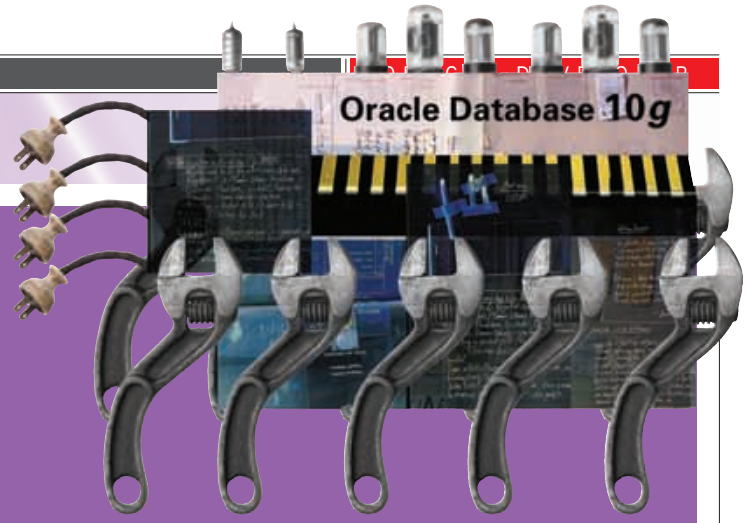


Linux에서 Oracle Database 10*g*의 튜닝

- Automatic Storage Management의 활용

글 ‖ **Bert Scalzo** <Quest Software의 제품 설계자> bert.scalzo@quest.com

Oracle Database 10*g*의 Automatic Storage Management 기능은 점점 늘어가는 스토리지 네트워크를 관리해야 하는 데이타베이스 관리자와 Linux 시스템 관리자에게 매우 유용한 기능이다. LInux에서 ASM을 사용해 Oracle Database 10*g*를 최적으로 튜닝해 보자.

오라클의 새로운 릴리즈를 대할 때마다 필자는 영화 '저크(The Jerk)'에서 Steve Martin이 새 전화번호부를 받았을 때 보이는 반응을 보이곤 한다. 오라클이 또 어떤 새로운 기능들을 선보일지 잔뜩 기대하게 되는 것이다. Oracle Database 10*g*를 대했을 때도 그 흥분은 결코 덜하지 않았다. 이 오라클 데이타베이스 최신 버전은 상당히 흥미로운 기능들을 대량으로 제공하는데, 여기서는 그 중의 단 하나, Linux에서 데이타베이스 디스크 공간 관리를 쉽게 만들어줄 ASM(Automatic Storage Management) 기능을 점검해 보기로 한다.

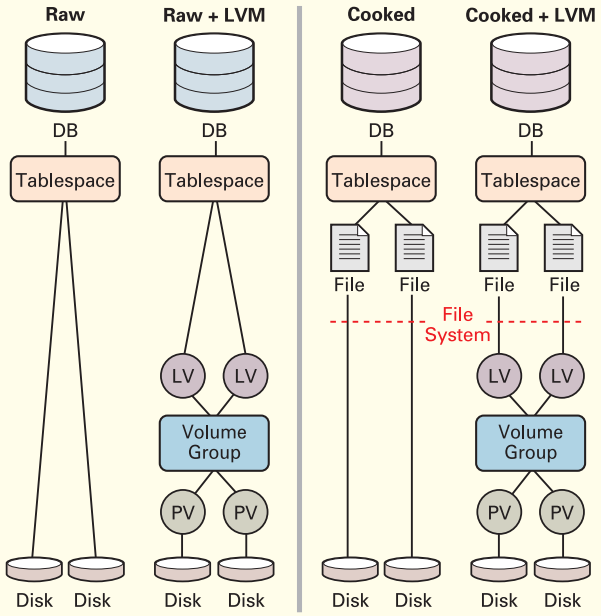
오늘날 데이타베이스 용량이 급격히 증가하고 SAN(Storage Area Network)과 NAS(Network-Attached Storage) 디스크 기술이 보편화되면서, 시스템 관리자와 데이타베이스 관리자는 단 하나의 데이타베이스를 위해서 수백, 수천 개의 디스크를 관리해야 하는 상황에 처해 있다. 이 많은 디스크의 용량 계획, 초기화, 할당, 관리 및 튜닝 업무는 상당한 부담이 아닐 수 없다. 그 결과, '조건부 항복'이 비일비재로 벌어지고 있는데, 많은 곳에서 디스크 스토리지 팜(farm)을 일종의 블랙박스로 간주하면서 데이타베이스의 복잡성을 애써 외면하고 있다. '세세히 알 필요가 없다'느니 '무조건 하드웨어만 믿으면 된다'는 말이 정당성을 얻기도 한다. 그러나, 이 블랙박스식 접근은 데이타베이스의 I/O 병목을 야기해 진단과 치료에 많은 시간이 들게 된다. 반면, 데이타베이스 관리자가 모든 디스크를 관리하는 곳에서는 이 관리자들의 소중한 시간이 너무 많이 뺏긴다는 문제가 있다. 이 양극단 모두 수용하기에는 분명 한계가 있는 것이다.

Oracle Database 10*g*의 새로운 ASM 기능은 앞에서 언급한 극단적인 해결책의 중도를 취함으로써 쉽고도 효과적인 대안을 제시한다. 모든 것이 잘 진행되리라는 막연한 희망만 가지고 데이타베이스의 복잡한 문제들은 외면한다거나, 아니면 복잡한 디스크 관리에 온통 시간을 쏟아붓는 대신, ASM이 모든 디스크를 관리하도록 하면 되는 것이다. 관리자는 템플릿을 통해 지정된 미러링과 스트라이핑 중 하나를 선택하면서 오라클에 디스크를 할당한 후 ASM이 이 스페이스를 관리하도록 하면 된다. 이렇게 함으로써 대용량 디스크 스페이스 관리에 필요한 기존 툴들, 즉, LVM(Logical Volume Manager)과 파일 시스템 그리고 이 둘을 관리하는 데 필요한 무수한 명령어들을 사용하지 않아도 된다. 따라서, 데이타베이스 수요 증가에 맞춰 Linux 데이타베이스 서버를 더 빨리, 더 쉽게 배포할 수 있으며, I/O 효율성도 한층 높일 수 있다.

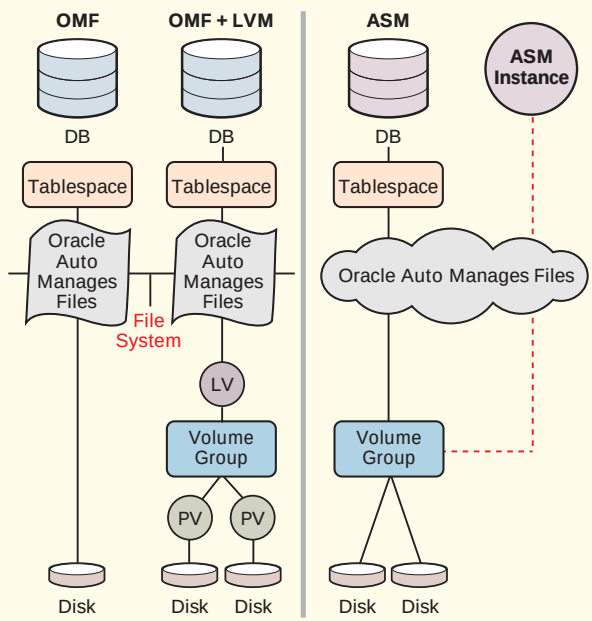
이 글에서는 Linux에서 LVM과 ASM을 사용하는 경우의 차이점을 자세히 살펴볼 것이다.

기존 방법들

본론에 앞서 지금까지 데이타베이스 관리자들이 오라클 데이타베이스에 디스크 스페이스를 할당한 방법을 검토해 보자. 먼저, ASM을 이용하면 작업이 얼마나 간편해지는지 도표를 통해 알아보자. 셋업 과정과 관리 작업이 얼마나 간편해지는지 알 수 있다. 그 다음에는 각각의 옵션들이 타당해 보이는 몇 가지 시나리오를 검토해 보자. 이 시나리오를 통해 ASM이라는 새로운 기술과 각 옵션들이 전체 데이타베이스 스토리지에 얼마나 적합한지 알 수 있을 것이다. <그림 1>은 기존 방법들을 도표화한 것이다.



<그림 1> 기존 방법에 의한 디스크 관리



<그림 2> Oracle9*i* Database의 Oracle Managed File과 Oracle Database 10*g* ASM의 비교

<그림 1>은 디스크를 관리할 때 기본적으로 두 가지 선택 - 미가공(raw)이냐 가공(cooked)이냐, 그리고, LVM을 사용하느냐 하지 않느냐? - 이 가능함을 보여준다. 이 그림들은 더욱 복잡해질 수 있는데, '디스크'로 명명되어 있는 것이 실제로는 유사 디스크일 수 있기 때문이다. 예를 들어, EMC 디스크 어레이에서 디스크는 '스핀들(spindle)'로 되어 있어, 하이퍼 디스크로 세분될 수 있다. Linux에서 '디스크'로 명명되어 있는 것은 단순히 디바이스 레퍼런스 - 실제 디스크의 파티션일 수 있다 - 이다. 그러나, 여기서 디스크는 세분되지 않는 것으로, 즉 도표에서의 각 디스크는 실제로 별개의 디스크 드라이브라고 생각하자.

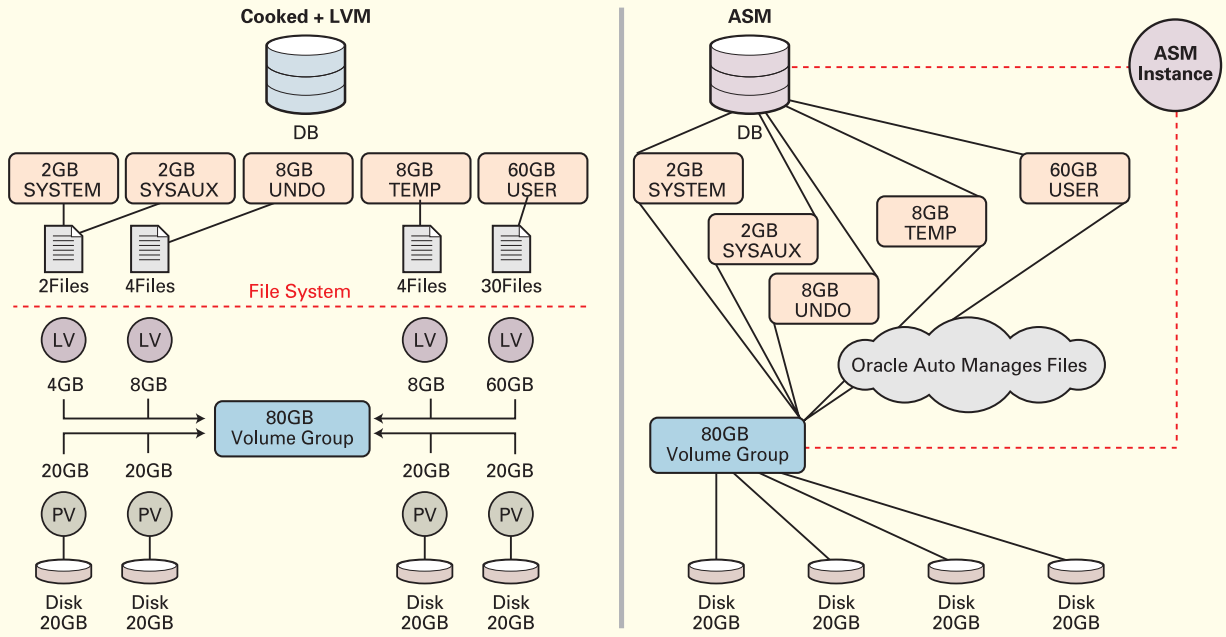
본론으로 돌아가, <그림 1>에서 옵션을 선택하도록 하자. 그런데, <그림 1>은 수백, 수천 개의 디스크로 확장될 수 있다. 디스크 수가 방대해지면, 일반적으로 LVM을 쓰게 된다. 그리고, 몇몇 운영체계에서는 미가공 디바이스를 선택하기도 하는데, Linux에서 미가공 디바이스는 결코 최선의 방법이 아니다(이에 대한 자세한 내용은 'Tuning an Oracle8*i* Database Running Linux(http://otn.oracle.com/oramag/webcolumns/2003/techarticles/scalzo_linux02.html)'를 참조하기 바란다.) 뿐만 아니라, 미가공 디바이스 사용에 따른 골치 아픈 관리 문제들도 감안해야 한다. 따라서 LVM을 이용하는 가공 파일 시스템(<그림 1>에서 가장 오른쪽에 있는 도표)이 최상의 선택이라는 결론에 이르게 된다. 그래서 여기서는 이 시나리오와 Oracle Database 10*g* ASM을 비교하기로 한다.

새로운 방법

그러면, ASM에 대해 좀더 자세히 살펴보기로 하자. <그림 2>는 Oracle9*i* Database의 OMF(Oracle Managed File)와 Oracle Database 10*g* ASM을 비교한 것이다. 오라클의 탁월한 다른 기능들과 마찬가지로, ASM을 제공하기까지 점진적인 발전을 거쳐왔다. 그 당시에는 미처 몰랐지만, Oracle9*i* Database의 OMF도 그런 발전 과정의 하나였다.

여러 면에서 OMF는 자동 파일 관리 개념을 오라클 데이타베이스 엔진 내에 구현한 첫 번째 시도였다. 그 자체는 상당한 발전이었지만, 몇 가지 유용하지 않은 측면도 있었다. 데이타베이스에서 디스크 관리의 복잡성을 제거하려던(앞에서 필자가 '블랙박스' 식 접근이라고 했던) 데이타베이스 관리자들은 DB_CREATE_FILE_DEST 파라미터를 통해 단 하나의 파일 목적지(destination)를 데이타베이스에 제공해야 했다. 이 파라미터 값은 오직 하나의 파일시스템 기반 스토리지 로케이션만 나타낼 수 있었는데, 이것은 하드웨어 RAID 뒤에 숨겨져 있는 복수의 디스크들을 나타내는 싱글 유사 디스크일 수도 있었고, 아니면, 여러 개의 물리적 디스크들을 포함하고 있는 싱글 로지컬 볼륨일 수도 있었다. 어떤 경우이든, Oracle9*i* Database는 모든 파일시스템 파일들을 자동으로 관리한다. '싱글 스토리지 로케이션'이라는 단점 외에 OMF의 큰 단점은 미가공 디바이스를 처리하지 않으므로 원래의 운영체계 기반 파일 시스템을 사용해야 한다는 제약이 있다는 것이다. 오라클 매뉴얼에도 OMF의 한계가 다음과 같이 기술되어 있다.

- 다음 사항에 의해 지원되는 데이타베이스
  - 스트라이핑/RAID와 동적으로 확장 가능한 로지컬 볼륨을 지원하는 로지컬 볼륨 매니저
  - 대형의 확장 가능한 파일들을 제공하는 파일 시스템
- 로우엔드 데이타베이스나 테스트 데이타베이스(그 자체로 선언문을 만들 수 있는)



<그림 3> ASM vs. LVM의 셋업 과정 비교

그러면, 다시 한 번 <그림 2>를 보면서 ASM(맨우측 도표)에 집중해 보자. LVM이나 파일 시스템이 필요 없음을 알 수 있다. 그리고, ASM 관리하에 단순히 디스크 그룹들만 있으며, 이 디스크 그룹들을 테이블스페이스에 사용할 수 있다. 파일 관리는 OMF에 의해 자동으로 이루어지지만, 이제는 완전히 내부적으로 해결된다. /etc/fstab 파일을 마운트하기 위한 파일 시스템이 없으며, ls, cp, tar 같은 운영체계의 파일 명령어들을 사용해 액세스해야 하는 파일도 없다. 이제는 데이타베이스에 질의해 모든 정보를 볼 수 있다.

또, ASM은 파일 레벨에서 데이타를 스트라이핑/미러링 할 수 있는데, 이것은 다른 방법을 이용하는 경우 디스크 레벨에서만 가능한 것이다. 그리고, 오라클은 디스크 그룹 내에 디스크가 추가/삭제되거나 장애를 일으킬 때마다 완벽한 미러링/스트라이핑 작업을 자동으로 관리하며, 그 와중에도 항상 데이타베이스를 온라인으로 유지한다.

오라클은 여러 LVM에서 사용되는 간단한 산술 알고리즘과는 근본적으로 다른, 매우 효율적인 새로운 방법을 사용함으로써 소기의 목적을 달성하고 있다. ASM 오라클 인스턴스가 추가로 필요하다고 강변할지 모르지만, 아래의 LVM과 ASM의 비교 예제를 보면, 셋업이 매우 쉽고 간편하며, 100% 자동화된 고도로 효율적인 관리가 가능하고, 모든 데이타베이스 인스턴스에 활용할 수 있음을 알 수 있을 것이다. ASM은 구현과 관리가 한결 쉬울 뿐 아니라 기대한 만큼의 탁월한 성능도 제공한다.

더욱 손쉬운 셋업

ASM은 파일 시스템이나 LVM에 비해 셋업이 훨씬 쉽고 간단하다. 다음과 같은 하드웨어/소프트웨어 조건을 가정하고, 사용자 데이타를 위한 데이타베이스와 그에 필요한 테이블스페이스만 작성하면 되는 간단한 데이타베이스를 생각해 보자.

- RAID 0 - 전체 드라이브의 모든 것을 스트라이핑한다.
  - 스트라이프 폭 = 4
  - 스트라이프 길이 = 64K
- 20개의 20GB IDE 디스크 - 각각은 한 개의 파티션을 가짐
- Linux ext3 파일 시스템(최대 파일 크기는 2GB)
- 5개의 테이블스페이스
  - SYSTEM, 2GB, 1개의 데이타 파일
  - SYSAUX, 2GB, 1개의 데이타 파일
  - UNDO, 8GB, 4개의 데이타 파일
  - TEMP, 8GB, 4개의 데이타 파일
  - USER, 60GB, 30개의 데이타 파일
- 디스크 드라이브당 1개의 피지컬 볼륨(PV)
- 1개의 볼륨 그룹(VG) - VG01
- 4개의 로지컬 볼륨(LV)
  - LV01, 4GB, SYSTEM과 SYSAUX
  - LV02, 8GB, UNDO
  - LV03, 8GB, TEMP
  - LV04, 60GB, USER

<그림 3>은 각 환경의 상대적 복잡성을 보여준다. 그러나, 이 그림은 단지 디스크 드라이브가 4개인 경우이므로, 실제로는 드라이브가 수천 개

인 경우를 상상해 보라.

그러면, 각 환경에서 데이타베이스를 구축하는 실제 과정을 비교해 보자. 단, LVM 예제에서는 간소화를 위해 실제 처리해야 하는 여러 가지 오버헤드 문제들은 의도적으로 제외했다. 그러나, 최적의 디스크 스페이스를 위해서는 이러한 오버헤드 문제들을 반드시 처리해야 할 것이다. 하지만, ASM을 사용한다면, 복잡한 스토리지 용량계획과 관리 문제들 중 상당수를 피할 수 있다.

LVM을 이용해 가공된 파일들을 사용하는 과정은 다음과 같다(이 예제에서 b~e 디바이스명이 /dev/hdb ~ /dev/hde이므로 IDE 디스크 드라이브는 두 번째부터 다섯 번째 드라이브라고 가정한다).

1. fdisk /dev/hdb set its type to 0x8e (LVM partition)
2. fdisk /dev/hdc set its type to 0x8e (LVM partition))
3. fdisk /dev/hdd set its type to 0x8e (LVM partition))
4. fdisk /dev/hde set its type to 0x8e (LVM partition))
5. pvcreate /dev/hdb /dev/hdc /dev/hdd /dev/hde)
6. vgcreate VG01 /dev/hdb /dev/hdc /dev/hdd /dev/hde)
7. lvcreate -L 4 G -i 4 -I 64 -n LV01 VG01)
8. lvcreate -L 8 G -i 4 -I 64 -n LV02 VG01)
9. lvcreate -L 8 G -i 4 -I 64 -n LV03 VG01)
10. -L 60 G -i 4 -I 64 -n LV04 VG01)
11. mkfs -t ext3 /dev/VG01/LV01)
12. mkfs -t ext3 /dev/VG01/LV02)
13. mkfs -t ext3 /dev/VG01/LV03)
14. mkfs -t ext3 /dev/VG01/LV04)
15. mount /dev/VG01/LV01 /home/oracle/oradata/LVMDB/system)
16. mount /dev/VG01/LV02 /home/oracle/oradata/LVMDB/undo)
17. mount /dev/VG01/LV03 /home/oracle/oradata/LVMDB/temp)
18. mount /dev/VG01/LV04 /home/oracle/oradata/LVMDB/user1)
19. edit /etc/fstab and add the new mount point entries)
20. Create initLVMDB.ora file)
    INSTANCE_TYPE = RDBMS)
21. SQL Plus connect as SYSDBA for SID=LVMDB)
22. STARTUP NOMOUNT PFILE=initLVMDB.ora)
23. CREATE SPFILE FROM PFILE=initLVM.ora)
24. Create Oracle database and user tablespace using SQL code. (<리스트 1> 참조)

<리스트 1> SQL 코드를 사용해 오라클 데이타베이스와 사용자 테이블스페이스를 작성하는 방법(LVM)

```
create database LVMDB
controlfile reuse
logfile       '/home/oracle/oradata/LVMDB/redo_log01.dbf' size 16M,
              '/home/oracle/oradata/LVMDB/redo_log02.dbf' size 16M
datafile      '/home/oracle/oradata/LVMDB/system/system01.dbf' size 2 G
sysaux datafile '/home/oracle/oradata/LVMDB/system/sysaux01.dbf' size 2 G
default temporary tablespace temp
tempfile      '/home/oracle/oradata/LVMDB/temp/temp01.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/temp/temp02.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/temp/temp03.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/temp/temp04.dbf' size 2 G
extent management local uniform size 64k
undo tablespace undo
datafile      '/home/oracle/oradata/LVMDB/undo/undo01.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/undo/undo02.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/undo/undo03.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/undo/undo04.dbf' size 2 G;

create tablespace USER
datafile      '/home/oracle/oradata/LVMDB/user1/user01.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user02.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user03.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user04.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user05.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user06.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user07.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user08.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user09.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user10.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user11.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user12.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user13.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user14.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user15.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user16.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user17.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user18.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user19.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user20.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user21.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user22.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user23.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user24.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user25.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user26.dbf' size 2 G,
              '/home/oracle/oradata/LVMDB/user1/user27.dbf' size 2 G,
```

```
                'home/oracle/oradata/LVMDB/user1/user28.dbf' size 2 G,
                'home/oracle/oradata/LVMDB/user1/user29.dbf' size 2 G,
                'home/oracle/oradata/LVMDB/user1/user30.dbf' size 2 G
extent management local uniform size 64k;
```

이제 수많은 데이타 파일을 가진 이러한 사용자 테이블스페이스가 수십 개 필요하다고 생각해 보자. 왜 이 예제를 쉽게 확장할 수 없는지 어렵지 않게 알 수 있을 것이다. 핫 스팟을 찾기 위해 모든 테이블스페이스, 데이타 파일, 로지컬 볼륨, 피지컬 볼륨 및 실제 디스크 드라이브를 추적하기란 여간 어려운 일이 아니다.

하지만, ASM을 사용한다면, 훨씬 간단해진다.

```
1. Create initASM.ora file
    . INSTANCE_TYPE = OSM
2. SQL Plus connect as SYSDBA for SID=ASM
3. STARTUP NOMOUNT PFILE=initASM.ora
4. CREATE SPFILE FROM PFILE=initASM.ora
5. CREATE DISKGROUP dgroup1 EXTERNAL REDUNDANCY DISK
  '/dev/hdb','/dev/hdc','/dev/hdd','/dev/hde'
6. Create initASMDB.ora file
    . INSTANCE_TYPE = RDBMS
    . DB_CREATE_FILE_DEST = '+dgroup1'
7. SQL Plus connect as SYSDBA for SID=ASMDB
8. STARTUP NOMOUNT PFILE=initASMDB.ora
9. Create Oracle database and user tablespace using SQL (<리스트 2> 참조)
```

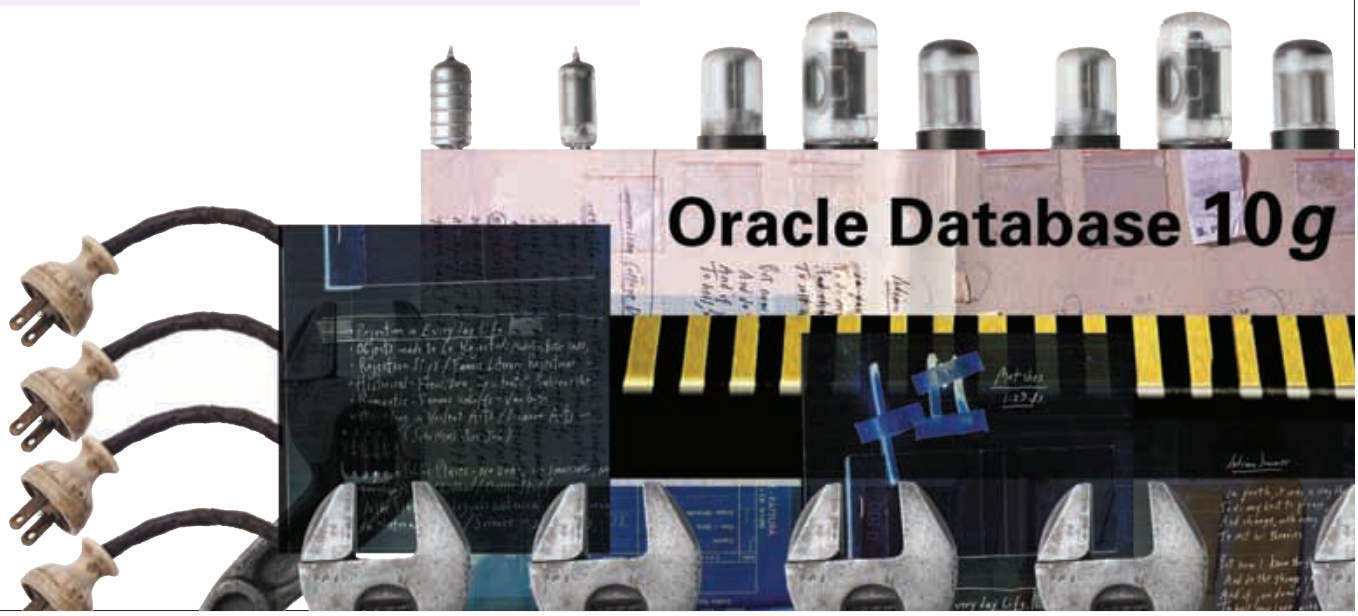
<리스트 2> SQL 코드를 사용해 오라클 데이타베이스와 사용자 테이블스페이스를 작성하는 방법(ASM)

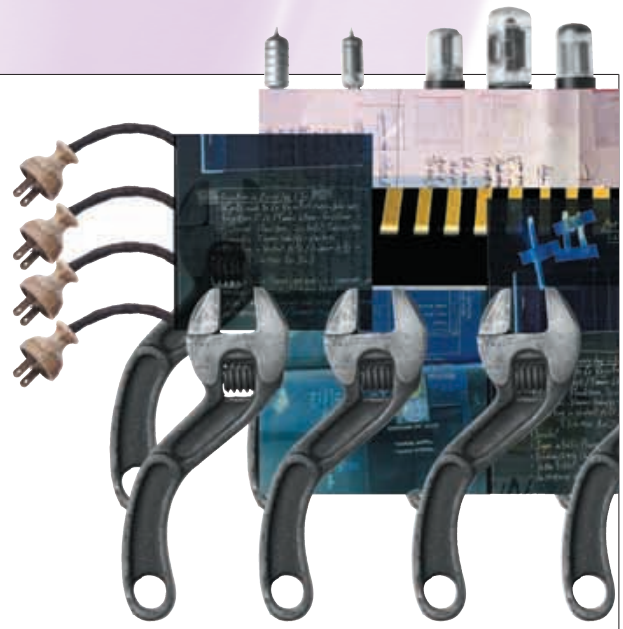
```
create database ASMDB
controlfile reuse
logfile      '+dgroup1' size 16 M
datafile     '+dgroup1' size 2 G
sysaux datafile '+dgroup1' size 2 G
default temporary tablespace temp
tempfile     '+dgroup1' size 8 G
undo tablespace undo
datafile     '+dgroup1' size 8 G;

create tablespace USER_LOCAL
datafile     '+dgroup1' size 60 G;
```

위의 신택스(syntax)는 데이타 할당 크기를 좀 더 정교히 제어하려 할 때의 신택스이다(사실, 디스크 레벨에서 작업하고 있으므로 그럴 필요가 없지만). 훨씬 더 간단한 신택스로 오라클이 모든 내부 스페이스 요구를 처리하도록 할 수 있다.

```
create database ASMDB;
```

이 신택스는 훨씬 짧아 읽기 쉬울 뿐만 아니라, 스트라이프된 디스크 그룹에서 생성된 리두 로그도 얻을 수 있다는 점에 주목하자. 중요한 점은 스토리지를 레이아웃하고 데이타베이스를 구축하는 프로세스가 수많은 디스크 드라이브를 관리하는 것(SAN이나 NAS)에 비해 훨씬 더 간단해져 Oracle Database 10*g* ASM으로 실패 없이 쉽게 업그레이드할 수 있다는 것이다.





손쉬운 변경

앞에서 살펴본 이점만으로 Oracle Database 10*g* ASM으로 바꿀 이유가 충분하지 않다면, 이제 4개 디스크를 추가하는 시나리오를 검토해 보면 얘기가 달라질 것이다. ASM의 진정한 가치가 여기에 있다고 해도 과언이 아닐 것이다.

하나의 USER 테이블스페이스에 10개의 테이블과 10개의 인덱스가 있고, 각 테이블이 각각 4GB, 각 인덱스가 각각 2GB를 차지해, USER 테이블스페이스가 꽉 찼다고 가정하자. 또 다른 테이블과 인덱스를 만들 공간이 없는 셈이다. 추가 스페이스 요구를 수용하기 위해 원래의 스토리지에 처음 4개와 똑 같은 디스크를 4개 더 추가한다고 하자. 즉, USER 테이블스페이스에 80GB를 추가한다고 하자.

LVM의 경우에는 다음 3가지 옵션이 있다.

- 새로운 로지컬 볼륨 LV05를 갖춘 새로운 볼륨 그룹 VG02를 만드는 것
- 볼륨 그룹 VG01을 확장해 새로운 로지컬 볼륨 LV05를 수용하는 것
- 로지컬 볼륨 LV04를 확장함으로써 볼륨 그룹 VG01을 확장하는 것

그러나, 여기서 새로 추가되는 80GB는 USER 테이블스페이스에 모두 할당된다고 가정했으므로, 새로운 로지컬 볼륨을 만드는 옵션은 제외된다.

기존 스토리지 설계를 유지하면서 스페이스만 추가하려 하므로, 세 번째 옵션을 선택해야 한다. 이 세 번째 옵션을 구현하는 방법은 다음과 같다.

1. fdisk /dev/hdf set its type to 0x8e (LVM partition)
2. fdisk /dev/hdg set its type to 0x8e (LVM partition)
3. fdisk /dev/hdh set its type to 0x8e (LVM partition)
4. fdisk /dev/hdi set its type to 0x8e (LVM partition)
5. pvcreate /dev/hdf /dev/hdg /dev/hdh /dev/hdi
6. vgextend VG01 /dev/hdf /dev/hdg /dev/hdh /dev/hdi
7. lvextend -L +80 G /dev/VG01/LV04
8. ext2online /dev/VG01/LV04
9. SQL Plus connect as SYSDBA for SID=LVMDB
10. Add new space to the tablespace using SQL code. (<리스트 3> 참조.)

<리스트 3> SQL 코드를 사용해 테이블스페이스에 스페이스 추가하는 방법 (LVM)

```
alter tablespace USER
add datafile    '/home/oracle/oradata/LVMDB/user2/user01.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user02.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user03.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user04.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user05.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user06.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user07.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user08.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user09.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user10.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user11.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user12.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user13.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user14.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user15.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user16.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user17.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user18.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user19.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user20.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user21.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user22.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user23.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user24.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user25.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user26.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user27.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user28.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user29.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user30.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user31.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user32.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user33.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user34.dbf'  size 2 G,
                '/home/oracle/oradata/LVMDB/user2/user35.dbf'  size 2 G,
```

```
'/home/oracle/oradata/LVMDB/user2/user36.dbf' size 2 G,
'/home/oracle/oradata/LVMDB/user2/user37.dbf' size 2 G,
'/home/oracle/oradata/LVMDB/user2/user38.dbf' size 2 G,
'/home/oracle/oradata/LVMDB/user2/user39.dbf' size 2 G,
'/home/oracle/oradata/LVMDB/user2/user40.dbf' size 2 G;
```

이 예는 매우 길 뿐만 아니라 여러 가지 문제도 있다. 첫째, 온라인 파일 시스템 크기 조정은 숙련된 작업을 요구한다. ext2online 유틸리티의 개발자는 '마운트되어 있는 파일 시스템의 크기 조정은 본질적으로 위험한 작업으로 파일 시스템을 손상시킬 수 있다' 고 전제하고 있다. 또, ext2online 유틸리티는 마운트되어 있는 파일 시스템을 확장만 할 수 있으므로, 파일 시스템을 확장하거나 줄이려 한다면, ext2resize를 사용해야 하는데, 이것은 마운트되어 있지 않은 파일 시스템에서만 사용할 수 있다. 따라서, ext2resize를 이용하려면, 테이블스페이스를 오프라인시켜야 한다.

또, LVM 방식은 명백히 드러나지 않는 몇 가지 단점들을 가지고 있다. 일반적으로 LVM을 사용할 때 8개 드라이브 모두에서 데이타가 스트라이프될 것으로 예상하겠지만, 실제로는 그렇지 않다. 로지컬 볼륨에 스페이스를 추가할 수 있다 해도, Linux에서 스트라이핑 속성은 변경시킬 수 없다(몇몇 UNIX LVM은 그런 기능을 제공하지만). 따라서, 각각 10개의 이전 테이블과 인덱스는 드라이브 b-e에서만 스트라이프되고, 새로운 테이블과 인덱스는 드라이브 f-i에서만 스트라이프된다(왜냐하면, USER 테이블스페이스가 이미 꽉 차 있으므로, 새 객체들은 새로운 스페이스에서만 만들어지기 때문이다). 심지어 테이블스페이스 객체들을 익스포트해 삭제하고 테이블스페이스를 통합한 다음, 합체한 테이블스페이스로 객체들을 다시 임포트해도, 로지컬 볼륨은 여전히 4웨이 스트라이핑만 가능할 것이다. 8웨이 스트라이핑이 가능하도록 하려면, 다음과 같은 작업을 수동으로 처리해야만 한다.

1. 테이블스페이스의 객체들을 익스포트한다(안전을 위해 한정된 세션에서만)
2. 테이블스페이스를 삭제한다.
3. 로지컬 볼륨을 삭제한다.
4. 새로운 로지컬 볼륨을 만든다(스트라이핑 파라미터를 -i 8로 설정)
5. 테이블스페이스를 만든다(이렇게 하면, 140GB에 대해 상당량의 데이타 파일 라인이 생길 것이다).
6. 이렇게 만들어진 테이블스페이스로 객체들을 임포트한다.

Next Steps

- Oracle Database 10*g* 제품 소개
  otn.oracle.com/products/database

바로 이런 이유로 ASM으로 바꿔야 한다. ASM을 이용하면, 한결 간단해지기 때문이다.

```
1. SQL Plus connect as SYSDBA for SID=ASM
2. ALTER DISKGROUP dgroup1 ADD DISK '/dev/hdf','/dev/hdg','/dev/hdh','/dev/hdi'
```

나아가, ASM은 디스크의 추가, 삭제 또는 장애 발생시 데이타베이스의 온라인 상태를 유지하면서 스트라이핑과 미러링을 자동으로 재조정한다. 따라서, 오라클은 사용자의 모든 객체를 완벽하게 스트라이프된 상태로 관리할 수 있다. 이것이 바로 ASM이 수동 튜닝 없이 최적에 가까운 I/O 밸런싱을 제공한다고 주장할 수 있는 근거이다. 가능한 많은 드라이브를 전반적으로 활용함으로써 핫 스팟을 제거하기 위해 데이타베이스 관리자들이 지난 수 년간 수동으로 해왔던 작업을 ASM은 완벽히 자동화한 것이다. 또, OSM_POWER_LIMIT나 다른 파라미터들을 통해 오라클이 언제 어떻게 수행될지를 제어할 수 있다(이에 대한 자세한 내용은 별도의 기사에서 다루어야 할 것이다).

탁월한 성능

Oracle Database 10*g*는 더 간단한, 더욱 자동화된 데이타베이스 관리를 목적으로 한다. 앞에서는 ASM만으로도 Oracle Database 10*g*로 업그레이드해야 하는 이유를 예증했다. 마지막으로, LVM과 ASM의 성능을 비교해 보는 것이 확실한 논증이 될 것이다. LVM과 파일 시스템을 제거하더라도, 동일한 작업을 수행하기 위해 오라클 기술을 사용해야 한다. 따라서, 이 오라클 기술이 얼마나 더 좋은 성능을 내는지 확인해야 할 것이다. 성능까지 뛰어나다면, 구현과 작성의 간편함과 더불어 금상첨화의 장점이 될 것이다.

<표 1>은 참조할 만한 몇 가지 비교 결과이다.

	ASM이 LVM에 비해
80GB 데이타베이스의 생성	11% 더 빠름
데이타베이스의 인덱스 구축	9% 더 빠름
200명 동시 사용자 액세스	5% 더 빠름

(Oracle Database 10*g* Beta 1으로 테스트한 결과임)

<표 1> ASM vs. LVM의 성능 비교

이 성능 비교 결과가 그리 대단한 것은 아니라 해도, 데이타베이스 관리자의 업무 부담을 덜어준다는 장점 외에 10% 정도의 성능 개선이라면, 업그레이드할 만한 충분한 근거가 될 것이다. 더구나, <표 1>의 수치는 Oracle Database 10*g*의 베타 버전으로 테스트한 결과임을 염두에 두어야 할 것이다.

이 글의 원문은 otn.oracle.com/oramag/webcolumns/2003/techarticles/scalzo_asm.html을 참고하기 바랍니다.