RPG게임서버의 원월드 설계를 위한 아키텍처 접근 탐색

기술부 박선용

INDEX

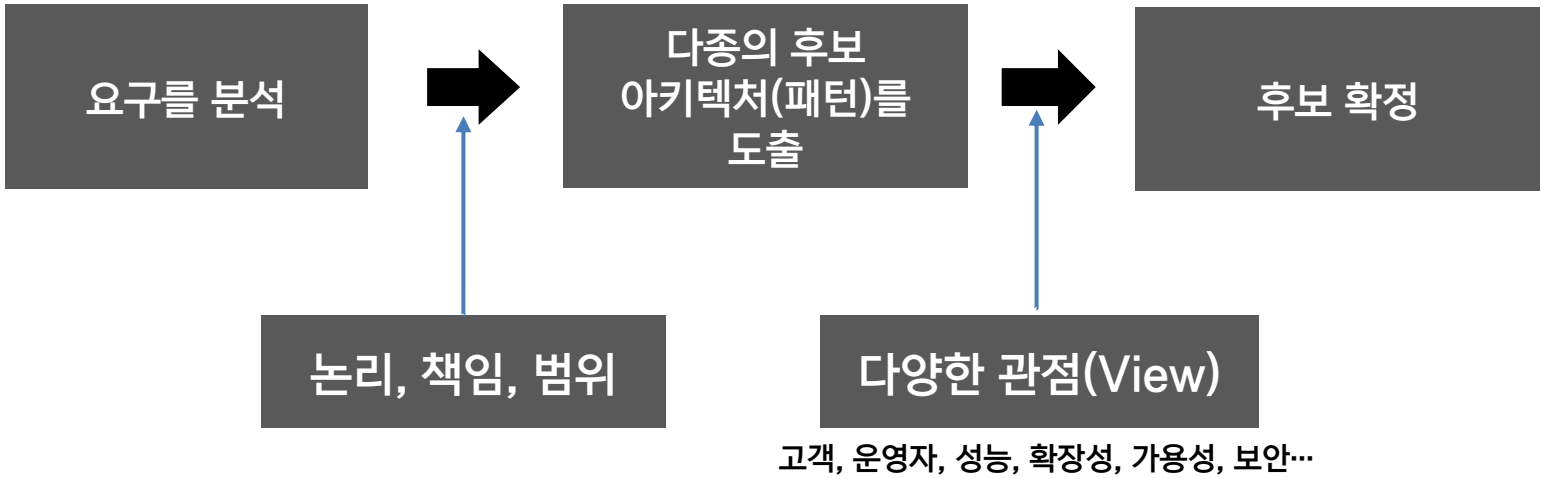
Architecture?

Architecture?

아키텍처?

- 프로그램이나 컴퓨팅 시스템의 소프트웨어 아키텍쳐는 소프트웨어 구성요소와 외부에 드러나는 특성, 그리고 구성요소들의 관계를 표현하는 시스템의 구조나 구조체를 말한다.
  소프트웨어 아키텍처 이론과 실재 (2008)

- What it does(The design of its functionality) 와 What it is(the design of its form) 을 결정하는 것



Architecture?

설계자란?

좋은 능력의 설계자란?	유사 업종에서 경험이 많음 자료 조사 능력이 매우 뛰어남 비슷한 업종의 전문가와 폭넓은 연계 컨퍼런스 및 학회 참여
설계시 유용한 방법	유사 시스템의 적용 경험 자료 조사 및 스터디 아는 사람에게 묻기
현실	고참 개발자 전문 직군화 부재 과연 좋은 설계란 무엇인가?
설계 유형	유사 시스템 설계 새로운 형태의 시스템 설계

⬇

효율적인 접근 방법은 무엇인가?

Architecture

영향을 주었던 개념들

1. 소프트웨어 공학
- 개발 프로세싱의 필요성

2. 소프트웨어 아키텍쳐링(CMU 등)
- 비즈니스 요구, 소프트웨어 구조, 품질속성, 참조패턴, 참조아키텍처, 설계전술, 평가

3. UML

4. 시스템 이론
블랙박스/화이트 박스
자극과 반응 이론　　Control Theory, 모델링 & 시뮬레이션
Chaos 이론

5. 성능분석론
- Metric, 3가지 접근 방법론(측정, 수리분석, 시뮬레이션), 큐잉 이론, Petri-Net

6. DA(Data Architecture)

7. 디자인 패턴

8. Agile 개발론
처음부터 완벽한 설계란 없다. 정확하게 정의할 수록 정확하게 틀린다.
Spiking, 점진적 진화, 커뮤니케이션

9. 다양한 도메인 경험
일반화와 특수화
미국 R&D 그룹과 대화 오픈소스, 디자인 철학?

Architecture

개념 범위

요소	범주	설명
성능 관점	품질 속성	. 정통적인 아키텍처링에서 중요한 요소 . 설계 후보를 선정하거나 시스템을 분할 통합할 때 매우 유용함
단일 책임의 원칙	설계 원칙	. 정통적인 소프트웨어 아키텍처링 뿐 아니라 다양한 영역에서 원칙으로 존재 . 설계 후보를 만들거나 설계를 리뷰 할 때 매우 유용함
Spiking	Agile 방법론의 주요 방법	. 빠른 후보자의 적합성 탐색 . 조건과 결과를 해석할 수 있는 능력이 있어야 함
아키텍처 평가 항목	설계 리뷰의 주요 방법	. 후보자 선정을 위한 정량적인 접근 . Apple to Apple 비교가 될 수 있도록 구성되어야 의미가 있음
오픈소스	후보패턴의 주요 대상	. 중요 후보 패턴(아키텍처) . 유용성의 범주를 파악하는 것이 관건

용어

후보 아키텍처	. 시스템의 행동과 품질 특성을 만족시키는 가능성을 가진 아키텍처 . Ex) 2tier/3tier
아키텍처 요소의 3대 특성	1. 명확한 책임들(A clearly defined set of responsibilities) 2. 명확한 범위들(A clearly defined boundary) 3. 명확한 인터페이스

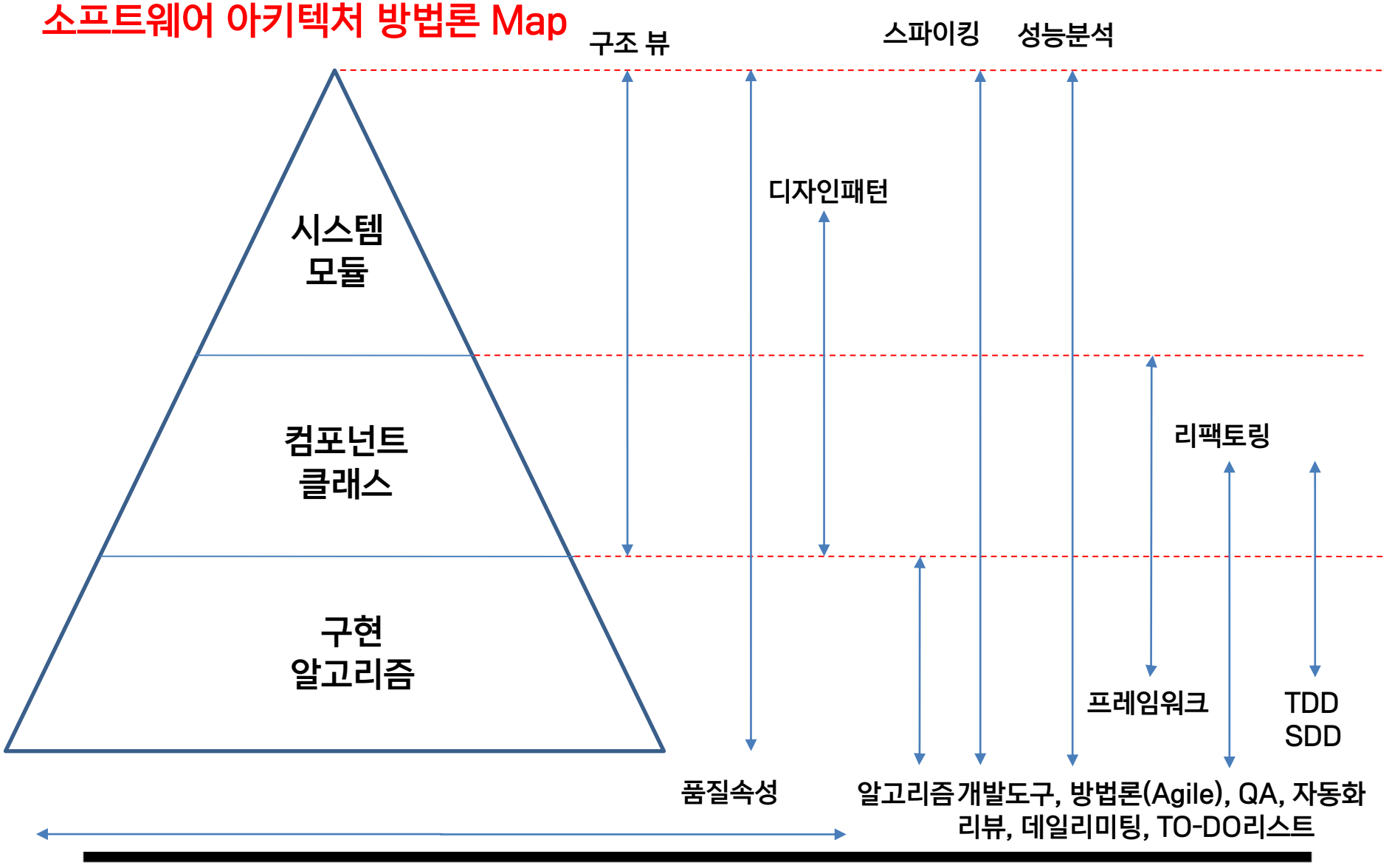
아키텍쳐에 영향을 주는 것

품질속성
(성능, 확장성,
가용성, 변경용이성..)

기능 요구사항

Architecture

소프트웨어 아키텍처 방법론 Map



무엇이 더 혁신적이었나?

인터넷 vs 세탁기
OOP vs 모듈 프로그래밍

Architecture

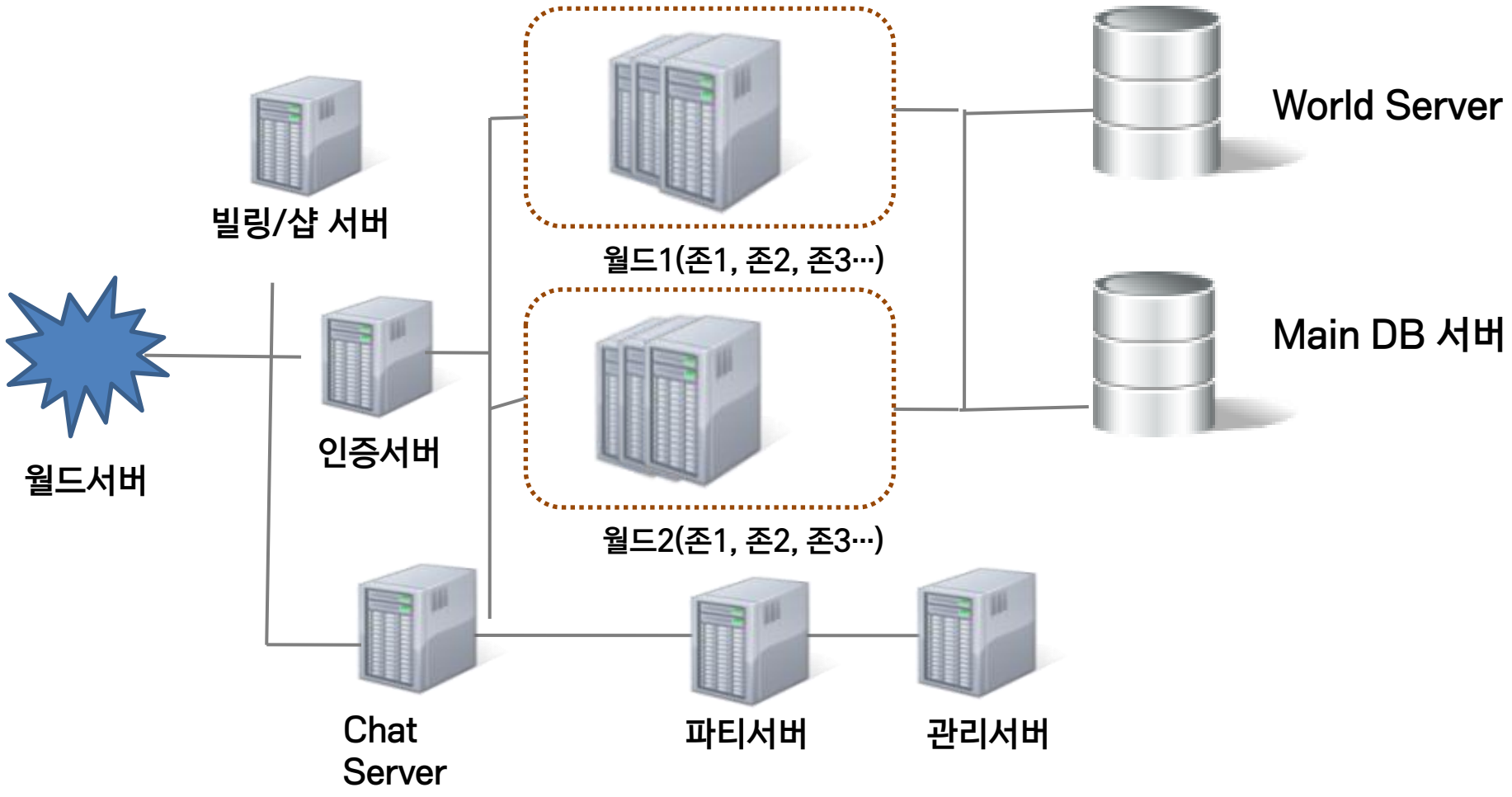
“완벽한 설계란 더 추가할 게 없을 때가 아니라, 더 뺄 것이 없을 때 이루어진다”

- 생텍쥐페리(프랑스 작가 겸 항공기 설계자)

RPG Game Server

Game Server

월드 분할 서버의 전형적 구조



월드가 분할되는 이유

- 데이터 저장장소 문제
  원 월드내 수많은 유저 정보를 실시간으로 저장할 수 있는 저장장소 문제
- 부하 분산
- 서버 용량의 한계
- 프로그래밍 모델 방식의 문제
- 월드의 크기

발생하는 문제

- 월드 분할에 따른 게임 유저간 데이터 분리
- 존 구성에 따른 이동간 몇 가지 문제 발생소지(딜레이, 중요 정보 분실)
- 공통 서비스를 위한 별개의 서버 존재
- 수평확장의 어려움, 관리의 복잡성
- 재활용 어려움

성능 개선을 위한 방법들

버퍼(큐) 객체
각각의 클라이언트들에 대한 Operation을 버퍼(큐)에 저장
유저 오퍼레이션을 매번 저장하지 않고, 일정 시간 간격(5~10분)을 두고 DB 저장

게이머 정보 메모리 로딩
한 서버내 동시 접속 게이머들의 정보를 서버 메모리에 모두 올려놓고, 게임 로직 실행시 참고
ex) 이동거리계산, 아이템 취득과 사용, 전투, 타격 데미지 등

Cache 서버 이용
일정 서버군을 묶어서 유저 정보, 오퍼레이션 등을 memcached/Redis 등에 저장

성능수치를 여기다 좀 넣을 것

Game Server

네오위즈 게임서버 예

- 각 게임 별 서버 구조 다양

- A형
  - Stateful 분산 객체 구조
  - Master군-Slaves 노드 존재
  - 서버 메모리에 유저 정보 등을 분산저장 → Fail-over 가능
  - 복잡한 구조 → 변경용이성 등에 한계 가짐

- B형(R서버)
  - 한 월드당 7대의 게임 서버(총 80여대) – DB Proxy – MS-SQL DB
  - 게임 서버 별로 부분적으로 데이터 캐싱
  - DB TPS : 800 TPS

M사 J서버의 예

- 전형적인 SOA(Service Oriented Architecture) 로 접근

- Scale-out이 가능한 게임서버 구축으로 원월드 서비스 지향

- C#환경에서의 통신기능이 내장된 서비스 컴포넌트로 분리

- 30대 물리서버로 동시 접속자 5만명 처리

J서버 설계특징

역할을 서비스로 분담, 가능한 작은 기능 단위를 서비스로 구현 → Fine Grained Services

'통합 네트워크' 레이어 서비스 간 통신을 중계 ← SOA의 Data BUS

통합 네트워크 라이브러리
서비스 구현, 서비스 간 통신 지원
서비스의 물리적인 위치를 게임 로직이 몰라도 됨
오퍼레이션: 서비스가 제공하는 기능 단위.
비응답 경우를 없애기 위해 항상 성공 또는 실패

서비스 로케이션 제공
서비스의 실제 위치를 다루는 서비스 → SOA SD(Service Directory)

one thread + event queue

하나의 TCP 연결을 통해 여러 오퍼레이션을 처리

테스트시 성능문제
한 서버가 한 개의 서비스를 처리하기 어려움
통합 네트워크 개선
성능 향상 (메시지 직렬화 성능 개선) + 서비스 분산 메커니즘 구현

Game Server-M사 J서버의 예

J서버 현존하는 문제

- DB가 병목 (단일 로그 DB 등에서 문제 발생)
- 오퍼레이션으로 실행되는 로직보다는 오퍼레이션 통신 자체로 인한 부하
- 엔터티 릭 현상
- Lag 현상 전파 : 한 서비스가 느려지면 그에 의존관계 있는 서비스가 모두 느려짐
- 과도한 옵저버가 존재하는 경우 내부 통신 급격 발생
- 프론트엔드가 너무 부하 큼(오퍼레이터 요청, 결과 처리, DB, IO 발생)
- 대부분 싱글스레드(스레드 안전성에 대해 신경쓰지 않아도 됨), 일부 멀티스레딩

견해

- SOA 아키텍처에서 Fine Grained로 인한 과도한 서비스간 통신 코스트 발생
- 저장 공간의 확장 한계성
- 게임서버내에서 SOA 형태의 아키텍처를 적용한 것은 커다란 도전으로 높이 평가

3

새로운 설계 접근

새로운 설계접근

아키텍처 목표

MMORPG 게임 서버 특징

Very low latency
Huge parallel operations of Users
Massive Users and interaction
Very quick game logic processing

목표

100만 동접자 원 월드 설계

설계 컨셉

낮은 지연과 방대한 유저 데이터 동시 처리
설계 품질 중 성능과 확장성 중심의 설계 전술 선택

➡ 성능 중심의 설계 전술 채택

새로운 설계접근

기본 설계 관점을 재점검

RDBMS의 문제
과거 파일 디비에서부터 RDB로 전환
현재 대부분의 게임 개발자들이 MS-SQL등에 익숙
mmo게임의 경우 CUD : R 비율이 50 : 50 혹은 그 이상일 수 있음. 통상적인 복제 DB는 의미 없음.

Concurrency 문제
싱글쓰레드 처리 : 이해하기 쉽고, 사용하기 편함.
과연 이것이 Concurrency 해결책?
큐를 이용한 멀티쓰레드 처리의 구조적 한계 극복 방법은 없는가?

서버확장
서버 기능별 분리가 필수인가?

OOP?
SOA는 OOP 개념과 다른 관점.
데이터와 처리, 인터페이스라는 절차적 프로그래밍 관점이 담긴 사상

분산처리?
분산컴포넌트? : 금융, 대규모 서비스 산업에서 많이 고려되었고, 성능상의 부분은 이미 많이 지적되어 옴
컴포넌트는 과중한 기능을 스스로 가지거나, 많은 기능을 컨테이너에 의존해서 처리하게 되므로 성능향상에 한계 존재
빠르고 안정적인 구현과 유지 보수성, 확장성에 장점 존재 → 고도의 성능이 요구하는 시스템에는 부적절

새로운 설계접근

RDBMS

일반적인 데이터베이스 Scale-out 방식
Master + Slave : Read 분산만 이루어짐. Read 오퍼레이션이 많은 경우에 적합

M-M Replication
복제나 기타 3rd Party 툴(ex 텅스텐)을 이용한 구성. CUD 확장의 한계

클러스터링
Oracle, MySQL 등

No-SQL
다양한 NoSQL 솔루션

RPG성 게임의 경우 높은 Updates 비율

➡ DB 클러스터링

새로운 설계접근

DB Cluster & NoSQL

MySQL Cluster

수평확장이 가능한 일종의 메모리 데이터베이스
물리적으로 분리된 메모리에 데이터 병렬 적재를 통한 99.999% Availability
8서버로 4.3M reads per second, 1.9M updates per second
30서버로 16M writes per second
7.2 버전에서 JOIN 성능향상, NoSQL API지원

카산드라

게임로그 저장에 적합
복잡하지 않은 로그성 데이터를 저장하기에 좋음
저 사양시스템에서도 높은 성능을 내는 현존하는 가장 좋은 아키텍처 중 하나
write 오퍼레이션이 read operation보다 훨씬 빠른 구조

MongoDB

상대적으로 Dynamic 쿼리에 장점. 쿼링 속도가 높음
특정 기능에서 MySQL등의 대용으로 사용 가능

CouchDB

데이터 축적이 주로 일어나고 변경은 상대적으로 작은 경우 유리
CRM, CMS

Redis

예측 가능한 크기의 데이터베이스 사이즈 안에서 매우 빠르게 데이터가 변하는 경우
Realtime Communication, Stock Price, 포털 주요 정보, 유저 정보, 세션 서버

새로운 설계접근

Stateless & 비객체화

OOP, 분산컴포넌트 → 로직과 처리데이터로 분리 & 객체간 통신 배제

OOP → 로직과 데이터 분리
게임처리를 위한 서버 로직을 데이터와 처리로직으로 분리
유저에 대한 처리 객체를 생성하지 않음.
특정 물리서버에 유저가 존재하지 않도록 event 처리 중심으로 구성

분산컴포넌트 → 로직 처리기, homogeneous 서버 구성
처리 프로세스(로직)을 모든 서버에 동일하게 배치
각각의 처리 로직은 서로 영향이 없도록 isolation화
처리 로직간 상호 통신 없음

큐 원형버퍼
Event – queue - single-thread → event – circular buffer - multi-thread

새로운 설계접근

Concurrency Single Thread?

Concurrency

단순히 두 개 이상의 Task가 병렬적으로 돌고 있는 것 아님.
문제의 소지 → 같은 리소스에 경쟁적으로 접근한다는 점. ex) 파일, 데이터베이스, 소켓, 특정 메모리 주소 등
멀티 쓰레드에서 성능 문제 및 개발 난이도가 높아지는 것은 Queue와 관련된 문제
Mutual Exclusion, visibility of change

Locking의 문제

상호 경쟁적으로 접근 요청이 많을 수록, Lock은 극단적으로 비싸지게 된다.
Lock 대신 CAS(Compare And Switch) 사용가능. X86 operation ex) *java.util.concurrent.Atomic* 패키지*
Lock기능을 이용한 Concurrency 프로그래밍은 어려움, CAS와 Memory barrier 이용할 경우 더욱 까다로와 지며, 정합성 검증자체가 불가능 할 수도 있음.

큐

처리 로직 사이에 이벤트를 큐에 넣었다 빼는 비용이 전체 비용의 대다수를 점유
통상 제안된 사이즈 큐를 선언. 무제한인 경우 메모리 재난! cf. 자바의 경우 GC문제
Consumer 와 Producer를 처리속도가 다르기 때문에, 통상 큐는 가득 차거나 비거나 하게 됨. → Lock 비용 극단적으로 높아짐.
통상 경쟁관계나 cache coherence가 발생하므로 매우 높은 비용을 요구하게 됨

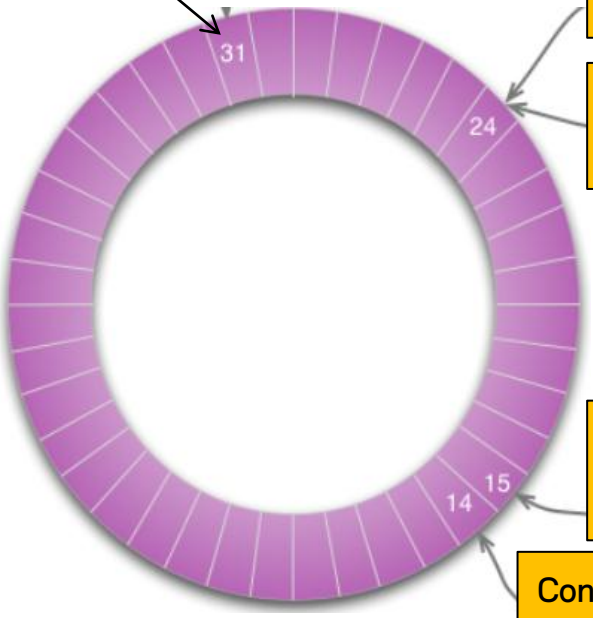
결론은 Single Thread?

➡ **원형 버퍼를 이용한 Concurrency 문제 해결.**

새로운 설계접근

원형버퍼

- 다수의 Producer와 다수의 Consumer를 둘 수 있음
- Producing과 Consuming 사이에 데이터 가공, 저장, 분류 등 다양한 Task 수행가능
- Multiple Task 관여하나 Lock Free → 각자 서로의 현재 Position 위치를 확인



Producer
사용자 요구 데이터 (id, 요구 기능, 상대 id, Message..)

Data Store – 인풋 데이터를 Sequential 하게 Disk에 저장가능
서버 Disaster의 경우 복구 가능

데이터 가공 ex) Un-marshaller

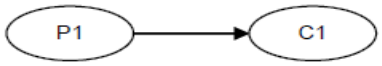
Pre logic processor
(데이터 검사, 로직 상세분배, 메시지 라우팅, 결과 라우팅)

Consumer
게임로직 처리기 – 처리 서비스 별로 분배(인벤토리, 메시지, 구매)

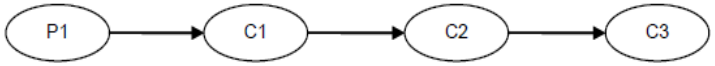
새로운 설계접근

원형버퍼 성능

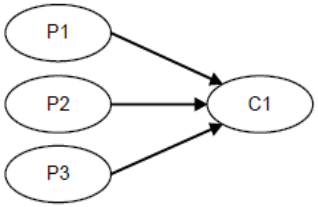
ABQ(Array Block Queue) – java.util.concurrent.ArrayBlockingQueue 와 비교



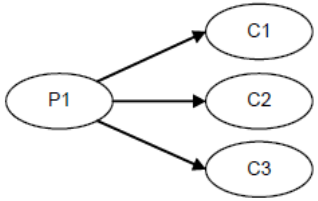
Unicast : 1P-1C



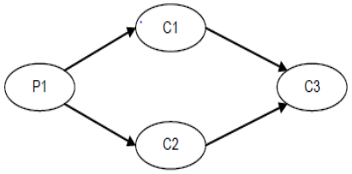
Three step pipeline : 1P-3C



Sequencer : 3P-1C



Multicast : 1P-3C



Diamond : 1P-3C

	Nehalem 2.8Ghz – Windows 7 SP1 64-bit		Sandy Bridge 2.2Ghz – Linux 2.6.38 64-bit	
	ABQ	Disruptor	ABQ	Disruptor
Unicast: 1P – 1C	5,339,256	25,998,336	4,057,453	22,381,378
Pipeline: 1P – 3C	2,128,918	16,806,157	2,006,903	15,857,913
Sequencer: 3P – 1C	5,539,531	13,403,268	2,056,118	14,540,519
Multicast: 1P – 3C	1,077,384	9,377,871	260,733	10,860,121
Diamond: 1P – 3C	2,113,941	16,143,613	2,082,725	15,295,197

Throughput (ops)

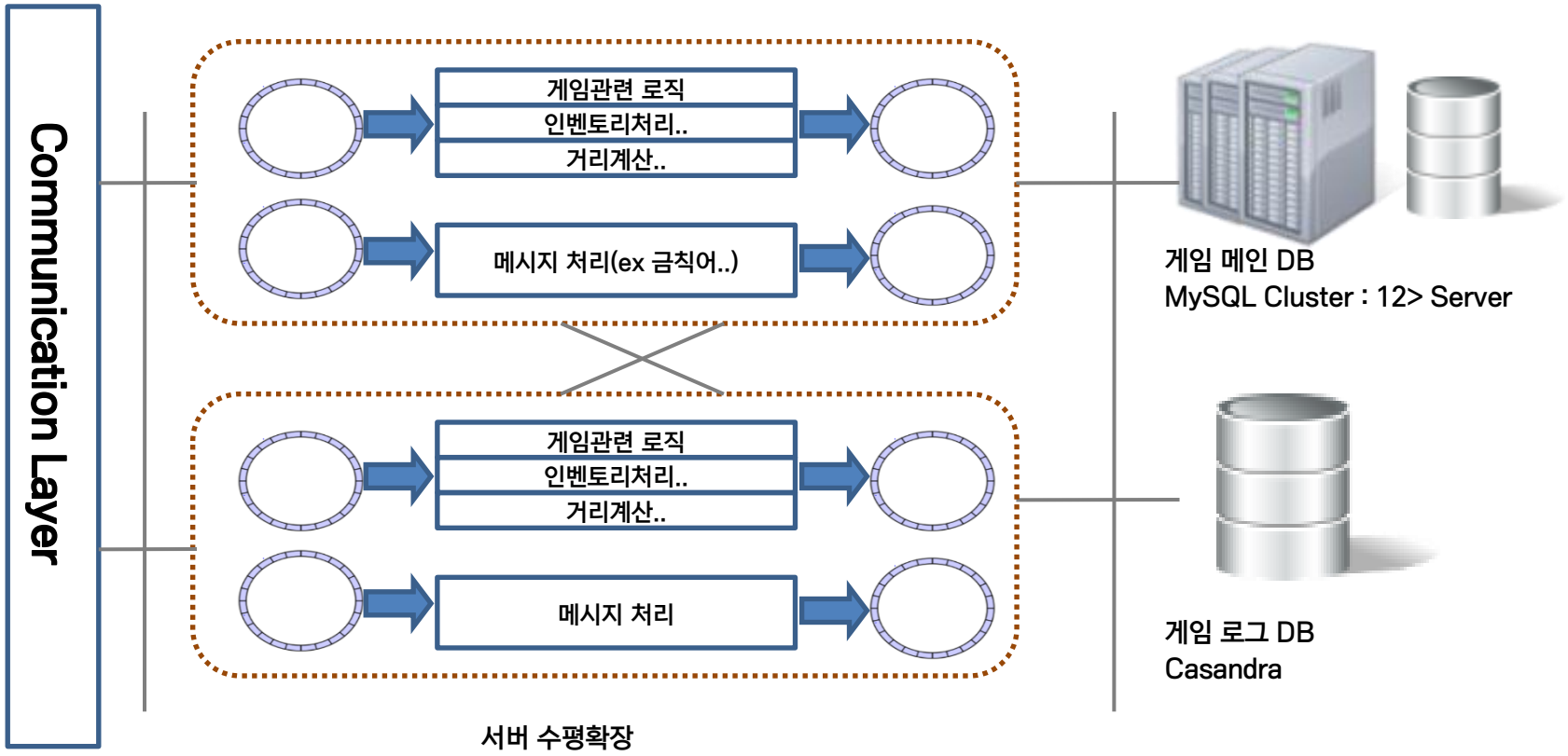
	Array Blocking Queue (ns)	Disruptor (ns)
Min Latency	145	29
Mean Latency	32,757	52
99% observations less than	2,097,152	128
99.99% observations less than	4,194,304	8,192
Max Latency	5,069,086	175,567

3 step pipeline latency

Disruptor 논문 데이터 인용

원월드게임 서버 아키텍처

- 서버 구성 : Stateless, Game Logic 처리기와 Pre-로직처리기들로 구성, 대부분의 데이터를 실시간 저장 → 수평확장 가능
- 목표 성능수치 : 동접자 100만, DB Throughput – 5M OPS (operations per second), 99.9 % 에서 100ms 이하 응답
- 1대당 기존 동접자 대비 5~10배 성능이 목표



새로운 설계접근

아키텍처 특징

- 데이터와 로직의 분리
- 게임 로직 처리의 Sequential Process 화
- 원형버퍼 + Multi-thread
- 서버간 통신 최소화
- Stateless + Homogeneous
- 게임 데이터 실시간성 저장
- DB 클러스터링
- NoSQL

- 각 개별 로직의 최적화도 반드시 병행되어야 함.
- 서버 로직 구현시 각종 최적화 테크놀러지 적용 필요 ex) factored code & small method, 에러 핸들링, 트랜젝션 처리 등
- 부분적으로 key-Value 메모리 서버 적용으로 DB부하 분산 가능

- 50대 (32 게임서버 + 12 데이터베이스 서버 + 6대 카산드라 서버) 로 100만 이상 동접자 처리 가능 예상. (1대당 3만명)

새로운 설계접근

참고 아키텍쳐

LMAX

Retail financial trading system

built on the JVM platform

Business Logic Processor can handle 6 million orders per second
(3Ghz dual-socket quad-core Nehalem based Dell server with 32GB RAM)

Without DB, in-memory processing

Event sourcing mechanism for disaster

IP multicasting

Distributed master-slave (all slave can replace master anytime)

향후 과제

개념 증명을 위한 Spiking
테스트 서버 구축과 성능 테스트

물리적 설계
사용 언어는?
여러 라이브러리나 framework 중 선택할 것은?
직접 구현 범위는?
Spiking 방안, 프로파일링 방법 및 성능테스트 방안?
최적화 방안은?

게임 서버 프레임워크화 가능성 여부 탐색 ex) Darkstar(Reddwarf)