JANT 한국항행학회논문지
Journal of Advanced Navigation Technology

기반 기술

J. Adv. Navig. Technol. 20(1): 79-85, Feb. 2016

NIO를 이용한 범용 웹 캐시 구현

General Web Cache Implementation Using NIO

이 철 희 · 신 용 현*
서울과학기술대학교 컴퓨터공학과

Chul-Hui Lee · Yong-Hyeon Shin*
Department of Computer Science & Engineering, Seoul National University of Science and Technology, Seoul 01811, Korea

[요 약]

최근의 웹 환경은 스마트폰과 같은 모바일, 페이스북과 같은 소셜 네트워크의 증가로 인하여 네트워크의 트래픽이 급격히 증가하고 있다. 본 논문에서는 WAS(web application server)의 애플리케이션에서 기존 자바의 단점인 I/O의 블로킹(blocking)과 버퍼에 가비지 컬렉션 (garbage collection)으로 인한 CPU 성능 감소 등의 문제를 NIO (non-blocking IO)의 다이렉트 버퍼와 DMA (direct memory access)를 이용하여 기존 시스템의 웹 응답 속도를 향상시켰다. 우선순위 변동 등으로 상대적으로 데이터 순환이 많은 키 값은 조작이 용이한 해시맵에 담아 캐시 우선순위 변경 알고리즘을 적용한다. 용량이 큰 응답 데이터는 속도가 빠른 다이렉트 버퍼에 분리 저장하여 성능을 높인다. 캐시 적중 시와 적중이 안 될 경우의 여러 상황에서의 실험을 통해 본 논문에서 제안한 NIO를 이용한 방법이 많은 성능 향상을 보여줌을 확인할 수 있다.

[Abstract]

Network traffic is increased rapidly, due to mobile and social network, such as smartphones and facebook, in recent web environment. In this paper, we improved web response time of existing system using direct buffer of NIO and DMA. This solved the disadvantage of JAVA, such as CPU performance reduction due to the blocking of I/O, garbage collection of buffer. Key values circulated many data due to priority change put on a hash map operated easily and apply a priority modification algorithm. Large response data is separated and stored at a fast direct buffer and improved performance. This paper showed that the proposed method using NIO was much improved performance, in many test situations of cache hit and cache miss.

Key word : Web, Cache, Nonbloking input-output, Direct buffer.

http://dx.doi.org/10.12673/jant.2016.20.1.79

Received 26 January 2016; **Revised** 5 February 2016
Accepted (Publication) 18 February 2016 (28 February 2016)

***Corresponding Author; Yong-Hyeon Shin**

Tel: +82-2-970-6709
E-mail: yshin@seoultech.ac.kr

Ⅰ. 서 론

최근의 웹 환경은 스마트폰의 폭발적인 보급 등으로 인하여 트위터(twitter), 페이스북(facebook) 소셜네트워크의 방대한 텍스트 및 이미지 데이터와 멜론(melon), 유튜브(youtube) 등의 스트리밍 데이터의 증가[1]와 IoT (internet of things) 등의 시장 확대로 네트워크의 트래픽은 더욱 증가되고 있다. 시스코사는 2018년 전 세계 IP 트래픽을 1.6 ZB (zetta byte)로 예측하고 있으며 2018년 한국 IP 트래픽 또한 68.6 EB (exa byte)로 2013년 대비 2배 증가할 것으로 예상하였다[2].

현재 기가 인터넷 등으로 네트워크 속도가 많이 향상되긴 했지만 데이터 및 사용자 증가 속도를 훨씬 못 미치고 있다. 그로 인하여 DW (data warehouse) 등으로 폭발적으로 늘어난 데이터 관리, 분산 서버 등을 이용한 네트워크 트래픽 분산 기술과 웹 캐시 방식을 사용한 계층 간 트래픽 감소 기술 등이 오래 전부터 연구되어왔다[3]. 위 예시에서의 데이터 특성에서 보듯이 멜론의 음악, 유튜브의 영상, 트위터와 페이스북의 메시지 타입의 글 또한 한번 작성되면 내용의 수정이 거의 없으며, 또한 여러 명의 사용자가 상대적으로 짧은 기간에 같은 데이터를 요청하여 캐시를 사용했을 시 높은 적중률을 보일 수 있는 패턴이다. 캐시 기법은 프락시 서버 등을 사용한 하드웨어 방식과 웹 브라우저에서 한번 조회되었던 정적 데이터를 로컬에 저장하는 브라우저 캐싱 기법 등이 있으며 WAS (web application server)에서 데이터베이스 호출 시에 사용되는 애플리케이션 캐시가 있다.

즉, 하드웨어의 발전 속도는 트래픽의 증가량을 감당할 수 없으므로 사용자만 다르고 일정 기간 같은 내용의 여러 형태의 콘텐츠를 빠르게 제공받기위한 방식으로 캐시를 적극적으로 활용할 수 있다.

Ⅱ. 웹 캐시 구현

2-1 시스템 개요

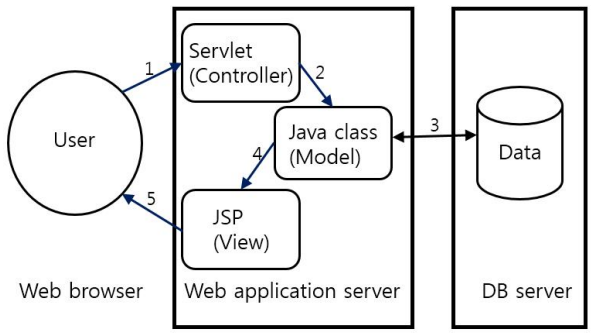

그림 1. 데이터 흐름
Fig. 1. Data flow.

본 논문에서 제안한 NIO를 이용한 범용 웹 캐시는 WAS에서 자바로 구성된 DB 및 파일서버 응답 캐시 기능을 구현한 것이다.

그림 1에서 볼 수 있듯이 요청(request)이 들어오면 WAS에서 DB 서버에 요청 후 받은 내용을 뷰(view)를 통해 사용자에게 응답(reply)해 주는 형태다. 이와 같은 내용은 웹 브라우저 캐시에서는 서로 다른 사용자 간에 캐싱된 내용이 존재하지 않으므로 캐싱이 불가능하고 프록시 서버에서 캐싱을 할 경우에는 WAS에서 화면 및 데이터가 동적으로 생성되는 동적 데이터로 인해 캐싱에 어려움이 있다. 본 논문에서는 사용자가 요청한 내용을 콘트롤러(controller)에서 체크하여 같은 요청이 있었을 시 다이렉트 버퍼 (direct buffer)에 저장되어있는 내용을 직접 뷰로 보내므로 DB 서버와의 I/O를 없앤다. 또한 다이렉트 버퍼를 서버 시작 시에 초기화하여 초기화 하는 시간을 줄이며 OS의 버퍼를 사용하여 자바 가비지 컬렉션 자원 감소를 피할 수 있게 하였다[6],[7].

2-2 상세 기능

1) 키 값과 데이터를 분리 저장하여 속도향상

사용자의 요청 정보와 응답 정보를 별도의 버퍼에 저장하여 데이터 우선순위 정렬시 속도를 향상한다. 요청 정보는 해시맵(hash map) 스타일의 변수에 담아서 여러 캐시 우선순위 스케줄링 알고리즘에 적용할 수 있게 하며, 저장된 데이터의 우선순위와 실제 DB 응답 값이 들어갈 다이렉트 버퍼의 좌표 정보를 담는다.

응답 정보는 데이터 순환이 어렵고 데이터가 크므로 데이터의 우선순위만 변경 시에는 변경이 없으며 삭제 및 신규 생성이 있을 경우만 수정 및 변경 작업을 하여 속도 향상을 꾀하였다.

2) 캐시 우선순위 변경 알고리즘

실제 캐시 사용 시에 캐시 우선순위 스케줄링 알고리즘이 적중률 (hit ratio)을 높이는 데 중요하다. 여러 스타일의 우선순위 스케줄링 알고리즘이 적용 가능하도록, 실제 적용되는 곳에서 실제 버퍼와는 별도로 요청 값의 키와 우선순위 순서를 적용하는 해시맵을 사용하여 데이터를 분리하였다.

2-3 시스템 구현

1) 시스템 구성

전체적인 구성은 기존의 MVC2 모델이며 거기에 추가로 캐시 기능을 첨부한 형태이다. 전체적인 시스템 로직은 그림 2와 같이 분기되어 이루어진다.

처음 요청 시, 캐시 데이터가 히트하면, 데이터 교환의 3가지 로직으로 이루어 있으며 버퍼가 데이터 응답 내용을 저장하며 해시맵에서 우선순위 컨트롤과 요청 키 값을 가지고 있다.

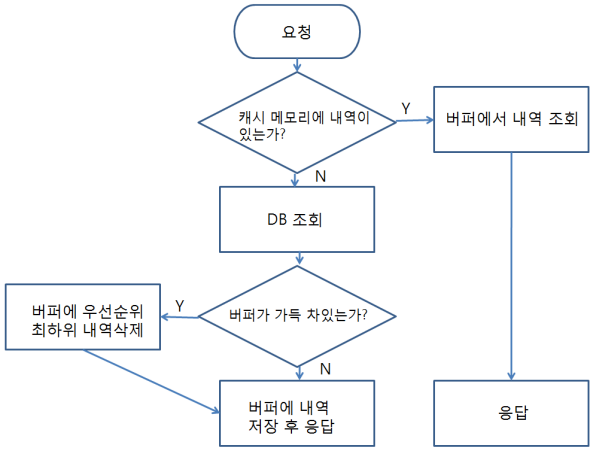

그림 2. 데이터 응답 순서도
Fig. 2. Response data flow diagram.

2) 캐시 시스템 구성

캐시 시스템 구성은 콘트롤러에서 사용자가 요청하는 것을 가로채서 메모리에 요청한 내용과 같은 것이 있는지 체크하는 체크로직과 실제 응답 내용이 들어가는 다이렉트 버퍼 저장방식 그리고 우선순위에 의해서 요청 내용과 응답 내용을 적용 및 삭제하는 우선순위 알고리즘으로 구성되어있다.

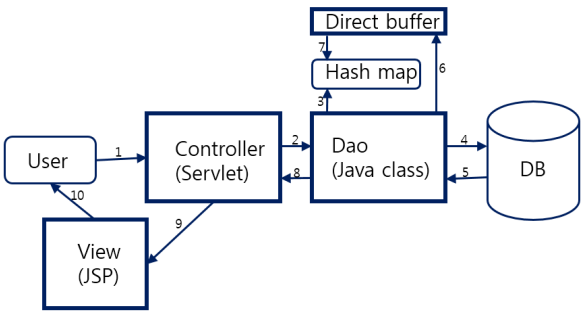

그림 3. 첫 요청
Fig. 3. The first request.

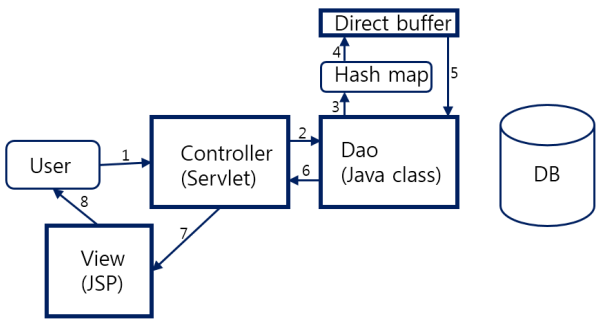

그림 4. 적중 되었을 경우
Fig. 4. In case of hit.

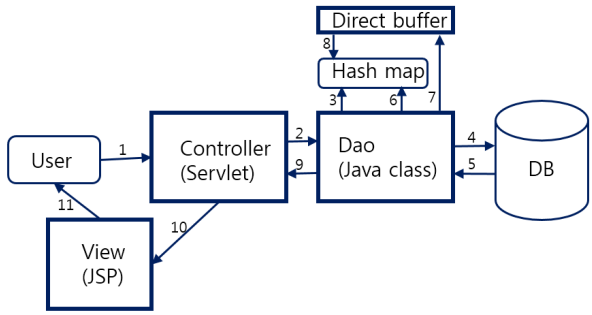

그림 5. 데이터 교환
Fig. 5. Data exchange.

서버 시작 후 사용자의 첫 번째 요청이 들어오면 전반적인 데이터 흐름은 같지만 그림 3과 같이 초기에 해시맵에 요청 값이 있는지 확인한 후 같은 값이 없으면 DB에서 응답 값을 구해온 후 다이렉트 버퍼에 값을 입력한다. 요청한 키 값과 버퍼의 시작 종료 지점을 해시맵에 저장한 후 DB 응답 값을 뷰에 전달한다.

사용자의 요청이 Dao에서 체크 하였을 때 해시맵에 키 값이 존재하게 되면 그림 4에서와 같이 확인된 키 값으로 다이렉트 버퍼의 요청 데이터의 시작 값과 종료 값으로 데이터를 요청하여 바로 콘트롤러에 값을 전달해 준다. DB 서버와의 I/O 없이 뷰에 바로 응답 값을 전달할 수 있으며 이후에 각 캐시 우선순위 스케줄링 알고리즘에 의한 우선순위를 해시맵에 적용하게 된다.

그림 5는 데이터 교환 시의 데이터 흐름이다. 사용자의 요청을 Dao에서 체크 후 해시맵에 내용이 없음을 확인하면 DB에서 값을 조회한 후 해시맵에 캐시 우선순위 스케줄링 알고리즘에 의한 우선순위를 체크하여 삭제 대상을 확인한다. 확인된 삭제 대상의 값으로 다이렉트 버퍼에 값을 삭제한 후 해시맵 값도 삭제하고 이번 요청 값을 신규로 입력하는 작업을 한다.

3) 자료 순환 구조

자료 순환 구성의 캐시 우선순위 스케줄링 알고리즘은 LRU (least recently used) 알고리즘[4][5]을 사용하여 설명하였으며 테스트 구현 또한 LRU알고리즘을 사용하여서 테스트하였다. 자료 순환 시의 세부내용은 첫 번째 요청 시 그림 6과같이 3개의 저장 공간을 가지거나 400이라는 저장 공간을 초기 값으로 설정한다. A 요청이 0~100의 크기로, 다음으로 B 요청이 0~200의 크기로, C 요청이 0~50의 크기로 순서대로 요청 시에 LRU 알고리즘일 때의 저장 상황을 표현하였다.

LRU 알고리즘의 경우 가장 오랜 기간 사용되지 않은 내용부터 삭제되므로 적중되었을 시 내용의 우선순위를 변경해야 한다. 그림 7에서처럼 A 요청이 적중되면 해시맵에의 우선순위를 변경하여 주고 다이렉트 버퍼에서는 데이터 삭제 및 수정을 하지 않는다.

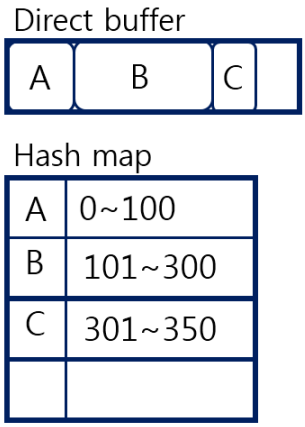

그림 6. 첫 요청
Fig. 6. The first request.

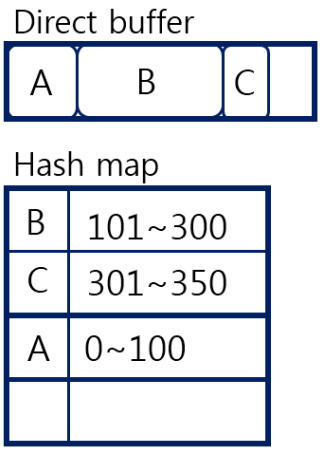

그림 7. 적중시 자료순환
Fig. 7. Hit data circulation.

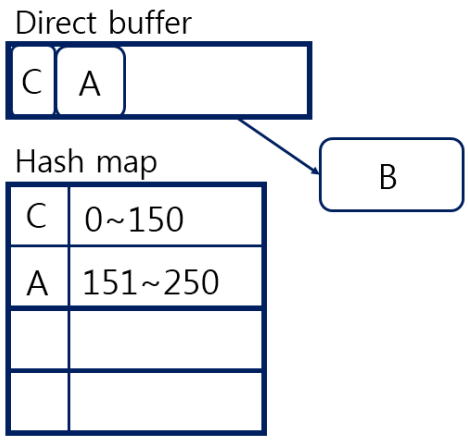

그림 8. 데이터 교환
Fig. 8. Data exchange.

데이터 교환 시에는 그림 8과 같이 우선순위 알고리즘에 의한 순서대로 해시맵 데이터를 삭제 후 같은 내용의 다이렉트 버퍼 값도 삭제해준다. 이후에 다이렉트 버퍼에 빈공간이 생길 경우 데이터 저장 및 삭제가 어려우므로 해시맵의 우선순위대로 값을 정렬하여 입력하는 작업을 하여준다. 이후 신규 조회된 DB 값을 저장한 후 저장 시작 값과 마지막 값을 해시맵에 저장하여 준다.

표 1. 구현 및 실험환경
Table 1. Implement and test environment.

DB	Oracle11g express edition
OS	Windows 7 professional K service pack 1 64bit
CPU	intel Core2Cpu 6600@2.4Ghz
RAM	8.00GB
DISK	SSD 125GB
development tool	Eclipse 4.5.1
Server	Apache tomcat 8.0
java version	jdk 1.8
existing cache	MyBatis 3.3.0
test tool	Apache JMeter version 2.13

Ⅲ. 실험 및 검토

3-1 실험 환경

실험 환경은 표 1과 같다. NIO 사용 시에 점유한 버퍼 메모리는 나머지 테스트에서는 점유를 풀고 테스트하였다.

각 테스트는 적중하였을 경우와 적중하지 않을 경우를 각각 테스트하였으며 비교 대상으로는 기업의 개발환경에 많이 사용되는 프레임워크인 SPRING 프레임워크와 MyBatis 프레임워크를 사용하였고 MyBatis의 캐시를 비교 값으로 사용하였다. 각각의 경우에 60회 요청을 가지고 응답 값의 크기를 30 byte, 1 MB, 2 MB로 구분하여 요청 시 와 응답 시의 시간차이를 비교하였다.

3-2 적중 시 결과 분석

1) 30초 동안 같은 요청 값으로 균등 조회

그림 9와 표 2에서와 같이 1 MB 조회 시 캐시 한 것과 안한 것의 차이가 나기 시작하며 시작 시부터 캐시 안정화된 전 구간에 걸쳐서 MyBatis 캐시에 비하여 NIO 캐시의 경우 약 3배 히트 구간에서는 약 7배의 성능향상을 보였다.

응답 값이 2 MB 일 경우는 그림 10과 표 3에서 보면 캐시 안한 것은 이전 요청의 오버헤드로 인하여 점점 응답시간이 늘어나고 있으며 기존의 MyBatis 캐시와 NIO의 응답시간에도 초기

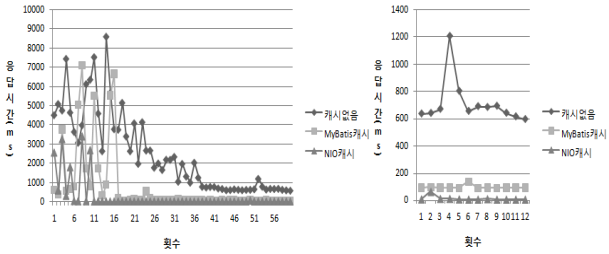

그림 9. 1 MB 응답 요청횟수별 응답시간 (좌:전체, 우:히트)
Fig. 9. 1 MB response times per request count (left: All, right: HIT).

표 2. 1 MB 응답시간
Table 2. 1 MB response time.

	sample count	Max(ms)	Min(ms)	Average(ms)
no cache	60	8609	597	2509.8
MyBatis cache	60	7109	94	799.4833
NIO cache	60	3410	7	254.65

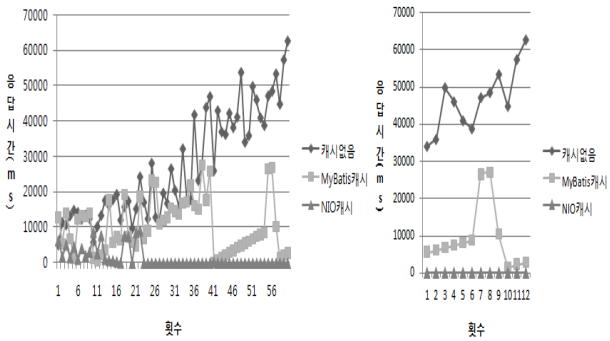

그림 10. 2 MB 응답 요청횟수별 응답시간 (좌:전체, 우:히트)
Fig. 10. 2 MB response times per request count (left: All, right: HIT).

표 3. 2 MB 응답시간
Table 3. 2 MB response time.

	sample count	Max(ms)	Min(ms)	Average(ms)
no cache	60	62727	5138	26938.8
MyBatis cache	60	27712	268	10530.68
NIO cache	60	9190	11	1444.05

MyBatis의 오버헤드로 인한 응답시간 증가와 더불어 전체 평균응답 시간에서는 약 7배의 성능 차이를 확인할 수 있었다.

2) 15 초 동안 같은 요청 값으로 균등 조회

응답 값이 1 MB 일 경우 그림 11와 표 4에서 보면 캐시 없음과 MyBatis 캐시는 오버헤드에 의하여 두 가지 모두 응답시간이 점점 증가하는 모습을 보이고 있는 반면에 NIO 캐시만이 안정화된 모습을 보이고 있다. 응답시간 차이는 전체 구간에서는 약 15배 차이를 보이고 있다.

응답 값이 2 MB 일 경우는 그림 12와 표 5에서와 같이 세 가지 모두 오버헤드에 의하여 응답시간이 점점 증가하는 모습을 보이고 있으며 그 중에도 NIO 캐시만이 약간의 응답시간 우세를 보이고 있다. 캐시 없음과 MyBatis 캐시는 거의 전 구간에서 같은 응답 그래프를 보이고 있다.

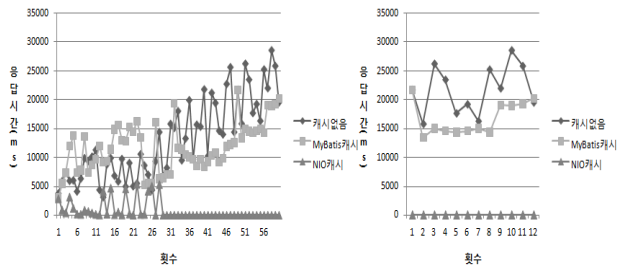

그림 11. 1 MB 응답 요청횟수별 응답시간 (좌:전체, 우:히트)
Fig. 11. 1 MB response times per request count (left:All, right:HIT).

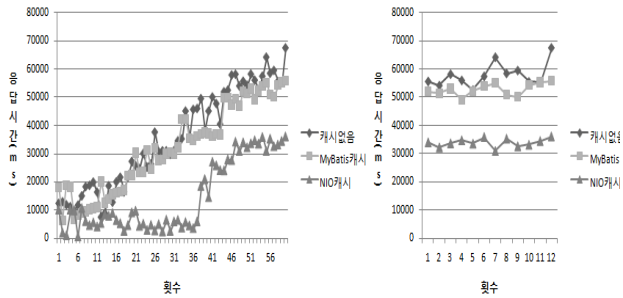

그림 12. 2 MB응답 요청횟수별 응답시간 (좌:전체, 우:히트)
Fig. 12. 2 MB response times per request count (left: All, right: HIT).

표 4. 1 MB 응답시간
Table 4. 1 MB response time.

	sample count	Max(ms)	Min(ms)	Average(ms)
no cache	60	28631	3110	13028.77
MyBatis cache	60	21760	3078	11672.98
NIO cache	60	5407	7	776.1167

표 5. 2 MB 응답시간
Table 5. 2 MB response time.

	sample count	Max(ms)	Min(ms)	Average(ms)
no cache	60	67450	7632	35364.08
MyBatis cache	60	55748	6574	32002.17
NIO cache	60	36183	573	14828.97

3-3 적중이 안 될 경우 결과 분석

데이터가 순환될 때의 성능 확인을 위하여 기존과 같은 방식이지만 매번 다른 요청을 60 번 조회한다. 부하 테스트를 위하여 30초와 15초 2가지 테스트를 응답 값 1 MB와 2 MB로 테스트하였다.

1) 30초 동안 같은 요청 값으로 균등 조회

그림 13에서의 우측 그래프가 1 MB 요청을 매번 다른 요청을 하여 자료순환이 이루어졌을 때의 그래프이며 좌측이 2 MB 요청일 시 그래프이다. 1 MB 요청의 경우 NIO 캐시는 안정적인 응답 속도를 보이고 있으며 표 6 에서와 같이 평균값과 최소값의 차이가 거의 균일하게 나오고 있다. 2 MB의 경우에는 두 모델 전부 오버헤드가 있지만 MyBatis에 비하여 좀 더 안정적인 모습을 보이고 있다. MyBatis 캐시에 비하여 2 MB일 경우는 약 1.4배, 1 MB의 경우는 2.2배 정도의 성능향상을 보이고 있다.

2) 15초 동안 같은 요청 값으로 균등 조회

그림 14과 표 7에서 보듯이 오버헤드에 의하여 점점 응답 시간이 증가하는 모습을 보이지만 NIO 캐시에서는 MyBatis에 비해서 약 1.1 배와 1.5 배의 응답 향상을 볼 수 있었다.

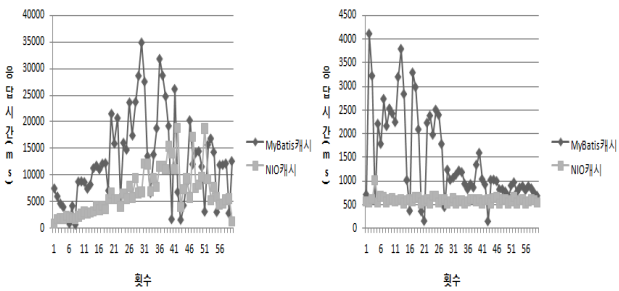

그림 13. 30초간 매번 다른 요청 시 요청횟수별 응답시간 (좌:2 MB, 우:1 MB)

Fig. 13. 30 seconds each time another request response time (left: All, right: HIT).

표 6. 30초간 매번 다른 요청 시 응답시간

Table 6. 30 seconds each time another request response time.

reply size	type	sample count	Max(ms)	Min(ms)	Average (ms)
2 MB	MyBatiscache	60	34989	685	12897.42
	NIO cache	60	16883	1472	8705.767
1 MB	MyBatis cache	60	4120	144	1475.767
	NIO cache	60	1728	488	658.9833

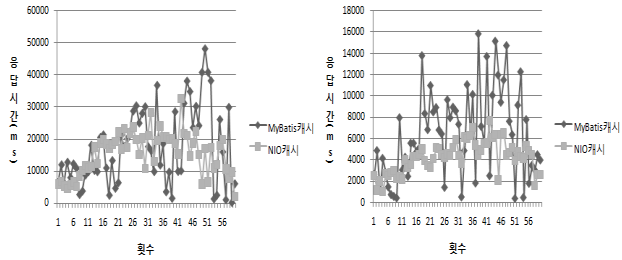

그림 14. 15초간 매번 다른 요청 시 요청횟수별 응답시간 (좌:2 MB, 우:1 MB)

Fig. 14. 30 seconds each time another request response time (left: All, right: HIT).

표 7. 15초간 매번 다른 요청 시 응답시간

Table 7. 15 seconds each time another request response time.

reply size		sample count	Max(ms)	Min(ms)	Average (ms)
2 MB	MyBatis cache	60	48305	185	17400.73
	NIO cache	60	32711	2191	15611.75
1 MB	MyBatis cache	60	15893	364	6433.85
	NIO cache	60	7721	1058	4278.533

Ⅳ. 결 론

본 논문에서는 기존 WAS에서 NIO를 이용한 범용 웹 애플리케이션 캐시 방식을 제시하여 응답 속도를 향상시켰다. 기존 자바 I/O의 블로킹과 가비지 컬렉션의 단점을 해소할 수 있도록 NIO를 사용하여 기존과 다른 방식으로 응답속도 향상을 시도 하였다. 캐시 우선순위 스케줄링 알고리즘은 작은 값을 여러 번 수정작업을 하므로 기존의 자바 메모리에서 처리하고, 내용 변동이 상대적으로 적고 데이터가 큰 DB 응답 값은 다이렉트 버퍼에 담는 방식의 하이브리드 메모리 캐시 방식을 적용하여 응답속도 및 자료 관리를 쉽게 하였다. 서버 등의 하드웨어 확장 없이 NIO를 이용하여 구현한 캐시만으로 웹 응답 속도를 향상할 수 있도록 하였다.

여러 종류의 테스트를 통하여 성능이 향상됨을 확인하였다. 기존의 현장에서 많이 쓰이는 프레임 워크인 SPRING+MyBatis 환경의 MyBatis 캐시에 비하여 적중 시 최대 약 2028배의 성능 향상을 볼 수 있었고 히트가 안 되는 자료 순환 시에도 약 2.2배의 성능향상을 볼 수 있었다.

본 연구 및 실험을 근거로 향후에 큰 데이터를 여러 사용자가 짧은 기간 요청했을 경우 나오는 오버헤드 상황을 해소할 수 있는 방안에 대한 연구가 필요하다.

참고 문헌

[1] Cisco Visual Networking Index. Global Mobile Data Traffic Forecast Update 2013-2018 White Paper: [Internet]. Available: https://www.cisco.com/web/KR/pdf/cisco_vni_fore cast_ qa.pdf

[2] Cisco Visual Networking Index:Global Mobile Data Traffic Forecast Update,2014-2019: [Internet]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.pdf

[3] T. Y. Kuo, Y. S. Chung, and J. Y. Park, "Cache layout management for reducing network traffic," in *Korea Information Science Society Conference*, Jeju: Korea, pp. 249-250, Jun.2012.

[4] H. Al-Zoubi, A. Milenkovic and M. Milenkovic, "Performance evaluation of cache replacement policies for the SPEC CPU2000 benchmark suite," in *Proceedings of the 42th Annual Southeast Regional Conference*, New York: NY, pp 267-272, 2004.

[5] C. Y. Chang, T. Mcgregor and G. Holmes, "The LRU *WWW proxy cache document replacement algorithm," in *Proceedings of the Asia Pacific Web Conference*, Hong Kong: China, 1999.

[6] P. R. Wilson, Uniprocessor Garbage Collection Techniques, Technical report, University of Texas, Jan 1994. Expanded version of the IWMM92 paper.

[7] P. Cheng, G. E. Blelloch, "A parallel, real-time garbage collector," in *Proceedings of the ACM SIGPLAN Conference on Programming Design and Implementation*, New York: NY, pp. 125-136, 2001.

이 철 희 (Chul-Hui Lee)
2016년 2월: 서울과학기술대학교 컴퓨터공학과 (공학석사)
현재: 프리랜서 개발자
※관심분야 : 알고리즘, 웹시스템 등

신 용 현 (Yong-Hyeon Shin)
2004년 2월 : 서울대학교 전기컴퓨터공학부 (공학박사)
2005년 3월 ~ 현재 : 서울과학기술대학교 컴퓨터공학과 교수
※관심분야 : 시스템소프트웨어, 웹시스템 등