VDTuner: Automated Performance Tuning for Vector Data Management Systems

연세대학교 컴퓨터과학과 최승연

2024년 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

Table of Contents

01 BACKGROUND AND MOTIVATION

- Vector Data Management System (VDMS)
  - 벡터를 효율적이고 확장 가능하며 신뢰성 있게 관리하기 위한 시스템
  - Similarity Search (유사성 검색)
  - 새 벡터가 주어졌을 때 저장된 데이터에서 상위 k개의 유사한 벡터를 검색

- VDMS의 특징
- Multiple Components
  - VDMS 아키텍처는 특정 기능을 위한 여러 계층으로 구성됨
    - Access, Coordinator, Worker, Storage
  - 이러한 구성 요소는 사용자가 지정할 수 있는 VDMS의 많은 조정 가능한 system parameter를 노출하며, 고정된 구성은 모든 시나리오에 적용될 수 없습니다.

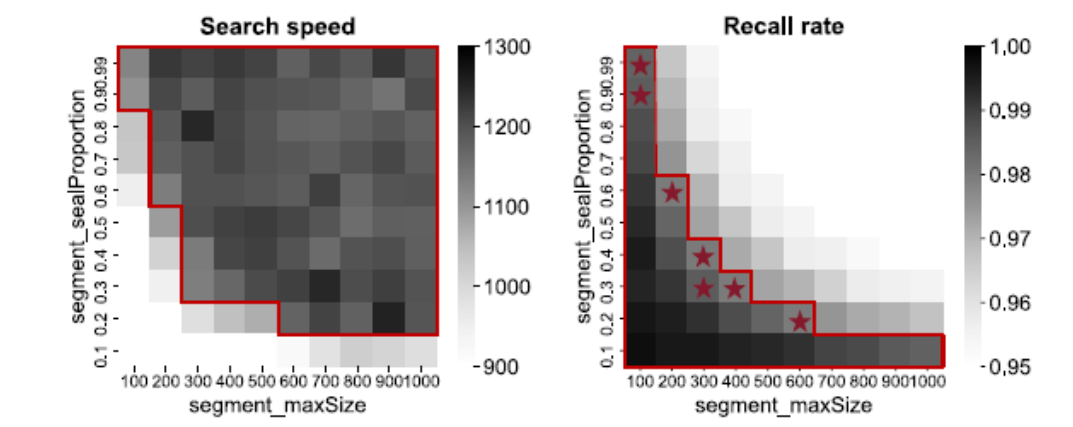
01 BACKGROUND AND MOTIVATION

- Multiple Index Types
  - Similarity search is noted for its high complexity
  - Approximate nearest neighbor search (ANNS) algorithm requires its own index type
    - -> VDMS usually need to maintain multiple index types
  - Complete indexed query process: 사용자가 하나의 인덱스 유형과 여러 인덱스 매개변수를 지정해야 함

- Multiple Performance Metrics
  - Search Speed: the request number that VDMS can handle per second
  - Recall rate: the ratio of correctly retrieved similar vectors to the total actual similar vectors
  - VDMS는 두 가지의 성능 지표를 동시에 고려해야 함

01 BACKGROUND AND MOTIVATION

- Challenges in Auto-Configuring VDMS
- VDMS 매개변수는 복잡하게 상호 의존적
  - 최적의 VDMS 구성을 찾기 위해서는 복잡하고 다차원의 검색 공간을 탐색해야 함
  - 검색 공간이 매우 커서 모든 가능한 구성을 다 시도해보는 것은 불가능
  - Index와 system parameter 간에도 상호 의존성 존재 -> 다른 시스템 configuration에서 최적의 index type이 다를 수 있음
  - Parameter 수와 범위가 자주 변경됨 -> 사전 도메인 지식 없이 이러한 parameter를 조정하는 방법 고려



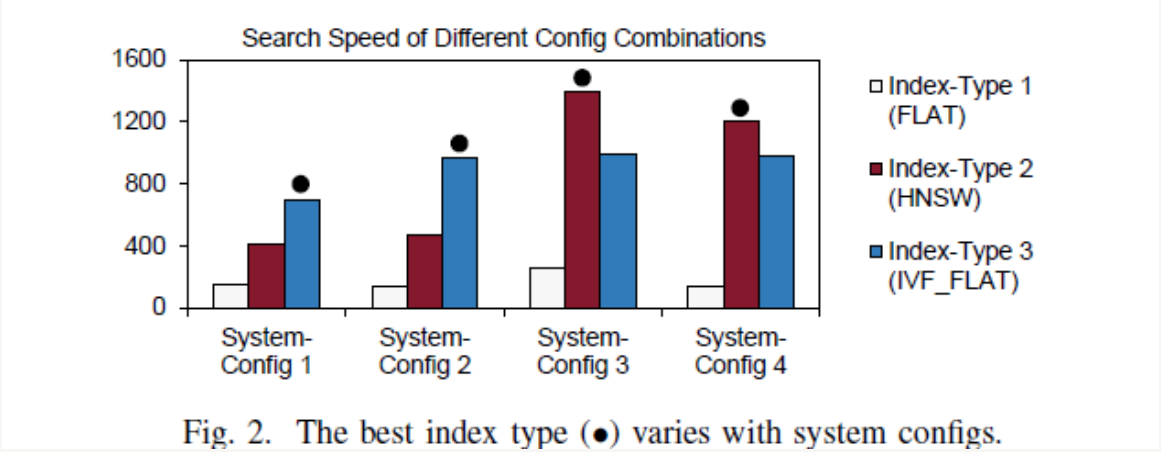
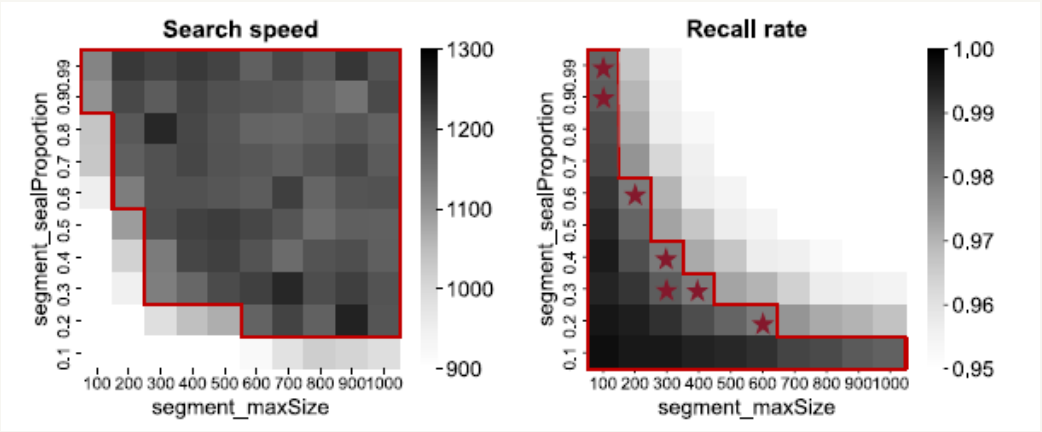


Fig. 2. The best index type (●) varies with system configs.

01 BACKGROUND AND MOTIVATION

- Challenges in Auto-Configuring VDMS
- VDMS focuses on two important metrics: Search speed & Recall rate
  - 둘은 본질적으로 상충되어서 두 가지의 균형을 맞추는 최적의 구성을 찾기 어려움
  - 하나의 목표만 최적화하면 다른 목표의 성능이 크게 저하될 수 있음
  - 상충되는 목표 간의 균형을 맞추어, 어느 한 목표도 크게 희생하지 않는 최적 구성을 찾아야 함.

  - 단순한 접근 방식: 인덱스 유형을 일반적으로 인정된 최상의 것으로 고정 -> 시나리오마다 좋은 인덱스 유형이 달라짐

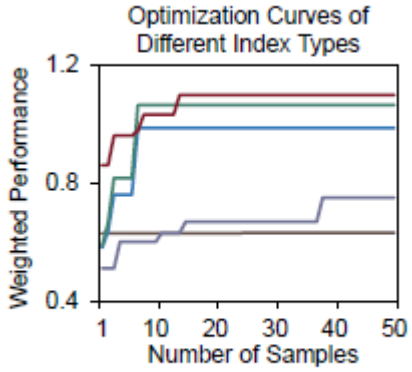


01 BACKGROUND AND MOTIVATION

- Challenges in Auto-Configuring VDMS
- Tunable parameters are different for different index types, while identifying the most suitable index type within limited tuning budgets is challenging.
  - Index type마다 조정 가능한 parameter의 조합 달라짐
  - 대부분의 기존 auto-tunning method는 고정된 세트의 조정 가능한 parameter를 가정하기 때문에 parameter와 범위가 변경되지 않음.
  - 각 인덱스 유형에 대해 개별적으로 매개변수를 조정하는 방법 생각
    - Time-consuming, 나머지를 무시하고 하나의 인덱스 유형만 선택됨
  - Index type만 최적화하는 것도 어려움
    - 샘플 수에 따른 각 인덱스 유형의 성능 변화
    - 단순한 샘플링 방법으로 최적의 index type 찾기 어려움
- 기존 auto-tunning solution들로는 VDMS의 튜닝 문제를 잘 해결하지 못 함.

TABLE I
INDEX TYPES AND CORRESPONDING PARAMETERS IN MILVUS.

Supported Index	Description	Building & Searching Parameters
FLAT	Exhaustive approach	N/A ; N/A
IVF_FLAT	Quantization-based	nlist ; nprobe
IVF_SQ8	Quantization-based	nlist ; nprobe
IVF_PQ	Quantization-based	nlist, m, nbits ; nprobe
HNSW	Graph-based	M, efConstruction ; ef
SCANN	Quantization-based	nlist ; nprobe, reorder_k
AUTOINDEX	Default configuration	N/A ; N/A



(c) Optimization Curves

02 VDTUNER: AN OVERVIEW

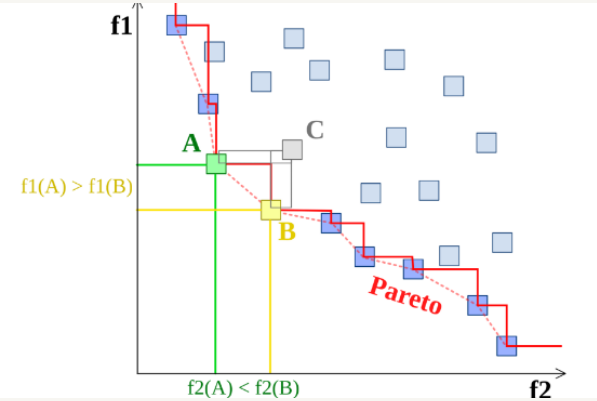
- **VDTuner**
  - Multi-objective Bayesian Optimization (MOBO)를 채택하여 VDMS를 자동으로 구성하는 framework

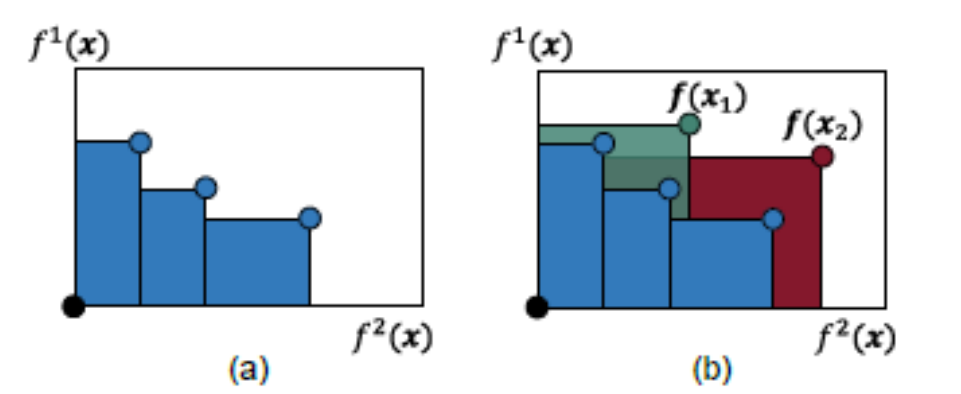
- Bayesian Optimization (BO)
  - 순차 모델 기반의 최적화 기법, 고비용 블랙박스 함수의 global optimum 찾기
  - 미지의 목적 함수를 근사하기 위해 확률적 대리 모델(probabilistic surrogate model)을 구성
    - Gussian Process를 사용
    - 새로운 함수 평가가 획득됨에 따라 반복적으로 업데이트 -> 새로운 정보 통합되고 모델 예측이 개선
  - 반복과정
    - Expected Improvement(EI) 또는 Probability of Improvement(PI)과 같은 Acquisition function를 사용해 평가할 다음 지점 결정
    - Acquisition functions는 exploration과 exploitation 간의 균형을 맞추어 제한된 함수 평가로 global optimum 을 효율적으로 검색

02 VDTUNER: AN OVERVIEW

- **Multi-Objective Bayesian Optimization (MOBO):** 여러 상충되는 목표를 다루기 위해 BO를 확장
  - 다양한 목표 간의 최적 trade-off를 나타내는 Pareto Front(최적 해 집합) 찾는 것

  - Acquisition functions 확장 -> EHVI(expected hypervolume improvement) 접근방식
    - 새로운 솔루션이 기존 솔루션 집합에 추가될 때 증가하는 hypervolume을 추정 -> 새로운 솔루션의 품질 평가
    - 모든 후보 솔루션 중에서 EHVI가 가장 높은 솔루션이 Acquisition function에 의해 선호
    - EHVI를 최적화 -> Pareto front의 전반적인 품질과 다양성을 개선하는 솔루션을 찾는 검색을 유도





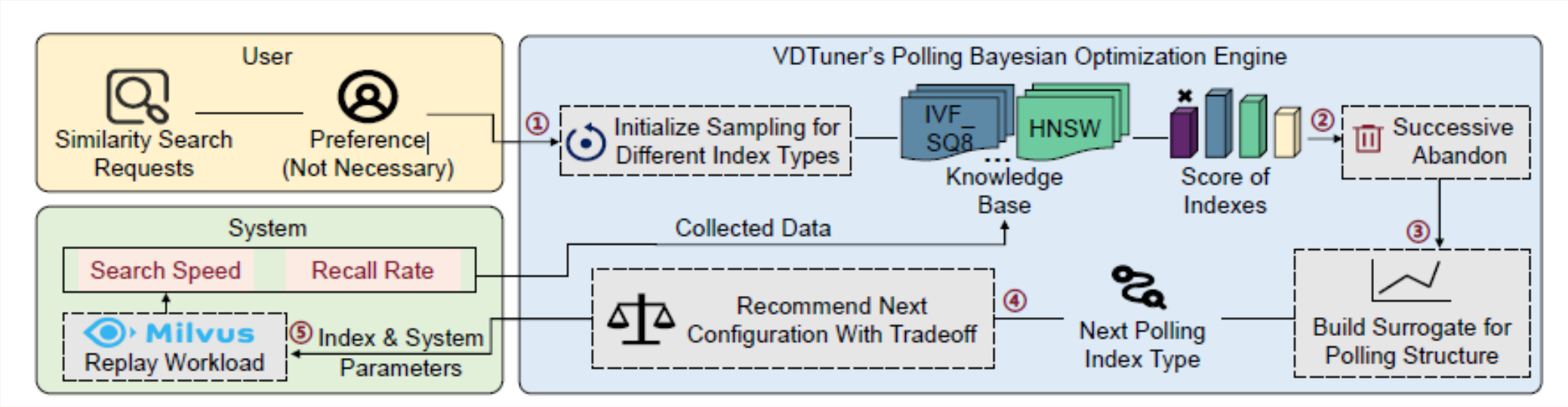
02 VDTUNER: AN OVERVIEW

- Why MOBO is Suitable for VDTuner
  - VDMS에 대한 사전 지식을 요구하지 않아 빠르게 변화하는 VDMS의 관리자의 부담을 덜어 줌
  - 복잡한 다차원 매개변수 공간을 효율적으로 탐색하여 과도한 Configuration 비용을 줄임
  - 다중 목표를 최적화하기 위해 설계되어서 Search speed와 Recall rate을 최적화하려는 요구 사항과 일치

- Challenges in Applying MOBO to VDMS Tuning
  - VDMS에서는 index type에 따라 튜닝 parameter가 고정되지 않아서 BO를 VDMS에 적용하기 위해서는 특수한 설계가 필요
  - Index type마다 성능 차이가 있어 모든 Index type에 튜닝 예산을 동일하게 할당하는 것은 비효율적, 더 효율적인 예산 할당 방식 필요
  - VDMS에서 모든 인덱스 유형이 공유하는 일부 글로벌 튜닝 매개변수 (예: 시스템 매개변수)
    - 한 인덱스 유형에서 학습된 지식이 다른 인덱스 유형에 영향 줄 수 있음
    - 다양한 인덱스 유형에서 학습된 지식을 최대한 활용하는 방법을 이용해야 함

03 VDTUNER: DESIGN AND IMPLEMENTATION

- VDTuner는 모든 인덱스 유형의 모든 조정 가능한 매개변수를 통합한 전체적인 BO 모델을 학습
- BO 모델을 학습하기 위해 VDMS의 구성을 반복적으로 샘플링



03 VDTUNER: DESIGN AND IMPLEMENTATION

- Surrogate Model
  - 사용 가능한 데이터를 기반으로 미지의 목적 함수를 근사
  - 평균 함수 m(x)와 공분산 함수 k(x, x′)로 특성화됨 => f(x) ~ GP(m(x), k(x, x′))
  - x는 모든 index type의 index parameter와 system parameter를 포함한 tunning parameter의 configuration
  - 입력 x가 주어지면, GP 모델은 이 구성 하에서 VDMS의 성능(Search speed, recall rate)을 추정
  - 평균 및 공분산 함수를 기반으로 업데이트함으로써 예측이 점점 더 정확해짐

- GP를 대략적으로 사용하면 현재 인덱스 유형에 의해 추천된 구성 주위에서 탐색
  - local optimal에 갇힐 위험성

- VDTuner는 서로 다른 인덱스 유형의 성능 변동성을 고려
  - GP의 입력 데이터를 정규화하는 polling surrogate model을 채택
  - 성능을 직접 사용하는 대신, 수정된 정규화 성능 개선(NPI)을 사용
    - 현재 최상의 구성과 비교하여 후보 구성의 상대적 개선 반영

$$(\hat{y}_i^{spd}, \hat{y}_i^{rec}) = (\frac{y_i^{spd}}{\overline{y}_t^{spd}}, \frac{y_i^{rec}}{\overline{y}_t^{rec}}), \quad (2)$$

$$(\overline{y}_t^{spd}, \overline{y}_t^{rec}) = \underset{(y^{spd}, y^{rec}) \in \mathcal{Y}_t}{\arg\max} \frac{1}{|y^{spd}/y_{max}^{spd} - y^{rec}/y_{max}^{rec}|}, \quad (3)$$

03 VDTUNER: DESIGN AND IMPLEMENTATION

- Acquisition Function
  - 지정된 index type에 대해 샘플링할 configuration을 추천하는 데 사용
  - 먼저 index type을 지정된 index type으로 설정, 이 index type에 속하지 않는 parameter는 기본값으로 설정
  - Surrogate Model의 예측에 따라 최대 utility 값을 달성하는 해당 index type의 parameter configuration을 추천
  - 두 가지 목표를 위해 EHVI를 사용
    - HV() 함수는 관측된 데이터의 hypervolume 측정
    - f(X′)는 대리 모델에 의해 예측된 성능 값을 측정
    - αEHVI는 각 후보 구성 X′을 추가하는 것에 대한 expected hypervolume improvement을 정량화

$$\begin{aligned}
&\alpha_{EHVI}(\mathcal{X}', \boldsymbol{r}, \mathcal{Y}) \\
&= \mathbb{E}\left[HV(\boldsymbol{r}, \mathcal{Y} \cup \{\boldsymbol{f}(\mathcal{X}')\}) - HV(\boldsymbol{r}, \mathcal{Y})\right] \\
&= \int_{-\infty}^{\infty} \left(HV(\boldsymbol{r}, \mathcal{Y} \cup \{\boldsymbol{f}(\mathcal{X}')\}) - HV(\boldsymbol{r}, \mathcal{Y})\right) d\boldsymbol{f}, \qquad (4)
\end{aligned}$$

03 VDTUNER: DESIGN AND IMPLEMENTATION

- Budget Allocation Among Index Types
- Round - robin 규칙
  - 인덱스 유형을 순환적으로 할당, 순차적으로 차례를 돌아가며 각각 순서를 지키도록 함
  - 특정 인덱스 유형을 우선하지 않지만, 좋은 인덱스 유형을 인식하지 못해 효율성이 떨어짐

- Round robin 개선 위해 Successive Abandon Strategy 수행
  - 설계된 함수에 따라 동적으로 점수가 매겨지고, 튜닝 과정에서 최악의 인덱스 유형은 연속적으로 포기
  - 유망한 인덱스 유형에 대한 탐색에 점차 집중
  - 특정 인덱스 유형의 데이터를 제외한 후 계산된 HV 값이 크게 감소하면, 해당 인덱스 유형이 좋은 구성을 찾는 데 크게 기여함을 의미
  - Score function

$$\Delta HV = HV(\boldsymbol{r}, \mathcal{Y}) - HV(\boldsymbol{r}, \mathcal{Y}/\mathcal{Y}_t), \quad (5)$$

$$Score(t) = \max_{t' \in \{1,..,T\}} (HV(\boldsymbol{r}, \mathcal{Y}/\mathcal{Y}_{t'})) - HV(\boldsymbol{r}, \mathcal{Y}/\mathcal{Y}_t). \quad (6)$$

  - 튜닝 과정 중 언제 포기를 실행할지 결정하는 것도 중요
    - VDTuner는 포기의 trigger 조건으로 window variance mertric 이용
    - Score function 값으로 인덱스 유형의 순위가 일정 기간 동안 지속적으로 최하위이면, 해당 인덱스 유형은 포기됨

03 VDTUNER: DESIGN AND IMPLEMENTATION

Algorithm 1 VDTuner's polling Bayesian optimization.

Input Index type set $\{1, .., T\}$, index and system configuration space $\mathcal{X} \in \mathbb{R}_d$, and workload for optimization.

Initialize Observed data $\mathcal{D}_t = \emptyset$ for each index type t and remaining index type set $\mathcal{T}_{remain} = \{1, .., T\}$.

1: **for** $t \in \{1, .., T\}$ **do**
2: Initialize sampling for t with its default configuration x_0.
3: Replay the workload under x_0.
4: Update $\mathcal{D}_t$ with x_0 and observed performance (y_0^{spd}, y_0^{rec}).
5: **end for**
6: **while** True **do**
7: **if** $len(\mathcal{T}_{remain}) > 1$ **then**
8: **for** $t \in \mathcal{T}_{remain}$ **do**
9: Calculate $Score(t)$ according to Equation 6.
10: **end for**
11: **if** Satisfy the windowed variance metric **then**
12: Remove $\arg\min_{t \in \mathcal{T}_{remain}} Score(t)$ from $\mathcal{T}_{remain}$
13: **end if**
14: **end if**
15: **for** $t \in \{1, .., T\}$ **do**
16: Normalize $\mathcal{D}_t$ by Equation 2.
17: **end for**
18: Build MOBO's surrogate with standardized data.
19: $t\_poll \Leftarrow$ next polling index type within $\mathcal{T}_{remain}$
20: Set search region $\mathcal{X}'$ for tunable parameters under t_poll.
21: Generate next configuration $x_{new} \in \mathcal{X}'$ by Equation 4.
22: Reply the workload and update $\mathcal{D}_{t\_poll}$ with feedback.
23: **end while**

Output Best found index type and configurations.

- Putting Them Together
  - 모든 인덱스 유형에 대한 초기 샘플링을 수행 (1-5행)
  - 튜닝 반복 (6-23행)
    - 남아있는 인덱스 유형이 하나 이상인 경우, VDTuner는 먼저 인덱스 유형을 Score function으로 평가, 최악의 인덱스 유형을 포기할지 결정 (7-14행)
    - 모든 index type의 parameter configuration를 사용하여 특화된 GP surrogate model을 구축 (15-18행)
    - 뽑힌 index type에 대해 acquisition function 최대화하는 좋은 구성을 생성 (19-21행)
    - 생성된 구성을 평가하고 Knowledge Base 업데이트 (22행)
  - 종료 조건은 고정되어 있지 않음, 샘플의 최대 수와 같이 유연하게 지정

03 VDTUNER: DESIGN AND IMPLEMENTATION

- Handling User Preference

- Constraint Model.
  - 사용자 시나리오에 따라 튜닝 목표의 선호도 달라짐
    - Recall rate은 정의된 임계값 이상만 유지하면서 Search speed를 최적화
    - Recall rate 제약 조건(e.g., rlim > 0.85) 내에서 탐색
  - 식 4를 constraint EI acquisition function으로 대체
  - 제약 조건 내에서 Search speed를 최대화하는 데 집중

$$\begin{aligned}&\alpha_{EHVI}(\mathcal{X}', \boldsymbol{r}, \mathcal{Y})\\&= \mathbb{E}\left[HV(\boldsymbol{r}, \mathcal{Y} \cup \{\boldsymbol{f}(\mathcal{X}')\}) - HV(\boldsymbol{r}, \mathcal{Y})\right]\\&= \int_{-\infty}^{\infty} \left(HV(\boldsymbol{r}, \mathcal{Y} \cup \{\boldsymbol{f}(\mathcal{X}')\}) - HV(\boldsymbol{r}, \mathcal{Y})\right) d\boldsymbol{f}, \quad (4)\end{aligned}$$

$$\begin{aligned}&\alpha_{CEI}(\mathcal{X}_{cand}, rlim) = \alpha_{EI}(\mathcal{X}_{cand}) \cdot Pr(f^{rec}(\mathcal{X}_{cand}) > rlim)\\&= \mathbb{E}(\max(f^{spd}(\mathcal{X}_{cand}) - best\_f, 0)) \cdot Pr(f^{rec}(\mathcal{X}_{cand}) > rlim). \quad (7)\end{aligned}$$

- Bootstrapping with Previous Data.
  - 사용자의 recall 선호도 값이 변동하는 경우
  - 새로운 recall 제약 조건마다 처음부터 학습하는 것은 비효율적
  - 이전 샘플링 데이터에서 공유할 수 있는 유용한 정보가 있을 수 있음
  - 이전 샘플링 데이터로 surrogate model을 초기화하여 자동 튜닝을 bootstrapping

04 EVALUATION

- Experiment Setting
  - Platform: Milvus(버전 2.3.1)에서 평가
  - Workloads: vector-dbbenchmark 사용하여 작업 부하 생성
  - 표 III에 나와 있는 세 가지 대표적인 데이터셋을 테스트
  - Baselines
    - Default, Random Sampling, OpenTuner, OtterTune, qEHVI

TABLE III
EVALUATED DATASETS.

Dataset	Num. of Vectors	Dimension	Distance
GloVe	1,183,514	100	Angular
Keyword-match	1,000,000	100	Angular
Geo-radius	100,000	2048	Angular

- Benefit of Auto-Configuration VDMS
  - Default Configuration과 비교한 VDTuner의 성능 향상 결과
  - VDTuner는 성능을 최대 14.12%의 Search speed와 186.38%의 Recall rate까지 크게 향상
  - 데이터셋에 따라 개선 정도가 다름 (Geo-radius > Keyword-match > GloVe)
    - 유사성 검색의 난이도가 데이터 분포와 벡터 차원 등에 따라 다르기 때문

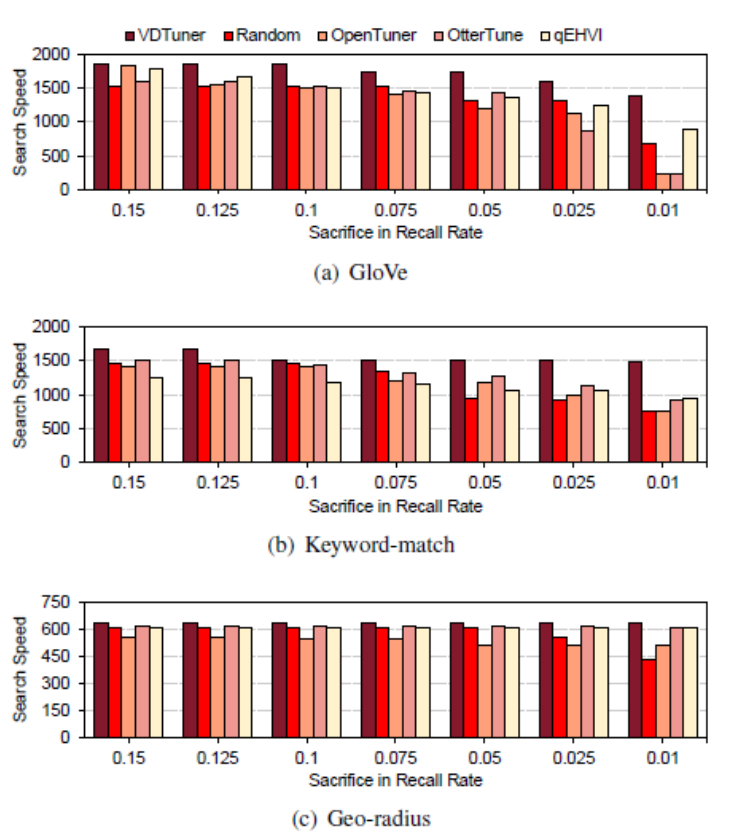
TABLE IV
PERFORMANCE IMPROVEMENT BY AUTO-CONFIGURATION.

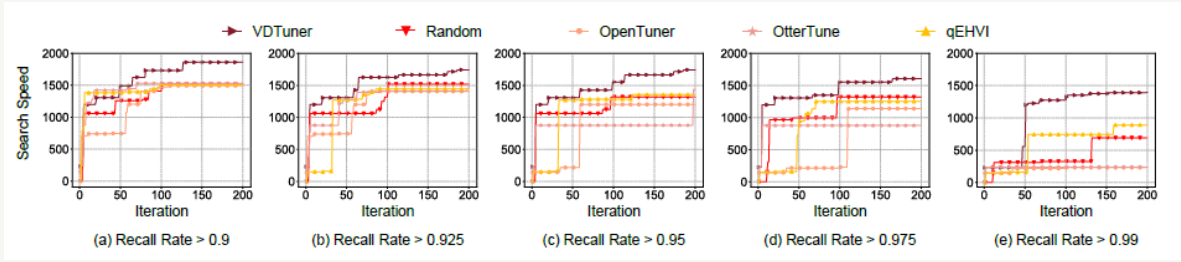
Metric	Datasets		
	GloVe	Keyword-match	Geo-radius
Speed Improvement	10.46%	11.17%	14.12%
Recall Improvement	17.16%	62.61%	186.38%

04 EVALUATION

- Tuning Efficiency
  - VDTuner는 Search speed와 recall rate 사이에서 균형을 이룸
  - 표준 편차가 낮을수록 목표 간의 균형이 더 잘 맞춰짐
  - 균형 능력의 순서: VDTuner, qEHVI, OtterTune, OpenTuner, Random
  - 두 목표 모두에서 baseline을 능가
  - Recall rate이 엄격하게 제한된 어려운 영역(0.01)에서도 가장 잘 수행됨

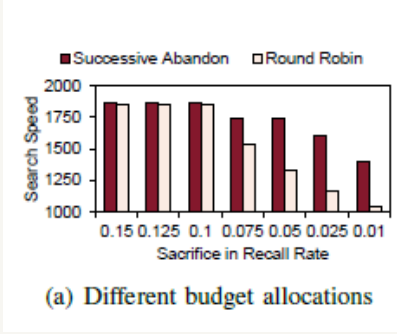
  - VDTuner는 경쟁 기준점과 비교하여 더 나은 Configuration을 훨씬 빠르게 식별
  - 더 적은 iteration으로 높은 Search Speed를 달성→ 우수한 Configuration을 가장 적은 샘플 수와 튜닝 시간으로 달성





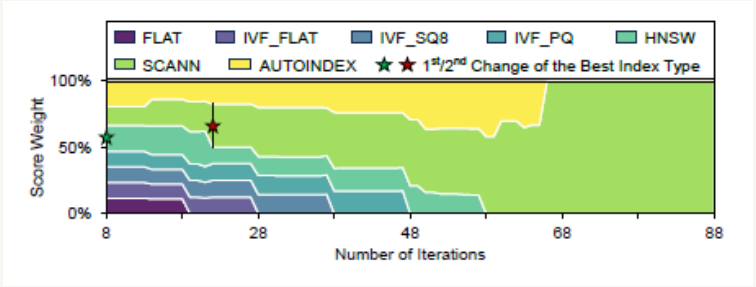
04 EVALUATION

- Why VDTuner Works Effectively
- Effectiveness of Budget Allocation
  - Successive abandon strategy 검색 속도를 향상시킴 (최대 34%)
  - VDTuner가 동적 score function을 통해 가장 적합한 index type을 식별할 수 있기 때문



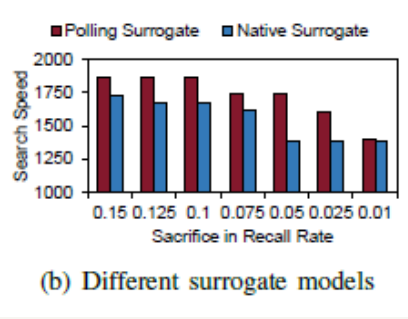
(a) Different budget allocations

  - 샘플 수가 증가함에 따라 각 인덱스 유형의 가중치를 나타내는 동적 점수 과정을 시각화

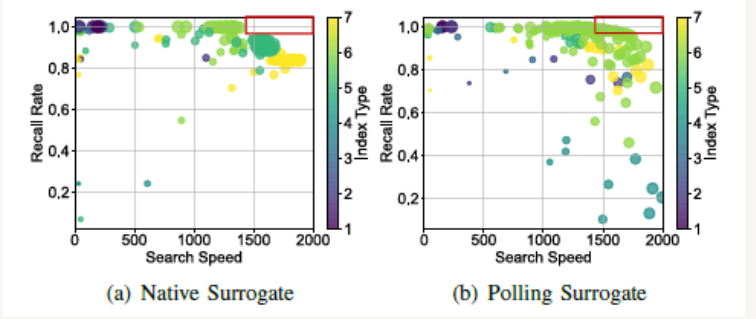


- Effectiveness of Surrogate Model
  - Native Gaussian Process surrogate model과 pooling surrogate model을 비교
  - Pooling surrogate model의 성능 향상 (최대 26%의 향상)

- Native Gaussian Process surrogate model
  - 같은 인덱스 유형에 대해 클러스터 현상, 더 유사한 성능 샘플링
- Pooling surrogate model
  - 다양한 recall rate 값의 넓은 공간을 탐색



(b) Different surrogate models



04 EVALUATION

- Holistic BO Model VS. Optimizing Each Index Type Individually.
  - 두 접근 방식에 의해 생성된 index type이 동일, 생성된 paramete도 매우 유사하다
  - 모두 SCANN을 최상의 index type으로 선택, 매개변수의 80% 이상에 대해 생성된 매개변수의 차이가 5% 미만

- 선택된 index type이 데이터셋에 따라 다름
- 데이터의 특성이 특정 index type의 효율성에 영향을 미침
- 다른 데이터 특성이 최적의 성능을 달성하기 위해 다른 parameter 설정을 요구
- -> 다양한 workload에 대한 VDMS 성능 튜닝이 필요

TABLE V
CHANGES OF INDEX AND PARAMETERS ACROSS DIFFERENT DATASETS.

	Datasets		
	GloVe	ArXiv-titles	Keyword-match
Index Type and Parameters of the Best Configuration	Index: SCANN nlist: 301 nprobe: 36 reorder_k: 283	Index: HNSW M: 64 efConstruction: 194 ef: 100	Index: SCANN nlist: 680 nprobe: 238 reorder_k: 465

- VDTuner가 생성한 매개변수의 변화를 반복 수와 함께 기록
- 튜닝 과정의 초기 단계에서는 모든 parameter의 변동이 비교적 크지만, 반복 횟수가 증가하면 모든 parameter는 기본적으로 작은 범위 내에서 변동하는 것으로 수렴

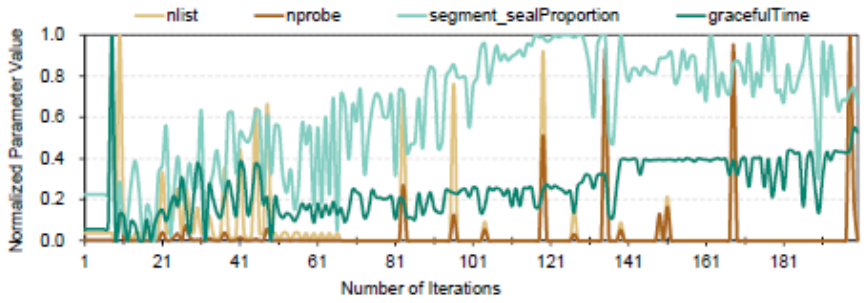
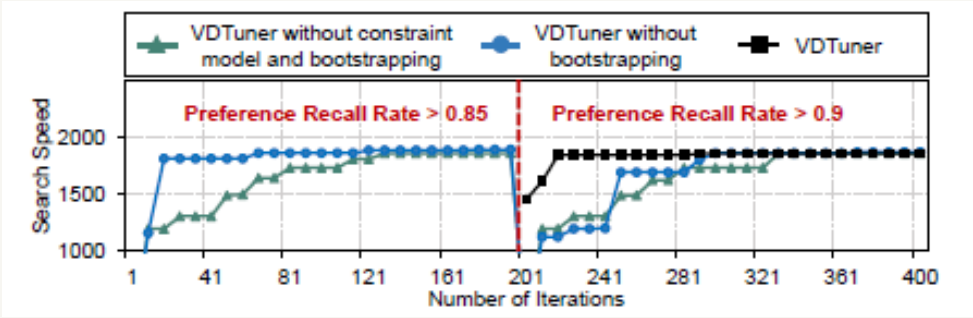


Fig. 11. Changes of parameters with the number of iterations.

04 EVALUATION

- Scalability
- Larger Datasets
  - VDTuner의 효과를 훨씬 더 큰 데이터셋(deep-image)에서 검증
  - 선택한 시나리오에서 최고 성능의 기준점(qEHVI)과 VDTuner의 성능을 비교
  - VDTuner가 8.1배 더 빠른 튜닝 속도를 보이면서 여전히 성능 면에서 상당한 이점을 유지

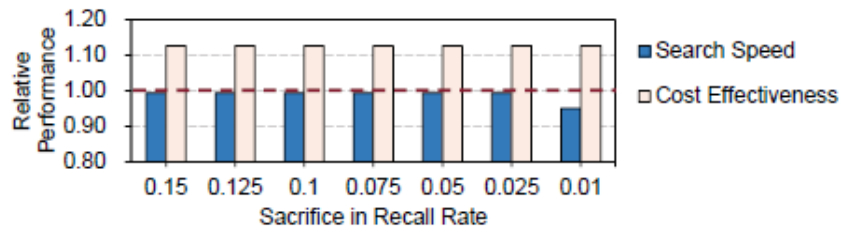
- Handling User Preference
- 사용자가 특정 recall에 대한 선호가 있을 때 VDTuner의 효과를 검증
- 제약 모델이 튜닝 효율성을 크게 향상시킴 -> 제약 모델이 있는 VDTuner가 recall rate이 임계값보다 높은 경우 검색 속도 최적화에 더 집중할 수 있음
- bootstrapping이 제약 모델이 있는 VDTuner의 자동 구성 효율성을 더욱 향상
  - 재현율 > 0.85 최적화한 이전 데이터를 사용하여 surrogate model을 워밍업
  - 고품질의 초기 구성과 대략적인 탐색 공간 분포를 제공



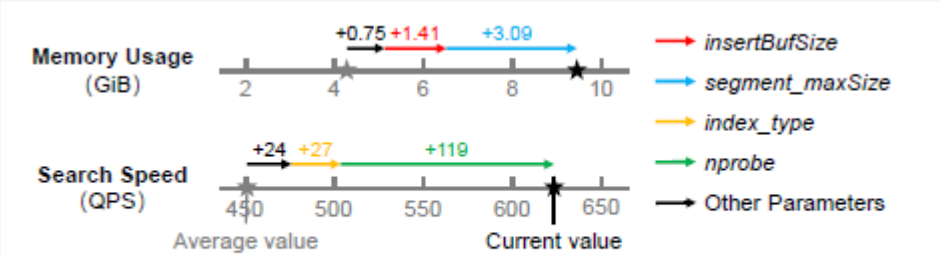
04 EVALUATION

- Scalability
- Cost-Effectiveness Optimization.
  - Search speed를 덜 필요로 하고, 메모리 사용량을 고려한 비용 효율성에 더 관심이 있는 경우
  - Search Speed(QPS)의 목표를 비용 효율성(QP$)으로 대체
  - 다른 자원 및 가격 함수를 최적화하는 것 → 비용 효율성의 정의를 수정

$$\text{Cost-Eff.} = \frac{\text{Search Speed (query/sec)}}{\text{Price (\$/sec)}} = \frac{\text{Search Speed}}{\eta \cdot \text{Memory Usage}}. \quad (8)$$



- 메모리 사용량과 검색 속도에 가장 중요한 parameter 조사
- 세그먼트 최대 크기(+3.09 GiB)와 인덱스 유형(+119 QPS)
- 검색 속도 최적화 버전은 메모리 사용량에 관계없이 더 높은 검색 속도를 위해 세그먼트 최대 크기를 추천하지만, 비용 효율성 최적화 버전은 검색 속도와 메모리 사용량 간의 균형을 맞추어 상대적으로 작은 세그먼트 최대 크기를 추천함



(b) Parameter's contribution to memory usage and search speed.

05 CONCLUSION

- VDMS 인덱스 및 시스템 구성을 최적화하는 학습 기반 성능 튜닝 프레임워크인 VDTuner를 제안
- VDTuner는 Search speed와 Recall rate간의 균형을 적극적으로 맞추며, pooling 구조, 특화된 surrogate model, successive abandon strateg의 예산 할당 전략을 통해 더 나은 성능을 제공
- 광범위한 평가를 통해 VDTuner가 효과적이며, 튜닝 효율성 측면에서 기준점을 큰 차이로 능가하고, 변동하는 사용자 선호도와 비용 인식 목표에 대해 확장 가능함을 입증
- 향후 VDTuner를 온라인 버전으로 확장하여 다양한 작업 부하를 적극적으로 포착하고, VDMS의 더 많은 수준(데이터 파티션)을 최적화하여 성능과 자원 활용을 더욱 향상시키고자 함.

감사합니다