(1)

안녕하세요.

소프트웨어 시스템 ARCUS를 발표하게된 이가빈, 이다솜. 그리고 발표자 박도희입니다.

(2)

저희는 ARCUS의 소개, 시스템 구성도, 소프트웨어 아키텍처 스타일순으로 발표를 하게 되었습니다.

(3)

기존 데이터베이스를 이용하는 large scale의 웹서비스들은 왼쪽의 그림처럼 데이터를 읽고 쓰는 과정에서 지속적으로 DBMS에 접근하였습니다. 그렇기 때문에 DBMS에 부하가 가게 되고 낮은 처리량과 느린 속도를 보이고 있었습니다.

그걸 보완하기 위해 웹서비스들은 각 어플리케이션마다 local cache를 두어 DBMS에 접근하는 횟수를 줄이고 local cache에 데이터를 저장해 읽고 쓰는 방식을 이용하였습니다. 하지만 local cache를 이용하는 경우 DBMS의 내용이 local cache마다 중복으로 들어가게 되고, local cache에서 데이터가 수정되면 데이터의 불연속성이 생기기 때문에 자주 접근되는 소량의 데이터나 변경되지 않는 데이터만 캐싱하는 등, 극히 제한된 용도로만 사용하여야 했습니다.

(4)

만약 데이터 베이스와 어플리케이션 사이에 캐시서비스가 있어 데이터를 읽을 땐 캐시에서 읽어오고 데이터를 쓸 때만 DBMS로 접근하게 된다면 DBMS로 가는 부하를 줄이면서 데이터의 접근 속도와 일관성을 유지할 수 있게 됩니다. 그게 바로 ARCUS의 개발 목적입니다.

(5)

ARCUS란 응용 서비스의 급변하는 사용자 요청에 유연하게 대응하기 위하여 NAVER에서 개발된 메모리 캐시 클러스터입니다.

ARCUS는 key-value타입을 지원하며 메모리 상에 위치하여 더 빠른 속도를 지원하는 오픈소스 소프트웨어 memcached와 분산 처리 환경에서 사용 가능한 데이터 저장소인 zookeeper를 기반으로 되어있으며, 자주 접근되는 데이터의 일부를 remote cached인 ARCUS에 적재함으로써 백엔드 저장소인 DBMS의 부하를 경감, 대용량 데이터 캐싱, 데이터간 consistency 유지가 가능하다는 장점을 가지고 있습니다.

(6)

다음은 ARCUS의 시스템 구성도입니다.

ARCUS는 application과 cache cloud 사이를 연결해주는 cloud admin인 zookeeper ensemble이 있으며, 리소스에 대한 모니터링, 캐시 노드의 data개수, memory 사용량, request에 대한 모니터링을 하는 ARCUS hubble로 이루어져 있습니다.

여기서 zookeeper는 cache들의 ip를 가지고 있다가 요청이 발생하면 client에게 전달하는 역할과, 캐시 서버가 다운될 경우 알려주는 역할을 하고있습니다.

(7)

master역할을 하는 zookeeper에 slave인 ARCUS 서버를 등록하는 방법으로는 cache list에 참여할 캐시 클러스터의 이름인 service code 아래 자신의 주소인 znode를 저장하는 방식이 있습니다.

(8)

그리고 ARCUS는 ZK ping과 MC heartbeat를 사용하여 서버의 자동 fail stop을 제공하고 있는데, ZK ping은 주기적으로 ping을 교환하여 다운된 Znode를 감지, 제거할 수 있도록 하고있으며 MC heartbeat는 간단한 요청과 응답으로 서버 hang을 탐지하여 ARCUS 서버가 스스로 종료될 수 있도록 하고있습니다.

(9)

ARCUS의 클라이언트는 접속할 service code의 ARCUS 서버 목록을 조회하여 service 코드가 있을 경우 ARCUS 서버로 연결이 가능하며, ZK watcher를 이용하여 서버의 관리가 가능합니다.

ZK Watcher는 zookeeper를 관리하는 프로그램으로 Znode를 모니터링하다가 변경 이벤트가 생기면 최신 ARCUS 서버 목록을 조회하여 갱신하는 기능을 가지고 있습니다.

(10)

마지막으로 ARCUS의 아키텍처 스타일입니다. ARCUS는 MASTER – SLAVE의 구조를 가지고 있으며, 작업 종류 또는 시스템의 상황에 따라 sync 통신이나 async 통신으로의 전환이 가능합니다. write는 항상 마스터에서 처리하고 slave에서는 read만 가능한데, read는 데이터의 특성과 replication의 상태에 따라 master 또는 slave에서 수행될 수 있도록 응용에서 지정이 가능합니다.

master slave구조는 master노드가 다운될 경우, 전체 시스템이 다운된다는 단점이 있었습니다. ARCUS는 master node가 비정상적으로 종료될 경우, 기존 slave node중 하나가 master node로 변경되어 전체 시스템의 다운을 방지하고, replica group 전체가 비정상 종료가 될 경우에는 ARCUS의 ketama hash가 변경됨으로써 failure를 일정부분 방지할 수 있습니다.

이를 통해 저희는 시스템 아키텍트가 master slave를 이용하여 확장성 있는 서버 아키텍처를 구현함과 동시에, master slave가 가지고 있는 단점을 replication을 통해 극복하여 안정성도 추구하였음을 발견할 수 있었습니다.

이상으로 발표 마치겠습니다. 감사합니다.