Database Resource Manager im praktischen Einsatz

Thomas Riedel PAYBACK GmbH,
Ulrike Schwinn Oracle Deutschland B.V. & Co.KG

Schlüsselwörter: Database Resource Manager, DBMS_RESOURCE_MANAGER, gv$rsrcmgrmetric_history

Einleitung

2009 fand bei der Payback GmbH eine Migration des bestehenden Payback-DWHs auf eine Exadata V1 statt. Im Zuge dieser Migration wurden alle Prozesse, die auf dem DWH ausgeführt wurden, an DB-Services gebunden. Es wurden Services implementiert, welche die Möglichkeit eines Loadbalancing über das komplette RAC bieten, aber auch Services, die nur auf einem dedizierten Knoten aktiv sind. Es war somit möglich eine gewisse "Ressourcenverteilung" durchzuführen.

Schon recht schnell nach der Migration fiel einigen Benutzern auf, dass ihre Ad hoc-Queries schneller laufen, wenn parallelisiert wird. Diese Information machte die Runde und auch kleine Prozesse, die bisher seriell gestartet wurden, liefen plötzlich mit hohem Parallelisierungsgrad. In Folge dieser "Ressourcen-Vereinnahmung" fuhren businesskritische Prozesse entweder mit "angezogener Bremse" oder brachen komplett ab. Nach Analyse und Eskalation durch die DBAs wurde festgeschrieben, dass bestimmte Benutzergruppen und deren Prozesse nur noch eine bestimmte Anzahl an parallelen Prozessen in Verwendung haben dürfen. Aber auch ein ORDER BY macht aus PARALLEL 4 schnell mal einen PARALLEL 8.

Wie kann nun sichergestellt werden, dass bestimmte Prozesse bzw. Benutzergruppen nur die Ressourcen allokieren, die ihnen ggf. vertraglich zugesichert/zugestanden werden? Eine Möglichkeit wäre der Einsatz von User Profiles. Mit User Profiles können nicht nur Passwörter sondern auch Systemressourcen wie CPU_PER_SESSION usw. eingestellt werden. Allerdings gibt es nur einige wenige Einstellungsvarianten und diese gelten pro User ohne Betrachtung des Gesamtsystems. Daher lieferte der Oracle Database Resource Manager die passende Antwort auf die Fragestellung. Als dann auch noch in einem weiteren Projekt beobachtet werden konnte, dass sich im OLTP Betrieb verschiedene Module der Anwendung gegenseitig Ressourcen streitig machten, war das Projekt klar: Die Payback GmbH benötigt ein globales Resource-Manager-Konzept!

Auf Basis der oben erwähnten Performance-Problemanalyse wurde entschieden, die Ressourcen-Allokierung auf CPU und Parallelitätsgrad zu beschränken. Die weiteren Möglichkeiten, wie UNDO-Allocation oder Einschränkung der maximalen Execution Time werden ggf. zu einem späteren Zeitpunkt eingeführt, da die Planung und Implementierung der aktuellen Ressourcen-Allokierung schon sehr viel Zeit in Anspruch nimmt. Bevor die Beschreibung am Beispiel in diesem Beitrag erläutert wird, soll eine kleine Einführung den Zugang in die Thematik erleichtern.

Der Datenbank Resource Manager – Eine Einführung

Ressourcen in der Datenbank mit dem Database Resource Manager zu verwalten, ist schon seit der Version 8*i* mit der Enterprise Edition der Datenbank möglich. Zu Beginn war diese Funktion der Datenbank noch wenig bekannt und fand nur spärlichen Einsatz. Gründe dafür lagen sicherlich in der

eingeschränkten Verwendbarkeit, die ausschließlich über PL/SQL Packages und für dedizierte Datenbankuser-Sessions möglich war.

Spätestens mit der Oracle Database Version 10*g* und der Verfügbarkeit über die graphische Oberfläche des Enterprise Managers und die Erweiterung der Nutzung über Services, Module, Programme usw. sind diese Gründe aufgehoben. Dabei soll unabhängig von der Datenbanklast gewissen Sessions ein Minimum an Ressourcen garantiert werden. So können zum Beispiel CPU-Ressourcen für verschiedene Applikationen priorisiert werden, langlaufende Operationen unterbunden oder die Anzahl der Sessions begrenzt werden, um nur einige Beispiele zu nennen.

Die folgende Liste gibt einen guten Überblick über alle Ressourcen, die über den Database Resource Manager verwaltet werden können (Stand Datenbank Version 11*g*):

- CPU Nutzung
- Parallelität
- Session-Anzahl
- Undo-Nutzung
- Idle-Zeit
- Execution Zeit
- I/O Limit

Auch intern findet das Ressourcen-Management über den Database Resource Manager schon seit Oracle Database 10*g* seine Anwendung. Beispiele dafür sind die "Automated Maintenance Tasks" in 11*g* oder die Aufgabe "Manage Optimizer Statistics" in 10*g*. Auch neue Features wie Instance Caging, Parallel Query Queuing oder Einführung einer maximalen Ressourcen Begrenzung werden ab 11*g* Release 2 über den Resource Manager implementiert.

Konzeptionierung des globalen Resource Plans bei Payback

Bevor man mit dem Resource Manager beginnen kann, sollten folgende Punkte in die Planung einbezogen werden:

- **Consumer Groups**: in diese Gruppen werden Sessions beim Connect gemapped
- **Consumer Group Mapping**: Connects werden anhand bestimmter Kriterien in bestimmte Gruppen gemapped
- **Plan Directives**: Hier werden Ressourcen an die Gruppen oder Sub-Pläne allokiert (hier: CPU und PARALLEL)
- **Consumer Group Priorities**: Prioritäten, anhand welcher die Mappings durchgeführt werden (z.B. Oracle-Service, Module und Action, Oracle User, etc.)

Voraussetzung für die Planerstellung, war die Nutzung von Services. Das folgende Beispiel zeigt die Aufteilung in Services (Namen anonymisiert):

Prozesse:	DWH-Load	Analysten	DBA	BI-Prozesse
Services:	S_SERVICE1	S_SERVICE2	S_SERVICE3	S_SERVICE4
# Nodes	4 Nodes	1 Node	1 Node	2 Nodes

Um nun an den "globalen Plan" zu gelangen muss "das Pferd von hinten aufgezäumt werden".

1. Ohne Consumer-Group gibt es keinen Plan. Ein Tipp aus der Erfahrung: Hierbei sollte man (bei einem globalen Ansatz) darauf achten sich nicht in zu vielen Details zu verlieren. Anhand einer genaueren Analyse der auf den verschiedenen Projekten befindlichen Prozesse, wurden folgende Gruppen erstellt:

    - High Prio High Load (HPHL): Prozesse, welche businesskritisch sind und mit sehr vielen Ressourcen arbeiten müssen (Parallelität, CPU, I/O).
    - High Prio Low Load (HPLL): Prozesse, welche businesskritisch sind, aber mit weniger Ressourcen und längerer Laufzeit durchgeführt werden können.
    - Low Prio High Load (LPHL): Prozesse, welche nicht businesskritisch sind, jedoch mit mehr Ressourcen laufen müssen, da die Laufzeit sonst zu hoch wird.
    - Low Prio Low Load (LPLL): Prozesse, von privaten Usern, die Ad-Hoc-Queries laufen lassen. Hier erfolgt auch das Monitoring via Enterprise Manager.

2. Im nächsten Schritt erfolgt die Gruppierung der Gruppen in Sub-Plänen. Diese Unterteilung wurde gewählt, um im "Masterplan" die verschiedenen "Level" (siehe Punkt 3) besser nutzen zu können:

    - HIGH_PRIO_PLAN beinhaltet HP%-Groups
    - LOW_PRIO_PLAN beinhaltet LP-%-Groups

3. Danach wird der Masterplan PB_RES_PLAN erstellt. Als das erste Grundgerüst des Masterplans stand, kam die Frage auf, ob der Plan durchgehend oder nur zu bestimmten Zeiten aktiv sein soll. Da der Workload auf den DBs der verschiedenen Projekte nicht in bestimmte Zeitabschnitte unterteilt werden kann, wurde entschieden, den Plan durchgehend aktiv zu halten, um eine dauerhafte Ressourcenzuteilung zu gewährleisten. Eine Folge dieser Entscheidung war, dass in das aktuelle Gerüst der Subplan ORA$AUTOTASK_SUB_PLAN und die Gruppe ORA$DIAGNOSTICS eingearbeitet werden mussten. Somit ist sichergestellt, dass die Auto-Maintenance-Task zu ihren bestimmten Laufzeiten (Maintenance-Windows) auch ihre benötigten Ressourcen allokiert bekommt.

Um einen Masterplan zu erstellen, ist die Zuweisung von Ressourcen erforderlich. Der Parallelitätsgrad gibt den Wert an mit dem eine Operation in einer bestimmten Consumer-Group parallel laufen kann. Wird also der Gruppe A ein Parallelitätsgrad von 4 zugewiesen, kann die Operation auch nur mit einem PARALLEL 4 gestartet werden. Erwartet die Operation mehr Parallelität zum Beispiel durch PARALLEL 16, wird die Anzahl automatisch nach unten geregelt (zur Veranschaulichung der Einstellungen bei Payback siehe Schaubild 1 unten). Der Grad der Parallelisierung gilt allerdings NUR für EINE Operation. Werden insgesamt mehrere Operationen gestartet, werden auch mehr parallele Ressourcen verbraucht.

Die CPU-Allokierung wird anhand von Prozent-Zahlen zugewiesen. Die Zuweisung erfolgt auf Gruppen/Subplan-Ebene anhand von LEVELs. Level geben die Möglichkeit ungenutzte Ressourcen weiter zu verteilen. Wenn demnach Gruppe A im Level 1 30% CPU nutzt, sind 70% für Level 2 übrig. In Level 2 bedeuten diese 70% jedoch wiederum 100%. Diese können dann in Level 2 verteilt werden. Wenn in Level 2 von den 100% (entspricht 70% Rest von Level1) nur 60% in Verwendung sind, werden die restlichen 40% an Level 3 weitergereicht und so weiter. Anhand des Schaubildes 1 zeigt sich diese Abstufung. Neu in 11*g* Release 2 ist die Möglichkeit, einer Gruppe/einem Subplan eine

bestimmte Anzahl an Ressourcen zuzuweisen und zusätzlich eine Ressourcen-Beschränkung festzulegen, d.h. darüber können Ressourcen nur bis zu einer bestimmten Grenze - falls verfügbar - allokiert werden. Dies wird mit dem Parameter MAX_UTILIZATION_LIMIT angegeben.

Nachdem die oben genannten Punkte herausgearbeitet waren, konnte der folgende Plan erstellt und verifiziert werden:

Abb. 1: Das Plankonzept

Die praktische Umsetzung

Ist ein Plan in der Theorie erstellt, geht der praktische Part recht leicht von der Hand. Es muss nur eine gewisse Reihenfolge eingehalten werden. Der folgende Ausschnitt aus einem Skript beschreibt die Implementierung des Masterplans PB_RES_PLAN. Das vollständige Skript zum Download bzw. alle weiteren Skripte, die für das Setup notwendig sind finden Sie im letzten Kapitel „Weitere Informationen“. Da eine Oracle 11*g* Release 2 Datenbank verwendet wird, kann auch das Feature der maximalen Ressource Begrenzung über den Parameter MAX_UTILIZATION_LIMIT genutzt werden.

```
BEGIN
  dbms_resource_manager.clear_pending_area();
```

```
  dbms_resource_manager.create_pending_area();
  dbms_resource_manager.create_plan(
              plan    => 'PB_RES_PLAN',
              comment => 'Default Resourceplan for PB-Projects');
  dbms_resource_manager.create_plan_directive(
              plan             => 'PB_RES_PLAN',
              group_or_subplan => 'SYS_GROUP',
              comment          => 'System Group',
              MGMT_P1          => 90,
              max_utilization_limit => 95);
  dbms_resource_manager.create_plan_directive(
              plan             => 'PB_RES_PLAN',
              group_or_subplan => 'HIGH_PRIO_PLAN',
              comment          => 'HighPrio Subplan',
              MGMT_P2          => 70,
              max_utilization_limit => 75);
  dbms_resource_manager.create_plan_directive(
              plan             => 'PB_RES_PLAN',
              group_or_subplan => 'ORA$AUTOTASK_SUB_PLAN',
              comment          => 'Subplan for Autotasks',
              MGMT_P2          => 25,
              max_utilization_limit => 35);
  dbms_resource_manager.create_plan_directive(
              plan             => 'PB_RES_PLAN',
              group_or_subplan => 'ORA$DIAGNOSTICS',
              comment          => 'Subplan for Autotasks',
              MGMT_P2          => 5,
              max_utilization_limit => 15);
  dbms_resource_manager.create_plan_directive(
              plan             => 'PB_RES_PLAN',
              group_or_subplan => 'LOW_PRIO_PLAN',
              comment          => 'LowPrio Subplan',
              MGMT_P3          => 30,
              max_utilization_limit => 50);
  dbms_resource_manager.create_plan_directive(
              plan             => 'PB_RES_PLAN',
              group_or_subplan => 'OTHER_GROUPS',
              comment          => 'Other Groups',
              MGMT_P4          => 10,
              parallel_degree_limit_p1 => 0,
              max_utilization_limit => 15);
  dbms_resource_manager.validate_pending_area();
  dbms_resource_manager.submit_pending_area();
END;
/
```

Testdurchführung

Um gewährleisten zu können, dass der Resource Manager auch wirklich funktioniert, wurden einige Testcases durchgeführt. Für das Monitoring wurden spezielle SQL Aufrufe verwendet. Zum Beispiel ist es gut zu wissen, ob eine Instance "CPU-bound" ist - d.h. ob die Instance mehr CPU benötigt als zur Verfügung steht. Hinweise auf dieses Verhalten liefert das Event "resmgr:cpu quantum" im AWR oder Informationen aus der View GV$RSRCMGRMETRIC_HISTORY bzw. V$RSRCMGRMETRIC_HISTORY. Mit der View GV$RSRCMGRMETRIC_HISTORY - neu in

11*g* - kann beispielsweise die aktuelle CPU Utilization pro Consumer Group ausgegeben werden. Das Ergebnis bei voller Last vergleicht man dann mit der Verwendung im Resource Manager (z.B. mgmt_p1 etc.).

Folgendes Beispiel zeigt den aktuellen CPU Verbrauch (auch utilization), den Verbrauch (auch consumed) und die Drosselung (auch throttled) pro Consumer Gruppe an (Anzeige pro Minute).

```
select inst_id,
       to_char(begin_time, 'HH:MI') time,
       consumer_group_name,
       60 * (select value from v$osstat
                          where stat_name = 'NUM_CPUS') total,
       60 * (select value from v$parameter where name = 'cpu_count')
       db_total,
       cpu_consumed_time / 1000 consumed,
       cpu_consumed_time /
         (select value from v$parameter where name = 'cpu_count') /
          600 cpu_utilization,
       cpu_wait_time / 1000 throttled
  from gv$rsrcmgrmetric_history
order by begin_time desc, consumer_group_name;
```

Nach der Einführung des Ressource Managers in der Exadata Umgebung bestätigte sich das Ergebnis aus der Testreihe. Der Ressource Manager griff - wie im Plan vorgesehen – regulierend ein, so dass das Ziel, dass bestimmte Prozesse bzw. Benutzergruppen nur die Ressourcen allokieren, die ihnen ggf. vertraglich zugesichert/zugestanden werden, erreicht werden konnte. Folgender Screenshot zeigt einen Performance Ausschnitt aus dem Enterprise Manager, wenn das System unter Last steht.

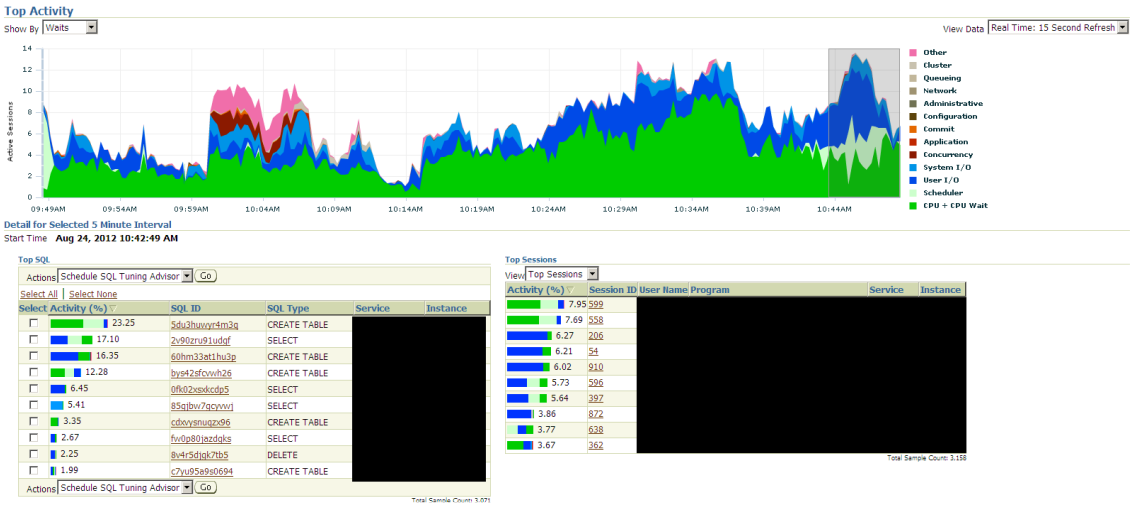

Abb. 2: Einsatz des Ressource Managers (Enterprise Manager Performance Page)

Die Abfrage von oben wird nun in Produktion ausgeführt. Man sieht recht deutlich, dass die Gruppe LPLL – die Gruppe mit geringster Ressourcenzuweisung – am meisten ausgebremst (auch „throttled“) wird.

```
select inst_id,
  to_char(begin_time, 'HH:MI') time,
  consumer_group_name,
60 * (select value from v$osstat where stat_name = 'NUM_CPUS') total,
60 * (select value from v$parameter where name = 'cpu_count') db_total,
  cpu_consumed_time / 1000 consumed,
  cpu_consumed_time /
  (select value from v$parameter where name = 'cpu_count') / 600
  cpu_utilization,
  cpu_wait_time / 1000 throttled
from gv$rsrcmgrmetric_history
order by begin_time desc, consumer_group_name;
```

INST_ID	TIME	CONSUMER_GROUP_NAME	TOTAL	DB_TOTAL	CONSUMED	CPU_UTILIZATION	THROTTLED
2	10:44	HPHL_GROUP	1440	1440	0	0	0
2	10:44	HPLL_GROUP	1440	1440	0	0	0
2	10:44	LPHL_GROUP	1440	1440	0,003	0,000208333333333333	0
2	10:44	LPLL_GROUP	1440	1440	221,139	15,356875	55,183

Um das Monitoring zu erleichtern, lohnt sich hier der Einsatz einer User Defined Metrik (kurz UDM). Interessant ist in unserem Fall zu sehen, wann ein entsprechendes Ausbremsen („Throtteling“) beginnt. So könnte dann eine UDM aussehen.

```
select 'resource_bottleneck_detected',count(*)
  from gv$rsrcmgrmetric_history rmh,
 (select distinct to_char(rh.begin_time, 'HH24:MI') time
  from gv$rsrcmgrmetric_history rh,
     (select max(begin_time) time
          from gv$rsrcmgrmetric_history
          group by inst_id) max_time
      where to_char(rh.begin_time, 'HH24:MI') >=
            to_char(max_time.time - 1 / 24 / 60 * 5, 'HH24:MI')
      group by rh.begin_time
      order by 1) act_time
      where  to_char(rmh.begin_time, 'HH24:MI')=act_time.time and
             round(cpu_wait_time / 60000, 2)> <zahl>
```

Fazit

Mit den Testcases konnte aufgezeigt werden, dass das umgesetzte Konzept erfolgreich funktioniert und in der Praxis eingesetzt werden kann. Der Einsatz der Technologie des Database Resource Managers war die richtige Entscheidung für die Lösung der eingangs beschriebenen Problematik.

Um die Technologie richtig einzusetzen, ist eine gewisse Einarbeitung notwendig. Stehen die entsprechenden Skripte für das Setup erst einmal zur Verfügung, ist die Nutzung allerdings recht einfach. Ein Monitoring über weitere Skripte erschien bei den Tests erforderlich um die Funktionsweise richtig einschätzen zu können. Auch der Enterprise Manager bietet ein Monitoring der Resource Manager Aktivitäten an. Dieses deckt leider nicht die volle Funktionalität der aufgeführten SQL Abfragen ab.

Bleibt abzuwarten welche weiteren Features in den nächsten Releases in den Database Resource Manager implementiert werden. Unabhängig davon ist für Payback nach erfolgreichem Einsatz schon eine Erweiterung der Funktionen angedacht.

Weitere Informationen

- Beispielskripte für das Setup
  - https://apex.oracle.com/pls/apex/GERMAN_COMMUNITIES.SHOW_RESOURCE_BY_FNAME?P_TIPP_ID=181&P_FILE_NAME=resouce_syntax.txt
- Testergebnisse und Szenarios
  - https://apex.oracle.com/pls/apex/GERMAN_COMMUNITIES.SHOW_RESOURCE_BY_FNAME?P_TIPP_ID=181&P_FILE_NAME=testcases_resource_manager_pb.docxt.docx
- Skripte für das Monitoring
  - https://apex.oracle.com/pls/apex/GERMAN_COMMUNITIES.SHOW_RESOURCE_BY_FNAME?P_TIPP_ID=181&P_FILE_NAME=resource_monitoring.txt
- Kapitel 27: Managing Resources with Oracle Database Resource Manager
- Using Oracle Database Resource Manager (White Paper) 11g Release 2 (11.2)
- Ressourcen managen mit Database Resource Manager (DBA Community Artikel)

Kontaktadressen

Thomas Riedel
IT-Operations
PAYBACK GmbH
Theresienhöhe 12, 80339 München
Telefon: +49 (0)89 961 609 – 2515
E-Mail: thomas.riedel@payback.net

Ulrike Schwinn
Oracle BU DB – Business Unit Database
ORACLE Deutschland B.V. & Co. KG
Riesstr 25, 80992 München
Telefon: +49 89 1430 1865
E-Mail Ulrike.Schwinn@oracle.com
Internet: http://blogs.oracle.com/dbacommunity_deutsch