UNIVERSITY OF CAMBRIDGE

BOAT: Building Auto-Tuners with Structured Bayesian Optimization

Besp**O**ke **A**uto-**T**uners

Indigo Orton – R244

Computer Laboratory

Key idea

- Bespoke auto-tuners for systems
- Inject developer insight
- Faster tuning

Motivation – Configuration parameters

- Diversity of workloads – one size doesn't fit all
- Configuration tuning – non-trivial
- Optimal configuration – moving target

Motivation – Auto-Tuners

- Auto-tuners – tuning is not always intuitive
- Many iterations
- Costly performance evaluation, generic tuners, & you
- High dimensionality

Example – Cassandra

- Built for high throughput
- JVM based – garbage collection pauses
- Tuning garbage collection
- 99th percentile – 19ms -> 7ms

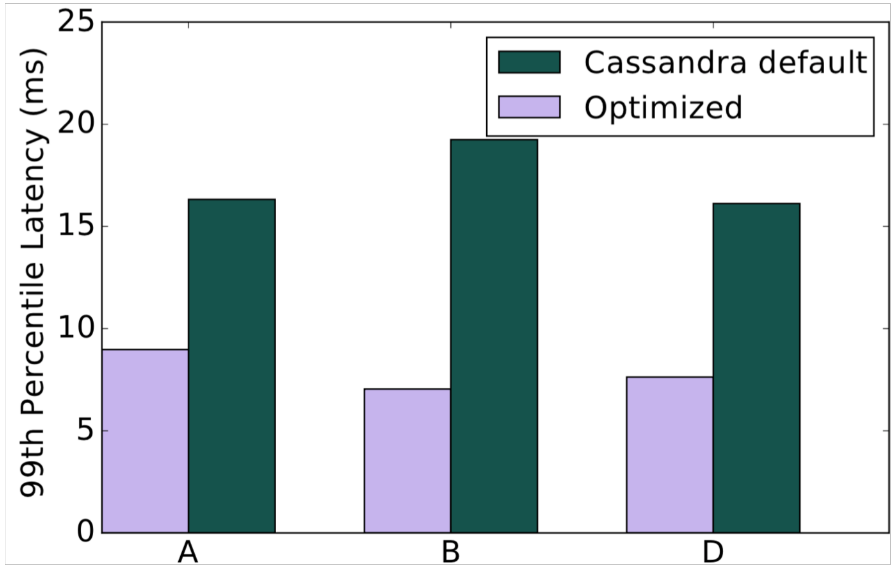

Figure from BOAT [1]

Example – Cassandra

- BOAT – within 10% of best after 2 iterations
- Spearmint [2] - 16 iterations, 4 hours

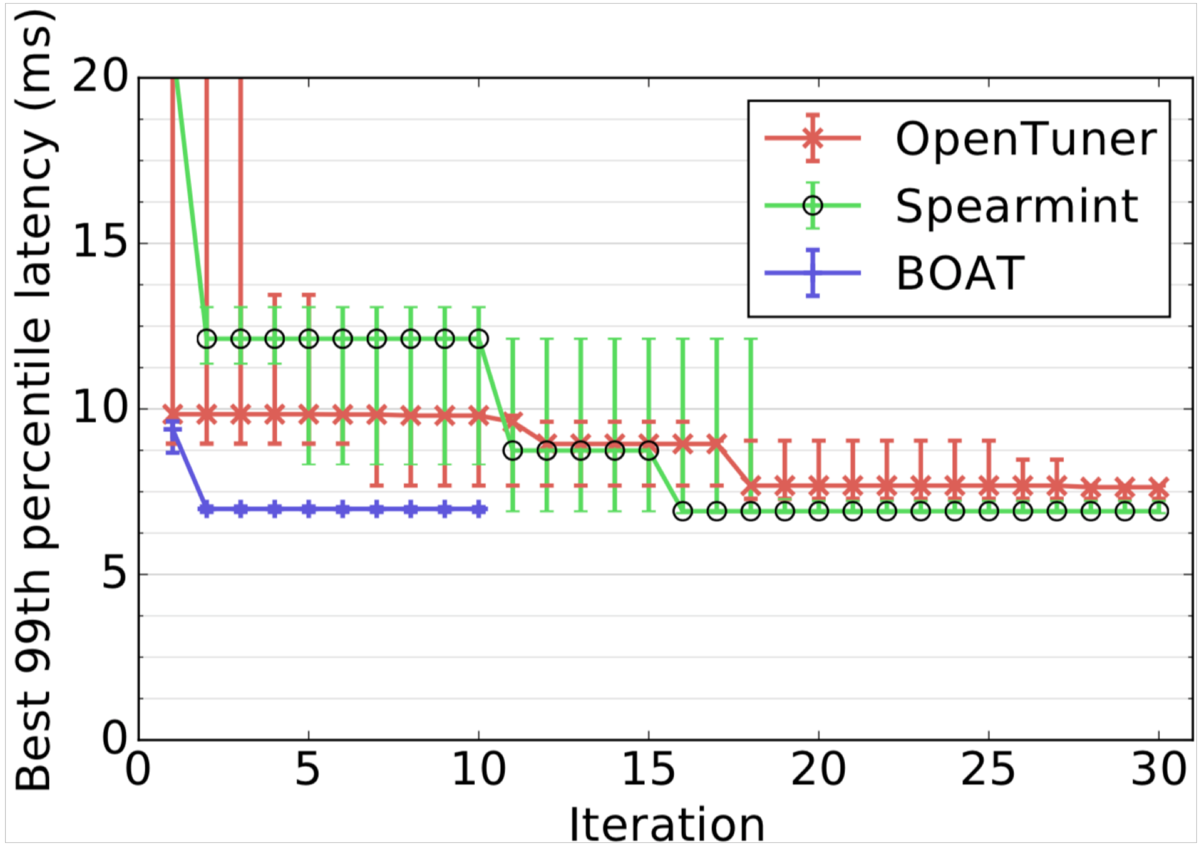

Figure from BOAT [1]

Details

Bayesian Optimization

- Basis of most generic auto-tuners
- Probabilistic modeling of objective function
- Gaussian process
- Curse of dimensionality – too many iterations

Structured Bayesian Optimization

- Extension of Bayesian Optimization
- Gaussian process -> developer structured probabilistic model
- Insight into objective function - incrementally
- Happy medium

Incremental structure

- Models Eden size
- Tuner models and minimizes latency
- Larger search space

```
struct GCRateModel : public SemiParametricModel<GCRateModel> {
  GCRateModel() {
    allocated_mbs_per_sec =
     std::uniform_real_distribution<>(0.0, 5000.0)(generator);
    // Omitted: also sample the GP parameters
  }
  double parametric(double eden_size) const {
    // Model the rate as inversly proportional to Eden's size
    return allocated_mbs_per_sec / eden_size;
  }
  double allocated_mbs_per_sec;
};
```

Figure from BOAT [1]

- Models latency
- Tuner minimizes set model of latency
- Smaller search space

```
struct CassandraModel : public DAGModel<CassandraModel> {
  void model(int ygs, int sr, int mtt){
    // Calculate the size of the heap regions
    double es = ygs * sr / (sr + 2.0);// Eden space's size
    double ss = ygs / (sr + 2.0);     // Survivor space's size
    // Define the dataflow between semi-parametric models
    double rate =     output("rate", rate_model, es);
    double duration = output("duration", duration_model,
                        es, ss, mtt);
    double latency =  output("latency", latency_model,
                        rate, duration, es, ss, mtt);
  }
  ProbEngine<GCRateModel> rate_model;
  ProbEngine<GCDurationModel> duration_model;
  ProbEngine<LatencyModel> latency_model;
};
```

Figure from BOAT [1]

Results – NN Training

- High dimensionality
- Optimize NN training
- Optimal distribution architecture based on available machines
- Communication time calculation (a max function) – hard to auto fit, easy to manually model
- 2 hour tuning time – large net benefit

Results – NN Training

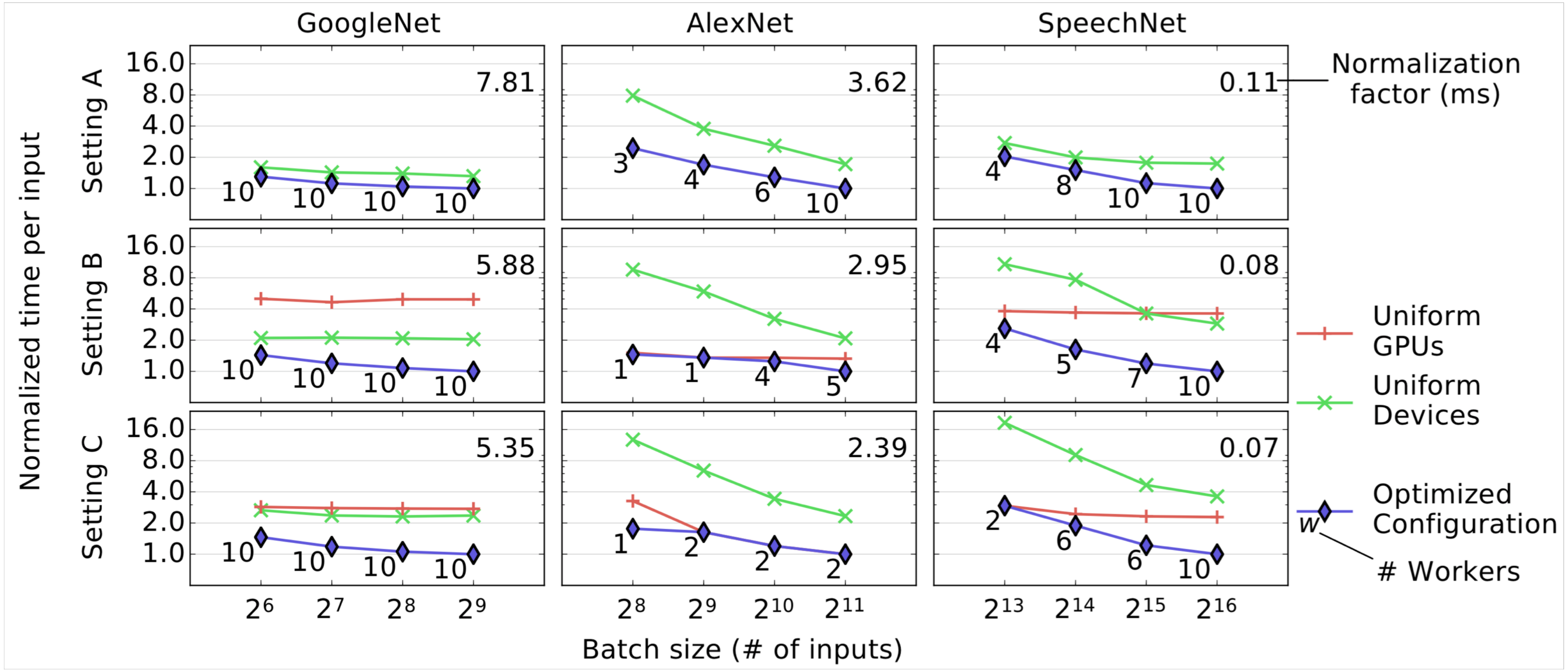

Figure from BOAT [1]

Context

- PetaBricks [4] – language based optimization
- OpenTuner [3] – domain specific search techniques
- Spearmint [2] – traditional Bayesian Optimization

Review

Encouraging highlights

- Practical integration of developer knowledge
- Retains benefits of auto-tuners
- Handles high dimensionality

Further questions

- Tuning the tuner
- Incremental structure – is there a heuristic?
- Model of parameters – configuration chooser

Conclusion

- Auto-tuning
- Inject developer insight
- Structured Bayesian Optimization
- Curse of dimensionality
- Happy medium

References

1. Dalibard, V., Schaarschmidt, M., & Yoneki, E. (2017). BOAT - Building Auto-Tuners with Structured Bayesian Optimization. *Www*, 479–488. http://doi.org/10.1145/3038912.3052662
2. Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian Optimization of Machine Learning Algorithms. *CoRR*, *stat.ML*.
3. Ansel, J., Kamil, S., Veeramachaneni, K., Ragan-Kelley, J., Bosboom, J., O'Reilly, U.-M., & Amarasinghe, S. P. (2014). OpenTuner - an extensible framework for program autotuning. *Pact*, 303–316. http://doi.org/10.1145/2628071.2628092
4. Ansel, J., Chan, C. P., Wong, Y. L., Olszewski, M., Zhao, Q., Edelman, A., & Amarasinghe, S. P. (2009). PetaBricks - a language and compiler for algorithmic choice. *Pldi*, 38. http://doi.org/10.1145/1542476.1542481