[Chapter 11]

소프트웨어융합학과/2020105742/한지훈

1. Disk Attachment
   A. Host attached: 일반적인 방식(버스에 I/O 포트로 직렬 연결)
   B. Network attached: HW 통해 연결 X, 외부 네트워크 통해 원격으로 연결
      I. NAS(Network-Attached Storage)
         - 개요: 기기에 하드디스크를 연결하여 외부 접근이 가능하도록 함
         - 특징: data를 storage에 저장할 때(read/write 등) file 단위로 access
            - 가능한 이유: NAS도 하나의 별도 컴퓨터이며, 설계상 별도의 file system을 가지고 있음
      II. SAN(Storage-Area Network)
         - 개요: NAS에 비해 대규모의 개념, 전체가 하나의 abstract된 storage라고 볼 수 있음 <-> NAS file system
         - 특징:
            - 시스템에 종속적으로 따라가기 때문에 block 단위로 I/O처리
            - 다수의 HDD간 물리적 연결 필요하기 때문에 인터페이스 빨라야 함
               - SAN 네트워크는 사용하는 file system 따로 있음(GFS)
   C. NAS vs SAN
      I. NAS: IP 네트워크 기반, file 단위로 access
      II. SAN: IP 사용 X, block 단위로 access

2. HDD
   A. 개요: SSD 쓰면 좋지만 HDD의 효용성이 높음(신뢰성/속도 높음, 가격 낮음)
   B. 구성: arm(동시에 움직임), sector(저장의 단위), track(원 상의 sector들), cylinder(track들의 집합), platter 등
   C. 장점: fairness(일단 큐에 들어오면 언젠가는 반드시 실행된다고 볼 수 있음)
   D. 단점: HDD마다 스펙이 달라서 OS가 다 고려하기 어려움
      - 논리적 단위(block)로 처리, controller가 변환해줌
   E. Arm seek: arm 이동 속도가 느리기 때문에 bottleneck이 매우 큼
      - Disk scheduling(arm 이동이 최소가 되도록 디스크 request의 처리 순서를 결정)

3. Disk scheduling 방법들
   A. FCFS
      I. 개요: 디스크 request가 들어오는 순서대로 처리하는 방식
      II. 문제: 직관적으로 보더라도 arm seek time이 매우 커지기 때문에 비효율적이라는 것을 알 수 있음
   B. SSTF(Shortest Seek Time First)
      I. 개요: 현재 head랑 가장 가까운 request부터 처리하는 방식
      II. 가능한 이유: arm으로부터의 거리는 충분히 예측할 수 있음
         - vs CPU의 SJF: 프로세스가 얼마나 빨리 끝날지 예측할 수 없기 때문에 불가능한 개념
      III. 장점: arm search 폭이 감소하여 성능이 증가함
         - HDD에 실제로 쓰이는 방식
      IV. 단점: request는 여러 개지만 controller는 1개이기 때문에 head로부터 먼 애들은 strarvation 발생 가능
   C. SCAN
      I. 개요: 방향을 정해놓고(증가/김소) 가다가 경계를 만나면 방향 전환
      II. 장점: starvation이 발생하지 않음
      III. 단점: 어떤 request는 wait를 많이 하고, 어떤 request는 wait를 적게 함
         - 성능 변동 폭이 큰 편

D. C-SCAN

I. 개요: 일단 증가하는 방향으로 request들을 수행하고, 경계를 만나면 0(최솟값)으로 땡기고 다시 증가

II. 장점: SCAN에 비해서 request wait time이 더 일관성 있음

III. 단점(SCAN, C-SCAN 공통): arm 속도 느린데 쓸데없이 양 끝까지 감

E. LOOK/ C-LOOK

I. 개요: 각각 SCAN/ C-SCAN의 업그레이드 버전으로, 큐에 더 큰 디스크 request data다 없으면 바로 turn

II. 장점: SCAN/ C-SCAN에 비해 arm 이동 거리를 줄일 수 있음

F. 실제로 쓰이는 방식들

I. 일반적인 경우: SSTF가 쓰임

II. 디스크에 큰 부하를 주는 시스템: SCAN/ C-SCAN이 쓰임

4. Disk Controllers

A. 개요

I. 초기의 Disk controller는 기능이 적었고, HW를 시스템에서 그대로 다루는 경우가 많았음

- 현대에는 복잡한 HW를 시스템이 그대로 다루기 어려움
- HW(디스크)를 abstraction하여 OS가 사용할 수 있음

B. Intelligent features

I. Read-ahead: spacial locality 확보를 위함(플래터가 도는 김에 읽을 sector 이후까지 읽어서 locality 확보)

II. Cashing: 자주 사용하는 블록들을 캐싱하여 성능을 높임

III. Request reordering: 프로세스별 request들에 대한 seek를 최적화하는 방법

IV. Bad block identification: bad sector 등 HW상 발생할 수 있는 문제들을 관리해줌

C. Swap-Space Management

I. 윈도우: swap space를 file 단위로 관리하기 때문에 크기 조절이 가능함

II. 리눅스: swap space를 파티션 단위로 나눠 여러 파티션에 설치하기 때문에 크기 조절이 어려움

5. RAID(Redundant Array of Inexpensive Disks)

A. 개요

I. HDD는 용량이 그렇게 크지 않고, 쉽게 깨지기 때문에 신뢰성이 낮은 저장 매체임

- 하지만 가격이 싸다는 장점이 있음
- 신뢰성과 성능을 높이기 위해 싼 디스크들을 배열로 묶어서 사용하자는 개념

B. 신뢰성을 높이는 방법

I. Mirroring(클론)

- 개요: 디스크(들) data를 동일 갯수의 디스크(들)에 완전히 똑같이 복사
- 가능한 이유: 디스크 가격이 싸기 때문에
- 장점: 디스크 1개가 망가져도 대체제가 있기 때문에 괜찮음
- 단점: 예를 들자면 가지고 있는 2TB 중에서 1TB밖에 사용하지 못함

II. Parity bit / error-correcting code

- 개요: 디스크 4개당 1개 정도 디스크를 따로 두고, 4개 디스크 중 일부 파괴시 해당 디스크로 복구
- 장점: Mirroring에 비해서 저장 공간을 적게 사용할 수 있음
- 단점: Mirroring과 달리 파괴 data 복구시 computing resource 사용해야 함

III. 가능한 선택들

- 신뢰성 매우 높이고 싶은 경우: Mirroring 사용, Parity bit / error-correcting code과 동시도 가능
- 신뢰성을 적당하게 가져가고 싶은 경우: Parity bit / error-correcting code 사용
- 신뢰성 신경 안쓰는 경우: 2가지 다 사용 안하면 됨

C. 성능을 높이는 방법

I. Striping

- 개요: 파일 A.dat을 디스크에 저장 시 여러개 디스크에 나눠서 저장하면 write/read(병렬) 성능 증가
  - 쪼개는 방법(Striping)을 고민해볼 수 있음(block-level / bit-level)

II. block-level striping

- 개요: 여러 개 디스크에 나눠 저장할 때 block 단위로 나눠서 저장

III. bit-level striping

- 개요: 여러 개 디스크에 나눠 저장할 때 bit 단위로 나눠서 저장

IV. 가능한 선택들

- 성능 매우 높이고 싶은 경우: block-level striping 사용
- 성능 적당하게 가져가고 싶은 경우: bit-level striping 사용
- 성능 신경 안쓰는 경우: 2가지 다 사용 안하면 됨

D. Disk Controller vs RAID Controller

I. Disk Controller: 디스크별로 따로 있다고 가정하여 관리

II. RAID Controller: 물리적으로 여러 개 디스크지만 논리적으로 1개 storage인 것처럼 가정하여 관리

6. RAID Levels

A. RAID 0

I. 개요: 신뢰성을 포기하고, block-level striping을 사용하는 RAID

II. 장점: 성능이 높음(예제 기준 4배의 성능 상승)

III. 단점: 디스크 1개만 깨져도 전체를 다 쓰지 못하게 됨

B. RAID 1

I. 개요: 신뢰성을 높이기 위해 디스크 Mirroring을 사용, striping 사용하지 않는 RAID

II. 장점: 신뢰성이 높으며, 성능도 좋을수도 있음(read를 병렬로 수행하면 2배 성능 상승 기대) – 일반적으로 X

III. 단점: 전체 디스크 중에서 실제로 사용할 수 있는 디스크 수는 절반밖에 되지 않음

C. RAID 2

I. 개요: 신뢰성을 높이기 위해 error-correcting code를 사용, bit-level striping을 사용하는 RAID

II. 장점: hamming code를 사용하여 data 양 많아서 computation 감소, 디스크 여러 개 망가져도 복구 가능

- 즉 hamming code가 parity bit보다 좋다고 볼 수 있음

III. 단점: block에 비해서 bit-level striping은 bit 단위로 읽고 쓰기 때문에 spacial locality를 확보할 수 없음

- Block-level striping은 spacial locality를 확보하고, 성능/관리 측면에서도 좋아 실제로 사용됨

D. RAID 3

I. 개요: 신뢰성을 높이기 위해 parity bit를 사용, bit-level striping을 사용하는 RAID

II. 장점: RAID 2에 비해 신뢰성에 쓰이는 디스크 수를 줄일 수 있음(3 > 1)

III. 단점: RAID 2에 비해 디스크 1개 망가지면 복구 가능, but 2개 이상 망가지면 복구 불가

IV. 특징:

- parity bit 동작을 위해서는 사용 디스크 최소 4장 필요
- 성능과 신뢰성의 trade-off 개념으로 볼 수 있음

E. RAID 4

I. 개요: 신뢰성을 높이기 위해 parity bit를 사용, block-level striping을 사용하는 RAID

II. 장점: RAID 3에 비해 어느 정도의 신뢰성 뿐만 아니라 spacial locality까지 가짐

III. 단점: parity bit 디스크 파괴되면 신뢰성 문제가 발생하여 parity bit를 다시 다 계산해야 함

F. RAID 5

I. 개요: RAID 4 문제 해결 위해 parity도 striping하는 RAID

II. 장점: parity가 흩어져있어 임의 디스크 1개 박살 시 복구가 더 uniform(어디 깨져도 똑같이 동작

III. 단점: 2개 이상 파괴되면 복구 불가

IV. 특징: 현재 실제로 쓰이는 RAID중 1개

G. RAID 6

I. 개요: RAID 5 문제 해결 위해 parity bit 대신 error-correcting code를 사용하는 RAID

II. 장점: 디스크 최대 2개 파괴돼도 복구 가능

III. 특징

- error-correcting code 동작을 위해서는 사용 디스크 최소 5개 필요
- 현재 실제로 쓰이는 RAID중 1개

H. RAID 0+1

I. 개요: 성능, 신뢰성을 다 최대로 하기 위해 RAID 0 -> RAID 1 순서로 합치는 RAID

II. RAID 0: block-level stripping 통해 높은 성능 제공

III. RAID 1: Mirroring 통해 높은 신뢰성 제공

IV. 단점: block-level stripping을 한 것을 Mirroring함

- 디스크 1개가 날아가면 전체를 쓰지 못하게 됨(비효율적)

I. RAID 10 (RAID 1+0)

I. 개요: 성능, 신뢰성을 다 최대로 하기 위해 RAID 1 -> RAID 0 순서로 합치는 RAID

II. 장점: 디스크 1개 날아가도 미러링 된 것을 stripping했기 때문에 그 디스크에 대한 클론 사용 가능

- 결국 나머지 디스크들 사용 가능(효율적)

J. 보통 사용하는 RAID: 5, 6, 10