IDC에서의 애플리케이션 이슈와 SSD 요구 특성


박 준 현
Naver Labs


신 기 빈
LINE+


송 창 현
Naver Labs

Ⅰ. 서 론

인터넷 데이터센터(IDC: Internet Data Center)에서 24시간 365일 서비스하고 있는 대부분의 서비스 애플리케이션들은 예측된 사용자 증가뿐만 아니라 여러 사회적 이슈로 인한 일시적인 요청 급증에 대해 유연하게 대응할 수 있어야 하고, 정해진 서비스 시간 내에 요청에 대한 응답을 사용자에게 줄 수 있어야 한다. 하나의 서비스를 담당하는 최상위 애플리케이션은 그 내부에서 여러 하위 애플리케이션들 또는 서비스들에 의존하는 경우가 대부분이다. 사용자 요청에 대해 최상위 애플리케이션은 여러 하위 애플리케이션들 또는 서비스들을 병렬적으로 호출하여 중간 처리 결과를 얻고 이들의 취합 및 요약 과정을 통해 최종 응답 결과를 만들어 낸다. 따라서, 그 서비스를 제공하는 최상위 애플리케이션은 물론이고 그 하위에 의존하고 있는 내부 애플리케이션들 또는 서비스들 모두에서 scale-out 구조 및 병렬 처리 등이 고려되어야 하고, OS 커널, 네트워, CPU, 메모리, 디스크 전반에 걸쳐 데이터가 이동/처리되는 모든 구간에서 비용 효율적인 성능 최적화가 필요하다[1-2].

IDC에서의 여러 기술 요소들 중 서비스의 요청 처리량과 응답 시간 관점에서 가장 큰 성능적 한계를 보이는 부분이 HDD(hard-disk drive) 기반의 디스크이다. 자기 기록/읽기를 위한 플래터(platter)의 회전 속도와 기계적인 헤드의 이동 속도의 한계로 인해, HDD 기반 디스크에서의 random IO 성능이 매우 떨어진다. 반면, NAND 플래시(flash) 기반의 SSD(solid-state drive)는 반도체 회로에 전자적인 방식으로 데이터를 저장/조회하기 때문에 HDD보다는 훨씬 좋은 성능 결과를 보이고 있다. 따라서, 많은 요청 처리량과 빠른 응답 시간을 필요로 하는 서비스 애플리케이션들에서 HDD를 보완하거나 대체

하는 관점으로 고성능의 SSD 검토 및 적용이 활발히 이루어지고 있다[10-11].

SSD는 성능적 이득 외에도, IDC 관점에서 여러 장점들을 가지고 있다. SSD는 HDD와 달리 기계적 장치가 없어서, 소음과 발열이 적고, 소비 전력과 냉각 비용이 매우 저렴하며, 서비스 요구 성능을 더 적은 수의 장비로 만족시킬 수 있어 상면 비용도 절감할 수 있다는 장점을 가진다. 반면, SSD는 쓰기 횟수 제한 문제를 안고 있고, 디바이스 자체의 가격이 비싸고, 제공되는 저장 용량이 적다라는 단점이 있다. SSD의 쓰기 횟수 제한은 해당 애플리케이션의 성격에 따라 문제 여부가 달라질 수 있다. 이미지, 오디오, 동영상과 같은 데이터는 한번 디스크에 기록되고 나면 그 이후에 조회만 되는 성격을 가지므로, SSD의 쓰기 횟수 제한이 문제되지 않는다. 반면, 데이터베이스 관리 시스템(DBMS: Database Management System)과 같이 데이터 삽입, 삭제, 변경이 많은 경우라면, SSD의 각 메모리 셀(cell)에 대해 대략 3000회 정도의 쓰기 횟수 제한은 문제될 소지가 있다. 이러한 애플리케이션은 메모리 셀에 1비트를 저장하는 SLC(single-level cell) 방식의 SSD를 선택하거나 메모리 셀에 2비트를 저장하는 일반 MLC(multi-level cell)에 비해 쓰기 횟수를 크게 늘려 놓은 eMLC(enterprise MLC) 또는 MLC-HET(high endurance technology) 방식의 SSD를 선택하여 쓰기 횟수 제한을 회피할 수 있다. 대신, 고 내구성(high endurance)을 지닌 SSD 사용은 단위 저장 용량 대비 고비용을 지불해야 한다. 결국, SSD 는 HDD에 비해 비싼 가격과 적은 저장 용량이 단점이다. 따라서, HDD는 대용량의 cold(접근 빈도가 낮은) 데이터의 저장/조회 용도로 사용하고, SSD는 중소용량의 hot(접근 빈도가 높은) 데이터의 저장/조회 용도로 사용하는 것이 보편적이다.

HDD는 대용량의 cold (접근 빈도가 낮은) 데이터의 저장/조회 용도로 사용하고, SSD는 중소용량의 hot (접근 빈도가 높은) 데이터의 저장/조회 용도로 사용하는 것이 보편적이다.

네이버 같은 국내 최대의 인터넷포털 업체는 HDD와 SSD중의 선택을 어떤 기준으로 할까? 첫째 기준은 해당 서비스 애플리케이션에 요구되는 성능 즉, 요청 처리량과 응답 시간을 만족시킬 수 있어야 한다는 것이다. 성능 조건이 만족되면, 그 다음으로 애플리케이션이 다루는 데이터 용량과 성격을 고려하여 비용 효율성을 검토하게 된다. 이 경우, IDC 관점의 소비 전력, 냉각 및 상면 비용과 디바이스 자체의 가격과 저장 용량이 모두 고려된다. 이중에, SSD 자체의 가격과 기업용 품질 보증 비용을 합한 가격이 상당히 고가이며, 이 비용이 선택에서 큰 영향을 미친다. SSD의 소비 전력, 냉각 및 상면 비용 등은 대규모 수요가 있을 시에 크게 고려된다. 이러한 검토 결과에 따라 HDD only, SSD only, HDD/SSD hybrid 등의 사용 방식을 결정한다. 추가 사항으로, 디바이스 사용 방식에 따라 애플리케이션 구조 및 구현 복잡성이 달라지고 이의 개발/유지보수에 드는 인적/시간 비용이 큰 차이를 보인다면, 이 또한 중요 비용으로 검토된다.

본 고에서는 앞서 기술한 HDD와 SSD의 일반적 특성 차이와 IDC 애플리케이션에서 이들의 선택 기준 외에, SSD 관련자 분들에게 유용할 만한 정보 제공을 위해 IDC 애플리케이션의 요구 성능에 가장 큰 영향을 미치고 있는 디스크 IO 부분을 자세히 살펴보고자 한다. 이를 위해, 네이버의 IDC에서 서비스하는 애플리케이션 중 디스크 상에 데이터를 저장하고 조회하는 대표 애플리케이션을 골라서, 이들의 중요 디스크 IO 특성과 이슈를 먼저 확인해 본다. 그리고 나서, 이러한 디스크 IO 특성과 이슈를 가진 IDC 애플리케이션의 요구 성능을 만족시켜 주거나 그 외의 관점으로 IDC 애플리케이션에 도움이 될 만한 SSD 특성들을 기술한다.

Ⅱ. IDC 애플리케이션의 디스크 IO 특성과 이슈

네이버에서 디스크 상에 데이터를 저장/조회하는 대표 애플리케이션은 아래와 같다.

1) DBMS
2) 분산 파일 시스템
3) 검색 시스템

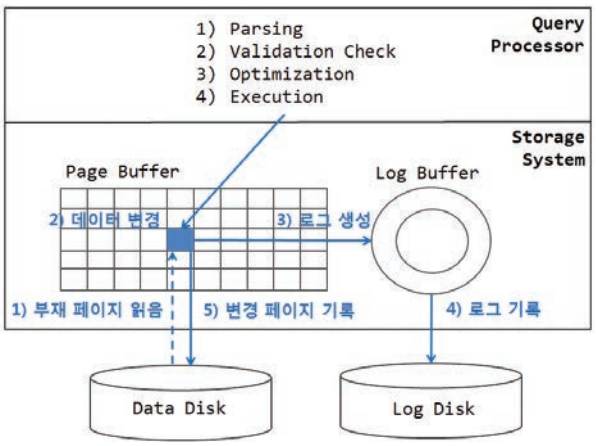
이 시스템들은 대부분 데이터 저장/조회 서비스를 개별적인 상위 애플리케이션에게 제공한다. 상위 애플리케이션의 기반 서비스 성격에 따라 이 시스템들이 다루게 되는 데이터 용량, 접근 빈도의 분포, 수행 연산의 성격이나 비율 등이 달라지고, 이에 따라 디스크 IO 워크로드가 달라진다. 본 장에서는 개별 서비스 성격에 의해 달라지는 디스크 IO 워크로드는 제외시키고, 위 시스템 각각에 대해 공통적으로 발생하는 디스크 IO 특성과 이슈 위주로 기술한다.

본 절에서는 Oracle, DB2, SQL Server, Cubrid, MySQL InnoDB와 같이 WAL (write-ahead logging)기반의 원위치 변경 (in-place update) 방식을 사용하는 가장 일반적인 DBMS를 대상으로 한다.

1. DBMS

본 절에서 언급하는 DBMS는 Oracle, DB2, SQL Server, Cubrid, MySQL InnoDB와 같이 WAL (write-ahead logging) 기반의 원위치 변경(in-place update)[3] 방식을 사용하는 가장 일반적인 DBMS를 대상으로 한다.

DBMS에서 데이터를 변경하는 SQL문의 처리는 다음과 같다. 질의 처리기가 파싱(parsing), 유효성 검사, 최적화를 통해 수행 계획(execution plan)을 생성하고, 그 수행 계획에 따라 저장 시스템의 데이터 변경 기능을 호출하여 수행한다. 저장 시스템은 (1) 변경할 데이터가 페이지 버퍼(page buffer)에 없으면 그 페이지를 디스크로부터 읽어 들여 (2) 그 데이터를 변경하고, (3) 변경 내역에 대한 로그 레코드(log record)를 로그 버퍼(log buffer)에 기록한다. (4) 로그 버퍼에 기록된 로그 레코드는 트랜잭션의 완료(commit) 시에 또는 변경 페이지(dirty page)가 디스크로 기록되기 전에 로그 디스크(log disk)에 먼저 기록하며, (5) 변경된 데이터 페이지는 검사점(checkpoint)[3] 작업 또는 버퍼 교체(buffer replacement)에 의해 디스크에 기록된다.



〈그림 1〉 DBMS의 데이터 변경 과정

〈그림 1〉 데이터 변경 과정에서 본 바와 같이 DBMS가 사용하는 메모리 상의 버퍼와 데이터 저장 디스크는 크게 아래의 두 유형이 있다. 각각에서 발생하는 디스크 IO 패턴이 서로 달라서, 높은 성능이 요구되면 디스크를 분리하여 사용한다.

1) 데이터 버퍼와 디스크: 응용 데이터 저장
2) 로그 버퍼와 디스크: 회복용 로그 데이터 저장

먼저, 데이터 버퍼와 디스크 사이에서는 DBMS의 페이지 단위로 IO가 발생하며, 이 과정에서 디스크 IO의 특성과 이슈를 기술한다. 아래 내용은 데이터 저장을 위하여 Raw 블록 디바이스와 다양한 유형의 OS 파일 시스템 중에 선택해 사용하는 방식이나 OS 커널의 블록 IO 담당 계층에서 사용하는 디스크 IO 스케줄링 방식에 종속적이지 않고, 공통적으로 나타나는 특성들이다.

다양한 페이지 크기. DBMS는 일반적으로 4KB, 8KB, 16KB, … 등의 다양한 페이지 크기를 제공하며, DBMS 응용은 이들 중 하나를 선택할 수 있다. 응용에서 다루는 대부분의 로우(row) 크기를 고려하여, 데이터 버퍼에서의 버퍼 적중률(buffer hit ratio)을 높이는 페이지 크기를 선택할 수 있다. 예를 들어, 대부분의 로우 크기가 수십~수백 바이트 정도로 작다면, 작은 크기인

4KB를 선택하는 것이 8KB 이상 크기보다 정해진 메모리 공간에서 버퍼 적중률을 높일 수 있다.

Random IO 패턴. 데이터 버퍼와 디스크 사이에서 발생하는 IO는 모두 random IO이다. 질의 처리에서 접근하는 페이지가 버퍼에 없으면 디스크로 부터 read하고, 버퍼에서 변경된 지 오래된 페이지나 버퍼 교체 대상이 되는 변경 페이지를 디스크에 write한다.

Read Latency와 Write Throughput이 중요. Read 연산은 질의 처리에서 접근해야 할 페이지가 버퍼에 없을 시에 요청되며, low latency이어야 질의 처리의 응답 시간을 줄일 수 있다. 반면, Write 연산은 주로 background 작업에서 요청된다. 예를 들어, 주기적인 검사점 수행이 버퍼에서 변경된 지 오래된 페이지를 write하거나, 페이지 기록 쓰레드(page flush thread)가 버퍼 교체 가능성이 높은 즉, LRU 꼬리 부분에 위치한 변경 페이지를 미리 write한다. 이러한 작업은 foreground 성격의 질의 처리에 미치는 영향을 최소화하면서도 high throughput을 추구하게 된다. 만약, 작은 메모리 용량의 버퍼에 단위 시간당 변경 페이지들의 발생 빈도가 높은 환경에서 write throughput이 충분하지 않아 버퍼 교체가 지연된다면, 이는 결국 질의 처리 성능의 저하를 야기시킨다.

> 디스크에서의 read 요청에 대해 average read latency 보다 maximum read latency를 줄인다면 일관된 서비스 품질을 유지하는 데 도움이 된다.

Maximum Read Latency 최소화. 질의 처리 시간의 변동폭이 크면, 해당 서비스의 응답 시간 관점에서 품질 관리가 어려워진다. 그 보다는 질의 응답 시간이 항상 예측 가능한 범위 내에 있으면서 그 시간을 최소화하는 것이 중요하다. 이를 위해, 디스크에서의 read 요청에 대해 average read latency 보다 maximum read latency를 줄인다면 일관된 서비스 품질을 유지하는 데 도움이 된다.

두 번째로, 로그 버퍼와 디스크 사이에 발생하는 IO를 살펴본다. 로그 레코드는 기본적으로 가변 길이(variable-length) 데이터이며, 이들이 데이터 변경 순서대로 로그 버퍼에 쌓이면서 그 순서대로 로그 파일에 write된다. 그리고, 어떤 고장에 의해 DBMS가 restart되면, 고장 발생 시점의 일관적인 상태로 데이터베이스를 회복(recovery)시키기 위해, 로그 데이터를 read한다. 이러한 로그 데이터의 IO에서 중요 특징과 이슈를 기술한다.

중소용량의 로그 디스크. DBMS에서 로그 데이터를 저장하기 위한 로그 디스크의 용량은 그리 크지 않아도 된다. 대부분 애플리케이션에서 오랫동안 수행되는(long-running) 트랜잭션이 거의 없으며, 데이터베이스 회복 시간을 줄이기 위해 오래된 변경 페이지를 디스크에 write하는 검사점이 자주 수행되기 때문이다. 따라서, 로그 디스크는 최근에 수행된 트랜잭션들의 로그 데이터만 저장할 수 있는 용량이면 된다.

다양한 트랜잭션 지속성(durability). 트랜잭션의 지속성 즉, 한번 완료된 트랜잭션은 어떤 고장이 발생하더라도 데이터베이스에 영원히 반영되어야 하는 속성을 준수하려면, 트랜잭션 완료 시마다 그 트랜잭션의 로그 레코드들이 모두 디스크에 기록됨을 보장하여야 한다. 이를 준수하면, DBMS 성능이 그 만큼 낮아지게 된다. 반면, 어떤 응용들은 고장 발생 시의 일부 데이터 손실을 허용하더라도, 더 좋은 성능을 DBMS에 요구하는 경우가 있다. 이에 따라 DBMS는 트랜잭션의 지속성 수준을 낮추는 기능[8]을 제공한다. 예를 들어, OS 파일 시스템의 페이지 캐시에 있는 변경 페이지 기록 방식과 주기에 의존하거나, 특정 시간 주기로 로그 데이터를 디스크에 기록하는 수준을 제공한다. 완벽한 지속성이 필요한 경우라면 write latency가 중요하고, 약화된 지속성을 사용해도 된다면 write throughput이 중요하다.

가변 길이 데이터의 Sequential Write. DBMS에서 로그 데이터의 write 패턴은 가변 길이 데이터의 sequential write이다. 일반 디스크와 같은 블록 디바이스(block device)에서는 블록 단위로 IO가 수행되어야 하므로, 디스크에서는 다른 패턴으로 IO가 발생한다. Write 요청된 가변 길이 데이터의 경계는 디스크

블록의 경계와 정렬(align)되지 않을 가능성이 높아서, 가변 길이 데이터의 경계에 해당하는 디스크 블록은 로그 데이터가 가득 채워지지 않은 채로 write된다. 그 다음의 가변 길이 데이터의 write 요청에 의해 그 데이터 블록이 로그 데이터로 가득 채워지고 다시 디스크로 write된다. 정리하면, DBMS가 write하는 로그 데이터의 경계와 블록 디바이스의 블록 경계가 서로 맞지 않아서, 하나의 디스크 블록에 대해 두 회 이상의 write가 요청되는 경우가 많다.

디스크 블록 단위의 Sequential Read. 데이터베이스 회복을 위하여 로그 데이터를 read하는 경우는 대부분의 DBMS가 OS 파일 시스템의 페이지 캐시를 이용하든 자체적으로 구현하든, 디스크 블록 하나에 대해 한 번의 read IO만 발생하게 한다.

추가 사항으로, 로그 디스크가 이용되는 다른 경우로, DBMS에서 고 가용성(high availability) 서비스를 위해 제공하는 복제(replication) 기능이 있다. 복제 용도의 로그 데이터는 DBMS 구현에 따라 회복용 로그 데이터에 함께 기록하거나 또는 별도의 로그 파일로 기록한다. 데이터베이스 복제는 대부분 비동기(asynchronous) 방식이며, 일부는 반동기(semi-synchronous)[9] 방식을 제공한다.

복제용 로그 데이터의 기록은 회복용 로그 데이터 기록과 유사하다. 트랜잭션 지속성 수준에 따라 복제용 로그 데이터 기록 방식이 달라지고, 가변 길이 데이터의 sequential write 패턴을 가진다. 차이점은 비동기 복제의 경우 로그 디스크에 read 연산이 꾸준히 발생한다는 점과 복제 네트워 고장이나 슬레이브(slave) 서버 고장으로 복제가 일시 중단될 수 있고 그 기간 동안의 복제용 로그 데이터 보관을 위해 기존보다 큰 용량의 디스크가 필요하다. 하지만, 앞서 기술한 로그 디스크 IO 특성을 크게 변화시킬 만큼의 차이는 없다.

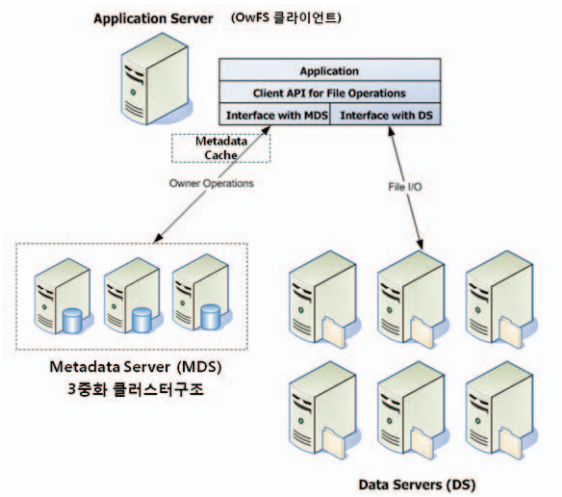
파일들의 전체 용량에 비해 파일 연산들의 요청이 극도로 많다면, DS (data server) 수를 늘리는 것 보다 고성능의 SSD 기반의 서버를 사용하는 것이 비용면에서 효과적일 수 있다.

2. 분산 파일 시스템

네이버는 대규모 인터넷 서비스에 적합한 분산 파일 시스템으로 OwFS(Owner-based File System)[4][5]를 자체 확보하여, N드라이브를 포함하여 다수의 서비스에 이용하고 있다. OwFS는 많은 수의 파일들을 효율적으로 관리하기 위해, owner라는 단위로 파일들을 그룹화하여 owner 단위로 파일 저장 위치를 관리하고 복제 본을 유지한다. 따라서, 개별 파일 단위로 저장 위치와 복제 본을 관리하는 HDFS[6]의 방식에 비해, 중앙에서 관리해야 할 메터데이터의 양과 그에 대한 연산들의 요청 횟수를 줄일 수 있다.

OwFS 구조는 〈그림 2〉와 같이, 실제 파일들을 저장하고 서비스하는 다수의 데이터 서버(DS: Data Server)들과 DS들의 동작 상태를 모니터링하고 owner들의 3개 복제 본 위치를 저장하고 서비스하는 메터데이터 서버(MDS: Meta Data Server)로 구성된다. 따라서, 해당 응용은 OwFS 클라이언트를 통해 owner의 복제 본 위치 정보를 MDS로부터 얻은 다음, 그 위치가 가리키는 DS에 접근하여 실제 파일 연산을 수행한다.

OwFS의 MDS는 메터데이터 관리를 위해 DBMS를 사용하고 있으므로, 앞 절에 기술한 DBMS의 디스크 IO 특성을



〈그림 2〉 OwFS 구조

가진다. DS들은 OS의 파일 시스템을 이용하여 실제 응용 파일들을 저장한다. 이러한 DS에서 발생하는 디스크 IO 특성과 이슈를 기술한다.

Random IO 패턴. 개별 파일에 대한 IO는 대부분 sequential read/write 이다. 하지만, 개별 파일 IO를 다수 클라이언트들이 요청하고 해당 파일들이 OS 파일 시스템으로 관리되므로, 디스크에서 발생하는 패턴은 random IO 이다.

대용량 디스크와 Scale-out 구조. 대용량 데이터 저장이 일반적이라 HDD 기반의 서버를 사용한다. 하나의 DS당 요청되는 파일 연산들이 많아져 높은 처리량이 요구된다면, DS들의 scale-out 방식으로 해결한다. 어떤 응용은 대량의 파일들 중 최근 생성된 파일들에 그 접근이 집중되는 특성을 가지고 있어서, 최근 생성된 파일들을 저장하는 SSD tier와 오래된 파일들을 저장하는 HDD tier를 구분하고, 생성 시간 기준으로 SSD에 저장된 파일을 HDD로 이동시켜 관리하는 SSD/HDD hybrid 방식을 사용한다. 만약, 파일들의 전체 용량에 비해 파일 연산들의 요청이 극도로 많다면, DS 서버 수를 늘리는 것보다 고성능의 SSD 기반의 서버를 사용하는 것이 비용 효과적일 수 있다.

Maximum Read Latency 최소화. 성능 요소 중 throughput은 DS 서버들의 scale-out 구조로 해결이 가능하므로, latency가 더 중요한 요소이다. OwFS의 write-to-read 비율에서 read 비율이 낮으면 1-to-1 정도, 높으면 1-to-10 이상이 된다. 대체로 read 요청이 많은 편이고, write 보다 read 요청의 latency에 대해 사용자는 참을성이 적어 read latency를 낮추는 것이 중요하다. OwFS에서 write는 3개 복제 본에 대해 병렬적으로 수행하며, write한 데이터가 디스크에 기록되기를 기다리지 않음으로 성능을 높인다. 대신, 어느 하나에 문제가 생기면, 정상 write된 다른 복제 본으로부터 해당 데이터를 가져와서 복구한다. Read는 하나의 복제 본에서 데이터를 읽는 연산이며, 항상 예측 가능한 서비스 시간을 보장하기 위해, average latency 보다는 maximum latency를 낮추는 것이 중요하다.

Bad Block 처리. OwFS는 고 가용성을 위해 3개 복제 본을 유지하고 있으며, 디스크 수준에서는 초기 RAID-0 구성을 사용하였으나, 디스크 파손 등의 고장 시에 복구할 데이터 양이 많아 현재는 RAID-5 구성을 사용하고 있다. RAID-5 구성의 경우도 용량 낭비와 RAID 콘트롤러 H/W 비용 등의 이슈가 있어, 개별 디스크를 마운트해 사용하는 방식도 고려 중에 있다. RAID-0나 개별 디스크를 직접 이용하는 경우엔, bad block 발생에 대해 RAID 콘트롤러의 bad block remapping 방식처럼 bad block 접근을 회피하는 로직이 필요하다. 즉, 어떤 파일에 bad block이 발생하면, 그 파일을 사용하지 않는 상태로 파일 이름만 바꾸어 남겨 두고, 다른 DS의 복제 본에서 그 파일의 내용을 읽어 현재의 DS에 동일 이름의 파일로 생성한다.

> **검색 시스템은 대량의 문서들 수집과 색인 생성을 정해진 시간내에 완료하여야 하고, 대량의 검색 질의들을 정해진 시간내에 처리하고 응답을 만들어 전달해야 하기에, 기본적으로 scale-out 구조를 가지고 분산 병렬 처리 방식을 사용한다.**

3. 검색 시스템

네이버의 검색 시스템은 크게 세 부분으로 나눌 수 있으며, 각각의 기능은 다음과 같다.

1) 문서 수집기 – 인터넷 상에 존재하는 대량의 웹 문서들과 블로그/카페/지식인 등의 포털 서비스의 DB 내용을 수집
2) 색인 생성기 – 수집된 문서들에 대해 정제된 색인어(term) 기반의 역 색인(inverted index) 생성
3) 검색 질의 처리기 – 생성된 역 색인 기반으로 다양한 검색 질의를 처리

검색 시스템은 대량의 문서들 수집과 색인 생성을 정해진 시간 내에 완료하여야 하고, 대량의 검색 질의들을 정해진 시간 내에 처리하고 응답을 만들어 전달해야

하기에, 기본적으로 scale-out 구조를 가지고 분산 병렬 처리 방식을 사용한다. 따라서, throughput 관점의 성능은 각 시스템에 담당 장비를 추가함으로써 확보할 수 있다.

문서 수집기에서 수집한 문서들은 DBMS 또는 NoSQL 기반의 분산 시스템을 이용하여 가공 처리하여 저장한다. 이에 대한 디스크 IO는 DBMS 부분에서 기술된 내용과 유사하므로, 본 절에서는 생략한다.

색인 생성 작업을 수행하는 각 서버는 자신이 맡은 문서들 각각에 대해 색인어 추출과 출현 빈도 계산 등의 전처리 과정을 통해 색인어 중심의 문서 위치 및 출현 빈도들을 별도의 파일로 생성하고 이들을 병합(merge)하는 과정을 통해 역 색인을 생성한다. 이 과정에서의 디스크 IO 특성은 다음과 같다.

Random IO 패턴. 색인 전 처리 과정에서 생성하는 개별 파일의 디스크 IO는 sequential write/read 패턴이다. 하지만, OS의 파일 시스템 상에서 여러 프로세스들이 동시에 이 작업을 수행하므로, 디스크 상에서는 random IO 패턴을 보인다.

Throughput 최대화. 색인 생성은 새로 서비스할 최신 색인을 만드는 배치(batch) 성격의 작업이다. 따라서, 한정된 자원 범위에서 단위 시간 당 최대한 많은 양의 문서들을 색인화하는 throughput 성능이 중요하다.

앞서 생성한 최신 역 색인은 검색 서비스 장비로 이동시켜 기존 역 색인과 교체한다. 이러한 교체 작업이 진행되더라도, 기존 역 색인 기반의 질의 처리 성능은 그대로 유지되어야 한다. 이를 위해, 검색 서비스 장비의 디스크로 최신 역 색인을 기록하는 작업은 최소의 자원을 이용하여 진행한다. 이 과정에서, 특별히 언급할 만한 디스크 IO 특성은 없어서 생략한다.

검색 서비스가 최근 역 색인을 사용하도록 전환하고 나면, 그 시점부터 code start 형태로 시작하게 된다. 이 시점에서 발생하는 디스크 IO 특징과 이슈를 기술한다.

Random IO 패턴. 검색 질의 처리 과정에서 접근해야 할 데이터가 메모리에 없으면 디스크에서 read한다.

Maximum Read Latency 최소화와 비용. 검색 질의 처리에 대한 요구 응답 시간이 아주 짧기 때문에, maximum read latency 최소화는 아주 중요하다. 이를 달성하기 위해, SSD나 메모리를 이용할 수도 있지만, 비용적인 측면도 많이 고려되고 있다.

Ⅲ. IDC 애플리케이션의 SSD 요구 특징

본 장은 앞서 기술한 IDC 애플리케이션의 디스크 IO 특징과 이슈들을 바탕으로 SSD에의 요구 사항들을 정리하여 기술한다.

1. Random IO 성능 극대화

IDC 애플리케이션에서 발생하는 대부분의 디스크 IO 패턴은 random IO이다. 따라서, random IO 성능을 극대화할 수 있다면, 아주 유용하다.

> IDC 애플리케이션에서 발생하는 대부분의 디스크 IO 패턴은 random IO 이며, 대부분 동기적 기록을 보장하지 않아도 된다. 이 경우에, SSD는 비용 효율적인 방법으로 random IO 성능을 높일 여지가 있다.

다행스러운 점은 random IO 패턴을 보이는 대부분 애플리케이션에서 write 요청은 디스크에 해당 데이터가 기록됨을 그 즉시 요구하지 않는다는 것이다. DBMS의 데이터 페이지 기록, 분산 파일 시스템의 응용 파일 기록, 검색 시스템에서의 색인 파일 기록 모두가 동기적(synchronous) 기록을 요구하지 않고 있다. 대신, IDC 애플리케이션들은 지속적인 서비스를 위해, power fault와 같은 고장에 대해 로그 데이터를 이용한 회복, 다른 복제본의 데이터를 가져와서 복구, 또는 재수행 방식과 같은 각 시스템마다 나름대로의 회복 및 복구 방법을 구현하고 있다.

동기적 기록을 보장하지 않아도 된다면, SSD는 비용 효율적인 방법으로 random IO 성능을 높일 여지가 있다. 예를 들면, SSD 내부에 큰 크기의 RAM 버퍼를 두어 활용할 수 있다. RAM 버퍼는 read 요청이 많은 데이터의 cache 역할을 할 수 있을 뿐만 아니라 최근

write된 데이터의 버퍼 역할을 하여 random IO 성능을 높일 수 있다. 반면, 동기적 기록을 보장하지 않아도 되므로, 정전과 같은 고장 시에 RAM 버퍼상의 변경 데이터를 플래쉬 칩에 기록하는 데 필요한 전기를 담아둘 필요가 없다. SSD 내부의 FTL(Flash Translation Layer)의 페이지 매핑 페이블, GC(Garbage Collection), wear leveling 등에 관련된 메타데이터의 관리에 필요한 소용량의 축전기(capacitor)만 있으면 된다. 이러한 방식으로, 비용을 낮추면서 고성능의 random IO를 제공할 수 있다고 본다.

2. Maximum Read Latency 최소화

IDC 애플리케이션은 대부분 정해진 시간 안에 응답을 만들어야 하고, 그 응답 시간이 예측 가능한 범위 내에 있도록 해서 관리 용이성을 제공해야 한다. 이를 위해, maximum read latency를 낮추는 것이 중요하다.

IDC 애플리케이션이 디스크에 read 요청을 하는 시점과 그 요청 양은 전혀 제어할 수 없는 부분이다. Read 요청은 해당 서비스에 대해 사용자 클릭과 같은 행위에서 시작되기 때문에, 임의 시점에 발생한다고 보면 된다. 반면, DBMS 같은 애플리케이션은 메모리 버퍼를 이용하여 변경 데이터를 관리하므로, 디스크에 대한 write 요청 시점과 양을 일부 조절할 수 있다. 이러한 조절도 결국은 read 연산의 latency에 영향을 주지 않으면서 최대 throughput을 얻기 위한 것이다.

SSD에서 read 요청에 대한 maximum latency를 최소화하는 방법으로, read 요청에 대해 write나 erase 연산보다 높은 우선순위(high priority)로 처리할 수 있다. SSD 내부의 큐들(queues)과 다수 플래쉬 칩(flash chip) 기반의 병렬 수행 구조에서 이러한 우선순위 조정은 가능하다고 본다.

또 다른 방법으로, SSD 내부에서 GC같은 background 작업의 수행 주기와 강도를 조절할 수 있다. 긴 주기로 한번에 많은 작업을 수행하는 방법 대신에, 짧은 주기로 작은 작업을 자주 수행하는 것이 전체 효율성 관점에서 불리하지만, maximum read latency를 낮추는 데 유리하다.

3. Average Write Latency 최소화

DBMS 같은 트랜잭션 처리 시스템은 회복을 위해 로깅 방식을 사용한다. DBMS의 로그 디스크 IO 특성과 이슈에서 기술한 바와 같이 로그 데이터는 중소용량의 디스크로 저장할 수 있고, 로그 데이터의 기록이 빈번히 발생한다. 완전한 트랜잭션 지속성을 준수해야 하는 환경에서 로그 데이터의 기록 성능이 해당 트랜잭션 처리 시스템의 전체 성능을 결정할 만큼 그 성능이 아주 중요하다. 이러한 로그 데이터 저장 용도에는 기본적으로 SLC 타입의 SSD가 적합하다고 볼 수 있다

로그 레코드는 LSN(log sequence number) 라는 순서 정보를 가지고 그 순서대로 기록되어야 하기에 주로 단일 쓰레드에 의해 기록된다. 트랜잭션 지속성의 완전한 보장은 일부 로그 데이터를 기록할 때마다 동기적 기록이 완료될 때까지 기다린다. 이러한 로그 데이터 기록을 위한 SSD는 적당 크기 이상의 RAM 버퍼와 축전기가 필요하고, SSD 내부의 여러 플래쉬 칩으로 write 요청이 골고루 분산되도록 하는 것이 average write latency를 최소화하는 방안으로 본다.

4. Throughput 성능 최대화

검색 시스템에서 색인 생성의 전처리 과정과 같이 다수 프로세서가 배치 성격으로 작업을 반복하는 경우, high throughput을 제공하는 SSD가 최적이다.

검색 시스템에서 색인 생성의 전처리 과정과 같이 다수 프로세스가 배치 성격으로 디스크에서 파일의 내용을 읽어 메모리 상에서 처리하고 그 결과를 다시 파일로 기록하는 작업을 반복하는 경우가 있다. 이 경우는 high throughput을 제공하는 SSD가 최상이다. 현재 대부분의 SSD가 high throughput 관점으로 최적화되었을 것으로 예상한다.

5. 단일 페이지의 원자적 기록

DBMS 같은 애플리케이션은 해당 DBMS 응용의 데이터 성격에 적합한 크기의 페이지를 선택하여 데이터를 저장하며, 그 페이지 단위로 IO를 수행한다. 애플리케이션이 사용하는 페이지 크기가 SSD 내부의 read/write 단위의 페이지 크기보다 크다면, 애플리케이션 페이지에 대한 write 연산의 원자성(atomicity)이 보장되지 않게 된다. 한 페이지의 write 원자성이 보장되지 않으면, 부분 기록(partial write) 페이지가 발생할 수 있으며, DBMS의 보편적인 물리논리적 로깅(physiological logging)[3] 방식은 부분 기록 페이지에 대해 정확한 회복을 보장하지 못하는 문제가 있다. 이러한 부분 기록 페이지 문제를 회피하기 위해, MySQL InnoDB는 이중 쓰기(double write) 방식[7]을 사용하고 있다.

애플리케이션에서 정의한 크기의 페이지에 대해 SSD 수준에서 기록 원자성을 제공할 수 있다면, 애플리케이션에서 부분 기록 페이지 회피를 위한 방안의 개발 및 그 방안에 의한 성능 저하를 막을 수 있다. 애플리케이션에서 선택하는 페이지 크기가 그리 다양하지 않으므로, SSD 내부에서 이러한 크기의 데이터 기록에 대해 원자성을 제공하는 것이 전체 성능 상 유리할 것으로 본다.

6. Bad Block 최소화

Bad block 회피 기능을 가진 RAID-5나 RAID-1 등을 사용하지 않는다면, 해당 애플리케이션에서 bad block에의 접근을 회피하기 위한 로직을 직접 가져야 한다. 이런 대응 로직은 다른 장비의 디스크에 저장된 파일을 읽어와서 복구해야 하므로, 일시적으로 서비스에 작은 부담이 될 수 있다.

SSD의 특성상 wear-out 등으로 인한 bad block 발생 가능성이 높은 페이지/블록을 사전에 알 수 있고, 이들을 사전에 unused 상태로 관리할 수 있을 것으로 본다.

7. 그 외의 SSD 요구 특성

앞서 살펴본 디스크 IO 특성과 이슈를 바탕으로 한 SSD 요구 특성 외에도, 네이버 내부에서 SSD 시험하면서 얻은 경험들[12]을 바탕으로 저장/조회 기능을 가진 애플리케이션들에 도움이 될 만한 SSD 요구 특성들을 기술한다.

- SSD 내부 동작 상태의 조회 기능 – SSD 내부의 페이지/블록 크기, GC 수행 시점과 주기, ware leveling, 에러 수정(error correction), read-ahead 등의 내부 상태를 조회하는 기능이 제공된다면, 그에 맞춰 해당 IDC 애플리케이션의 동작을 최적화할 수 있다.
- SSD 내부 동작의 제어 기능 – SSD 내부 동작을 제어할 수 있다면, 애플리케이션 상황에 맞게 SSD를 사용할 수 있다. 예를 들어, GC 수행 기준이나 주기를 제어하여, latency 또는 throughput 성능을 조절할 수 있다. 또한, GC의 일시적 중시/개시를 제어할 수 있다면, 애플리케이션의 write 요청들과 GC가 동시에 수행되는 경우를 줄일 수 있다.
- Transparent Compression – 분산 파일 시스템과 같이 대용량 데이터를 저장하는 경우에 유용한 기능으로, SSD 내부에서의 자동 압축/해제 기능이 있다.
- 20% 이상의 Over-Provisioning – Over-provisioning은 SSD의 저장 공간 중에 wear-leveling, 쓰기 증폭(write amplification) 및 dead cell 교체 등의 용도로 제공하는 여분의 저장 공간을 의미한다. Over-provisioning 비율에 따라, latency 관점의 성능이나 SSD 수명 및 내구성에 큰 차이가 있음을 확인하였다. 데이터 쓰기가 많은 IDC 애플리케이션을 위한 SSD라면, 20% 이상의 over-provisioning을 제공하여야 한다.

Ⅳ. 결 론

본 고에서는 IDC 애플리케이션들이 서비스 요구 성능 즉, 요구 처리량과 요구 응답 시간을 만족시키기 위해, 검토되어야 할 여러 기술들 중 성능 병목이 큰 디스크 IO 부분에 관하여 비용 효율적인 성능 최적화를 위한 기반 정보를 SSD 관련자 분들에게 전달하는 관점

으로 기술하였다.

먼저, 네이버의 IDC에서 데이터 저장/조회 기능을 제공하는 대표 애플리케이션으로 DBMS, 분산 파일 시스템, 검색 시스템을 선택하여, 이들 시스템에서 발생하는 디스크 IO의 고유 특성과 이슈를 살펴보았다. 대부분 random IO 패턴이 많았고, 서비스의 요구 응답 시간을 만족시키고 이의 관리 용이성을 위해 maximum read latency 최소화가 필요함을 확인하였다. 서비스의 요구 처리량은 대부분 시스템에서 scale-out 구조로 기본 해결은 가능하므로, 비용 효율성 관점에서 검토되어야 한다. 이에 관련한 디스크 IO 특성과 이슈를 기술하였다.

다음으로, IDC 애플리케이션의 디스크 IO 특성과 이슈를 바탕으로 SSD에 요구되는 특성들을 기술하였다. SSD에 요구된 특징들 중 일부는 서로 공존이 가능하고, 다른 일부는 상반되는 특징을 가진다. 따라서, 하나의 SSD 제품으로 모든 요구 특징들을 수용하기는 어렵고, 개별 특성을 따로 제공하는 차별화된 SSD 또는 애플리케이션에서 그 특성을 설정할 수 있는 SSD 등이 필요할 것으로 예상한다.

감사의 글

본 고의 작성에 기반이 되는 애플리케이션의 구조 및 디스크 IO 특성과 이슈에 관해 도움을 주신 DBMS의 강철규 님, 분산 파일 시스템의 주윤철 님, 검색 시스템의 강유훈 님 그리고 본 고에 기술되지 않았지만 동영상인프라의 디스크 IO 특성에 관해 알려주신 조성철 님께 감사 드립니다.

참 고 문 헌

[1] K. Kant, "Data Center Evolution: A Tutorial on State of the Art, Issues, and Challenges", Computer Networks, Vol. 53, 2009.

[2] R. Kapoor, G. Porter, M. Tewari, G. M. Voelker, and A. Vahdat, "Chronos: Predictable Low Latency for Data Center Applications", In Proceedings of the Symposium on Cloud Computing, 2012.

[3] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz. "ARIES: A Transaction Recovery Method Supporting Find-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging", ACM Transactions on Database Systems, Vol. 17, No. 1, 1992. 3.

[4] 김진수, 김태웅, "OwFS: 대규모 인터넷 서비스를 위한 분산 파일 시스템", 정보과학회지 제27권 5호, 2009. 5.

[5] 전성원, 주윤철, 김태웅, "클라우드 환경을 위한 대규모 분산 저장 시스템", SKT Telecommunications Review, 제 21권 3호, 2011.

[6] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. "The Hadoop Distributed File System", IEEE 26th Symposium on Mass Storage Systems and Technologies, 2010. 5.

[7] Peter Zaitsev, "Innodb Double Write", MySQL Performance Blog, http://www.mysqlperformanceblog.com/2006/08/04/innodb-double-write/, 2006. 8.

[8] Peter Zaitsev, "MySQL should have dynamic durability settings", MySQL Performance Blog, http://www.mysqlperformanceblog.com/2008/04/08/mysql-should-have-dynamic-durability-settings/, 2008. 4.

[9] Baron Schwartz, "How Does Semisynchronous MySQL Replication Work?", MySQL Performance Blog, http://www.mysqlperformanceblog.com/2012/01/19/how-does-semisynchronous-mysql-replication-work/, 2014, 4.

[10] 이기열, "SSD를 쓰면 DBMS가 빨라질까?", hello world NHN 개발자 블로그, http://helloworld.naver.com/helloworld/7005, 2012, 1.

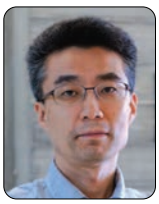
[11] 이혜정, "SSD는 소프트웨어 아기텍처를 어떻게 바꾸고 있는가?", hello world NHN 개발자 블로그, http://helloworld.naver.com/helloworld/162498, 2012, 9.

[12] 신기빈, "SSD 제대로 알고 쓰자", Naver Deview 2013, http://www.slideshare.net/deview/246-deview-2013

박 준 현

2000년~2004년 케이컴스, UniSQL storage engine team leader
2004년~2005년 이엠웨어, Embedded DBMS developer
2005년~2008년 리얼타임테크, Kairos storage engine team leader
2009년~현재 네이버, Arcus tech leader, DBMS/NoSQL performance engineer

〈관심분야〉
Storage engine, Query processor, DBMS, NoSQL, Distributed systems



송 창 현

1997년~2004년 Digital Equipment Corp, Compaq, Tru64 UNIX filesystem engineer
2004년~2006년 HP, Itanium virtual machine platform engineer
2005년~2006년 Microsoft, Disk imaging software engineer
2006년~2008년 Apple, Server performance engineer
2008년~현재 네이버, 네이버랩스 센터장

〈관심분야〉
Storage, Distributed system, System performance, File system, NoSQL

신 기 빈

2004년~2005년 Neowiz, Software engineer
2005년~2014년 네이버, Software engineer
2014년~현재 Line+, Software engineer

〈관심분야〉
System performance, Network system, C++ language