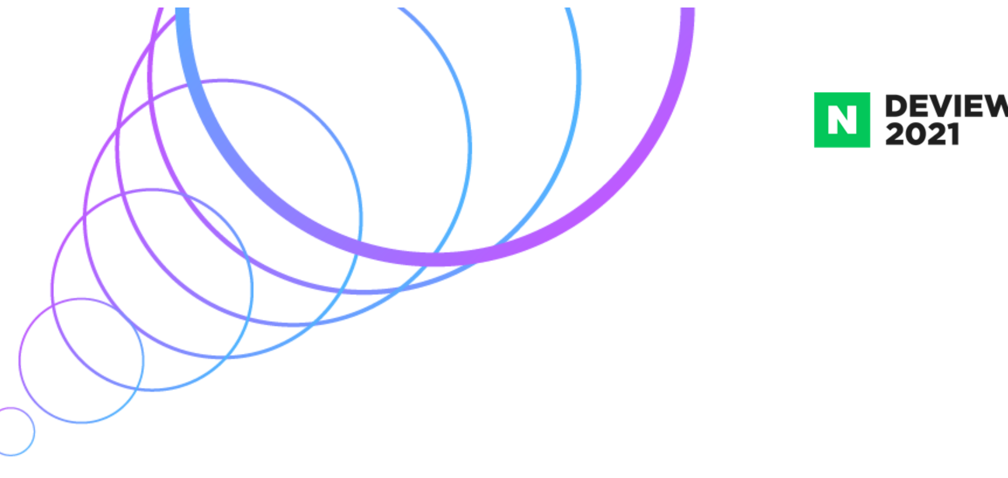


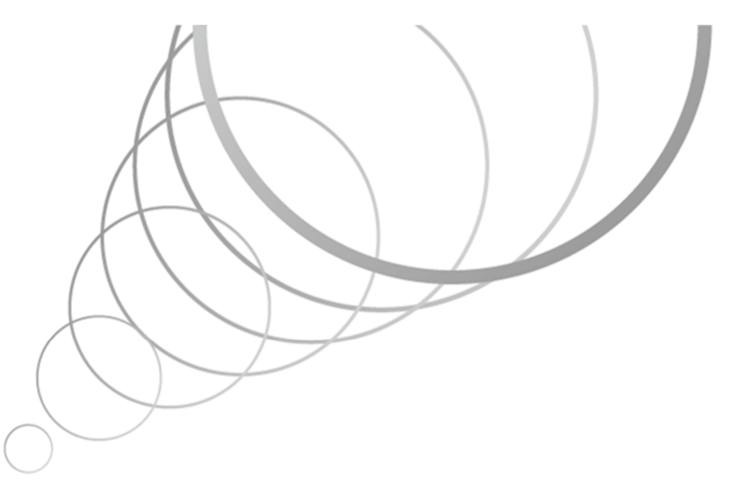
DEVIEW 2021

ARCUS 에서의 데이터 영속성 기술 구현

장수환 Jam2in

CONTENTS

1. ARCUS 소개
2. 데이터 영속성 기술의 개발 배경
3. 데이터 영속성 기술의 구현 방안
4. 데이터 영속성 기술의 구현 상세
5. 데이터 영속성 기술의 구현 성능
6. 마무리

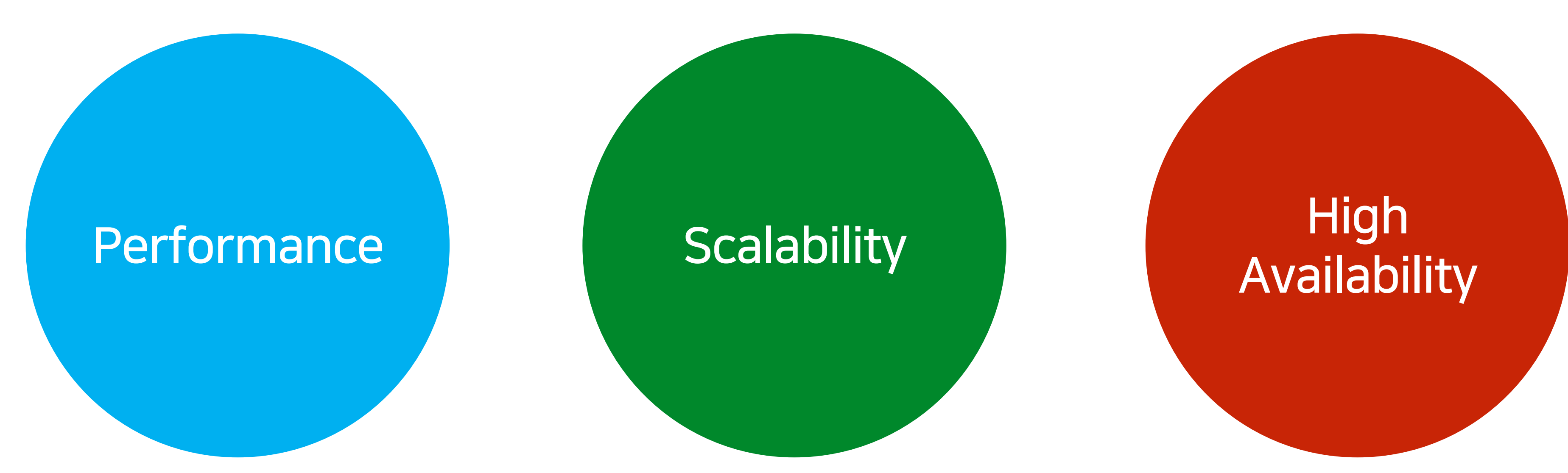


1.ARCUS 소개

1.1 ARCUS ?

Open Source, In-Memory Key-Value Data Store

Memcached 확장하고 Zookeeper 이용한 고성능 Elastic Cache Cluster



1.2 ARCUS 주요 기술 특징

- ✓ **메모리 캐시로 고성능 제공**

  100~200K requests/sec (1 cache node), 평균 1ms 이하 응답속도

- ✓ **확장된 Key-Value 데이터 모델**

  Collection(List / Set / Map / B+tree) 데이터 모델 지원

- ✓ **Elastic Cache Cluster 구현**

  Easy Scale-Out, Automatic Fail-Stop

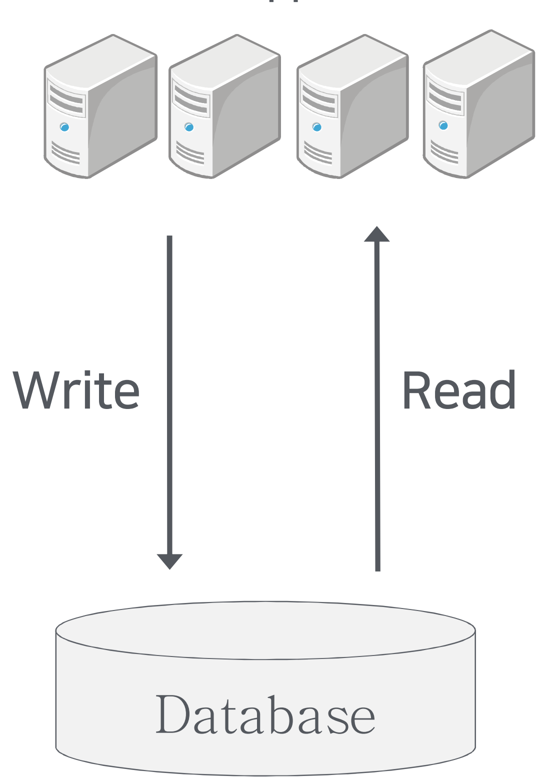
- ✓ **그 외의 주요 기능들**

  TTL(Time-To-Live), LRU 기반 eviction

  최적화된 메모리 관리, Semi-Sync 복제의 데이터 이중화

1.3 ARCUS Use Case

Remote Database Caching - Without Cache



Large Traffic

- 요청량 증가
- 데이터 증가

성능 이슈

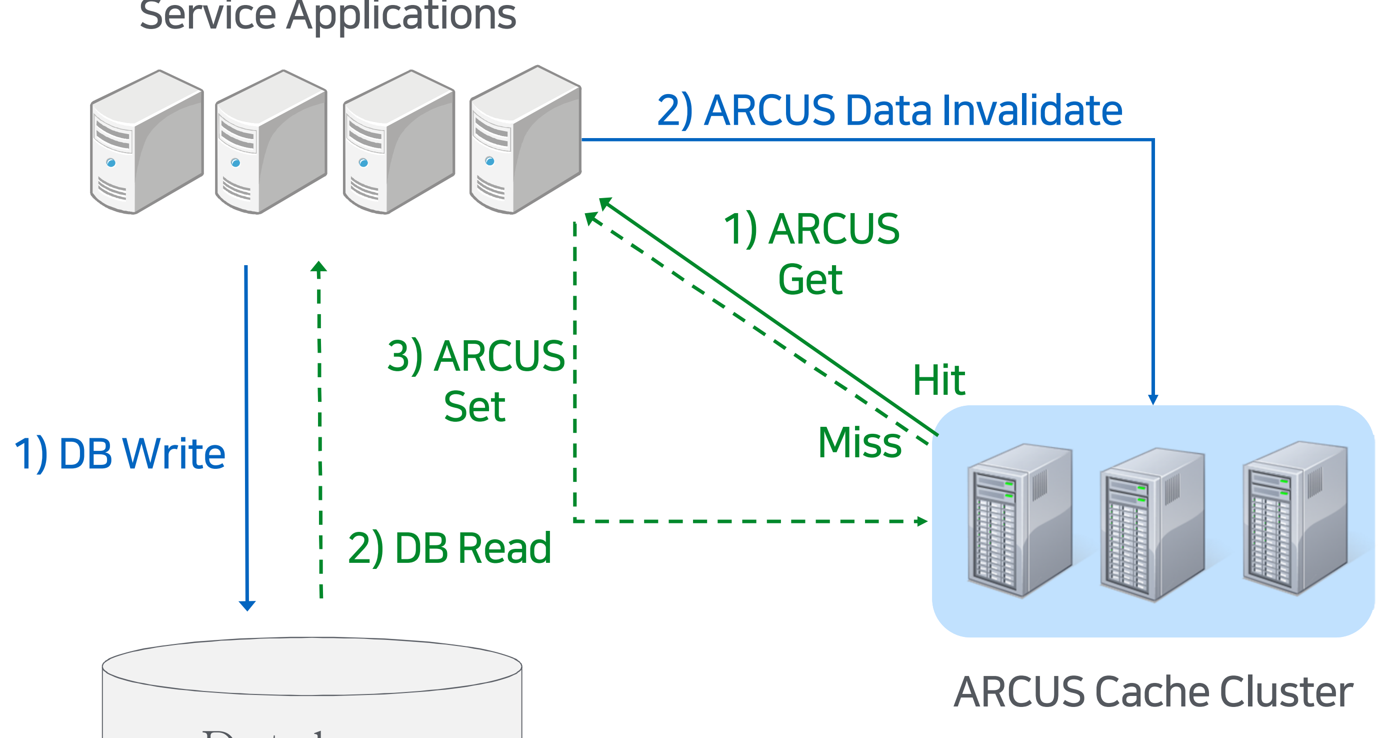
- 처리량 부족 / 느린 응답
- DB Scale-out 어려움
- 급변하는 요청 처리 미흡 (예, 이벤트)

고비용 이슈

- DB 라이선스, 장비

1.3 ARCUS Use Case

Remote Database Caching - With Cache (ARCUS)



Demand-Fill 방식

- DB 조회 결과
- 서비스 API 수행 결과 등

캐시 대상 데이터

- 변경보다 조회가 많은
- 생성(DB조회) 비용이 큰

2.데이터 영속성 기술의 개발 배경

2.1 데이터 저장 용도로의 확장

- ARCUS를 캐시 용도 외에 저장 용도로도 사용
  - ARCUS 캐시를 저장소 용도로 활용하는 기존 경우를 대체
  - ARCUS를 활용도가 높은 NoSQL로 성장시키기 위한 초기 단계

- 데이터 복제와 데이터 이관(Migration) 기능을 결합하면,
  완벽히 scale-in/out 가능한 Database Cluster로 거듭

데이터 보존이 가능하면서 대규모 트래픽을 처리할 수 있는
성능과 확장성을 갖춘 인메모리 데이터베이스

3.데이터 영속성 기술의 구현 방안

3.1 ARCUS 데이터 영속성

- 구현 방안:
  - 스냅샷(Snapshot)과 커맨드 로깅(Command Logging) 방식으로 캐시 데이터를 디스크에 기록하여 데이터 복구에 사용

- 구현 목표:
  - 재 구동 시 캐시 데이터를 종료 직전 상태로 완벽한 복구 보장
  - 데이터 영속성 처리 부담을 최소화하여 ARCUS 기존 고성능 유지

3.2 Logging 기법 비교

고성능 NoSQL에 Command Logging 방식이 적합 (Michael Stonebraker 제안)

	Physiological Logging	Command Logging
동작	변경 전과 후의 데이터 모두 기록	변경 대상 데이터만 기록
로깅 오버헤드	큼	작음
런타임 속도	느림	빠름
복구 속도	빠름 * 바뀐 데이터만 적용	느림 * 모든 커맨드를 재수행

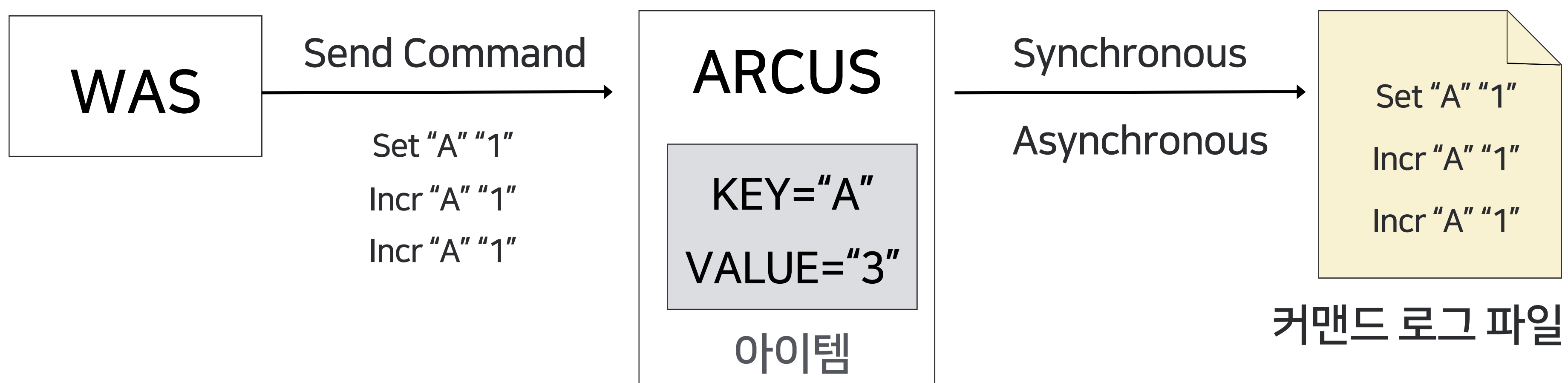
Paper: [Rethinking Main Memory OLTP Recovery]

3.3 Command Logging

응용이 보낸 데이터 변경 커맨드를 모두 디스크에

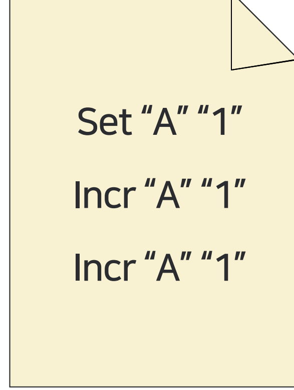
기록해두었다가 재구동 시 기록한 커맨드를 순서대로 재수행

3.3 Command Logging



3.3 Command Logging

복구



커맨드 로그 파일

Replay Command

ARCUS

복구 후 상태

ARCUS

KEY="A"

VALUE="3"

아이템

3.4 Checkpoint

응용의 요청에 비례하여 커맨드 로그 파일이 계속 커지면..

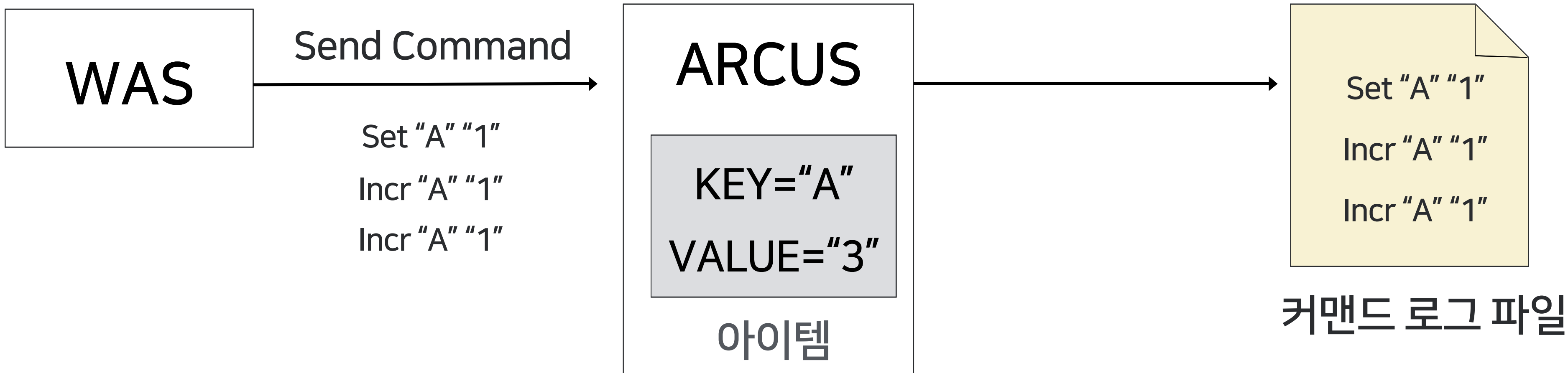
- Disk Resource 를 모두 사용
- 재구동 시에 재수행해야 할 커맨드가 많아 복구 시간이 오래 걸림

3.4 Checkpoint

- 커맨드 로그 파일이 운영자가 설정한 크기만큼 커지면, 자동으로 수행
- ARCUS에 저장된 전체 데이터를 스냅샷하여 백업 복사본을 만들고, 커맨드 로그 파일을 새로 기록

3.4 Checkpoint

체크포인트 전



3.4 Checkpoint

체크포인트 수행

WAS

Send Command

ARCUS

KEY="A"

VALUE="3"

아이템

스냅샷

KEY="A"

VALUE="3"

스냅샷 파일

Set "A" "1"

Incr "A" "1"

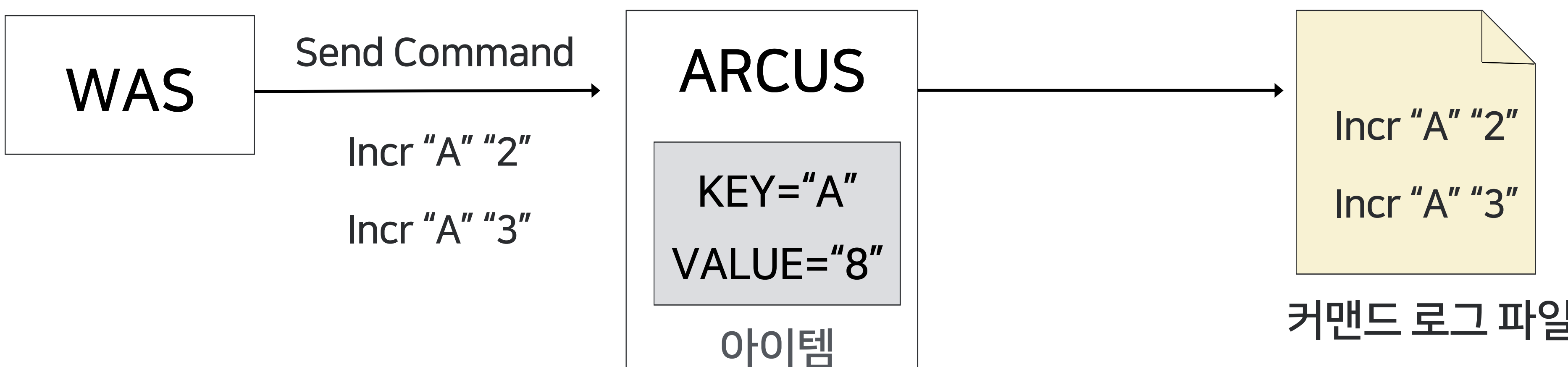
Incr "A" "1"

기존 커맨드 로그 파일

새로운 커맨드 로그 파일

3.4 Checkpoint

체크포인트 완료

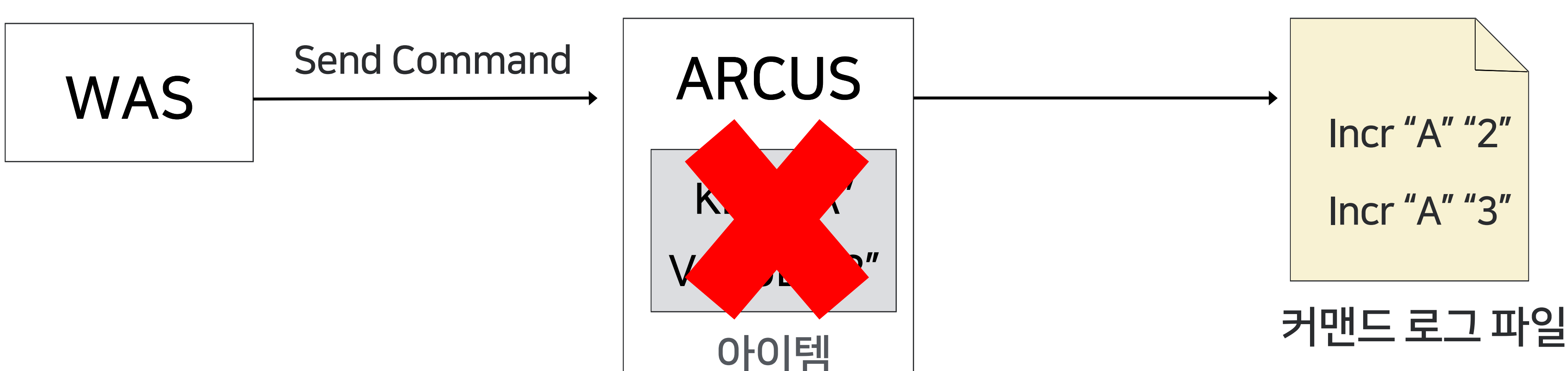


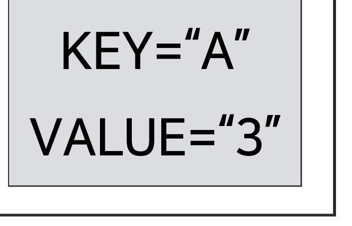
KEY=“A”
VALUE=“3”

스냅샷 파일

3.5 Recovery

다운 - 복구 데이터: (KEY=“A”, VALUE=“8”)

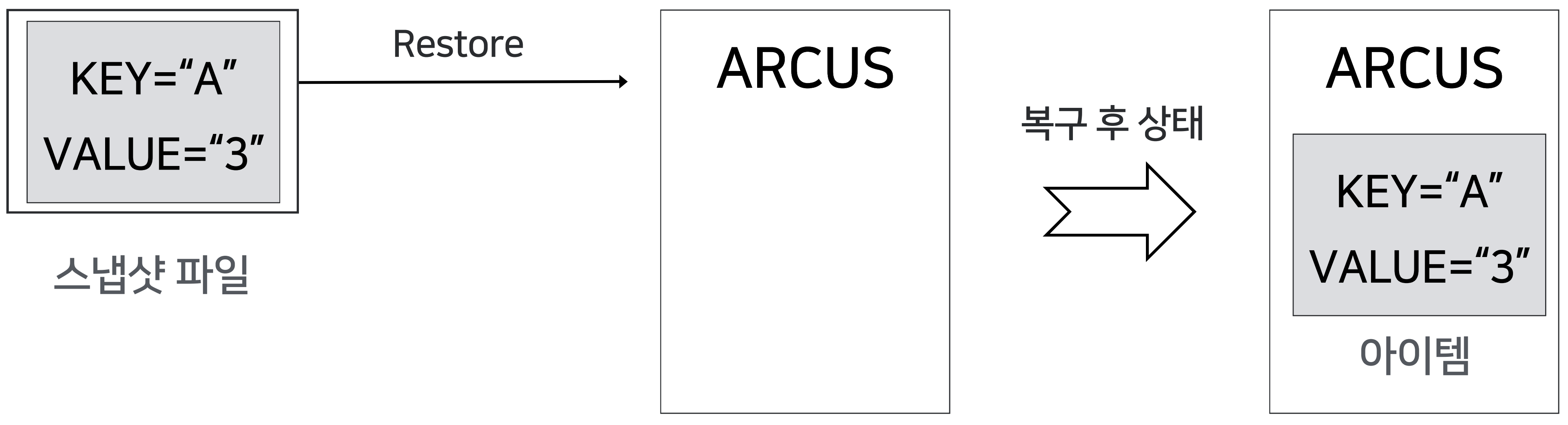




스냅샷 파일

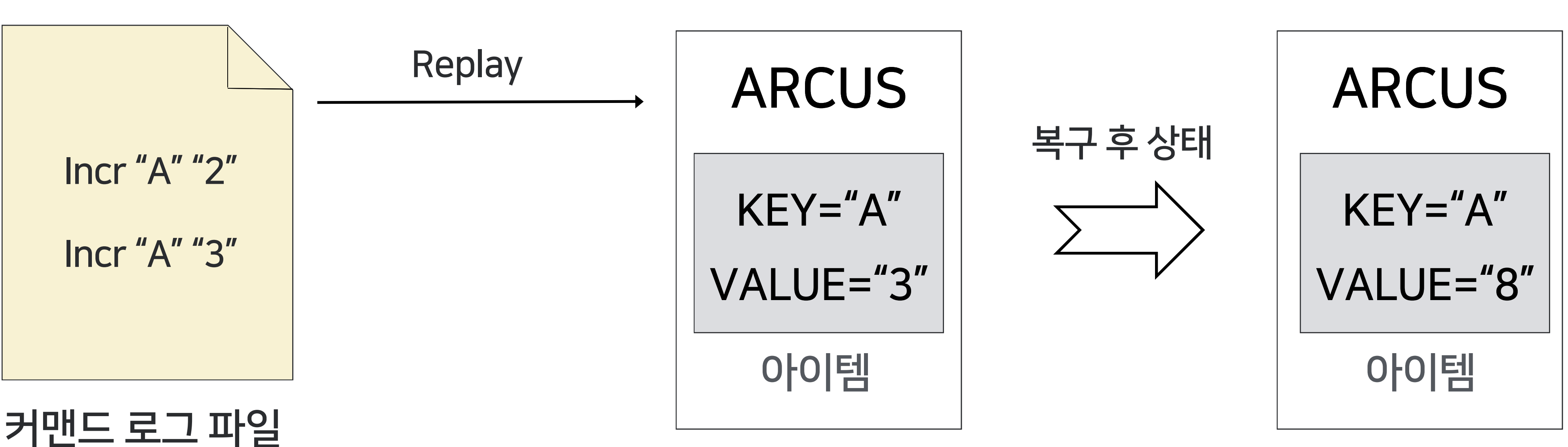
3.5 Recovery

먼저 스냅샷 파일을 이용하여 체크포인트 시점의 전체 데이터 복원



3.5 Recovery

커맨드 로그 파일에 기록된 데이터 변경 커맨드를 순차적으로 재수행



3.6 ARCUS 데이터 영속성 동작

커맨드 로깅과 스냅샷 방식의 체크포인트 방식으로 동작

- 커맨드 로깅으로 데이터 변경 커맨드를 기록해두고,
- 커맨드 로그 파일이 설정 크기를 넘어설 때마다 체크포인트 수행하여
  - 전체 데이터의 백업 복사본을 만들고,
  - 커맨드 로그 파일을 새로 생성하여 백업 복사본 이후의 변경을 기록
  - 이를 통해, 데이터 복구 시간을 최소화

4.데이터 영속성 기술의 구현 상세

4.1 데이터 영속성 기능 구현

DEVIEW 2021

ARCUS 기존 성능 영향을 최소화하면서 데이터 영속성 기능 구현

- 커맨드 로깅:

  - 모든 변경 커맨드를 디스크에 기록

- 체크포인트:

  - 체크포인트 수행 시점의 전체 캐시 데이터를 디스크에 기록

4.2 ARCUS 응용 요청 처리 구조

Core Module

Engine Module

WAS

Cache Lock

Execute Command

Hashtable

1
2
3
4
...
N

A
B
C
D

작업 스레드

Cache Lock 점유시간이 길어지면,
다른 작업 스레드의 요청 처리 지연

4.3 Command Logging

Core Module

Engine Module

Cache Lock

Execute Command

Generate Log Record

작업 스레드

Cache Lock 점유시간 길어짐

- Disk Stall: 응용 응답 지연
- Inefficient: 불필요하게 많은 fsync() 호출

write()

fsync()

Respond
To WAS

Release
Cache Lock

Log Disk

4.3 Command Logging

4.3 Command Logging

Core Module

Engine Module

Cache Lock

Execute Command

Generate Log Record

작업 스레드

Log Lock

memcpy()

write()

로그 플러쉬
스레드

작업스레드가 Log Lock 대기 시간이 길어지면
결국 Cache Lock 대기시간이 길어짐

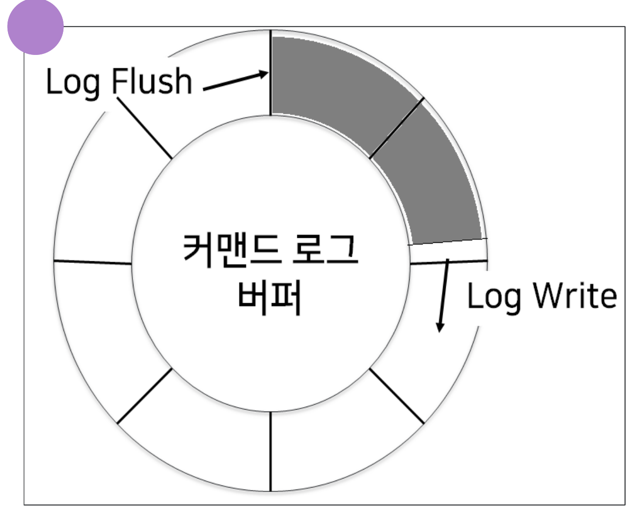
Respond
To WAS

Release
Cache Lock

Log Disk

4.3 Command Logging

- 로그 플러쉬 스레드는 아래 간단한 작업 시에만 Log Lock 점유하고,
  - Flush_Size 조회
  - Log Flush 위치 업데이트
- write() 는 Log Lock 점유하지 않고 수행
  - write(fd, &Log Flush, Flush_Size);



⇒ 작업 스레드가 Log Lock 대기시간은 아주 짧아
응용 요청 처리 지연 최소화

4.3 Command Logging

Core Module

Engine Module

Cache Lock

Execute Command

Generate Log Record

memcpy()

작업 스레드

로그 플러쉬
스레드

write()

Log Lock

Log Flush

커맨드 로그
버퍼

Log Write

Respond
To WAS

커맨드 로그를
디스크에 반영 후 응답해야 함

Log Disk

4.3 Command Logging

Core Module

Engine Module

Cache Lock

Execute Command

Generate Log Record

memcpy()

작업 스레드

write()

Log Lock

Log Flush

커맨드 로그 버퍼

Log Write

로그 플러쉬 스레드

Respond To WAS

로그 그룹 커밋 스레드

fsync() every short time

Log Disk

4.3 Command Logging

Core Module

Engine Module

Cache Lock

Execute Command

Generate Log Record

memcpy()

write()

Log Lock

Log Flush

커맨드 로그
버퍼

Log Write

작업 스레드

로그 플러쉬
스레드

Notify()

로그 그룹 커밋
스레드

fsync() every short time

Log Disk

Is my log record fsync() ?
Yes :
- Respond To WAS
No :
- Process Other Request

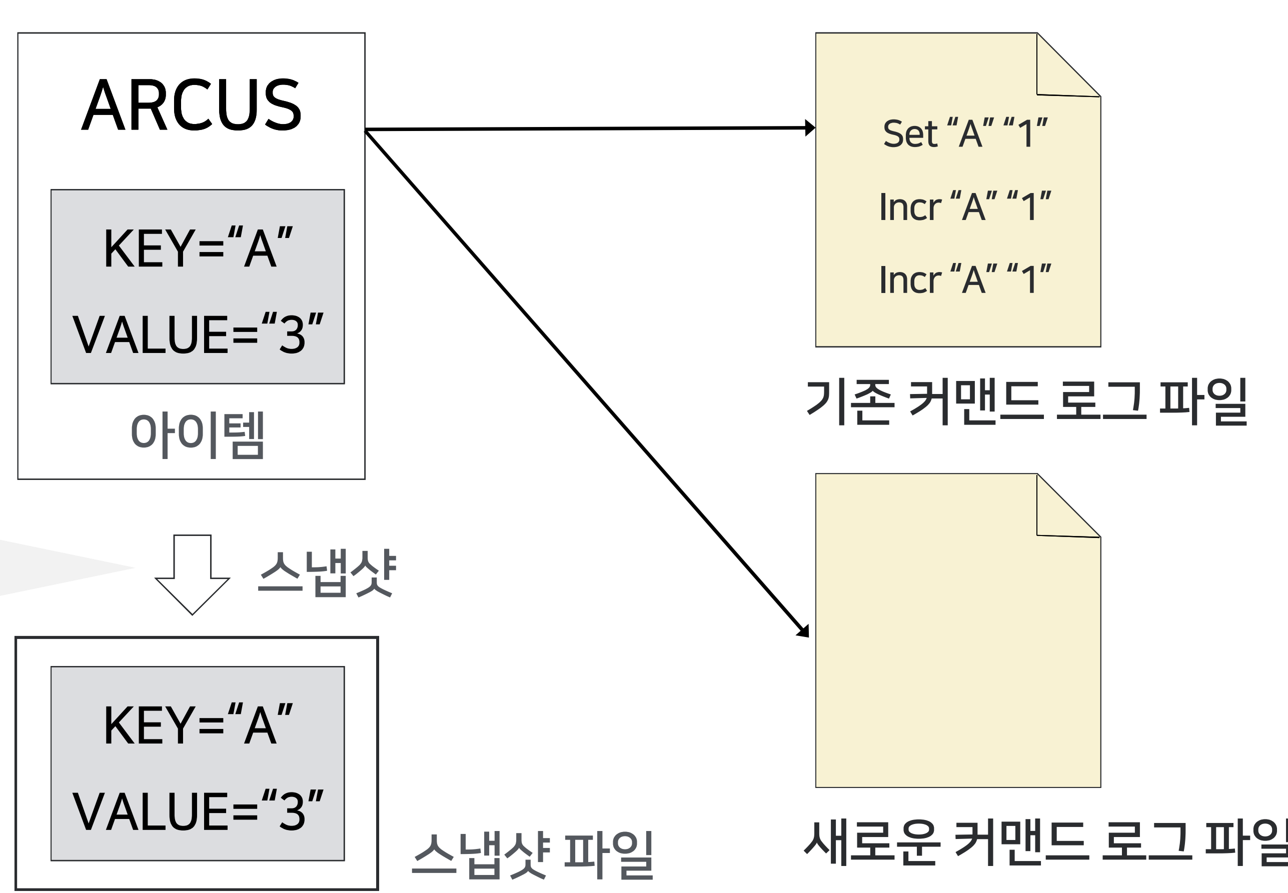
4.3 Command Logging

데이터 일관성 보장과 성능을 고려하여 두 가지 모드를 제공

동기 로깅	데이터 일관성 보장 시 변경 로그를 디스크에 반영 후 응용에 응답 이벤트 기반 처리로 성능 저하 최소화
비동기 로깅	성능 우선 시 변경 로그를 디스크에 반영을 보장하지 않고, 응용에 응답 캐시 노드가 비정상 종료되면, 일부 변경 로그를 잃을 수 있음

4.4 Snapshot

체크포인트 수행



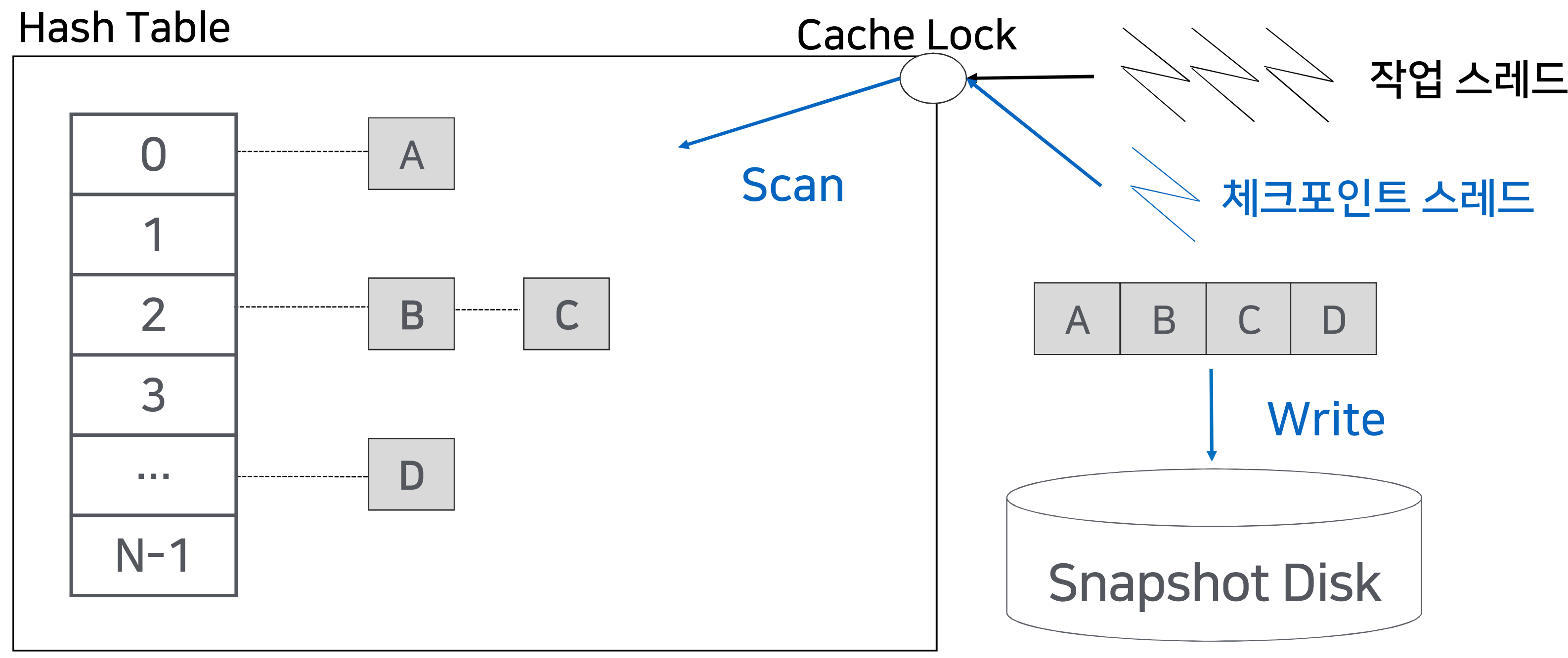
WAS

Send Command

전체 데이터를 기록하는
스냅샷 작업은 응용 요청 처리를
방해하지 않으면서 병렬 수행

4.4 Snapshot

- 해시 테이블을 스캔하여 전체 캐시 아이템을 스냅샷 디스크에 기록
- 별도의 체크포인트 스레드가 진행하여 응용 요청 처리와 병렬 수행



4.4 Snapshot

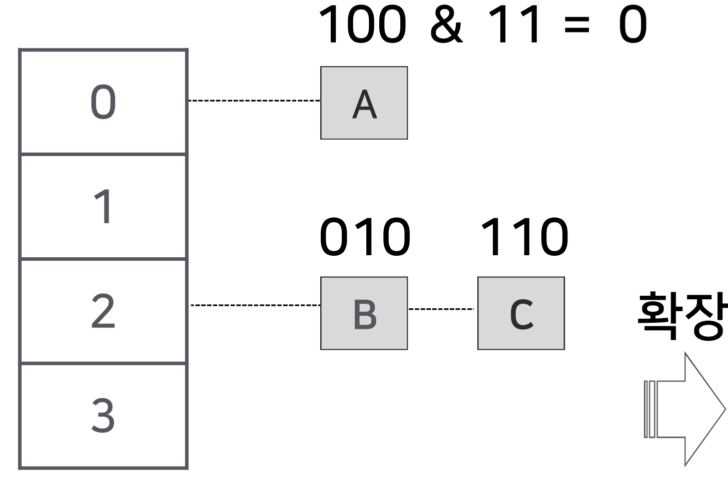
응용 요청 처리와 병렬로 수행하기 위한 방안:

=> 적은 수의 캐시 아이템을 스캔하는 과정을 반복

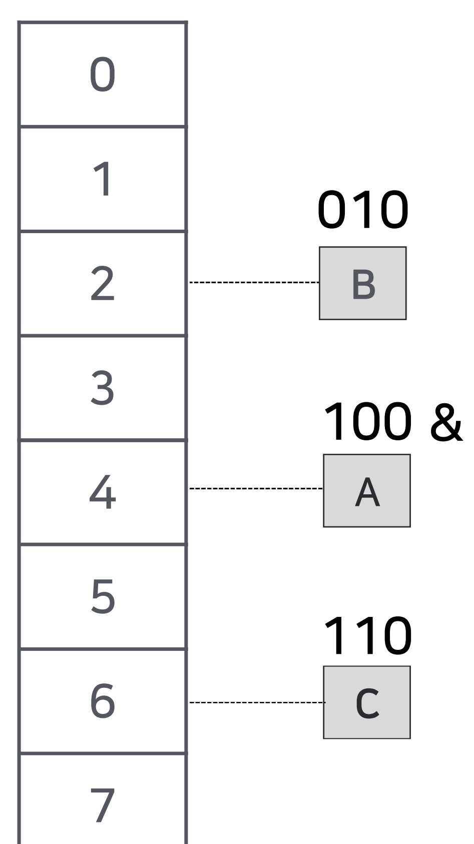
- 해시 테이블 확장 시 캐시 아이템 재분배로 인한 중복 스캔 문제
- 체크포인트 도중에 변경 요청의 커맨드 로그 기록

  - 기존 커맨드 로그 파일
  - 새로운 커맨드 로그 파일

4.5 아이템 재분배로 인한 중복 스캔 문제



확장



010

B

100 & 111 = 100

A

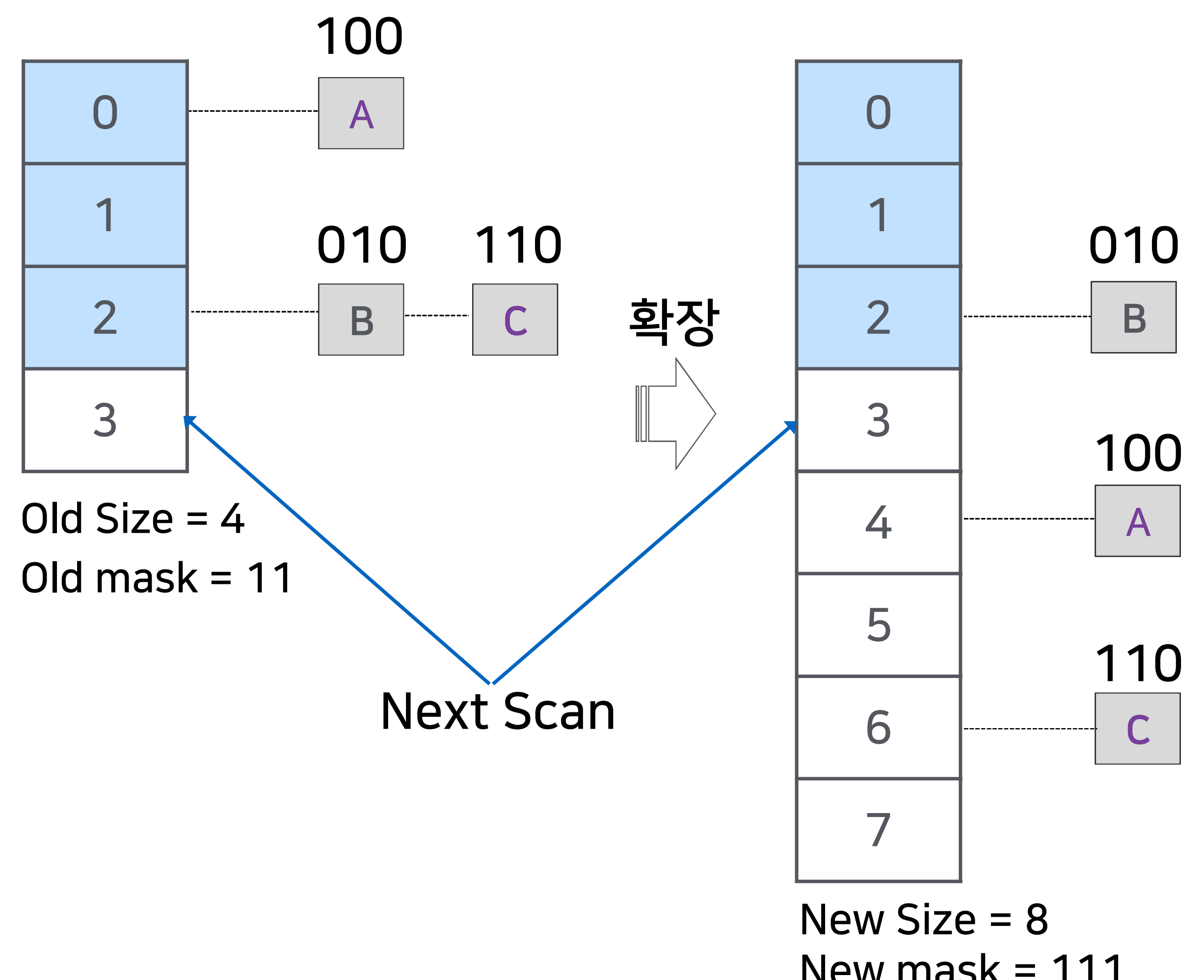
110

C

작업 스레드가 응용 요청 처리를 위해 버킷을 접근하면, 그 버킷의 캐시 아이템 재분배 작업 수행

재분배 시 캐시 아이템은 (버킷 인덱스 + Old Size) 번의 버킷으로 이동할 수 있음

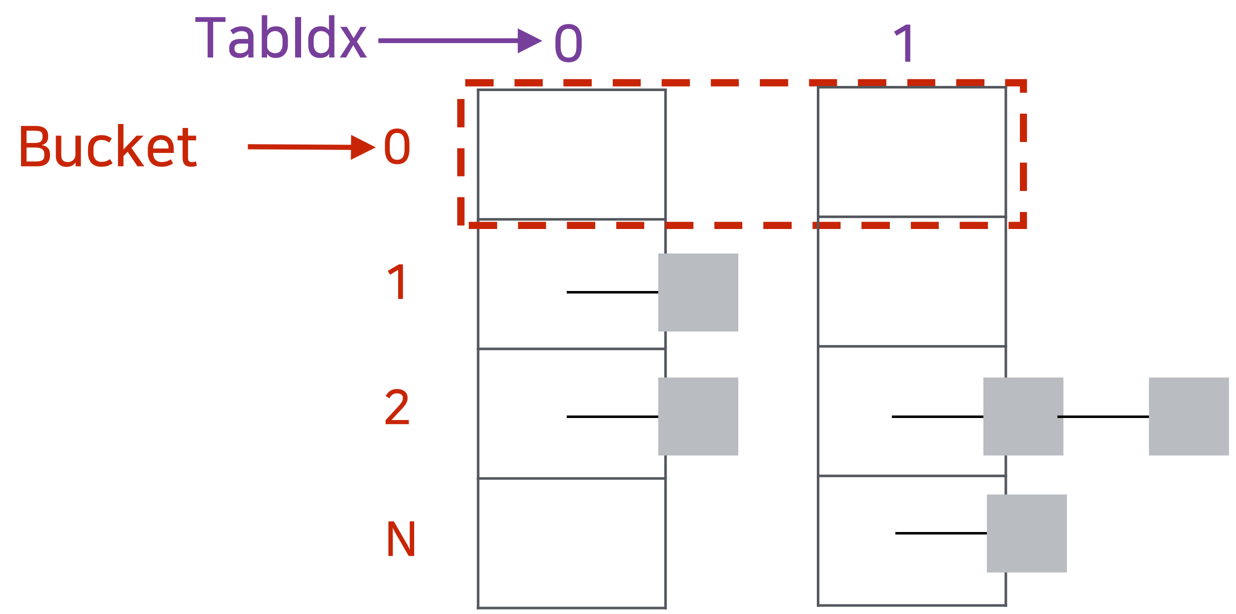
4.5 아이템 재분배로 인한 중복 스캔 문제



2번 버켓까지 스캔 완료하였다면,
재분배된 A와 C 아이템을
다음 스캔 시 중복 스캔

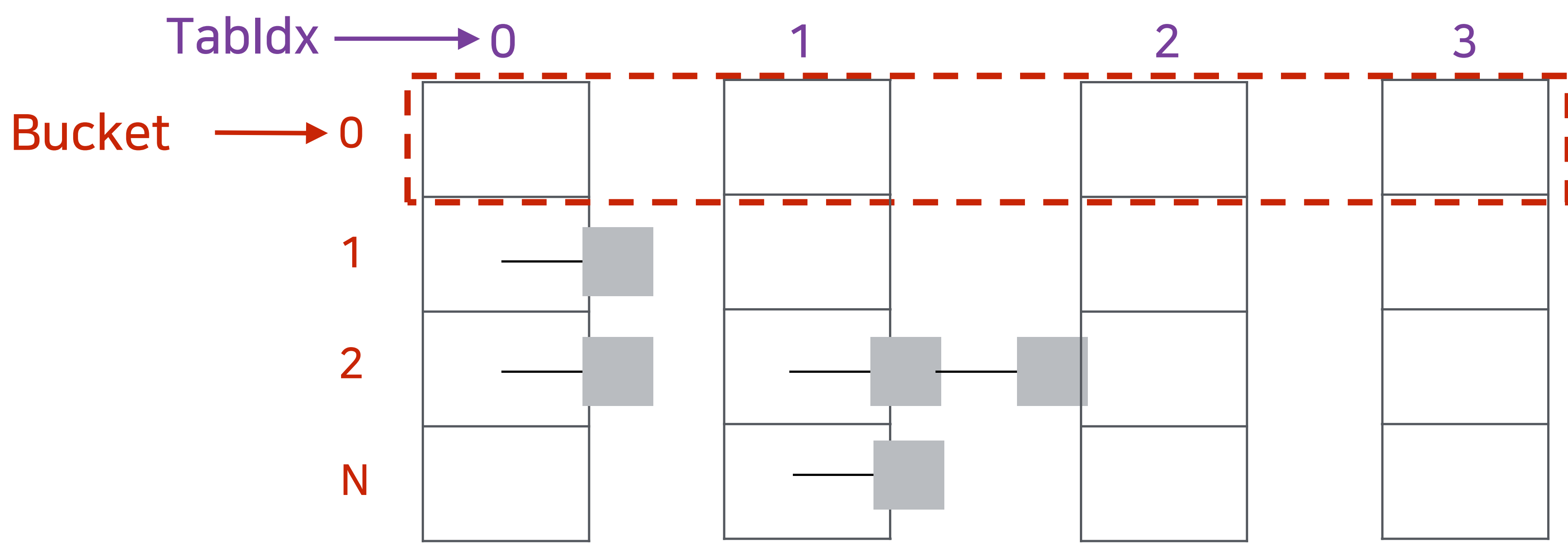
4.5 아이템 재분배로 인한 중복 스캔 문제

ARCUS Hashtable Implementation



4.5 아이템 재분배로 인한 중복 스캔 문제

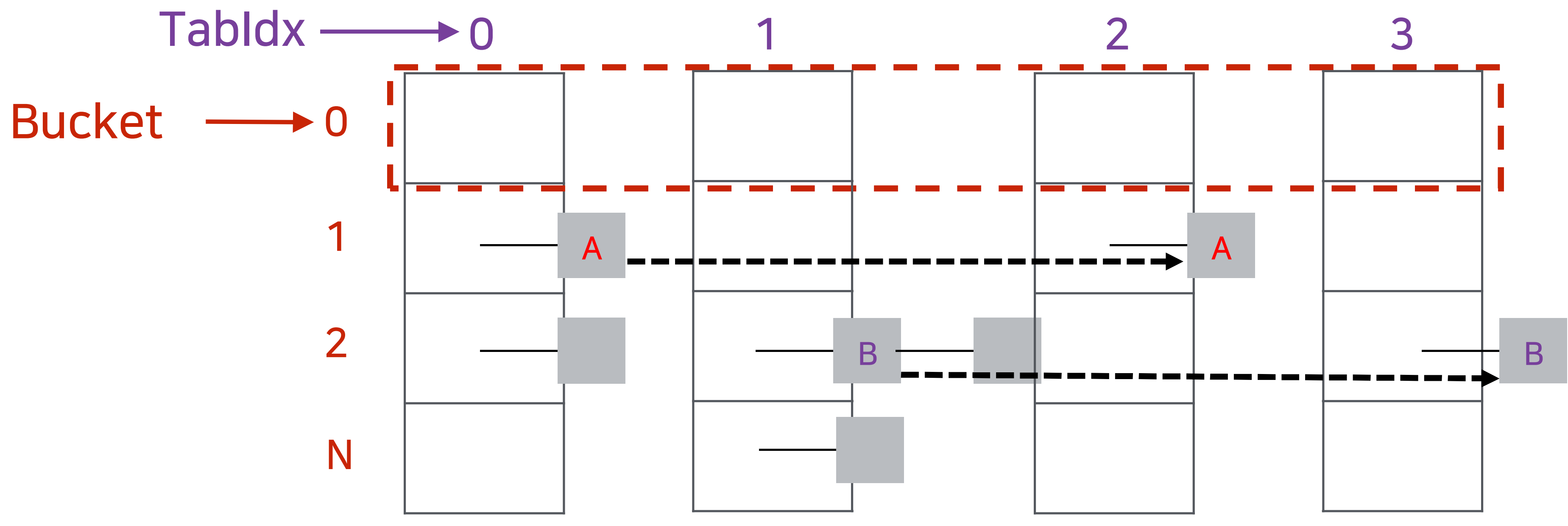
ARCUS Hashtable Implementation



해시테이블 확장 시 기존 크기와 동일한 해시 테이블을 2배씩 증가

4.5 아이템 재분배로 인한 중복 스캔 문제

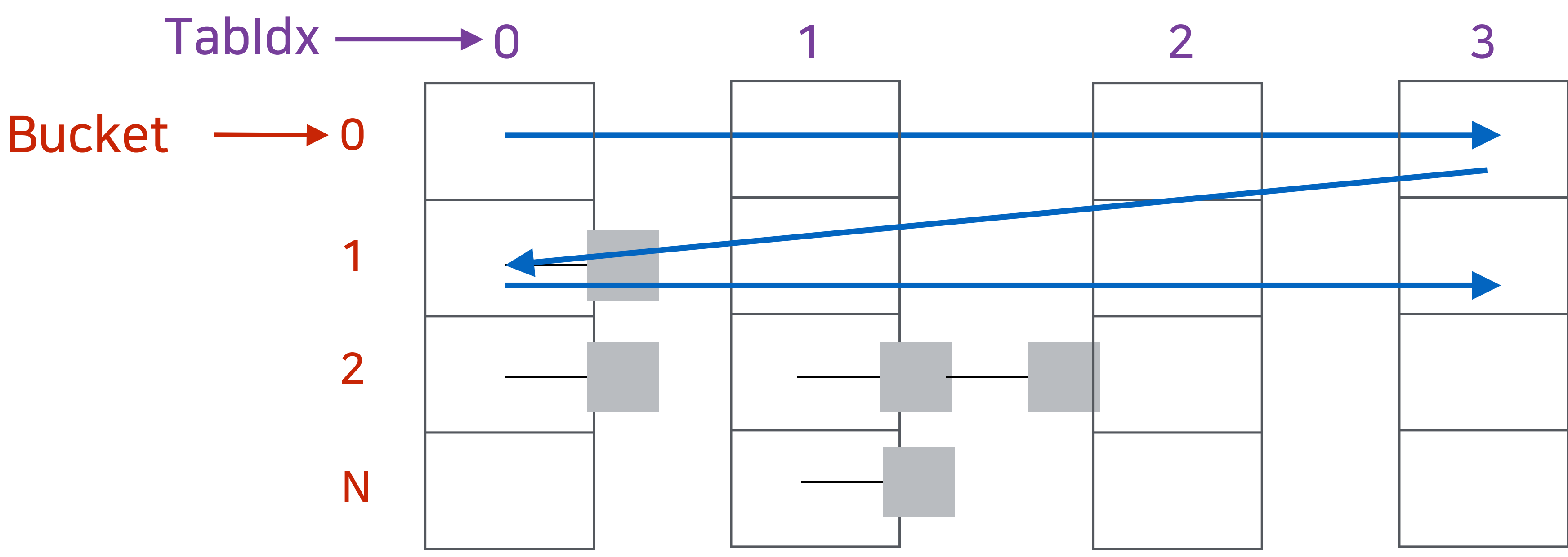
ARCUS Hashtable Implementation



캐시아이템 재분배 시 TabIdx 만 이동

4.5 아이템 재분배로 인한 중복 스캔 문제

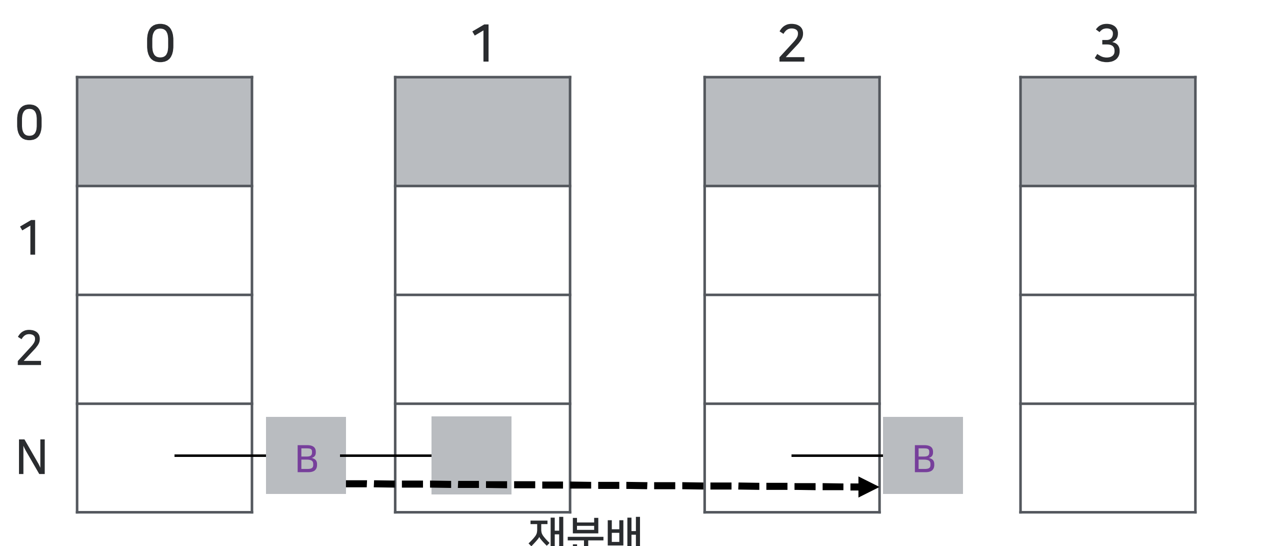
ARCUS Hashtable Implementation



Bucket 순서로 스캔 (TabIdx, Bucket)

(0, 0) -> (1, 0) -> (2, 0) -> (3, 0) -> … -> (3, N)

4.6 Lazy Redistribute



- 버켓 참조카운트를 증가시켜두어 해당 버켓의 재분배를 일시적으로 방지
- 버켓의 모든 해시 테이블(TabIdx 0 ~3)을 스캔 완료하면 참조카운트 감소

4.6 Lazy Redistribute

- 버켓 참조카운트를 증가시켜두어 해당 버켓의 재분배를 일시적으로 방지
- 버켓의 모든 해시 테이블(TabIdx 0 ~3)을 스캔 완료하면 참조카운트 감소

4.7 체크포인트 수행 중 커맨드 로깅

체크포인트 수행 중 요청이 들어오면 어느 파일에 기록?

4.7 체크포인트 수행 중 커맨드 로깅

기존 커맨드 로그 파일에는 모든 변경 요청을 기록 – 종료 또는 비정상 종료 대비

응용 요청 전송

ARCUS

KEY="A"

VALUE="3"

아이템

모든 변경 요청

Set "A" "1"

Incr "A" "1"

Incr "A" "1"

기존 커맨드 로그 파일

종료 후 재구동 시
기존 스냅샷 파일과
기존 커맨드 로그 파일을
이용하여 복구

스냅샷

KEY="A"

VALUE="3"

새로운 스냅샷 파일

새로운 커맨드 로그 파일

체크포인트 정상 완료해야
복구에 사용 가능

4.7 체크포인트 수행 중 커맨드 로깅

새로운 커맨드 로그 파일에는 스냅샷된 아이템에 대한 변경 요청만 기록 – 중복 기록 방지

응용 요청 전송

ARCUS

KEY="A"
VALUE="3"

아이템

스냅샷

KEY="A"
VALUE="3"

새로운 스냅샷 파일

스냅샷된
아이템 변경 요청

Set "A" "1"
Incr "A" "1"
Incr "A" "1"

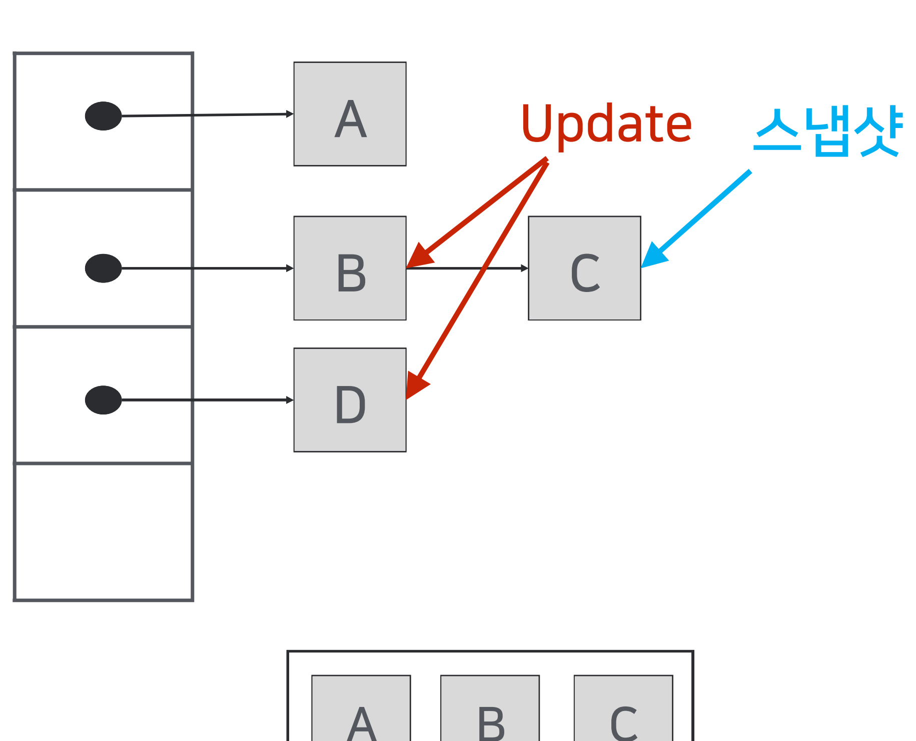
기존 커맨드 로그 파일

새로운 커맨드 로그 파일

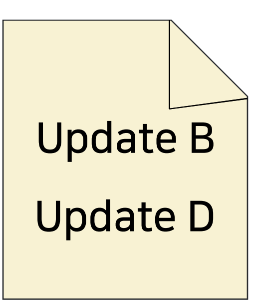
해시테이블 상에서
스냅샷 진행 중인 위치와
요청 타겟 아이템의 위치 비교

4.7 체크포인트 수행 중 커맨드 로깅

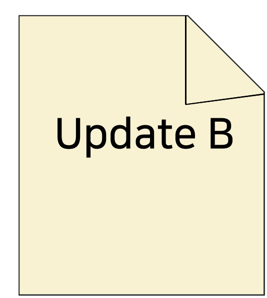
A, B, C: 스냅샷 완료
D: 스냅샷 예정



새로운 스냅샷 파일



기존 커맨드 로그 파일



새로운 커맨드 로그 파일

4.7 체크포인트 수행 중 커맨드 로깅

D 변경 내역은 스냅샷 파일에 기록될 예정이므로
새로운 커맨드 로그 파일에 중복 기록하지 않아야 함

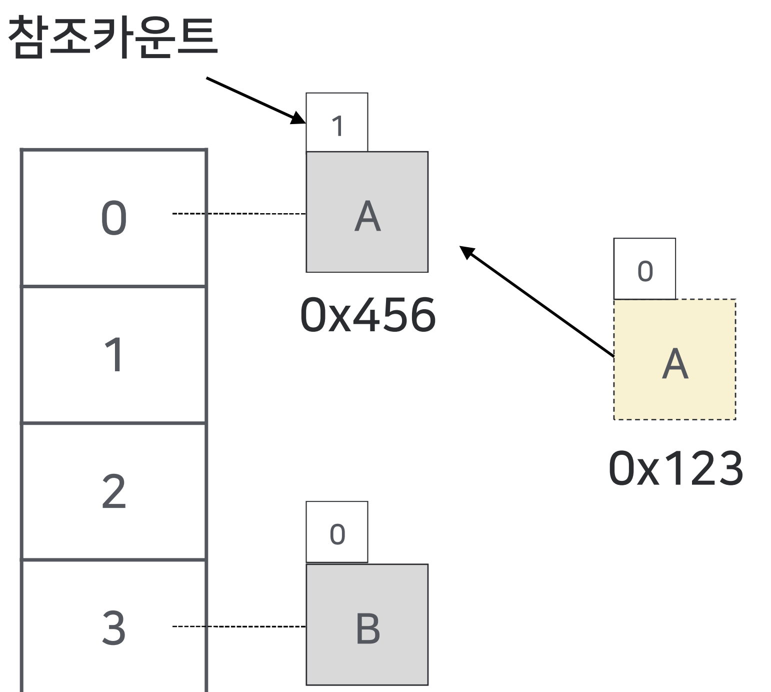
4.8 Scan Optimization

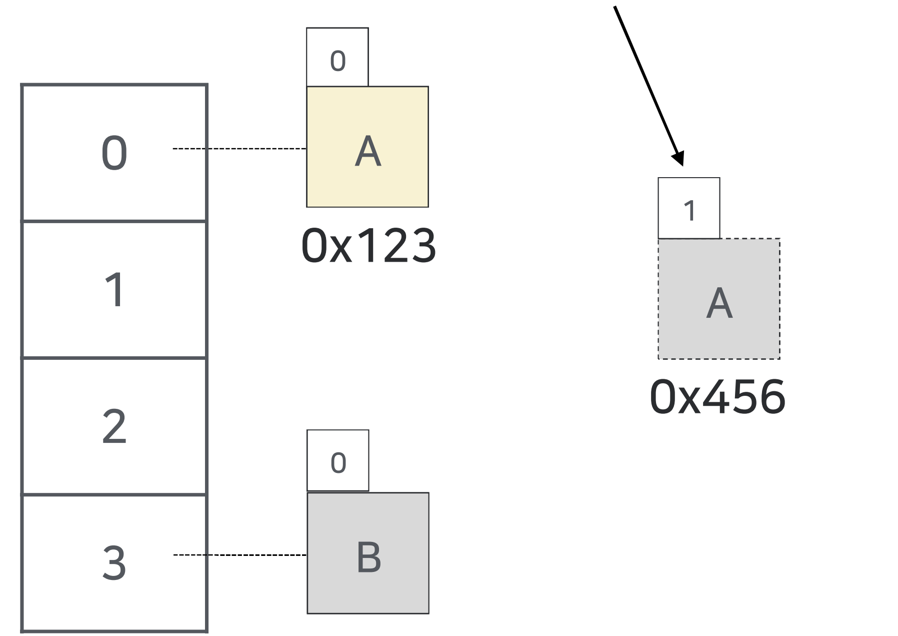
- Scan 작업을 가능한 빠르게 수행하는 방안:
  - 스캔 시 주소 값(포인터)만 저장해두었다가 (not memcpy())
    디스크에 기록 시에 포인터로 실제 데이터를 조회

- 디스크에 기록하기 전에 응용이 해당 아이템을 변경한다면?
  - 스캔 시점의 데이터를 잃어버리지 않도록
    아이템 참조카운트와 교체 변경으로 스캔 시점의 데이터 유지

4.9 스캔 시점의 데이터 보존

스캔한 아이템(A)은 참조카운트를 증가시켜두고,
응용 요청에 의해 변경된다면 교체(Out-Of-Place) 방식으로 변경





5.데이터 영속성 기술의 구현 성능

5.1 데이터 영속성 성능 시험 환경

	Server (Single Instance)	Client
OS	CentOS 7.3	
CPU	8vCPU	
Memory	8GB * 2	
Network	1Gbps	
Disk	HDD 50GB * 2	SSD 50GB

5.1 데이터 영속성 성능 시험 환경

벤치마크 도구 (Memtier_benchmark)

- Developed by Redis Labs
- ARCUS(Memcached) 명령 프로토콜 삽입(Set), 조회(Get) 지원
- 랜덤, 가우시안 분포 키 생성 기능 지원

모니터링 도구 (ARCUS Hubble)

- 장비 시스템 리소스 및 ARCUS 서버 통계 정보 수집
- 웹 브라우저를 통한 모니터링

5.2 커맨드 로깅 성능

삽입 시험 결과

* OFF : 기존 캐시 성능, ASYNC : 비동기 로깅 모드, SYNC : 동기 로깅 모드

평균처리량(ops/sec)

130000
110000
90000
70000
50000
30000
10000

120K
105K
82K

OFF
ASYNC
SYNC

응답시간(msec)

18
15
12
9
6
3
0

AVG
90%
95%
99%

OFF
ASYNC
SYNC

- 모든 요청이 삽입 연산으로 구성되어 있어 커맨드 로깅이 가장 많이 수행되는 시험
- SYNC 는 ARCUS 를 비정상 종료하더라도 데이터를 완벽히 복구가 가능하면서도 높은 성능을 보임

5.2 커맨드 로깅 성능

혼합 시험(1:9) 결과

* OFF : 기존 캐시 성능, ASYNC : 비동기 로깅 모드, SYNC : 동기 로깅 모드

평균처리량(ops/sec)

157K 155K 141K

160000 140000 120000 100000 80000 60000 40000

OFF ASYNC SYNC

응답시간(msec)

18 15 12 9 6 3 0

AVG 90% 95% 99%

OFF ASYNC SYNC

- 변경과 조회 비율이 1:9인 혼합 연산
- 커맨드 로깅이 적게 수행되어 조회 연산 처리 성능(170K / 3ms) 에 가까움

5.2 커맨드 로깅 성능

혼합 시험(3:7) 결과

* OFF : 기존 캐시 성능, ASYNC : 비동기 로깅 모드, SYNC : 동기 로깅 모드

평균처리량(ops/sec)

160000
140000
120000
100000
80000
60000
40000

148K
145K
118K

OFF
ASYNC
SYNC

응답시간(msec)

18
15
12
9
6
3
0

AVG
90%
95%
99%

OFF
ASYNC
SYNC

- 변경과 조회 비율이 3:7인 혼합 연산

5.3 체크포인트 영향

- 5천만건 데이터 삽입 후에 변경과 조회 비율 1:9인 혼합 연산 시험을 비동기 로깅 모드로 동작
- 평균 처리량은 변경 처리량(노랑)과 조회 처리량(초록)을 합한 150K
- 체크포인트 수행하는 2초 동안 100K 정도로 처리량이 감소하고, 총 5GB (5천만건) 데이터가 스냅샷됨

5.4 데이터 영속성 성능 시험 결과

- ARCUS 영속성 기능은 일반 요청 처리에 미치는 영향을 최소화하도록 설계하여 기존 캐시와 차이가 크지 않은 성능
- 대부분의 실서비스 워크로드 패턴은 조회 연산 비중이 많은 삽입과 조회를 혼합한 형태이며, 혼합 시험 결과를 보면 상당히 높은 성능
- 본 시험 환경에 사용한 디스크는 Naver Cloud Platform 사의 Standard VM장비의 기본 HDD인데, 높은 IOPS를 제공하는 NVMe SSD와 같은 디스크를 사용하면 본 시험보다 더 높은 성능 예상

6.마무리

6.1 마무리

- 커맨드 로깅과 체크포인트 방식으로 ARCUS 데이터 영속성 구현
  - 커맨드 로깅으로 데이터 변경 요청을 기록해두고,
  - 체크포인트로 커맨드 로그 파일 적정 크기 유지
- 향후 작업
  - 최적화를 통한 처리량과 응답 시간 향상
  - 운영 관점의 사용 편의성 향상
  - 데이터 복제와 이관 기능을 결합하여 클러스터에서의 사용 확장

Thank You
E-Mail : suhwan9185@jam2in.com