

White-Box Analysis over Machine Learning: Modeling Performance of Configurable Systems

Supplementary Material

1 SUBJECT SYSTEMS

The following paragraphs describe each system in more detail, including the scenario considered for the evaluation:

- Apache Lucene¹ is a library providing powerful indexing and search features, as well as advanced analysis and tokenization capabilities. As a workload, we executed the IndexFiles demo, shipped with the system. Similarly to its nightly benchmarks that index large Wikipedia exports², we indexed a large publicly available 512.5MB Wikimedia dump.
- H2³ is a fast, embeddable database hosted on GitHub with over 2600 stars and 850 forks. As a workload, we executed the RunBenchC benchmark, shipped with the system, which is similar to the TPC-C benchmark. The benchmark includes multiple transaction types on a complex database with several tables and with a specific execution structure.
- Berkeley DB⁴ is a high-performance embeddable database providing key-value storage. As a workload, we executed the MeasuredDiskOrderedScan performance benchmark, shipped with the system, which populates a database with 500K key/value pairs.
- Density Converter⁵ is a popular image density converting tool, processing single or batches of images to Android, iOS, Windows, or CSS specific formats and density versions. It is hosted on GitHub and has over 220 stars and 29 forks. The system does not come with any tests or benchmarks, and only provides small sample images that are processed in a couple of seconds. As we want to execute the program with a long-running representative scenario to observe the influence of options on the system's performance, we processed a publicly available 5616×3744 pixel JPEG photo.

2 SMALL WORKLOAD

For each subject system, we reduced the workload size of their benchmarks: for Lucene, we trimmed the file to index from 8 million to 300 thousand lines; for H2, we changed the size parameter from 100 to 5; for Berkeley DB, we changed the `n_records` parameter from 500 thousand to 10; and for Density Converter, we reduced the size of the photo to process by 90%. We selected these smaller workloads with the goals of (1) keeping the workloads representative, in order that the programs perform the same operations as with the regular workloads and (2) *significantly* reduce the cost of running the iterative analysis.

¹<https://lucene.apache.org/>

²<https://home.apache.org/mikemccand/lucenebench/>

³<https://h2database.com/html/main.html>

⁴<https://www.oracle.com/database/berkeley-db/java-edition.html>

⁵<https://github.com/patrickfav/density-converter>

3 EVALUTION

The following list details the learners that we used in our evaluation, including the hyperparameters that we changed from the default values. We used Scikit learn version 0.23.2 and MatLab version R2019b update 5.

- Simple linear regression (SL): We used Scikit learn's `linear_model.LinearRegression` function. The function implements an ordinary least squares linear regression.
- Pair-wise simple linear regression (PSL): We used the same simple linear regression function above, but set `interaction_only=True` and `degree=2`.
- Lasso linear regression (LL): We used Scikit learn's `linear_model.Lasso` function. The function implements a linear model trained with L1 prior as regularize, also known as the Lasso.
- Lasso linear regression (PLL): We used the same Lasso linear regression function above, but set `interaction_only=True` and `degree=2`.
- Stepwise linear regression (SLR): We used Matlab's `stepwiselm` function. The function implements a stepwise regression. We set `modelspec='linear'`.
- Elastic net linear regression (EN): We used Scikit learn's `linear_model.ElasticNet` function. The function implements a linear regression with combined L1 and L2 priors as regularizer. We set `random_state=25`.
- Decision tree (DT): We used Scikit learn's `tree.DecisionTreeRegressor` function. The function implements a decision tree regressor. We set `random_state=25`.
- Shallow decision tree (SDT):. We used the same decision tree regressor function above, but set `max_depth=3` and `random_state=25`.
- Random forest (RF): We used Scikit learn's `ensemble.RandomForestRegressor` function. The function implements a random forest regressor. We set `random_state=25`.
- Multi-layer perceptron (NN): We used Scikit learn's `neural_network.MLPRegressor` function. The function implements a multi-layer Perceptron regressor. We set `random_state=25`.

Table 1. MAPE comparison.

| Approach | Apache Lucene | H2 | Berkeley DB | Density Converter |
|-------------|----------------|----------------|----------------|-----------------------------|
| R50 & SL | 5.2 | 180.1 | 45.6 | 623.1 |
| R200 & SL | 4.3 | 186.6 | 32.7 | 424.8 |
| FW & SL | 7.7 | 107.5 | 104.3 | 1466.7 |
| PW & SL | 7.0 | 129.5 | 48.9 | 1430.0 |
| R50 & PSL | 6.3 | 133.4 | 39.7 | 567.1 |
| R200 & PSL | 4.5 | 131.9 | 18.3 | 1030.3 |
| FW & PSL | 7.7 | 107.5 | 550.7 | 1466.7 |
| PW & PSL | 11.7 | 109.3 | 135.0 | 1354.2 |
| R50 & LL | 6.3 | 106.5 | 39.0 | 451.8 |
| R200 & LL | 7.1 | 80.6 | 46.3 | 243.5 |
| FW & LL | 9.1 | 106.5 | 117.6 | 1651.5 |
| PW & LL | 9.2 | 109.0 | 111.2 | 1451.6 |
| R50 & PLL | 6.3 | 106.52 | 44.7 | 451.8 |
| R200 & PLL | 7.1 | 80.5 | 46.3 | 243.5 |
| FW & PLL | 9.1 | 106.5 | 117.6 | 1651.5 |
| PW & PLL | 9.2 | 109.0 | 111.2 | 1451.6 |
| R50 & SLR | 4.5 | 124.1 | 19.7 | 1037.2 |
| R200 & SLR | 2.9 | 93.9 | 14.9 | 434.5 |
| FW & SLR | 7.9 | 129.3 | 768.7 | 1596.0 |
| PW & SLR | 4.7 | 113.3 | 34.2 | 1596.0 |
| R50 & EN | 6.5 | 145.3 | 47.4 | 411.3 |
| R200 & EN | 6.9 | 176.0 | 48.2 | 293.7 |
| FW & EN | 9.1 | 106.5 | 118.1 | 599.1 |
| PW & EN | 9.2 | 109.0 | 108.5 | 660.8 |
| R50 & DT | 0.5 | 1.1 | 11.22 | 59.8 |
| R200 & DT | 0.5 | 0.9 | 1.0 | 2.2 |
| FW & DT | 11.5 | 125.1 | 143.12 | 1768.7 |
| PW & DT | 1.0 | 127.7 | 36.9 | 7.3 |
| R50 & SDT | 0.4 | 1.1 | 16.6 | 81.8 |
| R200 & SDT | 0.3 | 1.1 | 15.1 | 20.1 |
| FW & SDT | 11.5 | 124.8 | 144.3 | 1770.0 |
| PW & SDT | 1.0 | 126.8 | 42.0 | 18.2 |
| R50 & RF | 0.8 | 6.5 | 14.1 | 268.7 |
| R200 & RF | 0.3 | 0.7 | 1.1 | 5.5 |
| FW & RF | 8.7 | 119.0 | 106.1 | 1185.9 |
| PW & RF | 4.0 | 124.6 | 53.5 | 403.6 |
| R50 & NN | 31.2 | 152.5 | 72.4 | 415.7 |
| R200 & NN | 31.3 | 212.0 | 80.5 | 289.0 |
| FW & NN | 56.3 | 440.9 | 433.2 | 1265.4 |
| PW & NN | 65.8 | 406.1 | 464.2 | 1848.1 |
| Comprex (O) | 8.0 | 9.3 | 6.7 | 11.5 |
| Comprex (C) | 3.2±0.2 | 2.9±0.5 | 5.0±0.6 | 10.6±2.4¹ |

SL: Simple linear regression; PSL: Pair-wise simple linear regression; LL: Lasso linear regression; PLL: Pair-wise lasso linear regression; SLR: Stepwise linear regression; EN: Elastic net linear regression; DT: Decision tree; SDT: Shallow decision tree; RF: Random forest; NN: Multi-layer perceptron; R50: 50 random configurations; R200: 200 random configurations; FW: Feature-wise; PW: Pair-wise; O: Original model; C: Model corrected for profiler overhead, reporting mean and standard deviations over 5 corrections. **Bolded** values in **cells** indicate similarly low errors.

Table 2. Number of linear model terms with coefficients > 0.3 secs.

| Approach | Apache Lucene | H2 | Berkeley DB | Density Converter |
|------------|---------------|----|-------------|-------------------|
| R50 & SLR | 35 | 25 | 28 | 44 |
| R200 & SLR | 26 | 13 | 28 | 30 |
| FW & SLR | 5 | 4 | 15 | 11 |
| PW & SLR | 9 | 9 | 13 | 46 |
| Complex | 27 | 33 | 77 | 47 |

SLR: Stepwise linear regression; R50: 50 random configurations; R200: 200 random configurations; FW: Feature-wise; PW: Pair-wise;

4 DEBUGGING PARTITIONS

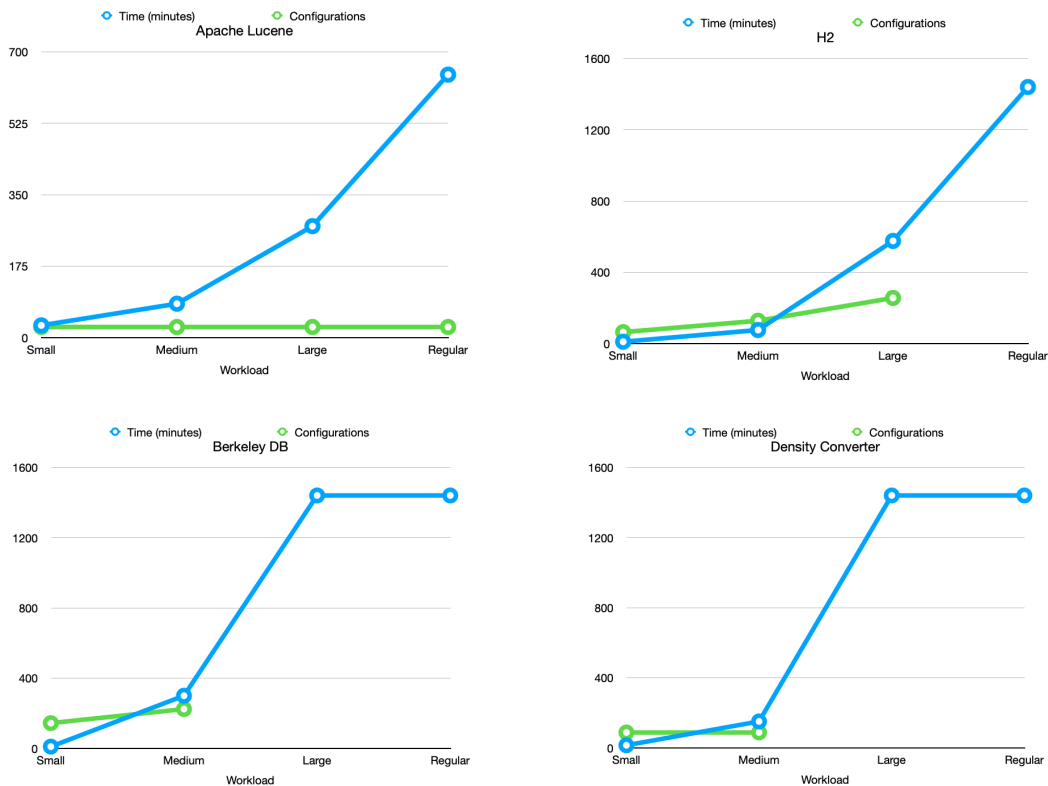


Fig. 1. Impact of workload size on the cost of the iterative dynamic analysis.