CGPTuner: a Contextual Gaussian Process Bandit Approach for the Automatic Tuning of IT Configurations Under Varying Workload Conditions

연세대학교 컴퓨터과학과 김환희

2023년 8월

과제명: IoT 환경을 위한 고성능 플래시 메모리 스토리지 기반 인메모리 분산 DBMS 연구개발

과제번호: 2017-0-00477

과학기술정보통신부
Ministry of Science and ICT

연세대학교
YONSEI UNIVERSITY

정보통신기술진흥센터
Institute for Information & communications Technology Promotion

Contents

Abstract

◆ **목적**

- **Historical knowledge base**에 의존하지 않고도 **빠르게 well-performing되는 configuration**을 찾고 **workload variation에 맞게 적응**하는 **DBMS auto-tuning**을 위한 새로운 튜닝 접근법을 제시

- **Cassandra와 MongoDB DBMS**를 **사용**하여 제안된 접근법을 평가하고,
  이를 다양한 IT application에서 사용할 수 있는지 확인

- **응답 시간(response time)**을 늘리지 않고 **메모리 소비를 최소화**하려고 노력

Abstract

◆ **목적**

- DBMS는 **Java 가상 머신(JVM)**이나 **운영 체제(OS)**와 같은 여러 계층으로 구성된 **복잡한 IT Stack** 위에 위치함

- 각 계층에는 최종적인 DBMS의 동작에 영향을 주는 **tunable parameters**이 있고,
  DBMS의 전체 성능 잠재력을 활용하려면 **전체 IT Stack**을 함께 tuning해야 함

- 불행히도 DBMS, OS, 하드웨어와 같은 특정 조합에 대한 optimization configuration을 찾을 수 없음

- Knowledge base에 의존하지 않고도 DBMS Workload 같은 **textual 작업에 특화된 머신러닝 알고리즘인**
  Contextual Gaussian Process Bandit Optimization을 기반으로 한 automatic configuration Tuner인 **CGPTuner** 제안

Abstract

◆ **목적**

- **CGPBO**는 **Bayesian Optimization framework**에 대한 **contextual extension**임
- **BO**는 이미 **autotuning problem에 성공적**으로 적용되었음

- **CGPTuner**는 **IT Stack의 multiple layer**와 **현재 workload**를 고려하여 IT 시스템의 **configuration**을 성공적으로 조정하며, 더 중요한 것은 **knowledge base에 의존하지 않음**
- Multiple layer인 경우 실질적으로 **knowledge base를 수집하는 것은 거의 불가능** 하기 때문

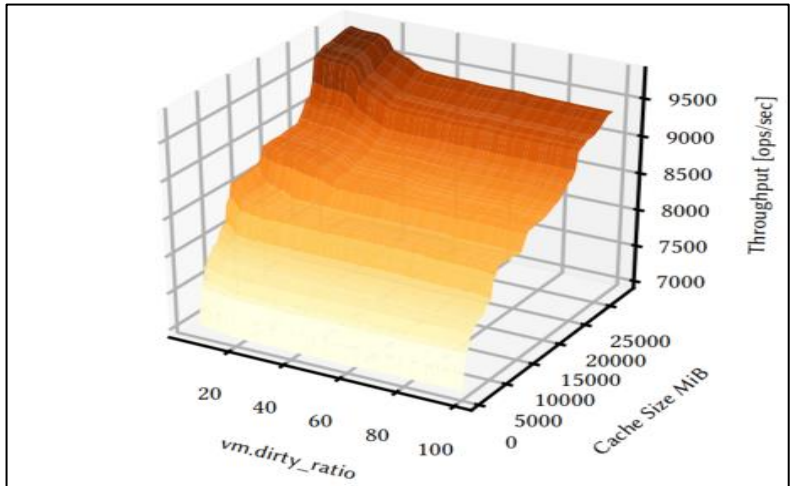
이후 **autotuning problem을 분석**하고 tuning 알고리즘에 대해 설명 후 evaluation methodology를 검토하고 논의할 예정

Abstract

◆ Motivation

- DBMS configuration을 tuning하는 것은 매우 까다로움 : **tunable parameters interact in complex ways, enormous search space, depends on the workload**
- DBMS는 **운영체제 위에 위치**해 있으며, 자신만의 **tunable parameter**을 가지고 있음

- 이러한 parameter의 영향을 강조하기 위해 **YCSB load injector**을 사용하여 **MongoDB 및 Cassandra DBMS**에서 일련의 실험을 수행
- **DBMS 처리량**을 측정하면서 tunable parameter 값들을 수정
- ***MongoDB 캐시 크기*** 와 ***리눅스 커널 vm.dirty_ratio*** 파라미터를 수정
- 후자 파라미터를 올바르게 설정하면 DBMS 성능이 크게 향상



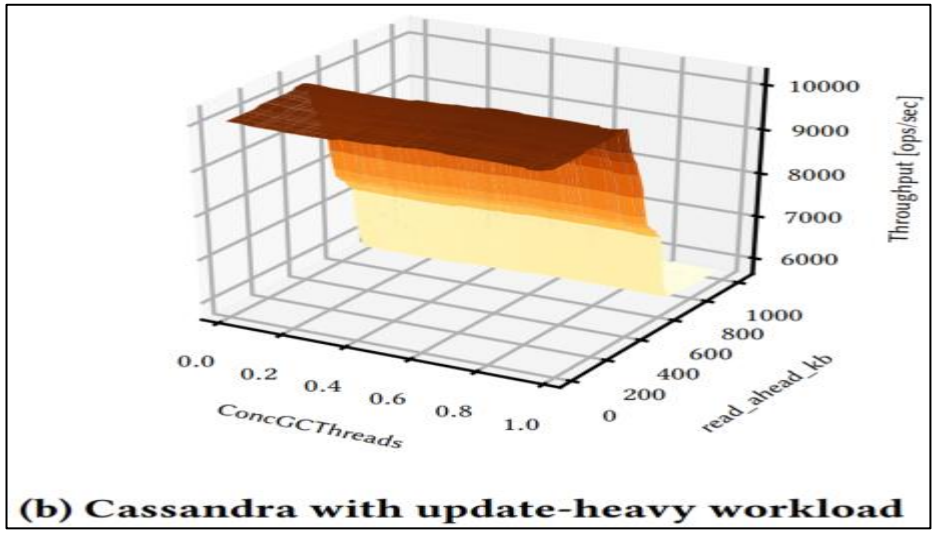
(a) MongoDB with update-heavy workload

Abstract

◆ Motivation

- **JAVA로 작성된 Cassandra**를 사용
- **Java virtual machine**은 **tunable parameter**을 가지고 있으며, ***garbage collection threads*** 의 수와 ***read_ahead_kb*** **리눅스 파라미터**를 **tuning**

- **read-ahead는 Cassandra성능에 큰 영향**을 미치며, 적절하지 않은 값을 선택하면 성능이 무너짐

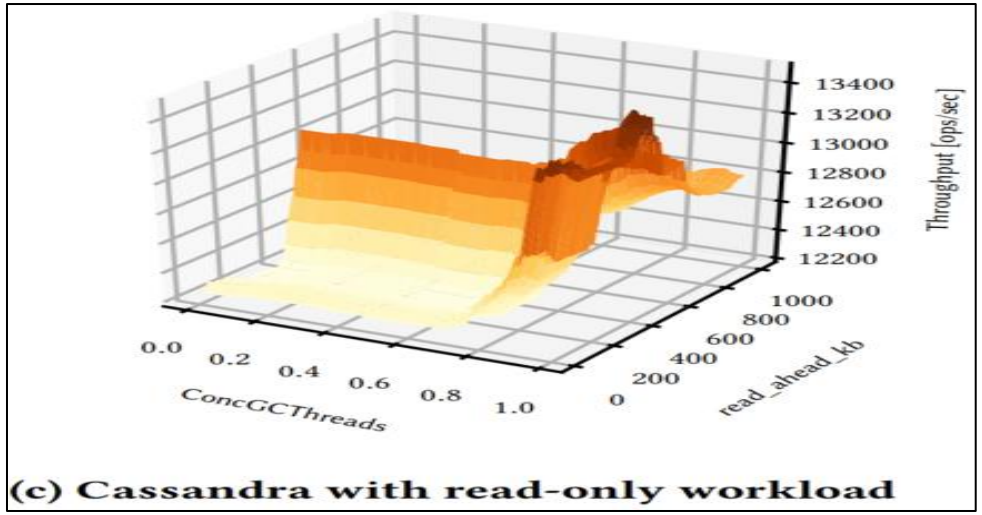


(b) Cassandra with update-heavy workload

Abstract

◆ Motivation

- **다른 YCSB 워크로드**를 사용하여 **동일한 두 parameter을 수정 ➔** ***update-heavy workload***에서 ***read-only workload***



(c) Cassandra with read-only workload

이 예제들은 DBMS튜닝 시 **전체 IT Stack**과 현재 **Workload**까지 고려해야 할 필요성을 보여줌

Abstract

◆ Motivation

- 전체 IT Stack을 Tuning하면서 **DBMS만 고려**한다면 **시스템의 성능이 떨어질 수 있음**
- 하지만 최적화할 target performance metric 선택만 올바르게 한다면 이 문제를 피할 수 있음

- 기존 **DBMS automatic configuration tuning**
  1. **iTuned**
  2. **OtterTune**
  3. **Bestconfig**

Abstract

◆ **Related work**

- **iTuned** : GP와 DBMS 성능에 대한 response surface를 생성하여 테스트할 **next configuration** 모델을 선택하는 방식

  하지만 **각 workload**에 대해 **다른 response surface**이 구축되고 잠재적으로 유용한 정보를 공유하지 않음

- **Ottertune** : **과거 경험과 새로운 정보**를 활용하여 **DBMS configuration tuning**. **지도/비지도 학습을 조합**하여
  (1) 가장 영향력 있는 knob 선택, (2) 이전 워크로드의 경험과 unseen DBMS workload 매핑,
  (3) knob setting recommendation

- **Bestconfig** : 주어진 애플리케이션 workload과 배포된 시스템의 자원 제한 내에서 최적의 configuration setting을 자동으로 찾는 자동 튜닝 시스템입니다. 이 시스템은 반복적 인 샘플링 전략을 기반으로 작동
  ➔ 광범위한 이전 실험이 필요, 모든 knob을 고려하지만 IT Stack의 더 많은 계층을 고려함에 따라 차원이 기하급수적으로 증가함

CGPTuner

◆ CGP Tuner

- **온라인 방식으로 진행**되고, **knowledge base에 의존하지 않는** 솔루션

- Tuner는 탐색공간을 탐색하고 동시에 **특정 시점까지 수집한 지식을 최대한 활용**하며 튜닝을 진행

- **BO** 기반의 솔루션 (exploration-exploitation trade-off을 다루기 위해 설계)

CGPTuner

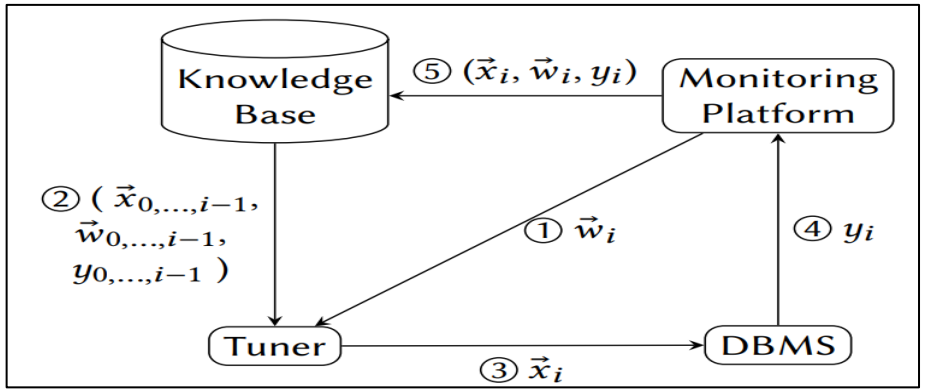
◆ CGP Tuner

- 최적화 프로세스 목표 : **구성공간 X**에서 **구성 벡터 $\vec{x}$**를 찾고 IT system에 적용하여 특정 성능 지표 y ∈ R를 최적화 하는 것.
- 시스템에 노출된 **특정 워크로드 $\vec{w}$** ∈ W를 고려하여 **$\vec{x}$를 선택**

W는 가능한 워크로드의 **공간**이며 **$\vec{w}$**는 특정 workload 특성화 방법론에서 제공되는 workload의 설명

시간 t에서 최적화 알고리즘은 현재 **workload $\vec{w}_t$**에 맞춰진 후보 **configuration $\vec{x}_t$**를 제안

workload $\vec{w}_t$ 에서 **$\vec{x}_t$**를 적용하면 **성능 측정치 y_t**



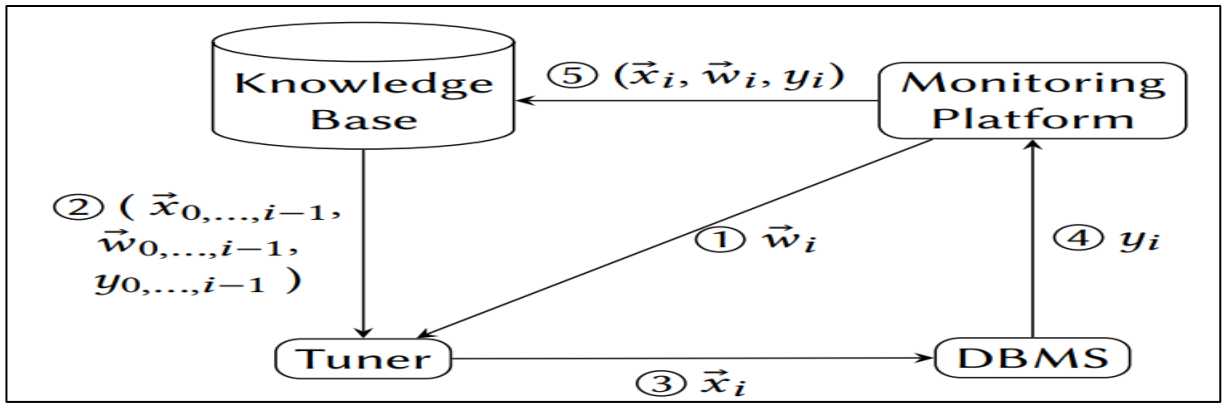
CGPTuner

◆ CGP Tuner

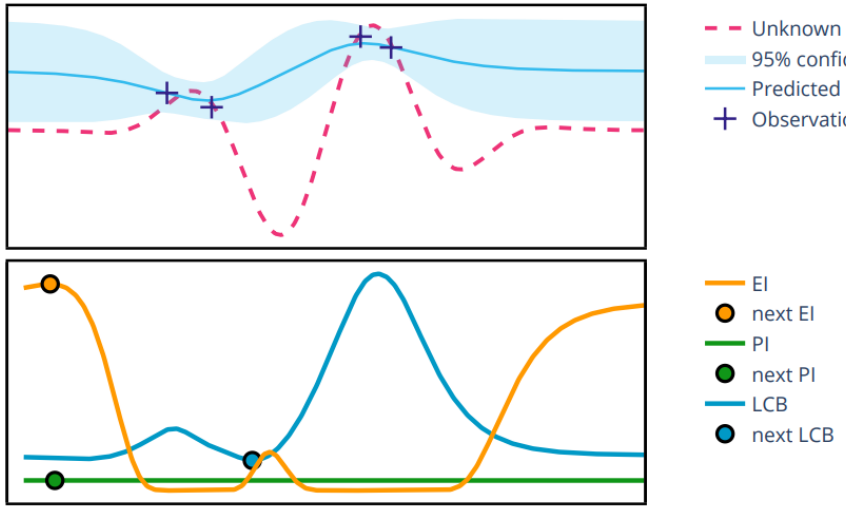
- 튜너는 이전 실험의 **지식 기반(KB)**에 액세스하고 **KB와 현재 workload**를 사용하여 **후보 configuration 벡터 $\vec{x}_0$**를 제안
- 시스템에 **configuration을 적용**하고 **관련 성능 y_0**를 측정
- 이 시점에서 튜닝 프로세스의 첫 번째 반복이 끝나고 KB에 얻은 정보를 저장
- 이후 프로세스를 반복 :

  새로운 workload **$\vec{w}_1$**을 측정하고 이전 결과를 고려하여 새 configuration **$\vec{x}_1$**을 얻고, 성능 **y1**이 됨
- 반복 i에서 튜너는 이전 반복의 모든 정보를 활용할 수 있음($\vec{x}_{0,...,i-1}$, $\vec{w}_{0,...,i-1}$, $y_{0,...,i-1}$)

  (그러나 다른 지식은 필요하지 않음)



◆ BO(Bayesian Optimization)



❖ EI (Expected Improvement)
Exploration 과 Exploitation 방법을 모두 일정 수준 포함하도록 설계

• **대략적인 수행 과정**

- 위 그림의 **분홍색 점선**은, 우리가 찾으려고 하는 **목적함수 f(x)** 를 나타냄
- **파란색 선**은, 지금까지 관측한 데이터를 바탕으로 우리가 **예측한 prediction function** 을 의미
- 파란색 선 주변에 있는 **파란 영역**은, **목적함수 f(x) 가 존재할 만한 confidence bound(function의 variance)** 를 의미

- 밑에 있는 **EI은**, **Acquisition function** 을 의미하며, **다음 입력 값 후보 추천** 시 사용됨
  - Acquisition function 값이 **컸던 지점**을 확인하고, 해당 지점의 hyperparameter 를 **다음 입력 값**으로 사용
- hyperparamter에 따라 prediction function 을 계속 update하면, **prediction function과 목적 함수 f(x) 가 흡사**해짐
- 관측한 지점 중 **best point**을 **argmax f(x)**로 선택

◆ BO(Bayesian Optimization)

- **자세한 수행 과정**

1. **입력값, 목적 함수 및 그 외 설정값**들을 정의합니다.
   - 입력값 x : **learning rate**
   - 목적 함수 f(x) : 설정한 입력값을 적용해 학습한, 딥러닝 모델의 성능 결과 수치(e.g. 정확도)
   - 입력값 x 의 탐색 대상 구간 : (a, b)
   - 입력값-함수결과값 점들의 개수 : n
   - 조사할 입력값-함수결과값 점들의 개수 : N

2. 설정한 탐색 대상 구간 (a, b) 내에서 처음 n개의 입력값들을 랜덤하게 샘플링하여 선택합니다.

3. 선택한 n개의 입력값 x_1, x_2, …, x_n을 각각 모델의 **hyperparameter(learning rate)**로 설정하여 딥러닝 모델을 학습한 뒤, 학습이 완료된 모델의 성능 결과 수치를 계산합니다.
   - 이들을 각각 함수결과값 $f(x_1)$, $f(x_2)$, …, $f(x_n)$ 으로 간주합니다.

4. 입력값-함수결과값 점들의 모음 $(x_1, f(x_1))$, $(x_2, f(x_2))$, …, $(x_n, f(x_n))$ 에 대하여 **Surrogate Model 로 확률적 추정**을 수행합니다.

◆ BO(Bayesian Optimization)

5. 조사된 입력값-함수결과값 점들이 총 N개에 도달할 때까지, 아래의 과정을 반복적으로 수행합니다.

- { 기존 입력값-함수결과값 점들의 모음 $(x_1, f(x_1)),(x_2, f(x_2)), \ldots, (x_t, f(x_t))$ 에 대한 **Surrogate Model 의 확률적 추정 결과**를 바탕으로, 입력값 구간 (a, b) 내에서의 **EI** 의 값을 계산하고, 그 값이 가장 큰 점을 **다음 입력값 후보 x_1** 로 선정합니다.
- **다음 입력값 후보 x_1 를 hyperparameter 로 설정**하여 딥러닝 모델을 학습한 뒤, 학습이 완료된 모델의 성능 결과 수치를 계산하고, 이를 $f(x_1)$ 값으로 간주합니다.
- 새로운 점 $(x_2, f(x_2))$ 을 기존 입력값-함수결과값 점들의 모음에 추가하고, 갱신된 점들의 모음에 대하여 **Surrogate Model 로 확률적 추정**을 다시 수행합니다. }

6. 총 N개의 입력값-함수결과값 점들에 대하여 확률적으로 추정된 목적 함수 결과물을 바탕으로, **평균 함수 μ(x) 을 최대로 만드는 최적해를 최종 선택**합니다. 추후 **해당값을 hyperparameter 로 사용하여 딥러닝 모델을 학습**하면, **일반화 성능이 극대화된 모델**을 얻을 수 있습니다.

◆ BO(Bayesian Optimization)

DBMS 자동 튜닝 문제와 거의 동일하지만, 먼저 워크로드 dependencies는 고려하지 않음

BO는 방정식의 최적화 문제를 해결하기 위한 **sequential model-based approach**

BO의 두 가지 핵심 구성 요소 : **surrogate model, acquisition function**

Surrogate model로 가우시안 프로세스(GP)를 중점적으로 사용하며, 이는 BO에서 인기 있는 선택 사항

Acquisition function에 대해 **GP-Hedge** 접근 방식을 따름

특정 AF에 집중하는 대신 **Online multi-armed bandit strategy**에 의해 관리되는 AF 포트폴리오를 채택

튜닝 반복마다 여러 다른 **AF**을 계산하고 이전 성능에 따라 점차 가장 좋은 것을 선택하는 것

CGPTuner

◆ Contextual Gaussian Process Bandit Optimization

지금까지는 **성능함수 f가 $\vec{x}$만으로 표현될 수 있다고 가정**

하지만 DBMS성능은 시스템에 노출된 워크로드 $\vec{w}$와 같은 다른 비 제어 변수에도 의존

BO는 이러한 상황을 처리하기 위해 확장

핵심 아이디어 : 최적화하려는 상관된 **함수 $f_{\vec{w}}(\vec{x})$가 여러 개 있다는 것**

Configuration space X와 Workload space W에 대한 작동하는 새로운 커널 함수를 정의

$$k((\vec{x}, \vec{w}), (\vec{x'}, \vec{w'})) = k(\vec{x}, \vec{x'}) + k(\vec{w}, \vec{w'})$$

$f_{\vec{w}}$ 는 **workload 간의 전반적인 추세**를 모델링하는 반면,

$f_{\vec{x}}$ **는 configuration-specific deviation를 모델링**

◆ 워크로드에 맞춘 사전 함수 매칭(Matching prior Function to Workload)

BO는 회귀 모델의 예측값과 예측 불확실성으로 부터 계산된 **acquisition함수**를 최적화하는 점을 반복적으로 제안

하지만 회귀모델은 GP로 관찰된 값과 사전분포를 결합하여 예측을 유도

이전에 관찰된 것과 **매우 다른 구성의 값을 예측하도록 요청**하면 **prior 분포에 의존**할 것임

➔ GP는 unknown configuration이 DBMS를 망칠 것으로 예측

(BO는 불확실한 영역의 탐색을 피하려는데 영향을 줌)

따라서 관련 사전 분포를 쉽게 얻으려면 **관측된 데이터를 표준화하는 것이 일반적**

$$NPI(\vec{x},\vec{w}) = \frac{f(\vec{x}_0,\vec{w})-f(\vec{x},\vec{w})}{f(\vec{x}_0,\vec{w})-f(\vec{x}^{+}_{\vec{w}},\vec{w})}$$

$\vec{x}$는 평가 중인 configuration, $\vec{x}_0$는 vendor default configuration이고, $\vec{w}$는 $\vec{x}$를 평가하고 있는 workload

$f(\vec{x},\vec{w})$은 configuration $\vec{x}$이 workload $\vec{w}$에서 얻은 성능 측정값이며, **$\vec{x}^{+}_{\vec{w}}$는 workload $\vec{w}$ tuning 중에 지금까지 찾은 optimization configuration**입니다.

CGPTuner

◆ 워크로드에 맞춘 사전 함수 매칭(Matching prior Function to Workload)

$x^{\rightarrow +}_{w^{\rightarrow}}$ **는 최적화를 진행하며 더 나은 configuration을 발견할 때**마다 변경

이러한 이유로 **각 반복 시**에 **과거 값들을 다시 정규화**해야 함

따라서 $NPI\ (\vec{x},\vec{w})$는 **configuration** $\vec{x}$가 **workload** $\vec{w}$에 대해 얼마나 최적인지를 측정

NPI 가 **0이면** workload $\vec{w}$에서 configuration $\vec{x}$의 성능이 **기준과 정확히 같다는 것**을 의미하고,

NPI 가 **1이면** configuration $\vec{x}$이 workload $\vec{w}$에 대해 지금까지 찾은 **최적의 것**임을 의미

CGPTuner

◆ 워크로드에 맞춘 사전 함수 매칭

마지막으로, DBMS configuration의 auto tuning을 위해 제안하는

알고리즘인 **CGPTuner**의 의사코드

최적화 과정을 온라인으로 계속 진행하므로 종료 기준을

제공하지는 않음

CGPTuner는 **BO 기법**이므로 최적 configuration을 찾으면 수렴

BO의 강력한 이론적 보장은 configuration을 새로운 workload에 적응시키고

explore과 exploit 사이에서 올바른 균형을 유지하면서 빠르게 최적 솔루션으로

수렴하도록 설정 공정을 계속 진행할 수 있음

Algorithm 1: CGPTuner.

input : A number n of starting random configurations

for $i \leftarrow 0$ **to** n **do**
  measure current workload $\vec{w}_i$;
  select a random configuration $\vec{x}_i$;
  apply $\vec{x}_i$ to the system under test;
  measure performance score y_i;
  store $\vec{x}_i, \vec{w}_i, y_i$ in the KB;
end
while *True* **do**
  get previous data from KB: $D_i = \{(\vec{x}_j, \vec{w}_j, y_j)\}_{j=0}^{i-1}$;
  foreach *oberseved workload* $\vec{w} \in KB$ **do**
    $\vec{x}^+_{\vec{w}} = \arg\max y_j, (\vec{x}_j, \vec{w}, y_j) \in KB$;
  end
  normalize previous data: $\tilde{NPI}_j = \frac{f(\vec{x}_0, \vec{w}_j) - f(\vec{x}_j, \vec{w}_j)}{f(\vec{x}_0, \vec{w}_j) - f(\vec{x}^+_{\vec{w}_j}, \vec{w}_j)}$;
  update the CGP with $\tilde{D}_i = \{\vec{x}_j, \vec{w}_j, \tilde{NPI}_j\}$;
  measure current workload $\vec{w}_i$;
  optimise acquisition function: $\vec{x}_i = \max_x a(\vec{x}, \vec{w}_i)$;
  apply $\vec{x}_i$ to the system under test;
  measure performance score y_i;
  store $\vec{x}_i, \vec{w}_i, y_i$ in the KB;
  $i++$;
end

Evaluation

◆ 실험 평가

MongoDB 4.0.3과 Cassandra 3.11.4 DBMS를 사용

Amazon EC2에서 실험을 진행

두개의 인스턴스 - YCSB 0.15.0, vCPU와 4GB RAM, c5.large EC2

- DBMS를 실행, vCPU, 30GB RAM, NVMe SSD 스토리지가 있는 i3.xlarge인스턴스

MongoDB : 15개

Cassandra : 24개

이 파라미터들은 수동으로 선택했음

Layer	Parameter
MongoDB	wiredTigerCacheSizeGB
MongoDB	eviction_dirty_target
MongoDB	eviction_dirty_trigger
MongoDB	syncdelay
OS	sched_latency_ns
OS	sched_migration_cost_ns
OS	vm.dirty_background_ratio
OS	vm.dirty_ratio
OS	vm.min_free_kbytes
OS	vm.vfs_cache_pressure
OS	Network RFS
OS	Storage noatime
OS	Storage nr_requests
OS	Storage scheduler
OS	Storage read_ahead_kb

Layer	Parameter
Cassandra	commitlog_compression
Cassandra	commitlog_segment_size_in_mb
Cassandra	commitlog_sync_period_in_ms
Cassandra	Compaction Strategy
Cassandra	compaction_throughput_mb_per_sec
Cassandra	concurrent_compactors
Cassandra	concurrent_reads
Cassandra	concurrent_writes
Cassandra	file_cache_size_in_mb
Cassandra	memtable_cleanup_threshold
JVM	CMSInitiatingOccupancyFraction
JVM	ConcGCThreads
JVM	GC Type
JVM	Xmx (max heap size)
JVM	MaxTenuringThreshold
JVM	NewRatio
JVM	ParallelGCThreads
JVM	SurvivorRatio
OS	CPUSchedNrMigrate
OS	MemoryTransparentHugepageEnabled
OS	MemoryVmDirtyExpire
OS	NetworkNetIpv4TcpMaxSynBacklog
OS	Storage scheduler
OS	Storage read_ahead_kb

Evaluation

◆ 실험 평가

Cassandra와 MongoDB는 모두 **YCSB 기본 workload** 중 세 가지를 선택합니다:

(a) 읽기와 쓰기 작업에서 50/50 비율로 업데이트가 많은 workload,

(b) 95/5 읽기/쓰기 비율로 읽기 위주 workload,

(c) 읽기 전용 workload

모든 workload는 직후 분포를 사용. 또한 YCSB 스레드 수를 10에서 90으로 늘림

YCSB를 사용하여 약 30GB 크기의 데이터베이스를 얻을 수 있는 30,000,000개의 레코드를 생성

Evaluation

◆ **실험 평가**

응답시간 R과 **메모리소비 M**을 최소화하는 문제

CGP 및 **다른 튜너**는 스칼라 값 최적화에 적합하므로 제약 최적화에서 일반적인 방법을 사용하여

제약 조건을 패널티 항으로 처리

- 따라서 **두 DBMS** 모두 **다음 함수를 최소화함**

M[MiB] + $\sigma \cdot R$[ms] (여기서 σ는 패널티 계수), 대부분의 실험에서 $\sigma = 10^3$를 사용

σ선택은 측정 단위와 제약 조건의 중요도에 따라 다름

R은 **ms**단위로 M은 **MiB**단위로 측정

완전성을 위해, $\sigma = 10^1$, $\sigma = 10^4$ 및 $\sigma = 10^5$ 인 실험을 추가로 진행

MongoDB에서는 ***wiredTigerCacheSizeGB*** 파라미터를 사용하여 M 측정

Cassandra에서는 ***file_cache_size_in_mb*** 와 ***JVM 최대 힙 크기(Xmx)의 합계*** 를 사용

Evaluation

◆ 실험 평가 - Data Collection and DBMS Models

DBMS의 configuration space를 탐색하는 다양한 실험을 수행하고 다양한 성능 메트릭을 수집

수집한 데이터를 사용하여 검색 공간 내의 모든 가능한 configuration의 성능을 예측하기 위해

랜덤 포레스트 회귀 모델을 구축

이를 통해 IT 시스템 성능의 모델을 만들고, 알고리즘 평가를 위한 튜닝 작업의 대상으로 사용

configuration test를 수행하기 전에 DBMS를 다시 시작하고 데이터베이스를 원래 버전으로 복원하여

다른 configuration test간의 괴리를 방지

실험은 45분 동안 진행되며, 처음 15분과 마지막 1분은 버리고 평가 기간 동안 평균 처리량과 응답 시간을 계산

Dataset	Tunable params	Workload params	Samples
MongoDB	15	2	4219
Cassandra	24	2	3728

❖ 모든 데이터세트에 대한 매개 변수 수와 수집 된 포인트 수

Evaluation

◆ 실험 평가 – 모델 정확도

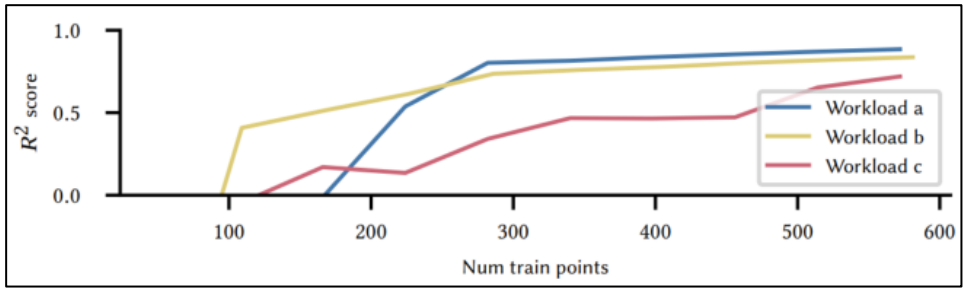
홀드아웃(holdout) 검증 접근법을 이용

측정을 두 세트로 나눔

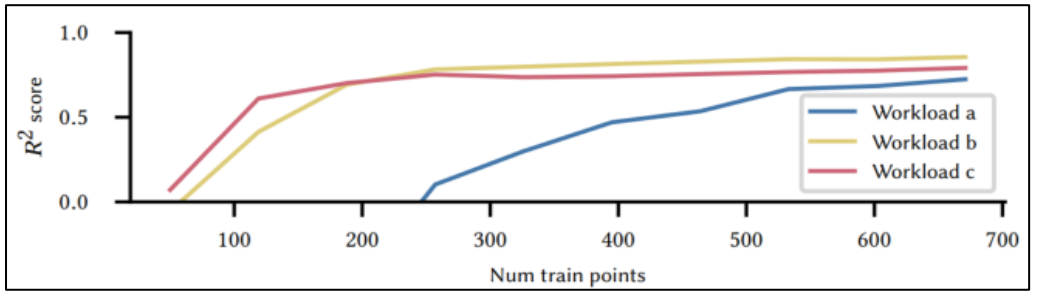
1 : 수집된 데이터의 25%를 포함하고 **테스트 세트**로 사용

2 : 나머지 데이터를 포함하고 **학습 세트**로 사용

이 후 학습 세트의 점진적으로 더 큰 부분집합을 고려하고 테스트세트에서 평가하는 다양한 회귀 분석을 학습하는데 사용



Cassandra



Mongo DB

Evaluation

◆ 실험 평가 – 평가 지표

다양한 튜너를 비교하기 위해 사용하는 지표

기술적 제약을 고려하지 않고, 자동 튜너를 실행하는 두 가지 방법

프로덕션 시스템에서 직접 작동하게 함(**온라인**) – **누적 보상 (CR)**

테스트 환경에서 시스템을 복제(**오프라인**) – **반복적 최적값 (IB)**

DBMS모델을 사용하면 각 workload 조건에 대한 최적 configuration을 알고 있음

튜너가 실제 최적점에 얼마나 가까운지 확인 할 수 있음

이 정보를 사용하여 정규화 된 NPI지표 도출

$$NPI(i)=\frac{achieved\ PI}{potential\ PI}=\frac{y_0-y_i}{y_0-y^*}=\frac{f(\vec{x}_0,\vec{w}_i)-f(\vec{x}_i,w_i)}{f(\vec{x}_0,\vec{w}_i)-f(\vec{x}_{\vec{w}_i}^{\ *},\vec{w}_i)}$$

$\vec{x}_0$ 은 vendor default configuration

evaluate iteration i, $\vec{x}_{\vec{w}_i}^{\ *}$은 optimal configuration

Evaluation

◆ 실험 평가 – 모델 정확도

이 지표는 튜너를 평가하기 위해서만 계산할 수 있음

튜닝 시간에 사용할 수 없음 (모든 $\vec{x}^*$를 알 수 없음)

NPI를 시작으로, 튜너를 비교하는 데 사용하는 두 가지 지표를 정의

1. **누적보상(CR)** – 강화학습에서의 표준 지표, 얻어진 NPI점수의 합계로 정의; $CR(i) = \sum_{j=0}^{i} NPI(j)$

1 : optimal configuration, 0 : 유지 , 음의 값(-) : 나쁨

2. **반복적 최적값(IB)** : $IB(i) = \max_{j:j\leq i, \vec{w}_j=\vec{w}_i} NPI(j)$

가능한 각 workload에 대해 **별도의 카운터를 유지**하고, 각 반복에서 적절한 카운터에 대해 **반복된 최대 값**을 계산

튜닝 작업을 16회 반복한 다음 각 반복에 대해 NPI, CR, IB 값을 계산합니다.

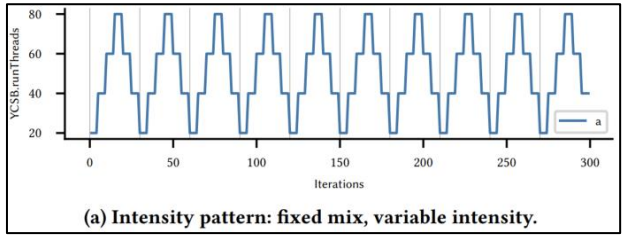
그런 다음 16회 반복에 대한 중앙값을 구하여 그림을 작성

Evaluation

◆ 실험 평가 – workload 패턴

MongoDB와 Cassandra의 평가에 사용되는 세 가지 Workload 패턴을 선택

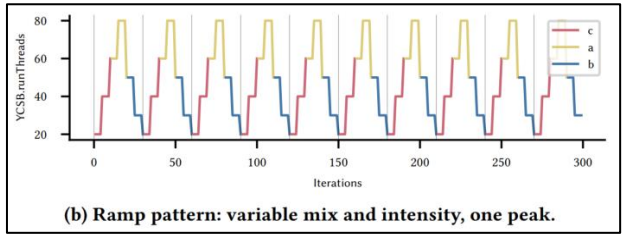
읽기/쓰기 혼합을 나타내는 색상과 number of YCSB 스레드를 나타내는 y축을 사용



(a) Intensity pattern: fixed mix, variable intensity.

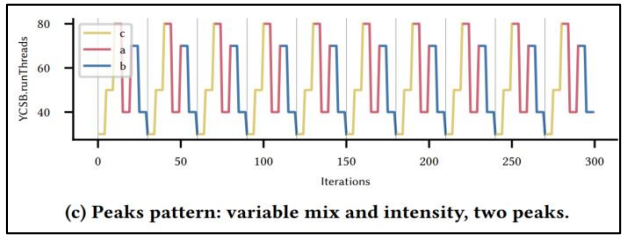
1. 가장 간단하며 하이퍼 파라미터 튜닝용으로만 사용

연결된 스레드 수를 변경하여 **simulating a gradual ramp in the road**



(b) Ramp pattern: variable mix and intensity, one peak.

2. **Ramp**라고 부르며, 연결된 스레드 수와 읽기/쓰기 혼합을 점진적으로 변경



(c) Peaks pattern: variable mix and intensity, two peaks.

3. **Peak**라고 부르며, 대부분의 workload가 집중된 근무 시간과 점심시간동안 감소하는 전형적인 일간 기반 작업량을 모방하려고 시도

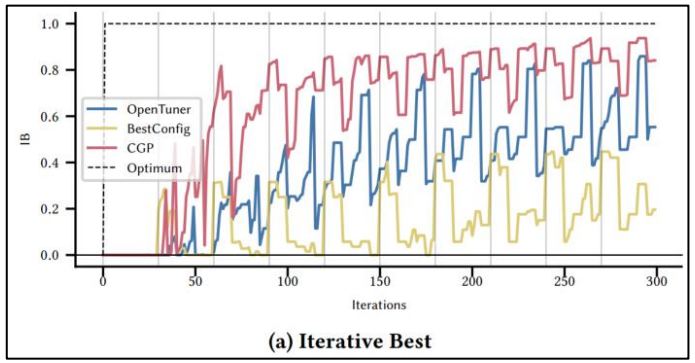
세 가지 패턴은 시간이 지남에 따라 동일하게 반복되지만, 어떤 튜너도 이를 활용할 수 없으며 현재의 workload만 중요

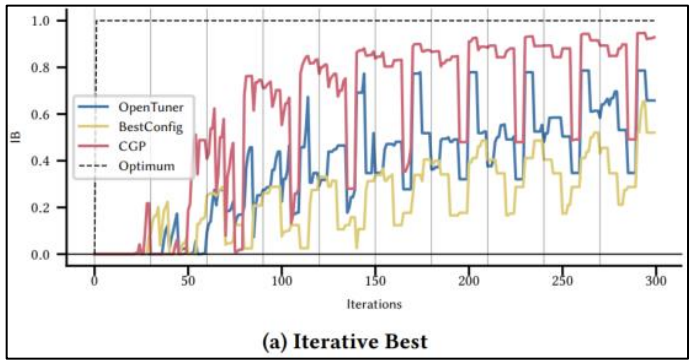
각 패턴 반복은 30회 반복이며, 각 측정 시간이 45분이므로 대략 하루 정도 소요

Evaluation

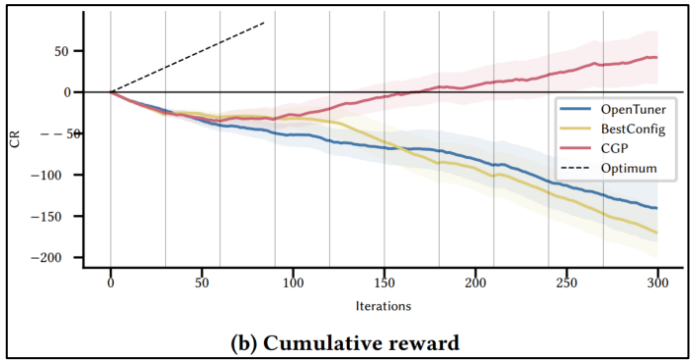
◆ 실험 결과

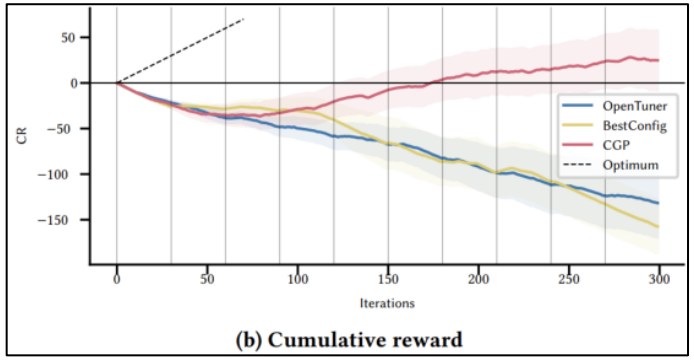
(offline)





(online)





Cassandra – Ramp pattern

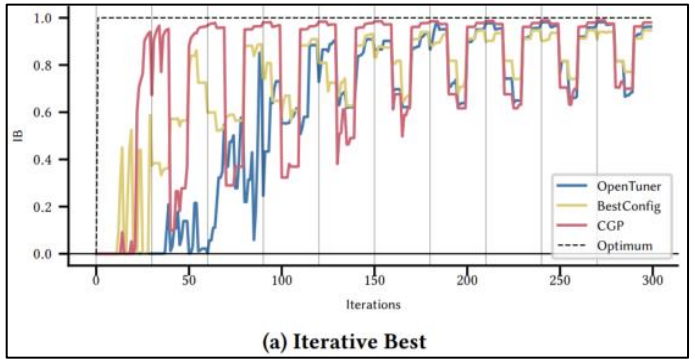
Cassandra – Peak pattern

Evaluation

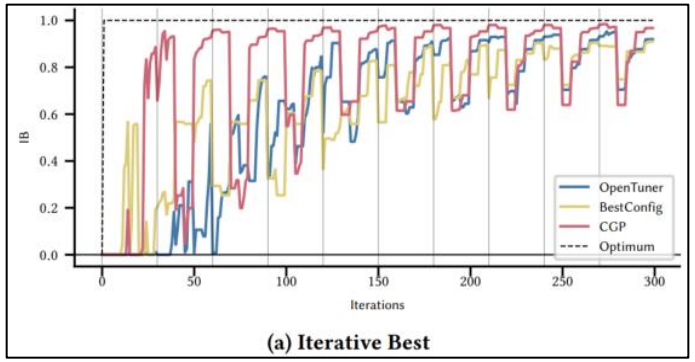
◆ 실험 결과

(offline)

(online)



MongoDB – Ramp pattern



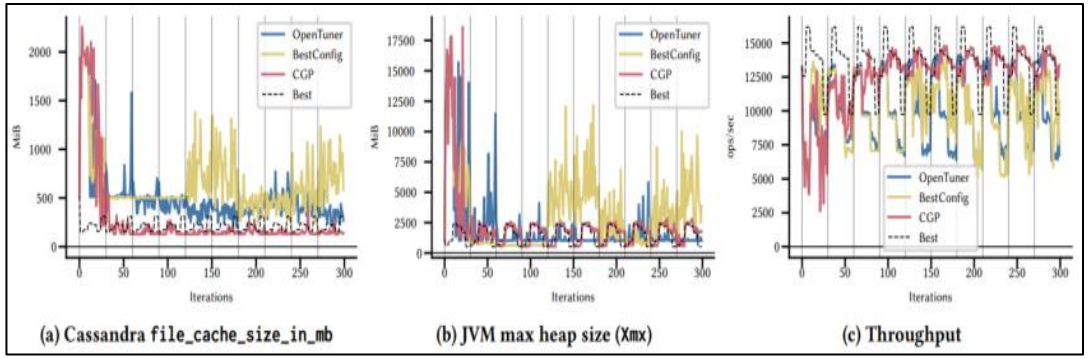
MongoDB – Peak pattern

σ [$\frac{MiB}{ms}$]	Scenario	Random	GP	Average IB score OpenTuner	BestConfig	CGP
10^1	Cassandra Ramp	0.00 ± 0.02	0.45 ± 0.20	0.72 ± 0.18	0.70 ± 0.20	**0.89 ± 0.03**
	Cassandra Peaks	0.01 ± 0.03	0.33 ± 0.15	0.68 ± 0.13	0.72 ± 0.16	**0.81 ± 0.09**
	MongoDB Ramp	0.33 ± 0.10	**0.95 ± 0.08**	0.78 ± 0.14	**0.92 ± 0.01**	**0.94 ± 0.05**
	MongoDB Peaks	0.31 ± 0.06	**0.93 ± 0.06**	0.77 ± 0.06	**0.92 ± 0.02**	**0.94 ± 0.02**
10^3	Cassandra Ramp	0.02 ± 0.03	0.14 ± 0.09	0.36 ± 0.12	0.19 ± 0.08	**0.63 ± 0.04**
	Cassandra Peaks	0.04 ± 0.04	0.15 ± 0.09	0.36 ± 0.12	0.24 ± 0.08	**0.57 ± 0.08**
	MongoDB Ramp	0.30 ± 0.07	0.30 ± 0.17	0.58 ± 0.10	**0.69 ± 0.10**	**0.73 ± 0.05**
	MongoDB Peaks	0.27 ± 0.07	0.44 ± 0.19	0.60 ± 0.07	0.60 ± 0.08	**0.75 ± 0.04**
10^4	Cassandra Ramp	0.42 ± 0.03	0.41 ± 0.06	0.60 ± 0.05	0.53 ± 0.03	**0.66 ± 0.03**
	Cassandra Peaks	0.44 ± 0.04	0.41 ± 0.04	0.59 ± 0.05	0.53 ± 0.03	**0.65 ± 0.04**
	MongoDB Ramp	0.51 ± 0.04	0.57 ± 0.07	0.63 ± 0.06	0.64 ± 0.05	**0.71 ± 0.05**
	MongoDB Peaks	0.59 ± 0.04	0.63 ± 0.07	0.65 ± 0.06	0.66 ± 0.05	**0.75 ± 0.03**
10^5	Cassandra Ramp	0.54 ± 0.02	0.52 ± 0.03	0.63 ± 0.05	0.58 ± 0.03	**0.70 ± 0.03**
	Cassandra Peaks	0.54 ± 0.03	0.53 ± 0.03	0.65 ± 0.05	0.58 ± 0.03	**0.68 ± 0.03**
	MongoDB Ramp	0.82 ± 0.02	0.82 ± 0.02	0.82 ± 0.03	0.75 ± 0.04	**0.86 ± 0.02**
	MongoDB Peaks	0.83 ± 0.01	0.83 ± 0.02	0.83 ± 0.02	0.77 ± 0.04	**0.88 ± 0.02**

σ [$\frac{MiB}{ms}$]	Scenario	Random	GP	Average NPI score OpenTuner	BestConfig	CGP
10^1	Cassandra Ramp	−0.99 ± 0.00	−0.90 ± 0.05	−0.01 ± 0.12	0.01 ± 0.17	**0.70 ± 0.07**
	Cassandra Peaks	−0.99 ± 0.01	−0.94 ± 0.03	0.01 ± 0.09	−0.38 ± 0.16	**0.47 ± 0.15**
	MongoDB Ramp	−0.92 ± 0.02	0.13 ± 0.37	0.18 ± 0.13	0.07 ± 0.21	**0.74 ± 0.11**
	MongoDB Peaks	−0.92 ± 0.01	−0.14 ± 0.26	0.21 ± 0.10	−0.04 ± 0.18	**0.76 ± 0.04**
10^3	Cassandra Ramp	−0.98 ± 0.01	−0.96 ± 0.02	−0.47 ± 0.13	−0.57 ± 0.10	**0.14 ± 0.10**
	Cassandra Peaks	−0.98 ± 0.01	−0.95 ± 0.02	−0.44 ± 0.13	−0.52 ± 0.10	**0.08 ± 0.11**
	MongoDB Ramp	−0.91 ± 0.02	−0.91 ± 0.06	−0.09 ± 0.13	−0.07 ± 0.15	**0.16 ± 0.06**
	MongoDB Peaks	−0.91 ± 0.02	−0.87 ± 0.13	−0.16 ± 0.12	−0.12 ± 0.10	**0.15 ± 0.07**
10^4	Cassandra Ramp	−0.69 ± 0.03	−0.73 ± 0.05	−0.25 ± 0.15	−0.28 ± 0.11	**0.32 ± 0.04**
	Cassandra Peaks	−0.68 ± 0.03	−0.73 ± 0.03	−0.20 ± 0.24	−0.27 ± 0.07	**0.34 ± 0.04**
	MongoDB Ramp	−0.43 ± 0.03	−0.37 ± 0.05	0.01 ± 0.08	0.04 ± 0.08	**0.35 ± 0.07**
	MongoDB Peaks	−0.42 ± 0.02	−0.33 ± 0.07	0.05 ± 0.09	0.10 ± 0.10	**0.34 ± 0.05**
10^5	Cassandra Ramp	−0.53 ± 0.04	−0.57 ± 0.07	0.23 ± 0.25	0.07 ± 0.13	**0.48 ± 0.06**
	Cassandra Peaks	−0.52 ± 0.04	−0.58 ± 0.05	0.24 ± 0.20	0.18 ± 0.11	**0.44 ± 0.03**
	MongoDB Ramp	0.46 ± 0.02	0.46 ± 0.05	0.58 ± 0.03	0.58 ± 0.09	**0.63 ± 0.02**
	MongoDB Peaks	0.49 ± 0.02	0.47 ± 0.04	0.61 ± 0.03	0.61 ± 0.04	**0.66 ± 0.02**

Evaluation

◆ **Suggested Configuration**



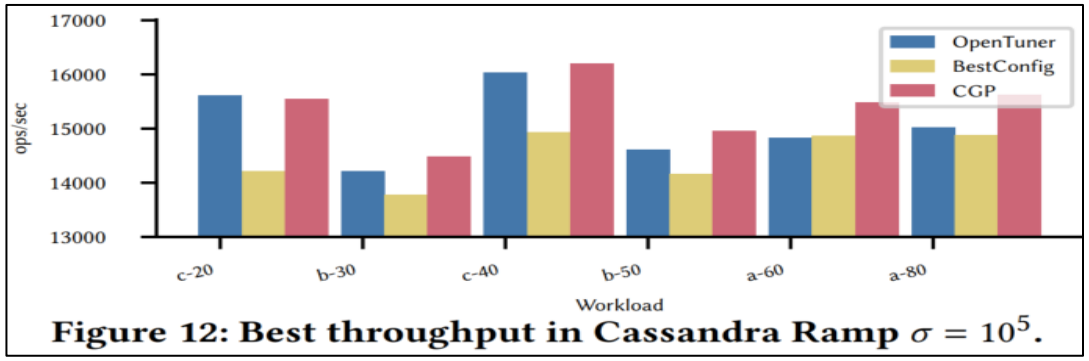
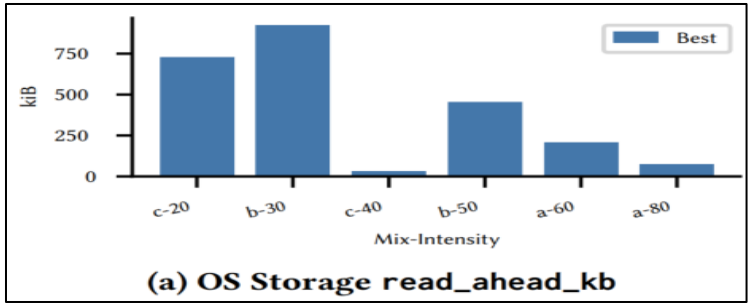
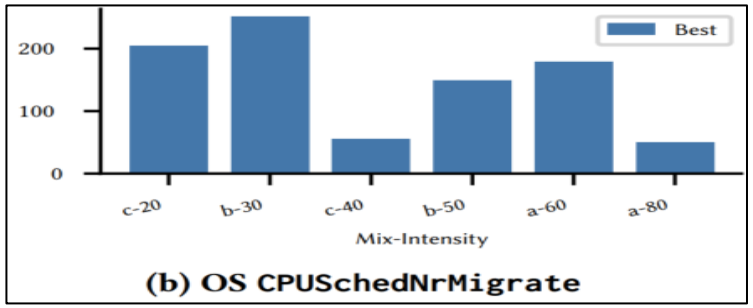
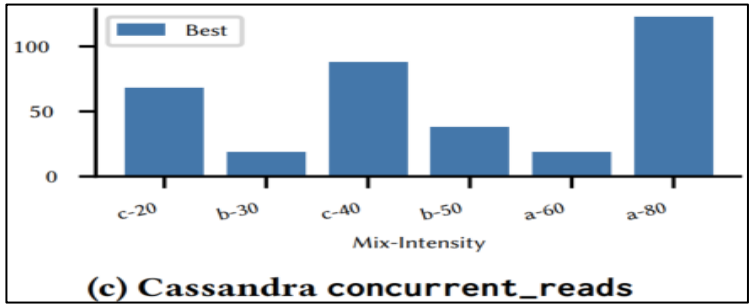


Figure 12: Best throughput in Cassandra Ramp $\sigma = 10^5$.

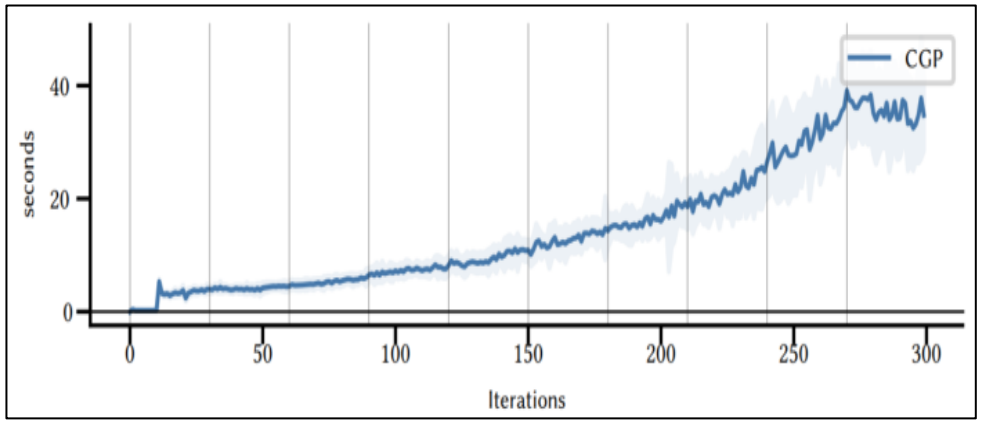






Evaluation

◆ **CGP complexity**



각 반복 시 CGP에 필요한 시간(초)
최대 1분 미만의 시간을 소요

Conclusion

◆ Conclusion

Knowledge base(KB)에 의존하지 않고 들어오는 workload에 적응하여 IT system의 configuration을 tuning하는 **Contextual Gaussian Process Bayesian Optimizer인 CGPTuner을 제안**

DBMS에서 CGPTuner를 평가하여 다양한 DBMS간의 portability를 입증

16가지 튜닝 시나리오를 평가하여 제안된 알고리즘이 구성 공간을 빠르게 이해하고 vendor default configuration을 뛰어넘는 것을 보여줌

이후 **더 많은 tuning iteration과 더 큰 검색 공간**에서 **작동하는 CGPTuner**를 개선할 계획