JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 20, No. 11, pp. 2227-2235, Nov. 2019

단일 노드 인스턴스에서 키-값 저장소 기반 분산 NoSQL 데이터베이스 성능 비교

서 용 철[1] · 박 성 훈[1*] · 김 영 목[1] · 이 재 엽[2] · 김 윤[3]
[1]충북대학교 컴퓨터공학과
[2]주식회사 에코시스텍
[3]한국복지대학교 컴퓨터정보보안과

Comparing Key-Value Store based Distributed NoSQL Database Performance on a Single-Node Instances

Yong-Cheol Seo[1] · Sung-Hoon Park[1*] · Yeong-Mok Kim[1] · Jae-Youp Lee[2] · Yoon Kim[3]

[1]Department of Computer Engineering, Chungbuk National University, Cheongju 28644, Korea
[2]Ecosystec co., Ltd, Bonghwanggongdan 2-gil, Gimje 54363, Korea
[3]Department of Computer Information Security, Korean National University of Welfare, Pyeongtaek 17738, Korea

[요 약]

최근 소셜 네트워킹 및 스트리밍 서비스의 확산으로 분산 데이터 저장소의 확장성 및 탄력성에 대한 요구사항은 구조화되지 않은 쿼리 언어인 NoSQL 데이터베이스 선택에 있어 개발자에게 더욱 중요해졌다. Redis는 캐시 시스템을 통해 데이터베이스 쿼리 결과를 메모리에 캐시할 수 있으므로 데이터베이스 액세스 수를 줄일 수 있다. 그러나 서버 메모리에 적합하지 않은 대량의 데이터가 있을 때 Redis 데이터베이스는 지속성에 대한 한계를 갖고 있어 안전성을 보장할 수 없다. 메모리 기반 Redis 데이터베이스의 새로운 대안으로 키-값 저장소의 디스크 기반 SSDB(Sorted Set DB) 데이터베이스가 있다. 본 논문은 Yahoo에서 제공하는 벤치마크 도구인 YCSB를 사용하여 Redis와 SSDB에 대한 CRUD 처리 성능을 평가 및 비교한다.

[Abstract]

With the recent proliferation of social networking and streaming services, the requirements for scalability and elasticity of distributed data repositories have become more important for developers in choosing a NoSQL database, an unstructured query language. Redis can cache database query results in memory through its caching system, reducing the number of database accesses. However, when there is a large amount of data that is not suitable for server memory, the Redis database has a limit on persistence and cannot guarantee safety. A new alternative to the memory-based Redis database is the disk-based Sorted Set DB (SSDB) database with Key-Value store. This paper evaluates and compares CRUD processing performance for Redis and SSDB using YCSB, a benchmark tool provided by Yahoo.

색인어 : 분산 데이터베이스, 키-값 저장소, NoSQL, Redis, SSDB
Key word : Distributed Database, Key-Value store, NoSQL, Redis, SSDB

http://dx.doi.org/10.9728/dcs.2019.20.11.2227

Received 26 September 2019; **Revised** 31 October 2019
Accepted 05 November 2019

***Corresponding Author; Sung-Hoon Park**

Tel: +82-43-261-3269
E-mail: ycseo@cbnu.ac.kr

Ⅰ. 서 론

최근 소셜 네트워킹 및 스트리밍 서비스의 확산으로 구조화되지 않은 쿼리 언어(NoSQL) 데이터베이스의 사용이 증가하는 추세를 보이고 있다. 주요 장점 중 하나는 관계형 데이터베이스 관리 시스템(RDBMS)보다 빠르고 효율적인 성능을 보장하기 때문이다[1]. NoSQL 데이터베이스는 빠르게 성장하는 클라우드 컴퓨팅 환경에 맞게 조정되어 탄력적으로 응용 프로그램 개발 단순화에 대한 대규모 확장을 가능하게 하고 있다[2]. 그림 1은 오스트리아 컨설팅 회사인 Solid IT에서 발표한 데이터베이스 인기 순위로 10위 내의 현황을 보여주고 있다. RDBMS에서는 Oracle, MySQL, Microsoft SQL Server, PostgreSQL이 상위권에 위치해 있으며 NoSQL에서는 MongoDB, Redis, Cassandra가 상위권으로 가파른 성장세를 보여주고 있다.

Rank			DBMS	Database Model	Score		
Sep 2019	Aug 2019	Sep 2018			Sep 2019	Aug 2019	Sep 2018
1.	1.	1.	Oracle	Relational, Multi-model	1346.66	+7.18	+37.54
2.	2.	2.	MySQL	Relational, Multi-model	1279.07	+25.39	+98.60
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1085.06	-8.12	+33.78
4.	4.	4.	PostgreSQL	Relational, Multi-model	482.25	+0.91	+75.82
5.	5.	5.	MongoDB	Document	410.06	+5.50	+51.27
6.	6.	6.	IBM Db2	Relational, Multi-model	171.56	-1.39	-9.50
7.	7.	7.	Elasticsearch	Search engine, Multi-model	149.27	+0.19	+6.67
8.	8.	8.	Redis	Key-value, Multi-model	141.90	-2.18	+0.96
9.	9.	9.	Microsoft Access	Relational	132.71	-2.63	-0.69
10.	10.	10.	Cassandra	Wide column	123.40	-1.81	+3.85

그림 1. 데이터베이스 엔진 순위
Fig. 1. Database Engines Ranking

Big Data 및 웹 응용 프로그램 개발에서 모든 NoSQL 데이터베이스가 비슷한 성능을 나타내는 것이 아니기 때문에 데이터베이스 선택에 있어 많은 주의가 필요하다[3]. NoSQL 데이터베이스는 아직까지 성숙 단계에 있으며 각기 서로 다른 성장 속도로 진행되기 때문에 관리자는 데이터베이스의 일관성, 성능, 보완, 확장성, 비용 및 기타 기능적 기준과 관련하여 많은 부분을 고려하여 NoSQL 데이터베이스를 신중하게 선택해야 할 필요가 있다[4].

상당수의 오픈 소스와 쉽게 사용할 수 있는 NoSQL 기반 시스템을 갖춘 웹 응용프로그램 개발자는 많은 NoSQL 데이터베이스 중에서 적절한 선택에 어려움을 경험하고 있다. 데이터베이스의 선택이 다양해진 만큼 개발자는 NoSQL 데이터베이스의 특성을 이해하고 유사한 NoSQL의 차이를 정확하게 이해해야 한다.

본 논문에서는 인 메모리(In-Memory) 기반 데이터베이스와 온 디스크(On-Disk) 기반 데이터베이스를 비교 설명하고 유사한 NoSQL 데이터베이스를 선택할 때 고려되어야 할 사항을 설명한다. 또한, 데이터베이스 성능 비교를 위해 NoSQL 데이터베이스 중 하나인 인 메모리 기반 데이터베이스인 Redis와 Redis의 새로운 대안으로 사용할 수 있는 온 디스크 기반 데이터베이스인 SSDB에 대해 성능 비교를 수행한다.

Ⅱ. 관련 연구

NoSQL의 데이터 모델의 유형은 Key-Value Store, Documents Store, Wide Column Store, Graph Store 네 가지 범주로 분류할 수 있다[4], [5].

키-값 저장소(Key-Value Store)는 키와 값으로 이루어진 심플한 구조의 데이터 모델 저장 스토어이며 수평적 확장이 용이하다[6]. 또한, 간단한 API를 제공하기 때문에 사용하기 쉬우며 질의 속도가 빠르다. 이러한 구조는 사용하는 이유는 속도가 빠르며 분산 저장 환경에 용이하기 때문이다. 주로 빠른 액세스를 위해 메모리를 기반으로 사용한다[7].

도큐먼트 저장소(Document Store)는 BSON(Binary JavaScript Object Notation) 또는 JSON(JavaScript Object Notation)의 형태로 문서 지향 데이터를 저장하는 데이터 모델이다[8], [9]. 키-값 데이터 모델에서 한층 진화한 데이터 모델로 데이터는 키와 도큐먼트의 형태로 저장된다. 키-값 데이터 모델과 다른 점은 값이 계층적인 형태인 도큐먼트로 저장된다는 것이다[10]. 주요 특징은 개체를 도큐먼트의 형태로 바로 저장 가능하기 때문에 검색에 최적화되어 있다.

컬럼-패밀리(Column-Family) 데이터 모델은 행(Row)인 키(Key) 값과 컬럼 패밀리(Column-family), 컬럼 네임(Column-name)을 가진다. Column-family 안에는 연관된 데이터들로 구성되어 있으며, 각자의 Column-name을 가진다[4]. 주요 특징으로 Column-family 데이터 모델은 클러스터링이 쉬우며, Time stamp가 존재해 값이 수정된 이력을 알 수 있다. 또한 값들은 바이너리 데이터로 저장되기 때문에 여러 형태의 데이터도 저장할 수 있다[11].

그래프(Graph Store) 데이터 모델은 관계를 데이터와 함께 저장한다[4]. 즉, 데이터의 연속적인 노드(node), 엣지(edge), 프로퍼티(property)를 그래프 형태로 저장한다[4]. 연관된 데이터를 검색해주는 검색 엔진이나 패턴 인식을 위한 데이터베이스로 적합하다[12].

[13]에서는 여러 NoSQL 데이터베이스에 대해 각 성능적인 측면을 상호 비교하였으나 차이의 원인을 자세히 다루진 않았다. 특히 Redis와 Redis의 새로운 대안인 SSDB와 성능을 비교 분석했다는 점이 다르다고 할 수 있다.

본 논문에서는 서버 메모리에 적합하지 않은 대량의 데이터가 있을 때 Redis 데이터베이스가 갖고 있는 한계와 Redis 데이터베이스의 새로운 대안으로 키-값 저장소의 디스크 기반 SSDB 데이터베이스를 NoSQL 벤치마크 도구인 YCSB(Yahoo! Cloud Serving Benchmarking)를 사용하여 주어진 워크로드를 처리하는 데 있어서 성능과 한계를 비교하고 분석하는 데 목적이 있다.

2-1 In-Memory와 On-Disk 기반 키-값 데이터 저장소

키-값 데이터 모델을 저장 시스템 기준으로 크게 두 가지 범주로 나누면 인 메모리와 온 디스크로 분류할 수 있다.

첫 번째 인 메모리 기반 키-값 데이터 모델은 메모리 내에서 데이터 저장소가 실행되면 데이터가 메모리에 완전히 로드 되므로 모든 작업이 메모리에서 실행된다[14]. 일반적으로 시스템에서는 데이터를 디스크에 주기적 또는 비동기적으로 저장하지만 모든 데이터는 메모리에서 검색한다[10]. 또한, 백업 및 시스템 종료를 위해 메모리 내 데이터를 디스크에 복사한다. 인 메모리 데이터 저장소의 주요 이점은 대기 시간이 짧고 처리량이 향상된다는 것이다. 또한, 메모리 내 키-값 저장소는 클라이언트와 서버 간 낮은 네트워크 통신 오버헤드가 필요로 하므로 높은 처리량에 크게 기여한다. 대표적인 인 메모리 데이터 저장소는 Redis 및 Memcached 데이터베이스가 있다[7],[15].

두 번째 온 디스크 기반 키-값 데이터 모델은 분산형 스토리지 시스템 또는 단일 노드 플랫폼에서 메모리에 자주 액세스하는 데이터가 있는 하드 디스크의 데이터를 저장하는 저장소 또는 메모리에 데이터를 저장할 수 있는 데이터 저장소가 될 수 있지만, 저장소를 연결하는 것도 허용한다[10]. 온 디스크 기능의 이점은 스토리지의 바이트 당 비용을 줄이고 스토리지 용량을 증가시키는 것이다. 실제로 디스크 공간에서 메모리 공간이 부족하거나 부족이 예상될 경우 디스크 데이터 저장소가 메모리 저장 유형의 대안으로 사용될 수 있다. 대표적인 키-값 데이터 모델인 분산형 데이터베이스로는 BerkeleyDB, Voldemort 및 Riak 등과 같은 것들이 있다[16].

2-2 Redis

Redis 데이터베이스는 캐시(Cache) 및 메시지 브로커(Message Broker)로 사용되는 오픈 소스(BSD 라이센스) 인 메모리 데이터 구조인 저장소이다[15]. 문자열(Strings), 목록(Lists), 집합(Sets), 정렬된 집합(Sorted Sets), 해시(Hash), HyperLogLogs 및 비트맵(Bitmaps)과 같이 다양한 종류의 데이터 형식이 지원되지만, 키-값 구조의 데이터 모델이다[15].

Redis에는 복제 기능이 내장되어 있으며 마스터-슬레이브(Master-Slave) 모델을 사용하여 복제할 수 있으며 마스터는 여러 개의 종속을 가질 수 있다[3],[15]. Redis는 TCP 소켓과 간단한 프로토콜을 사용하는 클라이언트/서버 모델을 사용하여 전송되는 일련의 명령을 통해 변경 가능한 데이터 구조에 대한 액세스를 제공한다. Redis는 LRU (Least Recently Used) 캐시로 사용할 수 있으며, 근사치 LRU 알고리즘을 사용하여 새 데이터가 추가될 때 기존 데이터를 제거할 수 있다. 또한, C로 작성되었고 Redis에 내장된 강력하고 가벼운 스크립팅 언어인 Lua를 사용하여 스크립팅 기능을 제공한다.

Redis는 Redis Sentinel이라는 기능을 사용하여 마스터가 예상대로 작동하지 않는 경우 자동 장애 조치를 지원한다. 이 기능은 슬레이브가 마스터로 승격된 장애 조치 프로세스를 시작하고 추가 슬레이브도 새 마스터를 사용하도록 재구성된다. Sharding은 Redis 클러스터를 통해 실행된다. 플랫폼에서는 데이터가 여러 Redis 노드에 자동으로 분할된다.

Redis의 Architecture는 그림 2와 같이 크게 3가지 영역인 메모리, 파일, 프로세스 영역으로 구성되어 있다[17].

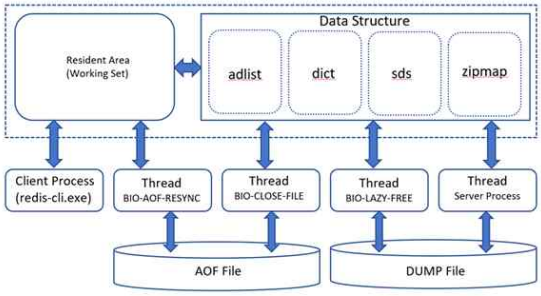


그림 2. Redis 아키텍처
Fig. 2. Redis Architecture

2-3 SSDB

SSDB는 Redis의 새로운 대안으로 개발되고 있는 데이터베이스로 수십억 개의 요소를 저장하는 빠른 NoSQL 데이터베이스이다. SSDB는 Redis와 같은 키-값 데이터 모델로 구성되어 있으며, 문자열, 목록, 집합, 정렬된 집합, 해시를 포함하고 있는 데이터 구조를 지원한다.

SSDB는 그림 3과 같이 C/C++로 개발된 Google의 LevelDB를 스토리지 엔진으로 사용하고 있으며 Redis의 네트워크 프로토콜과 오픈 소스 Redis 클라이언트를 지원하는 아키텍처로 구성되어 있다. 또한, Redis와 같은 복제(Master-Slave), 로드 밸런스 기능을 갖고 있으며 개발과 배포를 위한 사용하기 쉬운 클라이언트 API를 지원하고 있다[18].

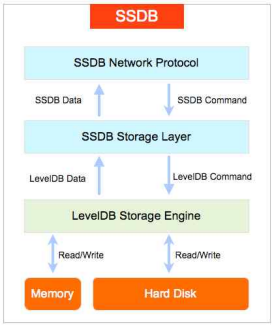


그림 3. SSDB 아키텍처
Fig. 3. SSDB Architecture

Ⅲ. 실험 및 결과

3-1 실험 환경

본 논문에서 진행하는 벤치마킹 프로세스는 단일 노드 인스턴스를 사용하여 Redis와 SSDB에 대한 NoSQL 데이터베이스의 성능을 비교하는 데 목적이 있다. 성능 분석 도구는 앞장에서 말한 YCSB를 사용했으며 워크로드의 A타입부터 F

타입까지 실행하는 방식을 사용하였다. 실험을 위한 시스템 환경은 표 1과 같다.

표 1. 성능 평가 시스템의 환경
Table 1. Environment of Performance Evaluation System

Performance Evaluation Tool	Yahoo! Cloud Serving Benchmarking (YCSB) 0.15.0	
Database	Redis 5.0.5	SSDB 1.9.4
OS	Ubuntu Server 18.04 LTS 64Bit	
CPU	Intel Xeon CPU E5-2620 (2.0GHz 2 * 6 Cores)	
RAM	48GB	
HDD	1TB	

3-2 성능 분석 방법

성능 비교의 결과는 제안한 것처럼 Redis의 대안으로 SSDB의 적합성을 입증하는 것이다. 성능 비교를 위해 선택한 벤치마크 도구는 YCSB이다. YCSB는 클라우드 서비스의 성능을 측정하기 위한 벤치마크 도구로써 쿼리 패턴이나 분포를 설정하여 분석할 수 있다. YCSB의 사용 목적은 NoSQL 도입을 검토하려 할 때 특정 상황에 맞는 벤치마킹 환경 및 측정 결과를 제공함으로써 해당 NoSQL이 선정에 적합한지 판단할 기준을 제공한다. YCSB는 플러그인 기반 아키텍처를 가지고 있으며 스크립트를 사용하여 쉽게 확장성을 제공하므로 Redis와 SSDB를 벤치마킹할 수 있는 NoSQL 벤치마크 프레임워크이다[2]. YCSB 아키텍처는 두가지로 구성이 되어있다. 첫 번째는 확장 워크로드를 생성하는 YCBS Client가 구성되어 있으며, 두 번째는 확장 워크로드 생성기에서 실행되는 워크로드 시나리오 세트를 제공해주는 Core workloads로 구성되어 있다. 따라서 YCBS를 사용하여 실제 서비스 요청과 같은 유사한 패턴 결과를 얻을 수 있다. 그림 4는 YCSB의 아키텍처를 나타내고 있다[3].

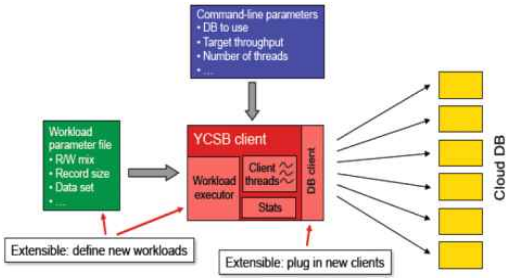


그림 4. YCSB 아키텍처
Fig. 4. The YCSB Architecture

YCSB를 사용하여 CRUD의 연산 비중을 다르게 설정하여 요청 패턴을 실험할 수 있으며 제공되는 대표적인 YCSB의 워크로드는 표 2와 같다. 대표적인 워크로드는 A타입부터 F타입까지 있으며 성능 비교에 필요한 측정값을 얻을 수 있다[2]. 표 2에서 Operations는 실행하는 워크로드의 연산 비중을 뜻한다. 예를 들어 워크로드 A의 Update heavy를 전체 연산 중에서 50%로 Read 연산을 진행하며, 50%로 Update 연산을 진행하는 것을 뜻한다. 또한, Record selection은 어떤 방법으로 연산할 레코드를 검색할지를 뜻한다. 지프 분포를 적용해 레코드를 탐색하는 Zipfian가 있고, 최근 데이터를 기준으로 레코드를 탐색하는 Latest가 있으며, Continuous Uniform Distribution으로 연속 균등 분포를 통한 확률 분포로 레코드를 탐색하는 Uniform이 있다. 표 2의 설정값들은 기본 설정이 되어있으며, 변경이 필요하다면 Operation 비중과 Record Select 타입을 실험 환경에 맞게 변경할 수 있다. 각각 연산에서 사용되는 쿼리문의 경우, 임의 값을 삽입하여 데이터를 입력하고, 수정 및 삭제, 탐색은 기본 키값에 의해 진행한다.

본 연구에서 임의의 랜덤 값을 이용하여 CRUD를 각 워크로드별로 비교하였으며 스레드의 증가에 따른 데이터의 수를 1000, 5000, 10000개까지 늘리며 각 100회씩 실험을 진행하였으며 그 결과를 기본적으로 제공하는 각 워크로드를 기준으로 도식화했다.

표 2. 코어 패키지의 워크로드[2]
Table 2. Workloads in the core package[2]

Workload	Operations	Record selection	Application Example
A (Update heavy)	Read: 50% Update: 50%	Zipfian	This workload has a mix of 50/50 reads and writes. An application example is a session store recording recent actions.
B (Read mostly)	Read: 95% Update: 5%	Zipfian	This workload has a 95/5 reads/write mix. Application example: photo tagging; add a tag is an update, but most operations are to read tags.
C (Read only)	Read: 100%	Zipfian	This workload is 100% read. Application example: user profile cache, where profiles are constructed elsewhere (e.g., Hadoop).
D (Read latest)	Read: 95% Insert: 5%	Latest	In this workload, new records are inserted, and the most recently inserted records are the most popular. Application example: user status updates; people want to read the latest.
E (Short ranges)	Scan: 95% Insert: 5%	Zipfian/ Uniform	In this workload, short ranges of records are queried, instead of individual records. Application example: threaded conversations, where each scan is for the posts in a given thread (assumed to be clustered by thread id).
F (Read-modify-write)	Read: 50% Read-modify-write: 50%	Zipfian	In this workload, the client will read a record, modify it, and write back the changes. Application example: user database, where user records are read and modified by the user or to record user activity.

Ⅳ. 비교 분석 결과

4-1 Workload A (Update heavy)

워크로드 A는 Update heavy로 작업한 결과를 나타내고 있다. SSDB는 스레드 수가 증가함에 따라 Redis 보다 성능이 우수하고 처리량이 크게 분산되고 있다. 그림 4와 같이 스레드의 수가 증가하면 SSDB의 성능이 향상된다는 것을 보여준다.

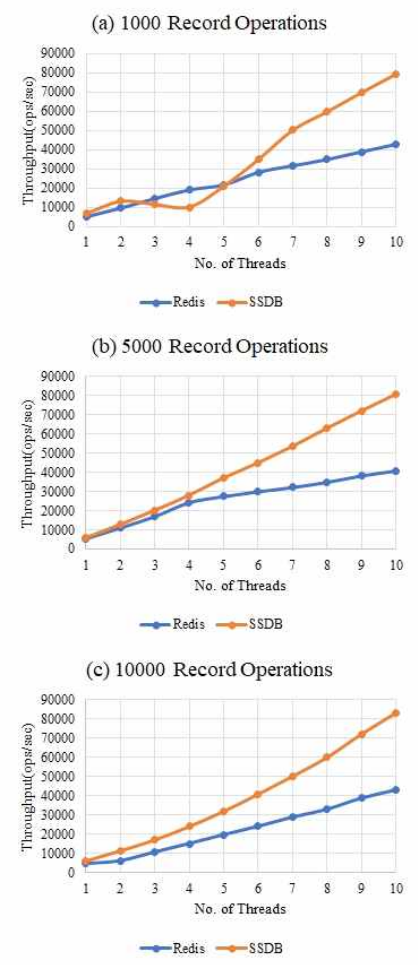


그림 4. 워크로드 A의 성능 비교
Fig. 4. Performance comparison of Workload A

4-2 Workload B (Read mostly)

워크로드 B는 Read mostly로 작업한 결과를 나타내고 있다. 결과는 Update heavy 작업과 유사하다. 그림 5와 같이 SSDB는 Redis 보다 성능이 우수하다. 그러나 그림 5(c)와 같이 레코드 건수가 10000개에 이르면 Redis 보다 성능이 떨어지는 것을 확인할 수 있다.

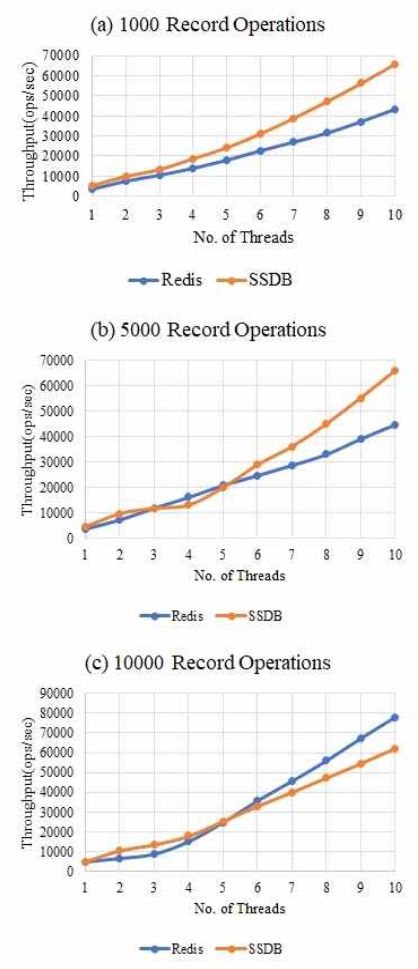
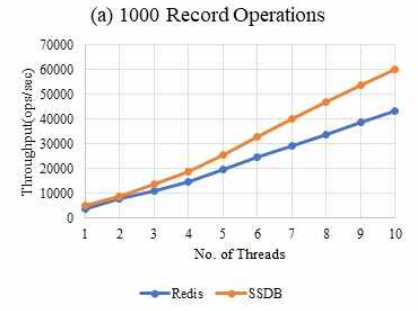


그림 5. 워크로드 B의 성능 비교
Fig. 5. Performance comparison of Workload B

4-3 Workload C (Read only)

워크로드 C는 Read only로 작업한 결과를 나타내고 있다. 그림 6과 같이 SSDB의 처리량은 배포된 스레드 수가 증가함에 따라 Redis의 처리량과 큰 차이를 나타내고 있다.



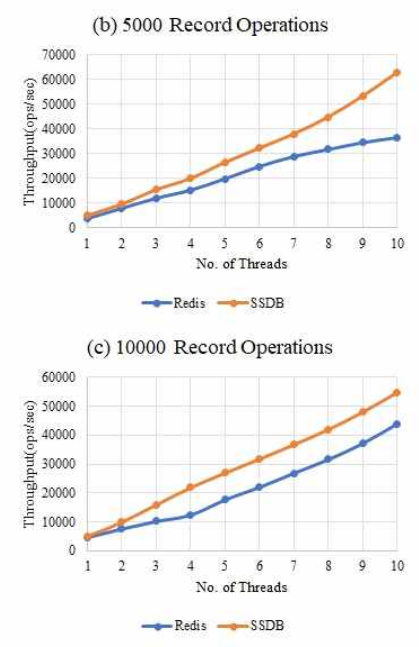
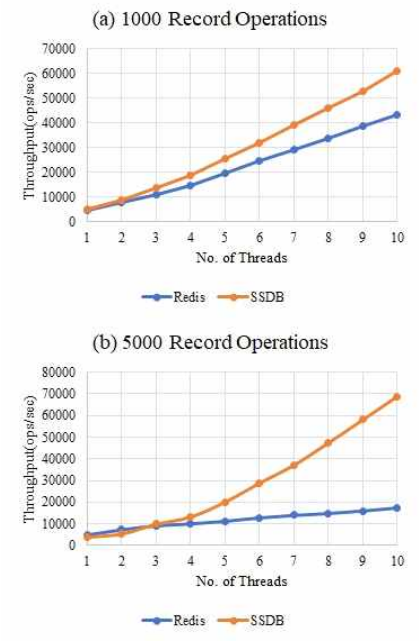


그림 6. 워크로드 C의 성능 비교
Fig. 6. Performance comparison of Workload C

4-4 Workload D (Read latest)

워크로드 D는 Read latest로 작업한 결과를 나타내고 있다. 그림 7과 같이 SSDB 처리량 결과가 Redis의 처리량을 어떻게 초과하는지 보여준다. 또한, 스레드 수가 10으로 증가함에 따라 처리량이 크게 향상됨을 나타내고 있다.



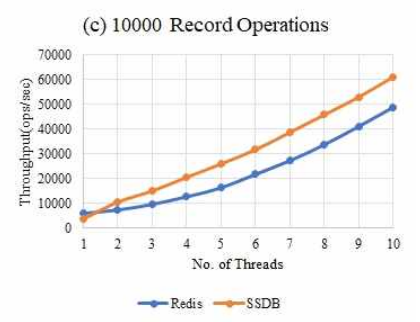


그림 7. 워크로드 D의 성능 비교
Fig. 7. Performance comparison of Workload D

4-5 Workload E (Short ranges)

워크로드 E는 Short ranges로 작업한 결과를 나타내고 있다. 그림 8과 같이 SCAN 작업에서는 Redis가 처리량 측면에서 SSDB보다 뛰어난 성능을 보여준다. 이는 SCAN 작업에서 SSDB는 스레드 수를 증가하더라도 성능 향상에 영향을 미치지 못함을 나타내고 있다.

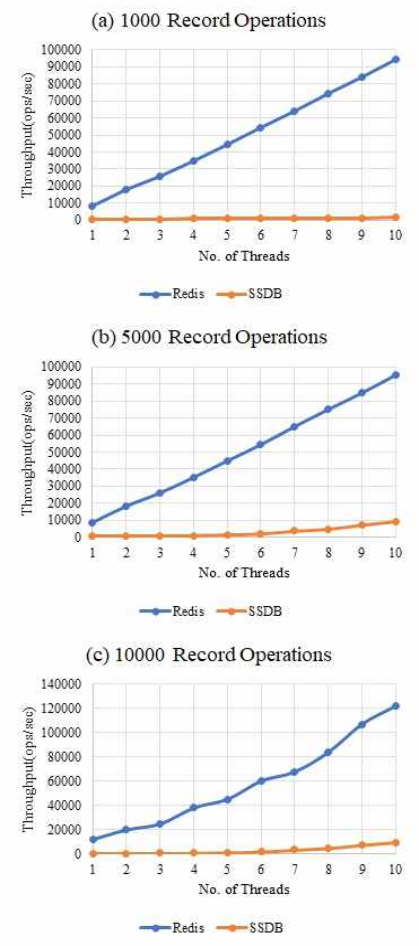


그림 8. 워크로드 E의 성능 비교
Fig. 8. Performance comparison of Workload E

4-6 Workload F (Read-modify-write)

워크로드 F는 Read-modify-write 작업한 결과를 나타내고 있다. 그림 9는 워크로드 D와 유사하다. SSDB는 스레드 수가 1에서 4까지 Redis와 유사한 성능을 나타낸다. 그러나 스레드의 수가 증가되면 성능의 격차가 커지는 현상을 보여주고 있다.

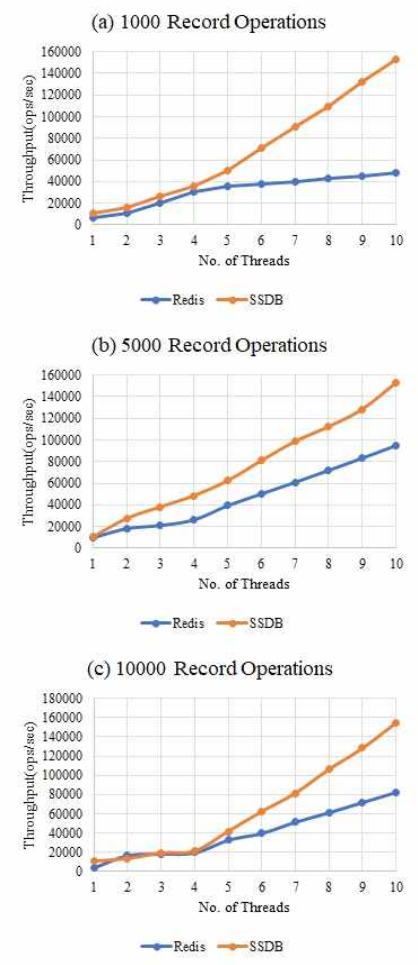


그림 9. 워크로드 F의 성능 비교
Fig. 9. Performance comparison of Workload F

4-7 비교 분석 결과

앞 절에서 진행된 실험에서는 Redis와 SSDB 두 데이터베이스의 단일 노드의 인스턴스만 사용되었다. SSDB는 Heavy Read 작업에서 Redis의 처리량보다 약 2배 이상의 큰 효과를 나타냈다. 그러나 Short ranges 작업과 같은 SCAN 작업에서는 약 10배 가까운 상당한 감소를 나타냈다. 이는 개발자가 Update, Heavy Read 및 Read-Modify-write 작업을 수행하는 응용프로그램 시나리오에서 탄력성을 위해 SSDB를 채택하는 것을 고려할 수 있음을 보여준다. Redis의 새로운 대안이라고 주장하는 SSDB는 6가지 워크로드 중 5가지 작업에는 유효한 성능을 보인 반면 SCAN 작업이 많은 워크로드 E는 성능면에서 Redis가 높은 성능을 보였다. 또한, 일부 워크로드에서는 스레드 수가 증가할수록 Redis 보다 SSDB가 우수한 성능을 나타내고 있지만 워크로드 E에서는 스레드의 수가 증가되면 SSDB보다 Redis가 우수한 성능을 나타내고 있어 SCAN 작업이 우선시 되는 응용프로그램 시나리오에서는 데이터베이스 선택에 있어 SSDB 보다 Redis가 더 적합하다는 것을 측정 결과로 나타내고 있다.

Ⅴ. 결 론

새로운 아키텍처를 가진 데이터베이스 솔루션들이 최근 많이 제안되어 개발되고 있는 추세다. 또한, 유사한 아키텍처를 가지는 NoSQL 데이터베이스도 있어 동일한 데이터 모델을 갖는 데이터베이스를 선택할 때 성능적인 요소들에 대한 충분한 검토가 선행되어야 한다.

본 논문에서는 NoSQL중 키-값 데이터 모델을 기반으로 하는 Redis와 SSDB를 비교하여 성능을 비교 분석하였다. 앞 장의 비교 분석 결과와 같이 데이터베이스를 선택할 때 높은 성능을 보여주고 있는 해당 워크로드의 작업에 알맞은 솔루션 선택 시나리오에 중요한 지표를 제시하고 있다. 본 논문에서는 Redis의 새로운 대안인 SSDB는 실험 결과에서 대부분 높은 성능을 나타내고 있음을 입증하였으며 향후 SSDB가 소셜 네트워킹 및 스트리밍 서비스 처리에 적합하게 사용될 것으로 기대된다.

향후 진행할 연구 계획으로는 NoSQL 데이터베이스를 선택할 때 고려해야 할 사항으로 성능 문제뿐만 아니라 데이터의 보안 문제도 검토되어야 한다. NoSQL의 성격과 특성(데이터의 양, 처리 속도, 다양성 및 신뢰성)으로 인해 보안 문제가 발생한다. 고려 대상 중 하나는 응용프로그램의 성능을 저하시키지 않으면서 암호화된 데이터베이스 시스템을 쿼리하는 방법이다. 이 연구가 보안 문제를 다루지는 않았지만, 추가적인 성능과 보안에 대해 향후 추가적인 연구를 진행할 예정이다.

감사의 글

본 논문은 교육부와 한국연구재단의 재원으로 지원을 받아 수행된 사회맞춤형 산학협력 선도대학(LINC+) 육성사업의 연구결과입니다(과제번호: 201914087001).

참고문헌

[1] C. Kumarasinghe, K. Liyanage, W. Madushanka, and R. Mendis, Performance Comparison of NoSQL Databases in Pseudo Distributed Mode: Cassandra, MongoDB & Redis, 2016.

[2] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, Benchmarking cloud serving systems with YCSB, *Proceedings of the 1st ACM symposium on Cloud computing*, pp. 143-154, 2010.

[3] Y. Abubakar, T. S. Adeyi, and I. G. Auta, Performance evaluation of NoSQL systems using YCSB in a resource austere environment, *Performance Evaluation*, Vol.7, No.8, pp.23-27, 2014.

[4] A. Oussous, F. Benjelloun, A. A. Lahcen, and S. Belfkih, Comparison and classification of nosql databases for big data, *Proceedings of International Conference on Big Data, Cloud and Applications*, Vol. 2, 2015.

[5] H. Khazaei, M. Fokaefs, S. Zareian, N. Beigi-Mohammadi, B. Ramprasad, M. Shtern, P. Gaikwad, and M. Litoiu, How do I choose the right NoSQL solution? A comprehensive theoretical and experimental survey, *Big Data and Information Analytics (BDIA)*, Vol.2, pp.1, 2016.

[6] E. Anderson, X. Li, M. A. Shah, J. Tucek, and J. J. Wylie, What consistency does your key-value store actually provide? *HotDep*, Vol. 10, pp. 1-16, 2010.

[7] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, Workload analysis of a large-scale key-value store, *ACM SIGMETRICS Performance Evaluation Review*, Vol. 40, No. 1, pp. 53-64, 2012.

[8] K. Ma and A. Abraham, Toward lightweight transparent data middleware in support of document stores, *2013 Third World Congress on Information and Communication Technologies (WICT 2013)*, pp. 253-257, 2013.

[9] C. Chasseur, Y. Li, and J. M. Patel, Enabling JSON Document Stores in Relational Systems. *WebDB*, Vol. 13, pp. 14-15, 2013.

[10] R. Cattell, Scalable SQL and NoSQL data stores, *Acm Sigmod Record*, Vol.39, No.4, pp.12-27, 2011.

[11] D. J. Abadi, Column Stores for Wide and Sparse Data. *CIDR*, Vol. 2007, pp. 292-297, 2007.

[12] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar, Bidirectional expansion for keyword search on graph databases, *Proceedings of the 31st international conference on Very large data bases*, pp. 505-516, 2005.

[13] E. Tang and Y. Fan, Performance comparison between five NoSQL databases, *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*, pp. 105-109, 2016.

[14] J. Han, E. Haihong, G. Le, and J. Du, Survey on NoSQL database, *2011 6th international conference on pervasive computing and applications*, pp. 363-366, 2011.

[15] About Redis[internet]. Available: http://redis.io.

[16] K. Grolinger, W. A. Higashino, A. Tiwari, and M. A. Capretz, Data management in cloud environments: NoSQL and NewSQL data stores, *Journal of Cloud Computing: Advances, Systems and Applications*, Vol.2, No.1, pp.22, 2013.

[17] About Redis Architecture[internet]. Available: https://sqlmvp.tistory.com/1321.

[18] About SSDB[internet]. Available: http://ssdb.io.

서용철(Yong-Cheol Seo)

2004년 : 충주대학교 컴퓨터공학과 (공학사)
2018년 : 충북대학교 컴퓨터공학과 (공학석사)

1997년~2018년: 자화전자(주), 책임연구원
2018년~현 재: 충북대학교 대학원 컴퓨터공학과 (박사)
※관심분야 : 분산 처리, 분산 데이터베이스, 미들웨어

박성훈(Sung-Hoon Park)

1993년 : 인디애나대학교 컴퓨터공학과 (공학석사)
2000년 : 인디애나대학교 컴퓨터공학과 (공학박사)

1994년~1996년: ㈜ 두산컴퓨터 연구소 소장
1994년~2004년: 남서울대학교 컴퓨터공학과 교수
2004년~현 재: 충북대학교 컴퓨터공학과 교수
※관심분야 : 분산/모바일/유비쿼터스 컴퓨팅, 알고리즘, 컴퓨터 이론, 미들웨어

김영목(Yeong-Mok Kim)

1983년 : 고려대학교 경영대학 경영학과 (경영학사)
2013년 : 충북대학교 컴퓨터공학과 (공학석사)
2017년 : 충북대학교 컴퓨터공학과 (공학박사)

1983년~2012년: ㈜ 두산 대표이사 CTO
2017년~현 재: 충북대학교 컴퓨터공학과 교수
※관심분야 : 분산/모바일/유비쿼터스 컴퓨팅, 알고리즘, 상호배제, 미들웨어

이재엽(Jae-Youp Lee)

1992년 : 한남대학교 전자계산학과 (공학사)
1999년 : 홍익대학교 컴퓨터공학과 (공학석사)
2010년 : 충북대학교 컴퓨터공학과 (공학박사)

2011년~현 재: ㈜ 에코시스텍 대표이사
※관심분야 : 분산처리, 데이터베이스

김 윤(Yoon Kim)

1982년 : 한양대학교 기계공학과 (공학사)
1988년 : Stevens Institute of Technology 컴퓨터과학과 (이학석사)
2011년 : 충북대학교 컴퓨터공학과 (공학박사)

1998년~2000년: ㈜ 엠차지정보기술 연구소장
2001년~현 재: 한국복지대학교 컴퓨터정보보안과 교수
※관심분야 : 분산컴퓨팅, 미들웨어, 디지털 포렌식