# 클라우드 데이터베이스에서의 꼬리응답시간 감소를 위한 가비지 컬렉션 동기화 기법

한승욱[0] 김지홍
서울대학교 컴퓨터공학부
{swhan, jihong}@davinci.snu.ac.kr

# A Garbage Collection Synchronization Technique for Improving Tail Latency of Cloud Databases

Seungwook Han[0] Jihong Kim
Seoul National University

**요 약**

클라우드 데이터베이스와 같이 분산 시스템 환경에서는 균일한 서비스 품질을 보장하기 위해 꼬리시간을 짧게 유지하는 것이 중요하다. 본 논문에서는 카산드라 데이터베이스를 대상으로 가상머신에서의 가비지 컬렉션과 SSD에서의 가비지 컬렉션이 연이어 발생하는 경우 꼬리 응답시간이 크게 증가함을 보이고, 두 종류의 가비지 컬렉션을 동기화하여 꼬리응답시간을 개선하는 기법을 제안한다. YCSB 벤치마크 결과, 제안한 기법이 꼬리응답시간인 $99^{th}$와 $99.9^{th}$-percentile을 각각 최대 32%, 75% 줄이고 있다.

## 1. 서 론

응답시간(Latency)은 소프트웨어의 서비스 품질(Quality of Service) 측면에서 사용자 경험을 결정하는 중요한 요소이다. 대용량 데이터를 다루는 클라우드 데이터베이스들은 여러 노드에 데이터를 분산시켜 응답시간을 줄이고자 하였다. 특히 페이스북(Facebook)에서 만든 카산드라(Cassandra)는 하루에 발생하는 수십억 개의 데이터 쓰기 요청을 처리하기위해 최적화된 분산 데이터 베이스 시스템이다 [1]. 하지만 카산드라 같은 분산 시스템 환경도 응답시간을 균일하게 유지하는 것은 어렵다. 분산 시스템의 응답시간 연구에 따르면, $99.9^{th}$-percentile의 응답시간은 중간응답시간(Median Latency)의 수십 배로 분석된다 [2]. 이러한 꼬리응답시간(Tail Latency)은 사용자경험(User Experience)에 매우 나쁜 영향을 주기 때문에 그 응답시간의 길이가 서비스 품질을 결정하게 된다. 따라서 꼬리응답시간을 줄이기 위한 많은 연구들이 선행되어 왔다 [2, 3, 4].

꼬리응답시간에 영향을 주는 요소는 크게 네트워크, 메모리 그리고 저장장치이다. 카산드라의 꼬리응답시간을 낮추기 위한 네트워크 수준의 연구 [2]는 선행되었고, 이를 제외한 단일 노드 환경에서의 요인을 분석하고자 한다.

메모리의 경우, 자바가상기계(Java Virtual Machine; JVM)가 메모리 관리를 위해 발생시키는 가비지 컬렉션(Garbage Collection; JVM-GC) 때문에 카산드라는 시스템이 멈추는 현상을 겪는다. 자바가상기계를 사용하는 다른 분산 시스템들 또한 실행시간의 대부분이 멈춰있는 문제를 겪고 있다 [5]. 카산드라는 멈춰있는 동안 사용자 요청을 처리할 수 없기 때문에 JVM-GC가 발생할 때는 응답시간이 길어지게 된다.

쓰기 요청에 대한 응답시간에는 데이터가 저장장치에 쓰이는 시간이 포함된다. 반면 카산드라는 요청된 데이터가 메모리에만 쓰여도 응답을 보내어 응답시간을 줄이는 정책을 지원하고 있다. 이후 메모리에 저장된 데이터들은 메모리 테이블(MemTable) 단위로 모여서 저장장치에 내려쓰게(Flush) 된다. 하지만 쓰기 요청이 집중적으로 발생하여 데이터를 저장할 메모리 공간이 부족해지면 카산드라는 사용자의 쓰기 요청을 더 이상 받지 않는다. 이러한 쓰기 요청 무시는 하나의 메모리 테이블이 저장장치에 내려 쓰여 메모리에 빈 공간이 확보될 때까지 지속된다. 이처럼 쓰기 요청이 집중적으로 발생하는 경우 저장장치의 쓰기 속도 또한 꼬리응답시간에 영향을 주게 된다.

한편 서버 시장에서는 저장장치가 하드 디스크 드라이브에서 플래시 메모리로 전환되는 추세이다. 하드 디스크 드라이브와 달리 플래시 메모리는 페이지 단위로 데이터를 쓰고 블록 단위로 삭제를 한다. 또한 물리적으로 덮어쓰기를 지원하지 않기 때문에 덮어쓰기는 데이터를 새 페이지를 쓴 뒤, 원래 페이지를 무효화하는 방식으로 처리된다. 무효페이지가 쌓여 더 이상 새 페이지가 없는 상황이 되면, 새 페이지를 얻기 위해 무효페이지를 수거하는 가비지 컬렉션(Garbage Collection; SSD-GC)이 발생한다. SSD-GC가 진행되는 동안에는 많은 페이지 복사와 블록 삭제로 인해 플래시 메모리의 성능이 떨어진다. 이처럼 SSD-GC가 진행되는 동안에 메모리 테이블을 내려쓰면 쓰기가 천천히 진행되어 완료시간이 길어진다. 따라서 메모리 테이블 내려쓰기 도중에 끼어든 SSD-GC가 카산드라의 꼬리응답시간에 영향을 주게 된다.

플래시 메모리의 성능 저하를 막기위해 시스템 문맥을 파악하고 입출력 요청이 없는 상황에서 SSD-GC를 발생시키는 기법이 제안되었다 [6]. 본 연구는 이러한 기법과 맥을 같이한다. 카산드라의 꼬리응답시간은 JVM-GC와 SSD-GC가 연이어 발생했

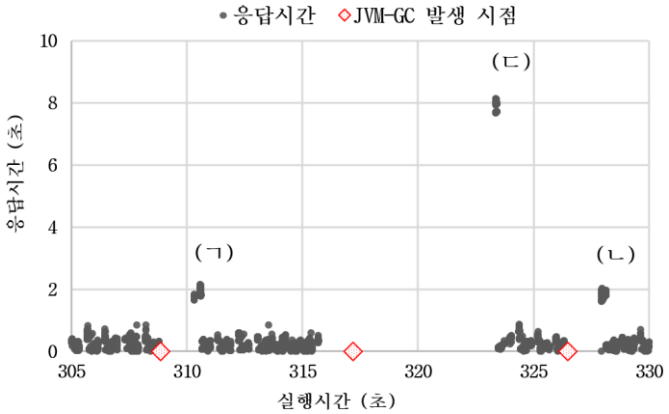



그림 1 쓰기요청과 JVM-GC 타임라인

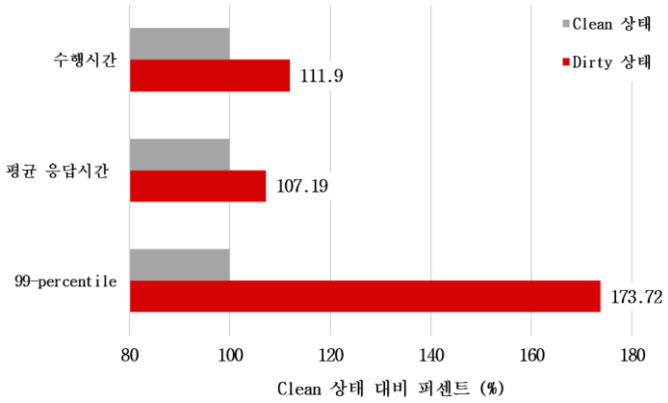



그림 2 SSD-GC가 꼬리응답시간에 미치는 영향

을 때 만들어진다. 우리는 JVM-GC와 SSD-GC를 동기화하여 카산드라가 JVM-GC로 멈춰있는 시간 동안에 SSD-GC가 미리 수행되도록 한다. 미리 수행한 SSD-GC로 메모리 테이블 내려쓰기가 일찍 완료되고 이를 통해 꼬리응답시간을 줄이는 것이 본 연구의 목표이다. 연구 기여는 카산드라의 꼬리응답시간이 JVM-GC와 SSD-GC가 연이어 발생했을 때 만들어진다는 것을 분석한 것과 가비지 컬렉션 동기화 기법을 제안하고 이를 통해 꼬리응답시간인 $99^{th}$, $99.9^{th}$ 그리고 $99.99^{th}$-percnetile을 각각 최대 32%, 75% 그리고 78% 줄인 것이다.

## 2. 꼬리응답시간 분석

카산드라의 꼬리응답시간은 JVM-GC 직후에 메모리 테이블이 저장장치로 내려쓰기 될 때 만들어진다. 그림 1은 YCSB 벤치마크를 통해 측정된 쓰기 요청들의 응답시간과 카산드라에서의 JVM-GC 발생 시점을 타임라인으로 표현한 것이다. 그림 1에 표현된 3번의 JVM-GC는 모두 1.5초 동안 지속된다. 때문에 JVM-GC가 발생한 다음에는 쓰기 요청의 응답시간이 지연되는 모습을 확인할 수 있다. 세개의 JVM-GC 모두 비슷한 수행시간을 가지지만 (ㄷ)에 해당하는 JVM-GC 이후에는 특별히 긴 시간인 8초 지연되는 모습을 보인다. 이것은 JVM-GC 이후 메모리 테이블의 내려쓰기가 완료될 때까지 카산드라가 사용자의 쓰기 요청을 무시하여 발생한다. 이처럼 JVM-GC 이후에 응답시간이 특별히 더 지연되는 경우가 관찰되며 그것들 중 일부가 꼬리응답시간이 된다. 반면 (ㄱ)과 (ㄴ)의 JVM-GC 이후에는 모두 약 2초 동안 지연되고 있다. JVM-GC가 1.5초 동안 진행한 점을 고려하면, 이 경우는 메모리 테이블 내려쓰기가 연달아 발생하지 않은 것으로 해석된다.

SSD-GC가 꼬리응답시간에 영향을 주는지를 분석하기 위해 실험을 진행하였다. 실험은 플래시 메모리의 상태를 Clean과 Dirty로 나누어 SSD-GC가 적게 발생하는 경우와 많이 발생하는 경우를 비교하고자 한다. Clean 상태는 무효페이지가 없는 초기 상태의 플래시 메모리를 뜻하고, Dirty 상태는 플래시 메모리의 최대 용량만큼 썼다가 지워 무효페이지가 많은 상태이다. 그림 2는 수행시간과 평균 응답시간 그리고 $99^{th}$-percentile을 Clean 상태와 비교하여, SSD-GC가 각 요소에 끼치는 영향을 나타내고 있다. 실험은 YCSB 벤치마크를 사용하였고 스레드(Thread)를 32개로 설정하여 쓰기 요청이 집중적으로 발생하도록 하였다. 수행시간과 평균 응답시간은 Clean 상태 대비 각각 11.9% 그리고 7.19% 증가하였다. $99^{th}$-percentile은 73.72%가 증가한 것으로 측정되었다. 실험을 통해 SSD-GC가 카산드라의 꼬리응답시간에 영향을 주는 것을 확인할 수 있다.

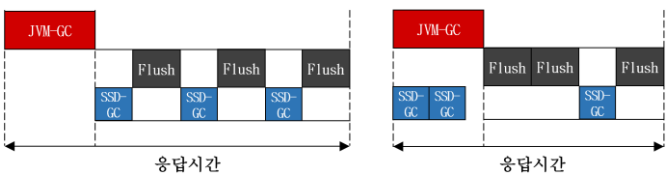
## 3. 가비지 컬렉션 동기화 기법

### 3.1. 구현

호스트가 SSD-GC를 조종하기 위해서는 플래시 메모리와의 인터페이스가 필요하다. 이를 위해 플래시 메모리는 SM843T SSD 모델을 사용한다. 호스트는 SM843T에게 리눅스 2.6 버전부터 사용할 수 있는 SG_IO(SCSI generic I/O) ioctl 명령으로 통신한다. 호스트는 ioctl 명령으로 현재 Free 블록 수를 확인할 수 있고 앞으로 확보할 Free 블록 수를 지정할 수 있다. 호스트가 확보한 Free 블록 수보다 많은 수를 지정하면 SSD-GC가 시작되고, 적은 수를 지정하면 SSD-GC가 멈추게 된다. 이처럼 ioctl 명령을 사용하여 SSD-GC를 시작하는 함수(SyncGC_start)와 멈추는 함수(SyncGC_end)를 구현하였다.

### 3.2. 예상 효과

구현한 함수들을 JVM-GC의 시작과 끝에 위치시켜, 그림 3과 같이 JVM-GC가 시작할 때 SSD-GC도 함께 시작하고 JVM-GC가 끝날 때는 함께 끝나도록 하였다. 기법을 적용하면 JVM-GC 이후의 메모리 테이블을 내려쓰는 시간이 줄고 꼬리응답시간 또한 줄어들 것을 기대한다.




(a) 기법 적용하지 않은 경우 (b) 기법 적용한 경우

그림 3 동기화 기법의 예상 효과

## 4. 실험

본 실험은 동기화 기법이 99th, 99.9th 그리고 99.99th-percentile을 단축하는 효과를 확인한다. 또한 평균 응답시간에 주는 영향을 함께 확인한다. 각 응답시간은 YCSB를 통해 측정하였다. 쓰기 요청이 집중적으로 발생하는 환경을 위해 스레드를 16개에서 64개로 늘려가며 실험을 진행하였다.

### 4.1. 실험 환경

실험은 Linux 4.4.0-38, Intel i7-2600 3.40 GHz x 8 CPU, 16GB 메인 메모리 그리고 SM843T SSD 240GB로 구성한 단일 노드 환경에서 진행하였다. OpenJDK 1.8.0을 힙 공간 12GB로 사용했다. Cassandra 2.2.8을 메모리 테이블 크기 3.33GB로 사용했다. YCSB(Yahoo! Cloud Serving Benchmark) [7]를 사용하여 총 50GB 쓰기 요청을 진행한다. 한번에 1MB 데이터를 쓰기 요청하며, 총 50000번의 요청이 이루어진다.

### 4.2. 실험 결과

실험은 최소 6번 이상 진행하였고 평균을 나타내었다. 표 1은 기법을 적용하지 않았을 때(Origin)와 기법을 적용했을 때(SyncGC)의 응답시간을 나타낸 것이다. 스레드 수가 16, 32 그리고 64일 때, 꼬리응답시간인 99th, 99.9th, 99.99th-percentile과 평균 응답시간이 기술되어 있다. 값의 단위는 ms이다. 모든 스레드에 대해 99th, 99.9th 그리고 99.99th-percentile 값이 줄어든 것으로 측정되었다. 16개의 스레드 경우 99.9th와 99.99th-percentile이 각각 75%와 78% 줄어들었다. 이는 동기화 기법이 미리 SSD-GC를 수행함으로서 얻은 효과이다. 하지만 평균 응답시간은 모든 스레드 수에 대해 증가하는 모습이다. 줄어든 꼬리응답시간과 비교해서, 이러한 평균 응답시간 증가는 작은 비용으로 볼 수 있다.

표 1 동기화 기법의 응답 시간 감소 효과

| | | 99-percentile | 99.9-percentile | 99.99-percentile | Average latency |
|---|---|---|---|---|---|
| Thread16 | Origin | 1507.990 | 14448.256 | 22170.822 | 144.214 |
| | SyncGC | 1316.799 | 3497.191 | 4661.836 | 144.432 |
| Thread32 | Origin | 2386.261 | 15179.356 | 16595.692 | 289.270 |
| | SyncGC | 1868.762 | 7878.459 | 8522.150 | 290.669 |
| Thread64 | Origin | 7663.921 | 20658.174 | 22071.205 | 646.642 |
| | SyncGC | 5189.528 | 14708.132 | 15581.687 | 694.283 |

## 5. 결론 및 향후 연구

본 논문에서는 분산 시스템인 카산드라의 꼬리응답시간을 분석하고 동기화 기법을 제안하였다. 기법을 통해 쓰기 요청이 많이 발생하는 환경에서 꼬리응답시간인 99.9th 그리고 99.99th-percentil이 각각 최대 75%, 78% 감소하였다. 일부 평가의 경우, 동기화 기법을 적용하면 평균 응답시간이 다소 증가하는 것이 관찰되었다. 향후 연구로 적응형 동기화 기법을 개발하여 이 문제를 개선할 계획이다.

## 6. 감사의 글

이 연구를 위해 연구장비를 지원하고 공간을 제공한 서울대학교 컴퓨터연구소에 감사드립니다. 이 논문은 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(NRF-2015M3C4A7065645). (교신저자: 김지홍)

## 참 고 문 헌

[1] A. Lakshman et al., “Cassandra: A Decentralized Structured Storage System,” in Proceedings of the ACM SIGOPS Operating Systems Review, Vol. 44, Issue. 2, pp. 35-40, 2010.

[2] L. Suresh et al., “C3: Cutting Tail Latency in Cloud Data Stores via Adaptive Replica Selection,” in Proceedings of the Symposium on Networked Systems Design and Implementation, pp. 512-527, 2015.

[3] V. Jalaparti et al., “Speeding up Distributed Request-Response Workflows,” in Proceedings of the ACM SIGCOMM Computer Communication Review, Vol. 43, No. 4, pp. 219-230, 2013.

[4] Y. Xu et al., “Bobtail: Avoiding Long Tails in the Cloud,” in Proceedings of the Symposium on Networked Systems Design and Implementation, pp. 329-341, 2013.

[5] L. Fang et al., “Interruptible Tasks: Treating Memory Pressure As Interrupts for Highly Scalable Data-Parallel Programs,” in Proceedings of the Symposium on Operating Systems Principles, pp. 394-409, 2015.

[6] S. Hahn et al., “To Collect or not to collect: Just-in-Time Garbage Collection for High-Performance SSDs with Long Lifetimes,” in Proceedings of the Design Automation Conference, pp. 191, 2015.

[7] B.F. Cooper et al., “Benchmarking Cloud Serving Systems with YCSB,” in Proceedings of the ACM symposium on Cloud Computing, pp. 143-154, 2010.