ISSN 2383-630X(Print) / ISSN 2383-6296(Online)
Journal of KIISE, Vol. 44, No. 8, pp. 767-773, 2017. 8
https://doi.org/10.5626/JOK.2017.44.8.767

클라우드 데이터베이스에서의 꼬리응답시간 감소를 위한 가비지 컬렉션 동기화 기법

(Garbage Collection Synchronization Technique for Improving Tail Latency of Cloud Databases)

한 승 욱 † 한 상 욱 † 김 지 홍 ††
(Seungwook Han) (Sangwook Shane Hahn) (Jihong Kim)

요 약 클라우드 데이터베이스와 같은 분산 시스템 환경에서는 균일한 서비스 품질을 보장하기 위해 꼬리 응답시간을 짧게 유지하는 것이 중요하다. 본 논문에서는 카산드라 데이터베이스를 대상으로, 긴 꼬리 응답시간에 해당하는 지연이 메모리 공간 부족으로 인해 발생한다는 것을 보이며, 이러한 지연이 메모리 공간 확보를 위해 버퍼에 저장된 데이터를 저장장치에 완전히 내려쓸 때까지 카산드라가 사용자의 요청을 받지 않기 때문임을 밝힌다. 버퍼에 저장된 데이터를 내려쓰는데 걸리는 시간은 저장장치 성능에 따라 결정되므로 SSD의 가바지 컬렉션으로 인한 성능 저하가 꼬리 응답시간을 더 길게 만들고 있음을 관찰하였다. 우리는 자바가상기계에서의 가비지 컬렉션과 SSD에서의 가비지 컬렉션을 함께 수행하여 SSD의 가비지 컬렉션 비용을 숨기는, SyncGC 기법을 제안한다. 실험 결과, SyncGC 기법을 통해 꼬리 응답시간인 99.9^{th}와 99.99^{th}-percentile을 각각 31%, 36% 줄일 수 있었다.

키워드: 꼬리 응답시간, 클라우드 데이터베이스, 카산드라, 가비지 컬렉션, 플래시 메모리

Abstract In a distributed system environment, such as a cloud database, the tail latency needs to be kept short to ensure uniform quality of service. In this paper, through experiments on a Cassandra database, we show that long tail latency is caused by a lack of memory space because the database cannot receive any request until free space is reclaimed by writing the buffered data to the storage device. We observed that, since the performance of the storage device determines the amount of time required for writing the buffered data, the performance degradation of Solid State Drive (SSD) due to garbage collection results in a longer tail latency. We propose a garbage collection synchronization technique, called SyncGC, that simultaneously performs garbage collection in the java virtual machine and in the garbage collection in SSD concurrently, thus hiding garbage collection overheads in the SSD. Our evaluations on real SSDs show that SyncGC reduces the tail latency of 99.9^{th} and, 99.99^{th}-percentile by 31% and 36%, respectively.

Keywords: tail latency, cloud database, Cassandra, garbage collection, flash memory

· 이 연구를 위해 연구장비를 지원하고 공간을 제공한 서울대학교 컴퓨터연구소에 감사드립니다.
· 이 논문은 2017 한국연구재단의 재원으로 서울대학교 컴퓨터공학부 BK21플러스 컴퓨터미래인재양성사업단의 지원을 받아 수행된 연구임(21A20151113068)
· 이 논문은 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(NRF-2015M3C4A7065645)
· 이 논문은 제43회 동계학술발표회에서 '클라우드 데이터베이스에서의 꼬리응답시간 감소를 위한 가비지 컬렉션 동기화 기법'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 서울대학교 컴퓨터공학부
swhan@davinci.snu.ac.kr
shanehahn@davinci.snu.ac.kr
†† 종신회원 : 서울대학교 컴퓨터공학부 교수(Seoul Nat'l Univ.)
jihong@davinci.snu.ac.kr
(Corresponding author임)

논문접수 : 2017년 2월 20일
(Received 20 February 2017)
논문수정 : 2017년 4월 17일
(Revised 17 April 2017)
심사완료 : 2017년 4월 17일
(Accepted 17 April 2017)

Copyright©2017 한국정보과학회 : 개인 목적이나 교육 목적인 경우, 이 저작물의 전체 또는 일부에 대한 복사본 혹은 디지털 사본의 제작을 허가합니다. 이 때, 사본은 상업적 수단으로 사용할 수 없으며 첫 페이지에 본 문구와 출처를 반드시 명시해야 합니다. 이 외의 목적으로 복제, 배포, 출판, 전송 등 모든 유형의 사용행위를 하는 경우에 대하여는 사전에 허가를 얻고 비용을 지불해야 합니다.
정보과학회논문지 제44권 제8호(2017. 8)

1. 서 론

응답시간은 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 만들고 실제로 사용되고 있는 카산드라(Cassandra)는 하루에 발생하는 수십억 개의 쓰기 데이터 요청을 처리하기위해 최적화된 분산 데이터 베이스 시스템이다[1]. 하지만, 분산 시스템에서의 응답시간 연구에 따르면 99.9^{th}-percentile의 응답시간이 중간 응답시간(Median Latency)의 수백 배로 분석되어 카산드라와 같은 분산 시스템 환경도 응답시간을 균일하게 유지하는 것이 어려운 상황이다[2]. 균일하지 않은 응답시간 분포에서 긴 시간에 속하는 꼬리 응답시간(Tail Latency)은 사용자 경험(User Experience)에 매우 나쁜 영향을 주기 때문에 꼬리 응답시간의 길에 따라 분산 데이터베이스가 보장하는 서비스 품질이 결정된다.

꼬리 응답시간에 영향을 주는 요소를 크게 네트워크, 메모리 그리고 저장장치로 나눌 수 있다. 카산드라의 꼬리 응답시간을 낮추기 위한 네트워크 수준의 연구[3-5]는 존재했지만, 단일 노드 환경에서 긴 꼬리 응답시간을 줄이는 노력은 간과되어 왔다. 본 논문에서는 메모리와 저장장치 수준에서 꼬리 응답시간의 원인을 분석하고 이를 개선하는 기법을 제안하고자 한다.

메모리의 경우, 자바가상기계(Java Virtual Machine; JVM)가 메모리 관리를 위해 가비지 컬렉션(JVM garbage collection; JVM-GC)을 수행할 때는 자바 응용프로그램들이 실행되지 않기 때문에 카산드라 역시 사용자의 새로운 요청을 받을 수 없게 된다. 때문에 JVM-GC가 발생했을 때는 사용자의 응답이 지연되며, 자바가상기계를 사용하는 다른 분산 시스템들 또한 JVM-GC로 인한 지연 문제를 겪고 있다[6].

카산드라는 요청된 데이터가 메모리에만 쓰여도 응답을 보내어 응답시간을 줄이는 정책을 지원하고 있다. 메모리에 쓰여진 데이터들은 메모리 테이블(Memtable) 단위로 모여서 저장장치에 내려쓰게(Flush) 된다. 하지만 쓰기요청이 집중적으로 발생하여 데이터를 저장할 메모리 공간이 부족해지면 카산드라는 사용자의 쓰기요청을 더 이상 받지 않는다. 이러한 쓰기요청 지연은 하나의 메모리 테이블이 저장장치에 완전히 내려쓰여 메모리에 빈 공간이 확보될 때까지 지속되기 때문에 저장장치의 쓰기 성능이 꼬리 응답시간에 영향을 미친다.

한편 서버 시장에서는 저장장치가 하드 디스크에서 플래시 메모리 기반의 SSD(Solid State Drive)로 전환되는 추세이다[7]. 플래시 메모리는 물리적으로 덮어쓰기를 지원하지 않기 때문에, 플래시 기반인 SSD는 데이터를 새 페이지에 쓴 뒤 원래 페이지를 무효화하는 방식으로 덮어쓰기를 처리한다. 무효페이지가 쌓여 더 이상 새 페이지가 없는 상황이 되면 무효페이지를 수거하는 가비지 컬렉션(SSD garbage collection; SSD-GC)이 발생한다. SSD-GC가 수행되는 중에는 빈 블록을 확보할 때까지 새로운 쓰기 요청을 SSD가 받지 않으며, 빈 블록 확보를 위해 추가로 발생하는 페이지 복사와 블록 삭제로 인해 SSD의 성능이 급격히 떨어진다. 때문에 SSD-GC가 수행될 때 메모리 테이블을 내려쓰면, 쓰기가 지연되어 메모리 테이블 내려쓰기의 수행시간이 길어진다. 늘어난 내려쓰기의 수행시간으로 인해 응답지연은 더 길어지고, 카산드라의 꼬리 응답시간 문제는 더욱 악화된다.

메모리 테이블 내려쓰기 도중에 끼어든 SSD-GC를 피하는 방법은 입출력이 없을 때 SSD-GC를 수행하여 필요한 빈 블록을 미리 확보하는 것이다. 우리는 JVM-GC가 진행되는 동안에 입출력이 이루어지지 않는 점을 이용하여 SSD-GC를 함께 수행하는 동기화 가비지 컬렉션 기법(SyncGC)을 제안한다. SyncGC를 통해 빈 블록을 미리 확보하여, 메모리 테이블 내려쓰기 중에 끼어든 SSD의 성능저하를 피함으로써 길어졌던 꼬리 응답시간을 줄이는 것이 본 연구의 목표이다.

연구 기여는 다음과 같다. 첫번째는 관찰을 통해 긴 꼬리 응답시간과 저장장치간의 관계를 파악한 것이다. 두번째는 SSD-GC가 메모리 테이블의 내려쓰기와 꼬리 응답시간에 미치는 영향을 분석한 것이다. 세번째는 가비지 컬렉션 동기화 기법인 SyncGC를 제안하고 이를 통해 꼬리 응답시간에 해당하는 99.9^{th} 그리고 99.99^{th}-percnetile을 각각 31% 그리고 36% 줄인 것이다.

2. 관련 연구

시시각각으로 응답시간이 변하는 단일 노드들로부터 일정한 응답시간을 보장하기 위해 분산 데이터베이스 시스템은 동일한 데이터를 여러 노드에 분산시킨다[4]. 그 이후에 사용자 요청이 발생하면, 여러 노드들 중 가장 빨리 응답할 수 있는 노드를 선택(Replica Selection)하고 요청을 보내어 꼬리 응답시간을 짧게 유지한다.

하지만 노드 선택을 잘못하게 되면 응답시간이 길어지고, 노드의 상태 정보를 모두 활용하기에는 노드 선택 과정에서 큰 비용이 생겨 응답시간이 길어지게 된다. 때문에 효과적으로 노드를 선택하는 것은 매우 어렵다.

노드들의 과거 응답시간 정보를 기반으로 현재 응답시간을 예측하여 노드 선택을 하는, 적응형 노드 선택 모델이 제안되었다[3]. 하지만 단일 노드의 응답시간에

영향을 주는 이벤트인, 메모리 관리를 위해 주기적으로 호출되는 가비지 컬렉션과 메모리 테이블 관리에서 요구되는 많은 양의 입출력을 직접 반영한 것이 아니기 때문에, 노드 선택을 잘못할 가능성이 여전히 있다[5]. 이처럼 네트워크 수준에서의 노드 선택 최적화로 단일 노드의 긴 응답시간을 극복하기에는 한계가 있기 때문에 단일 노드 환경에서 꼬리 응답시간을 분석하고 이를 줄이는 노력이 필요하다.

3. 꼬리 응답시간 분석

3.1 단일 노드 환경에서의 꼬리 응답시간

카산드라의 꼬리 응답시간을 분석하기 위해 YCSB 벤치마크로 쓰기요청에 대한 응답시간을 측정하였다. 그림 1은 측정된 응답시간을 누적 분포로 표현한 것이다. 쓰기요청의 99%는 0.1초 이하의 응답시간으로 서비스된 반면, 나머지 1%는 긴 응답시간을 보이고 있다. 긴 응답시간은 약 2초까지 점차적으로 증가하였는데, 그 이후에는 6초 이상으로 급증하는 모습이다. 특히 가장 긴 응답시간이 약 13초로, 99%에 해당하는 0.1초와 비교하면 이는 매우 긴 꼬리 응답시간이다. 카산드라는 쓰기요청된 데이터를 메모리에 저장한 뒤 바로 응답을 보내기 때문에 대부분의 경우에는 짧은 응답시간을 보이지만, JVM-GC 혹은 메모리 공간 부족으로 인한 지연 때문에 최악의 경우에는 수백 배의 시간을 보이고 있다.

3.2 꼬리 응답시간의 원인

그림 2는 하나의 메모리 테이블 내려쓰기가 시작해서 끝날 때까지의 응답시간 변화를 타임라인으로 나타낸 것이다. 또한 메모리 테이블 내려쓰기 동안에 발생한 JVM-GC의 진행 시간이 함께 표현되어 있다. 그림 2에서 확인할 수 있듯이, 총 3번의 JVM-GC가 약 1~2초간 진행되고 있다. 예를 들어 (a)로 표시한 응답시간 지연은 약 2초 동안 진행된 JVM-GC로 인해 발생하였으며, JVM-GC가 수행되는 동안에 어떠한 요청도 응답을 받지 못하고 있다. 이처럼 지연되던 요청은 JVM-GC가 끝난 후 비로서 응답을 받아 약 2초의 응답시간을 기록하고 있다.

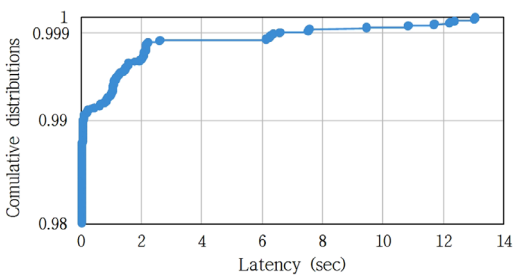


그림 1 긴 응답시간의 누적 분포 함수

Fig. 1 Cumulative distributions of long latency

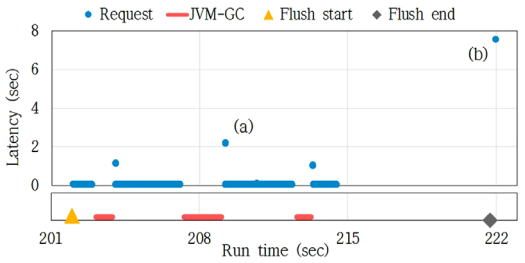


그림 2 메모리 테이블 내려쓰기 동안의 응답시간 변화

Fig. 2 Varied latencies during Memtable flush

JVM-GC로 인한 약 1~2초간의 응답시간 지연에 비해서, (b)에 해당하는 응답시간 지연은 특별히 긴 시간인 약 8초에 해당한다. 이러한 긴 지연은 메모리 공간이 부족한 상황에서 카산드라가 사용자의 쓰기요청을 받지 않을 때 발생한다. 메모리 테이블이 저장장치에 완전히 내려 쓰이기 전까지 카산드라는 해당 메모리 테이블을 메모리 공간에 유지하기 때문에, 메모리 공간 부족으로 인한 지연은 한번의 메모리 테이블이 완전히 내려쓰일 때까지 지속된다. 예를 들어 (b)로 표시한 긴 응답시간은 215초 시점부터 시작된 메모리 공간 부족으로 만들어진 지연이다. 내려쓰기가 완료되기 전까지 어떠한 요청도 응답을 받지 못하고 있으며, 이 요청들은 내려쓰기가 완료된 직후에 응답을 받아 약 8초에 해당하는 매우 긴 응답시간을 보이고 있다.

대부분의 요청이 0.1초 내에 응답을 받지만 JVM-GC와 메모리 공간 부족으로 인해 긴 응답시간 지연이 생겼다. 하지만 JVM-GC로 인한 약 2초간의 지연보다 메모리 공간 부족으로 인한 지연은 훨씬 심각한 8초로, 이는 그림 1에서 확인하였던 긴 꼬리 응답시간에 해당한다. 결국 긴 꼬리 응답시간에 해당하는 지연은 메모리 공간이 부족할 때 만들어지며, 이때 발생하는 지연을 줄여야 긴 꼬리 응답시간을 줄일 수 있을 것으로 분석된다.

3.3 꼬리 응답시간에 대한 SSD-GC의 영향

메모리 공간 부족으로 인한 지연은 메모리 테이블을 내려쓰는 속도의 영향을 받는다. 내려쓰기 속도는 SSD의 성능에 따라 결정되기 때문에, SSD의 성능에 따라서 꼬리 응답시간이 달라진다. 하지만 빈 블록을 확보하기 위해 수행되는 SSD-GC는 SSD의 성능을 급격히 떨어뜨린다. 우리는 SSD-GC로 인한 SSD의 성능 저하가 꼬리 응답시간에 얼마나 영향을 주는지를 분석하기 위해 실험을 진행하였다. 실험은 SSD의 상태를 Clean과 Dirty로 나누어 SSD-GC가 적게 발생하는 경우와 많이 발생하는 경우를 비교하고자 한다. Clean 상태는 무효페이지가 없는 초기 상태의 SSD를 뜻하고, Dirty 상태는 SSD의 최대 용량만큼 썼다가 지워 무효페이지가

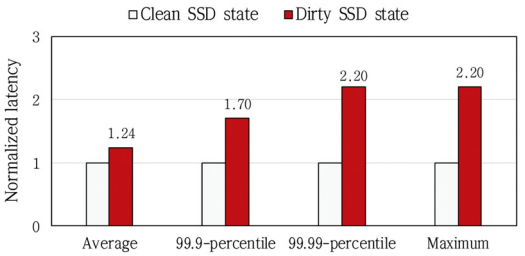
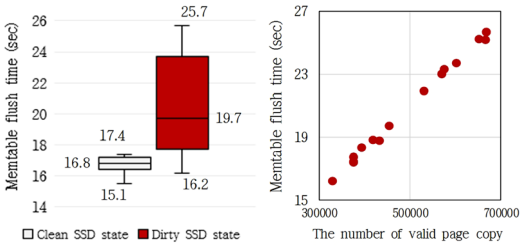


그림 3 SSD-GC가 꼬리 응답시간에 미치는 영향
Fig. 3 Impact of SSD-GC on tail latency



(a) Flush time by the SSD state

(b) Flush time by the number of page copy

그림 4 SSD-GC가 내려쓰기 시간에 미치는 영향
Fig. 4 Impact of SSD-GC on flush time

많은 상태이다. 그림 3은 평균, 99.9th, 99.99th-percentile 그리고 최대 응답시간을 Clean 상태와 비교하여 SSD-GC가 응답시간에 끼치는 영향을 나타내고 있다. 꼬리 응답시간인 99.9th, 99.99th-percentile 그리고 최대 응답시간이 Clean 상태 대비 각각 70%, 120% 그리고 120% 증가한 것으로 측정되었다. SSD-GC로 인한 SSD의 성능저하로 카산드라의 긴 꼬리 응답시간 문제가 더욱 악화된 모습을 확인할 수 있다.

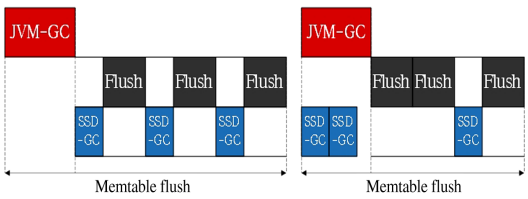
SSD-GC가 꼬리 응답시간에 미친 영향을 구체적으로 파악하기 위해, 메모리 테이블 내려쓰기의 수행 시간과 내려 쓰기 중에 끼어든 SSD-GC의 비용을 분석해 보았다. 50GB의 쓰기요청을 3.33GB 크기의 메모리 테이블로 내려쓰기 때문에 총 15번의 내려쓰기가 발생하는데, 최초와 마지막의 내려쓰기를 제외한, 총 13번의 메모리 테이블 내려쓰기의 수행 시간을 수집하였다. 그림 4(a)는 Clean과 Dirty 상태에서 내려쓰기 수행 시간을 박스 그래프로 나타낸 것이다. Dirty 상태에서 내려쓰기 수행 시간이 Clean 상태에 비해 평균 24% 증가하였다. 최고값이 Clean상태 대비 47% 증가하였는데, SSD-GC로 인한 성능저하가 내려쓰기의 수행 시간을 약 50% 가까이 증가시킨 것이다. 그림 4(b)는 SSD-GC 비용에 대부분을 차지하는 유효 페이지 복사(Valid Page Copy) 횟수와 내려쓰기의 수행 시간을 나타내었다. 메모리 테이블을 내려쓰는 동안에 이루어진 페이지 복사 수에 따라 내려쓰기 시간이 증가하며, 선형의 관계를 보이고 있다. 즉, 메모리 테이블 내려쓰기 중에 끼어든 SSD-GC의 비용이 클수록 내려쓰기 진행 시간이 증가하고 있다. 따라서 미리 SSD-GC를 수행하여 내려쓰기 중에 끼어든 SSD-GC를 피한다면, 내려쓰기 수행 시간을 줄여 긴 꼬리 응답시간이 줄어들 것이다.

4. 가비지 컬렉션 동기화 기법

SyncGC 기법은 JVM-GC와 SSD-GC가 함께 시작하고 함께 끝나는 것을 목표로 한다. 그림 5를 보면, SyncGC를 적용했을 때 JVM-GC와 SSD-GC가 함께 수행되어 메모리 테이블 내려쓰기 동안에 발생한 SSD-GC 시간이 줄어든다. 이처럼 SyncGC는 미리 SSD-GC를 수행하여 메모리 테이블을 내려쓰는데 걸리는 시간을 줄이고 결과적으로 긴 꼬리 응답시간을 줄이게 된다.

그림 6은 SyncGC 의 동작 순서와 전체 구조를 나타내고 있다. (1)~(2) 어플리케이션의 메모리 요청이 발생하면 자바가상기계가 힙에서 메모리를 할당하지만 힙에 충분한 공간이 없다면 JVM-GC가 수행된다. (3) JVM-GC가 시작되면 자바가상기계는 SyncGC 모듈에게 JVM-GC가 시작되었음을 알리고, (4) SyncGC 모듈은 JVM-GC의 시작을 탐지하여 SSD-GC 시작 명령을 SyncGC_Start 함수를 통해 내린다. (5) JVM-GC가 모두 완료되면 자바가상기계는 JVM-GC가 종료되었음을 SyncGC 모듈에게 전달하고, (6) SyncGC 모듈은 SyncGC_Stop 함수를 통해 SSD-GC 종료 명령을 내린다.

JVM-GC와 SSD-GC가 함께 시작되려면, SyncGC 모듈이 JVM-GC가 시작되었음을 먼저 탐지할 수 있어야 한다. JVM-GC 시작의 탐지는 JVM-GC 수행 함수가 시작될 때 자바가상기계가 JVM-GC가 시작됨을 SyncGC 모듈에게 알림으로써 이루어진다. OpenJDK 1.8.0 자바가상기계가 기본 설정으로 사용하는 가바지 컬렉션은 PSScavenge::invoke 함수에서 실행되는데, invoke 함수의 시작 위치에 SyncGC_Enter 함수를 추



(a) Original (b) SyncGC

그림 5 기존 방식과 SyncGC와의 비교
Fig. 5 Comparison of original scheme and SyncGC

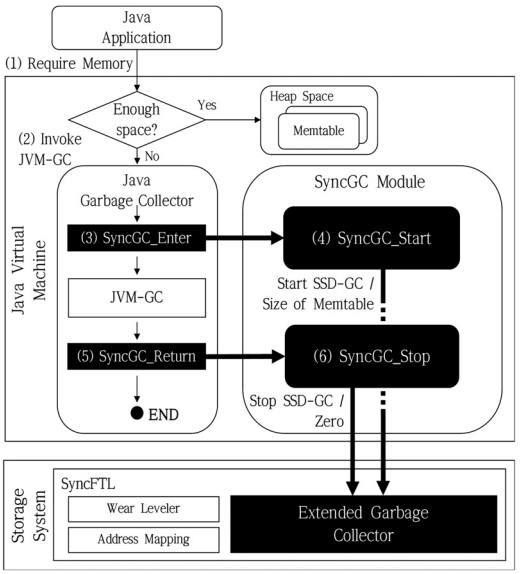


그림 6 SyncGC의 전체 구조
Fig. 6 Overall architecture of SyncGC

가하여 SyncGC 모듈이 JVM-GC가 시작되는 것을 탐지할 수 있도록 하였다.

JVM-GC가 시작되었음을 탐지한 SyncGC는 SSD-GC 시작 명령을 내리기 위해 SyncGC_Start함수를 호출한다. SyncGC_Start함수는 SG_IO(SCSI generic I/O) 인터페이스를 통해 SSD 내부 동작을 관리하는 FTL (Flash Translation Layer)에게 SSD-GC 시작 명령과 “목표 빈 블록 수”를 보낸다. 이때 FTL은 “목표 빈 블록 수”만큼의 빈 블록을 확보하기 위해 SSD-GC를 수행할 수 있어야한다. 때문에 상용 SSD인 ‘SM843T’의 FTL을 수정하여, “목표 빈 블록 수”만큼의 빈 블록을 확보할 때까지 SSD-GC를 수행하는 SyncFTL을 구현하였다[8]. SyncGC_Start 함수는 “목표 빈 블록 수”를 메모리 테이블 크기만큼의 블록 수로 설정하여 SSD-GC 시작 명령을 내리고, SyncFTL은 그 수만큼의 빈 블록을 확보할 때까지 SSD-GC를 수행한다.

이후 JVM-GC가 모두 완료되면, 자바가상기계는 PSScavenge::invoke 함수의 끝 위치에 추가한 SyncGC_Return 함수를 통해 SyncGC 모듈이 JVM-GC의 종료를 탐지한다. SyncGC 모듈은 SSD-GC 정지 명령을 내리기 위해서 SyncGC_Stop함수를 호출한다. SyncGC_Stop함수는 “목표 빈 블록 수”를 0으로 설정한 SSD-GC 정지 명령을 SG_IO 인터페이스를 통해서 SyncFTL에게 전달한다. “목표 빈 블록 수”로 0을 받은 SyncFTL은 빈 블록을 더 이상 확보할 필요가 없기 때문에 진행중인 SSD-GC를 멈추게 된다.

SyncGC는 JVM-GC가 시작되었음을 감지하면 항상 SyncGC_Start 함수를 호출하는데, 만약 JVM-GC 수행시간이 충분히 길지 않은 경우에는 SSD-GC를 통해 빈 블록을 하나도 얻지 못할 수 있다. 따라서 SyncGC가 SSD-GC 시작과 SSD-GC 종료를 위해 내린 두 번의 명령은 시간 비용이 된다. 하지만 SG_IO 인터페이스를 통해 호스트에서 SyncFTL로 데이터를 전송하는데 드는 시간적 비용은 약 160 us로, JVM-GC의 평균 수행 시간인 400 ms의 0.04%에 해당하는 매우 작은 비용이다. 따라서 두 번의 SG_IO 명령으로 발생한 시간 비용은 무시할 수 있는 수준이라고 할 수 있다.

5. 기법 평가

본 실험에서는 SyncGC 기법을 통해 99.9^{th}, 99.99^{th} 그리고 최대 응답시간이 단축된 정도를 확인한다. 또한 SyncGC 기법이 평균 응답시간에 주는 영향을 함께 확인한다. 각 응답시간은 YCSB를 통해 측정하였다. 쓰기요청이 집중적으로 발생하는 환경을 만들기 위해 스레드를 8개로 하여 실험을 진행하였다. 꼬리 응답시간 분석을 위해 진행한 실험 역시 동일한 환경에서 이루어졌다.

5.1 실험 환경

실험 환경은 Linux 4.4.0-38, Intel i7-2600 3.40 GHz × 8 CPU, 16GB 메인 메모리 그리고 삼성 SM843T SSD 240GB [8]에 호스트와의 인터페이스를 추가하여 구성하였다. OpenJDK 1.8.0을 힙 공간 12GB로 사용했다. Cassandra 2.2.8을 메모리 테이블 크기 3.33GB로 사용했다. YCSB(Yahoo! Cloud Serving Benchmark) [9]를 통해 총 50GB 쓰기 요청을 진행한다. 한번에 1MB 데이터를 쓰기요청하며, 총 50,000번의 요청이 이루어진다.

5.2 실험 결과

실험은 최소 5번 이상 진행하였고 중간 값의 결과를 나타내었다. 표 1은 동기화 기법을 적용하지 않은 경우(Original)와 동기화 기법을 적용한 경우(SyncGC)의 응답시간을 나타낸 것이다. 평균 응답시간과 99.9^{th}, 99.99^{th}-percentile 그리고 최대 응답시간이 기술되어 있다. 값의 단위는 ms이다. 99.9^{th}, 99.99^{th}-percentile 그리고 최대 응답시간 값이 모두 줄어든 것으로 측정되었다. 기법을 적용하지 않은 경우와 비교했을 때, 99.9^{th}와 99.99^{th}-percentile이 각각 31% 그리고 36% 줄어들었다. 이는 SyncGC가 미리 SSD-GC를 수행함으로서 얻은 효과이다. 평균 응답시간 역시 12% 가량 감소하였는데, 사용자 요청을 받지 못하는 JVM-GC 시간을 활용하였기 때문에 SyncGC는 큰 시간적 비용 없이 꼬리 응답시간을 낮출 수 있다.

표 1 동기화 기법으로 인한 꼬리 응답시간 감소
Table 1 Reduced tail latencies by SyncGC

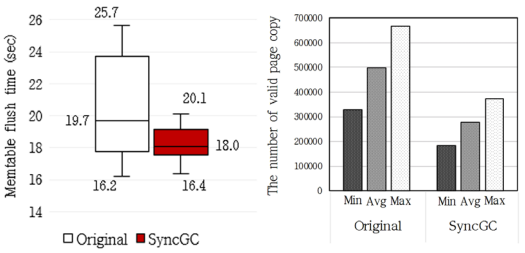
	99.9-percentile	99.99-percentile	Max latency	Average latency
Original	9473	13068	13079	51
SyncGC	6570	8358	8384	45

5.3 결과 분석

SyncGC를 통해 메모리 테이블의 내려쓰기 시간이 단축되는 정도를 분석해 보았다. 그림 7(a)는 메모리 테이블의 내려쓰기 수행 시간을 박스 그래프로 나타낸 것이다. SyncGC가 메모리 테이블의 내려쓰기 수행 시간을 평균 12% 감소시켰다. 특히 최고 수행 시간이 기법을 적용하지 않은 경우와 비교 했을 때, 약 5초 가량 감소하였다. 이는 표 1에서 확인 되는 최대 응답시간의 감소 정도와 같다. 특징적인 것은 SyncGC를 통해 내려쓰기 수행 시간의 변화 폭이 줄어들었다는 점이다. 이처럼 메모리 공간이 부족한 상황에서 SSD-GC로 인한 긴 지연을 줄이면, 단일 노드에서의 응답시간의 변화를 줄여 네트워크 수준의 노드 선택을 더욱 효과적으로 만들 것으로 보인다.

SyncGC을 통해, 내려쓰기가 진행될 때 끼어든 유효 페이지 복사 수가 얼마나 줄어드는지를 그림 7(b)에 나타내었다. 기법을 적용하지 않은 경우 대비, 평균과 최대의 끼어든 유효 페이지 복사 횟수는 모두 44% 감소하였다. 내려쓰기를 하지 않는 JVM-GC 시간에 페이지 복사를 미리 수행하였기 때문에 내려쓰기가 진행될 때 끼어든 페이지 복사 수가 줄어든 것이다.

그림 8은 응답시간의 누적 분포를 나타내고 있다. JVM-GC로 인한 약 2초 가량의 지연은 기법을 적용하여도 줄어들지 않았다. 하지만 6초 이상의 메모리 테이블 내려쓰기로 인한 지연에 대해서는 SyncGC를 통해 끼어든 SSD-GC를 피하여 응답시간 지연이 상당히 줄어 들었다.



(a) The reduced flush time (b) The reduced number of page copy

그림 7 SyncGC가 I/O 성능에 미치는 영향
Fig. 7 Impact of SyncGC on I/O performance

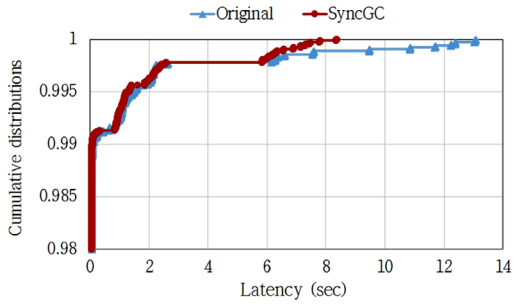


그림 8 감소한 응답시간의 누적 분포
Fig. 8 Cumulative distributions of reduced latencies

6. 결론 및 향후 연구

본 논문에서는 분산 시스템인 카산드라의 꼬리 응답시간을 분석하고 JVM-GC와 SSD-GC를 동시에 수행하는 SyncGC를 제안하였다. SyncGC를 통해 쓰기요청이 많이 발생하는 환경에서의 긴 꼬리 응답시간에 해당하는 99.9^{th}, 99.99^{th}-percentile을 각각 31%, 36% 줄일 수 있었다. 향후 연구로는 엔터프라이즈 환경에서 대용량 메모리 관리로 발생하는 매우 긴 JVM-GC 시간을 SSD-GC와 효과적으로 동기화하기 위해 적응형 동기화 기법을 개발할 계획이다.

References

[1] A. Lakshman and P. Malik, "Cassandra: A Decentralized Structured Storage System," *ACM SIGOPS Operating Systems Review*, Vol. 44, issue 2, pp. 35-40, 2010.

[2] Y. Xu, Z. Musgrave, B. Noble, and M. Bailey, "Bobtail: Avoiding Long Tails in the Cloud," *Proc. of the Networked Systems Design and Implementation*, pp. 329-341, 2013.

[3] L. Suresh, M. Canini, S. Schmid, and A. Feldmann, "C3: Cutting Tail Latency in Cloud Data Stores via Adaptive Replica Selection," *Proc. of the Networked Systems Design and Implementation*, pp. 512-527, 2015.

[4] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker, "Low Latency via Redundancy," *Proc. of the ACM Conference on Emerging Networking Experiments and Technologies*, pp. 283-294, 2013.

[5] J. Dean and L. A. Barroso, "The Tail at Scale," *ACM Communications*, Vol. 56, No. 2, pp. 74-80, 2013.

[6] L. Fang, K. Nguyen, G. Xu, B. Demsky, and S. Lu, "Interruptible Tasks: Treating Memory Pressure As Interrupts for Highly Scalable Data-Parallel Programs," *Proc. of the Symposium on Operating Systems Principles*, pp. 394-409, 2015.

[7] F. Ober and Q. Zhu. Intel: Next Generation SSDs & ForestDB Next Generation Storage Engine (CouchbaseConnect15). [Online]. Available: http://connect15.couchbase.com/agenda/intel-next-generation-ssds-forestdb-next-generation-storage-engine
[8] SAMSUNG. Data Center Series. [Online]. Available: http://memorysolution.de/mso_upload/out/all/SM843T_Specification_v1.0.pdf
[9] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking Cloud Serving Systems with YCSB," *Proc. of the ACM Symposium on Cloud Computing*, pp. 143-154, 2010.

한 승 욱
2016년 한양대학교 컴퓨터 공학부 학사
2016년~현재 서울대학교 컴퓨터공학부 석사과정. 관심분야는 플래시 저장장치, 임베디드 소프트웨어, 데이터베이스 시스템, 운영체제

한 상 욱
정보과학회논문지
제 44 권 제 3 호 참조

김 지 홍
정보과학회논문지
제 44 권 제 3 호 참조