Too Many Knobs to Tune?
Towards Faster Database Tuning by Pre-selecting Important Knobs

Konstantinos Kanellis, Ramnatthan Alagappan, Shivaram Venkataraman
Database tuning is important!

Realizing high performance requires finding optimal values for configuration knobs.

Properly tuned database systems can achieve 2-3x higher throughput (or lower 99-tile latency) compared to default configuration (PostgreSQL) [1]

[Cassandra Default Configuration]

...commitlog_sync_period_in_ms: 10000
commitlog_segment_size_in_mb: 32
compaction_throughput_mb_per_sec: 16
concurrent_reads: 32
concurrent_writes: 32
memtable_heap_space_in_mb: 2048
memtable_cleanup_threshold: 0.33
native_transport_max_threads: 128

[Tuning Process]

...commitlog_sync_period_in_ms: 50000
commitlog_segment_size_in_mb: 128
compaction_throughput_mb_per_sec: 16
concurrent_reads: 128
concurrent_writes: 64
memtable_heap_space_in_mb: 1024
memtable_cleanup_threshold: 0.85
native_transport_max_threads: 256

[Cassandra Tuned Configuration]

... but it’s hard ...

- 100s knobs in a typical system
- Most knobs take *continuous* values
- Unknown *interactions* among knobs
- Evaluating a single configuration is expensive

Earlier tuning efforts relied on experience from human experts

Recently proposed tuning frameworks can **automate** the procedure

Can achieve same (or even better) performance compared to manual tuning [2]

---

Automated database tuning

Most existing auto-tuning database frameworks consist of (a) initial **offline profiling** phase and (b) an online **tuning** phase.
Motivation

Offline profiling is **vital** for the quality of proposed configurations. Yet, this phase may account for **>95%** of the entire tuning time.

**All Tunable Knobs?**
**Mix of Workloads?**

How many knobs do we need to achieve “good” performance? Can we **exploit** this to accelerate the offline phase?
Experimental study

How many knobs do we need to achieve “good” performance?

Do similar results hold for different workloads?

Do similar results hold for a different database system?

Cassandra YCSB-A
5 out 155!

Cassandra YCSB-B
Same 5 knobs!

PostgreSQL YCSB-A, YCSB-B
Yes!
Outline

Background & Motivation

Methodology

Results

Towards Faster Database Tuning
Methodology

(I) Ground-truth dataset collection

(II) Identify most important knobs

Evaluate top-k knobs performance

Generate and evaluate configuration samples (many knobs)

Identify relationship of each knob with system performance

Generate samples and find one with best perf. (top-k knobs)

Compare with ground-truth
(I) Generate and collect configuration samples

Intractable configuration space – limited number of samples

Latin Hypercube Sampling (LHS)
- Uniformly and thoroughly cover configuration space
- Employed by multiple existing systems

Number of Samples
Knobs / Range of values

\{
commitlog_sync_period: 10 ms
concurrent_writes: 8
memtable_cleanup_threshold: 0.2
\}

\{
commitlog_sync_period: 5 ms
concurrent_writes: 64
memtable_cleanup_threshold: 0.8
\}
(II) Identify Important Knobs

Knobs values (features)

{ commitlog_sync_period: 10 secs
  concurrent_writes: 8
  memtable_cleanup_threshold: 0.2
}

{ commitlog_sync_period: 5 secs
  concurrent_writes: 64
  memtable_cleanup_threshold: 0.8
}

{ commitlog_sync_period: 2 secs
  concurrent_writes: 24
  memtable_cleanup_threshold: 0.5
}

...
Experimental Setup

**Machine hardware**
- Intel Xeon Silver 4114 CPU, 64 GB RAM, 480GB SSD, Ubuntu 18.04
- Employ 30 identical machines to parallelize the evaluation process (CloudLab)

**Ground-truth sample collection**
- Apache Cassandra v3.11, PostgreSQL v9.6
- YCSB-A (50% read/50% write), YCSB-B (95% read/5% write)
- **25,000 samples** with LHS – tweaking ~30 knobs for both systems
- Each sample takes ~9 minutes to evaluate
Outline

Background & Motivation

Methodology

Results

Towards Faster Database Tuning
According to the ML model, these 5 knobs have the most impact on system performance.

Most important knobs

- **concurrent_reads**: number of concurrent read operations
- **native_transport_max_threads**: number of threads used to handle requests
- **memory table–related** knobs: size of memtable, when to flush to disk
...but how much performance can we achieve?

Apache Cassandra – YCSB-A

Tuning just a few important knobs can still yield high performance!

<table>
<thead>
<tr>
<th>Best Configuration Performance</th>
<th>Throughput (ops/sec)</th>
<th>Read Latency (micro-seconds)</th>
<th>Write Latency (micro-seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuning 30 knobs</td>
<td>74780.33</td>
<td>744.34</td>
<td>302.82</td>
</tr>
<tr>
<td>Tuning 5 knobs</td>
<td>74304.42</td>
<td>750.56</td>
<td>308.08</td>
</tr>
<tr>
<td>% of tuning 30 knobs</td>
<td>99.36%</td>
<td>100.84%</td>
<td>101.41%</td>
</tr>
</tbody>
</table>
What about a different workload?

Apache Cassandra – YCSB-B

#1: A handful of knobs affect the performance for YCSB-B

#2: **Overlap** of important knobs across the two workloads
What about a different database system?

PostgreSQL – YCSB-A, YCSB-B

In general, we observe similar results for PostgreSQL Knob importance ranking more diverse between the workloads … still top-8 knobs are almost identical.
Outline

Background & Motivation

Methodology

Results

Towards Faster Database Tuning
Pre-selecting Important Knobs

Utilize the ML model to identify important knobs before running the tuner. Reduces configuration search space size / training dataset of tuners.

Current design

Our proposed two-level design

All Tunable Knobs
Mix of Workloads

Auto-Tuning Framework

Configurations

Pre-select Important Knobs

Important Knobs

Auto-Tuning Framework

Configurations

All Tunable Knobs
Few Workloads

Mix of Workloads

HotStorage’20
Pre-selecting Important Knobs

Utilize the ML model to identify important knobs before running the tuner.

Early results with an existing tuner, BestConfig. When tuning top-5 knobs the best performance is reached with 5x fewer iterations compared to tuning 30 knobs (Apache Cassandra, YCSB-A).

HotStorage'20
Discussion

Can we make the pre-selection step cheaper? (25,000 samples)
- With our ML-based method \(~400\) samples are needed (early results)
- Can we use some other (cheaper) method? (evaluate few workloads?)

How does the hardware affect the important knobs?
- Can we avoid (or minimize) tuner adaptation time to new hardware?

Can we account for system reliability when tuning?
- Existing tuners may sacrifice reliability for performance
- \(\text{fsync} / \) recovery-related flags / checkpointing settings
Summary

Tuning with few important knobs can yield high performance
- Trend seems to hold across different workloads and systems
- Significant overlap of top knobs across different workloads

Proposed an initial design to accelerate database auto-tuners
- Pre-selecting important knobs reduces configuration search space
- Exploit top knobs similarity across workloads to make it faster?
Tuning with few important knobs can yield high performance
- Trend seems to hold across different workloads and systems
- Significant overlap of top knobs across different workloads

Proposed an initial design to accelerate database auto-tuners
- Pre-selecting important knobs reduces configuration search space
- Exploit top knobs similarity across workloads to make it faster?