



12bme

길은 가면, 뒤에 있다.

CATEGORY

데이터베이스(DA, AA, TA)/대용량DB

[대용량DB] 대규모 웹 서비스란?

12bme 2017.05.15 02:20

대규모 웹 서비스

대규모 웹 서비스란, 거대한 데이터를 처리해야만 하는 웹 서비스를 말합니다.

대규모 서비스의 규모감, 대규모 데이터를 다루는 데 있어 어려운 점, 개발 모습을 인지하는 것은 대규모 서비스 개발자에게 필요한 지식입니다. 사용자가 이용하고 있는 대규모 서비스에 변경을 가할 때 규모를 고려하지 않고 어중간하게 구현해서 적용하다 보면, 시스템 정지를 초래할 수 있습니다.

대규모 웹서비스에 대해 알아보기전 크게 다음에 대한 개념을 잡고 가는 것이 좋습니다.

대규모 웹 서비스 개발이란?

- 대규모 데이터를 다룰 때의 과제, 다루기 위한 기본적인 사고방식과 요령.
ex) OS의 캐시(cache) 기능이나 대규모 데이터를 전제로 한 DB 운용 방법
- 알고리즘과 데이터 구조 선택의 중요성. 대규모 데이터를 예로 생각.
- RDBMS(Relational DataBase Management System)로 모두 다룰 수 없는 규모의 데이터 처리 방법.
ex) 전문 검색 엔진
- 대규모 서비스가 될 것을 전제로 한 서버/인프라 시스템.

대규모 서비스는 미들웨어 설정방법이나 프로그래밍 언어의 구문 등 세세한 부분까지 컨트롤해야하는 부분이 굉장히 많습니다. 대규모 서비스, 대규모 데이터를 다룰 경우를 대비한 기본적인 사고방식이나 개념, 개요에 대한 설명에 대해 학습하는 것을 목표로 합니다.

대규모 서비스 데이터 센터 모습의 예

- 등록 사용자는 100만 명 이상, 1,500만 UU(Unique User, 고유 사용자)/월
- 수십 억 액세스/월(이미지 등으로의 액세스는 제외)



12bme

길은 가면, 뒤에 있다.

CATEGORY

100만 명 이상의 사용자들이 블로그를 쓰거나 북마크를 등록하고 있고, 월간 1,500만 명이 방문하고 있습니다. 이 방문자 수에 의해 월간 수십억 액세스가 발생합니다. 이 정도의 액세스가 되면 일일 액세스 로그는 기본적으로 기가바이트(gigabyte) 크기가 되며, DB 서버가 저장하는 데이터 규모도 대략 기가바이트 수준, 많을 때는 테라바이트(terabyte) 정도가 됩니다.

물리적으로 한대의 서버 내에 복수의 호스트가 가동되고 있으므로, 500대 규모의 물리적 서버가 존재한다면 결과적으로 1,000대 이상의 호스트를 가지게 됩니다. 이정도 규모에서는 호스트 정보를 관리하기 위한 툴 등이 필요해집니다.

서비스의 규모는 서버대수 등으로 개략적으로 파악되는 경우가 많은데, 이런 관점에서 볼 때 백 대에서 수천 대 정도가 대규모 서비스라고 할 수 있습니다.

Google 및 해외에서 인기 있는 SNS로 최근 구글의 트래픽을 앞지른 전례가 있는 페이스북과 같은 세계 Top 클래스 사이트는 서버 대수가 수백만 대 규모이고, 처리하는 데이터는 테라바이트~페타바이트(petabyte)급의 초대규모 서비스입니다.

소규모 서비스와 대규모 서비스의 차이

서버 몇 대 정도의 소규모 서비스에는 없는, 대규모 서비스에만 있는 문제나 어려움이 있으며 그 내용은 다음과 같습니다.

1. 확장성 확보, 부하분산 필요

대량의 액세스가 있는 서비스에서는 **서버 1대로 처리할 수 없는 부하를 어떻게 처리할 것인지가 가장 큰 문제입니다. '스케일아웃(scale-out)'이 이 문제에 대한 전략의 기초가 됩니다.** 스케일아웃은 서버를 횡적으로 전개, 서버의 역할을 분담하거나 대수를 늘림으로써 시스템의 전체적인 처리능력을 높여서 부하를 분산하는 방법입니다. 반면 '스케일업(scale-up)'은 하드웨어의 성능을 높여 처리능력을 끌어올리는 방법입니다.

저가의 하드웨어를 횡으로 나열해서 확장성을 확보하는 것이 스케일아웃 전략입니다. 스케일아웃 전략을 채용한 경우는 비용이 절감되는 반면 다양한 문제가 발생합니다. 서버가 1대였을 때에는 전혀 생각지 않아도 될 문제입니다.

1) 사용자로부터 요청을 어떻게 분배할 것인가?

해답으로는 로드밸런서를 사용한다는 것인데, 서버 1대일 때에는 애초에 로드밸런서를 도입하는 것 자체를 생각할 필요도 없습니다.

2) 데이터 동기화는 어떻게 할 것인가?

DB를 분산시켰을 때 한쪽에 저장된 갱신 내용을 다른 한쪽 DB가 알지 못한다면 애플리케이션에 비정상 사태가 발생합니다.

3) 네트워크 통신의 지연시간(latency)을 어떻게 생각해 볼 수 있을까?

작은 데이터라도 이더넷(Ethernet)을 경유해서 통신한 경우는 밀리초(ms) 단위의 지연시간이 있습니다. 밀리초라고 하면 사람이 체감하기로는 그다지 긴 시간이 아니더라도 마이크로초나 나노초에 작동하는 컴퓨터에 있어서는 매우 긴 시간입니다. 통신의 오버헤드를 최소한으로 줄여가면서 애플리케이션을 구성해갈 필요가 있습니다.

2. 다중성 확보



12bme

길은 가면, 뒤에 있다.

CATEGORY

시스템 정지의 사회적 충격도 늘어나므로 더욱더 다중성 확보가 중요해집니다.

2001년 9월, 미국 동시다발적인 테러 발생 상황을 알고자 사람들이 일제히 야후에 액세스해서 야후 Top 페이지가 다운돼버리는 사태가 일어났다고 합니다. 야후는 이때 CDN 서비스인 Akamai에 콘텐츠를 캐싱해서 트래픽을 우회시킴으로써 장애를 복구했다고 합니다.

웹 서비스는 언제 어떠한 경우라도 고장에 대해 견고해야 합니다. 이는 상당히 어려운 과제인데, 시스템이 고장나면 그걸로 끝이 어도 괜찮은 시스템 구축과 고장 나더라도 다른 시스템이 자동적으로 처리를 인계받는 시스템 구축 간에는 기술적으로나 비용면에서 상당히 큰 차이가 있습니다.

3. 효율적 운용 필요

서버가 1대라면 때때로 상태를 확인하는 정도로 서버가 정상적으로 동작하고 있는지를 간단하게 파악할 수 있을 것입니다. 반면 서버 대수가 100대를 넘어서면 어떤 서버가 무슨 역할을 하고 있는지 등에 대한 관리가 어려워집니다. 부하는 괜찮은지, 고장 난 부분은 없는지, 디스크 용량은 아직 충분한지, 보안설정에 미비한 점은 없는지 등등... 이를 모든 서버에 대해 여기저기 잘 살펴야 합니다.

이에 대한 해결 방안으로 **소프트웨어를 사용하고 정보관리를 위한 툴을 사용하는 등 자동화를 하게 됩니다.** 그러나 이 감시 소프트웨어를 설치하거나 정보를 보는 것은 결국 인간입니다. 대규모 시스템을 건강한 상태로 얼마나 계속 유지해갈 수 있을 것인가? 이를 위한 효율적 운용을 수행해야 합니다.

4. 개발자 수, 개발방법의 변화

대규모 서비스가 되면 당연히 혼자서는 개발이나 운용이 어려워지므로 여러 기술자가 역할을 분담하게 됩니다. 애플리케이션을 각각의 기술자가 제멋대로 구현한 시스템의 전말을 생각하고 싶지 않습니다.

프로그래밍 언어를 통일하고, 라이브러리나 프레임워크를 통일하고, 코딩 규약을 정해서 표준화하고, 소스코드 관리를 버전관리 시스템으로 제대로 하는 등 이러한 사항들이 올바르게 실행되기 시작해야 여러 사람이 작업할 때 좋은 효율이 나타납니다. 여러 사람이 제각기 작업해서는 사람이 늘어나도 생산성은 오르지 않습니다.

대규모 데이터량에 대한 대처

컴퓨터는 디스크에서 데이터를 로드해서 메모리에 저장, 메모리에 저장된 데이터를 CPU가 fetch해서 특정 처리를 수행합니다. 또한 메모리에서 패치된 명령은 보다 빠른 캐시 메모리에 캐싱됩니다. 이처럼 데이터는 **디스크→메모리→캐시 메모리→CPU**와 같이 몇 단계를 경유해서 처리 되어 갑니다.

각 단계 간에는 속도차는 매우 크게 나는 것이 현대 컴퓨터의 특징입니다. 하드디스크에서 데이터를 읽어들이는 데에는 그 특성상 헤드 이동이나 디스크 원반의 회전이라는 물리적 동작이 수반됩니다. 따라서 전기적으로 읽어들이기만 하면 되는 메모리나 캐시 메모리와 비교하면 $10^6 \sim 10^9$ 배나 되는 속도차가 나게 됩니다.



12bme

길은 가면, 뒤에 있다.

CATEGORY

하지만 OS나 미들웨어 등의 소프트웨어에서 이런 구조를 통해 분발한다고해도 당연히 한계는 있습니다. 데이터량이 많아지면 처음부터 캐시 미스(cache miss)가 많이 발생하게 되고, 그 결과로 저속의 디스크로의 I/O가 많이 발생하게 됩니다. 디스크 I/O 대기에 들어선 프로그램은 다른 리소스가 비어 있더라도 읽기가 완료되기까지는 다음 처리를 수행할 수가 없습니다. 이것이 시스템 전체의 속도저하를 초래합니다.

대규모 웹 애플리케이션을 운용할 때 대부분의 어려움은 이러한 대규모 데이터 처리에 집중됩니다.

데이터가 적을 때에는 특별히 고민하지 않아도 모두 메모리에서 처리할 수 있으며, 복잡한 알고리즘을 사용하기보다 간단한 알고리즘을 사용하는 편이 오버헤드가 적기 때문에 더 빠른 경우도 종종 있으므로 I/O 부하는 일단 문제가 되지 않습니다. **그러나 서비스가 어느 정도 이상의 규모가 되면 데이터는 증가합니다. 이 데이터량이 분수령을 넘어서면 문제가 복잡해집니다. 그리고 응급처리로는 쉽사리 풀리지 않습니다. 이 점이 대규모 서비스의 어려운 점입니다.**

어떻게 하면 데이터를 적게 가져갈 수 있을까. 여러 서버로 분산시킬 수 있을까, 필요한 데이터를 최소한의 횟수로 읽어들이 수 있을까 등등.. 이것이 본질적인 과제가 됩니다.

시스템 규모확장에서의 시행착오

하테나의 규모확장에 대한 대처는 시행착오의 연속이었다고 합니다. 먼저 라우터는 Linux 박스로 저가에 구축, HTTP 요청 분산은 아파치의 mod_rewrite로 대응, DB분산은 MySQL의 레플리케이션 기능을 사용하기 시작하다 블로그 붐이 일어나며 트래픽 증가에 비해 시스템 확장이 따라가지 못하게 되었습니다. 당시는 분산뿐 아니라 다중화, 효율적 운용면에서도 미비한 점이 많았는데, 다중화된 시스템이 자동으로 가동되는 것이 아니라 근처에 살고 있는 엔지니어가 알아차리고 Hang-Up된 서버를 재가동하는 등의 응급처치로 견뎌냈습니다.

이후 트래픽이 지속적으로 증가하면서, 임대해서 쓰던 작은 서버룸이 서버 증설에 따라 전력이 부족하게 되면서 더이상 서버를 추가할 수 없는, 추가하게 되면 전류 차단기가 내려가버리게 되는 상황에 처하게 되었습니다.

데이터 센터로의 이전, 시스템 쇄신

더이상 손쓸 방법이 없는 상황이 되어서야 조직체제를 재점검하고 시스템 운용 전담팀을 구성하여 대응에 임했다고 합니다. 이때부터 1년에 걸쳐 작은 서버 룸에서 인터넷 데이터 센터로 서버를 이전하는 작업을 시작했습니다. 이전 작업을 하면서 네트워크 설계를 근본적으로 재수정하고 낡은 서버는 모두 교체하는 방침을 채택했습니다.

우선은 사전에 기존 시스템의 부하상황을 정리했습니다. 이 정보를 활용해서 각 서비스의 구성 중에 병목지점을 측정, 판정하고 I/O 부하가 높은 서버는 메모리를 중요시하고 CPU 부하가 높은 서버는 CPU를 중요시하는 형태로 서버 용도에 맞게 최적의 구성을 갖는 하드웨어를 준비해갔습니다.



12bme

길은 가면, 뒤에 있다.

CATEGORY

서버의 정보관리를 위해 독자적인 웹 기반 서버 정보관리 시스템도 개발하였으며, 이에 따라 서버의 용도나 부하상태와 같은 각종 정보에 액세스하기 쉬워져서 시스템 전체를 파악하기 용이해졌습니다.

서버/인프라 측면의 시스템 구성뿐만 아니라 애플리케이션의 각종 로직이나 DB 스키마 등도 재검토해서 비효율적인 부분을 서서히 배제해갔습니다. 필요에 따라 검색 엔진 등의 서브시스템을 독자 개발하는 경우도 있었습니다.

그러나 시스템이 안정되었다고해도 거기서 멈춰버리는 것이 아니라 결국 그 이후의 성장을 위해서는 다시 예전과 동일한 작업을 반복해야만 합니다. 그러지 않기 위해 현재도 개발 담당, 인프라 담당이 하나가 되어 밤낮없이 시스템 품질개선을 수행을 진행 중입니다.

대규모 서비스에서 기술팀

대규모 서비스에서 서비스 개발부는 각종 서비스 구현을 담당하며, 매일 애플리케이션 측면의 개선을 담당합니다. 서버/인프라 시스템은 1,000대 규모의 호스트를 보유하는 큰시스템이므로 전담팀이 그 운용을 담당하게 되는데, 고장, 과부하, 설정미비, 노후화로 인한 교체 등등. 이런 문제들을 전담해서 맡으면서 가상화나 클라우드 등 보다 세련된 새로운 모습으로 시스템을 확장해 가는 것도 인프라부의 일입니다. 서비스 개발부는 서버/인프라 시스템이 지탱하는 기반 위에 서비스를 개발합니다. 부하 상황은 서버/인프라팀이 감시하고 있고, 인프라 부분의 개선으로 대처할 수 있는 문제는 그들이 즉시 대응하게 되지만, 애플리케이션에 원인이 있는 경우 등 구현이 관련된 경우에는 서비스 개발부와 협력해서 대응합니다.

서비스발부에서도 담당하고 있는 서비스의 성능을 트래킹하고 있으며, 주요한 페이지가 어느 정도의 응답시간에 응답하고 있는지를 정량화해서 매일 그것을 지표로 한계값(threshold)를 밑돌지 않도록 목표를 설정한 개선을 진행합니다.

엔지니어가 개발에 사용하고 있는 툴의 종류

- 프로그래밍 언어
- 주요 미들웨어
- 웹 애플리케이션 프레임워크
- 주위 머신의 OS 및 에디터
- 버전관리, BTS(Bug Tracking System, 버그추적 시스템)



12bme
길은 가면, 뒤에 있다.


CATEGORY

[대용량DB] 대규모 웹 서비스란? (0)

[대용량DB] 대규모 웹 서비스란? (1)

TAG 대규모서비스, 대용량db

댓글 **1**



는세
이런 고민들을 지금까지 필요가 없다는 이유로 한번도 안해봤다는게 창피하네요.....잘보고 갑니다ㅠㅠ
2019.01.03 09:54 신고

여러분의 소중한 댓글을 입력해주세요

이름

비밀번호

비밀글

입력

1 ... 251 252 253 254 **255** 256 257 258 259 ... 341

TAG

more

튜닝 서버관리 네트워크 시스템프로... C 오라클 자바 리눅스 유닉스 RDBMS 성능 생각 깃 쉘스크립트 MySQL 버전관리 알고리즘 자바스크립트 정보보안 빅데이터

최근에 올라온 글

[대용량데이터] 대용량 처.. [개발용어] 가용성 [Vue.js] 공식 가이드 문서.. [Vue.js] 공식 가이드 문서.. [Vue.js] 공식 가이드 문서.. [Vue.js] 공식 가이드 문서.. [Vue.js] 공식 가이드 문서.. [Vue.js] 공식 가이드 문서.. [프로그래밍] IntelliJ (변.. [개발용어] 전체백업 / 증.. [SCALA] object, case clas..

링크

네이버 개발 블로그 서버개발 레퍼런스 블로그 개발실무 레퍼런스 블로그 오라클 레퍼런스 블로그 오라클 레퍼런스 블로그 오라클 엔지니어링 블로그 MySQL 레퍼런스 블로그 SK플래닛 기술 블로그 카카오 DB 기술 블로그

Total

677,169

Today 201 Yesterday 3,065

Blog is powered by [Tistory](#) / Designed by [Tistory](#)

https://12bme.tistory.com/100

6/6