ORACLE Server Parameter Tuning (I/O)

오동훈
DB기술팀
㈜한국오라클

Contents

- 데이터베이스 구조와 I/O
- Sorting의 최적화
- Rollback Segment의 튜닝

데이터베이스 구조와 I/O

Oracle 프로세스와 파일

프로세스	오라클 파일			
	데이타	로그	아카이브	컨트롤
CKPT	쓰기			쓰기
DBWRn	쓰기			
LGWR		쓰기		
ARCn		읽기	쓰기	읽기/쓰기
Server	읽기/쓰기			

I/O 성능향상 기본법칙

- Disk I/O를 최소화한다
- I/O를 여러 개 disk 장치와 controller에 분산한다
- 가능하면 Locally Managed Tablespace를 사용한다

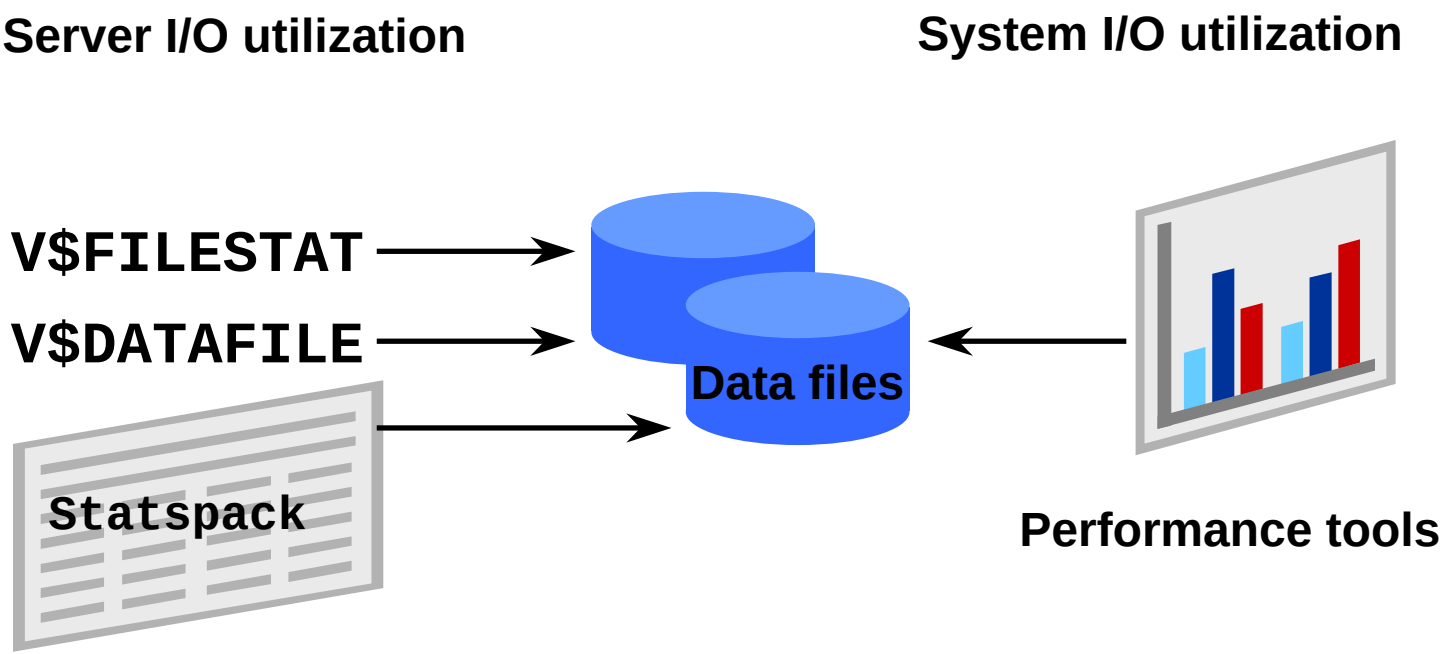
여러 Device에 걸친 File의 분산

- Datafile과 redo log file을 분리한다
- Table data를 striping한다
- Server와 관련없는 I/O를 줄인다

테이블스페이스의 사용법

- ‘SYSTEM’ 테이블스페이스는 Data Dictionary Objects를 위해서만 사용한다
- Space Management 문제를 피하려면 Locally Managed Tablespace를 사용한다
- Table과 Index는 별개의 테이블스페이스에 분리한다
- 롤백세그먼트를 위한 별도의 테이블스페이스를 사용한다
- 매우 큰 object는 그것만을 위한 테이블스페이스를 사용한다
- 하나 이상의 temporary 테이블스페이스를 생성한다

I/O 통계값을 확인하는 진단 Tool



V$FILESTAT의 사용

- 각각의 file에 대해 disk I/O 회수를 확인할 수 있다
- 각 컬럼과 의미
  - File# : File Number
  - Phyrds : physical read의 회수
  - Phywrts : physical write의 회수
  - Phyblkrd : physical read한 block의 개수
  - Phyblkwrt : physical write한 block의 개수
  - Readtim : read하는 데 소요된 시간
  - Writetim : write하는 데 소요된 시간
- 예제
  - ```
    Select phyrds, phywrts, d.name
    from v$datafile d, v$filestat f
    where d.file# = f.file# order by d.name;
    ```
    ```

IO 통계의 예

```
SQL> select d.tablespace_name TABLESPACE_NAME,
  2 > d.file_name, f.phyrds, f.phyblkrd,
  3 > f.readtim, f.phywrts, f.phyblkwrt,
  4 > f.writetim
  5 > from v$filestat f, dba_data_files d
  6 > where f.file# = d.file_id
  7 > order by tablespace_name, file_name;
```

TABLESPACE	FILE_NAME	PHYRDS	PHYBLKRD	READTIM	PHYWRTS	PHYBLKWRT	WRITETIM
UNDO1	/u02/undots01.dbf	26	26	50	257	257	411
USERS	/u03/users01.dbf	65012	416752	38420	564	564	8860
SAMPLE	/u02/sample01.dbf	8	8	0	8	8	0
SYSTEM	/u01/system01.dbf	806	1538	1985	116	116	1721
TEMP	/u04/temp01.dbf	168	666	483	675	675	0
QUERY_DATA	/u01/query_data01.dbf	8	8	0	8	8	0

6 rows selected.

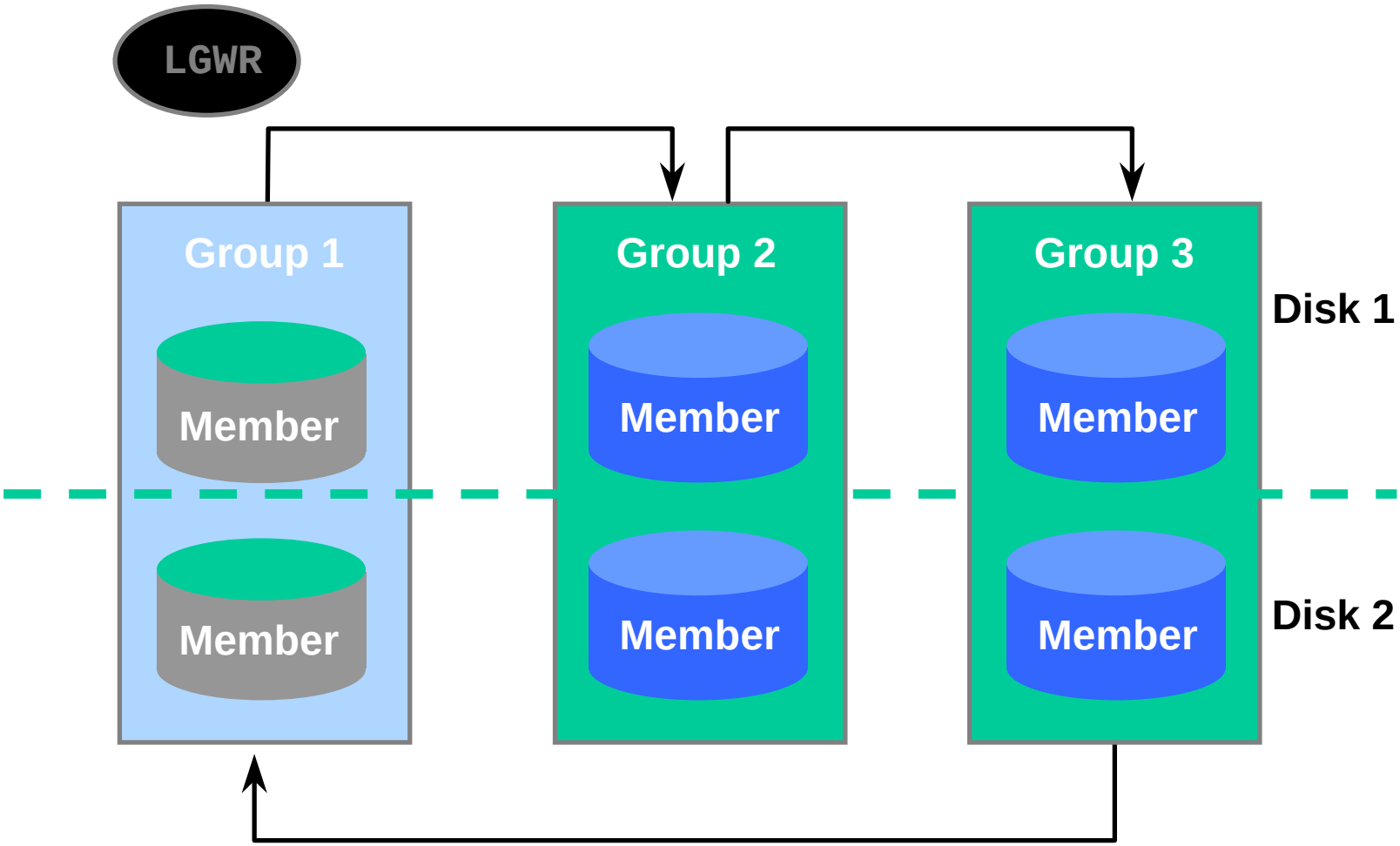
File Striping

- OS Striping
  - OS Striping S/W(예: Logical Volume Manager)
  - RAID
- Manual Striping
  - CREATE TABLE
  - ALTER TABLE ¯ ALLOCATE
- Striping 사용하기
  - Stripe Size = N * DB_FILE_MULTIBLOCK_READ_COUNT * DB_BLOCK_SIZE
  - Parallel Query 기능을 사용하여 FTS가 진행되는 경우 Manual Striping이 의미있다

Full Table Scan의 Tuning

- FTS 찾기
  - SQL> select name, value from v$sysstat
    2 > where name like '%table scan%';
- DB_FILE_MULTIBLOCK_READ_COUNT를 설정한다
  - 서버프로세스가 한번에 읽는 block 개수가 결정된다
  - Cost-based Optimizer가 생성하는 실행계획에 영향을 미친다
- V$SESSION_LONGOPS를 통해 오래도록 수행되는 FTS를 모니터링한다
  - SQL> select sid, serial#, opname
    2 > to_char(start_time,'hh24:mi:ss') as start
    3 > (sofar/totalwork) * 100 as percent_complete
    4 > from v$session_longops;
  - SID SERIAL# OPNAME START PERCENT_COMPLETE
    --- ------- ------------ -------- ---------------
    8 219 TABLE SCAN 13:00:09 48.98098

Redo Log Group과 Member



Online Redo Log Files

- 구성
  - I/O 경합을 최소화 할 수 있도록 크기를 산정한다
  - Log switch가 지연되지 않도록 충분한 개수의 group을 유지한다
  - 하나의 group에 속한 member는 서로 다른 device로 분리한다
- 모니터링
  - 동적 성능 뷰
    - Vlog, vlogfile, v$log_history
    - V$system_event에서 'Logfile Parallel write' wait event를 확인한다
  - OS Utility
    - sar(System Activity Reporter)
    - iostat

Archive Log Files

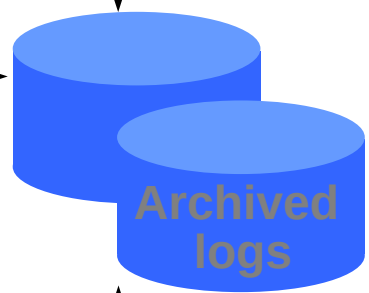
- 구성
  - Archive logfile은 online redo logfile과는 다른 디스크에 둔다
  - ‘Archive log dest’는 full되지 않게 관리한다
- Archiving work을 나누어 같이 수행한다
  - SQL> alter system archive log all
    2 > to ‘<location>’
- Archiving 속도 컨트롤
  - Log_archive_max_processes
  - Log_archive_dest_n

진단 Tools

V$ARCHIVE_DEST

V$ARCHIVED_LOG

V$ARCHIVE_PROCESSES

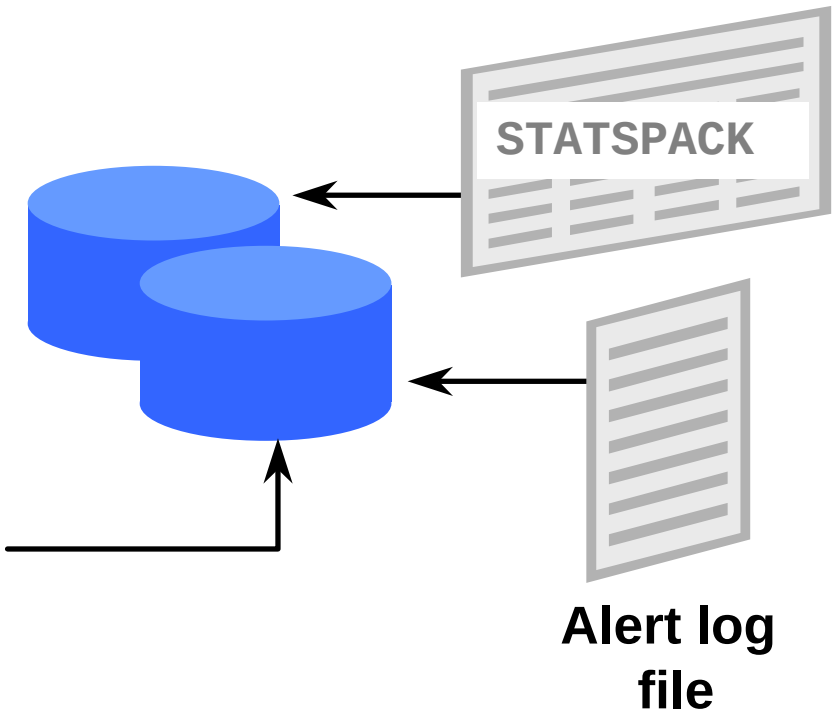


LOG_ARCHIVE_DEST_STATE_*n*

Checkpoints

- Checkpoint가 발생하면
  - DBWn이 I/O를 수행한다
  - CKPT가 controlfile과 datafile header를 갱신한다
- 빈번한 Checkpoint는
  - Instance Recovery 시간을 줄인다
  - 시스템의 운영 시 성능을 떨어뜨린다

Checkpoint 성능 튜닝에 사용되는 진단 Tools



LOG_CHECKPOINT_INTERVAL
LOG_CHECKPOINT_TIMEOUT
FAST_START_IO_TARGET
LOG_CHECKPOINTS_TO_ALERT

Checkpoint 성능 튜닝 Guidelines

- **Checkpoint 성능 튜닝**
  - Checkpoint 회수를 줄일 수 있도록 Online redo logfile의 크기를 정한다
  - Online redo log group의 수를 증가시켜, LGWR가 redo logfile을 덮어쓰기까지의 시간을 늘린다
  - "Checkpoint not complete; unable to allocate file"
    - Log Switch로 인한 checkpoint 회수와 I/O 성능간의 차이
- **Checkpoint의 주기를 조정한다**
  - I/O로 인해 시스템 성능을 크게 저하되지 않는 정도의 주기를 유지한다
  - 필요한 만큼만의 회수로 유지한다
    - Recovery에 소요되는 목표시간으로 판단된다
  - 일반적으로 대략 30분에 1회 정도로 유지한다

FASTSTART Checkpointing의 모니터링

- V$instance_recovery를 통해 모니터링 한다
  - Recovery_estimated_ios
    - Recovery 진행 중에 처리되어야 할 block의 개수
  - Actual_redo_blks
    - 현재 상태에서 Recovery에 필요한 redo block의 개수
  - Target_redo_blks
    - Recovery에 필요한 redo block 개수의 최대값
  - Log_file_size_redo_blks
    - Checkpoint가 완료된 이후에 log switch가 발생함을 보장할 수 있기 위한 redo block의 개수
  - Log_chkpt_timeout_redo_blks
    - Log_checkpoint_timeout이 적용되었을 때, recovery시 처리될 redo block의 개수
  - Log_chkpt_interval_redo_blks
    - Log_checkpoint_interval이 적용되었을 때, recovery시 처리될 redo block의 개수

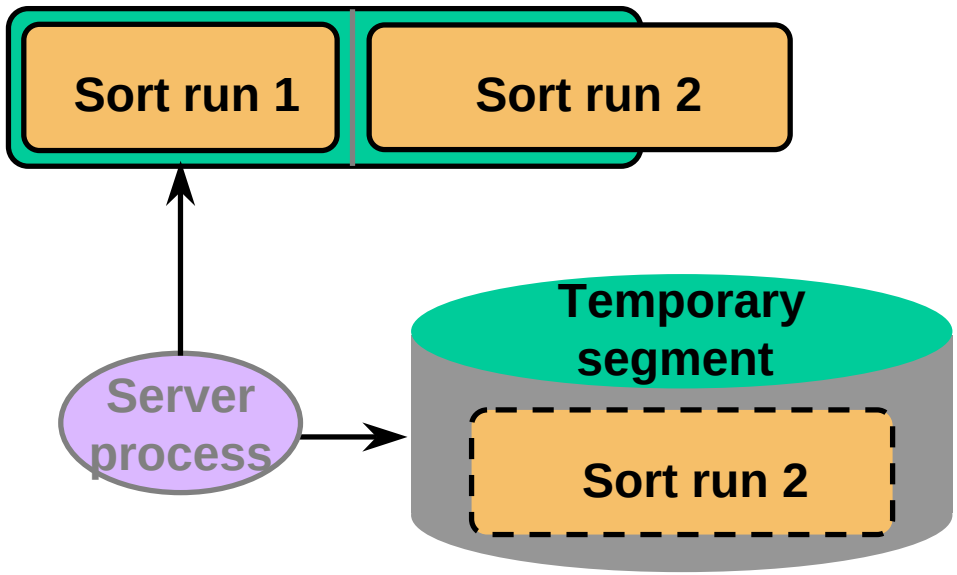
FASTSTART Checkpointing의 모니터링

- Fast_start_io_target_redo_blks
  - Fast_start_io_target을 충족시켰을 때, recovery시에 처리될 redo block의 개수
  - Null
- Target_mttr
  - Fast_start_mttr_target으로 부터 계산된, 목표 MTTR
- Estimated_mttr
  - 현재의 dirty buffer와 log block의 개수로, 예측한 MTTR
  - Fast_start_mttr_target이 설정되지 않으면 값은 0
- CKPT_BLOCK_WRITES
  - Checpoint에 의해 write된 block의 개수

Sorting의 최적화

Sorting의 과정

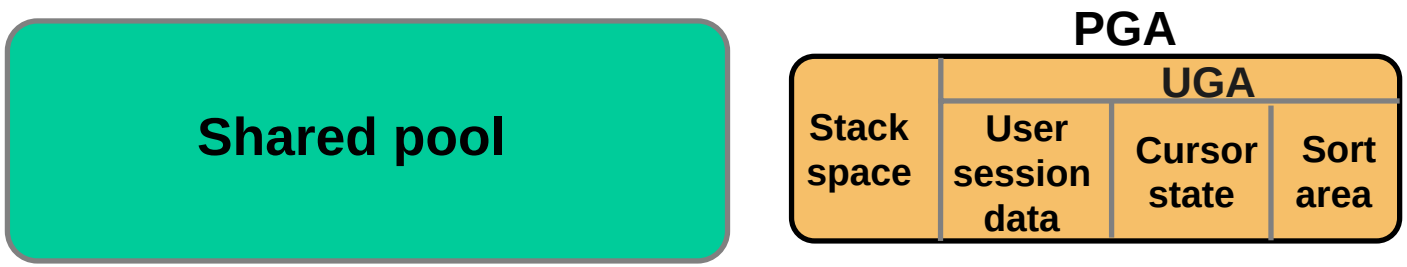
실제 Sorting에 필요한 공간이 sort_area_size보다 크다면:



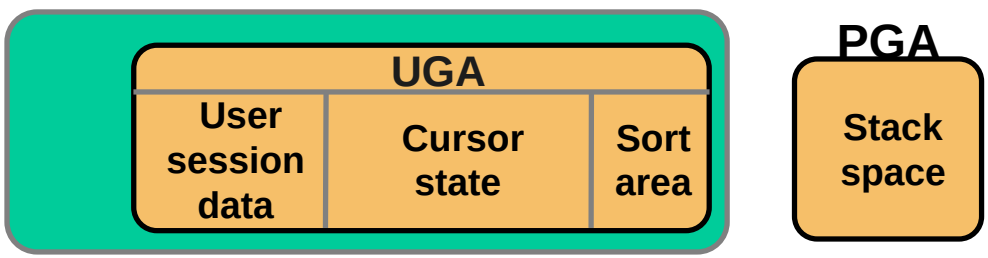
- 데이터가 몇 개의 조각으로 나뉘고(sort run), 각각은 개별적으로 sorting의 대상이 된다
- 메모리에서 Sorting되고 있는 것 이외의 sort run은 temporary segment에 저장된다
- Merge pass를 통해 Merge되어 최종의 정렬된 결과를 생성한다

Sort 영역의 위치

- 전용서버 환경 → PGA



- 공유서버 환경 → Shared Pool



Sort 영역과 초기화 변수

- Sort_area_size
  - 동적 변경 가능 : alter session, alter system deferred
  - 기본 값 : OS dependent
  - 변경 : DSS, batch job 등의 large operation에는 증가시킨다
- Sort_area_retained_size
  - Sorting이 완결된 후, sort area는 그 크기가 sort_area_retained_size로 지정된 크기로 줄어들 수 있다
  - 메모리는 sorting된 row를 더 이상 필요로 하지 않으면, sorting했던 동일 서버 프로세스의 UGA로 반환된다
  - 기본 값 : sort_area_size와 동일

Bitmap Indexing과 초기화 변수

- Create_bitmap_area_size
  - 동적 변경 불가능
  - Bitmap 생성에 할당되는 메모리 크기
  - 기본 값 : 8MB
  - Index 대상 키의 cardinality에 따라 크기 설정을 변경할 수 있다
    예) cardinality = 2 인 경우, 수 KB 만으로도 사용 가능
- Bitmap_merge_area_size
  - 동적 변경 불가능
  - Index의 range scan으로부터 가져온 bitmap을 합하는데 할당되는 메모리 크기
  - 기본 값 : 1MB

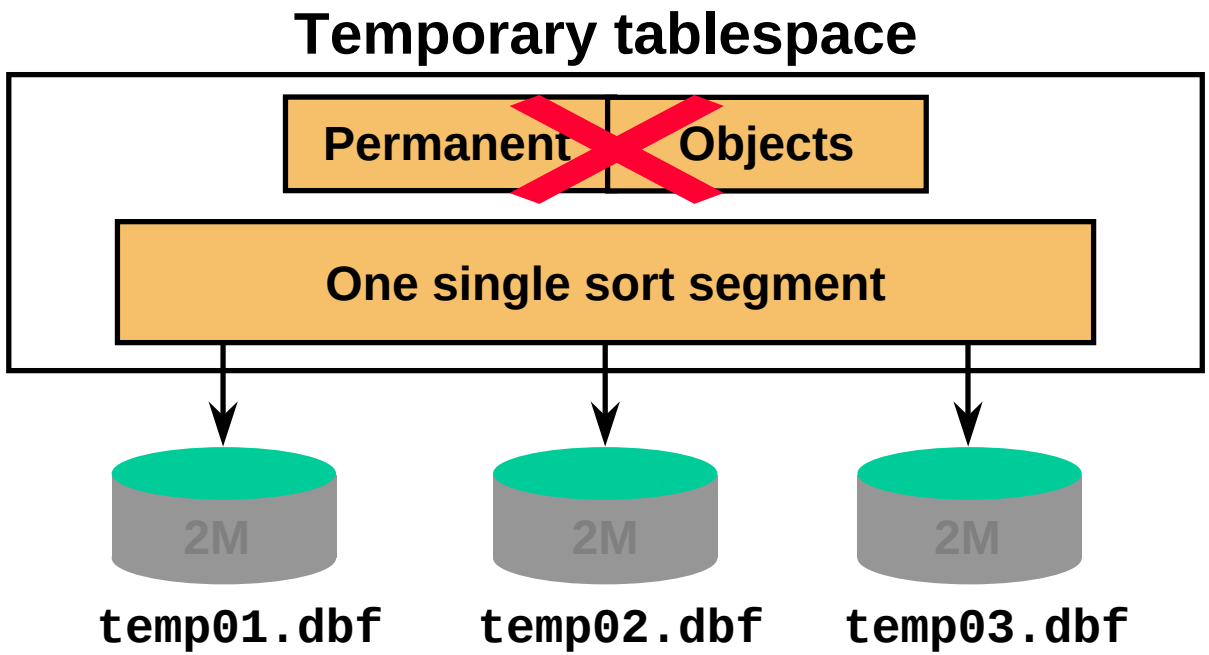
새로운 Sort 영역 관련 초기화 변수

- Pga_aggregate_target
  - 범위 : 10MB ~ 4000GB
  - 모든 서버 프로세스의 PGA를 합했을 때의 최대값
- Workarea_size_policy
  - 범위 : auto, manual
  - Auto : pga_aggregate_target이 설정되었을 때만 'auto'로 지정가능
  - Manual : work area의 크기가 *_area_size의 설정 값에 의해 결정된다

Sorting의 튜닝

- 가능하면 runtime시의 sorting은 피한다
- 메로리 sorting을 최적화하여, disk에 부하를 줄인다
  - Sorting이 Swapping이나 Paging없이 memory에서 진행되도록 한다
- Space 관리의 부하를 줄인다
  - Temporary space의 지정과 할당을 효율적으로 한다
  - Permanent tablespace에서는 temporary segment의 할당과 할당해제의 부하가 많이 발생한다

Temporary Space



- 생성 문장
  - CREATE TEMPORARY TABLESPACE TEMP TEMPFILE ‘$HOME/ORADATA/u06/temp01.dbf’ size 200M;

Temporary Space

- Temporary Tablespace
  - Sorting에 필요한 공간의 할당과 할당 해제를 위한 space management의 부하가 적다
  - Permanent object를 생성할 수 없다
  - ‘tempfile’을 가져야 한다
  - Tempfile은 backup하지 않아도 된다

Temporary Space Segment

- 첫번 째 sort에 의해 생성된다
- 필요하면 segment의 영역이 확장된다
- Extent별로 서로 다른 sorting에 사용할 수 있다
- SGA의 영역의 Sort Extent Pool(SEP)에 사용상황이 있다
- Database가 닫히면, 삭제된다

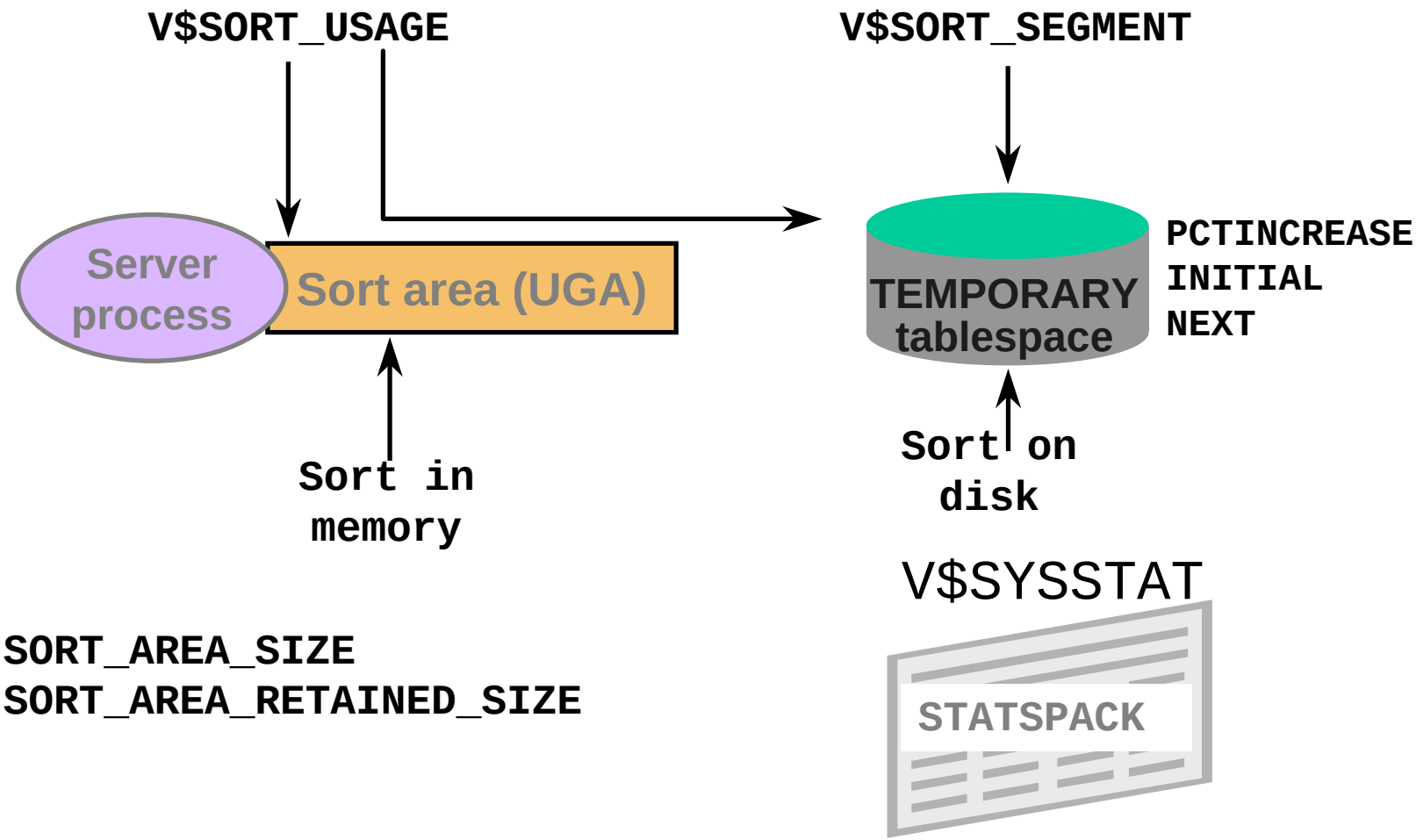
Sorting이 필요한 경우

- 인덱스 생성
- 인덱스 관리가 포함된 Parallel Inserting
- ‘order by’ 나 ‘group by’가 사용된 경우
- ‘distinct’가 사용된 경우
- ‘union’, ‘intersect’, ‘minus’ 연산자가 사용된 경우
- Sort-merge join이 발생한 경우
  - SQL> select department_name, last_name
    2 > from employees e, departments d
    3 > where e.department_id = d.department_id;
- ‘anlayze’가 사용된 경우
  - SQL> execute sys.dbms_utility.analyze_schema(‘HR’, ‘COMPUTE’);

Sorting의 회피

- 인덱스 생성시 'NOSORT' 지정한다
  - 미리 sort된 data를 SQL*Loader로 upload한 뒤에, 인덱스 생성시 사용한다
  - CPU가 여러 개라면, Parallel로 수행하는 것이 더 효과적이다
- 'UNION' 대신 'UNION ALL'을 사용한다
  - Union all은 중복 키 값을 허용하기 때문에 sorting이 필요없다
- 인덱스를 접근하여 Table을 조인한다
  - Nested Loop Join을 사용한다
  - SQL> select department_name, last_name
    2 > from employees e, departments d
    3 > where e.department_id = d.department_id;
- 'ORDER BY'를 사용할 컬럼에 대해 인덱스를 생성한다
- Analyze를 필요한 컬럼에 대해서만 실행한다
  - Analyze ... for columns
  - Analyze ... for indexed columns
- 큰 object에는 analyze 옵션으로 compute 대신 estimate를 실행한다

Sorting에 대한 진단



SORT_AREA_SIZE
SORT_AREA_RETAINED_SIZE

Sorting에 대한 진단

- V$sysstat
  - SQL> select * from v$sysstat
    2 > where name like '%sorts%';

```
STATISTIC# NAME                 CLASS    VALUE
---------- -------------------- ------  -------
       161 sorts (memory)          64        154
       162 sorts (disk)            64          4
       163 sorts (rows)            64     571768
```

- Statspack

```
Statistic                      Total  Per Transact    Per Logon    Per Second
------------------------------ ------- ------------  ----------  -----------
sorts (disk)                       4          .02          .41          .01
sorts (memory)                   154          .27         5.77          .12
sorts (rows)                  571768        39.62       862.59        18.19
```

Sorting의 진단과 Guide Line

- SQL> select d.value “Disk”, m.value “Mem”
  2> (d.value/m.value)*100 “Ratio”
  3> from v$sysstat m, v$sysstat d
  4> where m.name = ‘sorts (memory)’
  5> and d.name = ‘sorts (disk)’;

Disk	Mem	Ratio
23	206	11.165049

- Disk sorting이 Memory sorting의 5% 이하가 되도록 sort_area_size 값을 조정한다

Temporary Tablespace의 모니터링

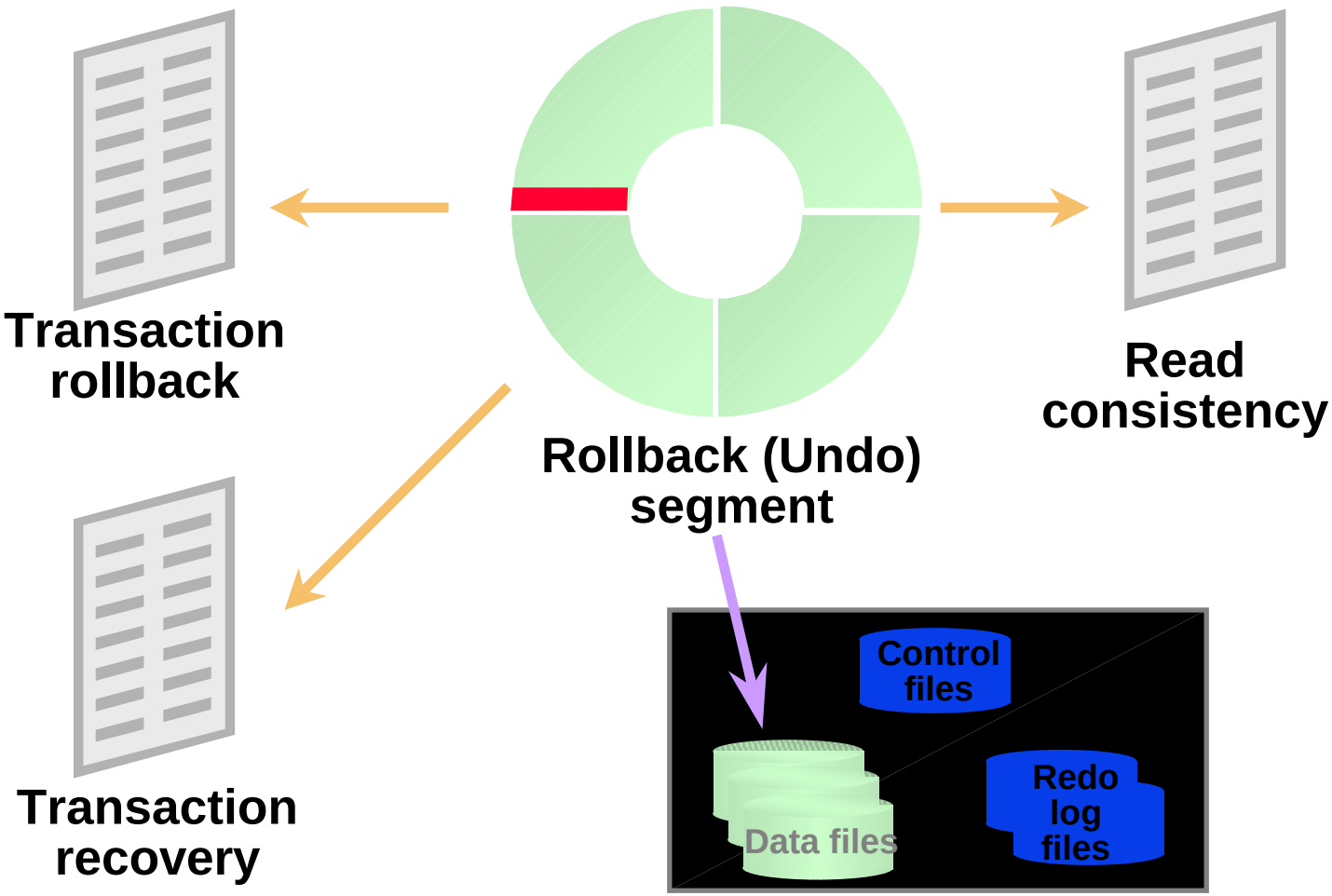
- SQL문장
  - SQL> select tablespace_name, current_users,
    2 > total_extents, used_extents,
    3 > extent_hits, max_used_blocks
    4 > max_sort_blocks
    5 > from v$sort_segment;
  - TABLESPACE_NAME CURRENT_USERS TOTAL_EXTENTS USED_EXTENTS EXTENT_HITS MAX_USED_BLOCKS MAX_SORT_BLOCKS
    ---------- ---------- ---------- ---------- ---------- ---------- ----------
    TEMP 2 4 3 20 200 200
  - Sort segment의 extent가 unlimited라는 것 외에는 Temporary tablespace의 default storage에 설정된 내용을 따른다

Temporary Tablespace의 구성

- Storage 값을 적절하게 설정
  - Initial과 next 값은 sort_area_size의 정수배로 설정한다
  - Pctincrease를 0으로 한다
- Sorting의 양에 따라, 사용할 temporary tablespace를 구분한다
  - SQL> select session_num, tablespace, extents, blocks
    2 > from v$sort_usage;
  - SESSION_NUM TABLESPACE EXTENTS BLOCKS
    ---------- ---------- --------- ---------
    16 TEMP 4 200
- Temporary tablespace의 datafile을 striping한다
- 모니터링
  - V$tempfile, dba_temp_files

Rollback Segment의 튜닝

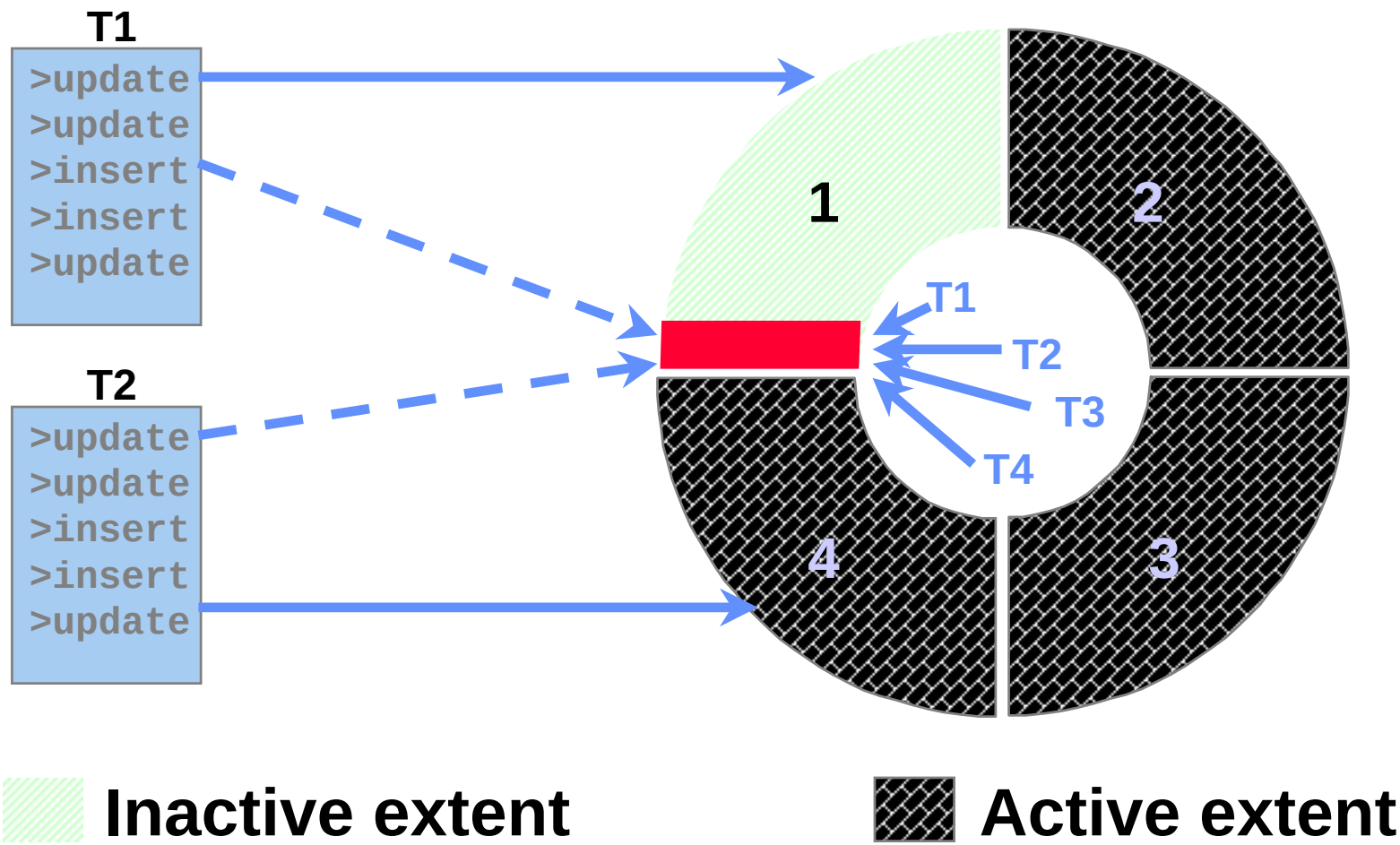
Rollback Segment의 쓰임새



Rollback Segment의 쓰임새

- 트랜잭션 롤백
  - 유저가 롤백을 실행했을 때, 트랜잭션의 commit되지 않은 부분을 롤백한다
- 트랜잭션 리커버리
  - 트랜잭션이 진행 중에 실패하는 경우, 해당 트랜잭션의 commit되지 않은 부분을 롤백한다
- Read Consistency 유지
  - 하나의 트랜잭션의 진행 중에 발생한 commit되지 않은 변경 내용은 다른 트랜잭션은 볼 수 없다.
  - 하나의 문장에서는 그 문장이 시작된 이후에 commit된 변경 내용을 볼 수 없다.

Rollback Segment Activity

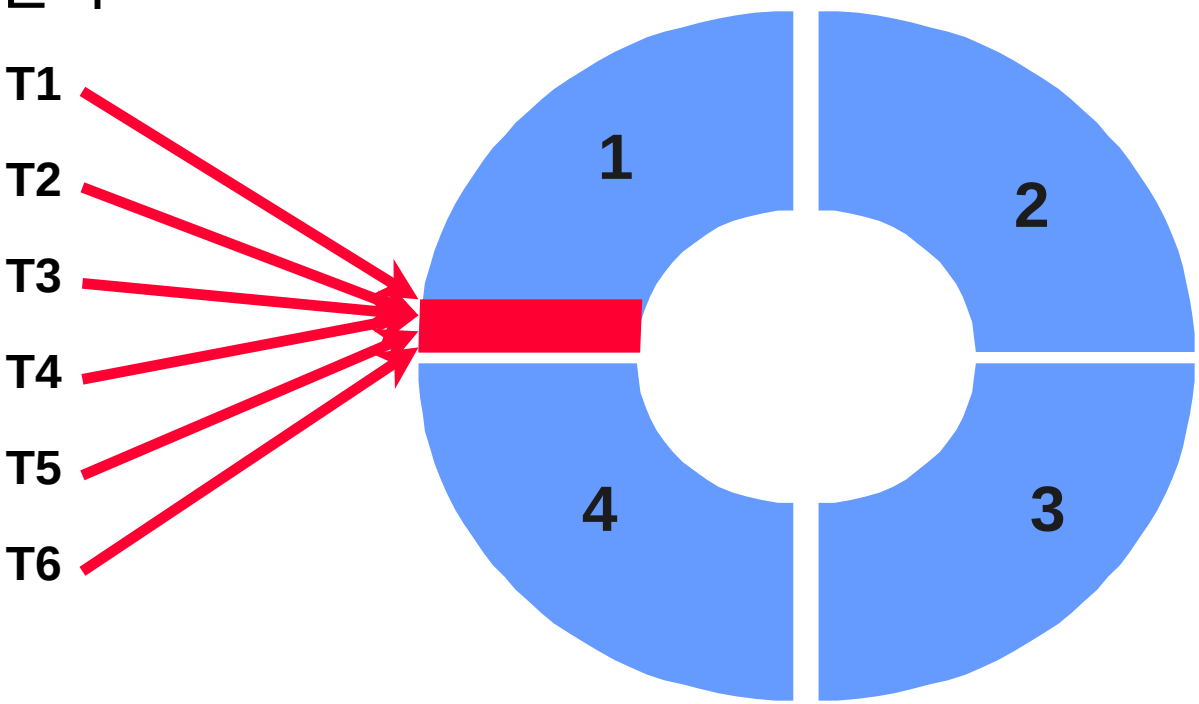


Rollback Segment Activity

- 활성 / 비활성 Extent
  - 트랜잭션은 롤백세스먼트의 익스텐트들을 순서대로 순환하면서 반복적으로 사용한다
  - 하나 이상의 트랜잭션이 동일한 익스텐트을 사용할 수 있다
  - 하나의 Undo Block은 하나의 특정 트랜잭션에 대한 undo 정보만을 가질 수 있다
- Rollback Segment Activity
  - Undo data를 저장할 undo block도 버퍼캐시에 올려져야 한다
  - Undo Data가 많이 발생하게 되면, 버퍼캐시도 그에 따라 증가시키거나 그렇지 않으면 물리적인 I/O가 빈번하게 된다

Rollback Segment Header Activity

- 롤백세그먼트의 헤더에는 각 트랜잭션의 정보가 들어 있다
- 모든 트랜잭션은 롤백세그먼트의 헤더를 갱신하기 위해 접근한다



Rollback Segment Header Activity와 그 영향

- Activity
  - 각 롤백세그먼트의 헤더에는 Transaction Table이 있다
  - Transaction Entry가 Transaction Table에 기록되면서 트랜잭션이 시작된다
  - Transaction Table에서 특정 트랜잭션의 가장 최근 Undo Data가 존재하는 undo block을 알 수 있다
- 영향
  - 트랜잭션이 짧고 많은 OLTP 시스템에서는 Rollback Segment Header의 Cache Hit Ratio가 성능에 영향을 미친다
  - 모든 트랜잭션이 Trasaction Table에 Update Access를 하기 때문에, 트랜잭션간의 경합을 없앨 수 있도록 Rollback Segment는 충분히 있어야 한다
  - Oracle9*i*에는 Automatic Undo Management 기능이 있다

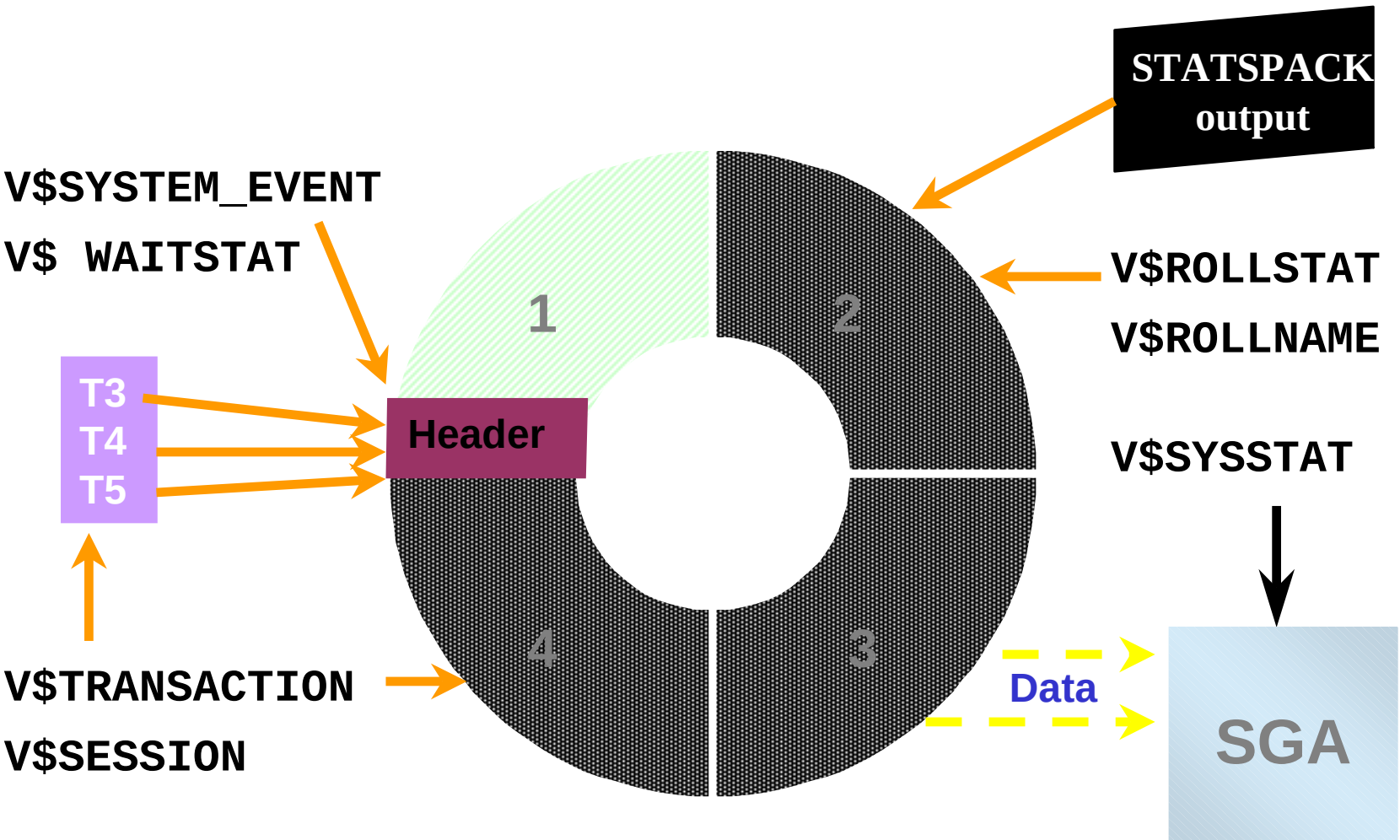
Rollback Segment의 확장(Extend)과 그 영향

- Extending
  - 현재 사용중인 extent에서 새로운 free undo block을 찾지 못하면, 그 다음 extent에서 undo block을 할당받는다
  - Current extent의 next extent가 현재 사용 중이라면, 먼저 새로운 extent를 할당받고 거기서 undo block을 할당받는다
- 영향
  - 일상적인 상황에서 rollback segment의 확장이 일어나는 것은 바람직하지 않다
  - 현재 발생하는 transaction의 길이를 고려하여 모든 undo data를 저장할 수 있을 정도의 extent를 가지고 있어야 한다
  - Dynamic space management를 피해야 한다

Rollback Segment의 Tuning Goal

- Transaction은 Rollback Segment에 접근하기 위해 기다리는 일이 없어야 한다
  - 충분한 개수의 Rollback segment가 있어야 한다
- 일상적인 상황에서 Rollback Segment의 확장은 없어야 한다
  - 세그먼트마다 크기와 개수가 적절한 extent를 가져야 한다
  - 가능한 한 rollback을 적게 사용하도록 해야 한다
- 어떠한 경우에도 rollback space가 모자라는 경우는 없어야 한다
  - Rollback segment의 크기를 제대로 결정해야 한다
  - 큰 트랜잭션은 작게 나뉘어져야 한다
- Read-consistent 이미지는 언제가 제공되어야 한다
  - Rollback segment의 크기와 개수에 의해 결정된다

진단 Tools


STATSPACK output
V$SYSTEM_EVENT
V$ WAITSTAT
V$ROLLSTAT
V$ROLLNAME
V$SYSSTAT
1
2
3
4
T3
T4
T5
Header
V$TRANSACTION
V$SESSION
Data
SGA

Rollback Segment Activity의 모니터링

- Dynamic Views
  - V$rollname : online rollback segment의 이름과 번호
  - V$rollstat : online rollback segment의 activity 통계
    - Transaction table에 대한 wait 회수
    - Transaction에 의해 기록된 undo data량
  - V$system_event
    - 시스템 전체에서의 event 정보 → 'Undo Segment Tx Slot' 이벤트는 Transaction Table에 대한 경합을 의미한다
  - V$waitstat
    - 모든 Rollback segment의 header와 data block에 대한 wait 정보의 누적 통계
  - V$sysstat
    - Consistent get과 data block get을 보여준다
    - Data에 대한 request 회수를 가지고 wait 회수를 계산할 수 있다
  - V$transaction
    - Rollback segment를 사용하는 현재 transaction에 대한 정보를 이용하여 필요한 rollback space를 알 수 있다

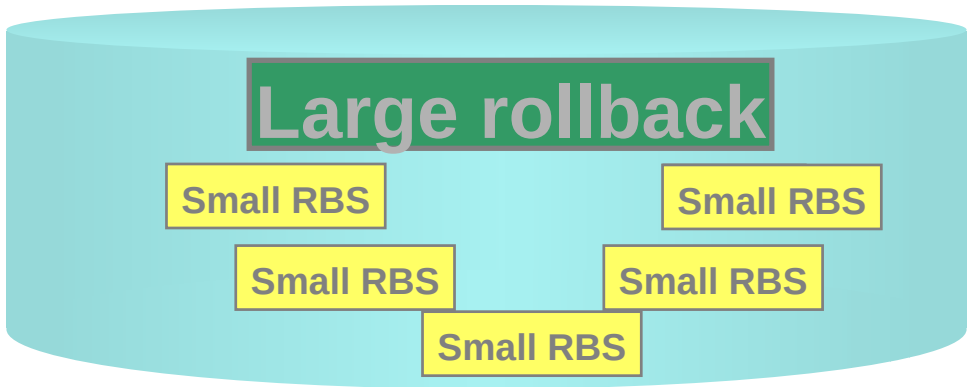
Rollback Segment Header에 대한 경합의 진단

- SQL
  - SQL> select class, count from v$waitstat
    2 > where class like '%undo%';
  - SQL> select sum(value) from v$sysstat
    2 > where name in ('db block gets', 'consistent gets');
  - SQL> select sum(waits) * 100 / sum(gets) "Ratio",
    2 > sum(waits) "Waits", sum(gets) "Gets"
    3 > from v$rollstat;
- Statspack

RBS No	Trans Table Gets	Pct Waits	Undo Bytes Written	Wraps	Shrinks	Extends
0	5.0	0.00	0	0	0	0
1	66.0	0.00	5,636	0	0	0
2	439.0	0.00	358,772	5	0	0
3	50.0	0.00	6,314	0	0	0
4	53.0	0.00	7,004	0	0	0

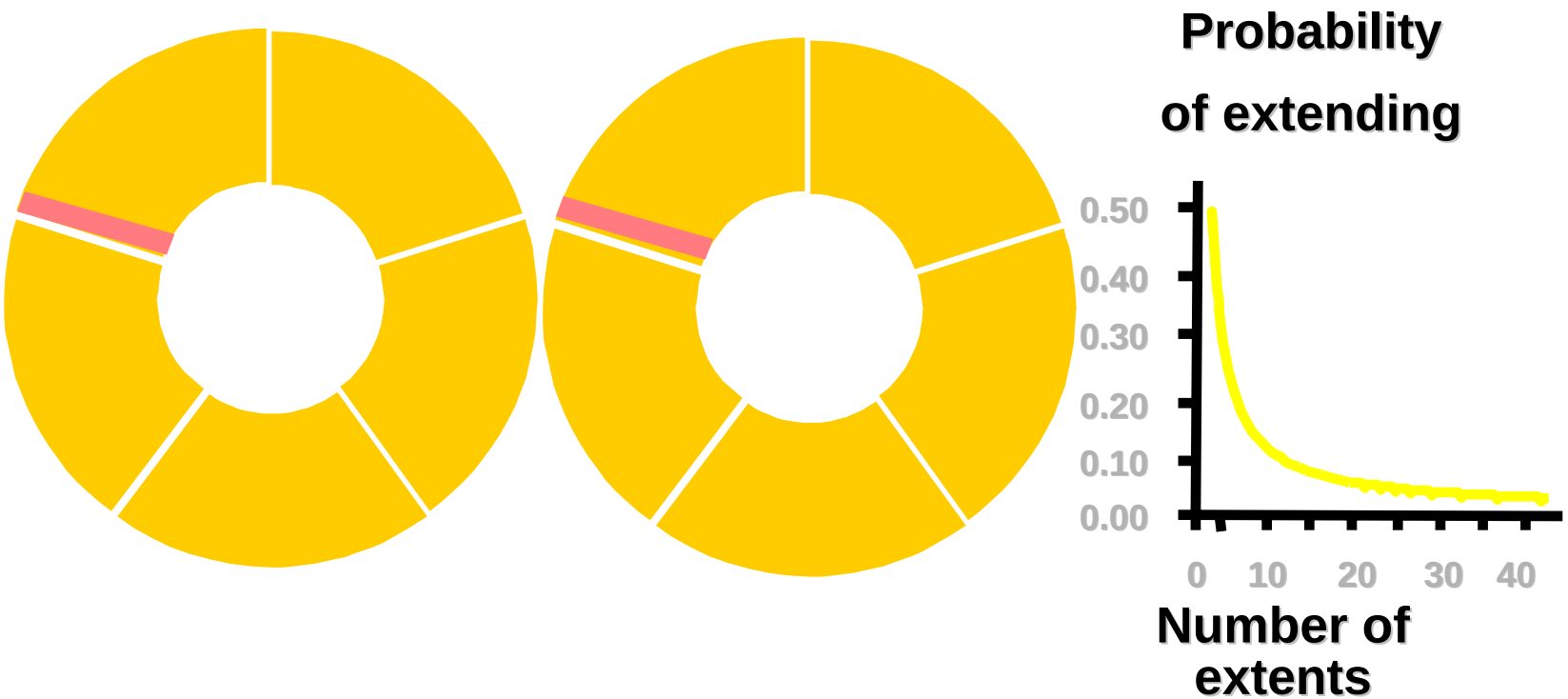
  - ----------

Rollback Segment 의 개수에 대한 Guide Line



- OLTP 성 Transaction
  - RBS 하나가 4개의 transaction을 처리하는 정도로 개수 조정
- Batch 성 Transaction
  - RBS 하나가 하나의 transaction을 처리하도록 개수 조정
  - 할당된 RBS가 확장 가능하도록 구성한다
    - Tablespace를 아주 크게 만들거나 자동 확장 가능하도록 한다
    - Maxextents를 무제한으로 한다
  - Transaction 시작 시에 특정 RBS를 지정하여 사용함
    - SQL> set transaction use rollback segment large_rbs;
    - SQL> execute dbms_transaction.use_rollback_segment('large_rbs');

Rollback Segment의 크기에 대한 Guide Line



Rollback segment 1 = Rollback segment 2

INITIAL = NEXT = 2^n, MINEXTENTS = 20

OPTIMAL 설정

Rollback Segment의 크기에 대한 Guide Line

- Storage 변수
  - Initial
    - 작은 transaction : 8KB, 16KB, 32KB, 64KB
    - 큰 transaction : 128KB, 256KB, 512KB, 1MB, 2MB, 4MB
  - Next = Initial
  - 사용하지 않는 large rollback segment 는 offline 시킨다
  - Minextents = 20으로 관리한다
    - Extent가 20일 때, extent 확장이 발생할 확률이 제일 적다
- Tablespace 크기
  - Large rollback segment의 확장이 가능하도록 충분한 여유공간을 유지한다
  - Rollback segment에 optimal 설정을 고려한다

Transaction의 롤백 데이터 크기

- DML과 Rollback 데이터의 양
  - Delete는 비싸다
  - Insert는 최소한의 rollback space만 사용한다
  - Update는 해당 column의 개수에 따라 rollback space의 양이 다른다
  - DML이 index data의 변경을 수반하면, 필요한 rollback space는 늘어난다
- Transaction의 롤백 데이터 크기 산정
  - SQL> select s.username, t.used_ublk, t.start_time
    2 > from v$transaction t, v$session s
    3 > where t.addr = s.taddr;
  - SQL> select usn, writes from v$rollstat;

Rollback 줄이기

- 일반 유저가 주기적으로 commit하도록 Application을 구성한다
- 개발과정에서 불필요하게 긴 트랜잭션을 피한다
- Import
  - Commit=y 를 사용한다
  - Buffer 옵션을 사용한다
- Export
  - Consistent=n 를 사용한다
- SQL*Loader
  - Rows 옵션을 사용하여 commit 주기를 지정한다

Rollback Segment가 작아서 생기는 문제

- Rollback space가 부족하면 transaction이 실패한다
  - 확장 가능한 최대 extent까지 모두 확장된 경우
  - Tablespace에 rollback segment가 더 이상 확장할 공간이 없는 경우
- Snapshot too old(ora-1555) 가 발생할 수 있다
  - Query의 대상인 block의 ITL이 재사용되었고, block의 SCN이 query가 시작되었을 때의 SCN보다 최근일 때
  - 롤백세그먼트의 헤더에 있는 Transaction Table에서 필요한 transaction slot이 재사용되었을 때
  - Transaction이 commit된 이후, 롤백세그먼트의 Undo Data가 덮어써졌을 때

Oracle9*i* 자동 Undo 관리

- 초기화 변수
  - AUTO
    - UNDO_MANAGEMENT
    - UNDO_TABLESPACE
    - UNDO_RETENTION
      - 보존해야 할 undo data의 양을 결정한다(초 단위)
      - UNDO SPACE = (undo_retention * (undo blocks per second * db_block_size)) + db_block_size
      - Undo blocks per second
        SQL> select sum(undoblks)/sum(((end_time – begin_time)* 86400))) from v$undostat;
    - UNDO_SUPPRESS_ERRORS
      - SMU로 설정된 상황에서 application이 SMU로 변경되지 않은 경우 사용 가능하다
  - MANUAL
    - ROLLBACK_SEGMENTS
    - RELLBACK_SEGMENTS_PER_TRANSACTION

자동 Undo 관리를 위한 Tablespace

- Tablespace의 생성
  - ‘create database’할 때, ‘undo tablespace’를 지정한다
  - ‘create undo tablespace’ 명령을 이용한다
  - Database 생성시 지정하지 않으면, ‘SYS_UNDOTBS’가 생성된다
    - Datafile : $ORACLE_HOME/dbs/DBU1<SID>.dbf
    - Autoextend ‘ON’ 으로 생성된다
- Tabespace의 속성 변경
  - SQL> alter tablespace undotbs1
    2 > add datafile ‘/u02/oradata/testdb/undotbs1_02.dbf’
    3 > size 500M autoextend on;
- 제약사항
  - Undo Tablespace에 일반 object를 생성할 수 없다
  - Tablespace에 대해 datafile과 extent management절 만 지정할 수 있다
  - Minimum과 default storage는 자동으로 생성된다

자동 Undo 관리를 위한 Tablespace

- Undo Tablespace의 교체
  - 여러 개의 Undo Tablespace 중에서 어느 시점에 active인 것은 하나 뿐이다
  - SQL> alter system set undo_tablespace=undotbs2;
- Undo Tablespace의 삭제
  - SQL> drop tablespace undotbs_2;
  - Active가 아닐 때 drop할 수 있다
    - 해당 undo tablespace를 사용하는 instance가 없을 때
    - 해당 undo tablespace에 존재하는 transaction table에 uncommitted transaction이 없을 때
    - Drop된 undo tablespace의 committed data를 access하는 query는 실패하게 된다

자동 Undo 관리의 모니터링

- V$undostat
  - Manual과 auto 모드 모두에서 undo segment의 모니터링을 위해 사용 가능하다
  - Undoblks 칼럼은 할당된 undo block의 개수를 보여준다