung in einer Shared-Infrastruktur aufgebaut wurde, sollten dann alle Umgebungen auf der Exadata-Landschaft konsolidiert werden. Das stellte das Projekt vor die Herausforderung, die Ressourcen nach unterschiedlichen Fragestellungen zu verteilen:

- zwischen Datenbanken
- innerhalb einer Datenbank zwischen Applikationen
- innerhalb einer Applikation zwischen Verbrauchern
- innerhalb des Tages mit unterschiedlichen Anforderungen
- ausgedehnt auf das Thema IO!

Zu guter Letzt soll das Ressourcen Management flexibel erweiterbar und auch angepasst werden können.

Zum Zeitpunkt des Aufsetzens des DBRM wurde auf der Datenbankversion 11.2.0.3 BP 14 (Januar 2013) gearbeitet, mittlerweile ist der Juli Patch (BP 20) eingespielt. Auf den Cell Servern läuft Version 11.2.3.2.1. Der Flash Cache wird „nur“ lesend genutzt.

Umsetzung

Im Folgenden wird die Produktionsdatenbank mit P1, die beiden Integrationsdatenbanken mit I1 und I2 sowie die Entwicklungsumgebung mit T1 bezeichnet.

Memory Ressourcen

Zunächst einmal gilt es, den verfügbaren Hauptspeicher zwischen den Umgebungen zu verteilen. An dieser Stelle haben wir dann entschieden, I2 und T1 als Single Instance Umgebung zu fahren, I2 auf Datenbankknoten 1 und T2 auf Knoten 2. Damit haben wir eine Gleichverteilung von 3 Instanzen auf jedem Datenbankknoten.

Wie sind die Memory Ressourcen jetzt verteilt? Zunächst einmal wurde ein Teil für das Betriebssystem und als Reserve reserviert. Der Rest wiederum wurde im Verhältnis 2 : 1 zwischen Produktion und „Rest“ aufgeteilt. Innerhalb einer Instanz wird der Hauptspeicher auf SGA und PGA gleich verteilt. Für die SGA ist jeweils ein Bereich von „huge pages“ auf Betriebssystemebene definiert.

Hinweis: bitte vorsichtig mit den Memoryressourcen planen, insbesondere bei Instanzen mit DWH-Applikationen. Durch Parallelisierung kann der Wert für PGA_AGGREGATE_TARGET schon einmal überschritten werden. Ein ins Swapping geratener DB-Knoten wird dann vom Cluster automatisch neu gestartet. Auch ein Totalausfall einer Exadata ist durch ein zu extensives Verteilen der Memoryressourcen eingetreten.

CPU-Ressourcen

Alle dynamisch zuweisbaren Ressourcen (CPU, IO) sollen für die niedriger priorisierten Umgebungen möglichst nicht hart limitiert werden, sondern nur bei gleichzeitiger Anforderung von höher priorisierten Umgebungen. Ausgenommen hiervon ist die Zahl der maximal nutzbaren CPUs: hier soll die Produktion die maximale Anzahl CPUs ziehen können, die anderen Umgebungen dürfen nur eine Teilmenge der CPUs belegen, selbst dann, wenn die CPUs von Produktion nicht gebraucht werden (Instance Caging). Durch Instance Caging wird zunächst eine Priorisierung zwischen Datenbanken vorgenommen; Dies wird durch Setzen des Initialisierungsparameter CPU_COUN T pro Datenbank erreicht.

ALTER SYSTEM SET cpu_count=<zahl>;

Man kann zwischen zwei Ansätzen wählen, die Anzahl CPU genau auf die Datenbanken verteilen (ressourcenkritische DBs), oder durch Mehrfachverteilung eine bessere Ressourcenausnutzung zu erreichen.

Die CPU-Nutzung innerhalb einer Datenbank wird dann über Consumergruppen und Kategorien gesteuert.

Consumergruppen

Zu Beginn des Projektes wurde entschieden, alle Applikationsteile auf einer Datenbank zu betreiben. Nebem dem Data Warehouse und einer CRM-Applikation (OLTP) läuft ein Kampagnenmanagement in diesem Umfeld. Mit diesem Hintergrund gibt es im Wesentlichen 6 Anwendergruppen:

- SYS_GROUP: Benutzer zur Administration der Umgebung (wie OTHER_GROUP, muss auch nicht angelegt werden)
- OLTP: Anwender der CRM-Applikation
- DWH: Betrieb aller Data Warehouse Funktionalitäten
- KMGT: Benutzer des Kampagnemanagements
- BI: Anwender des Data Warehouse
- EXTERNAL: Zugriffe von Extern
- OTHER_GROUPS: wird explizit vom Database Resource Manager (kurz DBRM) bereitgestellt, muss nicht angelegt werden.

Diese Gruppen werden dem DBRM per "dbms_resource_manager.create_consumer_group“ bekannt gemacht.

Pläne und Direktiven

Seitens der Applikationen gibt es die Anforderung, dass tagsüber die OLTP-Anwendung sowie das BI-Geschäft Vorrang haben muss. Nachts müssen die Batchprozesse mit Priorität behandelt werden, dem OLTP-Geschäft muss dabei allerdings weiterhin einen gewissenProzentsatz an Ressourcen gewährt werden. Gleichzeitig gilt es, der Datenbankwartung (Statistikberechnung) Ressourcen einzuräumen.

Dazu wurde im Wesentlichen ein Tages- und Nachtplan wie folgt definiert:

```
exec dbms_resource_manager.create_plan( plan => 'DAYTIME', comment =>
'Plan fuer Geschaeftszeit von 8 bis 20 Uhr');

exec dbms_resource_manager.create_plan_directive( plan => 'DAYTIME' ,
group_or_subplan => 'SYS_GROUP' , mgmt_p1 => 30 , mgmt_p2 => 10 , mgmt_p3
=> 5);

exec dbms_resource_manager.create_plan_directive( plan => 'DAYTIME' ,
group_or_subplan => 'OLTP' , mgmt_p2 => 40 , mgmt_p3 => 35 ,
parallel_degree_limit_p1 => 12);
```

…

Für jede Consumergruppe können dann für jeden Plan folgende Ressourcen eingeteilt werden:CPU-Verbrauch auf bis zu 8 Ebenen, maximale Anzahl aktiver Verbindungen, maximale Parallelität, Anzahl UNDO Informationen in KB sowie maximale Wartezeit auf nicht aktive blockierende Verbindungen. In diesem Projekt haben wir folgende Ressourcen in den Plänen verteilt: CPU (auf 3 Ebenen), maximale Anzahl paralleler Prozesse und möglicher maximaler Prozentsatz der zu Verfügung stehenden parallelen Prozesse definiert.

```
dbms_resource_manager.create_plan_directive
        ( plan => 'TAGESPLAN' ,
        group_or_subplan => 'EXTERNAL' ,
        mgmt_p3 => 5 ,
        parallel_degree_limit_p1 => 12 ,
```

parallel_target_percentage => 5);

Zum Abschluss werden alle Datenbankbenutzer einer Consumergruppe zugewiesen und aktiviert:

```
dbms_resource_manager.set_consumer_group_mapping
  (attribute => dbms_resource_manager.oracle_user , value =>  'DWH _USER' ,
     consumer_group => DWH');
dbms_resource_manager_privs_PRIVS.grant_switch_consumer_group
    ( grantee_name   => 'DWH_USER' ,   consumer_group =>'DWH' ,   grant_option   => FALSE);
```

Zur Steuerung der Parallelität wird für einige Benutzer per Logon-Trigger DOP auf „Auto“ gesetzt:

```
ALTER SESSION ENABLE PARALLEL QUERY;
ALTER SESSION SET PARALLEL_DEGREE_POLICY=AUTO
```

Mit dieser Konfiguration lassen sich CPU-Verbrauch und auch Parallelität auf Datenbankebene korrekt steuern. Auch Änderungen sind jederzeit dynamisch möglich. Zwischen den einzelnen Plänen wird automatisch durch den Datenbank-Scheduler gewechselt.

IO Ressourcen Management

Auf der Ebene der IO-Ressourcen können dann Regeln zwischen Datenbanken (dbPlan, Inter Database IORM) oder in Kombination mit dem Resource Manager innerhalb einer Datenbank (catPlan, Intra Database Plan) definiert werden. Definierte Grenzen aber greifen erst dann innerhalb des IORM, wenn die Ressourcen auf Cellserver Ebene unter Last geraten. Dass das IORM erst unter hoher Last aktiv wird, hatte uns zu der Vermutung veranlasst, dass das IORM nicht greift. In dem Projekt haben im ersten Schritt die IO-Ressourcen nur auf Ebene 1 zwischen den Datenbanken verteilt und ganz aus Komplexitätsgründen erst einmal auf Pläne innerhalb von Datenbanken (also catplan) verzichtet.

```
alter iormplan -
                dbplan=( (name=P1, level=1,allocation=68), -
                (name=I1, level=1, allocation=16), -
                (name=I2,level=1,allocation=8), -
                (name=T1,level=1,allocation=6), -
                (name=other,level=1,allocation=2))
```

Zum Plan muss jetzt eine Directive zur Steuerung gewählt und der Plan aktiviert werden:

```
alter iormplan objective='balanced'
alter iormplan active
```

Die OBJECTIVE Option kann dabei auf BASIC, AUTO, LOW_LATENCY, BALANCED oder HIGH_THROUGHPUT eingestellt werden. In unserem Fall stellte BALANCED die beste Option dar:

- low_latency: präferiert OLTP-Applikationen
- high_throuput: geeignet für DSS-Applikationen
- balanced: Mischung aus beiden
- auto: der IORM justiert sich automatisch

Eine genaue Defintion der einzelnen Optionen findet sich im Handbuch im Abschnitt zu ALTER IORMPLAN.Diese Kommandokette muss auf allen Cellservern ausgeführt werden, z.B. durch:

dcli -g mycells -x activate_iorm.scl

Wie kann man den IORM wieder deaktivieren? Dies ist ähnlich den DBRM-Plänen auf Datenbankebene:

```
alter iormplan dbplan ='', catplan=''
alter iormplan objective='off'
alter iormplan inactive
```

Diese Kommandokette muss wiederum auf allen Cellservern ausgeführt werden.

Monitoring IORM

Der Oracle Enterprise Manager Cloud Control 12c (kurz OEM) und dem Exadata Plugin bietet im Monitoring der Exadata, auch bezüglich IORM, eine Menge Möglichkeiten. Über „Exadata Grid“ -> Administration -> Manage IO Resource kann man sich den eingestellten Plan anschauen und verändern. Außerdem werden alle Ressourcenverbräuche für bestimmte Zeitfenster angezeigt. Die Ergebnisse zeigt der OEM sowohl summiert für alle Cell Server als auch für jeden einzelnen an.

Nach den Einstellungen aus Abschnitt IO Ressource Management ist der IORM auf allen Cell Servern aktiviert sein. Wie kann ich es auf Kommandoebene kontrollieren?

dcli –g cell_group –x list_status.scl #:LIST IORMPLAN DETAIL

mit folgender Ausgabe:

```
cellserver1: name:cellserver1_IORMPLAN
cellserver1: catPlan:
cellserver1: dbPlan:name=P1,level=1,allocation=68
cellserver1: name=I1,level=1,allocation=16
cellserver1: name=I2,level=1,allocation=8
cellserver1: name=T1,level=1,allocation=6
cellserver1: name=other,level=1,allocation=2
cellserver1: objective: balanced
cellserver1: status: active
```

Eine mögliche Ursache, dass der IORM-Plan nicht aktiv ist, ist die fehlende Aktivierung des Resource Managers auf allen Instanzen. Wie teste und verifiziert man die Funktionalität des IORM? In unserer Archtitektur fahren wir insgesamt 11 Cell Server, 7 davon mit HP-Platten und 4 mit HC-Platten. Auf den HC-Platten liegt eine Diskgruppe, in der wir auf P1 und I1 jeweils einen Tablespace generiert haben. Parallel lassen wir jetzt auf P1 und I1 jeweils SQL-Befehle mit „create table … as select …“, die auf der einen Seite große Datenmengen lesen, auf der anderen Seite aber auch hohe Schreiblasten erzeugen.

Die ersten Tests mit der Messung der reinen Laufzeit ließen uns zu der Behauptung hinreißen, dass der IORM nicht funktioniere. Laufzeiten auf P1 und I1 stiegen mehr oder weniger gleichmäßig stark an. Schnell war klar, aus diesen Laufzeiten selbst kann man nicht unbedingt Schlüsse auf den IORM ableiten. Welche Zahlen aber helfen, wie bekommt man einen tieferen Einblick in das IO-Subsystem?

Einen ersten Ansatz liefert das Skript **metric_iorm.pl**, zu finden bei dem Oraclesupport (siehe Informationen). Es liefert Metriken des IORM sortiert nach Datenbanken und Consumergruppen:

```
cellserver1: Time: 2013-06-05T09:06:53+02:00
cellserver1: Database: P1
cellserver1: Utilization:     Small=0%    Large=66%
cellserver1: Flash Cache:     IOPS=29.9
cellserver1: Disk Throughput: MBPS=0
cellserver1: Small I/O's:     IOPS=9.2    Avg qtime=0.0ms
cellserver1: Large I/O's:     IOPS=1248    Avg qtime=77.8ms
cellserver1: Consumer Group: OLTP
cellserver1: Utilization:     Small=0%    Large=0%
cellserver1: Flash Cache:     IOPS=28.4
cellserver1: Disk Throughput: MBPS=0
cellserver1: Small I/O's:     IOPS=2.3    Avg qtime=0.0ms
cellserver1: Large I/O's:     IOPS=0.0    Avg qtime=0.0ms

…
cellserver1: CELL METRICS SUMMARY
cellserver1:
cellserver1: Cell Total Utilization:     Small=0%    Large=22%
cellserver1: Cell Total Flash Cache:     IOPS=19.6
cellserver1: Cell Total Disk Throughput: MBPS=427.411
cellserver1: Cell Total Small I/O's:     IOPS=14.5
cellserver1: Cell Total Large I/O's:     IOPS=427.1
cellserver1:
cellserver1: Cell Avg small read latency:  14.74 ms
cellserver1: Cell Avg small write latency: 0.54 ms
cellserver1: Cell Avg large read latency:  66.38 ms
cellserver1: Cell Avg large write latency: 0.02 ms
```

Für den ersten Ansatz hat man Zahlen, wir konnten an Hand dieser Zahlen auch die Funktionsweise des IORM nachweisen. Aber dieses Skript erzeugt eine feste Ausgabe, über die man eigentlich nur manuell eine Auswertung durchführen kann.

Der Oracle Exadata Storage Server Software User's Guide beschreibt in Kapitel 7 das Monitoring und Tuning der Exadata Storage Server. Die auf den Cellservern gesammelten Daten lassen sich mit den v$-Tabellen auf Datenbankebene vergleichen, es wird eine Menge an Performancezahlen wie „MB read in large/small blocks, MB written in large/small blocks, number of requests to read large/small blocks, average service times, …" und die auf folgenden Ebenen:

Cells: CL_*
Cell disks: CD_*
Consumer groups: CG_*
Categories: CT_*
Databases: DB_*
Flash cache: FC_*

Flash Log: FL_*
Grid disks: GD_*
Network: N_*

Wie kann man die gewünschten Informationen lesen und möglicherweise weiter auswerten? Gelesen werden die Daten mit dem Befehl „list metrichistory“, der wie ein SQL-Befehl eingeschränkt werden kann, z.B.

list metrichistory where name=${mymet} and collectionTime \> \'${myfrom} \' and collectiontime \< \'${myto}\'" \,

wobei über in diesem Shellaufruf über MYMET die Metrik und über MYFROM und MYTO der Zeitraum eingeschränkt wird.

Die Ausgabe wird umgeleitet in eine Datei auf dem Datenbankknoten, die Ausgabe textuell ein wenig aufbereitet und dann per External Table in die Datenbank eingelesen und dort ausgewertet.

Jetzt gilt es, die Spreu vom Weizen zu trennen. Welche Parameter sind aussagekräftig für welche Applikation. Während DWH-lastige Applikationen sicherlich mehr große Blöcke lesen und auch schreiben, nutzen OLTP-Transaktionen mehr den Flash Cache. Greift man an dieser Stelle ein und verlagert Tabellen in den Flash Cache? Wie kann man aus all diesen Zahlen Performanceaussagen bezüglich IO gewinnen und daraus Maßnahmen ableiten?

Fazit/Ausblick

Das Ressourcenmanagement bietet eine umfangreiche Funktionaltität zur Verwaltung der Ressourcen wie CPU und Parallelität - auf einer Exadata sogar auch des IOs. Die Implementierung geschieht über mitgelieferte Packages. Aber ein Nachweis bzw. ein aussagekräftiges Monitoring des IOs ist einfach und muss erarbeitet werden. Aus meiner Sicht ist aber die Nutzung des DBRM inkl. IORM auf einer Exadata, die als Kondolidierungsmaschine genutzt wird, ein MUSS.

Was bieten jetzt die Informationen des IO-Subsystems in Richtung Tuning von Applikationen? Welche Parameter sind in welcher Richtung auf der Exadata aussagekräftig? Dies gilt es jetzt im Rahmen von Applikationsstests bzw. IORM-Tests noch herauszufinden.

Zum Abschluss sollen noch ein paar Neuigkeiten aus dem aktuellen Oracle Database 12c Release erwähnt werden. Ein wichtiges Ziel war die Multitenant Architektur zu unterstützen. Aus diesem Grund gibt es neuerdings die Möglichkeit, Ressource Pläne auf Container DB (auch CDB) und auf PDB Ebene einzustellen. CDB Ressource Pläne vergeben dabei ihre gesamten Ressourcen für die einzelnen PDBs über sogenannte Shares. Innerhalb der PDBs können wiederum Ressourcen pro Consumer Groups alloziert werden. Bei Ressourcen handelt es sich dabei um CPU und Anzahl der Parallel Execution Servers. Unabhängig von der Architektur kann der Database Resource Manager in Oracle Database 12c fein granularer beim SWITCH_GROUP arbeiten – Einstellungen per Elapsed

Time und Logical IO sind nun möglich. Im Bereich Parallel Statement Queuing ist es jetzt auch möglich Statement von gewissen Consumer Gruppen beim Queuing auszunehmen. Möchte man sich detailliert über diese Neuerungen informieren, kann man die Beschreibung der Prozedur CREATE_PLAN_DIRECTIVE im PL/SQL Packages and Type References Handbuch heranziehen. Bleibt abzuwarten, was in den nächsten Releases an Neuerungen implementiert wird. Ein Memory Ressource Management pro CDB beispielsweise wäre wünschenswert. Fakt ist, dass der Resource Manager ein wichtiger unverzichtbarer Bestandteil der Datenbank ist und bleibt,und in jedem Release erweitert wird.

Informationen

Bücher/White Paper

- Oracle Exadata Storage Server Software User's Guide (Kapitel 6)
- Oracle Database PL/SQL Packages and Types Reference (Package DBMS_RESOURCE_MANAGER)
- Oracle Database Administrator's Guide (Kapitel 27)

- Skripte aus Oracle Exadata Recipes (John Clarke)
- Effective Resource Management Using Oracle Database Resource Manager (white paper june 2011)

MOS Notes:

- Tool for Gathering I/O Resource Manager Metrics: metric_iorm.pl (Doc ID 1337265.1)
- Configuring Resource Manager for Mixed Workloads in a Database (Doc ID 1358709.1)
- Master Note for Oracle Database Resource Manager (Doc ID 1339769.1)

Kontaktadresse:

Reinhold Boettcher
System Architect Infrastructure
---
arvato systems GmbH
An der Autobahn 18
33311 Gütersloh

Reinhold.Boettcher@bertelsmann.de
Tel.: +49 (0) 52 41 / 80-3571
Fax.: +49 (0) 52 41 / 80-63571