ISSN 2383-6318(Print) / ISSN 2383-6326(Online)
KIISE Transactions on Computing Practices, Vol. 28, No. 6, pp. 332-338, 2022. 6
https://doi.org/10.5626/KTCP.2022.28.6.332

Redis 성능 향상을 위한 비선형 기계학습 기반의 파라미터 튜닝 연구

(A Study on Parameter Tuning Based on Non-linear Machine Learning to Improve Redis Performance)

서 주 연 † 이 지 은 † 김 경 훈 †
(Juyeon Seo) (Jieun Lee) (Kyeonghun Kim)

JIN HUIJUN † 박 상 현 ††
(HUIJUN JIN) (Sanghyun Park)

요 약 인메모리 데이터베이스인 Redis는 싱글 스레드 기반의 프로그램으로, 한 번에 하나의 명령어만 처리 가능하기 때문에 지속성 방법과 같은 다양한 작업을 백그라운드로 수행한다. 그러나 빈번한 백그라운드 작업은 Redis의 데이터 처리 지연 및 메모리 사용량 증가와 같은 추가적인 부하를 초래한다. 이러한 Redis의 작업 부하를 완화시키기 위해, 파라미터 튜닝을 수행한 선행 연구에서는 의미 있는 파라미터를 추출하기 위하여 선형적 관계만을 고려한다. 이를 개선하기 위해 본 논문에서는 비선형 기계학습 기법을 활용한 파라미터 튜닝을 수행하였다. 해당 모델은 Redis의 성능에 영향력 높은 파라미터를 분석하여, 보다 높은 데이터 처리 성능을 도출해내기 위해 각 파라미터마다 최적의 값을 설정한다. 추출한 파라미터 조합을 이용하여 Redis가 사용하는 지속성 방법에 따른 단위 시간당 처리량을 측정한 결과, 기존 파라미터 대비 최대 45.9%의 성능 향상을 확인하였다.

키워드: DBMS 파라미터 튜닝, 레디스, 기계학습, 지속성 방법

Abstract Redis, in-memory database which is single threaded, performs various operations in the background, such as persistence methods processing, because it can execute only one command at a time. However, frequent background operations lead to additional loads on Redis such as delayed processing and more memory usage. To mitigate these loads on Redis, a preceding study that performed parameter tuning only consider linear correlation to extract meaningful parameters. To address this problem, we performed parameter tuning using non-linear machine learning methods. The model analyzed parameters which are influential on Redis performance, and decides the optimal values

· 이 논문은 2022년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원(IITP-2017-0-00477, (SW스타랩) IoT 환경을 위한 고성능 플래시 메모리 스토리지 기반 인메모리 분산 DBMS 연구개발)과 국토교통부의 스마트시티 혁신인재육성사업의 지원을 받아 수행된 연구임
· 이 논문은 2021 한국컴퓨터종합학술대회에서 '비선형 기계학습 기반의 Redis 파라미터 튜닝 연구'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 연세대학교 컴퓨터과학과 학생
sjy9728s@yonsei.ac.kr
jieun199624@yonsei.ac.kr
kkh115505@yonsei.ac.kr
jinhuijun@yonsei.ac.kr
†† 종신회원 : 연세대학교 컴퓨터과학과 교수(Yonsei Univ.)
sanghyun@yonsei.ac.kr
(Corresponding author임)

논문접수 : 2021년 9월 27일
(Received 27 September 2021)
논문수정 : 2022년 1월 13일
(Revised 13 January 2022)
심사완료 : 2022년 3월 23일
(Accepted 23 March 2022)

정보과학회 컴퓨팅의 실제 논문지 제28권 제6호(2022. 6)

for each parameter to draw higher data processing performance. Through the extracted configuration, we measured throughput for each persistence method and confirmed the performance improvement up to 45.9% over the default parameter setting.

Keywords: DBMS parameter tuning, Redis, machine learning, persistence method

1. 서 론

다양한 도시 문제를 해결하기 위한 스마트시티에서는 시민들의 일상 활동으로부터 방대한 양의 데이터가 생성되고 있다. 센서 및 GPS 등을 통해 수집된 데이터를 실시간으로 분석하여, 국민의 삶의 질을 높이고 도시 경쟁력을 강화할 수 있다[1,2]. 최근 실시간 데이터 생성 속도가 증가하여 우수한 데이터 처리 성능을 가진 서비스가 요구됨에 따라, 메모리 기반 데이터베이스(In-Memory Database)의 사용이 증가하고 있다. 메모리 기반 데이터베이스는 메인 메모리를 데이터 주 저장장치로 사용한다. 따라서 디스크 입출력이 빈번한 디스크 기반의 데이터베이스보다 응답 및 데이터 조회 속도가 빠른 장점이 있다.

메모리 기반 데이터베이스인 Redis[3]에서는 모든 데이터가 메모리에 상주하기 때문에 데이터 접근 지연이 적다. 또한, 각 데이터를 키-값 쌍의 형태로 저장하여 데이터 조회 속도가 빨라 실시간 데이터 처리가 요구되는 환경에서 사용되고 있다[4]. Redis는 데이터를 DRAM에 유지하기 때문에 휘발성으로 인한 데이터 유실의 위험이 있으며, 이를 방지하기 위한 지속성 방법을 제공한다. 지속성 방법은 현재 데이터셋에 대한 로그 레코드를 생성하여 데이터를 보존하지만, 데이터 처리 지연 및 메모리 사용량 증가와 같은 성능 저하가 동반한다.

Redis에서 제공하는 지속성 방법으로는 RDB(Redis Database)와 AOF(Append-Only File)가 있다. RDB는 현재 데이터셋에 대해 일정 간격으로 스냅샷(Snapshot)을 생성하는 방법이고, AOF는 데이터 적재, 수정, 삭제와 같이 데이터셋을 변경하는 명령에 대한 로그 레코드를 생성하여 로그 파일에 덧붙인다. 두 방법을 통하여 데이터 지속성을 높일 수 있지만, 디스크 접근을 요구하는 로깅 작업으로 인해 데이터 처리 성능 저하와 같은 추가적인 부하가 유발된다[5,6].

싱글 스레드 기반 프로그램인 Redis는 한 번에 하나의 명령만 실행할 수 있어, 데이터 처리 효율을 높이기 위해 다양한 백그라운드 작업을 수행한다. 메인 프로세스에서 특정 작업이 수행될 경우, 사용자로부터 요청된 명령어는 처리되지 못하고 지연된다. 따라서 특정 시간이 초과된 클라이언트의 연결을 해제하거나, 데이터 지속성 작업과 같이, 오랜 시간이 요구되는 작업들은 백그라운드로 처리한다. 그러나 이러한 노력에도 불구하고, Redis에서는 여전히 데이터 처리 성능 저하가 발생한다.

Redis에서 수행하는 지속성 방법과 부가적인 백그라운드 작업으로 인해 유발되는 부하는 Redis의 파라미터 값을 조절하여 완화시킬 수 있다. 서로 다른 워크로드에 대한 DBMS의 최적의 파라미터 값을 튜닝하는 작업은 데이터 처리 성능을 높이는 데에 중요한 단계다. 그러나 Redis의 수많은 파라미터가 가질 수 있는 값은 광범위하기 때문에, 전문가가 다양한 워크로드마다 최적의 파라미터 값을 찾는 데는 한계가 있다.

최근 데이터베이스의 성능에 높은 영향을 미치는 파라미터를 선별하기 위해 기계학습 기법을 사용한 연구가 활발히 진행되고 있다[7-9]. CDBTune[7]은 강화학습을 도입하여 클라우드 데이터베이스의 파라미터 구성을 학습하고 추천하는 최초의 end-to-end 튜닝 시스템을 제안했다. [8]은 RocksDB의 데이터 처리 성능을 극대화하기 위해 Bayesian Optimization[10]을 적용한 최적화 모델을 개발했다. OtterTune[9]은 기계학습 기법을 활용하여, 다양한 워크로드에서 분석한 결과를 기반으로 사용자가 요청한 워크로드에 대한 최적의 파라미터 조합을 찾는다. 저장소에 축적된 기존의 워크로드 데이터와 입력된 워크로드를 매핑하여 최적의 configuration을 추천하는 Gaussian Process[11] 기반의 방법을 제안하여 튜닝에 소요되는 시간과 자원을 감소시켰다. 그러나 해당 연구는 추출한 데이터 간의 선형적 관계만을 고려하여 최적의 파라미터를 선정하는 한계가 있다. 이를 개선하기 위해 본 연구에서는 OtterTune의 구조를 착안하여, 비선형적 기계학습 기법을 Redis에 적용한 RS-OtterTune(Redis Simplified OtterTune)을 제안한다. 본 모델은 비선형 회귀 기반의 기계학습 기법으로 RandomForest[12]와 XGBoost[13]를 사용한다. RS-OtterTune의 성능 향상을 검증하기 위해 Redis의 기본 성능과 선행 연구와의 비교 실험을 진행하였고, 최대 45.9%의 성능 향상을 확인하였다.

2. 모 델

본 연구는 Redis 파라미터 튜닝을 위해 비선형 기계학습 기법을 적용한 RS-OtterTune을 제안한다. RS-OtterTune은 그림 1과 같이 샘플 생성(Sample Generation), Metrics 단순화(Metrics Simplification), Knobs 랭킹(Knobs Ranking) 그리고 추천(Recommendation) 단계로 구성되며, 결과적으로 최적의 configuration을 도출한다. 이때, knob은 하나의 파라미터를 뜻하고, configuration은 knob들의 집합을 의미한다.

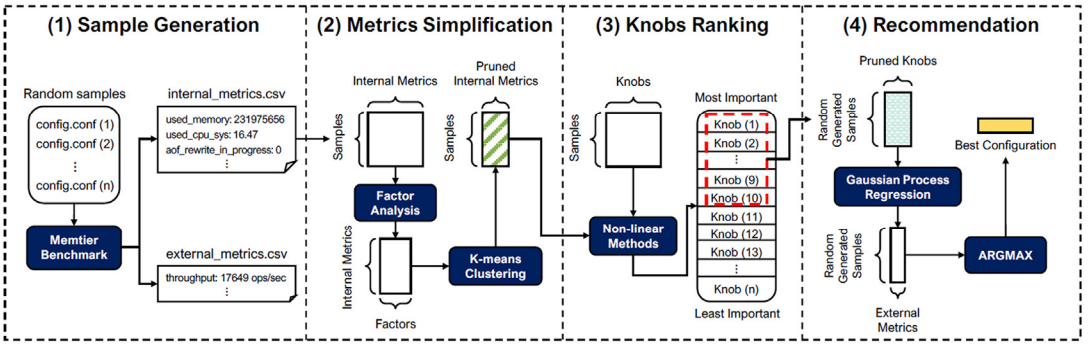

그림 1 RS-OtterTune 모델 구조
Fig. 1 RS-OtterTune Architecture

2.1 데이터 샘플 생성

공개되어 있는 Redis 워크로드 데이터셋이 없으므로, 본 연구는 훈련에 필요한 데이터 샘플 생성이 요구된다. 다양한 샘플을 얻기 위해 랜덤한 파라미터 값을 가지는 Redis configuration 파일들을 생성한다. 그다음, 각 파일들의 내용으로 설정된 Redis 서버에서 Memtier-Benchmark[14]를 통해 내부(Internal) metrics와 외부(External) metrics를 측정하여 각각의 csv 파일로 저장한다. 내부 metrics는 Redis 서버 정보와 메모리 사용량과 같은 통계값 등을 포함하고, 외부 metrics는 데이터 처리 성능을 평가하기 위한 단위 시간당 처리량(Throughput) 등을 포함한다. knob은 총 n개이며 내부 metrics는 (N, I), 외부 metrics는 $(N, 2)$의 크기를 가진다. 이때, N은 생성한 Redis configuration 파일의 개수를 의미하고, I는 내부 metrics의 지표의 개수를 나타낸다. 외부 metrics는 단위 시간당 처리량과 99번째 백분위수 지연 시간(Latency) 2개로 구성되어 있다.

2.2 Metrics 단순화

내부 metrics는 각 샘플마다 Redis 버전과 같은 동일한 값을 가지는 지표를 포함하여, 다양한 실행 결과값을 가진다. 내부 metrics 종류의 다양성은 추후 모델 훈련에 소요되는 시간을 증가시키므로, Metrics 단순화 단계에서 비슷한 특징을 가지는 metrics를 찾아 단순화시킨다. 먼저 인자 분석(Factor Analysis)을 통해 모든 내부 metrics 사이의 상관관계에 대한 분산을 찾는다. (N, I) 크기의 내부 metrics를 인자 분석을 통해 $(N, 5)$와 같이 5개 인자로 추출한다. 추출한 인자들에 K-평균 군집화(K-means Clustering)를 적용하여, 비슷한 성격을 갖는 metrics로 k개의 군집을 얻는다. 각 군집의 중심에 가장 가까운 metrics를 선택하여 k개의 pruned metrics로 단순화한다. Metrics 단순화를 통해 (N, I)의 내부 metrics를 (N, k)의 pruned metrics로 단순화하였다.

2.3 Knobs 랭킹

Knobs 랭킹 단계에서 RS-OtterTune은 Metric 단순화 단계에서 얻은 k개의 pruned metrics를 이용해 성능에 영향을 주는 knob들을 선정한다. 기계학습 기법을 통해 knob들의 기여도를 측정하여 정렬하고, 상위 m개의 knob들을 튜닝에 활용한다. 본 연구는 기계학습 기법으로 k개의 pruned metrics를 사용하여 각 외부 metric를 예측하는 비선형 회귀 모델 RandomForest와 XGBoost를 사용한다.

2.3.1 RandomForest

RandomForest는 훈련을 통해 생성된 여러 의사결정트리(Decision Tree)들로부터 분류를 취합하여, 결과를 예측 또는 회귀 분석을 수행하는 배깅(Bagging) 방법을 사용하는 앙상블 모델이다. RandomForest 회귀 모델은 각 특징의 불순도(Impurity)를 계산하고, 불순도평균감소량을 통해 분류에 대한 특징 기여도를 산출한다. 회귀 모델에서 상대적으로 중요도가 높은 변수를 추출해낼 수 있다. 또한, 대용량 데이터 처리에 용이하고, 과적합(Overfitting)을 방지하는 특징이 있다.

2.3.2 XGBoost

배깅 방법을 이용하는 RandomForest와 달리 XGBoost는 부스팅(Boosting) 방법을 사용하는 기계학습 앙상블 모델이다. 부스팅 방법은 약한 예측 모형들의 학습 에러에 가중치를 두고, 순차적으로 다음 학습 모델에 반영하여 강한 예측 모형을 만든다. 기존 부스팅 방법을 사용하는 알고리즘들은 계산량이 많고 과적합이 발생한다는 문제점이 있다. XGBoost는 병렬 처리를 이용하여 기존 모델들보다 빠르고, 과적합을 방지하기 위한 규제항이 포함되어 있다. 수식은 다음과 같다.

$$\hat{y} = \alpha \times tree_A + \beta \times tree_B + \gamma \times tree_C + \cdots \quad (1)$$

$$L = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \quad (2)$$

수식 (1)에서 $tree$는 트리 기반의 예측 모형이고, 각 예측모형에는 가중치(α, β, γ)를 곱한 뒤 더해 예측값 $\hat{y}$를 구한다. 목적 함수 L은 $\hat{y}$와 실제값 y 사이의 손실 함수와 정규화 항 Ω을 더함으로써 구한다. 이때, n은 데이터 샘플의 개수, K는 트리의 개수를 의미한다.

2.3.3 상위 Knobs 추출

RS-OtterTune은 RandomForest와 XGBoost를 통해 분석한 knob들을 성능에 대한 중요도를 기준으로 정렬한다. 이때, knob들은 Redis 성능과 관여가 깊을수록 상위에 위치하게 된다. 본 연구에서는 기여도가 높은 순으로 정렬되어 있는 knob들 중에서 상위 m개를 추출하여 최종 configuration 추천 단계에서 사용한다.

2.4 추천

RS-OtterTune의 추천 단계에서는 특정한 워크로드에 대하여 개선된 성능을 이끌어낼 수 있는 최적의 파라미터 조합을 추천하기 위해 GP(Gaussian Process) 회귀 모형을 사용한다. GP 모형에서 x는 Knobs 랭킹에서 얻은 pruned knobs이며, y는 외부 metrics으로 학습한다. GP 모형은 각 외부 metrics별로 독립적으로 구성한다. 그리고 pruned knobs을 기반으로 랜덤한 값을 가지는 다수의 Redis configuration 파일을 생성한 후, 학습된 GP 모델을 통해 데이터 처리 성능을 예측한다. 추출한 예측값들 중 최댓값을 가지는 Redis configuration 파일을 최종적으로 추천하게 된다.

3. 실험 및 결과 분석

3.1 실험 환경

본 논문에서는 서로 다른 두 가지 워크로드에 대한 데이터 샘플 생성과 다양한 기계학습 기법의 성능 비교 실험을 Google Cloud Platform(GCP)에서 진행하였다. 데이터 생성을 위한 32개, 파라미터 튜닝을 위한 2개의 GCP 인스턴스를 생성하였고, 시스템 실험 환경은 표 1과 같다. 모든 실험은 메모리 기반 데이터베이스 성능 측정을 위한 벤치마크인 Memtier-benchmark를 사용하였다.

본 실험은 RandomForest와 XGBoost를 모델에 적용하였을 때 성능을 비교하기 위해, 두 가지 워크로드에서 단위 시간당 처리량을 측정하였다. 워크로드는 데이터

표 1 시스템 실험 환경
Table 1 System Experimental Setting

OS		Ubuntu 16.04.7 LTS
CPU		Intel® Xeon® CPU @ 2.00GHz
RAM	Data Generation	DIMM 4G
	Tuning	DIMM 16G
Redis Version		5.0.2

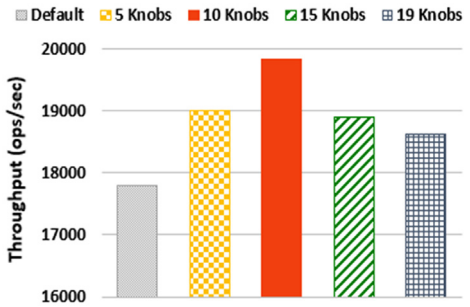

그림 2 튜닝하는 Knob 개수에 따른 데이터 처리 성능
Fig. 2 Throughput for Each Number of Tuning Knobs

적재만 수행하는 Write-Only, 데이터 조회와 적재 비율이 같은 Read-Write(1:1)에 대해, 지속성 방법으로 RDB와 AOF를 사용한 경우를 구분하여 진행하였다. 적재되는 1,000,000건의 키-값 데이터의 키의 크기는 16 B로 고정된 크기를 가지며, 값의 크기는 128 B로 설정하였다.

3.2 Knobs 랭킹 세부 실험

RS-OtterTune은 비선형 기계학습 기법을 통해 의미있는 knob을 분석 후 성능에 영향력 높은 knob들을 결정한다. 효율적으로 파라미터 튜닝을 수행하여 특정 워크로드에 대한 최적화된 configuration 파일을 추천하기 위해, 본 연구에서는 상위 knob 개수에 따른 비교 실험을 수행하였다.

그림 2는 튜닝하는 knob의 개수를 달리하여 도출한 각 configuration을 Redis에 적용하였을 때 데이터 처리 성능을 파라미터 튜닝을 수행하지 않은 Default와 비교한 실험 결과를 나타낸다. 해당 실험의 워크로드는 데이터 적재만 수행한 Write-Only의 경우이며, Redis는 지속성 기법으로 RDB 작업을 수행했다. 각 knob의 기여도를 측정하기 위한 기계학습 기법으로 XGBoost를 사용하였다.

그림 2에 나타나 있듯이, 튜닝하지 않은 Default의 경우 가장 낮은 성능을 보였다. 각 파라미터가 미치는 영향도를 분석하여 최적의 값을 도출한 모든 경우에서 성능 향상이 나타났으며, 의미 있는 knob들 중 상위 10개를 튜닝하였을 때 가장 높은 데이터 처리 성능이 측정되었다. 상위 knob의 개수가 10개를 초과하는 기점으로 Redis의 성능이 점차 감소하는 경향을 보였다. 이는 과도한 knob 튜닝으로 인하여 Redis 성능이 오히려 저하되었음을 의미한다.

3.3 결과 분석

그림 3은 Redis가 지속성 방법으로 AOF를 수행하고 데이터 적재 명령어를 처리할 때, XGBoost를 통해 파라미터 기여도를 평가한 결과이다. 각 파라미터의 중요도는 Gain 방법을 통해 계산된다[13]. 분석한 파라미터

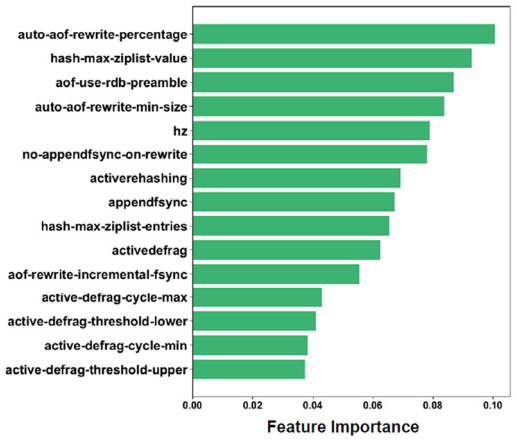

그림 3 Write-Only_AOF의 파라미터 중요도 분석
Fig. 3 Importance of Parameters for Write-Only_AOF

들을 중요도 순으로 정렬한 후, 상위 10개의 파라미터 값을 최적화하여 configuration 파일에 반영하였다.

파라미터 중요도 분석 및 정렬 결과, 가장 상위에 위치한 *auto-aof-rewrite-percentage*는 AOF 동작 시 수행하는 Rewrite 작업과 연관이 있다. Rewrite 작업은 현재 데이터셋을 구축하는 데 필요한 로그 레코드만을 남겨, 디스크에 작성하는 로그 파일의 크기를 감소시키기 위해 수행한다. 해당 작업은 로그 파일의 크기가 특정 임계값에 도달하면 최초로 동작하게 되는데, *auto-aof-rewrite-percentage*는 이 임계값을 결정하는 파라미터이다. Rewrite 작업이 동작하는 동안에는 메모리 사용량이 급격하게 증가하고 데이터 처리 성능이 낮아지는 현상이 발생한다[6]. 따라서, Rewrite 작업의 동작 시점을 결정하는 두 파라미터는 성능에 미치는 영향이 크기 때문에, 중요도에 따른 파라미터 정렬 결과에서 상위에 위치하게 된다.

두번째로 높은 순위에 위치한 파라미터는 *hash-max-ziplist-value*이다. Redis는 Hashes 구조로 데이터를 저장하며, Hashes는 내부적으로 두 가지 데이터 타입을 제공한다. 필드 개수가 적거나 또는 값의 길이가 작을 때는 Zip list를 사용하고, 필드 개수가 많거나 값의 길이가 클 때는 Hash Table을 사용한다. *hash-max-ziplist-value*는 Redis에서 데이터를 저장할 때, 필드 개수 또는 값의 길이에 따른 저장 기준을 결정하는 파라미터이다. Zip list와 Hash Table는 데이터 저장 시 수행하는 연산에 차이가 있으며, 메모리 사용량과 직접적으로 연관되기 때문에 Redis 성능에 직접적으로 영향을 미친다.

Redis의 파라미터 튜닝을 수행할 경우, Redis configuration 파일의 각 파라미터는 Redis의 성능을 개선시킬 수 있는 값으로 변경된다. *appendfsync*는 Redis가 지속성 기법으로 AOF를 사용할 때 변경되는 파라미터 중 하나이다. *appendfsync*는 메모리 버퍼의 로그 레코드들을 디스크의 로그 파일로 기록하기 위한 함수인 *fsync()* 함수의 호출 주기를 결정한다. 만약 *appendfsync* 파라미터의 값이 *always*일 경우, 데이터 적재, 수정, 삭제와 같이 데이터셋을 변경하는 명령이 요청될 때마다 *fsync()* 함수를 수행한다. 따라서 데이터 지속성은 완전히 보장되지만, 디스크 연산이 매우 빈번하게 수행되어, 심각한 성능 저하가 발생한다. *appendfsync* 파라미터의 기본값인 *everysec*은 *fsync()*함수를 1초마다 호출하고, 또 다른 옵션값인 *no*로 설정될 경우 해당 작업은 운영체제가 책임지게 되며, 리눅스 시스템은 30초마다 *fsync()* 함수를 호출한다. *appendfsync* 파라미터가 가질 수 있는 3가지 옵션값 중 디스크 연산이 가장 적게 수행되는 *no*로 설정되었을 때 데이터 처리 성능 저하가 가장 적으므로, 파라미터 튜닝 수행 시 해당 파라미터는 *no*로 설정된다.

Redis의 파라미터를 튜닝하지 않은 Default와 선형 기법인 Lasso를 사용한 OtterTune, 비선형 기법인 RandomForest와 XGBoost를 사용한 RS-OtterTune을 통해 튜닝한 Redis의 성능 비교 실험을 진행하였다. 그림 4는 Redis의 지속성 방법과 워크로드를 달리하여 측정한 데이터 처리 성능을 나타낸다. 파라미터를 튜닝하지 않은 기존 Redis보다 기계학습 기법을 이용하여 파라미터 값을 최적화한 경우 더 높은 데이터 처리 성능을 보였다.

의미 있는 knob들을 추출할 때 선형 기법인 Lasso를 활용한 OtterTune보다 비선형 기법을 이용한 RS-OtterTune의 경우 더 높은 성능을 이끌어낼 수 있다. 특히, 부스팅 방법인 XGBoost를 적용하여 knob들의 기여도를 분석하였을 경우, 배깅 방법인 RandomForest를 사용하였을 때보다 더 높은 성능을 보였다. 본 실험을 통해 knob과 metrics의 선형적인 관계를 얻는 것보다 비선형

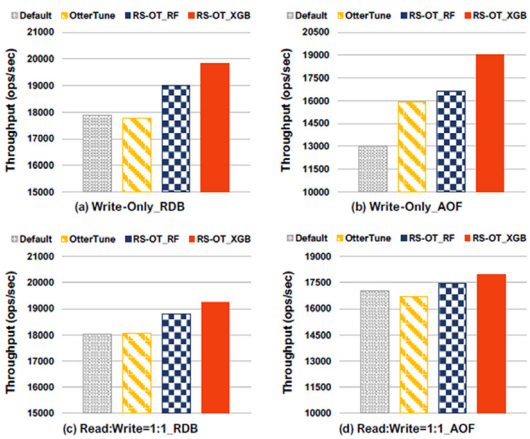

그림 4 워크로드 별 데이터 처리 성능
Fig. 4 Throughput for Each Workload

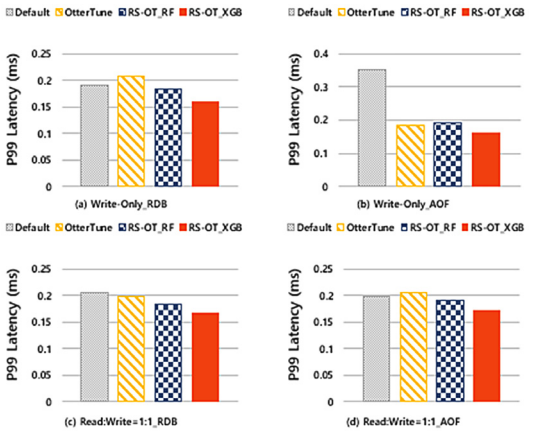

그림 5 워크로드 별 99번째 백분위수 지연 시간
Fig 5 99th Percentile Latency for Each Workload

성을 이용할 때 유용한 관계 정보를 추출함을 검증하였다.

Redis는 지속성 방법으로 AOF를 수행할 때 과도한 디스크 입출력으로 인한 데이터 처리 성능 저하와 메모리 사용량 급증이 발생한다. 특히, 데이터 적재가 빈번한 환경에서 다수의 로그 레코드를 생성하여 디스크의 파일에 작성하기 때문에, 그림 4(b)의 Default의 경우 가장 낮은 데이터 처리 성능을 보였다. 기계학습 기법을 이용하여 파라미터 튜닝 후 비교 실험을 수행한 결과, 해당 워크로드에서 XGBoost를 활용하여 튜닝하였을 때 Default보다 약 45.9% 향상되었다. 그림 5에 나타나 있듯이, 99번째 백분위수 지연 시간을 측정하였을 때도 데이터 처리 성능 실험과 유사한 결과를 보였으며, 데이터를 적재하는 환경에서 XGBoost를 활용하여 튜닝한 경우 Default보다 최대 54.7%의 개선 효과를 확인하였다.

4. 결 론

본 논문에서는 Redis에서 지속성 방법과 같은 백그라운드 작업으로 인해 발생하는 부하를 완화하기 위해, 기계학습을 이용하여 Redis의 파라미터 값을 최적화하는 연구를 수행하였다. DBMS 파라미터 튜닝을 위해 선형 기계학습 기법을 사용한 선행 연구와 달리, 비선형 기법을 활용한 RS-OtterTune을 통하여 의미 있는 metric과 파라미터를 분석하였다. 추출한 결과는 성능에 기여도가 높은 순으로 정렬된 knob들을 보여주며, 본 연구에서는 상위 10개의 knob을 최적화한 결과를 반영하여 configuration 파일을 추천하였다.

Redis에서 수행한 파라미터 튜닝의 효과를 평가하기 위해, 서로 다른 워크로드에서 데이터 처리 성능 비교 실험을 진행하였다. 튜닝을 수행하지 않은 Default보다 기계학습 기법을 수행하여 튜닝하였을 때 데이터 처리 성능이 향상되었고, 데이터 적재 시 Redis의 지속성 기법으로 AOF를 사용하는 경우 데이터 처리 성능은 최대 45.9%까지 개선되었다. 또한, 해당 실험을 통해 선형 기법보다 비선형 기법을 활용하여 영향력 있는 파라미터를 분석하였을 때, Redis의 성능 개선 정도가 더 높음을 확인하였다.

본 연구에서는 데이터 적재 명령어만을 수행할 때와 적재 및 조회 명령어가 1:1인 워크로드만을 고려하였다. 향후 연구에서는 데이터 적재와 조회, 수정 명령어의 비율을 조절하여, Redis가 다양한 워크로드에서 향상된 성능을 이끌어내기 위해 최적의 파라미터 조합을 도출할 수 있는 모델을 개발하고자 한다.

References

[1] M. M. Rathore, A. Paul, A. Ahmad, N. Chilamkurti, W.-H. Hong, and H. Seo, "Real-time secure communication for Smart City in high-speed Big Data environment," *Future Generation Computer Systems*, Vol. 83, pp. 638-652, Jun. 2018.

[2] A. Sharif, J. Li, M. Khalil, R. Kumar, M. I. Sharif, and A. Sharif, "Internet of things－smart traffic management system for smart cities using big data analytics," *Proc. of the 2017 14th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, IEEE, pp. 281-284, 2017.

[3] Redis. [Online]. Available: https://redis.io

[4] 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, Sep. 2014.

[5] H. Sung, M. Jin, M. Shin, H. Roh, W. Choi, and S. Park, "LESS: Logging Exploiting SnapShot," *Proc. of the 2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*, IEEE, pp. 1-4, 2019.

[6] J. Seo, H. Sung, W. Choi, and S. Park, "A Study on Improvement of Log File Reconstruction Overhead Utilizing Data Parallelism," *Database Research*, Vol. 36, No. 3, pp. 56-75, Dec. 2020. (in Korean)

[7] J. Zhang, Y. Liu, K. Zhou, G. Li, Z. Xiao, B. Cheng, J. Xing, Y. Wang, T. Cheng, L. Liu et al., "An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning," *Proc. of the 2019 International Conference on Management of Data*, pp. 415-432, 2019.

[8] S. Alabed and E. Yoneki, "High-Dimensional Bayesian Optimization with Multi-Task Learning for RocksDB," *Proc. of the 1st Workshop on Machine Learning and Systems*, pp. 111-119, 2021.

[9] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang,

"Automatic Database Management System Tuning Through Large-scale Machine Learning," *Proc. of the 2017 ACM International Conference on Management of Data*, pp. 1009-1024, 2017.
[10] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian Optimization of Machine Learning Algorithms," *Advances in Neural Information Processing Systems*, Vol. 25, 2012.
[11] C. E. Rasmussen, "Gaussian Processes in Machine Learning," *Summer School on Machine Learning*, Springer, pp. 63-71, 2003.
[12] L. Breiman, "Random Forests," *Machine Learning*, Vol. 45, No. 1, pp. 5-32, Oct. 2001.
[13] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," *Proc. of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785-794, 2016.
[14] Memtier-Benchmark. [Online]. Available: https://github.com/RedisLabs/memtier_benchmark

서 주 연
2020년 건국대학교 컴퓨터공학과(학사)
2022년 연세대학교 컴퓨터과학과(석사)
관심분야는 데이터베이스, 기계학습

이 지 은
2019년 인천대학교 컴퓨터공학부(학사)
2019년~현재 연세대학교 컴퓨터과학과 석박사통합과정. 관심분야는 데이터베이스, 기계학습

김 경 훈
2020년 경희대학교 응용수학과, 컴퓨터공학과(학사). 2022년 연세대학교 컴퓨터과학과(석사). 관심분야는 자연언어처리, 강화학습

JIN HUIJUN
2018년 연변과학기술대학 컴퓨터과학과(학사). 2022년 연세대학교 컴퓨터과학과(석사). 2022년~현재 연세대학교 컴퓨터과학과 석사 후 연구원. 관심분야는 데이터베이스 튜닝, 기계학습

박 상 현
1989년 서울대학교 컴퓨터공학과 졸업(학사). 1991년 서울대학교 대학원 컴퓨터공학과(공학석사). 2001년 UCLA 대학원 컴퓨터과학과(공학박사). 1991년~1996년 대우통신 연구원. 2001년~2002년 IBM T. J. Watson Research Center PostDoctoral Fellow. 2002년~2003년 포항공과대학교 컴퓨터공학과 조교수. 2003년~2006년 연세대학교 컴퓨터과학과 조교수. 2006년~2011년 연세대학교 컴퓨터과학과 부교수. 2011년~현재 연세대학교 컴퓨터과학과 교수. 관심분야는 데이터베이스, 데이터 마이닝, 바이오인포매틱스, 빅데이터 마이닝 & 기계 학습