Redis에서 AOF Rewrite 기법의 오버헤드 분석

진민화*, 박상현**

Overhead Analysis of AOF Rewrite Method in Redis

Min-Hwa Jin*, Sang-Hyun Park**

이 논문은 2015년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (NRF-2015R1A2A1A05001845).

요 약

최근 들어 많은 데이터의 실시간 처리 및 효율적인 비정형 데이터 관리를 위해 인메모리 기반의 오픈소스 키-밸류 저장소인 레디스를 주로 사용한다. 레디스는 데이터셋을 변경하는 모든 명령을 로그 파일에 기록하는 AOF(Append Only File)방식을 통해 데이터의 지속성을 보장하고, 더 나아가 AOF 파일의 크기가 계속해서 커지는 현상을 방지하기 위한 AOF Rewrite 기법을 제공한다. 하지만 AOF Rewrite 기법은 AOF Rewrite 버퍼의 사용으로 인해 메모리 사용량을 급격히 증가시키고 AOF Rewrite 버퍼의 내용을 디스크에 기록하는 과정에서 상당한 오버헤드를 유발하고 Out-Of-Memory 현상이 나타날 수 있다. 이러한 문제점에도 불구하고 기존 연구에서는 레디스의 성능을 분석하는 방향으로만 진행이 되었고, AOF Rewrite 기법으로 인한 오버헤드와 관련된 연구는 많이 부족하다. 따라서 본 논문에서는 레디스에서 AOF Rewrite 기법의 오버헤드를 구체적으로 분석한다. 실험 결과, 레디스에서 AOF Rewrite를 사용하는 경우는 AOF Rewrite를 사용하지 않은 경우에 비해 워크로드 수행을 위한 메모리 사용량이 최대 3배까지 증가하였고, 처리량은 49.9%까지 감소하는 것을 확인하였다.

Abstract

In recent years, in-memory based key-value store Redis is often used for real-time processing of large amounts of data and unstructured data processing. Redis guarantees the durability of the data through AOF (Append Only File) technique which records all commands that change the dataset in the log file. Furthermore, Redis provides a method called AOF Rewrite to prevent the AOF file from growing in size continuously. However, AOF Rewrite causes a significant increase in memory usage due to AOF Rewrite Buffer and throughput overhead in writing AOF Rewrite Buffer contents to disk and an out-of-memory may occur. In the existing research, it is only in the direction of analyzing the performance of the Redis and the research related to the overhead due to the AOF Rewrite technique is insufficient. In this paper, we analyze the overhead of the AOF Rewrite method. In our experimental results, In the case of using AOF Rewrite in Redis, the memory usage for the workload was increased up to 3 times and the throughput was reduced to 49.9% compared to the case without AOF Rewrite.

Keywords

Key-Value Store, In-Memory Database, Redis, Persistence, Memory Overhead

* 연세대학교 컴퓨터과학과
** 연세대학교 컴퓨터과학과 교수(교신저자)
· 접 수 일: 2017년 1월 16일
· 수정완료일: 2017년 2월 7일
· 게재확정일: 2017년 2월 9일

Received: Jan. 16, 2017, Revised: Feb. 7, 2017, Accepted: Feb. 9, 2017
Corresponding Author: Sang-Hyun Park
Dept. of Computer Science, Yonsei University, Sinchon-dong, Seodaemun-gu Seoul, 120-749, Korea
Tel.: +82-2-2123-5714, email: sanghyun@yonsei.ac.kr

Ⅰ. 서 론

최근 몇 년간 대규모 웹 서비스 및 IOT의 대중화로 인해 많은 데이터가 생산되었다. 따라서 많은 양의 데이터를 관리하고 처리하기 위한 수요가 급증하였고, 실시간으로 많은 양의 데이터 분석이 필요하게 되었다. 하지만 기존의 데이터를 저장하고 관리하는 대표적인 시스템인 관계형 데이터베이스(RDBMS)는 많은 데이터를 다루기에 부적합하다. 관계형 데이터베이스는 데이터의 무결성과 중복제거를 위하여 릴레이션 구조로 저장하기 때문에 많은 데이터를 다수의 서버로 분산하여 저장하는 스케일-아웃(Scale-Out) 기법을 적용하는데 제약이 따른다. 게다가 관계형 데이터베이스에서 데이터를 저장하기 위해 사용하는 릴레이션 구조는 웹 데이터, 미디어 데이터, GPS 데이터 등과 같은 비정형 데이터들을 효율적으로 다루기 힘들다.

따라서 스케일-아웃의 용이함과 효율적인 비정형 데이터들의 관리를 위한 데이터베이스들이 등장하게 되었다. 이 중 가장 대표적인 데이터베이스는 키-밸류 저장소(Key-Value Store)이다[1]-[3]. 특히, 비휘발성 저장장치의 입출력 지연시간을 제거하고 실시간 처리를 위한 성능을 발전시키기 위하여 Redis, Memcached, RAMCloud 등과 같은 메인메모리에 데이터를 저장하고 관리하는 인메모리 기반의 키-밸류 저장소가 등장하였다[4]-[7]. 이 중 Redis(Remote Dictionary Server)는 대표적인 인메모리 기반의 오픈소스 키-밸류 저장소로 디스크가 아닌 메인메모리에 데이터를 저장하고 관리하기 때문에 접근 속도가 상당히 빠르다. 그리고 클러스터 기능과 마스터-슬레이브 복제 기능을 제공하여 실시간으로 특정 노드의 데이터셋을 다른 노드에 복제할 수 있다. 또한, 레디스에서 사용하는 밸류의 저장방식은 단순 문자열 뿐 아니라 리스트, 해시테이블, 세트, 비트맵 등과 같은 다양한 자료구조를 제공하여 비정형 데이터 관리에 매우 적합하다.

레디스와 같은 인메모리 기반의 키-밸류 저장소는 데이터를 저장하고 관리할 때 메인 메모리를 사용한다. 메인 메모리는 디스크와 달리 휘발성 저장장치이기 때문에 인메모리 기반의 키-밸류 저장소는 전원 고장(Power Failure)과 같은 상황에서 데이터를 지켜야 한다. 이를 위해 인메모리 기반의 키-밸류 저장소는 현재 메모리에 존재하는 데이터를 디스크에 저장하는 영속성(Persistence) 기법을 이용하여 지속성(Durability)을 보장한다. 레디스는 지속성을 보장하기 위해 RDB(Redis Database)와 AOF(Append Only File)라는 두 가지의 영속성 기법을 제공한다.

RDB는 특정 시점의 데이터에 대한 바이너리 형태의 스냅샷을 생성하여 디스크에 기록하는 방식이다. 따라서 RDB는 AOF에 비해 파일의 크기가 작고 복구 속도도 빠르다. 하지만 RDB는 데이터의 변경을 모두 반영한 것이 아니라 특정 시점의 데이터셋을 저장한다. 따라서 RDB 생성 직후의 데이터 변경에 대한 지속성을 보장하지 못한다.

AOF는 레디스 내에 존재하는 데이터셋을 변경하는 모든 명령을 로그 레코드 형태로 AOF 파일에 기록하는 방식이다. AOF는 데이터셋을 변경하는 모든 명령을 기록하기 때문에 특정 시점의 데이터셋을 저장하는 RDB에 비해 지속성이 더 높다. 따라서 레디스를 인메모리 캐시가 아닌 데이터 저장소로 사용할 경우에는 주로 AOF를 사용한다. 하지만 AOF를 계속해서 사용한다면 데이터의 모든 변경을 로그 형태로 기록하기 때문에 AOF 파일의 크기가 계속 커진다. 이는 AOF 파일의 크기가 저장장치의 용량을 넘어서 AOF 기능이 중단 될 수 있고, AOF 파일을 이용하여 복구를 수행하는 시간이 오래 걸릴 수 있는 문제를 유발할 수 있다. 따라서 레디스는 AOF 파일이 계속해서 커지는 현상을 방지하기 위해 기존 AOF 파일의 크기보다 작은 새로운 AOF 파일을 생성하는 AOF Rewrite 기법을 제공한다. AOF Rewrite 기법은 현재 데이터셋에 대한 이전 기록들을 모두 없애고 최종 데이터에 대한 로그 레코드만을 생성한다. 따라서 AOF Rewrite 기법은 기존 AOF 파일의 크기보다 작은 새로운 AOF 파일을 생성하게 된다.

그러나 AOF Rewrite 기법은 AOF를 이용하여 영속성을 보장할 때 필수적으로 사용해야 하는 기능이지만 동작 방식은 메모리의 큰 부하와 성능 저하를 일으킬 가능성이 있다. 이에 본 논문은 앞서 소개한 AOF Rewrite 기법을 메모리 사용량과 실시간 처리량의 관점에서 실험에 대한 결과를 분석한다.

실험 및 분석 결과, 본 논문에서 발견한 AOF Rewrite 기법의 문제점은 크게 두 가지가 있다.

첫째, AOF Rewrite 수행 도중에 클라이언트로부터 요청받은 명령에 대한 로그 레코드를 일시적으로 저장하는 AOF Rewrite 버퍼의 사용량이 급격히 증가하게 된다. 이는 메모리 용량이 제한적인 환경에서 Out-Of-Memory 현상으로 인하여 레디스가 비정상적으로 종료되고, 그로 인해 데이터의 지속성을 보장하지 못할 수 있다.

둘째, 레디스는 AOF Rewrite를 통해 새로운 AOF 파일 생성을 완료한 후, 새로운 AOF 파일에 AOF Rewrite 버퍼에 저장된 로그 레코드들을 기록하는 동안 다른 작업을 할 수 없는 상태(Blocking State)가 된다. 따라서 레디스는 AOF Rewrite 버퍼에 존재하는 로그 레코드를 기록하는 동안 클라이언트로부터 요청받은 명령을 처리할 수 없으므로 전반적인 성능이 감소한다. 따라서 본 논문에서는 위에서 제시한 두 가지의 문제점을 여러 실험을 통하여 분석한다.

본 논문의 나머지 구성은 다음과 같다. 2장은 관련연구에 대해 설명하며 3장에서는 본 논문에서 AOF Rewrite 기법의 동작방식에 대해 자세하게 기술한다. 4장에서는 AOF Rewrite의 문제점을 확인하기 위해 실험을 수행하고 메모리 사용량과 처리량의 관점에서 분석한다. 마지막으로 5장에서는 결론에 관해 기술하고 본 연구에 대한 향후 연구에 대하여 기술한다.

II. 관련 연구

인메모리 기반의 데이터 저장소의 수요가 급증하면서 이에 따라 인메모리 기반 데이터 저장소의 영속성에 관한 연구가 진행되어왔다[8]-[10]. 특히 레디스에서 제공하는 영속성 기법을 사용했을 때의 전반적인 시스템 성능을 분석한 여러 연구가 진행되었다[11]-[12].

[11]은 레디스에서 RDB Only, AOF-Rewrite, AOF-NoRewrite 모드에서 다양한 워크로드를 이용하여 복구 속도와 각 파일의 크기 등을 분석하였다. 실험 결과, AOF Rewrite를 수행하였을 때 로그 레코드의 수가 줄어들기 때문에 같은 데이터셋에 대해서 복구 시간을 줄일 수 있음을 보였다. 또한 실험 데이터셋의 크기의 따른 RDB와 AOF 파일의 크기를 분석한 결과, 같은 데이터셋에 대해 RDB의 파일 크기가 AOF 파일 크기보다 작게 측정되었다. 텍스트 형태로 로그 레코드를 기록하는 AOF 파일과 달리 RDB 파일은 LZF Compression 기법을 적용하여 데이터셋을 바이너리 형태로 저장하기 때문이다. RDB Only, AOF-Rewrite, AOF-NoRewrite 모드에서 워크로드 수행 완료 후 생성된 RDB, AOF 파일의 크기와 RDB와 AOF의 복구 속도를 비교한 [11]과 달리 본 논문은 AOF Rewrite 기법이 동작하는 순서에 따른 메모리 사용량과 처리량의 변화를 다양한 워크로드를 이용하여 구체적인 분석을 보인 점에서 그 의미가 있다.

그리고 [12]는 레디스와 멤캐시드의 전반적인 성능 비교를 보였다. 해당 논문에서는 큰 단위의 워크로드와 작은 단위의 워크로드 두 가지로 나누어 실험을 진행하여 레디스와 멤캐시드의 CPU 및 메모리 사용량과 수행 속도를 분석하였다. 그리고 레디스에서 AOF Rewrite 수행 시 발생할 수 있는 문제점을 간략하게 언급하였다. 추가로 레디스에서 제공하는 여러 가지 자료구조에 대한 메모리 오버헤드를 분석하였다.

즉, 지금까지의 연구에서는 AOF Rewrite로 인한 메모리 사용량의 변화와 AOF Rewrite를 수행하는 동안의 처리량에 대한 연구는 미미한 실정이다. 게다가 레디스에서 AOF Rewrite의 문제점을 해결하기 위한 구체적인 연구는 아직 알려진 바가 없다. 따라서 본 논문에서는 AOF Rewrite를 수행하는 동안의 메모리 사용량 및 처리량을 분석하고 더 나아가 레디스에서 AOF Rewrite가 발생하기 위한 AOF 파일의 최소 크기(auto-aof-rewrite-min-size) 옵션에 따른 처리량과 최대 메모리 사용량의 영향을 확인한다.

III. AOF Rewrite 동작과정

레디스에서 AOF를 사용하는 경우에 AOF 파일의 크기가 계속해서 커진다면 저장장치의 용량을 넘어서 AOF 기능이 중단되거나 AOF 파일을 이용하여 복구하는데 시간이 오래 걸리는 문제가 발생할 수 있다. 따라서 이러한 문제들을 해결하기 위하여 기

존의 AOF 파일보다 작은 크기의 새로운 AOF 파일을 생성하는 AOF Rewrite 기능을 제공한다. AOF Rewrite는 AOF 파일이 일정 크기 이상으로 커지게 되면 현재 데이터셋을 재구성 할 수 있는 로그 레코드들의 시퀀스를 생성하여 새로운 AOF 파일에 기록하는 과정을 거쳐 기존의 AOF 파일을 대체하는 기법이다. AOF Rewrite의 구체적인 동작 방식은 다음과 같다.

AOF 방식은 버퍼 공간을 활용하여 로그 레코드를 생성한다. 클라이언트에서 레디스에 요청을 보내면, 우선 요청에 대한 로그 레코드를 먼저 AOF 버퍼라는 공간에 저장한다. 그리고 설정 파일에서 설정한 주기에 따라 AOF 버퍼의 저장된 로그 레코드들을 AOF 파일에 기록한다. 기본적으로 1초에 한 번씩 AOF 버퍼의 저장된 로그 레코드들을 AOF 파일에 기록한다.

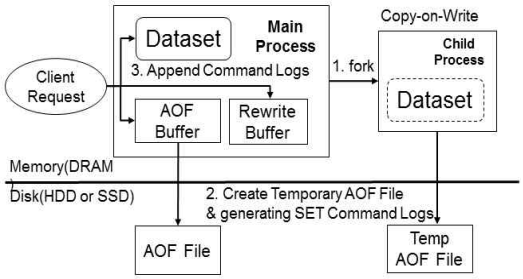


그림 1. AOF Rewrite 시작 및 자식 프로세스 생성
Fig. 1. AOF Rewrite starts and produces a child process

부모 프로세스는 AOF 파일의 크기를 지속해서 모니터링을 하고, 만약 현재 AOF 파일의 크기가 AOF Rewrite 가 발생하기 위한 AOF 파일의 최소 크기(auto-aof-rewrite-min-size) 이상이 된다면 그림1과 같이 Fork[13] 시스템 콜을 호출하여 자식 프로세스를 생성한다. 자식 프로세스는 새로운 AOF 파일을 생성하고 Copy-On-Write[14] 기법을 이용하여 현재 데이터셋에 대한 로그 레코드들의 시퀀스를 만들어 새로 만든 AOF 파일에 기록하는 작업을 수행하게 된다. 따라서 메인 프로세스는 자식 프로세스와 별도로 클라이언트로부터 받은 요청을 처리할 수 있게 된다.

그리고 Fork 시스템 콜에 의해 자식 프로세스가 생성된 시점부터, 클라이언트로부터 받은 명령에 대한 로그 레코드는 AOF Rewrite 버퍼라는 임시 메모리 공간과 기존의 AOF 파일에 동시에 기록을 한다. 이는 AOF Rewrite 수행 도중에 전원 고장이 발생했을 경우에 기존의 AOF 파일을 이용하여 데이터셋을 복구하기 위함이다.

자식 프로세스는 새로운 AOF 파일에 로그 레코드들의 시퀀스를 전부 기록하게 되면 그림 2와 같이 부모 프로세스에게 종료 시그널을 보낸다.

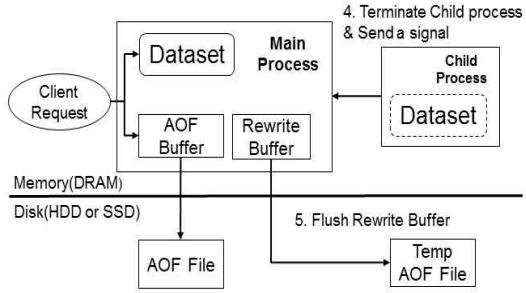


그림 2. 자식 프로세스에서 AOF 파일 생성 완료 후 AOF Rewrite 버퍼 내 로그 레코드를 새로운 AOF 파일에 기록하는 과정
Fig. 2. After a sequence of command logs are created, the Flush AOF Rewrite Buffer to the Temporary AOF file

종료 시그널을 받은 부모 프로세스는 AOF Rewrite 버퍼 안에 있는 로그 레코드들을 새로운 AOF 파일에 기록한다. 마지막으로 새로운 AOF 파일의 이름을 기존의 AOF 파일의 이름으로 변경하는 방식으로 기존의 AOF 파일을 대체하면서 AOF Rewrite 과정이 끝난다.

Ⅳ. 실험 및 결과 분석

본 장에서는 여러 워크로드를 통해 레디스에서 AOF Rewrite 기능을 사용했을 때 메모리 사용량 및 처리 성능을 비교한다.

4.1 Memtier-Benchmark

본 논문에서 실험을 위해 사용한 벤치마킹 시스템은 RedisLabs에서 개발한 Memtier-Benchmark[15]이다. Memtier-Benchmark는 레디스 뿐만 아니라 멤캐

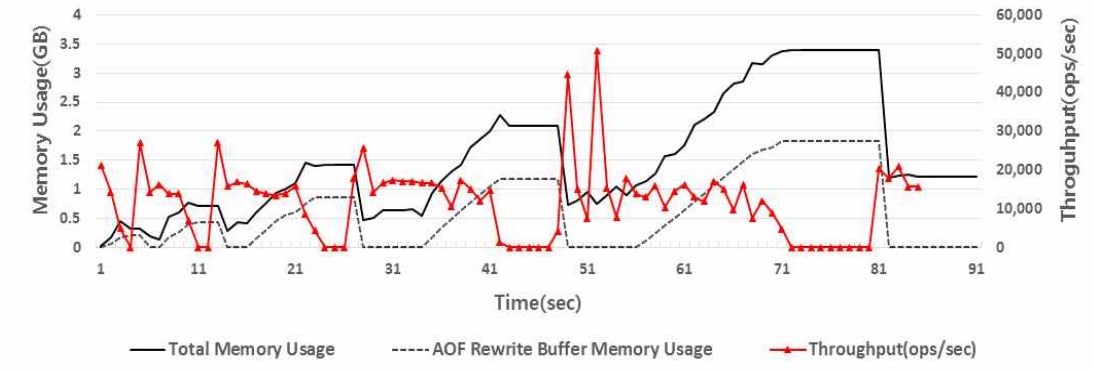


그림 3. 자식 프로세스에서 AOF 파일 생성 완료 후
AOF Rewrite 버퍼 내 로그 레코드를 새로운 AOF 파일에 기록하는 과정
Fig. 3. After a sequence of command logs are created, the Flush AOF Rewrite Buffer to the Temporary AOF file

시드 에 대해서도 워크로드를 생성하여 실험할 수 있고, 다양한 워크로드 구성을 위해 키 또는 밸류의 크기, 클라이언트 개수와 같은 여러 가지 변수를 변경하여 워크로드를 구성할 수 있다. 또한 워크로드를 수행하는 동안의 초 당 처리하는 명령의 수(ops/sec), Latency, 초 당 처리하는 데이터의 크기(KB/sec) 등을 제공한다. 본 논문에서는 1.2.7 버전의 Memtier-Benchmark를 사용하였다.

4.2 실험 환경

본 실험을 위해 Quad-Core 3.4 GHz Intel i7-3770 프로세서와 메모리 장치는 DDR3 64GB의 DRAM, 비휘발성 저장 장치로는 HDD 1TB WDC WD10EZEX-22RKKA0 으로 구성하였다. 실험을 위해 사용한 운영체제는 CentOS 6.6 버전, 리눅스 커널 버전은 2.6.32-504.el6.x86_64을 사용하였다. 그리고 스케일-아웃을 위해 레디스를 클러스터 모드로 구성하는 경우에도, 각 노드에 대한 내부 로깅 방식은 동일하며 독립적으로 작용하기 때문에 다른 노드들의 로깅에 영향을 주지 않는다. 따라서 단일 노드로 구성된 3.0.7 Standalone 버전 레디스를 이용하여 실험을 수행하였다. 그리고 레디스가 사용할 수 있는 최대 메모리 사용량은 30GB로 설정하였고 AOF Rewrite가 발생하기 위한 AOF 파일의 최소 크기는 64MB, AOF 버퍼를 디스크에 기록하는 주기는 1초로 설정했다. 마지막으로 Memtier-Benchmark에서 제공하는 초 당 처리하는 명령의 수(ops/sec)의 정보와, 레디스에서 제공하는 redis-cli에서 info 명령어를 이용하여 워크로드를 수행하는 동안의 메모리 사용량을 측정하였다.

4.3 단일 워크로드에서의 메모리 사용량 및 처리량 측정

먼저 AOF Rewrite를 발생시키는 워크로드를 이용하여 벤치마크를 수행하는 동안의 메모리 사용량과 처리량을 초 단위로 측정하였다. 그리고 일반 문자열 형태로 데이터를 저장하는 SET 명령에 대한 로그 레코드의 크기와 LPUSH와 HSET 등과 같이 다른 자료구조에 대한 삽입 연산으로 생성된 로그 레코드의 크기의 차이가 없다. 따라서 다른 자료구조들에 대한 삽입 명령으로 생성된 로그 레코드는 AOF Rewrite 버퍼의 메모리 사용량에는 영향이 없어 SET 명령에 대한 실험만을 수행하였다. 해당 실험에서는 전체 메모리 사용량 대비 AOF Rewrite 버퍼로 사용하는 메모리의 양을 측정하기 위해 100,000개의 SET 명령과 중복된 900,000개의 SET(UPDATE) 명령으로 구성된 워크로드를 구성하였다. 해당 워크로드에서 사용한 키의 크기는 16B, 밸류의 크기는 10KB로 구성하였고, 워크로드의 전체 데이터셋 크기는 1.18GB다. 워크로드를 수행하는 동안의 메모리 사용량과 처리량을 측정한 결과

는 다음과 같다.

실험 결과, 그림 3에서와 같이 해당 워크로드를 수행하는 동안 총 5번(2-5s, 7-13s, 17-27s, 35-48s, 57-81s)의 AOF Rewrite가 발생하였다. 자식 프로세스에서 새로운 AOF 파일에 로그 레코드들의 시퀀스를 기록 동안, 클라이언트로부터 요청받아 생성된 로그 레코드들은 AOF Rewrite 버퍼에 저장된다. 이때 AOF Rewrite 버퍼의 메모리 사용량과 레디스가 사용하는 전체 메모리 사용량이 비례하여 증가하는 것을 확인할 수 있다. 이는 AOF Rewrite 버퍼의 사용으로 인해 전체 메모리 사용량이 증가함을 나타낸다.

그리고 메모리의 사용량이 증가하는 동시에 처리량이 감소하게 된다. 왜냐하면, 자식 프로세스가 현재 데이터셋을 구성할 수 있는 로그 레코드들의 시퀀스를 생성하는 과정에서 CPU 사용량이 급격히 증가하기 때문이다.

자식 프로세스의 작업이 끝난 이후 AOF Rewrite 버퍼의 메모리 사용량이 증가하지 않고 처리량이 일정하게 0인 구간이 발생한다. 이러한 현상은 레디스가 단일 스레드 기반으로 동작하기 때문에 부모 프로세스가 AOF Rewrite 버퍼에 저장된 로그 레코드들을 새로운 AOF 파일에 기록하는 동안 다른 작업을 할 수 없게 되는 것이다. 따라서 이 작업을 하는 동안에는 클라이언트로부터의 요청을 처리할 수 없어 처리량이 0이 된다.

마지막으로 AOF Rewrite 버퍼에 저장된 로그 레코드들을 새로운 AOF 파일에 기록하는 작업을 마치게 되면 다시 처리량이 증가하게 된다. 동시에 AOF Rewrite 버퍼로 할당된 메모리를 해제하게 되면서 AOF Rewrite 버퍼의 메모리 사용량과 전체 메모리 사용량이 동시에 감소하게 된다.

해당 실험결과는 레디스에서 1.18GB의 데이터셋을 저장하기 위한 시스템에서는 데이터셋 크기의 3배 정도인 3.5GB의 메모리 공간이 필요하다는 것을 확인할 수 있다.

4.4 여러 워크로드에 대한 최대 메모리 사용량 측정

해당 실험에서는 레디스에서 AOF Rewrite Mode / No Rewrite Mode로 나누어 각 워크로드를 수행했을 때의 메모리 최대 사용량을 측정하였다. SET 명령어에서 사용하는 밸류의 크기와 SET 명령어의 개수를 조절하여 워크로드를 구성하였다.

실험 결과, 전반적으로 AOF Rewrite Mode가 No Rewrite Mode보다 최대 메모리 사용량이 많다는 것을 확인할 수 있다. 이는 자식 프로세스에서 Fork

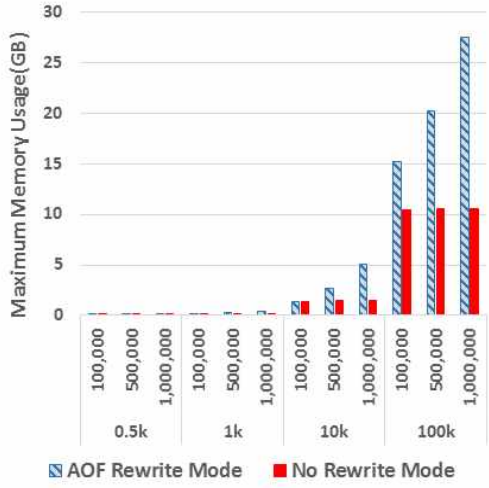


그림 4. 밸류의 크기 및 명령어 수에 따른 최대 메모리 사용량

Fig. 4. Maximum memory usage of various number of requests and value sizes

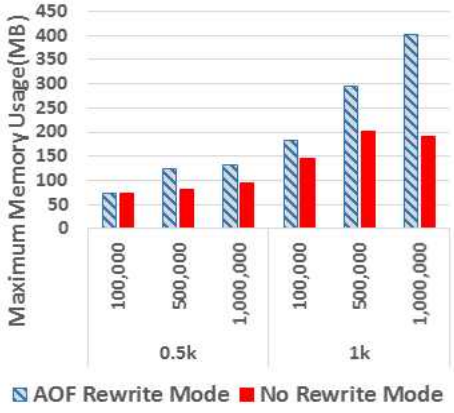


그림 5. 밸류의 크기 및 명령어 수에 따른 최대 메모리 사용량(0.5KB, 1KB)

Fig. 5. Maximum memory usage of various number of requests and value sizes(0.5KB, 1KB)

표 1. 전체 실험 결과
Table 1. Overall experimental result

Parameters		No Rewrite Mode		AOF Rewrite Mode		처리량 증감 비율	최대 메모리 사용량 증감 비율
SET 명령어 개수	밸류 크기	처리량 (ops/sec)	최대 메모리 사용량(KB)	처리량 (ops/sec)	최대 메모리 사용량(KB)		
100,000	0.5k	31558	72091	31550	72091	1.000x	1.000x
100,000	1k	32642	146538	31270	183402	0.958x	1.252x
100,000	10k	15624	1376530	12665	1392347	0.811x	1.011x
100,000	100k	1586	10443130	792	15236030	0.499x	1.459x
500,000	0.5k	86090	81356	67841	124848	0.788x	1.535x
500,000	1k	83184	201563	59523	294220	0.716x	1.460x
500,000	10k	14888	1453232	14088	2689370	0.946x	1.851x
500,000	100k	1389	10560519	879	20238682	0.633x	1.916x
1,000,000	0.5k	91417	95013	79296	131988	0.867x	1.389x
1,000,000	1k	83484	192065	69568	400962	0.833x	2.088x
1,000,000	10k	15063	1477662	13409	5030355	0.890x	3.404x
1,000,000	100k	1685	10604854	958	27466284	0.568x	2.590x

시스템 함수 호출 시점에 대한 데이터셋을 재구성할 수 있는 로그 레코드들을 새로운 AOF 파일에 기록하는 동안 부모 프로세스에서 AOF Rewrite 버퍼에 로그 레코드들을 저장하기 때문으로 판단된다. 특히 본 논문의 실험 조건일 경우에 한해 밸류의 크기가 10KB, SET 명령어의 개수가 1,000,000개로 구성된 워크로드에서는 NO Rewrite Mode 대비 AOF Rewrite Mode에서의 최대 메모리 사용량이 3.4배까지 늘어나는 것을 확인할 수 있다. 또한 밸류의 크기가 100KB, SET 명령어의 개수가 1,000,000개인 경우에는 최대 메모리 사용량으로 설정한 30GB와 근접한 것을 알 수 있다. 따라서 현재 실험을 위해 구성한 시스템에서는, 위에서 제시한 경우보다 밸류의 크기나 명령어의 개수를 더 늘리게 된다면 Out-Of-Memory 현상을 유발할 수 있다.

그리고 밸류의 크기가 0.5KB, SET 명령어의 개수가 100,000개인 경우에는 AOF Rewrite Mode와 No Rewrite Mode의 최대 메모리 사용량이 같은 것을 알 수 있다. 이는 해당 워크로드를 전부 수행했을 때의 AOF 파일의 크기가 AOF Rewrite가 발생하는 최소 파일 크기인 64MB보다 작아 AOF Rewrite가 한 번도 발생하지 않았기 때문에 메모리 사용량이 같은 것으로 나타났다.

4.5 여러 워크로드에 대한 평균 처리량 측정

본 절에서는 AOF Rewrite Mode / No Rewrite Mode로 나누어 여러 워크로드에 대한 처리량을 측정하였다. 실험에서 사용한 워크로드는 4.4절에서 사용한 워크로드와 같다.

실험 결과, 그림 6과 같이 대부분의 워크로드에서 AOF Rewrite Mode가 NO Rewrite Mode보다 처리량이 낮다는 것을 확인할 수 있다. 특히 표1에서 밸류의 크기가 100KB, SET 명령어의 개수가 100,000개로 구성된 워크로드에서는 평균 처리량이 No Rewrite Mode에 비해 49%까지 감소하였다. 이는 AOF Rewrite 버퍼에 저장된 로그 레코드들을 새로운 AOF 파일에 기록하는 과정에서 발생하는 오버헤드가 발생하였기 때문이다.

마찬가지로 밸류의 크기가 0.5KB, SET 명령어의 개수가 100,000개로 구성된 워크로드에서는 AOF Rewrite가 발생하지 않기 때문에 처리량이 같음을

확인할 수 있다.

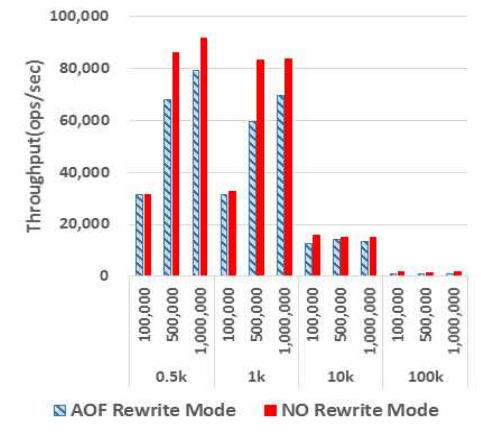


그림 6. 밸류의 크기 및 명령어 수에 따른 처리량
Fig. 6. Throughput of various number of requests and value sizes

4.6 AOF Rewrite가 발생하기 위한 AOF 파일의 최소 크기에 대한 워크로드 수행 결과

본 절에서는 AOF Rewrite가 발생하기 위한 AOF 파일의 최소 크기(auto-aof-rewrite-min-size) 옵션에 따른 처리량과 최대 메모리 사용량의 영향을 확인하기 위하여 밸류의 크기가 작은 워크로드와 큰 워크로드로 나누어 실험을 수행하였다.

표 2. AOF Rewrite 발생 최소 AOF 파일 크기에 따른 처리량 및 최대 메모리 사용량(밸류의 크기 : 1KB)
Table 2. Throughput and maximum memory usage of various number of auto-aof-rewrite-min-size(Value size : 1KB)

AOF Rewrite 발생 최소 AOF 파일 크기	처리량 (ops/sec)	최대 메모리 사용량(KB)
32MB	68631	385491
64MB	69568	400962
128MB	70319	392764
256MB	71025	472967
512MB	74230	501512

먼저 밸류의 크기가 1KB, SET 명령어의 개수가 1,000,000개로 구성된 워크로드를 이용하여 실험을 수행하였다. 표2와 같이 같은 워크로드에서 auto-aof-rewrite-min-size의 값이 증가하면 처리량과 최대 메모리 사용량이 동시에 증가하는 것을 확인할 수 있다. 이는 auto-aof-rewrite-min-size의 값이 크면, AOF Rewrite가 발생하는 횟수가 줄어들기 때문에 전체 처리량이 증가하기 때문이다. 그러나 auto-aof-rewrite-min-size의 값이 증가하면 새로운 AOF파일을 생성하기 위한 데이터셋의 크기도 동시에 증가한다. 따라서 새로운 AOF파일을 생성하는 시간이 늘어나게 되어 AOF Rewrite 버퍼에 저장되는 로그 레코드의 수가 많아지게 되므로 최대 메모리의 사용량이 증가하게 된다. 그러나 밸류의 크기가 작아서 로그 레코드의 크기 또한 작다. 이로 인해 전체 시스템에 영향을 줄 수 있는 정도로 메모리의 사용량이 증가하지 않는다. 게다가 작은 용량의 AOF Rewrite 버퍼를 디스크에 기록하는 오버헤드가 밸류가 큰 워크로드에 비해 작기 때문에 전체 처리량이 증가하게 된다. 따라서 밸류의 크기가 작은 워크로드에서는 auto-aof-rewrite-min-size의 값을 크게 설정하는 것이 유리하다.

그리고 밸류의 크기가 100KB, SET 명령어의 개수가 1,000,000개로 구성된 데이터셋의 크기가 10GB인 워크로드에 대해서 실험을 수행하였다. 그리고 레디스가 사용할 수 있는 최대 메모리 사용량을 30GB로 설정하였을 때 Out-Of-Memory 현상으로 인해 레디스가 비정상적으로 종료되는 경우가 발생하여 최대 메모리 사용량을 40GB로 변경하여 실험을 수행하였다.

표 3. AOF Rewrite 발생 최소 AOF 파일 크기에 따른 처리량 및 최대 메모리 사용량(밸류의 크기 : 100KB)
Table 3. Throughput and maximum memory usage of various number of auto-aof-rewrite-min-size(Value size : 100KB)

AOF Rewrite 발생 최소 AOF 파일 크기	처리량 (ops/sec)	최대 메모리 사용량(KB)
32MB	940	28029513
64MB	958	27466284
128MB	891	35710819
256MB	887	37881956
512MB	871	40436284

실험 결과, 표3에서와 같이 큰 워크로드를 수행하였을 때, auto-aof-rewrite-min-size의 값을 증가시키면 처리량이 감소하고 최대 메모리 사용량이 증가

한다. 앞 실험과 마찬가지로 auto-aof-rewrite-min-size의 값이 증가하면 새로운 AOF 파일을 생성하기 위한 데이터셋의 크기가 증가한다. 게다가 밸류의 크기가 큰 워크로드에서는 로그 레코드의 크기가 밸류의 크기와 비례하여 증가하기 때문에 AOF Rewrite 버퍼로 인한 최대 메모리 사용량이 급격히 증가한다. 마지막으로 많은 양의 메모리를 사용하는 AOF Rewrite 버퍼의 내용을 디스크에 기록하는 과정에서 상당한 크기의 오버헤드가 발생하여 AOF Rewrite가 발생하는 횟수가 감소하더라도 전체적인 처리량이 감소한다. 따라서 실험을 통하여 밸류의 크기가 100KB로 구성된 10GB의 쓰기 연산 워크로드를 수행하는 시스템에서는 auto-aof-rewrite-min-size의 값에 따라 메모리의 용량을 26GB 에서 최대 40GB 가까이 필요하다는 것을 알 수 있다.

따라서 밸류의 크기가 작은 워크로드를 수행하는 시스템에서는 auto-aof-rewrite-min-size의 값을 크게 설정하는 것이 처리량의 관점에서 유리하고, 밸류의 크기가 큰 워크로드를 수행하는 시스템을 운영하는 경우에는 auto-aof-rewrite-min-size의 값을 작게 설정하는 것이 메모리 사용량 관점에서 효율적이다.

Ⅴ. 결론 및 향후 과제

본 논문은 레디스에서 AOF Rewrite가 발생하는 워크로드를 수행했을 때 레디스의 전체 성능 및 메모리 사용량에 어떠한 영향을 미치는지 실험을 통해 분석하였다.

실험 결과, 같은 워크로드에서 AOF Rewrite를 사용했을 때 AOF Rewrite를 사용하지 않은 경우에 비해 메모리 사용량이 최대 3배까지 증가하는 것을 확인하였다. 이에 대한 원인은 AOF Rewrite를 수행하면서 새로운 AOF 파일을 생성하는 동안, 클라이언트로부터 받은 요청에 대한 로그 레코드를 AOF Rewrite 버퍼에 저장하는 과정에서 메모리 사용량이 급격히 증가하기 때문이다. 그리고 처리량은 AOF Rewrite 기능을 쓰지 않는 경우보다 49.9%까지 감소하는 것을 실험을 통하여 확인 할 수 있었다. 처리량이 감소한 원인은 AOF Rewrite 버퍼에 저장된 로그 레코드들을 새로운 AOF 파일에 기록하는 동안 다른 작업을 할 수 없기 때문이다.

그리고 더 나아가 밸류의 크기가 작은 워크로드를 수행하는 시스템에서는 AOF Rewrite가 발생하기 위한 AOF 파일의 최소 크기를 크게 설정하는 것이 처리량 측면에서 효율적이고, 밸류의 크기가 큰 워크로드를 수행하는 경우에는 AOF Rewrite가 발생하기 위한 AOF 파일의 최소 크기를 작게 설정하는 것이 메모리 사용량 측면에서 유리하다는 점을 실험을 통해 확인할 수 있었다.

하지만 본 논문에서 제시한 AOF Rewrite가 발생하기 위한 AOF 파일의 최소 크기 설정은 AOF Rewrite로 인한 본질적인 성능 저하문제를 해결하지 않았다. 따라서 AOF Rewrite의 한계점을 극복하기 위한 연구를 본 논문의 향후 과제로 한다.

References

[1] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., ... and Gruber, R. E. “Bigtable: A distributed storage system for structured data” ACM Transactions on Computer Systems (TOCS), Vol. 26, No. 2, June, 2008.

[2] Lim, H., Fan, B., Andersen, D. G., and Kaminsky, M. “SILT: A memory-efficient, high-performance key-value store” In Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles pp. 1-13, October, 2011.

[3] Lakshman, A., and Malik, “Cassandra: a decentralized structured storage system.” ACM SIGOPS Operating Systems Review, Vol. 44, No. 2, pp. 35-40. April, 2010.

[4] Redis https://redis.io [Accessed: Jan. 03, 2017].

[5] Memcached https://memcached.org [Accessed: Jan. 04, 2017].

[6] RAMCloud http://web.stanford.edu/~ouster/cgi-bin/projects.php [Accessed: Jan. 04, 2017].

[7] Lim, H., Han, D., Andersen, D. G., and Kaminsky, M. MICA: a holistic approach to fast

in-memory key-value storage. In 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14) pp. 429-444, April, 2014.

[8] Hee-Wan Kim, Yeon S. Ahn. "Application Performance Evaluation in Main Memory Database System." Journal of Digital Contents Society, Vol. 15, No. 5, pp. 631-642, October, 2014.

[9] Kwang-Keun Lee, Sung-Je Cho. “Mobile Main Memory Database Recovery System Improvement using EHPLD.” Journal of KIIT, Vol. 9, No. 12, pp. 73-82, December, 2011.

[10] Garcia-Molina, Hector, and Kenneth Salem. "Main memory database systems: An overview." IEEE Transactions on knowledge and data engineering Vol. 4, No. 6, pp. 509-516, August, 2002.

[11] Cao, W., Sahin, S., Liu, L., and Bao, X. “Evaluation and Analysis of In-Memory Key-Value Systems”. IEEE International Congress on Big Data (BigData Congress) pp. 26-33, July, 2016.

[12] Xianqiang Bao, Ling Liu, Nong Xiao, Yutong Lu, and Wenqi Cao. “Persistence and Recovery for In-Memory NoSQL Services: A Measurement Study” IEEE International Conference on Web Services (ICWS) pp. 530-537, July 2016.

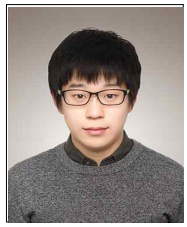
[13] Smith, Jonathan M., Gerald Q., and Maguire Jr. "Effects of copy-on-write memory management on the response time of UNIX fork operations." Computing Systems Vol. 1, No. 3, pp. 255-278, 1988.

[14] Peterson, Zachary Nathaniel Joseph. Data placement for copy-on-write using virtual contiguity. Diss. UNIVERSITY OF CALIFORNIA SANTA CRUZ, 2002.

[15] Memtier-benchmark https://github.com/RedisLabs/Memtier_Benchmark [Accessed: Jan. 02, 2017].

저자소개

진 민 화 (Min-Hwa Jin)



2015년 8월 : 인하대학교 컴퓨터정보공학과 졸업(학사)
2015년 9월 ~ 현재 : 연세대학교 컴퓨터과학과 석사과정
관심분야 : 데이터베이스, 인메모리 시스템, NoSQL, SSD

박 상 현 (Sang-Hyun Park)

1989년 : 서울대학교 컴퓨터공학과 졸업(학사)
1991년 : 서울대학교 대학원 컴퓨터공학과 졸업(공학석사)
2001년 : UCLA 대학원 컴퓨터과학과 졸업(공학박사)
1991년 ~ 1996년 : 대우통신 연구원
2001년 ~ 2002년 : IBM T. J. Watson Research Center Post-Doctoral Fellow
2002년 ~ 2003년 : 포항공과대학교 컴퓨터공학과 조교수
2003년 ~ 2006년 : 연세대학교 컴퓨터과학과 조교수
2006년 ~ 2011년 : 연세대학교 컴퓨터과학과 부교수
2011년 ~ 현재 : 연세대학교 컴퓨터과학과 교수
관심분야 : 데이터베이스, 데이터아미닝, 바이오인포매틱스, 적응적 저장장치 시스템, 플래시 메모리 인덱스, SSD