System Performance Tuning(I/O)

Author	이영진 대리, 강경구 대리
Creation Date	**2008-02-21**
Last Updated	**2008-02-21**
Version	**1.0**

Version	변경일자	변경자(작성자)	주요내용
1	**2008-02-21**	이영진대리, 강경구 대리	문서 최초 작성
2	2008-02-25	이영진대리, 강경구 대리	문서 업데이트
3			

Goodus
Breathing Life into Technology

Contents

1. Performance management & Tuning

1.1. 개요

대부분의 유닉스 초심자들은 시스템의 여러 가지 세팅을 적절히 조절해 주면 시스템의 성능이 매우 크게 향상될 것이라는 생각을 가지곤 한다. 하지만 이는 커다란 착각(?)이라고 말해주고 싶다. 왜냐하면 이미 웬만한 상업용 유닉스 시스템들은 어느 정도의 최적화가 된 상태로 시장에 나오는 것이기에 정확한 지식 없는 시스템 튜닝은 오히려 부작용을 일으킬 수 있는 것이다. 물론 Linux 나 FreeBSD 와 같은 비 상업용 유닉스를 쓰는 경우는 논외로 하겠다. 그러므로 유닉스 시스템 성능 향상을 위한 튜닝의 기본적인 원칙은 다음과 같다.

시스템이나 네트워크에 너무 많은 부하가 걸리지 않도록 한다. 부하가 많이 걸리게 되면 자원을 기다리는 프로세스들이 많아지게 되고 그러면 점점 누적되어 시스템의 성능이 급격히 저하되기 때문이다.

불필요한 기능이나 쓸모 없는 일을 위해 시스템 여분의 성능을 낭비하지 않는다. 예를 들면 시스템의 쿼터(용량제한)나 CPU 사용계산(Accounting) 등이 별로 필요 없는 경우 이에 대한 시스템의 불필요한 사용은 시스템의 성능저하의 한 몫만을 할 뿐이다.

추가적으로 고려할 사항은 프로그램을 만들 때는 시스템 성능을 고려한 프로그램의 작성이 요구되며 필요한 시스템 작업이 요구되는 경우 부하가 적은 야간을 이용해 cron 등 스케줄링하여 JOB 돌려놓으면 좋다.

시스템 성능을 향상시키는 것은 대부분의 IT 기업이 끊임없이 추구하는 목표이다. 처리해야 하는 데이터의 양이 증가하고 더욱 다양해짐에 따라 특정 작업을 실행하는데 허용되는 시간은 더욱 짧아지고 있다. 이를 위해 향상된 알고리즘이 개발되거나 또는 처리 속도가 더욱 빠른 시스템이 개발되어야 한다.

1.1.1. 성능 향상을 위한 일반론

다음은 시스템의 성능을 향상시키기 위해서 필요한 것들이다. 이것은 유닉스 시스템뿐만 아니라 다른 운영체계의 시스템에도 해당되는 것이다.

시스템이 충분한 메모리 크기를 가지도록 하자. 일반적으로 운영체계에서 요구하는 기본 메

모리 양이 있다. 이를 충분히 수용할 수 있어야만 그 시스템은 기대되는 성능을 낼 수 있는 것이다.

프로그램의 잘못된 사용을 바로잡는다. 앞서도 언급하였지만 시스템의 성능을 고려하지 않고 프로그램을 짜서 수행시키는 경우나 한꺼번에 많은 프로그램들을 불필요하게 수행시키는 등의 일은 시스템의 성능을 떨어뜨리는 결과를 초래한다. 또한 불필요한 Demon 의 수행도 마찬가지의 결과를 가져오는 것이다.

시스템의 하드 디스크나 파일 시스템을 정리한다. 이는 파일 시스템의 여러 가지 옵션을 조정하는 것과 쓸모 없는 디렉터리나 파일의 정리도 포함된다.

항상 네트워크 상태를 모니터링 한다. 특히 Error 율이 낮은지를 netstat 명령을 통해 체크한다. Kernel 재조정을 통해 필요 없는 드라이버와 옵션 등을 제거한다. 예를 들면 자신의 시스템에 장착되어 있는 하드웨어를 제외한 다른 회사의 드라이버들은 제거해도 아무 지장이 없다. 다만 만일의 경우를 고려해 기존의 드라이버들이 모두 있는 Kernel 백업은 받은 후에 제거한다.

1.1.2. 시스템 성능에 영향을 주는 요소들

시스템 성능에 영향을 주는 요소들은 다양하게 존재한다. 이 요소들은 여러 가지로 파악될 수 있지만 그 중에서도 시스템의 프로그램들이 사용하는 리소스의 측면에서 파악하는 것이 일반적이다. 시스템의 성능에 영향을 주는 요소들을 리소스 측면에서 파악하면 다음과 같다.

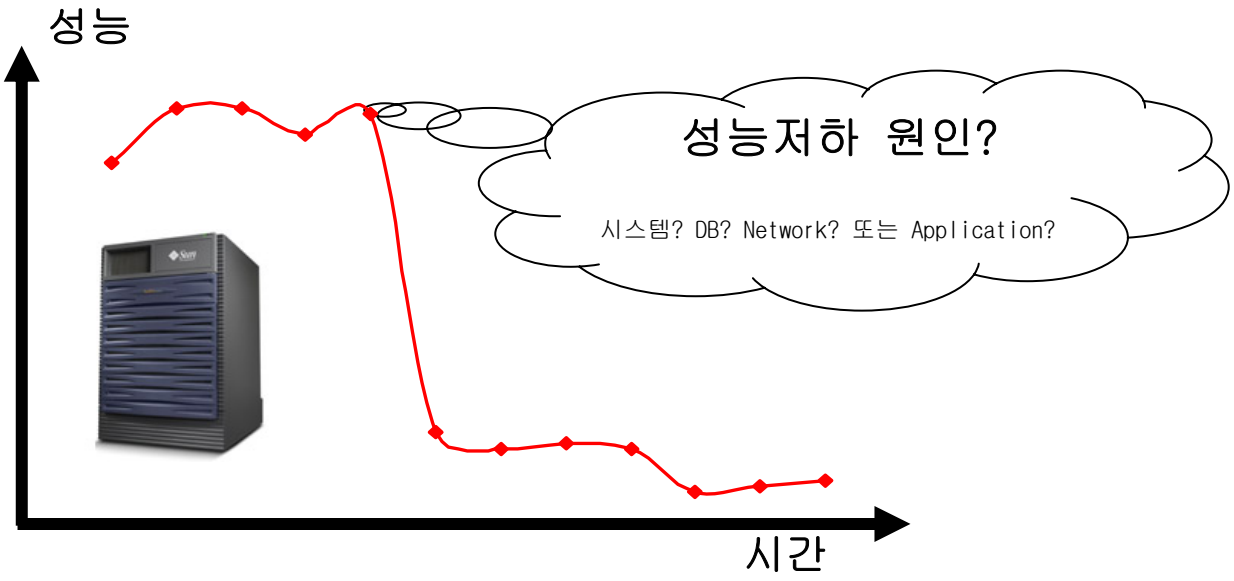
- CPU 사용 시간
- 메모리
- 하드 디스크 I/O 양
- 네트워크 I/O 양

모든 프로세스는 시스템 리소스의 일부를 사용한다. 이러한 리소스들을 여유있게 프로세스들이 사용한다면 시스템의 성능은 양호한 것이다. 하지만 그렇지 않고 리소스가 없어 프로세스들이 기다리는 경우가 늘어날수록 그 시스템의 성능은 감소되는 것이다. 프로세스들이 기다리는 시간을 시스템 성능의 척도로 사용하는 경우도 이러한 이유 때문이다.

1.2. 시스템 속도의 저하 원인 규명

- 시스템의 상태를 실시간 모니터링 하여 성능 저하로 인한 원인을 발견하고 대책을 수립함.
- 장애의 원인은 무엇입니까?

- 성능 저하의 원인이 어디에서 비롯되었습니까?
- 어플리케이션의 처리 속도를 어떤 방법으로 향상시킬 수 있습니까?
- 시스템 자원을 언제, 얼마만큼 증설하면 됩니까?

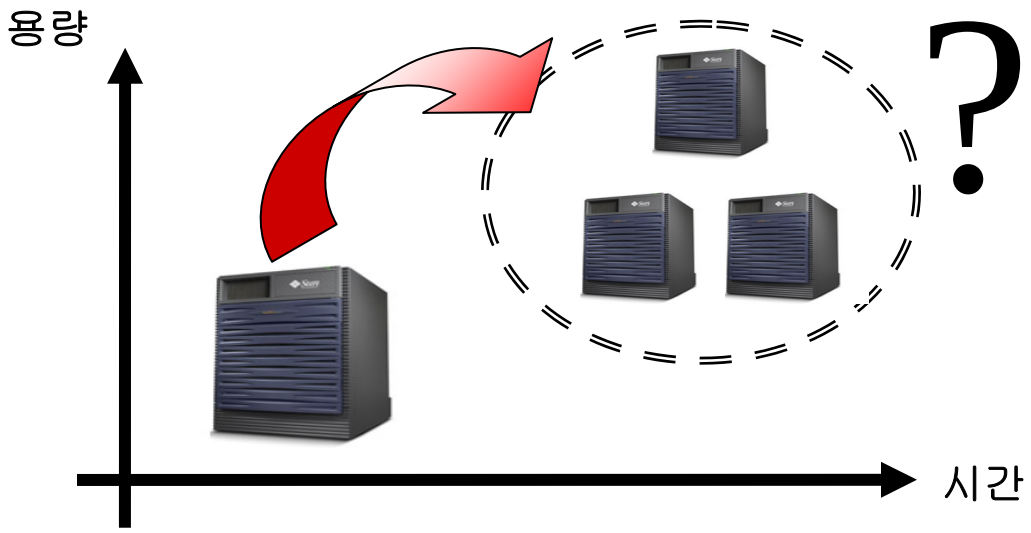


1.3. 시스템 통합으로 투자/관리 비용 감소

- 시스템 performance tuning 으로 다중 분산 화 된 Application 을 단일 시스템에 통합하여 최적화 할 수 있음.
- 이로 인한 시스템의 재투자를 줄이고 관리에 소요되는 비용을 줄일 수 있으므로 보다 경쟁력 있는 자사의 전산 환경을 구축 할 수 있음.

1.4. 시스템 자원 용량 계획(capacity planning)

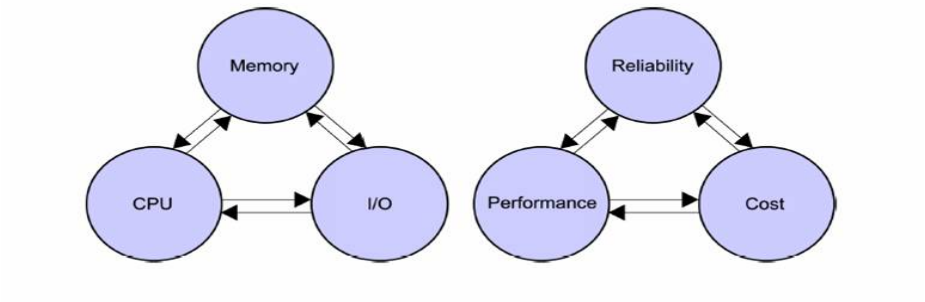
- 특정 기간 동안 모니터링 된 시스템의 데이터를 분석하고, 그 분석 결과를 토대로 미래의 자원 부족을 예측 할 수 있다.
- 미래에 필요한 자원을 미리 준비하여 가까운 미래에 자원 부족으로 인한 시스템 성능 저하를 줄이므로 안정적인 전산 환경을 유지 할 수 있다.



2. Intro Tuning

2.1. trade-offs in performance tuning

서론에서도 언급하였듯이 Tuning 을 하기 위한 절차를 계획하고, 계획된 순서에 맞게 진행을 하면서 적절한 부분을 monitoring & tuning 하여야 한다.



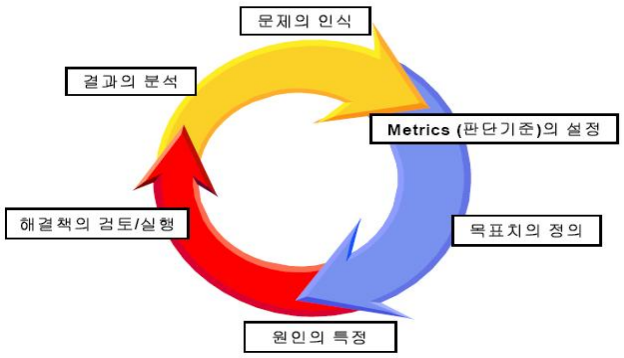
● 튜닝은 서로 영향을 준다. → 따라서, 관련부분을 같이 튜닝하여야 한다.
● CPU 문제를 해결 후 보통 I/O 문제가 발생할 수 있다. (I/O 문제가 해결되면, CPU 문제가 발생)
● Bottleneck 은 전이(傳移) 가능하다.
● 작업특성을 고려해서 반드시 Trade-off 를 만들어 놓고 튜닝 포인트를 잡아서 작업을 진행한다.

균형적인 튜닝을 위해서 권장되는 것이 반드시 Trade-Off 작업 계획서를 기반으로 튜닝을 진행하라는 것이다. 한 가지 시스템 자원(Resource)을 튜닝하여 개선할 경우 다른 자원에 어떤 영향을 미칠 것인가를 반드시 분석하여 놓고 Tuning 을 실행하여야 한다. 보편적으로 Trade-Off 는 성능 대 비용의 상관관계를 분석할 때 많이 사용하지만, Performance Management & Tuning 과정에서도 반드시 적용되어야 한다.

2.2. Basic tuning procedure

시스템을 Tuning 할 경우, 다음의 Process 에 따라 작업을 진행하는 것이 일반적이다. 시스템 도입시의 Tuning 도, (0)의 부분을 제외하고 같은 Process 가 된다.

(0) Performance 의 문제가 있는지를 판단한다.
(1) Performance 의 판단기준(Metrics)을 설정한다.
(2) Performance 의 목표치를 정의한다. 동시에 최우선해야 할 과제를 정의한다.
(3) 문제의 요인을 도출한다.
(4) 문제의 요인에 대응하는 해결책을 검토, 실행한다.
(5) 해결책을 실행한 결과를 분석한다.
(6) 목표가 달성되지 않았다면 위의 작업을 되풀이한다.



문제의 인식, 실태의 파악, 목표치의 정의, 원인의 특정, 해결책의 검토/실행, 결과의 분석이라는 6 항목으로 구성된다. 이 Cycle 을 되풀이하여 계속적으로 Tuning up 에 몰두하는 자세가 필요하다.

3. Disk Tuning

디스크 서브시스템의 성능 분석 및 기타 Tool 을 설명하며, 특히 성능 향상의 주요부분 및 디스크 관련 기술을 집중적으로 다루고자 한다.

3.1. 디스크의 성능 정의

디스크 서브시스템의 성능을 결정하는 몇 가지 중요한 요소가 있다.

(1) 디스크 용량 : 디스크 서브 시스템 이나 디스크에 저장 될 수 있는 데이터 양

(2) seek time :

디스크 헤더가 한 트랙에서 타 트랙으로 이동하는데 걸리는 시간으로 성능에 영향이 큼

(3) rotational speed : 디스크 spin 의 속도

(4) 데이터 전송 속도 : 디스크에서 host 로의 데이터 전송 속도

(5) zone bit recording :

디스크에 가변밀도로 데이터를 기록할 수 있는 기술로 높은 량을 제공하며, 디스크 드라이버의 전송속도 및 용량을 증가 시킨다

(6) bus bandwidth : 시스템 버스에 의한 데이터 전송 가능한 양

(7) controller : 호스트와 디스크 드라이버간 read/write 를 조정하는 디바이스

(8) buffer size : 디스크 콘트롤러 에 있는 cache 의 양

(9) mean time between failure(MTBF) : failure 간 평균 시간

(10) mean time to data loss(MTDL) : 데이터 손실 전 까지 의 디스크 평균 시간

3.2. 디스크의 성능 정의

호스트와 디스크 인터페이스간의 표준 (EIDE, SCSI, Fibre channel)

3.2.1. EIDE

Enhanced Integrated Driver Electronics(EIDE)는 low-end system 의 표준이며 16.7MB/sec 의 전송속도, SCSI 보다는 비교적 저렴한 비용이든다. EIDE 는 Sun 의 Ultra 5,10 에서 제공된다

3.2.2. Small Computer Systems Interface(SCSI)

SCSI 는 지난 10 년간 컴퓨터의 Interface 로 꾸준히 사용되었으며, SCSI-I 은 5MB/sec 의 전송속도를 가졌으며, SCSI-II 라 불리는 향상된 버전은 16MB/sec 의 Fast SCSI Interface 이며, 16 bit Fast/Wide SCSI Differential SCSI, Ultra SCSI 등이 있다. 16bit 는 40MB/sec 의 전송속도를 지원.

	SCSI	Fast SCSI	Fast and Wide SCSI	Ultra SCSI
Asynchronous/ Synchronous	Asynchrononous or synchronous Single-ended	Synchronouns Differential	Synchrononous Differential	Synchrononous Differential
Transfer Rate	5 MB/second	10 MB/second	20 MB/second	40 MB/second
Number of Pins	50 pins	50 pins	68 pins	68 pins
Width	8 bits	8 bits	16 bits	16 bits
Maximum Calbe Length	6 m	3m synchronous 25 m differential	3m synchronous 25 m differential	3m synchronous 12 m differential
Mumber of Targets	7	7	15	15

3.2.3. Fibre Channel

대규모 시스템과 워크스테이션, 주변장치 간 고성능의 point-to-point 방식의 통신을 제공하며, 여러 가지의 이점을 제공한다.

(1) 효과적인 고성능
(2) 확장성
(3) 장거리 동작
(4) 연결의 용이성
(5) switched 네트웍

FC-AL 의 특성

(1) Gigabit bandwidth
(2) 초당 100MB/sec 의 전송속도
(3) Suitability 네트웍
(4) Building 네트웍 을 위한 허브 및 switch 제공
(5) SCSI 프로토콜 사용

(6) SCSI 보다 장거리 지원

(7) 10KM 까지 지원 가능

(8) 고유 디바이스 명 지원

(9) 64bit World Wide Device 명 지원

3.3. RAID

Redundant Arrays of Inexpensive Disks(RAID)는 1987 년 버클리 연구소에서 처음으로 시작되었으며, 소규모의 전원으로 값비싼 디스크들의 결합을 통해 고 성능의 저렴한 디바이스를 만들고자 함 이었다. 때문에 RAID 는 다중 spindle 의 채택으로 드라이버중 하나가 Fail 시 손상되는 데이터를 어떻게 보존할 수 있느냐가 관건이었다.

디스크 서브시스템의 성능을 가름하는 두 가지의 요소가 있다.

(1) 전송속도 (Transfer Rate) :

이는 대 규모의 데이터를 read 와 write 하는 초당 전송속도를 말한다.

(2) 초당 I/O 동작 (IOPS) :

IOPS 는 주어진 시간안에 I/O 와 다중 I/O 를 처리하는 수치로, 데이터 베이스와 Transaction Processing System 은 높은 I/O Rate 를 요구한다.

(3) 복구 :

하나의 디스크가 Fail 시 Array 는 Reduced mode 로 동작하는데 이는 Array 의 전체 성능을 저하시키며, RAID Level 에 따라 사뭇 다르다. 이때 시스템 관리자는 Fail 된 디스크의 교체작업을 진행해야 하며, 정상동작을 위해 데이터의 복구가 필요하며, 어떤 시스템은 hot spare 라 불리는 미리 정의된 디스크 드라이버를 통해 자동 복구도 가능하다. Hot sparing 은 정상 성능을 내기위한 빠른 복구를 지원하며, 관리자에게 좀 더 편한 관리를 지원 한다.

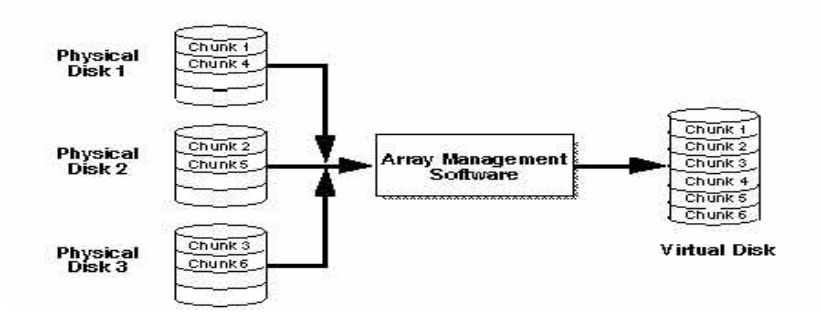
(4) 전형적인 RAID Level :

RAID 구성은 획기적으로 성능을 증대 시킬 수 있으며, 여러 가지 RAID Level 마다 특성을 가지고 있다.

3.3.1. Level 0 (Striping)

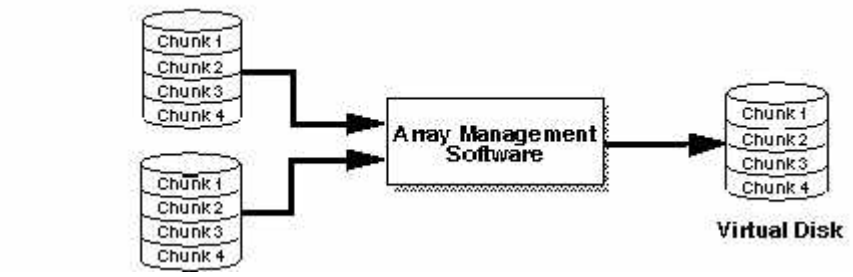
Striping 은 같은 크기의 chunk 나 stripe block 을 여러 디스크에 나누어 위치시키며, 각 chunk 는 Array 의 연속적인 드라이브에 순차적 write 가 된다.

RAID 0 는 여러 디스크에 걸쳐 I/O 의 균형을 유지하므로 우수한 IOPS 를 제공한다.



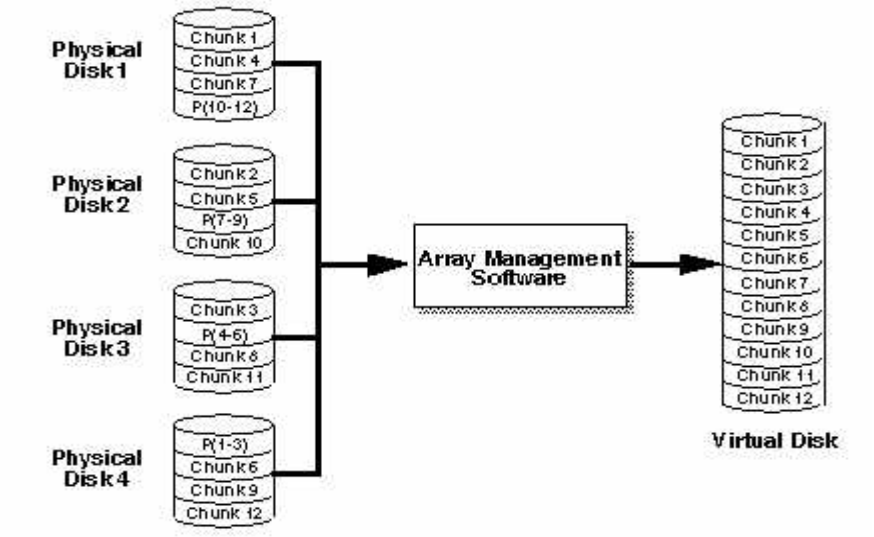
3.3.2. Level 1 (Mirroring)

각 write 마다 별도의 하나 이상의 mirror 된 디스크로 복사한다. Mirrored 시스템은 read 시 높은 IOPS 성능을 제공하며, write 성능은 개별 디스크의 write 보다는 약간 떨어진다.



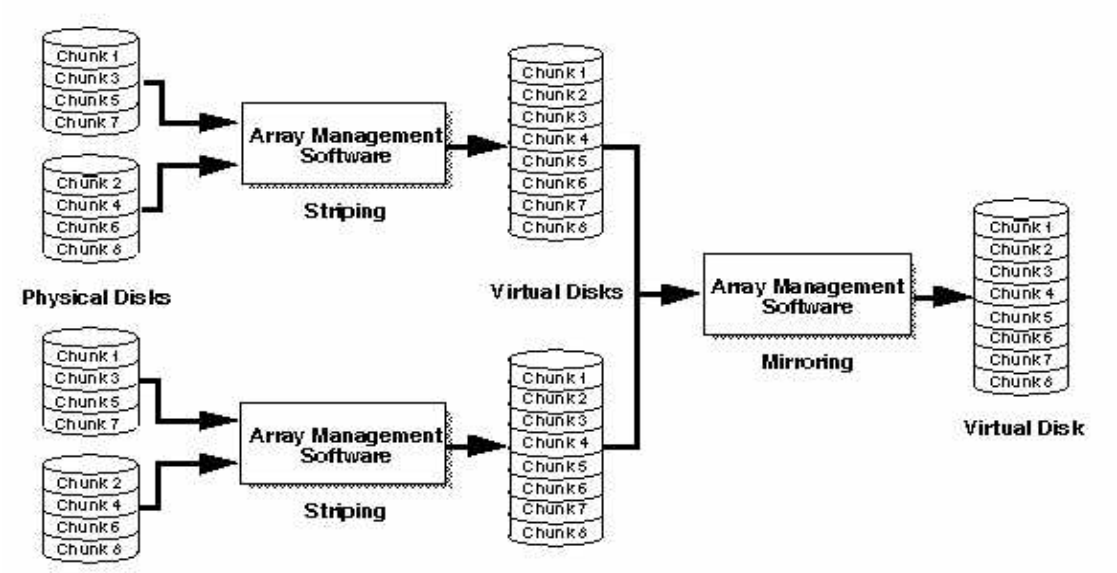
3.3.3. Level 5 (Strinping with Interleved Parity)

RAID 5 는 하나의 그룹 안에 각 디스크 드라이브에 striped 된 데이터를 저장 시키며, 데이터의 에러 검출 정보(parity)를 또한 저장 시킨다. 이는 훌륭한 sequencial 성능을 보장하며 하나의 디스크 fail 시 데이터 손실 없이 남아있는 디스크로부터 복구가 가능하다.



3.3.4. RAID 0 + 1 (striping plus mirroring)

두개의 RAID 를 조합하여 구성되며, 보편적으로 RAID 0+1 으로 불러지며, 성능적 측면의 RAID 0 와 안정적 측면의 RAID 1 을 혼용하여 사용한다.



3.3.5. RAID 구성의 최적화

RAID 는 구별되는 장/단 점을 보유하고 있으며, 여러 종류의 업무성격에 따라 선택적으로 사용 되어야 한다. 예를 들어, random 성능이 필요한 데이터 베이스 시스템, 안정성, 성능 등 다방면의 검토가 필요하다.

3.3.6. Interlace Size

chunk size 라 불리 우며, 이는 stripe 된 개별 디스크에 쓰여지는 데이터 양으로, 최적의 Interlace size 는 read/write 의 성능을 좌우할수 있는 요소이며, I/O 형태에 따라 적절한 size 가 정의되어야 한다.

(1) 높은 random I/O 가 요구될 시 interlace size 를 크게 할 것

(2) sequencial I/O 시 작은 I/O 를 사용할 것

3.4. 디스크의 성능 측정하기

디스크의 성능을 측정하기 위해 sar 명령어로 성능 정보를 수집할 수 있으니 여기서는 iostat 명령어로 설명한다. 다음은 iostat 명령의 결과를 나타낸다.

```
# iostat -x
```

extended device statistics									
device	r/s	w/s	kr/s	kw/s	wait	actv	svc_t	%w	%b
fd0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
sd0	0.9	0.1	7.2	0.4	0.0	0.0	11.9	0	0
sd1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
sd6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
ssd0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0

r/s 는 초당 read 수이고, w/s 는 초당 write 수이다. kr/w 는 초당 읽은 KByte 수를 나타내고, kw/s 는 초당 쓴 KByte 수이다. Svc_t 는 평균 I/O 서비스 시간(Milli second)을 말하는데 큐에서 대기하는 시간(Queue Time)까지가 포함된 시간이다. %b 는 디스크가 비지한 상태인 시간의 백분율로 디스크에 실행되고있는 백분율을 나타낸다.

일반적으로 svc_t 가 30ms 이하가 일반적이고, 그 이상이고 50ms 미만이며 %b 가 5% 이하일 경우도 보통의 상태이다. 하지만 50ms 이상이되고 %b 가 20% 이상일 경우는 디스크가 병목 현상을 일으킨다고 볼 수 있다. %b 는 큐가 비어있지 않는 시간의 백분율을 나타내는데, 서비스를 위해 기다리는 시간의 백분율이다. 보통 %w 가 5% 이상일 경우 디스크에 부하가 매우 많이 걸리고 있다고 할 수 있다.

- **디스크의 성능 분석을 위한 대표적인 모니터링 항목**

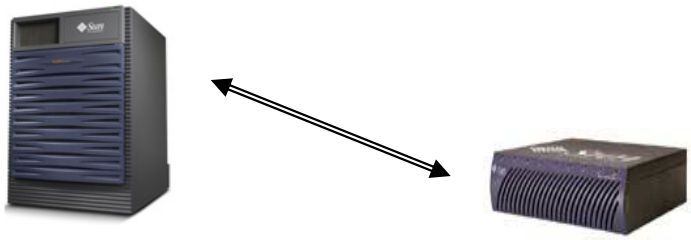
항 목 명	단 위	설 명
Kr/s(Read Size)	**KB/second**	초당 디바이스에서 발생한 물리적인 읽기(Read)를 수행한 값, 이는 어떤 디스크에 얼마만큼의 물리적인 읽기(Read)가 발생하는지를 나타내므로 디스크 사용률의 측정 기준이 되며 이 값을 기분으로 특정디스크의 읽기(Read)에 대한 부하도를 알 수 있다.
Kw/s(Write Size)	**KB/second**	초당 디바이스에서 발생한 물리적인 쓰기(Write)를 수행한 값. 이는 어떤 디스크에 얼마만큼의 물리적인 쓰기(Write)가 발생하는지를 나타내므로 디스크 사용률의 측정 기준이 되며 이 값을 기준으로 특정 디스크의 쓰기(Write)에 대한 부하도를 알 수 있다.
Svc_t(평균 Service Time)	**Msec**	디스크에 액세스한 이후 디바이스가 실질적인 서비스 작업을 완료할 때까지의 시간
Wait(평균 Wait Time)	**Mesc**	디스크에 액세스를 하지 못하고 큐에서 기다리는 시간. 디스크를 액세스하여 읽기(Read) 혹은 쓰기(Write)를 수행하려 할 때 해당 디스크가 이전명령에 의한 엑세스에 대한 읽기 혹은 쓰기가 아직 진행 중이라면 액세스를 하지 못하고 액세스 작업이 완료될 때까지 큐에서 기다려야 한다.

4. Tuning Tip (System I/O Tuning)

4.1. IOPS ?

4.1.1. IOPS 의미? (The number of I/O operations per second)

- 단위시간(기본 1초) 동안에 디스크로부터 Input / Output 한 수치.
- IOPS 를 줄이면 단위 시간 동안 많은 양의 데이터를 한번에 전송할 수 있으므로 시스템의 Response Time (응답시간)을 줄 일수 있다.



4.1.2. IOPS 와 관계되는 사항들

- bandwidth of the bus
  - bus 의 대역폭 (bandwidth)
  - scsi card 종류 (scsi1 , scsi2 , scsi3 , ultrascsi)
- amount of available memory
  - 사용가능 한 memory 양
- Capacity of th cache
  - memory cache 의 데이터 보유 양
  - buffer cache, DNLC, inode cache size
- speed of the disks
  - seek time, latency time (RPM)
  - 적절한 DISK RAID 정책

성능관리를 시작하기 전에 분명히 규정하고 넘어가야 할 것들이 있는데, 바로 Performance Tuning 을 위한 용어 정의이다. 용어에 대한 이해가 선행되지 않으면 개념적으로 난해한 요소들에 부딪힐 수 있기 때문이다. 그 중에서 몇 가지 핵심적인 용어의 의미를 명확히 정의하여 보자. 이 용어들은 용량 관리(Capacity Planing & Sizing) 분야의 이해를 위해서라도 꼭 이해

하고 있어야 한다.

● 성능관리 분야의 용어 정리

♦ Bandwidth :

* 초과될 수 없는 최상의 capacity
* Overhead 를 무시한 측정치
* The best(perfect) case number
* 실행시 도달할 수 없는 어떤 값(실행시 inefficiency(무능, 무효과, 비능률), Overhead 가 있기 때문)

♦ Throughput :

* 특정시간 동안 실제 할 수 있는 작업의 량(what you really get)
* Bandwidth 중 실제로 사용되는 capacity
* 실제 Throughput 의 측정값은 다양한 요소들에 의해 영향 받는다.(H/W, S/W, human, Random access)
* 정확한 Throughput 값의 측정은 불가능하며, 단지 근사치(Approximated value)만을 구할 수 있다.
* Maximum Throughput 을 측정한다면 100% 사용율(utilization)에서 얼마나 많은 작업량이 가능한가를 평가 하는 것이 된다.

♦ Response Time :

* 총 경과시간 (+ wait time 포함, + Run Queue 대기시간)
* user 가 응답을 받기까지 걸린 총 시간
* Elapsed Time for an operation to complete
* Usually the same thing as latency

♦ Service Time :

* 실제 request 를 처리하는데만 사용된 시간(Queue 대기시간, Wait Time 제외) = actual processing time
* The best(perfect) case number
* Response Time 에서 Wait Time(Queue Delay)을 빼면 Service Time 이 된다.

♦ Utilization :

* 측정대상 작업을 수행하기 위해 사용된 자원의 사용량(독자적이건, 종합적이건)
* 자원량 : Throughput 을 수행하기 위한
* 백분율(%)로 표시되면 일반적으로 busy%(률)로 나타낸다.

이 용어들은 서로 상관관계를 가지고 있는데, Utilization 이 100%에 도달하는 순간이 되면 Response Time 은 빠르게 되지만(자원의 이용방법에 따라 다르다), Troughput 은 포화상태에 도달하여 떨어지게 되어 있다. 이 시점의 Throughpu 을 "Drop-Off"라고 한다. 결국, Performance Tuning 은 100% Busy 상태에 도달한 자원(Drop-Off 된 자원)의 소재를 찾아서 그 가용성을 높여 주는 것이 목표이다.

4.2. Reduce IOPS method

- IOPS 를 줄일 수 있는 방법
  - 한번의 I/O 발생시 많은 양의 DATA 를 읽고 쓰기 함.
- read ahead & write behind 기능 사용
  - UFS cache 를 tuning 하여 cache-hit 율을 증가 시킴.
- 적절한 RAID 정책을 설정함.
  - strip size & cluster size 를 설정함.

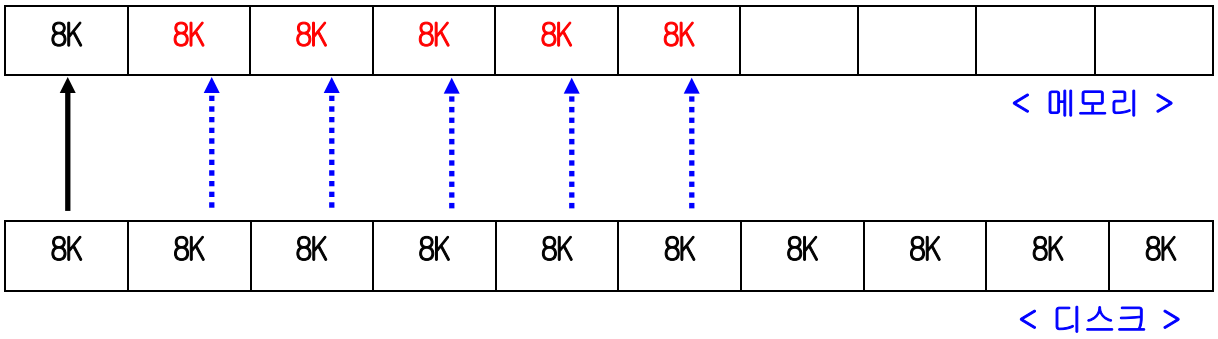
4.3. File System read ahead

Read ahead 라는 것은 현재 시점에서 파일의 사용이 미리 있을 것 이다는 것을 염두에 두고 파일에서 일정 내용을 사용되기 전에 미리 읽어오는 것이다. 이것은 시스템이 디스크에 대한 연산을 줄이는 효과가 있으며, 이렇게 함으로서 전체적인 I/O 의 성능을 개선할 수 있다. 물론 이것은 전적으로 커널에서 행해야 하며, 사용자 프로세스는 모른다. 물론 이것을 하기 위해선 커널의 메모리를 사용하게 되기에 비효율적으로 관리하면 오히려 역효과가 날 수 있다. 나중에 사용자 프로세스는 다시 파일에 대한 연산을 하려고 할 때 이곳에서 데이터를 가져올 수 있게 된다. 따라서, I/O 가 일어나길 기다릴 필요 없이 미리 읽어온 데이터를 가지고 연산을 계속 할 수 있게 되는 것이다.

이 튜닝은 ufs 에만 해당하면 zfs 인 경우에는 해당하지 않습니다. 또한, 이 옵션은 SATA 와 SAS 디스크의 경우에는 해당되지 않는다. 동시에 동시 전송 블럭의 수를 늘렸기 때문에 물리적인 디스크와 사용하게될 버퍼의 크기는 1M(=1048576)으로 설정할 필요가 있다.

1) 파일시스템의 기능 중에서 데이터를 읽어 오는 방식 중 미리 읽기 기능이 있다. 이것을 설정하면 한 block 을 읽어올 때 여러 block 을 미리 읽어오는 기능 이 있다.

Ex) Read ahead 를 '6' 으로 설정한 예제



2) 설정 방법은 UFS file system setting 값 중에서 maxcontig 값을 설정.

3) Maxcontig 값의 단위는 1 block (8192byte)

4) fstyp 명령을 이용하여 maxcontig 정보를 확인.

```
Ex) root# fstyp -v /dev/rdsk/c0t0d0s0  |  grep maxcontig
  ➔ maxcontig 6    rotdelay 0ms    rps     120
```

5) tunefs 명령이나 newfs 명령을 사용하여 설정

```
Ex) root# tunefs  -a  128  /dev/rdsk/c0t0d0s0
  ➔ maximum contiguous block count changes from 6 to 128
    root# newfs  -C  128  /dev/rdsk/c0t1d0s3
```

% 128 block 이므로 128*8192 = 1,048,576byte 의 데이터를 한번에 미리 읽어오게 된다.

➔ maxcontig 값이 '6' 일때와 '128' 로 변경 후 I/O 성능을 비교

- Maxcontig 값 설정 전

```
root# fstyp -v /dev/rdsk/c0t0d0s0 | grep maxcontig
maxcontig 6     rotdelay 0ms    rps     120
root# cp 500m /ARCH1/500m &
[1]     20376
root@SVRDB1 # iostat -x 5
                    extended device statistics
device       r/s    w/s   kr/s   kw/s wait actv  svc_t  %w  %b
sd0        492.4    0.0 10543.8    0.0  0.0  0.5    1.1   1  48
sd1          0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
sd6          0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
ssd0         0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
```

```
ssd1        0.0    0.0    0.0    0.0  0.0  0.0    0.0    0    0
ssd2       13.0   31.0  120.5 20650.3  0.0  6.0  137.8    2  100
```

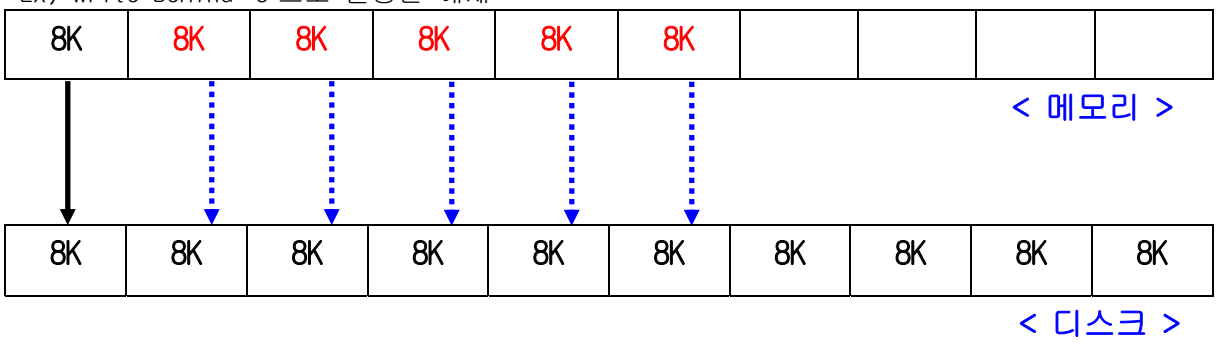
- Maxcontig 값을 128 로 설정 후

```
root# tunefs -a 128 /dev/rdsk/c0t0d0s0
maximum contiguous block count changes from 6 to 128
root# cp 500m /ARCH1/500m &
[1]     8416
root@SVRDB1 # iostat -x 5
                  extended device statistics
device       r/s    w/s   kr/s   kw/s wait actv  svc_t  %w  %b
sd0         95.5    0.0 15047.4    0.0  0.0  0.5    5.0   0  47
sd1          0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
sd6          0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
ssd0         0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
ssd1         0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
ssd2         5.0   34.0   30.2 21879.9  0.0  6.2  159.4   1  99
```

4.4. File System Write Behind

1) 파일 시스템의 기능에는 데이터를 메모리 버퍼에 임시저장 후 한번에 저장하는 기능이 있다. 이 기능을 Write Behind 라고 한다. 연속적으로 데이터를 기록하는 작업 환경 에서는 filesystem 의 Write Behind 기능을 설정하여 IOPS 양을 줄일 수 있다.

Ex) Write Behind '6'으로 설정한 예제



2) 설정 방법은 UFS filesystem setting 값 중에서 maxcontig 값을 설정.

3) Maxcontig 값의 단위는 1 block (8192byte)

4) fstyp 명령을 이용하여 maxcontig 정보를 확인.

```
Ex) root# fstyp -v /dev/rdsk/c0t0d0s0  |  grep maxcontig
➔ maxcontig 6     rotdelay 0ms     rps      120
```

5) tunefs 명령이나 newfs 명령을 사용하여 설정.

```
Ex) root# tunefs  -a  128  /dev/rdsk/c0t0d0s0
➔ maximum contiguous block count changes from 6 to 128
   root# newfs  -C  16  /dev/rdsk/c0t1d0s3
```

** 128 block 이므로 128*8192 = 1,048,576byte 의 데이터를 한번에 저장 하도록 설정한다.

➔ maxcontig 값이 '6' 일때와 '128' 로 변경 후 I/O 성능을 비교

- Maxcontig 값 설정 전

```
root# fstyp -v /dev/rdsk/c0t0d0s0  |  grep maxcontig
maxcontig 6     rotdelay 0ms     rps      120
root# mkfile 500m 500m &
[1]     29699
root# iostat -x 2
                    extended device statistics
device       r/s    w/s   kr/s   kw/s wait actv  svc_t  %w  %b
sd0          0.0  141.0    0.0 6743.3  7.5 168.2 1246.4  33 100
sd1          0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
sd6          0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
ssd0         0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
ssd1         0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
ssd2         8.0    4.0   32.5    9.5  0.0  0.1    5.6   0   7

real    1m26.18s
user    0m0.11s
sys     0m5.21s
```

- Maxcontig 값을 128 로 설정 후

```
root# fstyp -v /dev/rdsk/c0t0d0s0  |  grep maxcontig
maxcontig 6     rotdelay 0ms     rps      120
root# tunefs  -a  128 /dev/rdsk/c0t0d0s0
```

```
maximum contiguous block count changes from 6 to 128
root# mkfile 500m 500m &
[1]     10920
root@SVRDB1 # iostat -x 2
                    extended device statistics
device       r/s    w/s   kr/s    kw/s wait actv  svc_t  %w  %b
sd0          0.0   38.5    0.0 25411.4  0.0 13.6  353.6   0  99
sd1          0.0    0.0    0.0     0.0  0.0  0.0    0.0   0   0
sd6          0.0    0.0    0.0     0.0  0.0  0.0    0.0   0   0
ssd0         0.0    0.0    0.0     0.0  0.0  0.0    0.0   0   0
ssd1         0.0    0.0    0.0     0.0  0.0  0.0    0.0   0   0
ssd2        10.0    5.5   56.5    25.7  0.0  0.1    5.3   0   8


real    0m27.59s
user    0m0.14s
sys     0m5.77s
```

➔ Maxcontig 값을 '128'로 변경 후 3 배 이상의 I/O 성능이 향상 되었다

** 단 위의 명령중 newfs -C 명령은 시스템 운용중에는 불가능하며 새로운 파일시스템을 구성시에 적용 할 수 있다.

➔ Ex) Oracle Datafile 추가 예제

```
→ maxcontig 값이 '6'
SQL> alter tablespace TEST1 add datafile '/test/test1_2.dbf' size 200m;
Tablespace altered.
Elapsed: 00:00:28.55

→ maxconfig 값이 '128'
SQL> alter tablespace TEST2 add datafile '/test/test2_2.dbf' size 200m;
Tablespace altered.
Elapsed: 00:00:09.06
```

4.5. Kernel Parameter Tuning

Read ahead 와 write behind 기능과 관련된 kernel parameter

- Read ahead & Write Behind 를 설정 하였다 하여도 scsi 의 최대 전송 size 가 작다면 scsi 전송용량 size 보다 더 많은 데이터는 전송하지 못한다. 예를 들어 scsi 의 최대 전송용량이 128k 인데 maxcontig 값이 512k 라면 데이터는 128k 이상 전송 되지 않는다.
- RAID 0 (striping) : 클러스터 크기는 스트라이프 크기에 의한 여러 개의 스트라이프 멤버 수와 같다.
- RAID1 (mirroring) : 클러스터의 크기는 디스크의 절반과 같다.
- RAID0+1 (striping + mirroring) : 클러스터의 크기는 스트라이프 크기에 의한 여러 개의 미러당 스트라이프 멤버의 수와 같다.
- Solstice DiskSuite 와 Veritas Volume Manager(VxVM)과 같은 볼륨을 사용자가 이용할 때, 매우 큰 클러스터의 크기를 요구하면, 큰 전송을 하기 위해 사용자는 /etc/system 커널설정 파일에서 SCSI 나 Volume manager 관련 파라메터를 설정할 필요가 있다.
- 아주 큰 SCSI I/O 전송을 하기 위해는(byte 단위)
  set maxphys = 1048576
- 아주 큰 Solstice DiskSuite 전송을 하기 위해서는(byte 단위)
  set md_maxphys = 1048576
- 아주 큰 VxVM I/O 전송을 위해서는(블록 단위 : 512byte 단위)
  set vxio:vol_maxio = 2048 로 설정 할 수 있다.

maxphys 는 상당히 오래된 커널 변수인 관계로 최신 디스크나 어레이등을 사용 할 때는 반드시 사용하는 것이 좋습니다.
#echo "set maxphys=1048576" >> /etc/system
설정하고 재부팅이 필요합니다. 이 튜닝은 상위 화일 시스템과 상관없이 사용 하는 것이 좋습니다.

4.6. Write Behind & read ahead integrated

Read ahead 및 write behind 설정 시 반드시 filesystem 의 Workload (작업 형태)를 확인 한 후에 설정을 하여야 한다.

- 연속된 큰 파일을 읽어 들이는 작업 형태일 경우 (예: Database) 많은 수의 block 를 미리 읽기로 설정 하면 성능을 높일 수 있다.
- 비 연속된 작은 파일의 I/O 가 발생하는 작업 형태일 경우 (예: web, mail) read ahead 나 write behind 의 기능을 정지 하여야 한다.

예를 들어 한번에 I/O 시 1kbyte 사이즈 보다 작은 파일을 읽어 들이는 작업 형태에서 read ahead 를 128 로 설정 했다면 필요하지 않은 1,048,576byte 의 데이터를 미리 읽어 오기 때문에 오히려 I/O 시간이 늘어나고 메모리를 낭비하는 결과를 갖는다.

** 위의 ufs 튜닝 옵션들은 모두 서비스 제공중에 설정가능한 것이나, 최적의 성능을 위해서라면 화일시스템을 새로이 구성하시는 것이 바람직합니다

5. Cache Management

5.1. DNLC (Directory Name Lookup Cache)

- 최근에 참조 했던 directory name 과 vnode 를 기억하고 있는 cache 이다.
- directory name 검색 시 filesystem 의 inode 정보와 directory data block 의 정보를 읽어 오기 때문에 disk 의 I/O 가 발생한다. 최근에 검색 했던 directory name 을 메모리에 cache 하여 중복된 directory 내용 검색 시 메모리의 cache 에서 directory 정보를 찾도록 하여 disk 의 I/O 를 줄일 수 있다.
- 'vmstat -s' 으로 모니터링 시 DNLC cache hit 율이 90% 이상이 되도록 함.
  (만일 90% 이하의 값이면 DNLC cache size 를 증가 하여야 한다.)
- Ufs, nfs 에서 사용된다.
- 커널의 파라미터명은 ncsize 이다.

➔ vnode 란?

- 다양한 파일시스템에 대한 구조체 제공
- 여러 종류의 파일 시스템에 대하여 같은 시스템 호출 인터페이스 사용
- 서로 다른 파일 시스템이 하나의 계층적 논리적인 디렉토리 목록으로 사용 가능하도록 제공
- Vnode (Virtual node)는 하나의 파일을 나타냄
- 다수의 파일 시스템을 동시에 제공
- 디스크 파티션마다 파일 시스템 타입이 다를 수 있다.
- 네트워크 상에서 파일 공유를 지원한다.
- 새로운 파일 시스템 타입을 쉽게 생성 추가 가능하다.

➔ 파일에 접근 하면 접근 했던 기록이 아래의 그림처럼 memory 내의 특정 영역에 cache

되어 VNODE number 와 file 의 full path 이름이 기록 된다.

< DNLC Cache >

Full path 명	Vnode 번호
/usr/bin/ksh	00000300024f8090
/usr/bin/csh	00000311024f9091
/etc/vfstab	00000323024f5900
/etc/shadow	00120311033f9541

➔ mdb utility 를 이용하여 현재 시스템의 DNLC 정보를 확인한다.

```
# mdb -k
Loading modules: [ unix krtld genunix ip usba logindmux ptm md cpc random nfs ]
> ::dnlc
VP                DVP               NAME
00000300018960a0 000003000189be78 d46
00000300016a60a0 0000030000d497b0 md@0:1,125,raw
00000300018080a0 000003000180d128 d23
00000300017780a0 0000030001778288 S01MOUNTFSYS
00000300018a60a0 0000030000d497b0 pts@0:4
.
.
.
00000300017e2110 00000300017f6a60 d116
000003000105e118 0000030000c92e18 mount
000003000186e118 0000030000d620c8 cfg
> 000003000105e118::vnode2path
/sbin/mount
> 0000030000c92e18::vnode2path
/sbin
> ::quit
```

➔ DNLC 는 kernel parameter 의 'ufs_ninode' 이며 기본값은 아래와 같다.

- ncsize = 4 x (maxusers + max_nprocs) + 320
- 설정 방법은 /etc/system 파일에 아래와 같이 하면 된다.

```
#vi /etc/system
set ncsize=4000
set ufs_ninode=4000
#reboot
```

5.2. Inode Cache

- Open 한 Inode 목록을 포함
- DNLC 를 이용하여 디렉토리 목록의 위치 참조
- Inode cache 와 DNLC 는 매우 유사하다.
- DNLC 의 모든 목록은 inode cache 의 목록이 된다.
- inode 목록을 빠르게 찾는 것은 디스크를 다시 읽을 필요가 없기 때문이다.
- DNLC, inode cache 는 LRU(Least Recently Used)를 기본으로 하여 관리된다. 오래된 목록은 새로운 목록으로 교체를 한다.

5.3. DNLC 와 Inode Cache 관리

- DNLC 와 inode cache 들은 같은 크기를 가진다. (DNLC 목록들은 inode 목록을 가진다.)
- 기본적으로 4 * (maxusers + max_nprocs) + 320 의 크기를 가진다.
- Inode cache 크기는 ufs_ninode 에 의해 설정
  - NFS 서버의 경우 높게 설정(50,000 이상)
  - DNLC Hit 비율은 90% 이상 유지
- DNLC 와 inode cache 들은 DNLC 목록이 inode cache 목록을 가지기 때문에 같은 크기를 가진다.
- Inode cache 로부터 삭제된 목록은 DNLC 에서도 같이 삭제 된다.
- Inode cache 는 오직 UFS(inode) 데이터에 대해서만 cache 를 하고, DNLC 는 NFS(rnodes)와 UFS(inode) 데이터에 대해서만 cache 를 하기 때문에 NFS 와 UFS 동작 비율은 ncsize 가 수정이 될 때 좌우된다.

➔ Boot 후부터 현 시점까지 DNLC cache hit 율 확인(hit 율이 90% 이하이면 DNLC cache 의 부족을 판단함.)

```
# vmstat -s
        0 swap ins
        0 swap outs
        0 pages swapped in
        0 pages swapped out
    46973 total address trans. faults taken
        .
        .
        .
  1318405 system calls
```

```
126424 total name lookups (cache hits 96%) → 90% 이하이면 ncsize 값 증가
   2196 user   cpu
   4482 system cpu
7863209 idle   cpu
   2753 wait   cpu
```

6. 결 언

- UFS 파일 시스템상의 I/O 튜닝을 소개 하였습니다. 실질적으로 나타나는 증상별로 해결방안을 찾아야 하는 어려움이 있으나 일반적으로 사용되는 파일시스템의 내부구조와 커널의 파라메터등을 소개함으로써 단순히 운영체계를 설치후 사용하던 관리자들에게 새로운 사항들을 접할 수 있는 기회라 생각되어 정리하였습니다.
- 자신의 작업환경을 고려해 성능을 스스로 평가해보는 것이 가장 좋은 튜닝 방법이라 할 수 있고 여기에 소개된 튜닝에 대한 방법들은 페이퍼에서 흔히 볼 수 있는 조언들입니다.
- 시스템 관리자들에게 뚜렷한 해결방안을 제시해 주기보다는 소개된 사항들을 데이터 집약적 환경설정에서 성능을 향상시키고자 하는 분들에게 매우 유익하다고 볼 수 있습니다.
- Solaris8 이전 버전을 사용하고 계시는 관리자들에게 필요한 튜닝 방법과 solaris8 이상을 사용하고 계시는 관리자 분들과는 구분되는 부분이 있을 수 있으니 이점 주의하시길 바랍니다.

참조문서

http://docs.sun.com
http://sunsolve.sun.com
Sun Performance Guide / Sun Press / 2004

7. Pro-Active Tuning 성공사례

고객사

000 상담원

구축기간

2007.07.19 ~ 2007.07.26

고객사 이슈 및 구축 배경

1. Application 에서 요구하는 데이터 처리(SQL) 프로그램의 Access 가 비효율적인 패턴을 보임.
2. 고가용성 시스템으로 구축되지 않음.
3. RDBMS 가 개발단계부터 통계정보가 없음
4. 업그레이드 된 서버의 자원(CPU, Memory, Disk I/O)에 맞게 DBMS 가 최적화 되지 않음.
5. Index 전략 부재 (결합인덱스 전략 없이 Single Index 로 구성된 사례 많음)
6. 향후 데이터가 늘어날 경우를 대비한 부분범위 처리 형태의 SQL 전략 없음.

고객사 이슈 해결 방안

효율적인 access 패턴을 위한 sql tuning 및 index tuning 진행

rule base 사용 기반으로 하는 tuning 계획 수립진행

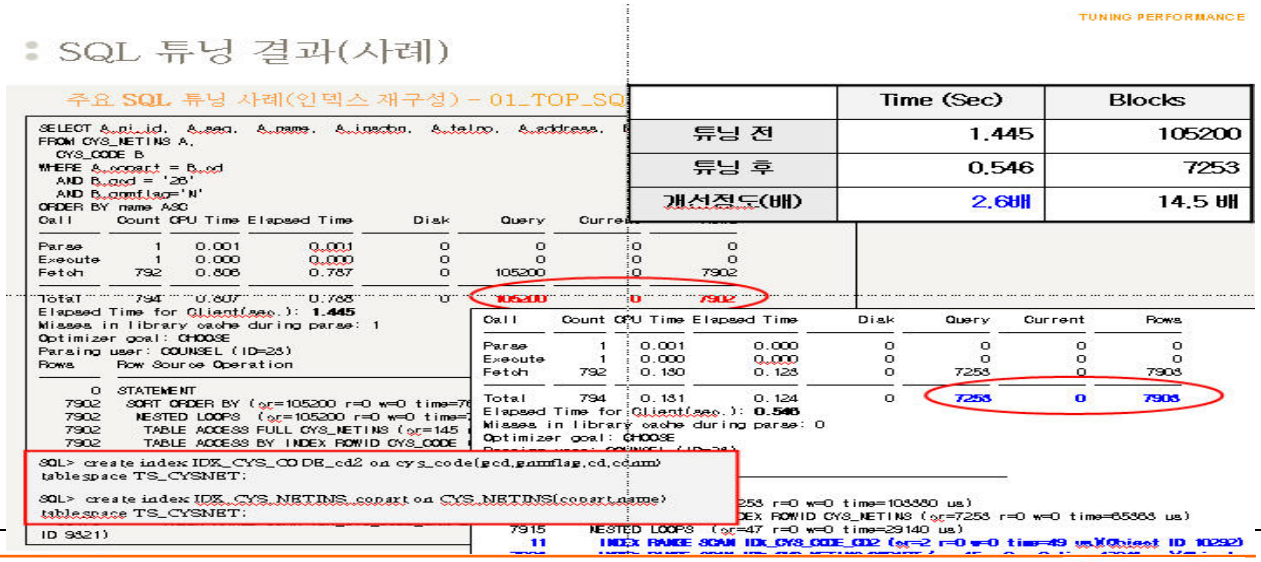
부분범위 처리형태로 프로그램(SQL) 튜닝

로그 성 데이터의 주기적인 Cleansing 을 위한 Range partition 의 검토와 parallel processing 에 대한 검증 및 도입이 요구됨

구축후 개선점

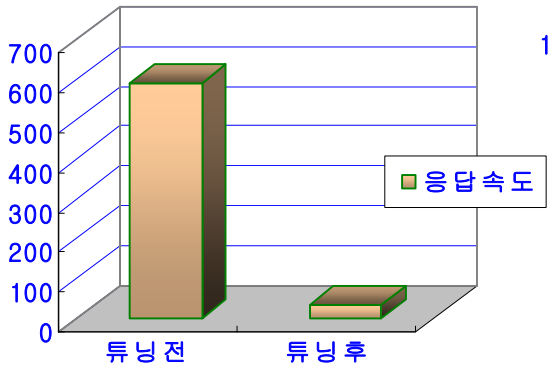
각종 wait event 비율이 현저히 감소하였으며 CPU 자원 사용량 감소와 불필요한 DISK I/O 제거를 통하여 database 전반에 걸친 resource 사용량 감소, system performance 향상에 영향을 주었습니다.

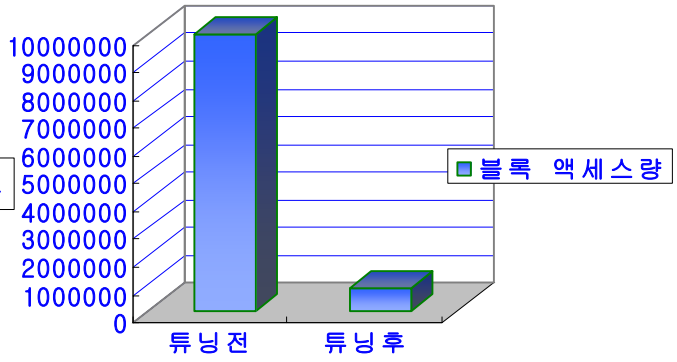
- SQL Tuning 사례



- 프로그램/SQL 들의 튜닝 전후 개선율 비교

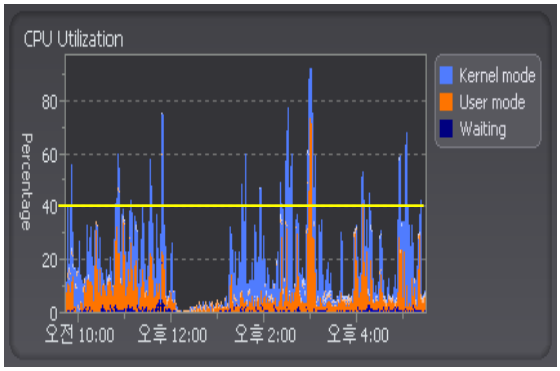
튜닝전 시간 합계	튜닝후 시간 합계	개선율	튜닝전 블럭액세스량 합계	튜닝후 블럭액세스량 합계	개선율
590.997 초	33.17 초	17.8 배	37,539,159 개	841,817 개	44.5 배



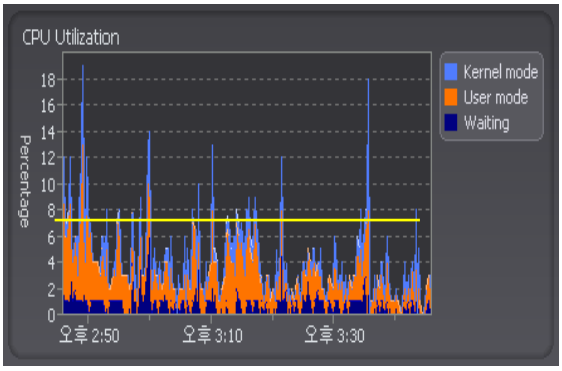


CPU 사용량

Tuning 전 (2007.07.19)



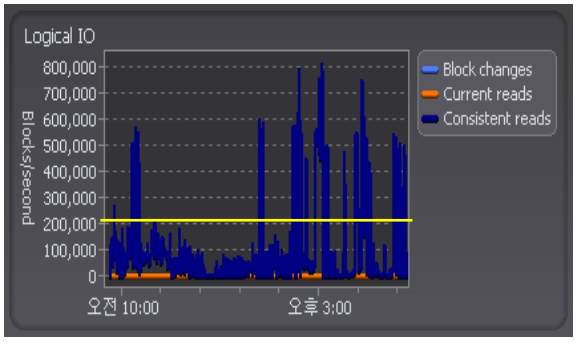
Tuning 후 (2007.07.26)



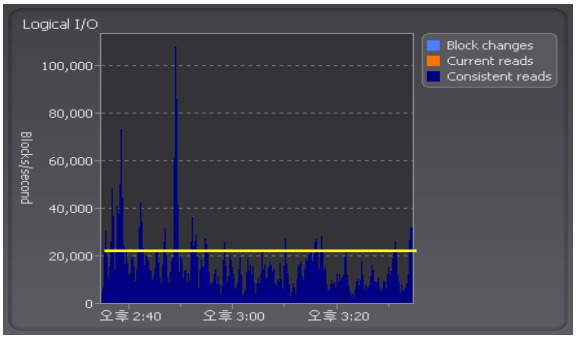
■ **튜닝전에는 최고 95% 의 CPU 사용을 나타내나, 튜닝후에는 20% 미만의 사용량을 나타 냄.**

Logical I/O

Tuning 전 (2007.07.19)



Tuning 후 (2007.07.26)



■ **튜닝전에는 최고 초당 800,000 blocks 를 scan 하였지만 튜닝후에는 최대 110,000 blocks 미만을 나타냄. 튜닝후 평균적으로 약 10 배(200,000 : 20,000) 성능 향상 보임.**

■ 현재 데이터 증가추이로 볼때 향후 **3** 년 정도는 무난히 사용할 수 있을 것으로 보임. 또한 추가적 지속적인 **tuning** 시 **system(hard ware) Life Time** 을 **5** 년까지 유지시킬 수 있음.

그러나 데이터가 현재보다 **2** 배 이상으로 증가하거나 또는 대량의 데이터를 가공하여 통계를 생성하는 작업이 많아진다면 **I/O** 성능개선은 꼭 필요할 것으로 보인다.