Auto-Tuning RocksDB by machine learning

Yuanli Wang

William Batu

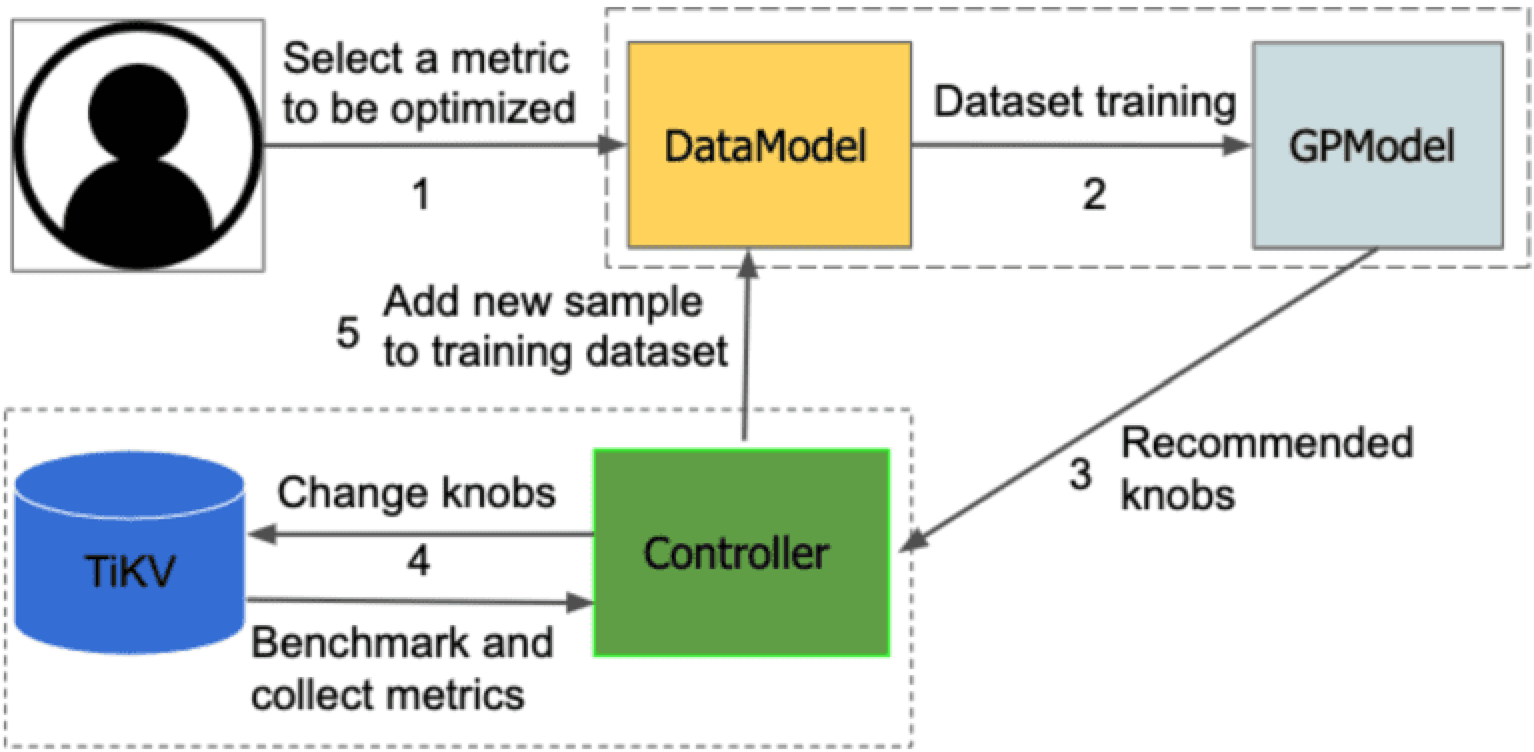
Background and motivation

- TiKV: open-source transactional key-value database
  - Use RocksDB as backend storage engine
  - Raft consensus algorithm
  - https://github.com/tikv/

- RocksDB: a persistent key-value store engine
  - RocksDB has many configurations. It is hard to choose proper values in production.
  - The goal is to auto-tune RocksDB in real time for different workloads

- Dana Van Aken et al, Automatic Database Management System Tuning Through Large-scale Machine Learning, SIGMOD 2017 .

Pipeline


Select a metric
to be optimized
1
DataModel
Dataset training
2
GPModel
5
Add new sample
to training dataset
3
Recommended
knobs
TiKV
Change knobs
4
Controller
Benchmark and
collect metrics

ML model

- Gaussian Process Regression: a non-parametric model based on the Gaussian Distribution
- Then apply the estimation to Bayesian optimization
  - Use GPR to estimates the distribution of the sample—the mean of X, m(X), and its standard deviation, s(X).
  - Use the acquisition function to guide the next sample, and give the recommended value.
- Explore && exploit
  - Exploration: The function explores new points in unknown areas where there is currently insufficient data.
  - Exploitation: The function uses the data for model training and estimation to find the optimal prediction in the known areas with sufficient data.
  - Use Upper Confidence Bound function to do tradeoff
    - U(X) = m(X) + k*s(X)

Workload && knobs

- Workloads: generated by ycsb
  - write-heavy, range-scan (both long and short), point-lookup [2]

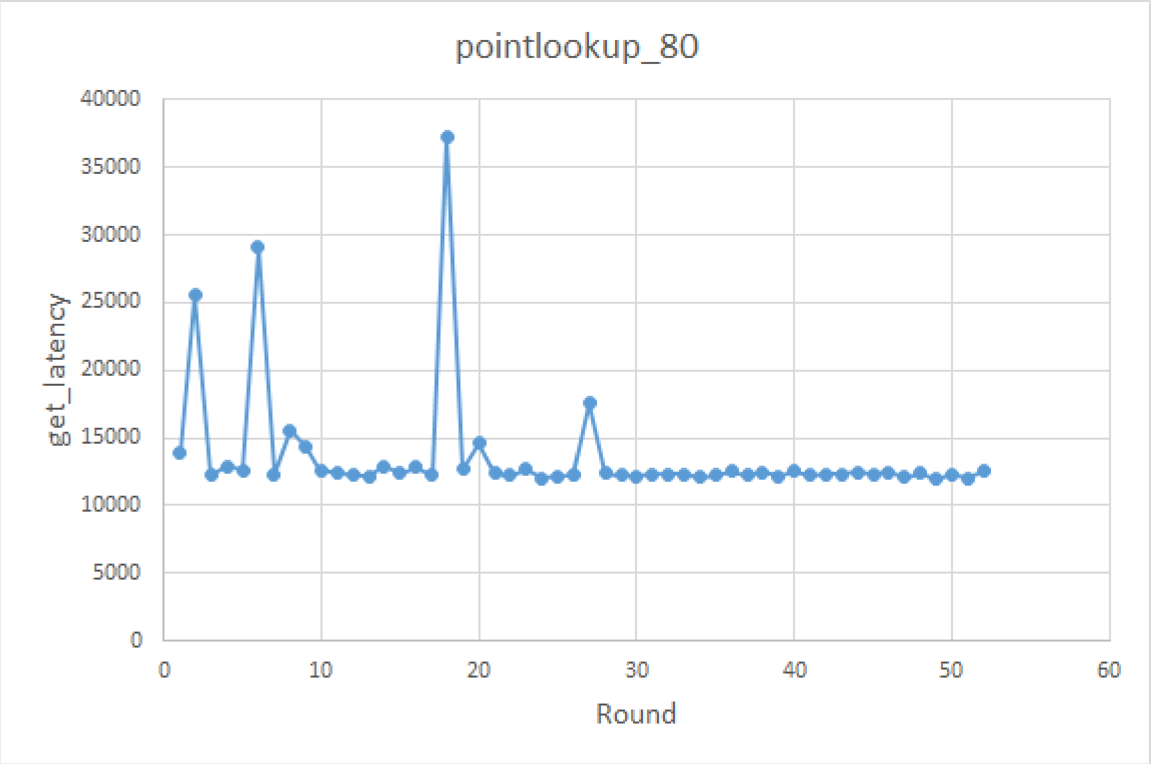
- Knobs:

Parameter	Workload/expected behaviors	Valid range/value set
disable-auto-compactions	write-heavy: turning on is better point-lookup, range-scan: turning off is better	{1, 0}
block-size	point-lookup: the smaller the better range-scan: the larger the better	{4k,8k,16k,32k,64k}
bloom-filter-bits-per-key	point-lookup, range-scan: larger the better	[5,10,15,20]
optimize-filters-for-hits	point-lookup, range-scan: turning off is better	{1,0}

- Metrics: Throughput / Latency

Evaluation

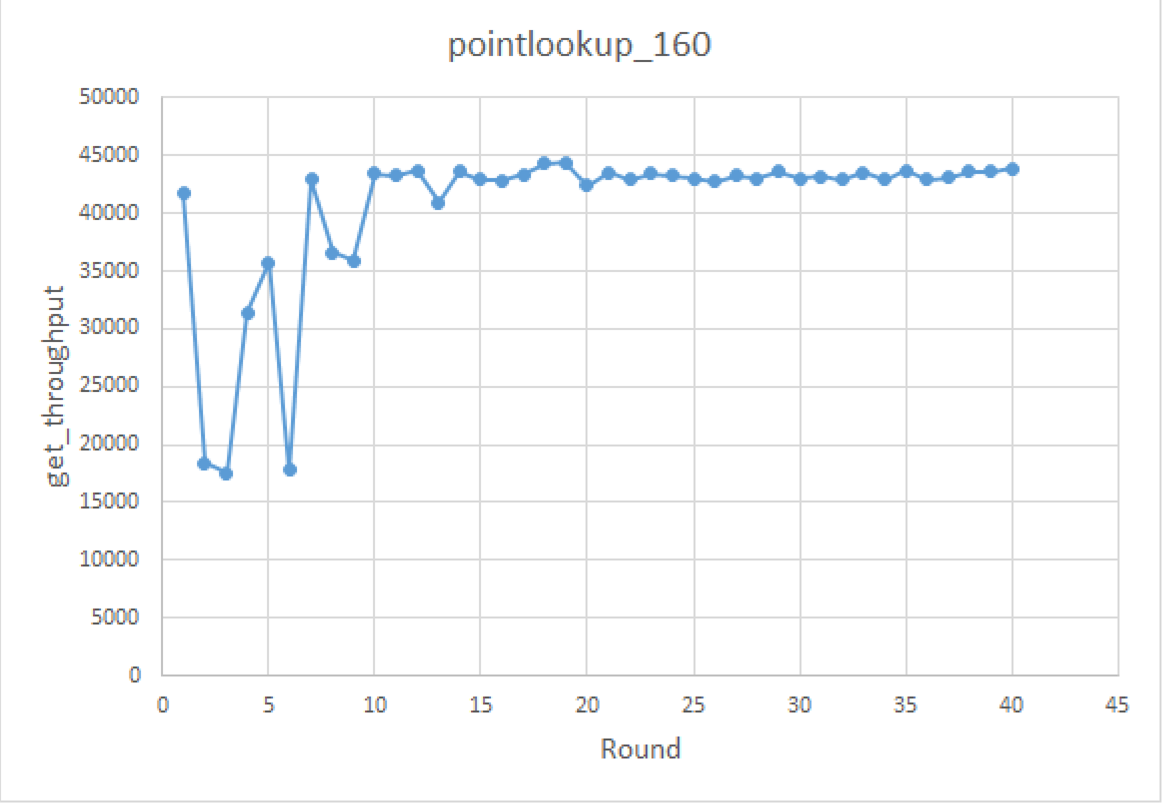
- `workload=pntlookup80`
- `knobs={'bloom-filter-bits-per-key', 'optimize-filters-for-hits', 'block-size', 'disable-auto-compactions'}`
- `metric=get_latency`



More details:
https://www.cncf.io/blog/2019/12/10/autotikv-tikv-tuning-made-easy-by-ai-and-machine-learning/

Evaluation

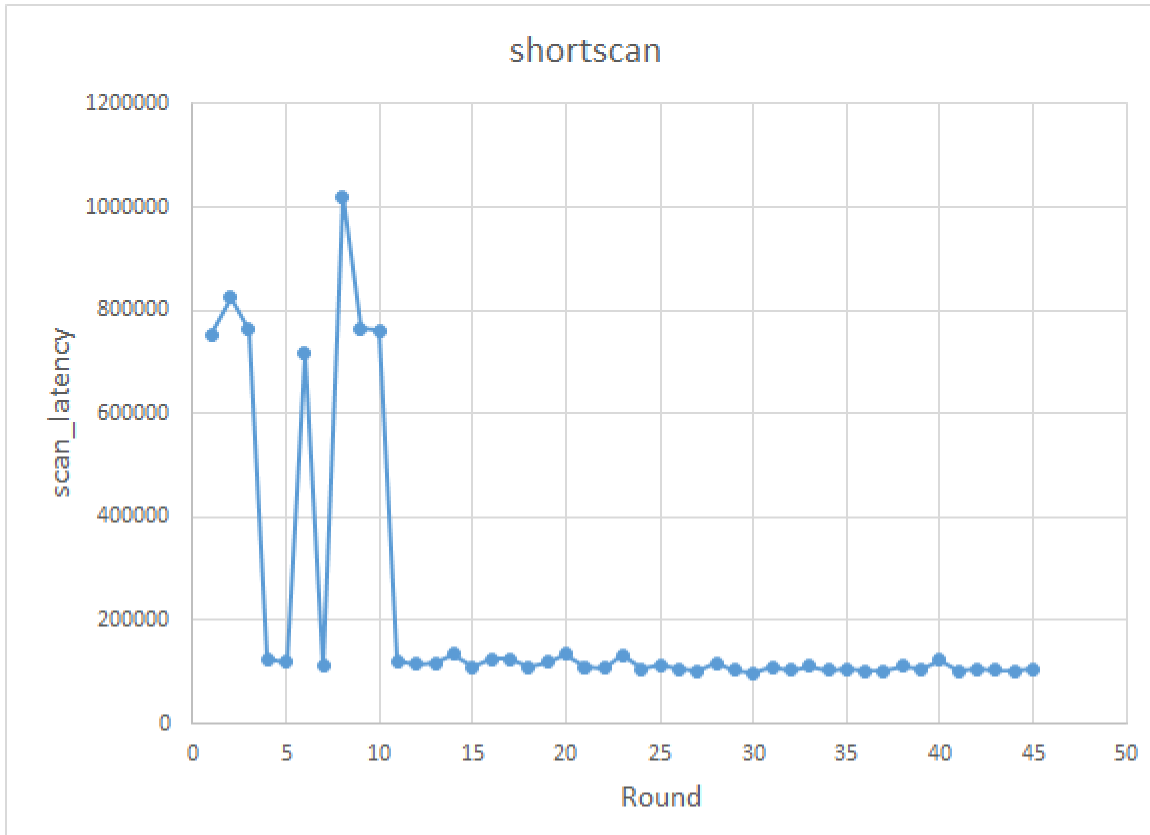
- workload=pntlookup80
- knobs={rocksdb.writecf.bloom-filter-bits-per-key, rocksdb.defaultcf.bloom-filter-bits-per-key, rocksdb.writecf.optimize-filters-for-hits, rocksdb.defaultcf.block-size, rocksdb.defaultcf.disable-auto-compactions}
- metric=get_throughput



More details:
https://www.cncf.io/blog/2019/12/10/autotikv-tikv-tuning-made-easy-by-ai-and-machine-learning/

Evaluation

- workload=shortscan
- knobs={'Bloom-filter-bits-per-key', 'optimize-filters-for-hits', 'block-size', 'disable-auto-compactions'}
- metric=scan_latency



More details:
https://www.cncf.io/blog/2019/12/10/autotikv-tikv-tuning-made-easy-by-ai-and-machine-learning/

Conclusion and limitations

Conlcusions:

- ML can help finding patterns that might be omitted by DBA
  - Some parameters have little effect on the results.
  - The effect of some parameters is in contrary to expectations.
  - Some workload may trigger other background operations that DBA does not know.

Limitations:

- Changing some knobs may need restarting DB.
  - -> CANNOT restart!
- Use static ycsb setting.
  - -> Workloads in production are dynamically changed

More details:
https://www.cncf.io/blog/2019/12/10/autotikv-tikv-tuning-made-easy-by-ai-and-machine-learning/

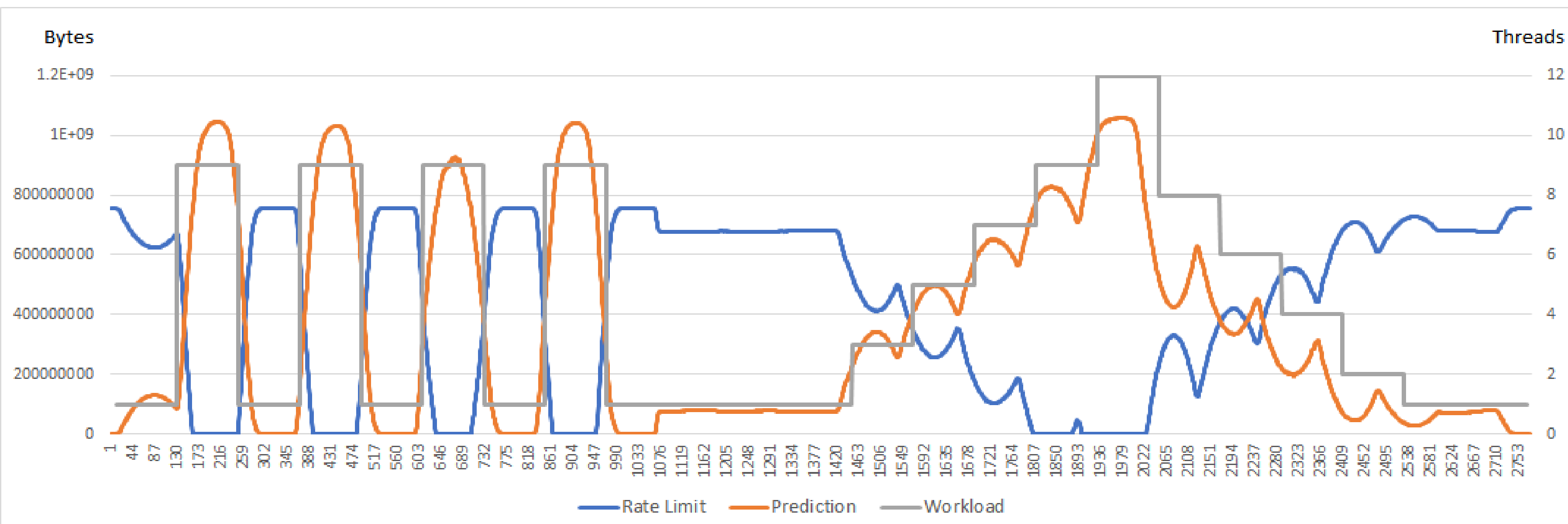
Auto-Tune RocksDB Rate Limiter

- Rate-Limiter: control the speed of background write operations, like compaction and flush.
  - https://github.com/facebook/rocksdb/wiki/Rate-Limiter
  - https://rocksdb.org/blog/2017/12/18/17-auto-tuned-rate-limiter.html
- Large/Burst write operations when doing compactions may cause a large read latency on user side.

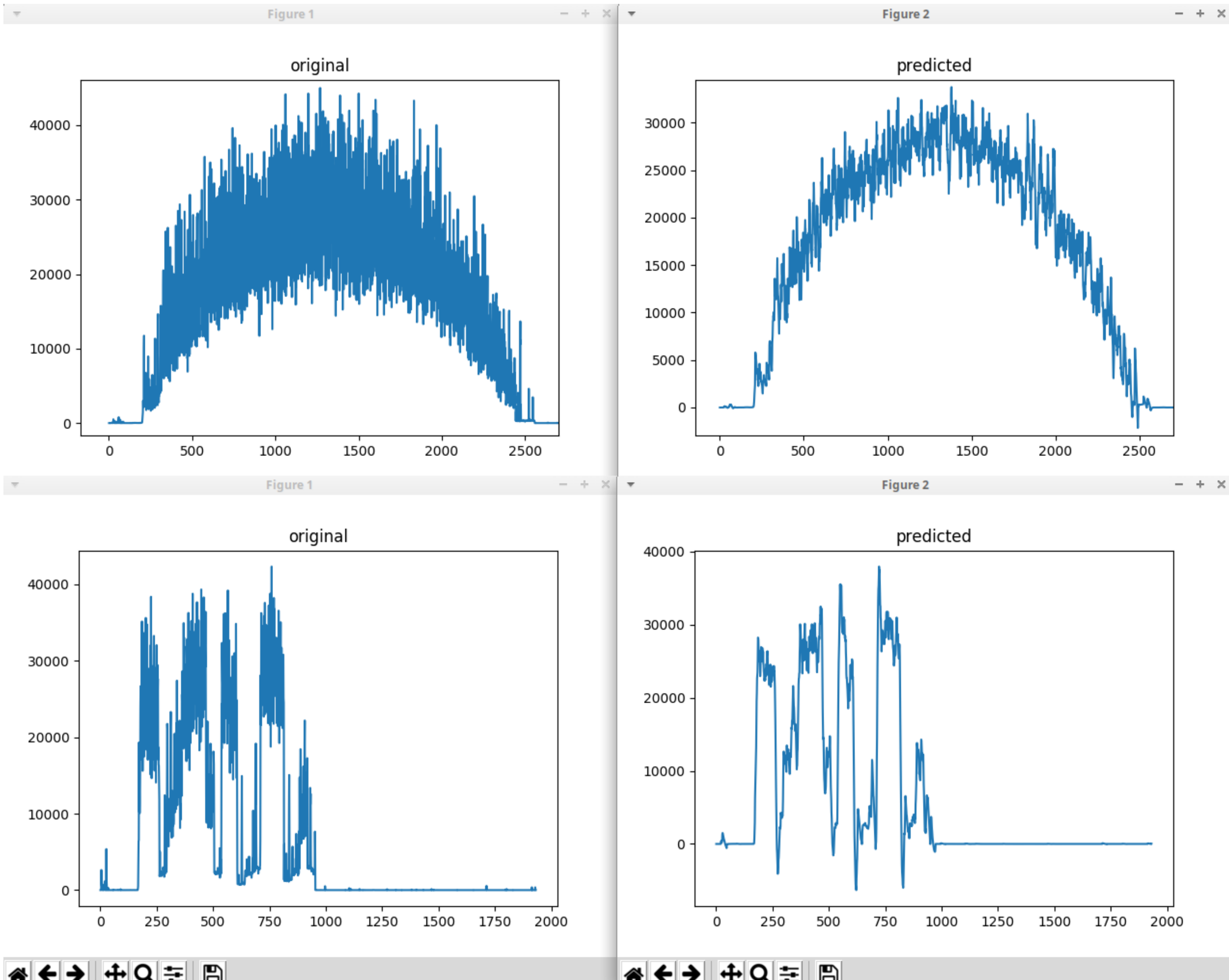
- Proposal:
  - Forecast the upcoming read I/O from user
  - Auto tune the upper-bound of rate limiter(write I/O) based on predicted value

Workload forecast

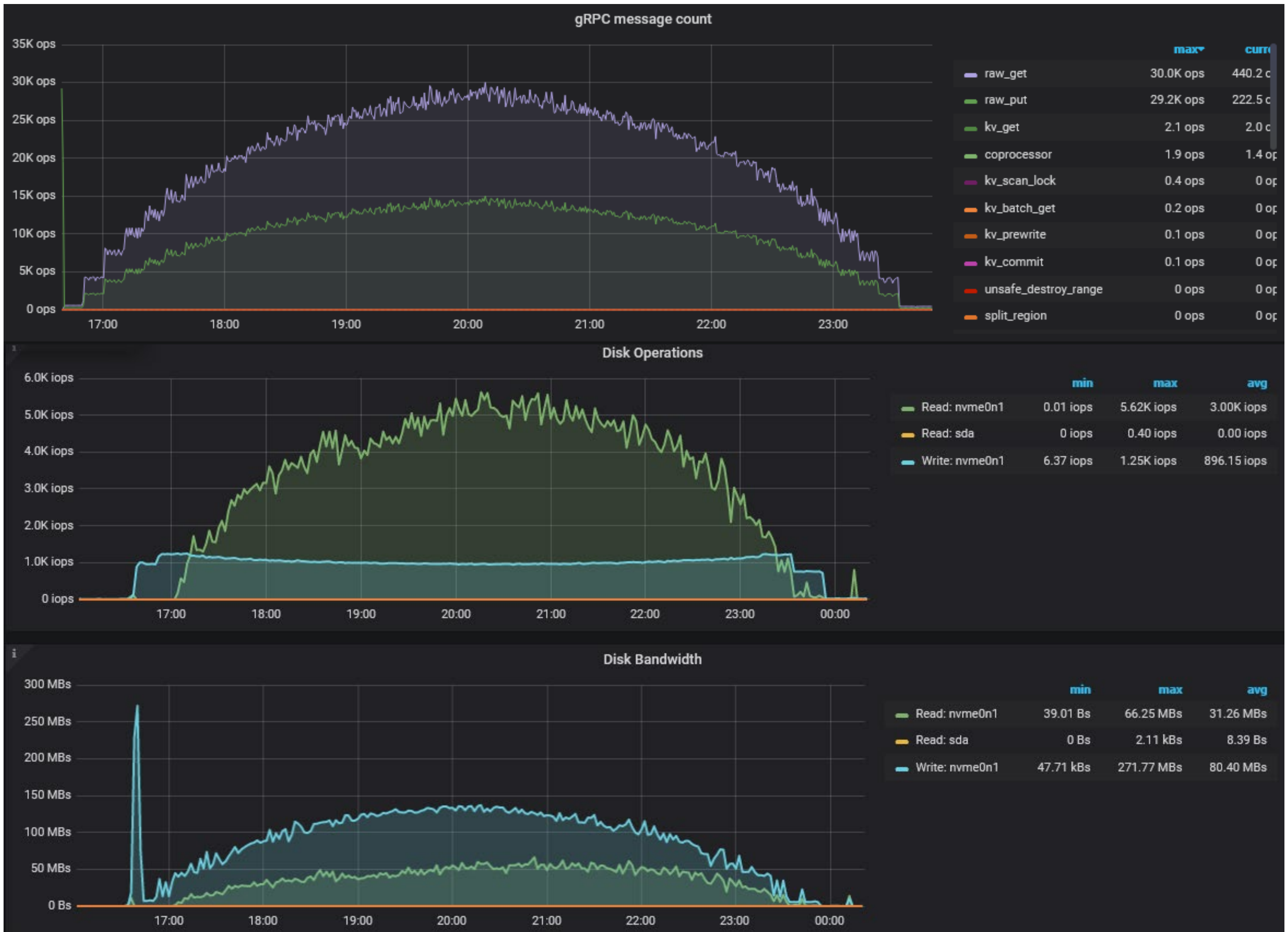
- Query-based Workload Forecasting for Self-Driving Database Management Systems [SIGMOD ‘18]
- Linear Regression in a recent time window
  - Workloads — real read workload threads
  - Prediction — predicted read I/O (Bytes)
  - Rate Limit — auto-tuned rate limiter value (Bytes)



Workload forecast



Workload forecast



Implementation

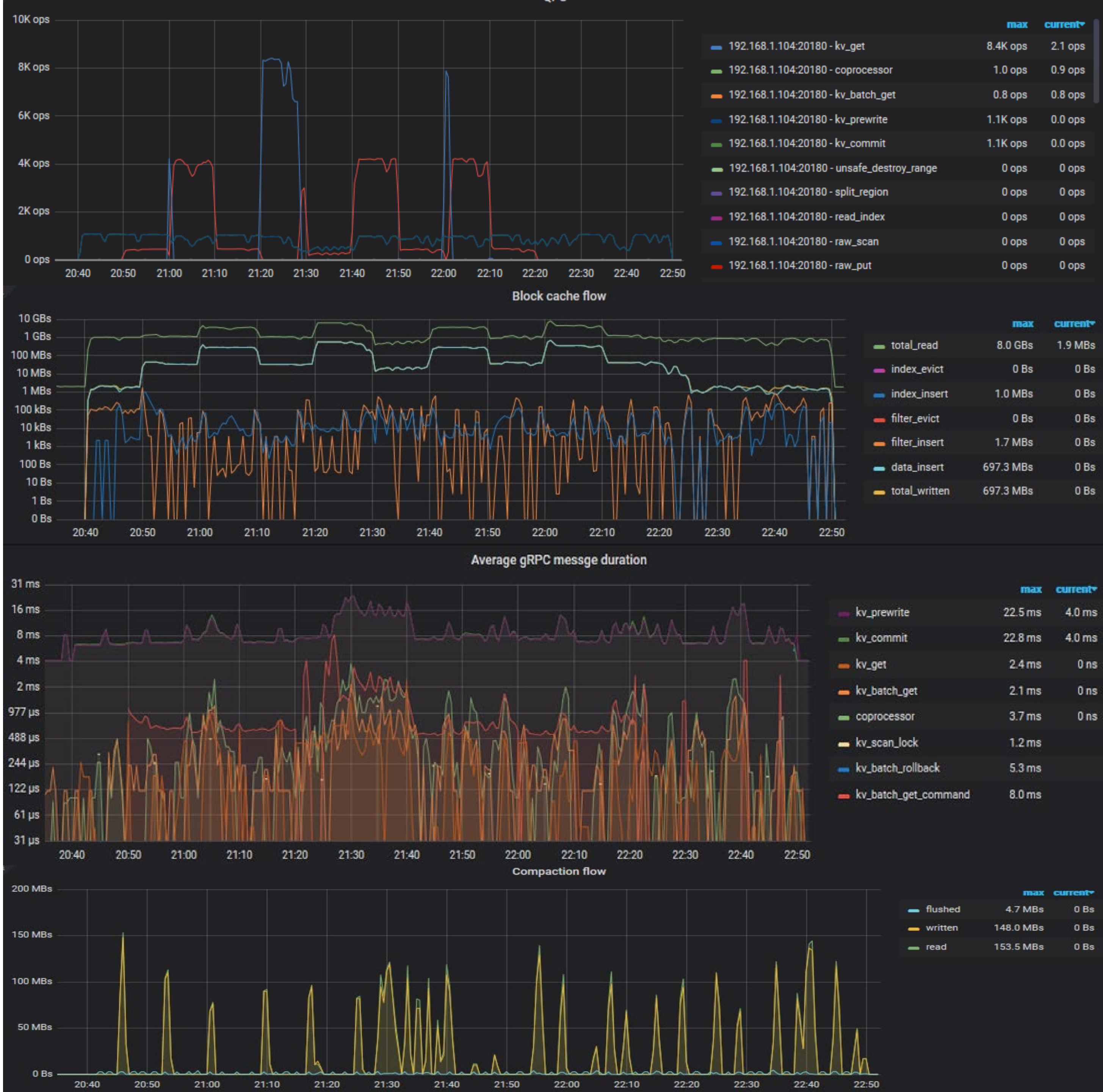
- Implemented in C++[RocksDB side] and Rust[TiKV side]
- Predict and re-config every 5 seconds

Evaluation #1

- Periodic read workload:
  - Round_1: 1 from 20:49:21 to 20:59:21
  - Round_2: 9 from 20:59:23 to 21:09:23
  - Round_3: 1 from 21:09:25 to 21:19:25
  - Round_4: 9 from 21:19:27 to 21:29:27
  - Round_5: 1 from 21:29:29 to 21:39:29
  - Round_6: 9 from 21:39:31 to 21:49:31
  - Round_7: 1 from 21:49:33 to 21:59:33
  - Round_8: 9 from 21:59:35 to 22:09:35
  - Round_9: 1 from 22:09:37 to 22:19:37
- Steady write workload

Evaluation #1

Without auto-tuned Rate Limiter

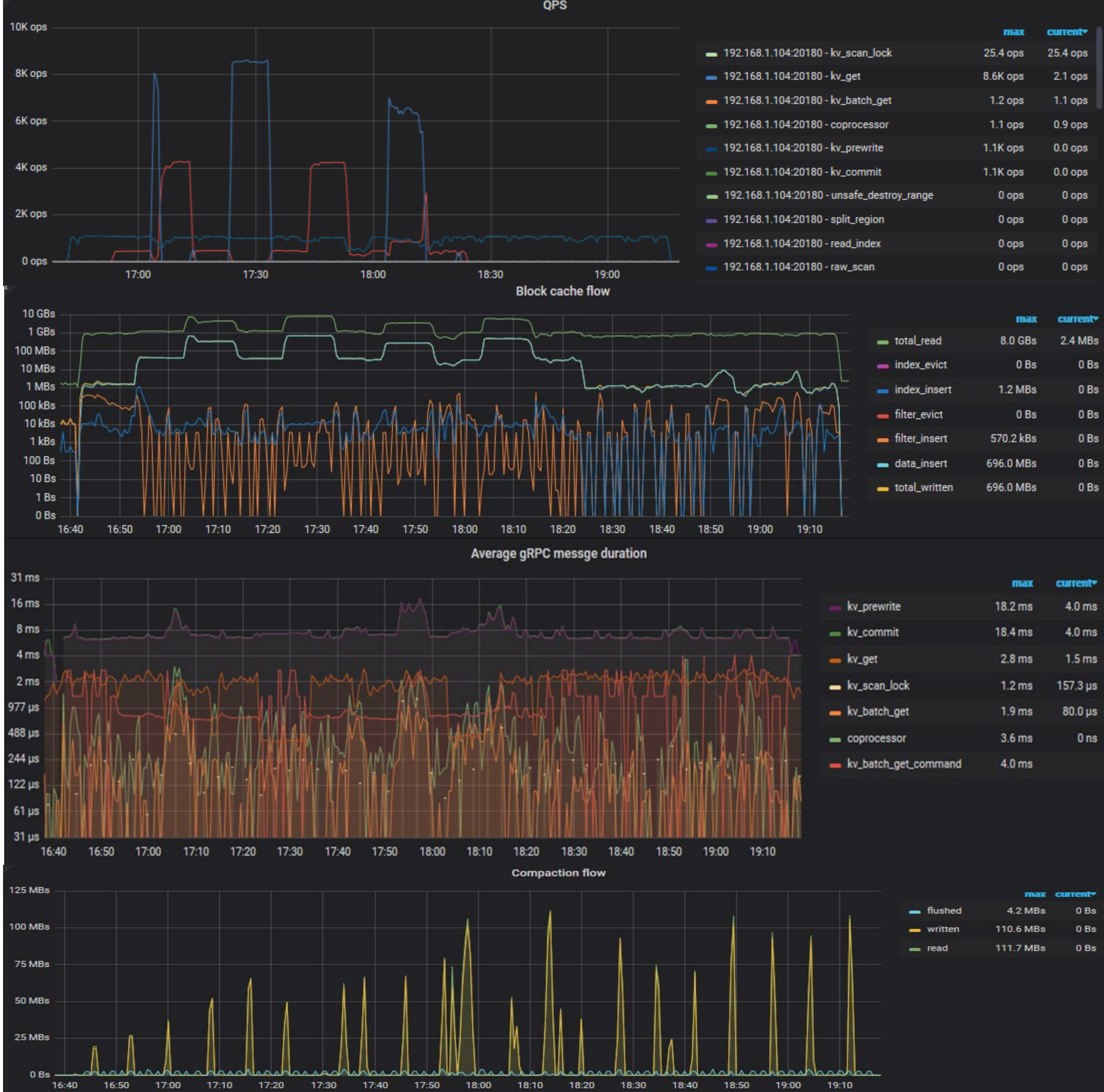


Evaluation #1

With auto-tuned Rate Limiter

Reduced Latency [average gRPC message duration]

Improved throughput [QPS]



Evaluation #2

- Running steady read workload, and suddenly inject a write workload(to trigger burst compaction/flush operations).

Evaluation #2

Without auto-tuned Rate Limiter

Evaluation #2

With auto-tuned Rate Limiter

Reduced Latency [average gRPC message duration]

Less fluctuation [QPS]

Potential future of self-driving database

- Self-driving
- Elastic (Automatically scale on cloud/serverless environment)