Redis 에서 Physiological RDB 회복 기법 제안

진민화, 최원기, 박상현
연세대학교 컴퓨터과학과
e-mail : mhjin@yonsei.ac.kr, cwk1412@cs.yonsei.ac.kr, sanghyun@cs.yonsei.ac.kr

Physiological RDB Recovery Method in Redis

Minhwa Jin, Wonki Choi, Sanghyun Park
Dept. of Computer Science, Yonsei University

요 약

Redis는 In-Memory 기반 Key-Value Store로서 Lists, Sets, Sorted Sets, Hashes와 같은 효율적인 자료구조를 제공한다. 그리고 분산 데이터 관리를 위한 Master-Slave Replication, Sharding 등을 제공하기 때문에 Twitter, Weibo, KakaoTalk과 같은 많은 기업에서 사용되고 있다. In-memory 기반의 Redis는 전원이 공급되지 않으면 데이터가 손실되는 문제점이 존재한다. 따라서 Redis에서는 데이터를 생성한 프로그램의 실행이 종료되더라도 데이터의 손실을 방지하는 특성인 영속성(Persistence)을 보장하는 기법을 제공한다. 먼저 Write/Update 연산 자체를 모두 Log파일의 마지막에 기록하는 AOF(Append Only File) 기법과 메인 메모리 내에 있는 데이터 전체를 주기적으로 이미지 파일 형태로 디스크에 저장하는 RDB(Snapshot) 기법을 이용한다. 그 중에서 RDB 기법은 메인 메모리에 존재하는 데이터의 크기가 클 경우 많은 CPU 점유율과 Disk I/O가 발생할 수 있기 때문에 Redis의 성능 저하로 이어질 수 있다는 문제점이 있다. 본 논문에서는 이러한 문제를 해결하기 위해 기존의 RDB 기법을 개선한 Physiological RDB 기법을 통하여 I/O 성능을 향상하는 방법에 대하여 제시하고자 한다.

1. 서론

기존의 데이터 관리를 위해 주로 사용하는 디스크 기반 데이터베이스는 데이터를 디스크에 저장을 하고 필요한 데이터의 대한 질의나 삽입, 삭제, 수정 등의 작업을 디스크에 접근하는 방식을 이용했다. 효율적인 디스크 I/O를 위한 여러 가지 인덱스와 자료구조들이 제시되었고, 기존의 HDD를 대신하여 Flash SSD를 이용해 I/O 향상을 이루었다. 하지만 컴퓨팅 성능의 발전 속도를 디스크와 기존의 디스크 기반 데이터베이스 시스템이 따라가지 못하게 되었고, 분석 데이터의 실시간 처리와 더 빠른 데이터 접근을 위해 디스크 I/O 없이 메모리에 데이터를 저장하고 관리를 하는 Memcached[1], Redis[2]와 같은 In-memory 기반 데이터베이스가 등장하게 되었다.

In-Memory 기반 데이터베이스는 메인 메모리에 설치되어 운영되는 데이터베이스 관리 시스템이다. 즉 데이터베이스의 데이터를 일부 또는 전부를 메인 메모리에서 관리를 하게 되어 기존의 디스크 기반 데이터베이스보다 좋은 성능을 나타낸다.

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

Redis는 기존의 Key-Value Store와 달리 Lists, Sets, Sorted Set 등과 같은 여러 가지 자료구조와 Master-Slave Replication 과 Sharding 기능 등을 제공하기 때문에 대용량 분산 데이터 관리 시스템에서 사용되고 있다. 그러나 Redis에서 데이터 접근은 메모리에서 일어나지만 Redis 인스턴스가 종료되더라도 데이터의 손실을 막기 위해서 Persistence Storage를 이용한다[3]. Redis 는 대표적인 두 가지 방법인 AOF(Append Only File), RDB(Snapshot) 기법을 이용하여 데이터의 손실이 있어도 데이터 복구가 가능함을 보장한다. 이 때 RDB 기법은 메모리에 있는 모든 데이터를 이미지 형태로 디스크에 기록하기 때문에 메모리상에서 관리하는 데이터가 많으면 I/O 비용이 증가하게 된다.

이에 본 연구는 기존 RDB 기법에 대해 알아보고 기존 RDB 기법을 개선한 Physiological RDB 기법을 제안하고 I/O 비용을 효과적으로 줄이는 방법에 대하여 다뤄보고자 한다.

이 논문의 구조는 다음과 같다. 2장에서는 Redis가 구체적으로 어떠한 방식으로 영속성(Persistence)을 보장하는지 살펴보고, 구체적으로 RDB 파일의 구조에 대해 알아본다. 그리고 3장에서는 개선된 RDB 기법을 소개한다. 마지막 4장에서는 논문의 결론과 향후 연구 방향을 제시한다.

2. 배경

2.1 Redis Persistence 기법(AOF, RDB)

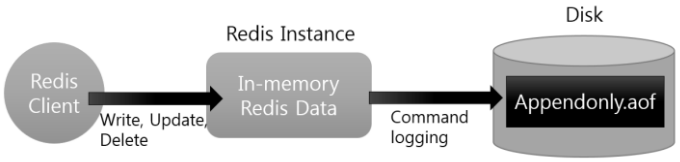

그림 1. AOF 기법

먼저 AOF(Append Only File)는 Redis가 수신하는 모든 쓰기 명령을 AOF 파일에 Log형태로 기록하는 기법이다. AOF파일은 수정이나 삭제가 필요 없으며 명령어를 파일의 마지막에 기록한다.

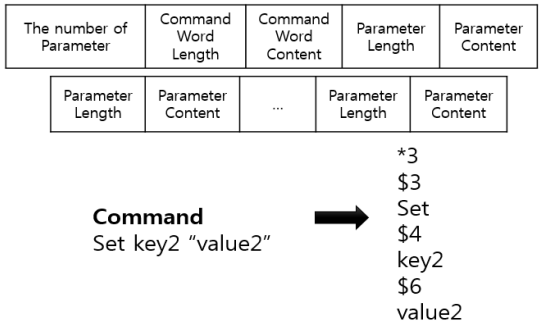

그림 2. AOF 파일에 저장되는 레코드 포맷

그림 2는 실제로 Redis에서 명령어가 AOF 파일로 기록되는 레코드 포맷을 나타낸 그림이다[4]. 전체 Parameter의 개수, 그리고 Parameter의 길이와 값들이 차례대로 저장되는 방식이다. 만들어진 AOF 파일은 복구 시 기록된 Log들을 읽어가면서 Log에 기록된 명령어와 데이터를 확인하여 메모리에 다시 적용하는 방식으로 복구작업을 수행하게 된다.

하지만 AOF 기법의 단점으로는 복구 시 종료되기 직전의 상태로 돌아가기 위해 AOF파일의 Log를 읽어가면서 메모리에 데이터들을 기록하는 과정에서 저장된 Log가 많기 때문에 RDB 방식에 비해 복구 속도가 느리다는 단점이 존재한다.

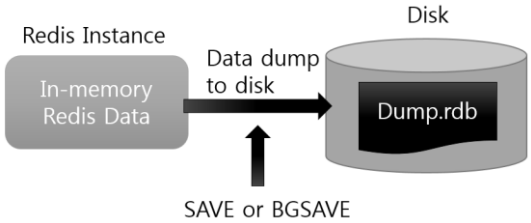

그림 3. RDB 기법

두 번째로 RDB(Snapshot)방식은 사용자가 SAVE 또는 BGSAVE 명령어를 통해서 또는 설정 파일에서 설정한 조건을 만족하는 경우 메모리 상에 있는 모든 데이터에 대한 RDB파일을 만들어 보관하는 방식이다. RDB파일은 다음과 같은 순서로 만들어진다. 우선 Redis는 디스크에 임시 RDB파일을 생성하고 메모리에 있는 데이터들을 기록 한다. 마지막으로 임시 RDB파일에 기록을 완료하면 임시 RDB파일이 기존에 있는 RDB파일을 대체한다. RDB파일은 복구 시 RDB파일 내에 있는 Key, Value 값들을 읽어 메모리에 적용하는 방식으로 복구작업을 수행하게 된다.

하지만 메모리에 저장된 데이터의 크기가 크다면 RDB파일을 만드는데 이에 따른 많은 CPU의 부하와 I/O를 발생할 수 있다. 그리고 RDB파일 추출 후 서버가 다운되면 RDB파일 추출 이후 데이터는 유실된다는 단점이 존재한다. 따라서 일반적으로 RDB방식만 사용하지 않고 AOF방식을 함께 사용하여 데이터 유실을 최소화 시킨다.

2.2 RDB 파일 구조

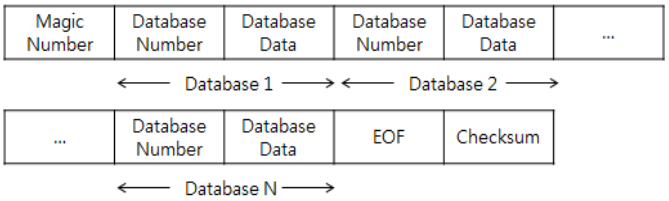

그림 4. RDB 파일 구조

그림 4는 RDB의 전체적인 구조를 나타낸 것이다[4]. 맨 처음에 RDB파일의 고유 파일 값(Magic Number)가 나오고 Redis를 Database Number로 나누어지게 된다. 그리고 Database Number에 해당하는 Key, Value 데이터 값들이 RDB파일에 기록된다. 마지막으로 EOF와 추가적으로 Checksum 옵션을 두게 되면 RDB 파일이 만들어 질 때 Checksum이 추가된다.

그림 5. Key, Value 레코드 포맷

그림 5는 그림 4에서 나타낸 Database Data에 저장되는 Key, Value 쌍이 저장되는 레코드 포맷을 나타낸 그림이다. 먼저 Value의 Type(String, Hash, List, Set 등) 값이 들어가고 Key의 길이, 그리고 Key, Value의 길이, 마지막으로 Value값이 들어가게 된다.

3. Physiological RDB 기법

3.1 기존의 RDB 기법의 문제점

100만개 Key, Value 가 들어있는 RDB 파일 크기	15,626KB
25,600개의 Key에 대한 Value 값 수정 후 RDB 파일이 변경된 데이터 크기	400KB

표 1. 기존 RDB 파일과 Update 연산 후 RDB파일 크기 비교

표 1은 실제 Redis Instance에서 5Byte 크기의 Key 값과 8Byte 크기의 Value 값을 100만 개 Insert연산을 수행한 후 RDB 파일 크기와 25,600개의 Value 값 수정 후 RDB 파일을 생성했을 때의 변경된 파일의 크기를 비교한 것이다. 변경되지 않는 데이터의 크기가 RDB 파일 전체 크기와 거의 비슷하지만 기존의 RDB 방식에서는 모든 데이터를 그대로 저장하기 때문에 15,626KB 만큼의 데이터를 Write 하게 된다. 따라서 변경이 없는 15,226KB만큼의 I/O가 발생하는 것을 알 수 있다.

3.2 Physiological RDB 기법 원리

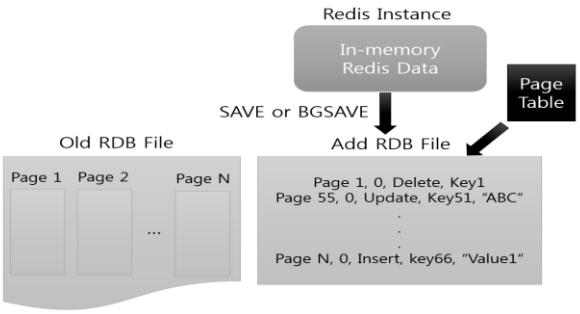

그림 6 Physiological RDB 기법

이 논문에서 제안하는 기법은 메모리 전체 데이터가 아닌 변경된 데이터에 대한 정보만을 저장한다. 그리고 Redis에서 별도의 페이지 테이블을 관리를 하게 된다. 페이지 테이블은 RDB 내부에 Key, Value 데이터가 저장된 논리적인 주소를 페이지 단위로 관리하고 물리적인 주소와 페이지 Number를 제공하는 기능을 한다. RDB를 생성할 때 초기의 Old RDB파일은 그대로 유지하고 수정된 데이터에 대한 페이지의 위치를 페이지 테이블을 참조하여 페이지 Number를 구하고 오퍼레이션 정보와 Key, Value 쌍에 대한 정보들에 대해 RDB 파일을 생성한다.

Page Number (1byte)	Type Of Value (1byte)	Op Code (1byte)	Key (N byte)	Value (M byte)

그림 7 개선된 RDB의 저장 레코드 포맷

그림 7은 구체적으로 디스크에 저장하는 RDB 파일의 레코드 포맷을 나타낸 것이다. 따라서 이 논문에서 제안하는 기법은 오퍼레이션 코드와 페이지의 변경사항을 저장하는 Physiological Logging 방식과 유사한 기법을 이용한다.

기존 RDB 기법과의 차이점은 메모리에 있는 모든 데이터를 임시 RDB에 생성 후 디스크에 반영되면 기존의 RDB파일을 대체하는 방식이지만 논문에서 제시한 개선된 RDB 기법은 초기의 Old RDB파일은 그대로 유지하고 각각의 변경된 페이지에 대한 오퍼레이션 정보를 계속해서 추가하는 기법을 이용한다. 그리고 Physiological RDB 기법은 모든 쓰기 명령을 AOF 파일에 추가하는 AOF 기법과 달리 페이지 테이블을 참조하여 데이터가 저장된 페이지의 위치를 함께 저장하고 복구 작업 시 변경된 페이지에 대해서만 복구작업을 진행한다.

전체 key, value 쌍의 개수를 n, 변경이 안 된 key, value 쌍의 개수를 m이라고 했을 때, Disk에 저장되는 데이터의 크기를 수식으로 표현하면 다음과 같다.

$$\sum_{i}^{n}(Ksize(i)+Vsize(i))-\sum_{i}^{m}(Ksize(i)+Vsize(i))$$

위 식에서 m의 크기가 클수록 RDB 작업을 수행할 때 Disk에 저장되는 데이터의 크기가 작아지는 것을 알 수 있다. 따라서 제시한 기법을 사용하면 중복되는 데이터에 대한 디스크 I/O 횟수를 효과적으로 줄일 수 있게 된다.

4. 결론 및 향후 연구방향

본 논문에서는 기존의 RDB 기법을 개선한 Physiological RDB 기법을 통하여 I/O 성능을 향상한 방법을 제안하였다.

기존의 RDB 기법은 메모리에 존재하는 모든 데이터를 임시의 RDB파일을 생성하고 이전 RDB파일을 대체하게 된다. 따라서 메모리에 저장된 데이터의 크기가 클수록 I/O 비용이 커지게 된다. Physiological RDB 기법은 변경된 데이터와 페이지의 위치 정보, 그리고 오퍼레이션 정보만을 저장하는 방식을 사용한다. 따라서 변경되지 않은 데이터에 대한 디스크 I/O가 줄어들게 된다. 특히 변경되지 않은 데이터의 크기가 많은 경우 상대적으로 갱신된 데이터의 크기가 작기 때문에 I/O 비용이 효과적으로 줄어들 것이다.

본 논문에서 제시한 Physiological RDB 기법을 Redis 뿐만 아니라 다른 종류의 In-memory 기반 데이터베이스에서 적용한다면 RDB를 수행할 때 디스크에 저장하는 비용을 개선할 수 있을 것이라 생각된다.

참 고 문 헌

[1] Memcached http://memcached.org/

[2] Redis http://redis.io/

[3] Hagmann, R.B, A Crash Recovery Scheme for a Memory-Resident Database System, *Computers, IEEE Transactions on,* Volume C-35 , Issue 9, Sept 1986

[4] Ming Xu, Xiaowei Xu, Jian Xu, Yizhi Ren, Haiping Zhang, Ning Zheng, "A Forensic Analysis Method for Redis Database based on RDB and AOF File," in *Journal of Computers,* Vol 9, No 11, 2538-2544, Nov 2014