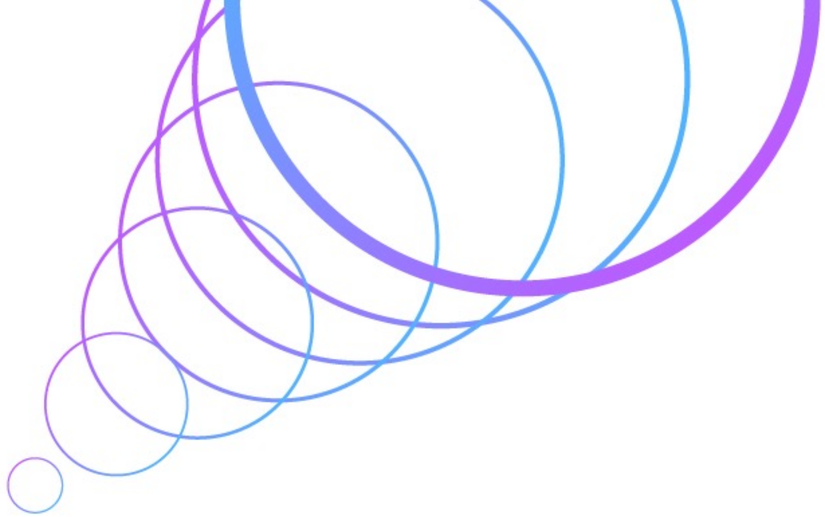


DEVIEW 2021

네이버는 대규모 메모리 클러스터를 어떻게 효율적으로 관리하나

이기열 Storage Platform

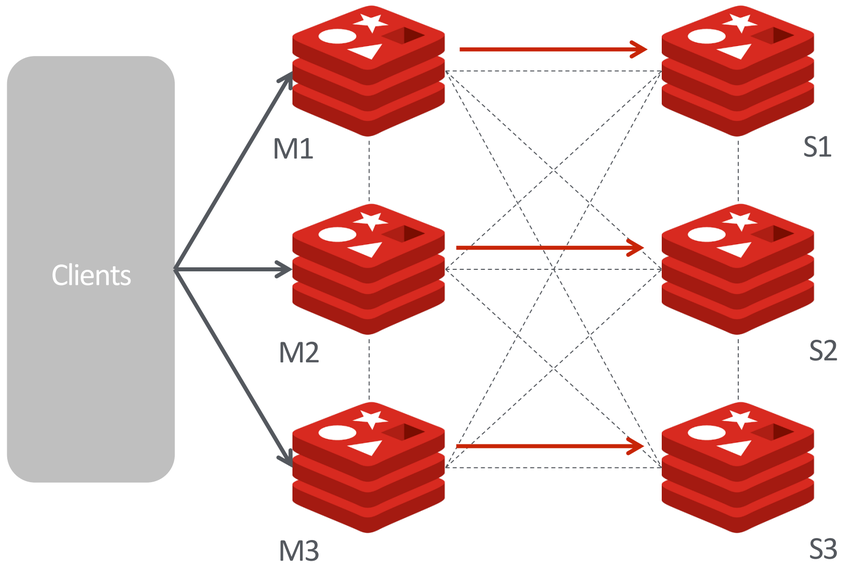
CONTENTS

1. 분산 메모리 스토리지의 구조

1.1 Open Source Redis Cluster

기본 구성

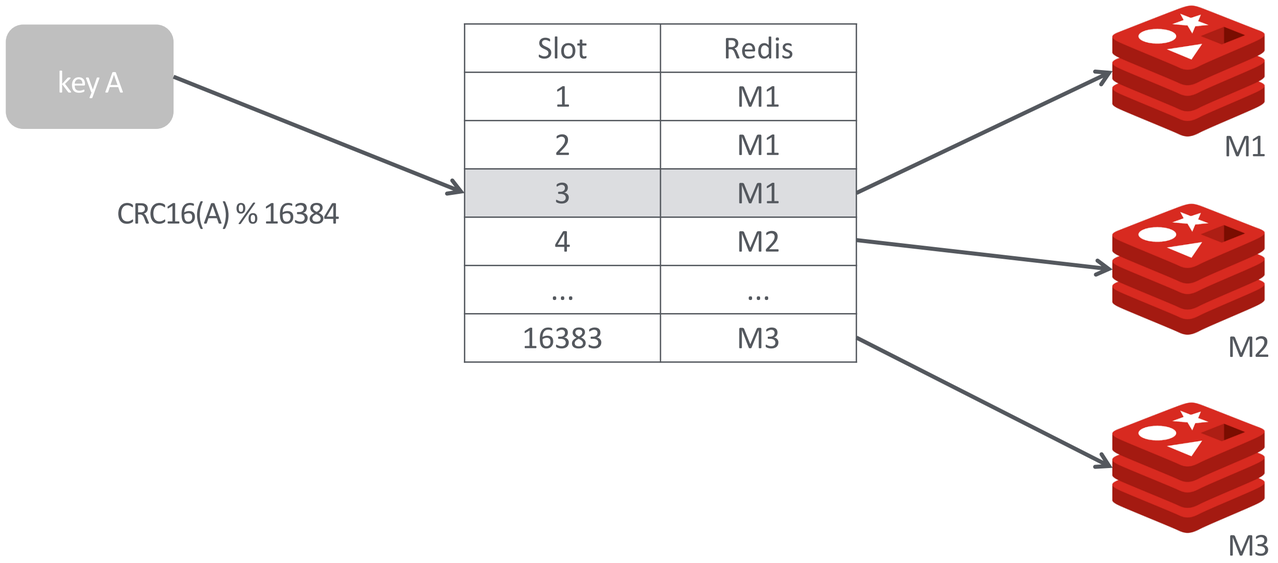
- memory 저장소
- cluster bus 로 자체 형상관리



1.1 Open Source Redis Cluster

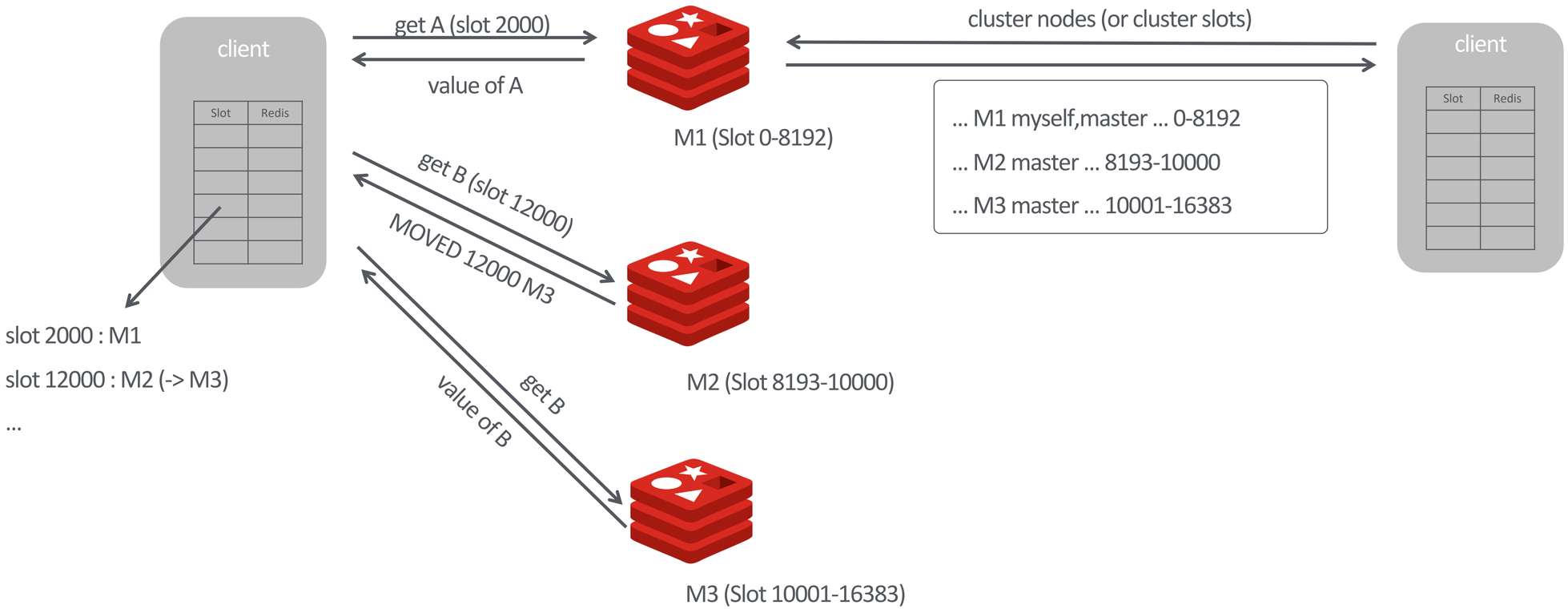
DEVIEW 2021

Partitioning



1.1 Open Source Redis Cluster

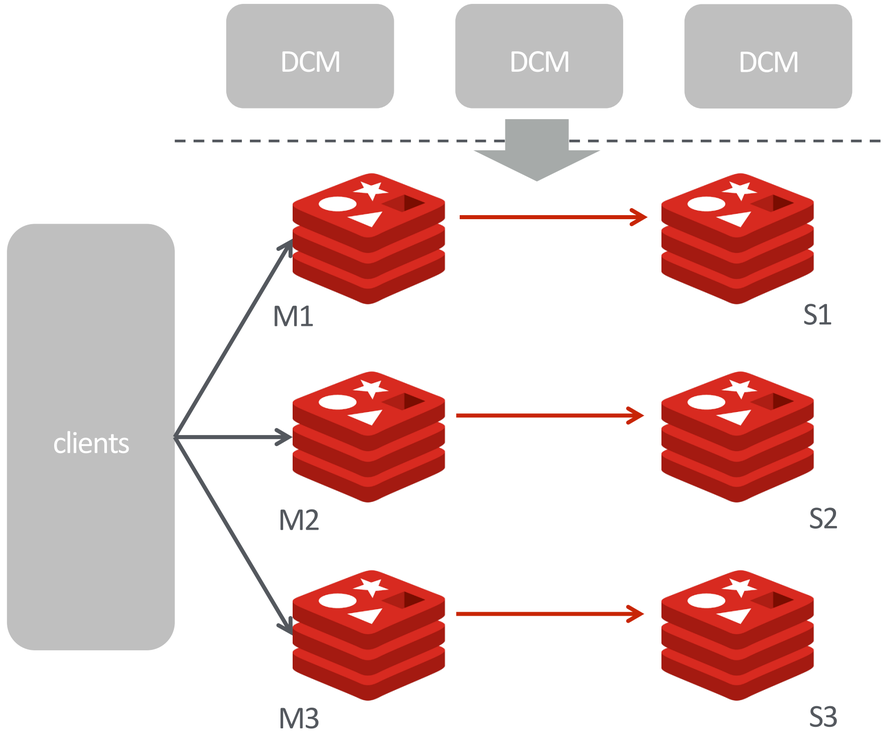
Clustering



1.2 Naver Redis Cluster

기본 구성

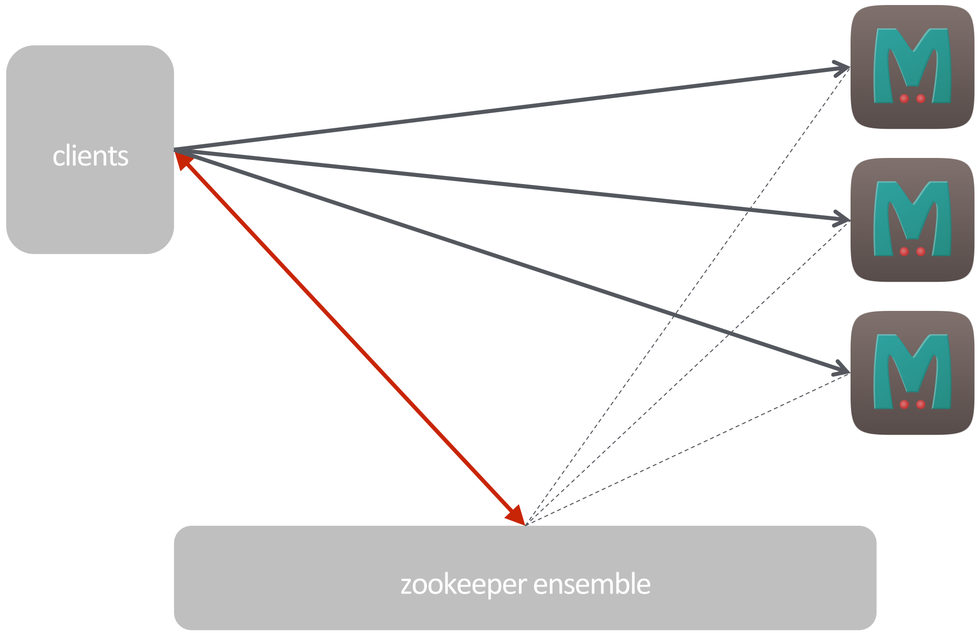
- DCM (Decentralized Configuration Manager) 을 통한 형상 관리
- Open Source Redis Cluster 와 API 호환을 유지하며 쓰기 안전성을 보장



1.3 Naver Memcached Cluster

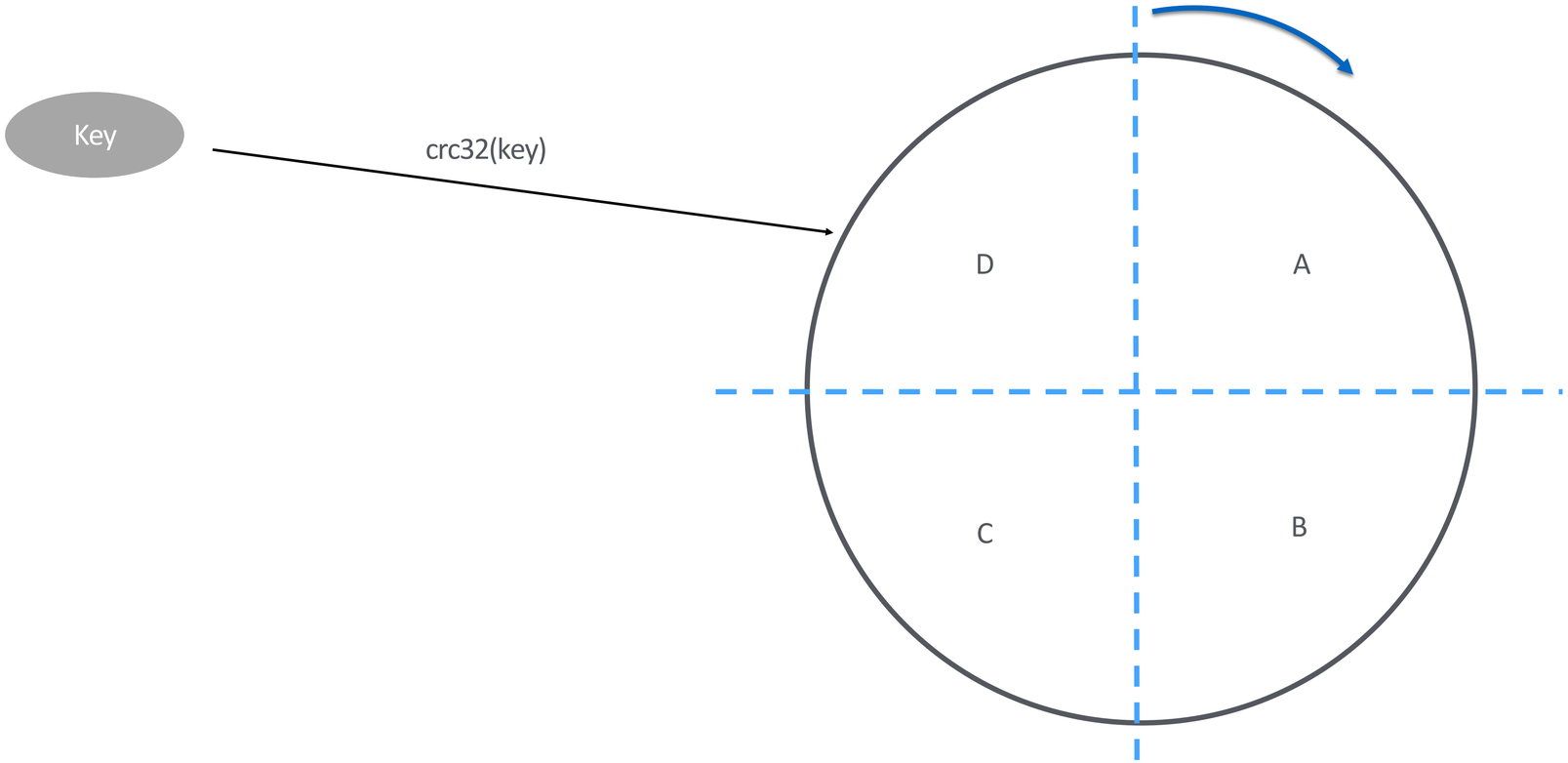
기본 구성

- Memory Cache Cluster
- ketama consistent hashing 으로 cluster 기능 제공



1.4 Ketama Consistent Hashing

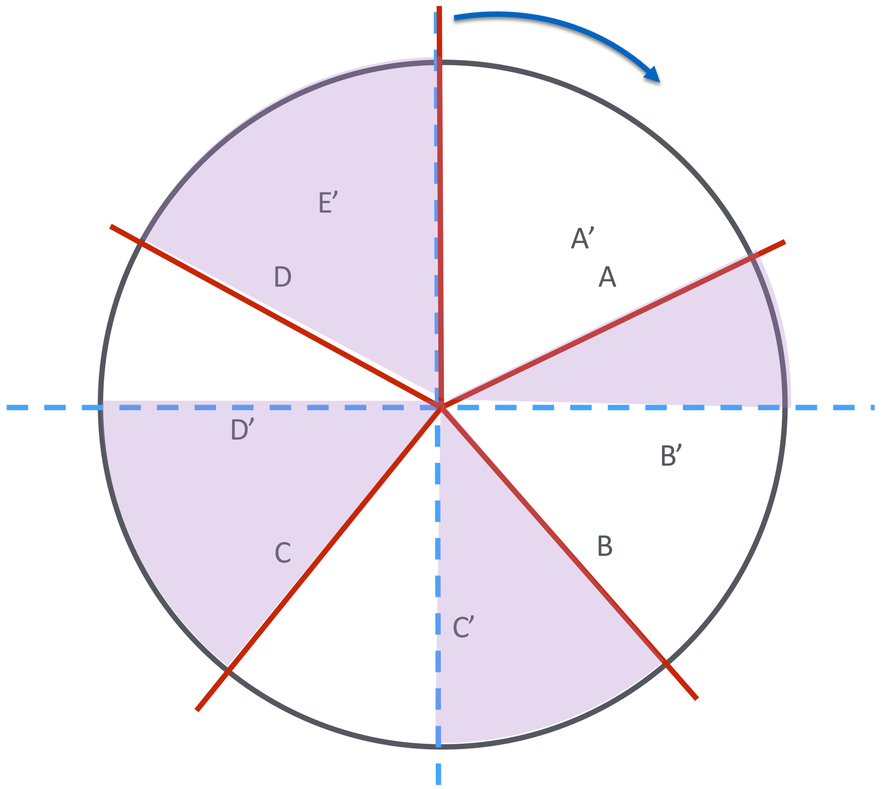
간단한 1/N hashing



1.4 Ketama Consistent Hashing

DEVIEW 2021

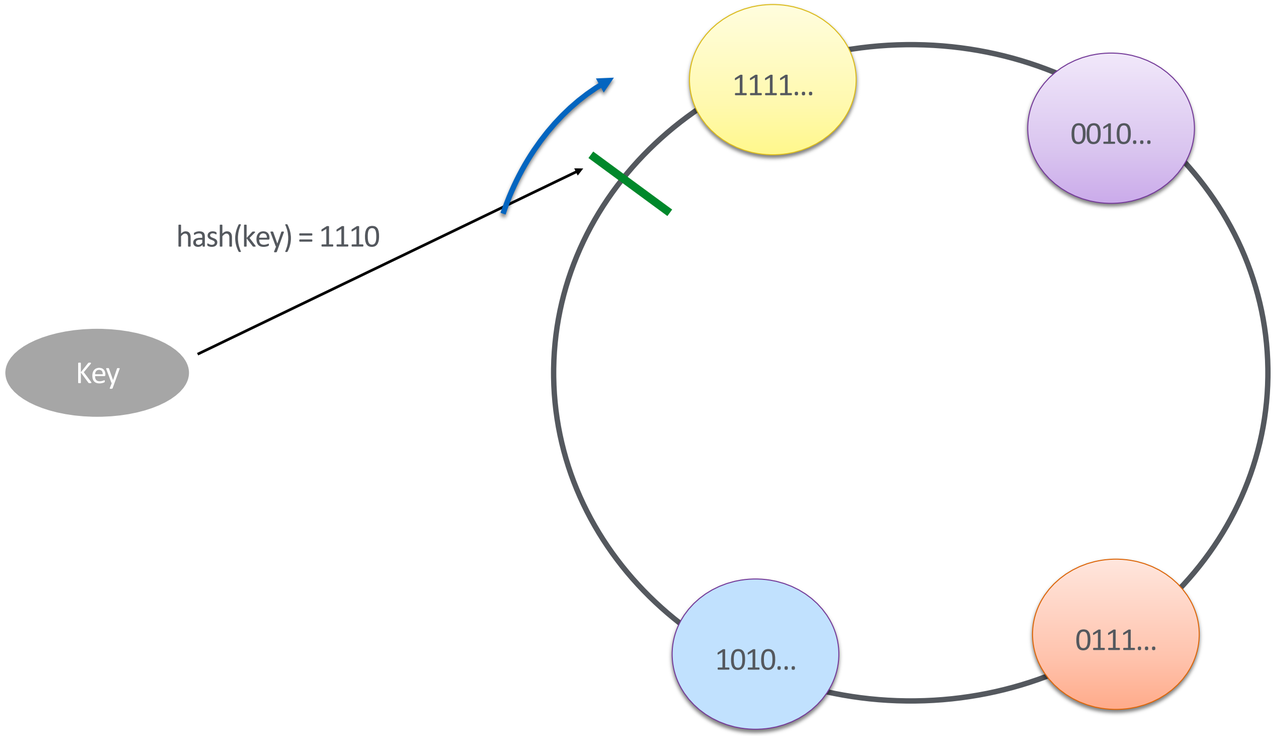
1/N hashing 의 단점



1.4 Ketama Consistent Hashing

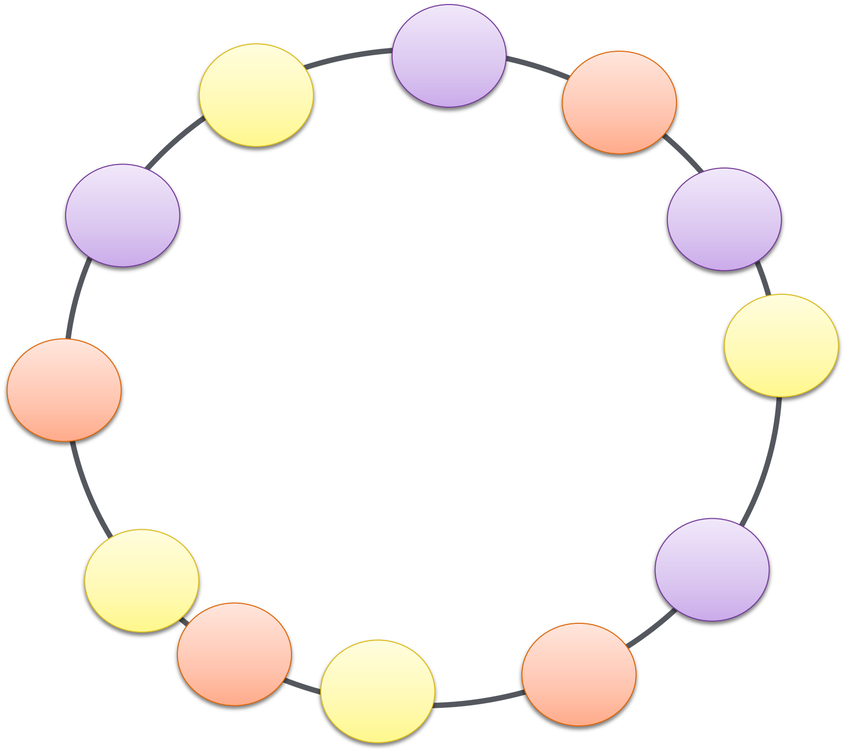
포인트 방식

- 서버를 hashing 하여 그 값을 포인트 hash ring 에 배치



1.4 Ketama Consistent Hashing

Hash Range 불균형 해소



2. 대규모 메모리 클러스터 운영 시 고려해야 하는 점

2.1 네이버 메모리 클러스터 운영현황

구성 클러스터: 1000+

서버대수: 1500+

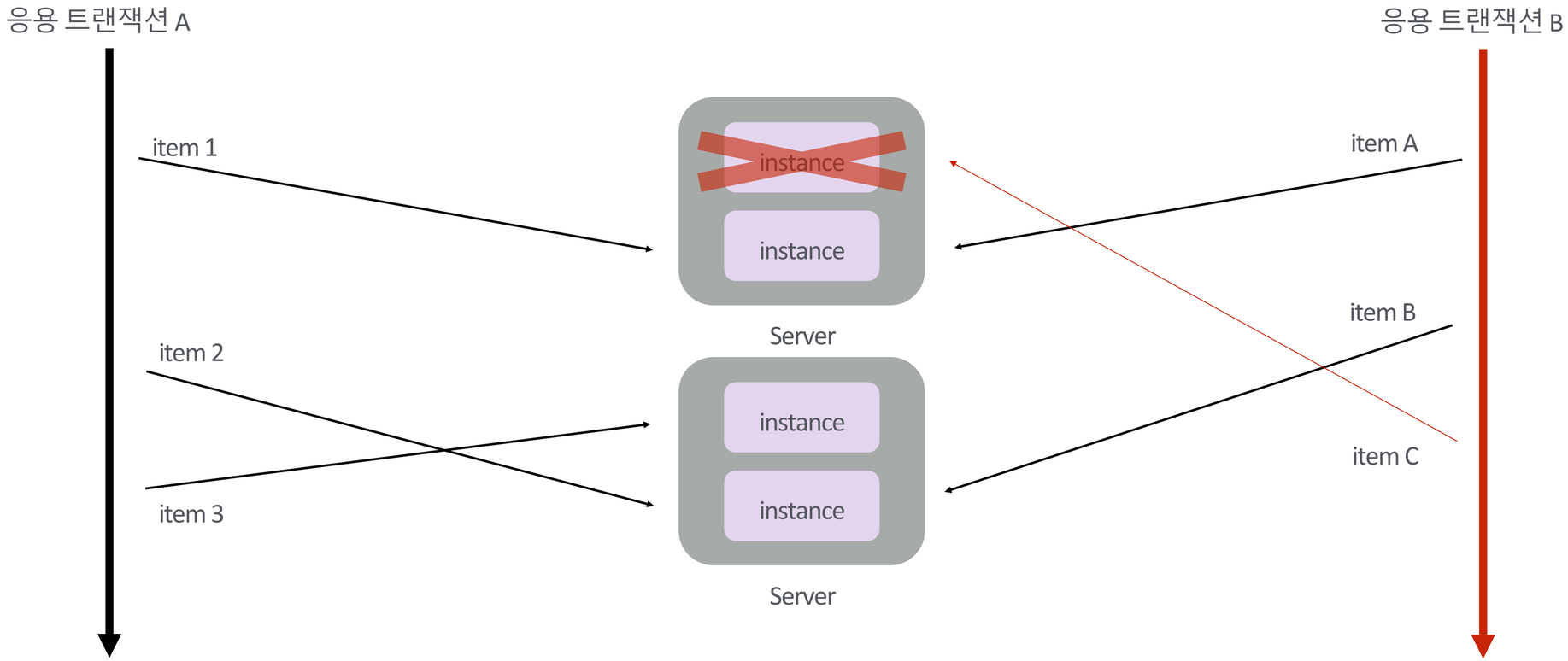
메모리 용량: 100TB+

평균 OPS: 20M+ / sec

2.2 일부 장애시에도 전면 장애 발생 가능

클러스터의 일시장애가 서비스 전면 장애로 파급될 수 있다.

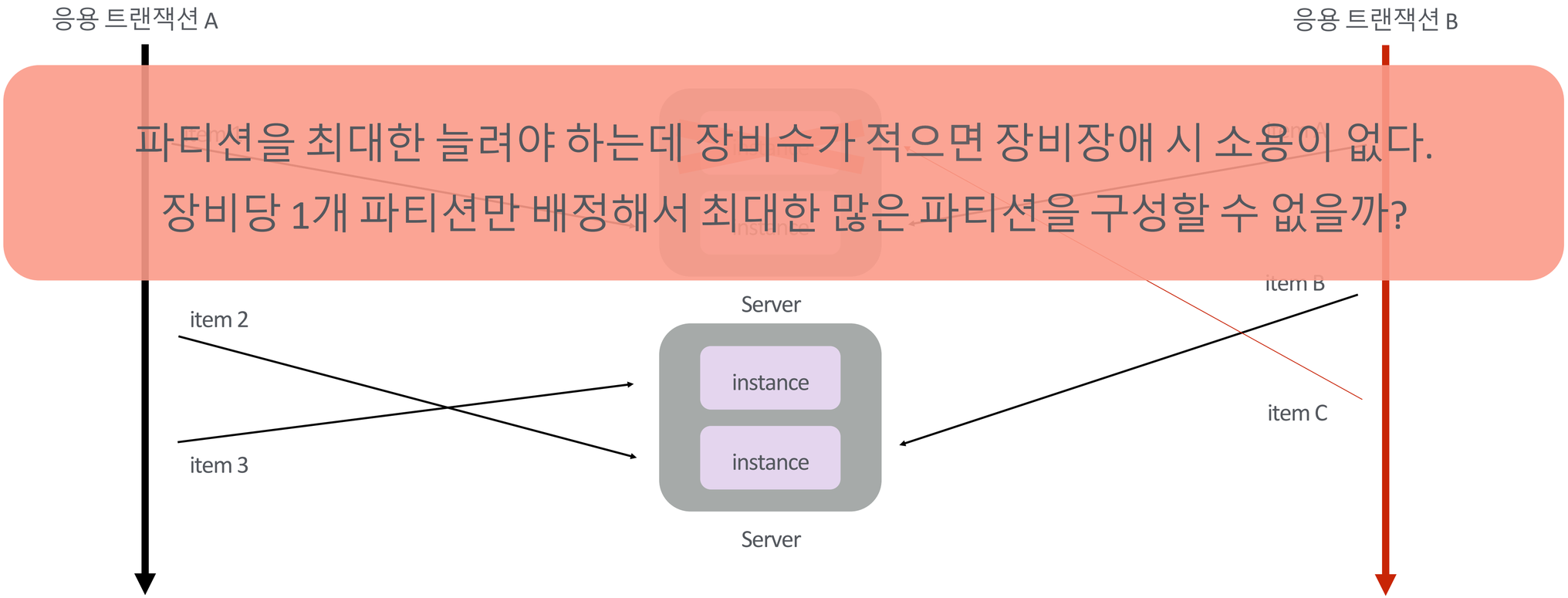
- 소규모 클러스터의 문제점
- 파티션 비율보다 큰 파급효과



2.2 일부 장애시에도 전면 장애 발생 가능

클러스터의 일시장애가 서비스 전면 장애로 파급될 수 있다.

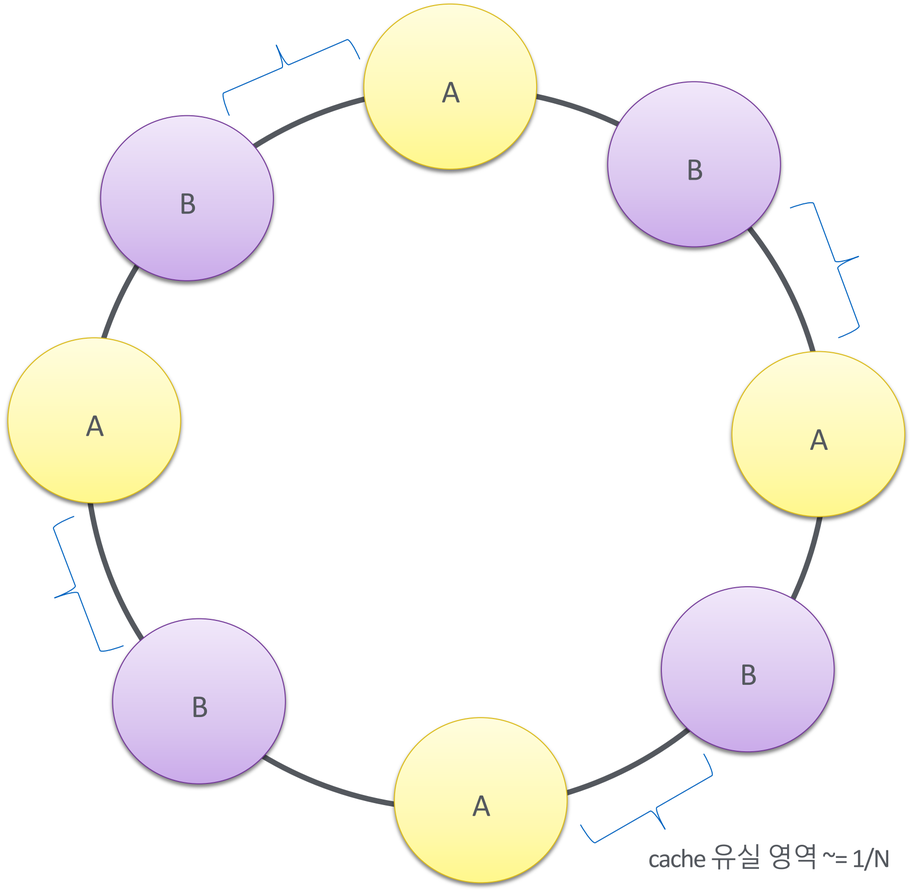
- 소규모 클러스터의 문제점
- 파티션 비율보다 큰 파급효과



2.3 Rehashing 의 영향

Rehashing 시 발생하는 cache miss 와 스토리지 부담

- A 의 담당 영역이 B 로 rehashing
- 유실 영역으로의 요청은 cache miss (~= 1/N)
- miss 비율만큼의 요청이 storage 로 전달

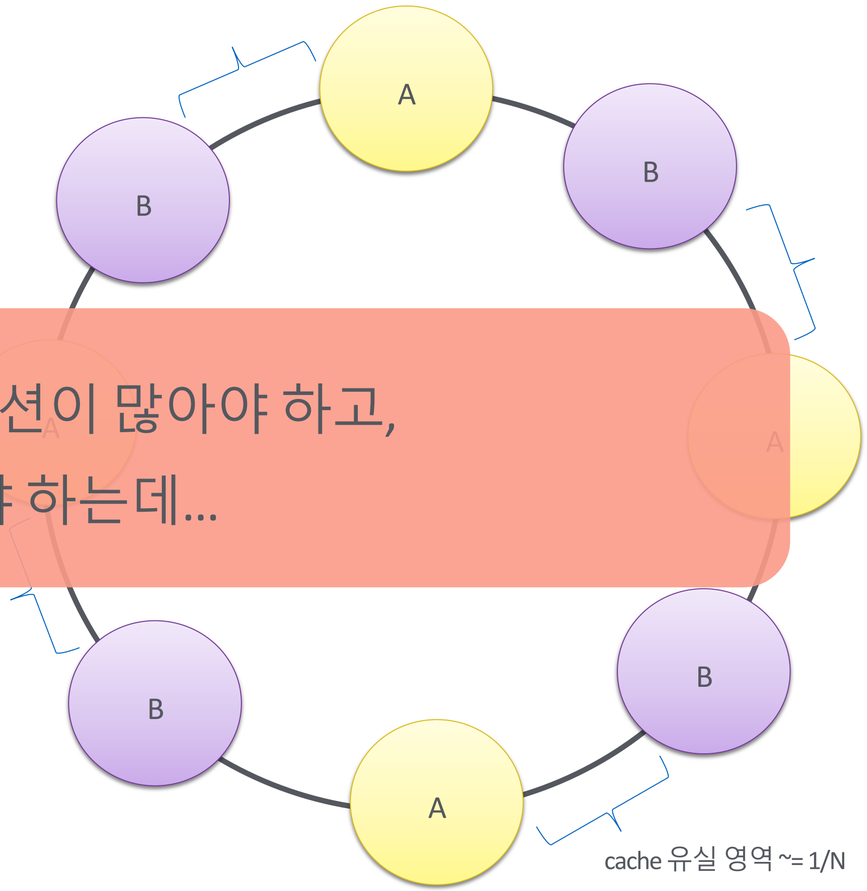


2.3 Rehashing 의 영향

Rehashing 시 발생하는 cache miss 와 스토리지 부담

- A 의 담당 영역이 B 로 rehashing
- 유실 영역으로의 요청은 cache miss (~= 1/N)
- miss 비율만큼의 요청이 storage 로 전달

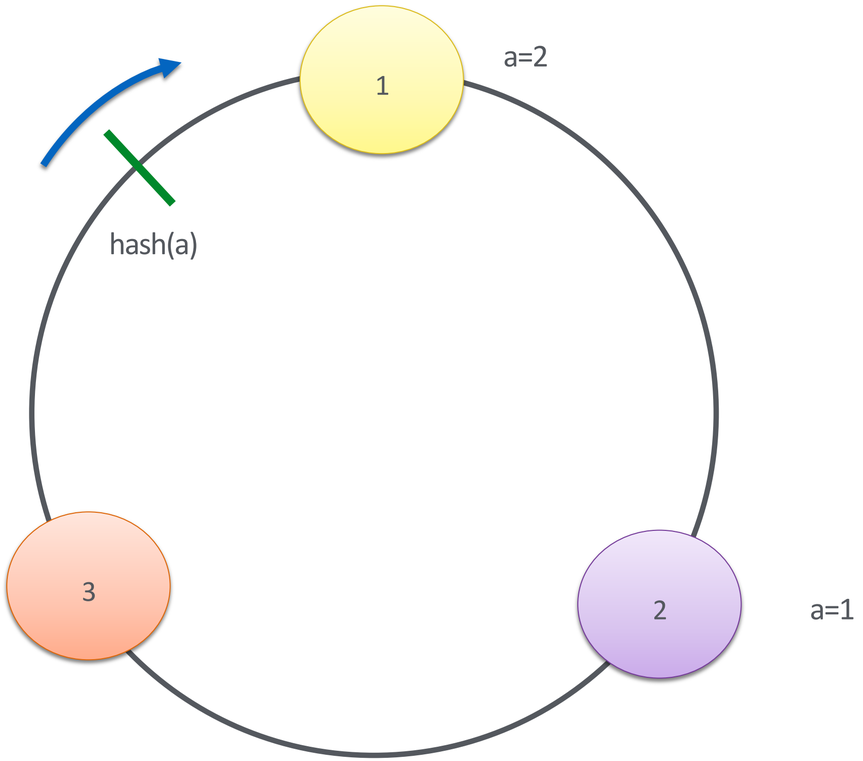
Rehashing 파급을 줄이려면 역시 파티션이 많아야 하고,
증설, 축소가 천천히 진행되야 하는데...



2.4 Old Data 참조문제

Failover, Failback 반복 시 old data 참조 문제

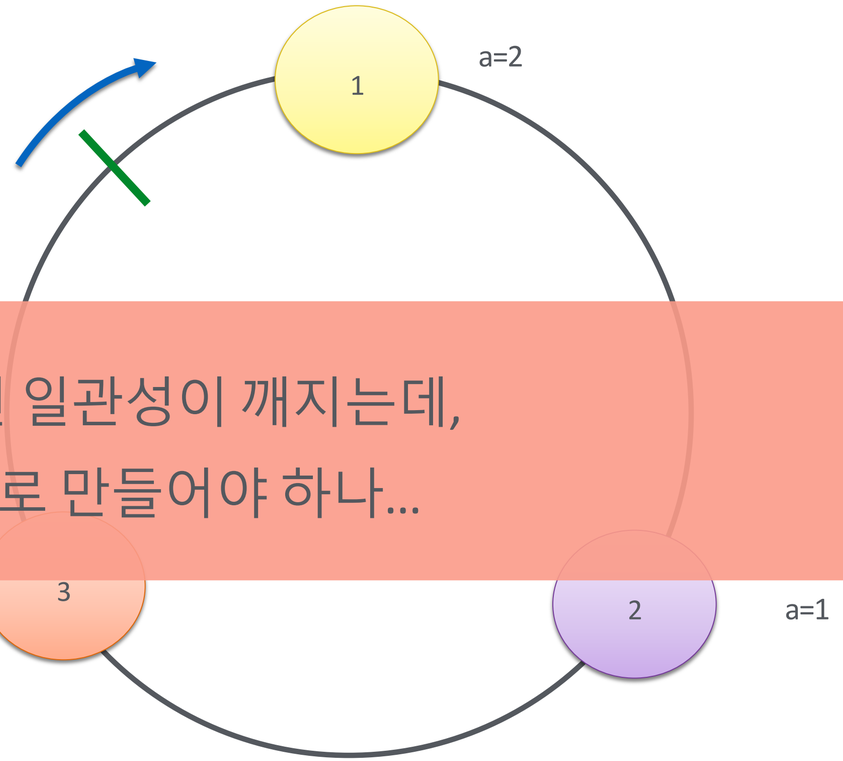
- (1번 노드가 없을 때) 2번에 a=1 저장
- 1번 노드 투입한 후 a=2 저장
- 1번 노드 장애로 제외
- 2번 노드에서 a=1 조회



2.4 Old Data 참조문제

Failover, Failback 반복 시 old data 참조 문제

- (1번 노드가 없을 때) 2번에 a=1 저장
- 1번 노드 투입한 후 a=2 저장
- 1번 노드 장애로 제외
- 2번 노드에서 a=1 조회



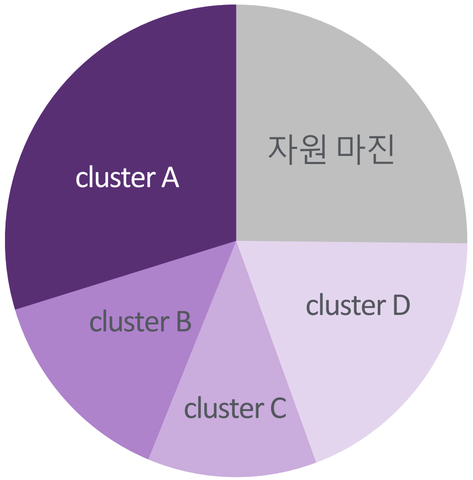
Old Data 를 scrub 하는 걸 잊으면 일관성이 깨지는데,
이를 관리해주는 전담자를 따로 만들어야 하나...

2.5 떨어지는 실사용율

실 운영 시 실사용율이 떨어짐

- 소규모 전용 풀의 문제
- 선구축으로 인한 갭
- 사용 양 급증에 대한 안전 마진

소규모 전용풀



공용풀과 자원 마진 상황

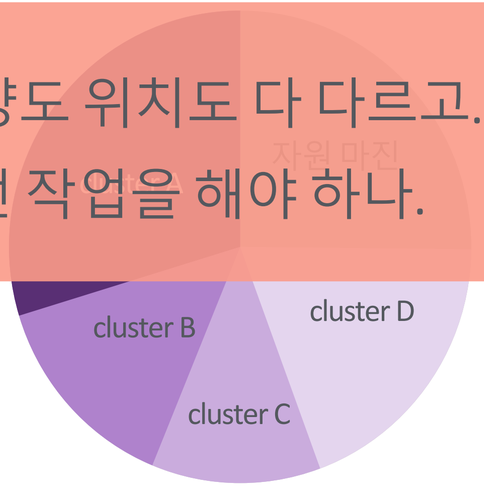
2.5 떨어지는 실사용율

실 운영 시 3X.% 대로 실사용율이 떨어짐

- 소규모 전용 풀의 문제
- 선구축으로 인한 갭
- 사용 양 급증에 대한 안전 마진

큰 풀을 만드는 게 좋긴 한데, 서버들 사양도 위치도 다 다르고...
신규 장비를 다시 구매하고 대량 이전 작업을 해야 하나.

소규모 전용풀



공용풀과 자원 마진 상황

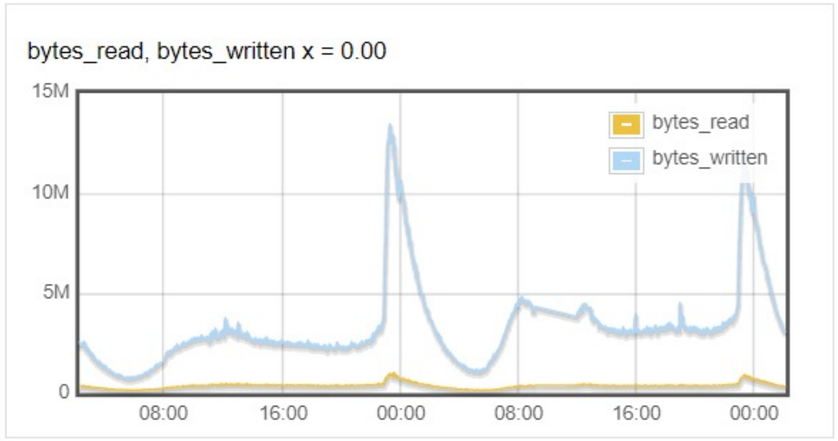
2.6 자원사용 패턴문제

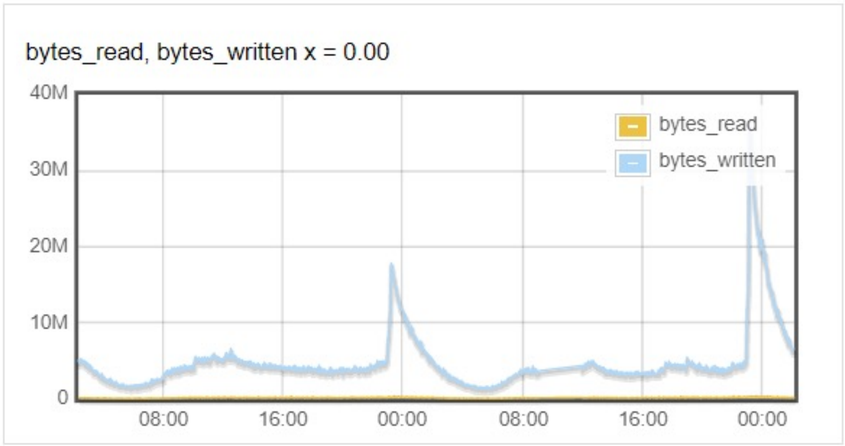
클러스터별 상이한 자원 사용 패턴

- network, CPU, memory 중 하나가 한계점에 먼저 돌입하는 경우가 많음

같은 서비스 그룹의 패턴 유사성

- 서비스별 전용풀을 구성하면 패턴 유사성으로 자원 평탄화가 되지 않음





11시에 오픈 되는 서비스 클러스터의 네트웍 사용패턴

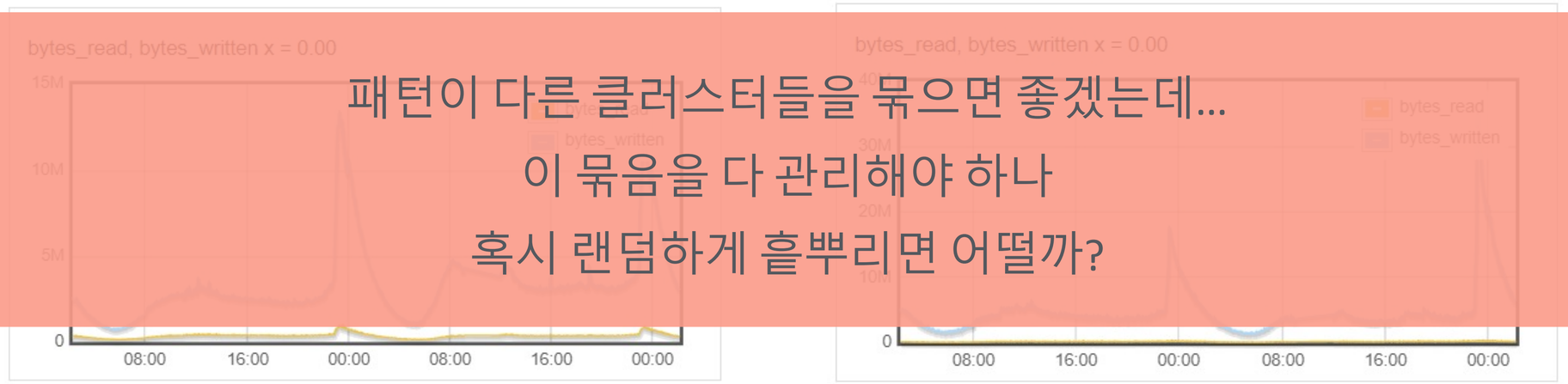
2.6 자원사용 패턴문제

클러스터별 상이한 자원 사용 패턴

- network, CPU, memory 중 하나가 한계점에 먼저 돌입하는 경우가 많음

같은 서비스 그룹의 패턴 유사성

- 서비스별 전용풀을 구성하면 패턴 유사성으로 자원 평탄화가 되지 않음



패턴이 다른 클러스터들을 묶으면 좋겠는데...

이 묶음을 다 관리해야 하나

혹시 랜덤하게 흩뿌리면 어떨까?

11시에 오픈 되는 서비스 클러스터의 네트웍 사용패턴

2.7 해결책

가용성, 효율성 문제의 해결책은 대규모 pool 에 구성

- 대규모 물리 pool 을 다시 구성하는 것보다 기존 K8S pool 활용

일부 장애시에도 전면장애 발생 가능 → 장비당 1개씩 파티션을 늘리고 싶다

failover rehashing 의 부담 → 파티션을 늘리고, 변경 작업을 천천히 진행되도록 하는 관리자가 있으면 좋겠다.

old data 참조문제 → 자동으로 old data 를 scrub 하도록 관리, 감시했으면 좋겠다.

떨어지는 실사용률 (자원 사용 패턴) → 자원 패턴이 다른 클러스터들을 큰 풀에 자동으로 뒤섞고 싶다

→ K8S Statefulset

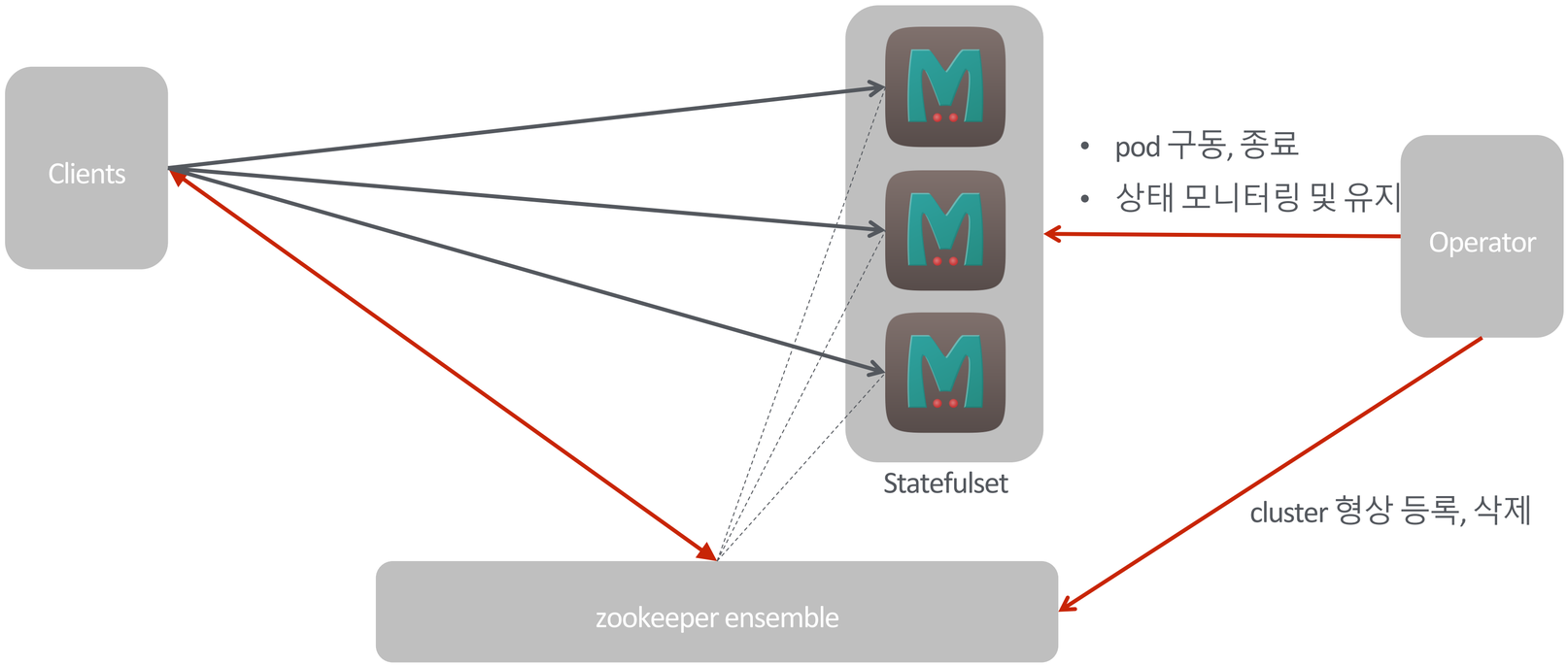
- 기존 구성된 K8S pool 로 전환
- operator 로 안전을 위한 감시, 관리 구현

3. K8S에 메모리 캐쉬 클러스터 적용

3.1 Cache Cluster 적용

Statefulset 으로 구성

- operator 가 cluster 관리
- 형상 정보는 기존 zookeeper 를 사용



3.1 Cache Cluster 적용

자원 격리는 Memcached 자체기능을 사용

- memory, CPU, disk 사용이 pod 설정 용량을 넘지 않음
- 관리 편의성 & k8s limit 초과로 인한 문제 차단

CPU 위주의 WAS pod 과의 시너지

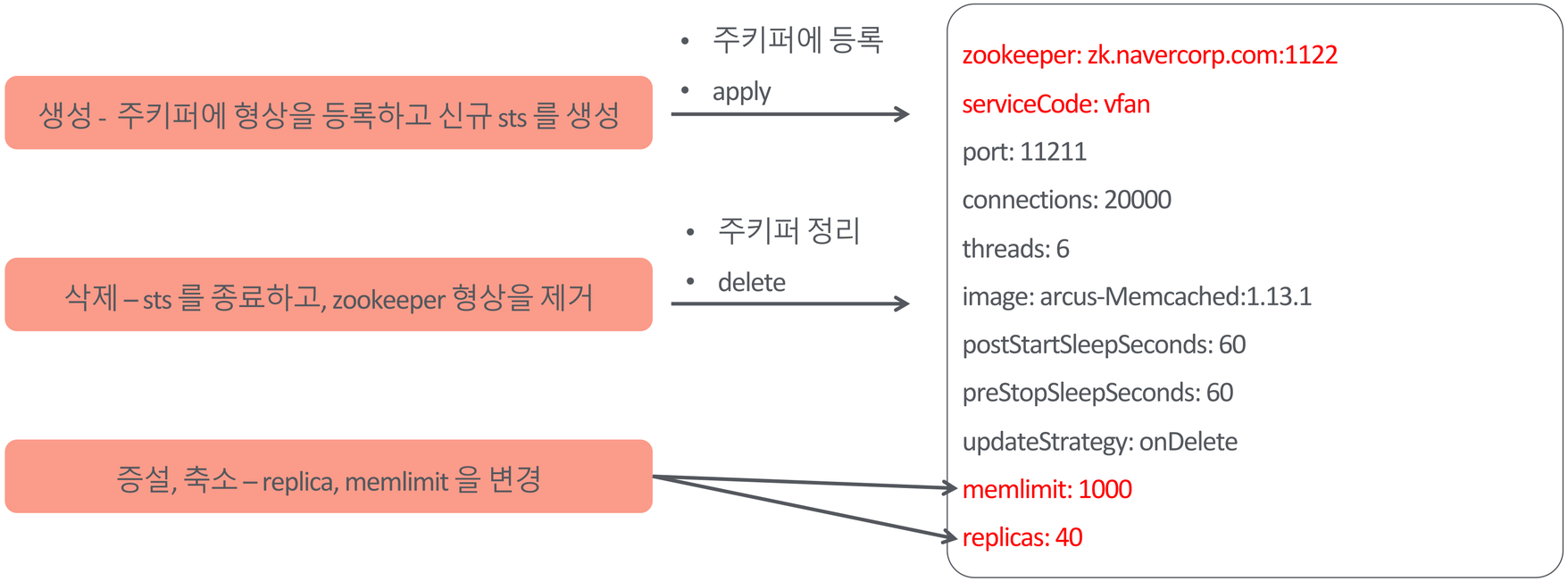
- cache 는 memory, network 에 주로 bound
- WAS 의 경우 보통 CPU에 bound

3.2 변경작업을 안정적으로 만들기

DEVIEW 2021

관리 안전성

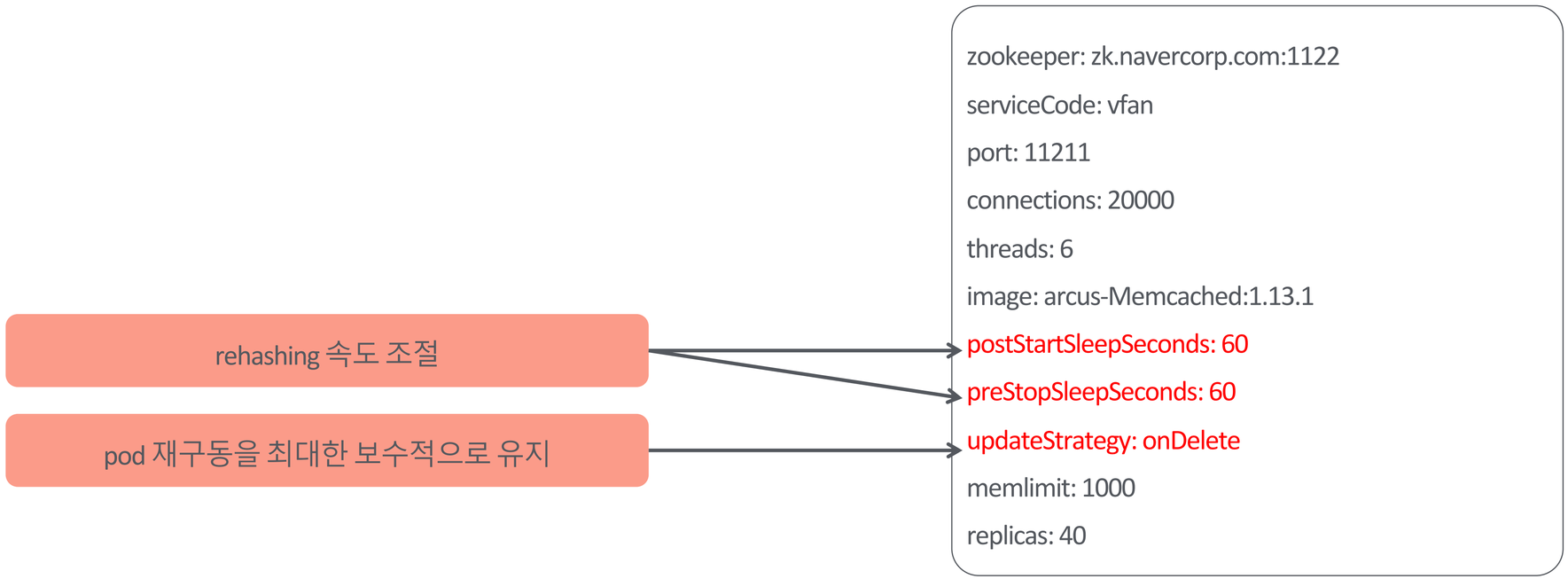
- statefulset 의 관리 항목이 복잡하여 cli, yaml 으로 관리 시 실수 소지
- 생성, 삭제, 증설, 축소로 운영작업을 제한하고 메타데이터로 검증



3.2 변경작업을 안정적으로 만들기

Rehashing 관리

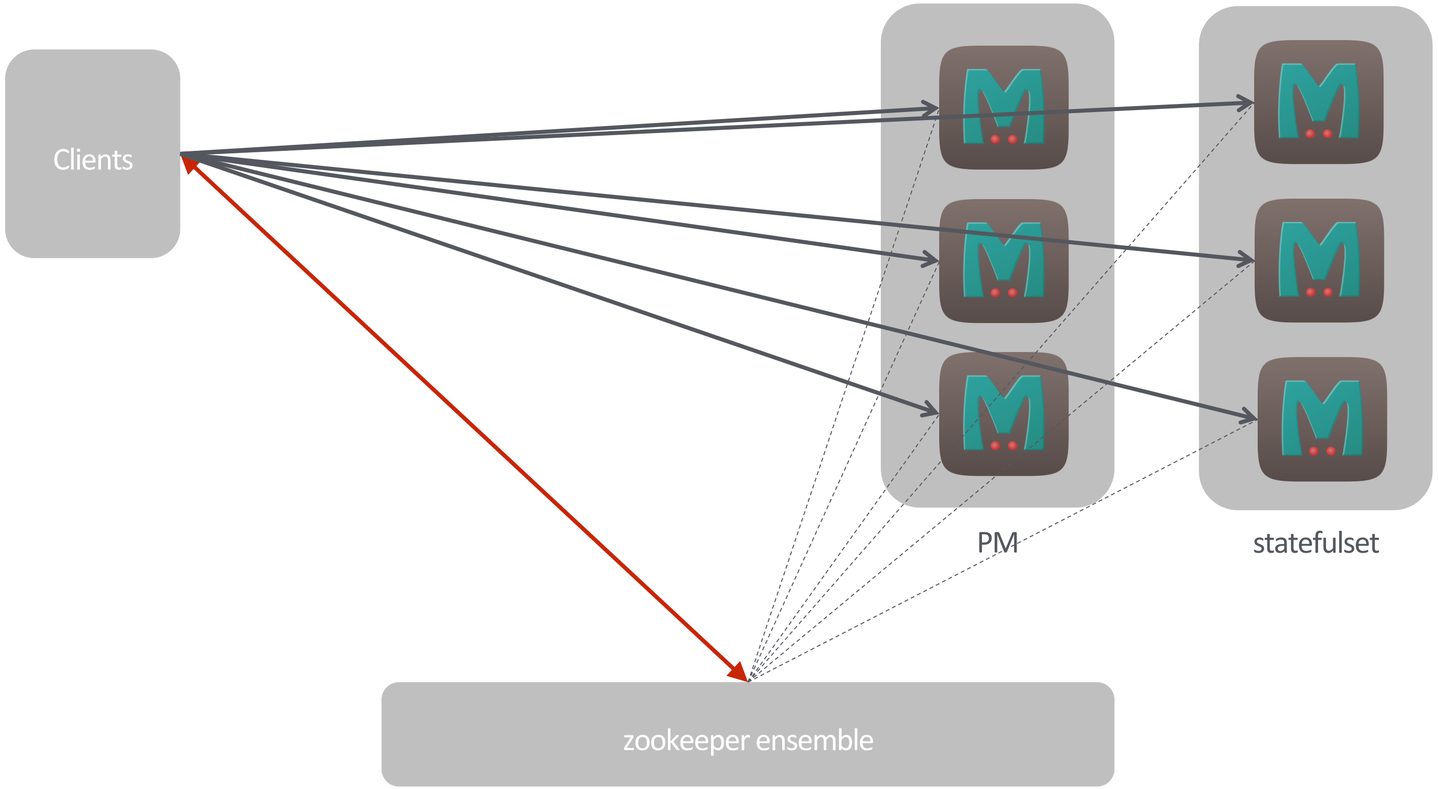
- updateStratagy 는 onDelete 로 운영
- replica 개수 변경 시 delay 도입 (rehashing 오버헤드 감소)



3.3 K8S 로의 마이그레이션

순차적인 이전

- 형상 정보가 주키퍼에 있어 병행 운영, 이전이 가능 (일부 native, 일부 pod)



3.3 K8S 로의 마이그레이션

DEVIEW 2021

K8S 풀의 분리

- 예상보다 잦은 pod 재시작
- replicaset 보다 높은 상태 안전성 요구
- 변경이 적은 K8S cold zone 을 설정하고 우선 배치

4. 모니터링, 트러블 슈팅 개선

4.1 모니터링 고도화 필요성

기존 체계의 한계

- 특성상 최소한의 로깅만 하고 있음 (이상상황 시의 분석 비용 높음)
- 손이 많이 가는 저빈도 모니터링 체계

가상화 환경에서의 모니터링

- 증설 및 대응을 빠르게 할 수 있는 기반 마련되었으나 여전히 수동작업

4.1 모니터링 고도화 필요성

기존 체계의 한계

- 특성상 최소한의 로깅만 하고 있음 (이상상황 시의 분석 비용 높음)
- 손이 많이 가는 저빈도 모니터링 체계

가상화 환경에서의 모니터링

- 증설 및 대응을 빠르게 할 수 있으나 여전히 수동작업

이상상황 발생 시 replica 변경으로 대응할 수 있지만,
자동화 하려면 이상상태 탐지가 선제조건 인데
1초단위로 수집 확인할 수 있을까?

4.2 고빈도 모니터링 - 요구사항

DEVIEW 2021

고빈도, 고성능 수집능력

- 수천개의 장비, 장비 당 수천 metric
- 빠른 조회, 집계 속도

확장성 및 관리 용의성

- 네트워크 bandwidth, 저장용량

4.2 고빈도 모니터링 - 요구사항

DEVIEW 2021

고빈도, 고성능 수집능력

- 수천개의 장비, 장비 당 수천 metric
- 빠른 조회, 집계 속도

확장성 및 관리 용의성

- 네트워크 bandwidth, 저장용량

- scale out 부담없이,
  1초 단위 수집을 위해 장비에 구성
- 관리 편의성을 위해 단독 구동, 단일 경량
  프로세스

4.2 고빈도 모니터링 - 요구사항

고빈도, 고성능 수집능력

- 수천개의 장비, 장비 당 수천 metric
- 빠른 조회, 집계 속도

확장성 및 관리 용의성

- 네트워크 bandwidth, 저장용량

서비스에 대한 영향 최소화

- Disk I/O, Network ingress/egres
- Memory Usage

- scale out 부담없이,
  1초 단위 수집을 위해 장비에 구성
- 관리 편의성을 위해 단독 구동, 단일 경량
  프로세스

4.2 고빈도 모니터링 - 요구사항

고빈도, 고성능 수집능력

- 수천개의 장비, 장비 당 수천 metric
- 빠른 조회, 집계 속도

확장성 및 관리 용의성

- 네트워크 bandwidth, 저장용량

서비스에 대한 영향 최소화

- Disk I/O, Network ingress/egres
- Memory Usage

- scale out 부담없이,
  1초 단위 수집을 위해 장비에 구성
- 관리 편의성을 위해 단독 구동, 단일 경량
  프로세스

- disk I/O 를 방해하지 않고 빠른 처리를 위해
  메모리에 기록,
- 단 time series 압축으로 최근 1주일간 유지

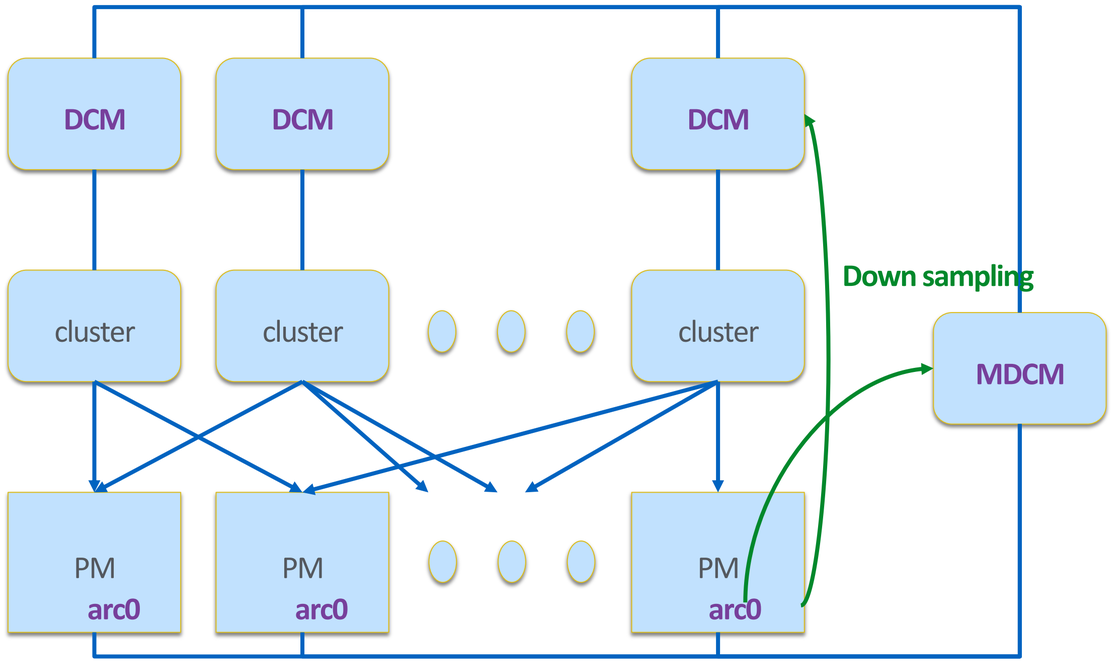
4.3 고빈도 모니터링 – Arc0

분산화된 in-memory TSDB

서비스	• **RESP (redis protocol)** • **간단한 명령어 셋 (**tls, tquery, tsdb-put, tsdb-metric-del...)
수집	• **/proc 을 사용한 기본 시스템 모니터링 (CPU, Memory, File System, TCP 등)** • **서버 프로세스 (사전 정의된** Command line pattern을 이용해 자동 수집)
TSDB Lib.	• **ctrie 를 활용한** Snapshot isolation. 다수의 reader가 writer와 무관하게 scan • Metric의 data point를 diff 값으로 변경하고 **zstd** 압축 (16 byte -> 1.6 byte) • 초당 수만 data points put, 수천만 data points scan

4.4 관리 모니터링 체계

단기 고빈도, 장기 저빈도 모니터링 체계



클러스터 관련 metric 5분 평균 저장

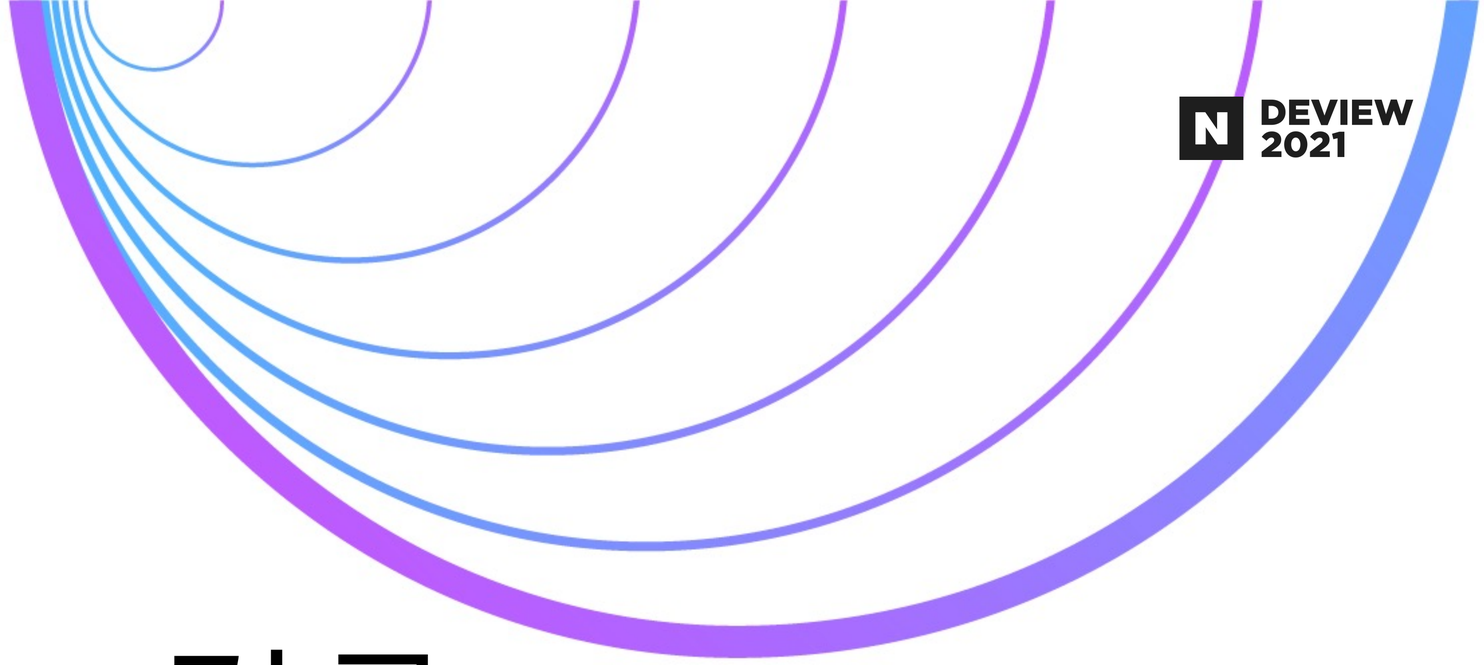
➔ workload 변경 detection 등의 관리 고도화에 활용 (예정)

PM 관련 metric 5분 평균 저장

➔ 리소스 할당, 추이 예측 등의 운영 자동화에 활용

장비 별 고빈도 모니터링

➔ 문제 파악 및 drill down에 활용



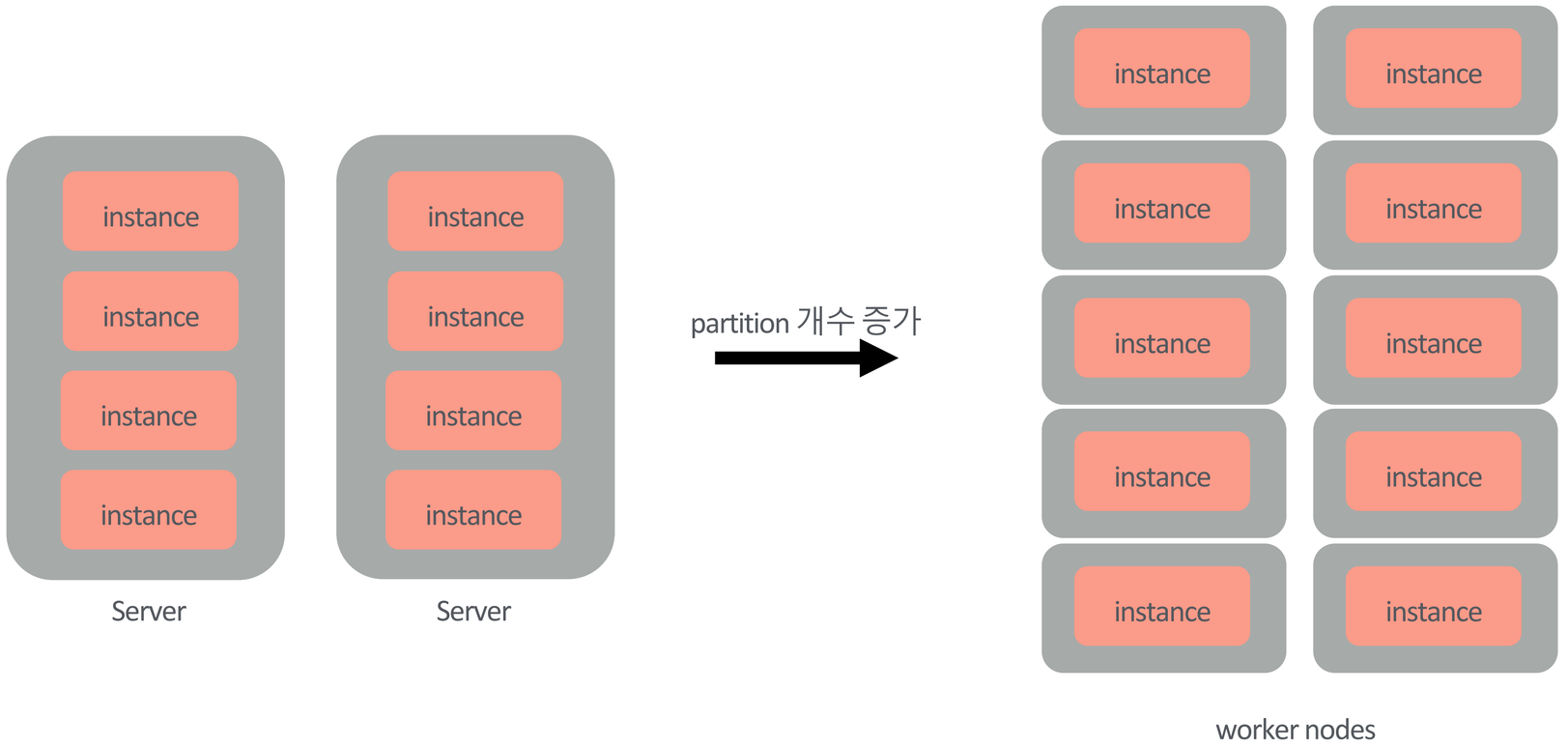
5. 결론

5.1 파티션 증가로 가용성 향상

DEVIEW 2021

작은 pod 을 더 많이 투입하여 partition 개수 증가

- 추가 비용 없이 기존 대비 2배 이상 파티션 증가, 가용성 향상



5.2 관리 편의성 향상

DEVIEW 2021

장비 관리를 하지 않고 볼륨으로 관리

- 증설 작업이 장비 상황, 운영 작업에 의한 지연 없이 가능
- 부하 급증 발생 시 자동 대응 가능

모니터링과 연계한 자동관리로 발전

- 이상상황 탐지 및 자동 대응
- 부하 예측과 사전 대응

Thank You