OOPredictor

Predicting Object-Oriented Accesses using Static Analysis

Hassan Arafat

David Bremner

Kenneth Kent

Julian Wang

2024/11/12

CDP @ CASCON

CAS–Atlantic

Motivation

- The memory/CPU gap is growing.
- Caches used to mitigate the effects of the gap.
- Cache performance depends on the locality of access.
- Modern CPUs attempt to schedule around long loads.
- If scheduling cannot buy time, the CPU is forced to stall.
- Load stalls cause reduced performance.

Background

- Prefetching: speculatively load data the CPU is expected to soon use.
- Better ordering of objects on memory, increasing spatial locality.
- Object-Oriented accesses cause a lot of hard-to-predict pointer chasing patterns.
- Prior knowledge of those patterns can be used by a myriad of optimizations.

Literature Review

- Data Aware GC that can use data collected during profiling to better guide the GC (Chilimbi et al. 1998; Chen et al. 2006; Serrano et al. 2009).
- Markov chains show the least fluctuations when modelling of database accesses of object-oriented applications (Garbatov and Cachopo 2011).
- Using static analysis to make regexes to model access patterns, then restructure the way objects in memory are allocated in C++ (Jeon et al. 2007).

Shortcomings

- Profiling fine-grain information is expensive, even when cost mitigation strategies are employed.
- Profiling can require an ahead-of-time profiling run.
- Previous work on prediction models didn't address memory Object-Oriented accesses, only database accesses.
- Regular Expressions are limited in the amount of information they provide.

Design goals

- Predict the Object-Oriented access pattern.
- No added profiling cost.
- Model branch biases.
- Markov chain output.
- Implement as an optimization within Eclipse OMR, used with Eclipse OpenJ9 .

Basic operation

- Static analysis is traditionally used to enable beneficial code transformation, e.g., code motion.
- Can be used to predict how the program accesses objects of a certain type.
- In JIT environments, readily available profiling data can be used to increase prediction accuracy.
- That information can then be used to find a better order of related objects in memory.

Statically Derive Access Patterns

- Static analysis at compile time.
- No effect on program running time.
- Process the Control Flow Graph (CFG) already built by the optimizer.
- Record accesses to object fields that are references.
- Block frequency information used to assign weights.
- Remove the blocks that don't have any Object-Oriented accesses while retaining their control flow information.

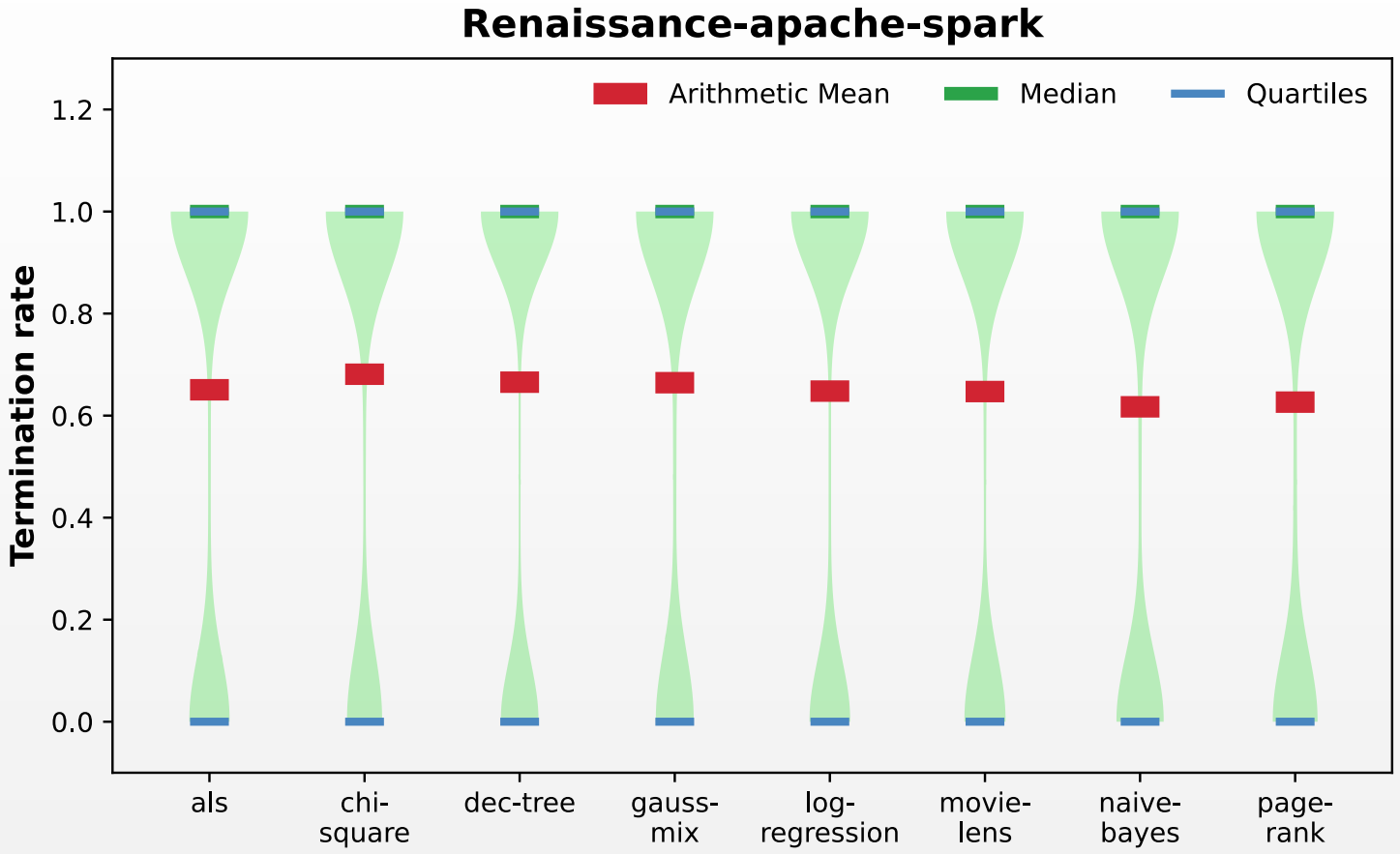
Evaluation

- Instrument the OpenJ9 interpreter to record every *getfield* and *putfield to reference fields.*
- *Perform two runs, one with the predictor optimization enabled, one with instrumented interpreter.*
- Evaluated our predictor using the Renaissance benchmark suite.
- Also use SPECJBB2015 and SPECJBB2005 for business-like workloads.

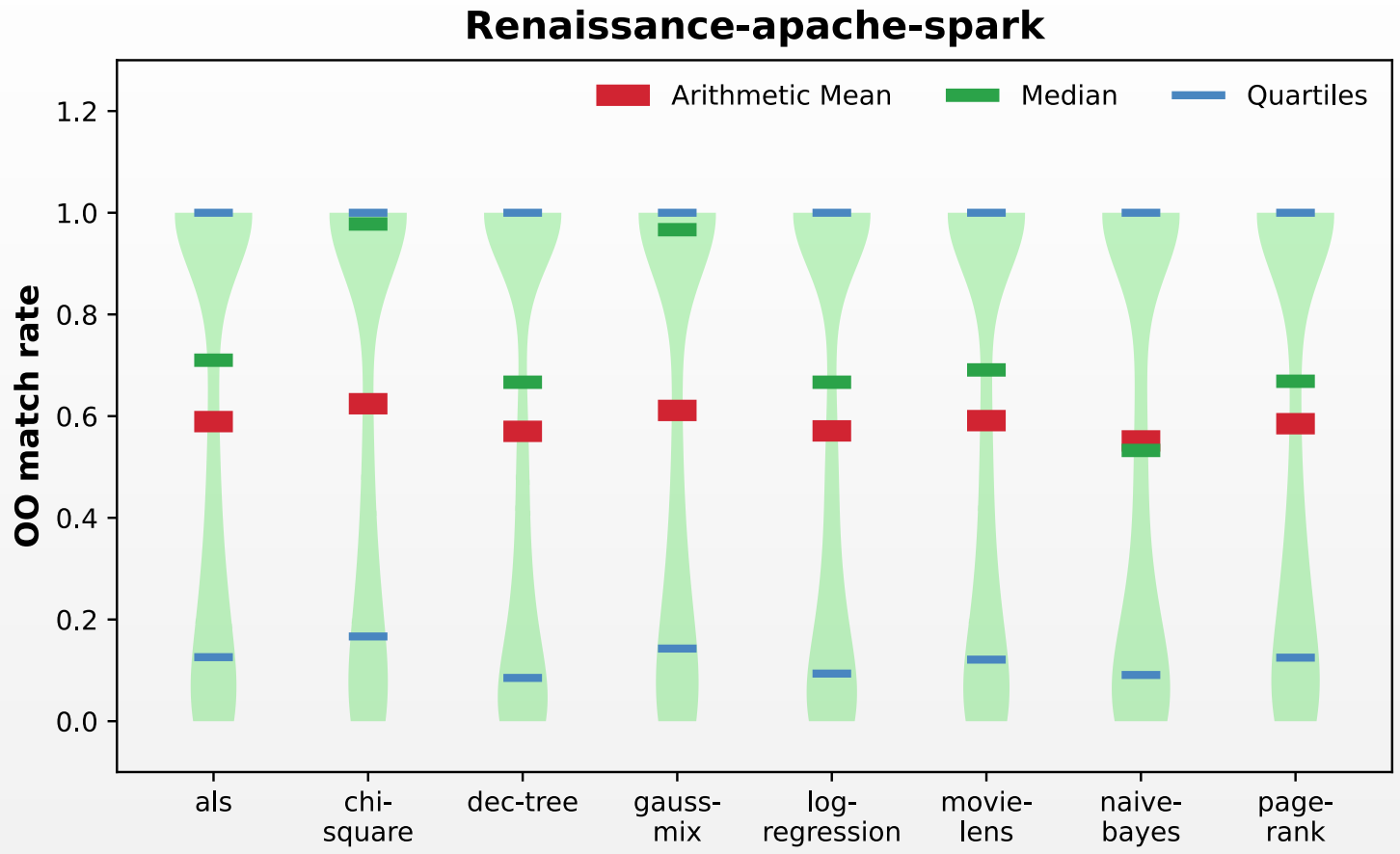
Metrics

- We use two metrics to characterize the accuracy of the predictor models:
  - Termination rate: The percentage of calls that ended with the model in a final state.
  - OO match rate: The percentage of object-oriented accesses that were correctly predicted by the model.
- We will be visualizing the data as violin plots that show the arithmetic mean, median, 1st quartile, 3rd quartile and a Kernel Density Estimation (KDE) on top.

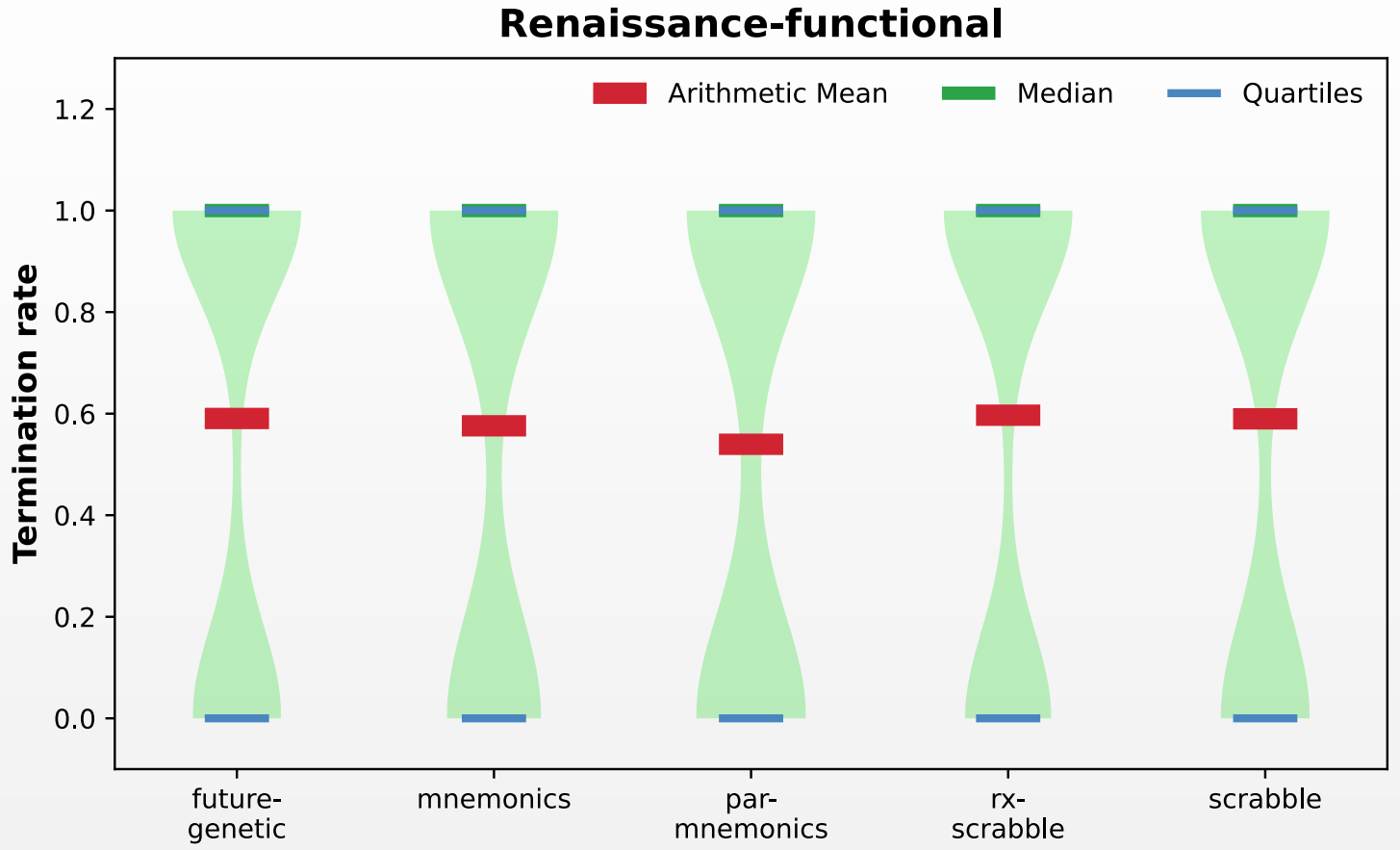
Apache-spark results



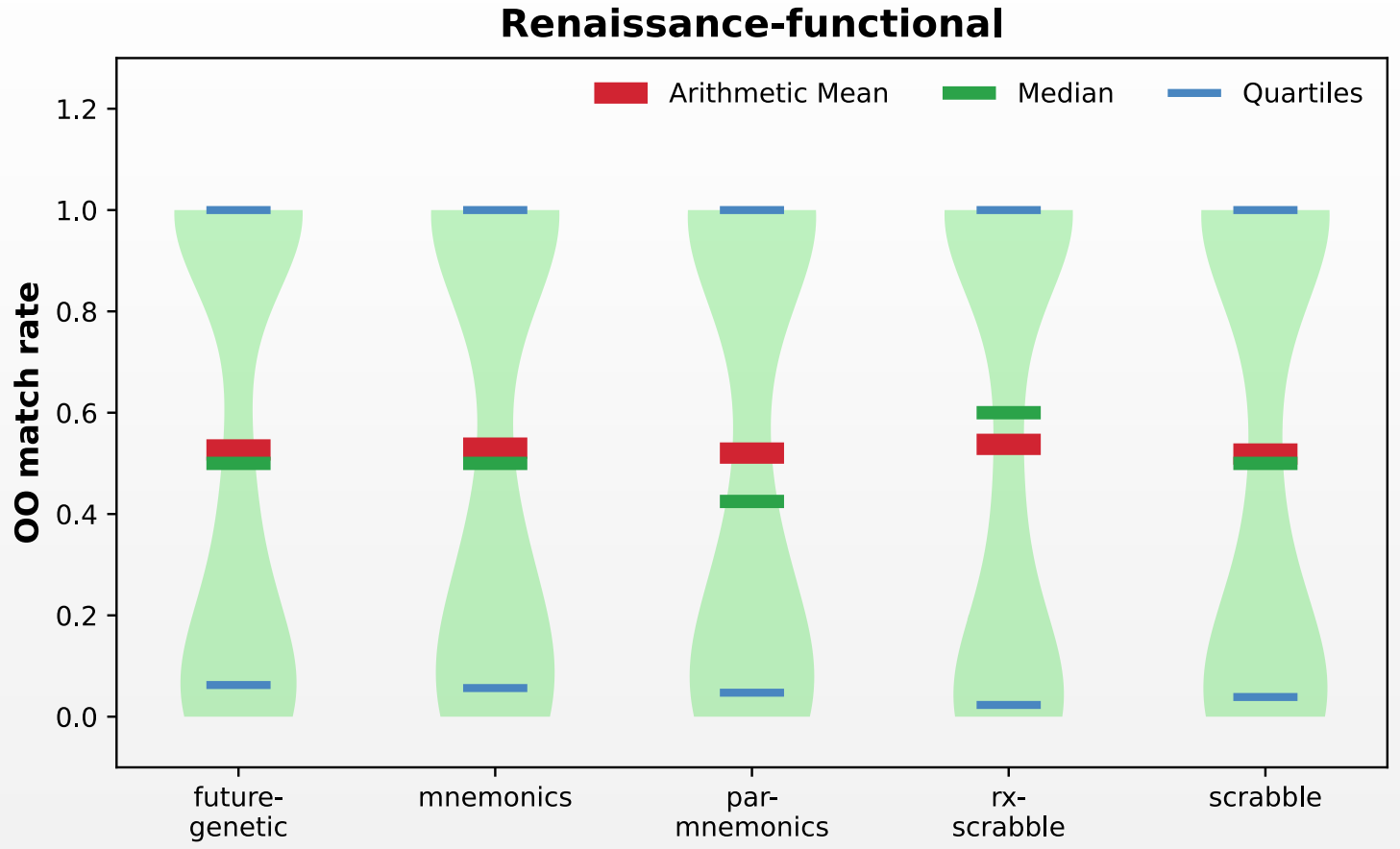
Apache-spark results, cont'd



Functional results



Functional results, cont'd



Conclusion

- Load stalls adversely affect performance.
- Current approaches require profiling/ access barriers.
- We can derive access patterns with static analysis.
- The predictor performs very well for some methods, but very poorly for others.
- The predictor can be used to guide minimally intrusive optimizations that have a low cost of wrong prediction.