Que gagneriez-vous en passant sur Exadata ?

Partie II – Estimez l'efficacité SmartScan pour votre base

Franck Pachot, dbi-services

Dans la première partie, nous avons vu qu'il faut d'abord faire des 'direct-path read' pour que le SmartScan rentre en jeu. Et nous avons vu le mode simulation qui permet d'estimer l'efficacité du SmartScan. Nous allons maintenant entrer dans le détail pour comprendre pourquoi le volume éligible peut être filtré ou non. Toujours en mode simulation. Le but est de savoir, au-delà de la théorie et du marketing, si l'activité de votre base, avec vos applications, vos utilisateurs, et vos données, va gagner en performance en passant sur Exadata.

Au-delà du volume éligible, l'efficacité du SmartScan

Ce n'est pas parce que vous faites beaucoup de 'direct path read' et que vous avez vérifié qu'une bonne partie de vos 'physical read bytes' sont couverts par 'cell simulated physical IO bytes eligible for predicate offload', que ce SmartScan va vous apporter quelque chose. Son efficacité dépend de ce qu'il va filtrer.

Projection offloading

La projection, c'est la sélection des colonnes dont vous avez besoin dans votre clause SELECT. Par exemple, je me mets en mode simulation avec le paramètre vu plus haut, et je lance un 'SELECT *' sans WHERE clause. Voici le plan après exécution (format 'allstats last +predicates') *(listage 1).*

On voit avec '+projection' toutes les colonnes du select. Et voici les statistiques significatives lors de l'exécution *(tableau 1).*

Ok, tout a été sujet au SmartScan, mais rien n'a été filtré. Je n'y ai donc rien gagné.

Même chose maintenant mais avec seulement quelques colonnes : SELECT ID, SEQ FROM DEMO2 *(listage 2)*.

Les colonnes sont visible dans la projection de l'opération 'STORAGE' et cette projection est donc passée – via iDB – avec l'appel I/O *(tableau 2)*.

Sur le volume éligible, seulement 23% a été retourné par le SmartScan. Les colonnes autres que ID et SEQ ont été éli-

Statistic	Total
cell simulated physical IO bytes eligible for predicate offload	2,340,585,472
cell simulated physical IO bytes returned by predicate offload	2,121,558,080
physical reads bytes	2,346,206,208
physical reads direct	*285,716*

Tableau 1.

```
-------------------------------------------------------------------------------------------------------
| Id | Operation                 | Name  | Starts | E-Rows     | A-Rows  | Buffers | Reads  |
-------------------------------------------------------------------------------------------------------
|  0 | SELECT STATEMENT          |       |      1 |  2000K     | 2000K   |   285K  |  285K  |
|  1 | TABLE ACCESS STORAGE FULL | DEMO2 |      1 |  2000K     | 2000K   |   285K  |  285K  |
-------------------------------------------------------------------------------------------------------

Column Projection Information (identified by operation id):
-----------------------------------------------------------
   1 - "ID"[NUMBER,22], "DEMO2"."SEQ"[NUMBER,22], "DEMO2"."D"[NUMBER,22],
       "DEMO2"."E"[NUMBER,22], "DEMO2"."F"[VARCHAR2,1000], "DEMO2"."G"[VARCHAR2,4000]
```

Listage 1.

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name  | Starts  | E-Rows     | A-Rows  | Buffers | Reads   |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |       |      1  |   2000K    |  2000K  |    285K |   285K  |
|   1 |  TABLE ACCESS STORAGE FULL | DEMO2 |      1  |   2000K    |  2000K  |    285K |   285K  |
-------------------------------------------------------------------------------------------------------

Column Projection Information (identified by operation id):
-----------------------------------------------------------
   1 - "ID"[NUMBER,22], "SEQ"[NUMBER,22]
```

Listage 2.

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name  | Starts  | E-Rows     | A-Rows  | Buffers | Reads   |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |       |      1  |   1000K    |  1000K  |    285K |   285K  |
|*  1 | TABLE ACCESS STORAGE FULL  | DEMO2 |      1  |   1000K    |  1000K  |    285K |   285K  |
-------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - storage(MOD("ID",2)=0)
       filter(MOD("ID",2)=0)

Column Projection Information (identified by operation id):
-----------------------------------------------------------
   1 - "ID"[NUMBER,22], "SEQ"[NUMBER,22]
```

Listage 3.

Statistic	Total
cell simulated physical IO bytes eligible for predicate offload	2,340,585,472
cell simulated physical IO bytes returned by predicate offload	55,272,816
physical reads bytes	2,341,306,368
physical reads direct	285,716

Tableau 2.

Statistic	Total
cell simulated physical IO bytes eligible for predicate offload	2,340,585,472
cell simulated physical IO bytes returned by predicate offload	27,967,088
physical reads bytes	2,343,059,456
physical reads direct	285,716

Tableau 3.

minées des blocs retournés à la base de donnée en réponse à l'appel I/O.A noter que les statistiques de la simulation s'appellent 'predicate offload' mais incluent en fait tous les 'offload' – predicate et projection. On a la preuve ici.

Predicate offloading

A la requête précédente, je rajoute un prédicat WHERE MOD(ID,2)=0 pour éliminer la moitié des lignes *(listage 3)*.

On voit dans le plan d'exécution que le prédicat peut être traité par le storage, et effectivement le volume retourné par SmartScan est la moitié du précédent *(tableau 3)*.

Join offloading

On ne va pas rentrer dans le détail ici, mais il faut garder à l'esprit que le temps de réponse d'une requête ne vient pas que de la lecture des données. Dans des requêtes complexes, il y a des jointures, des tris, des agrégations, et cela se passe uniquement sur le serveur de base de données, en utilisant les tempfiles, et SmartScan n'apporte rien ici.

Il y a une exception pour la partie filtrage des jointures : lors de la création de la table de hachage (la plus petite des deux tables d'un HASH JOIN) un Bloom filter est créé, qui va permettre d'éliminer beaucoup de lignes en lisant la deuxi-

ème table, sans même à avoir à faire la correspondance avec la table de hachage. Et ce Bloom filter est comme un prédicat : il est exécuté sur le serveur de stockage lorsqu'on fait du SmartScan. On peut le voir comme une très grande clause 'IN()' qui fait un premier filtre grossier.

La figure 3. Montre le fonctionnement : lorsqu'on construit la table de hachage : un filtre représentant cette table, mais pouvant contenir de faux positif, est créé, et sera utilisé par le SmartScan de la deuxième table, ce qui limite le volume à transmettre à la base où il ne reste plus que les faux positifs à éliminer.

Donc si vous ne regardez que les wait events lorsque vous avez des jointures entre deux grosses tables, vous allez voir beaucoup de 'direct path read temp' qui ne sont pas sujets au SmartScan. Mais en allant plus loin avec la librairie de simulation vous verrez que le Bloom filter va peut-être vous éviter une partie de ces lectures/écritures de tempfiles.

Là encore, regardez les statistiques de simulation. Le Bloom filter est censé n'apparaître qu'en Parallel Query jusqu'à la 12.1.0.2 où il est utilisé en Serial pour In-Memory. Mais en vérité il y a d'autres cas où on le voit en Serial.

En bref

Lire quelques colonnes d'une table, ou lire plus de quelques lignes, ou faire des HASH JOIN entre des grosses tables,... le type de base de données où SmartScan montre son avantage est clair : c'est pour du reporting, datawarehouse (ETL et interrogation), requêtes analytiques, DSS,...

Pour un ERP qui va lire toutes les colonnes (SELECT *) de quelques lignes (WHERE PK=...), ce n'est pas SmartScan qui va apporter quelque chose. Je connais une application bancaire qui tout stocke dans des CLOB (XML) et qui n'y accède que par la clé primaire. Oubliez Exadata pour ça.

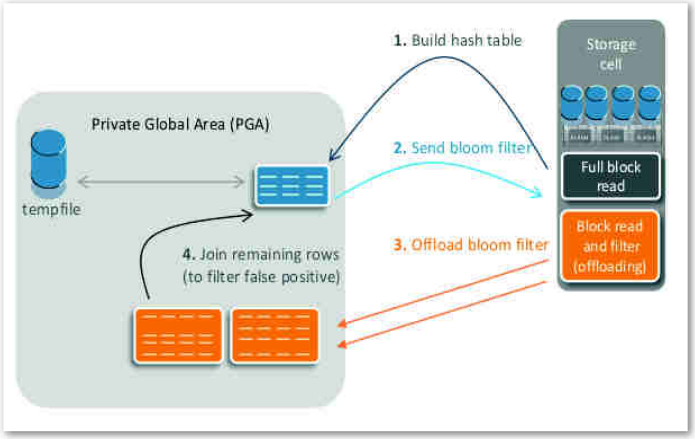

Figure 1: Bloom Filter for Hash Join offloading.

Oubliez In-Memory aussi. Ce n'est pas fait pour ça. Il y a aujourd'hui des bases NoSQL pour ces modèles de données qui ont justement été conçus avant les bases relationnelles SQL.

Sur les ERP ou autres qui définissent les paramètres optimizer_index_caching, optimizer_index_cost_adj pour forcer les accès par index, SmartScan n'apportera rien non plus.

Evaluation du gain de SmartScan

Pour résumer, prenez un rapport AWR et regarder la section Top Events. Le '% DB Time' en face du 'direct path read' vous donnera le gain maximal que vous pouvez espérer.

Quelques exemples *(tableau 4, 5)*. Cette base a une contention sur le buffer cache, mais fait essentiellement des lectures single block. SmartScan n'apportera rien ici. Il faut d'abord comprendre ces lectures. Probablement un plan d'exécution incorrect va lire et relire toujours les mêmes lig-

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
read by other session	46,201,741	158,777	3	*50.2*	*User I/O*
db file sequential read	87,834,244	102,268	1	*32.3*	*User I/O*
CPU time		27,646		*8.7*	
db file scattered read	6,313,301	5,369	1	*1.7*	*User I/O*
enq: TX - row lock contention					

Tableau 4: Top 5 Timed Events avec contention buffer cache.

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
DB CPU		1,685		*49.63*	
direct path read	347,955	926	*3*	*27.26*	*User I/O*
db file sequential read	599,906	271	*0*	*7.99*	*User I/O*
enq: TX - row lock contention	185	89	*481*	*2.62*	*Application*
log file sync	*23,503*	*76*	*3*	*2.24*	*Commit*

Tableau 5: Top 5 Timed Foreground Events avec lectures directes SmartScan.

Statistic	Total	
cell physical IO interconnect bytes	221,656,831,180	206 GB
cell simulated physical IO bytes eligible for predicate offload	336,000,856,474	313 GB
cell simulated physical IO bytes returned by predicate offload	187,871,446,630	175 GB
physical reads bytes	361,291,243,520	336 GB
physical write bytes	1,785,430,016	1.6 GB

Tableau 6: Simulation du offloading.

Event	Waits	Total Wait Time (sec)	Wait Avg(ms)	% DB time	Wait Class
DB CPU		2.2		54.3	
enq: KO - fast object checkpoint	1	1.7	1690.24	42.7	Application
direct path read	1,130	.3	0.28	8.0	User I/O
db file sequential read	535	.2	0.30	4.0	User I/O
SQL*Net more data to client	402	.1	0.12	1.3	Network
enq: RO - fast object reuse	2	0	1.56	.1	Application
SQL*Net message to client	213	0	0.01	.0	Network
reliable message	3	0	0.43	.0	Other
control file sequential read	137	0	0.01	.0	System I/O
log file sync	1	0	0.73	.0	Commit

Tableau 7: Top 10 Foreground Events by Total Wait Time avec modifications.

nes. Vous n'échapperez pas au tuning en achetant une nouvelle machine.

Ici, SmartScan peut adresser 27% du temps de réponse. C'est intéressant d'aller plus loin pour estimer le gain. Et pour aller plus loin, on active la simulation avec _rdbms_internal_fplib_enabled et on compare le volume retourné par SmartScan avec les 'physical read bytes' pour évaluer l'efficacité du filtrage.

Voici les statistiques intéressantes, j'ai rajouté le volume en GB par simplicité *(tableau 6)*.

Que voit-on ici ? On avait 336 GB à lire ('physical reads bytes') et pourtant seulement 206 GB ont été transférés entre la couche stockage (simulée ici par fplib) et la couche base de données.

Pour être plus précis, il faut enlever les écritures, qui sont aussi incluses dans 'cell physical IO interconnect bytes'. Ici, mon exemple n'est pas en ASM redundancy, donc le volume transféré pour écriture est égal à 'physical write bytes'. En normal redundancy, il aurait fallu le multiplier par deux.

Donc, J'ai eu à lire 336GB, parmi lesquels 313GB sont éligibles au SmartScan. 175GB ont été retournés par le SmartScan, et 206-1x1.6-175=29GB ont été retournés sans passer par le SmartScan.

En reprenant le chiffre global des lecture + écriture, je n'ai finalement eu à transférer que 206/(336+1.6)=61% du volume nécessaire.

Etant donné que les I/O faisaient 27.26% et 7.99% du DB time, je peux m'attendre à un gain global de (0.2726+0.0799)*0.61=21.5% sur le DB time en passant sur Exadata. C'est bien, mais ne vous engagez pas au-delà de cela auprès des utilisateurs.

Modifications en cours

La simulation ne sert pas seulement à évaluer le volume filtré. L'efficacité du SmartScan va aussi dépendre des modifications concurrentes. Tout d'abord, on a vu que s'il y a beaucoup de blocs modifiés, le checkpoint serait trop couteux et le SmartScan ne se fera pas. Mais il y a aussi le problème de consistance : on ne doit pas voir les modifications non-commitées, ni celles commitées seulement après le début de notre requête. Et du coup il se peut qu'à un certain moment SmartScan doive renvoyer le bloc complet sans rien filtrer car seul le code de la base de données peut reconstruire une image consistante.

Voici un extrait AWR de ce cas *(tableau 7)*. Vous voyez : seulement 8% du temps est sujet à SmartScan (les 'direct path read'). Et on a passé 42% du temps à faire ce checkpoint justement pour pouvoir faire ces 'direct path read'.

Mais il y a pire. J'ai mis ici toutes les statistiques relatives à 'cell' en mode simulation *(tableau 8)*.

Les statistiques 'cell blocks processed by ...' indiquent le volume traité par chacune des différentes couches du

SmartScan. La figure 4. Les représente à partir d'un 'Flame-Graph' pris lors d'un SmartScan.

J'ai lu 142870 blocs en mode direct. Tous ces blocs ont été traités par le 'cache layer' mais à partir du 'transaction layer' j'en ai moins. SmartScan a vu des blocs dont le SCN est au-delà du SCN de la requête, et ces blocs doivent appliquer du undo pour reconstruire une image consistante. Le storage cell ne peut pas faire cela : il n'a pas accès aux disques des autres, et ne peut donc pas voir tout le undo.

Le SmartScan a donc été bien moins efficace que prévu ici. Et si vous regardez les chiffres, vous comprenez pourquoi j'ai fait des calculs compliqués plus haut pour mesurer l'efficacité. Certains outils (Oracle inclus) font simplement le ratio 'eligible' par rapport à 'returned' mais dans ce cas ils oublient tous les blocs éligibles qui n'ont pas pu être traités par le SmartScan. Ici le ratio paraîtrait bon : la moitié du volume est renvoyé par le SmartScan. Mais en vérité, le volume échangé ('cell physical IO interconnect bytes'), même en enlevant les écritures ('physical write bytes'), est le même que le volume éligible. Il n'y a sur cet exemple aucun gain. Seulement une perte de temps à cause du checkpoint.

SmartScan n'est pas fait pour optimiser l'OLTP pour lequel les modifications sont aussi importantes que les lectures, car il faut alors appliquer aux blocs des fonctions qui ne sont pas disponibles au niveau des serveurs de stockage.

Conclusion

Il est important de comprendre sur quoi le SmartScan peut s'appliquer ou non, car ce sera le facteur principal de l'amélioration des performances sur Exadata par rapport à une autre plateforme.

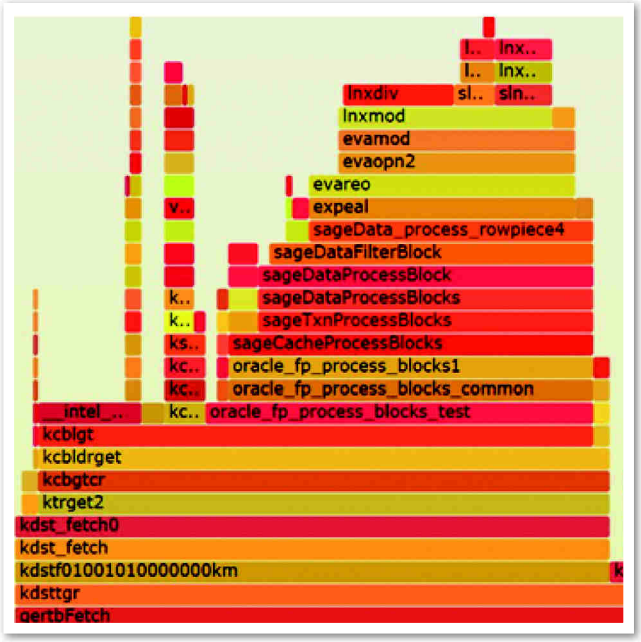

Figure 2: La pile des appels aux fonctions 'sage' (Storage Appliance for Grid Environnements – le nom initial d'Exadata) en mode simulation correspondent aux statistiques du rapport AWR.

D'après ce qu'on voit, Exadata est plutôt fait pour du data-warehouse :

- Full Table Scan ou Index Fast Full Scan
- Parallel Query ou Serial Direct Read
- Peu de mises à jour concurrentes
- Compression

Statistic	Total	per Second	per Trans
cell IO uncompressed bytes	1,170,391,040	213,965,455.21	97,532,586.67
cell blocks helped by minscn optimization	10	1.83	0.83
cell blocks processed by cache layer	142,870	26,118.83	11,905.83
cell blocks processed by data layer	78,128	14,283.00	6,510.67
cell blocks processed by txn layer	78,128	14,283.00	6,510.67
cell commit cache queries	36,182	6,614.63	3,015.17
cell physical IO interconnect bytes	1,215,807,488	222,268,279.34	101,317,290.67
cell scans	5	0.91	0.42
cell simulated physical IO bytes eligible for predicate offload	1,170,391,040	213,965,455.21	97,532,586.67
cell simulated physical IO bytes returned by predicate offload	539,262,144	98,585,401.10	44,938,512.00
physical read bytes	1,174,036,480	214,631,897.62	97,836,373.33
physical reads	143,315	26,200.18	11,942.92
physical reads cache	445	81.35	37.08
physical reads direct	142,870	26,118.83	11,905.83
physical write bytes	23,306,240	4,260,738.57	1,942,186.67

Tableau 8: Simulation avec modifications en cours.

Et je vais plutôt conseiller Exadata lorsqu'on a déjà été au bout de ce qu'on peut faire sur l'infrastructure actuelle. Commencez par maîtriser Parallel Query pour utiliser toutes les CPU licenciées. Eventuellement RAC pour utiliser la puissance de plusieurs serveurs. Optimisez cela, et lorsque vous n'avez plus d'autre contention que la bande passante I/O sur les 'direct path read' alors vous pourrez vraiment passer à la vitesse supérieure avec Exadata. Je connais un client où ça s'est passé exactement comme ça dès l'identification des problèmes de performances.

Sur une charge OLTP, il n'y aura pas d'avantage dû au SmartScan, qui n'améliorera pas :

- Des requêtes 'SELECT *'
- Des accès à peu de lignes via index (par PK pare exemple)
- Des optimisations en FIRST_ROWS
- Des mises à jour concurrentes, utilisant le buffer cache et les undo, mécanismes très efficaces. C'est grâce à eux qu'Oracle a dominé le marché des bases transactionnelles (hors des mainframes).

Bien entendu, vous pouvez choisir Exadata pour accélérer votre datawarehouse, et y mettre aussi vos bases OLTP dans un but de consolidation. Même si le SmartScan ne s'applique pas, vous bénéficiez quand même d'une machine puissante. Il faut par contre faire attention à trois choses pour les charges OLTP sur Exadata :

- Ne pas oublier que la compression HCC est faite pour des tables qui sont principalement en lecture seule. Soyons clairs, à ce jour, aucune compression n'a jamais été efficace pour les modifications.
- Passer en RAC ne veut pas dire qu'il faut ouvrir les services sur tous les nœuds. Travailler sur les mêmes blocs à partir de différentes instances va augmenter toute contention existante.
- N'allez pas supprimer tous vos index. En OLTP les index servent aussi à valider les contraintes d'unicité, à éviter de verrouiller des tables, etc.

Avant de décider votre nouvelle infrastructure, étudiez vos bases actuelles et essayez d'estimer le gain possible. C'est possible d'avoir une première idée en regardant un rapport AWR. Et on peut en savoir plus en simulant SmartScan et en faisant un peu de calcul sur les statistiques correspondantes.

Mais je ne veux pas donner une image négative d'Exadata qui est une machine extraordinaire. Vous pouvez bien sûr y mettre vos bases OLTP aussi, mais ce n'est pas là que vous verrez ses performances exceptionnelles. Je vais faire une comparaison simpliste : vous pouvez faire de la ville avec votre voiture de sport, mais ce n'est pas pour cela que vous l'avez achetée.

Franck Pachot
franck.pachot@dbi-services.com

Franck Pachot est consultant, formateur, et technology leader Oracle à dbi services, Oracle ACE et OCM 11g.

Mentions légales

secrétariat SOUG:
Dornachstrasse 192, 4053 Basel
Tel.: 061 367 93 30, Fax: 061 367 93 31
sekretariat@soug.ch

rédaktion:
Geatano Bisaz
gaetano.bisaz@soug.ch

réalisation / DTP:
DOAG Dienstleistungen GmbH
Tempelhofer Weg 64, 12347 Berlin
office@doag.org

impression:
adame GmbH
www.adame.de

rédaction Newsletter:
nl@soug.ch

inscription aux événements SOUG:
event@soug.ch

questions des membres:
sekretariat@soug.ch

site du web:
www.soug.ch

Titel: © tashatuvango / 123rf.com