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 Lucene[1] is a library providing powerful indexing and search features, as well as advanced analysis/tokenization capabilities. As a workload, we executed the `IndexFiles` demo, shipped with the system. Similarly to its nightly benchmarks that index large Wikipedia exports[2], we indexed a large publicly available 512.5MB Wikimedia dump.
- H2[3] is a fast, embeddable database hosted on GitHub with over 2200 stars and 710 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 DB[4] is a high-performance embeddable database providing key-value storage. As a workload, we executed the `MeasureDiskOrderedScan` performance benchmark, shipped with the system, which populates a database with 500K key/value pairs.
- Density Converter[5] 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 26 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 8M to 300K lines; for H2, we changed the `size` parameter from 100 to 5; for Berkeley DB, we changed the `n_records` parameter from 500K 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

Author's address:

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 (L): We used Scikit learn's `linear_model.LinearRegression` function. The function implements an ordinary least squares linear regression.
- Pair-wise simple linear regression (LP): We used the same simple linear regression function above, but set `interaction_only=True` and `degree=2`.
- Lasso linear regression (LS): 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 (LSP): We used the same Lasso linear regression function above, but set `interaction_only=True` and `degree=2`.
- Stepwise linear regression (SWL): We used Matlab's `stepwiselm` function. The function inplments 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 decission tree regressor function above, but set `max_depth=3` and `random_state=25`. Note that a decision tree of this depth is potentially much easier to interpret than other trees and could be considered as interpretable as linear models.
- 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 & L	**5.2**	180.1	45.6	623.1
R200 & L	**4.3**	186.6	32.7	424.8
FW & L	**7.7**	107.5	104.3	1466.7
PW & L	**7.0**	129.5	48.9	1430.0
R50 & LP	**6.3**	133.4	39.7	567.1
R200 & LP	**4.5**	131.9	18.3	1030.3
FW & LP	**7.7**	107.5	550.7	1466.7
PW & LP	11.7	109.3	135.0	1354.2
R50 & LS	**6.3**	106.5	39.0	451.8
R200 & LS	**7.1**	80.6	46.3	243.5
FW & LS	**9.1**	106.5	117.6	1651.5
PW & LS	**9.2**	109.0	111.2	1451.6
R50 & LSP	**6.3**	106.52	44.7	451.8
R200 & LSP	**7.1**	80.5	46.3	243.5
FW & LSP	**9.1**	106.5	117.6	1651.5
PW & LSP	**9.2**	109.0	111.2	1451.6
R50 & SWL	**4.5**	124.1	19.7	1037.2
R200 & SWL	**2.9**	93.9	14.9	434.5
FW & SWL	**7.9**	129.3	768.7	1596.0
PW & SWLL	**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±2.4**[1]

L: Simple linear regression; LS: Lasso linear regression; LR: 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. We excluded results for SPLatDelayed as it behaves similarly to Brute-Force.

Table 2. Linear correlation coefficient comparison.

Approach	Apache Lucene	H2	Berkeley DB	Density Converter
R50 & L	0.816	0.730	0.748	0.463
R200 & L	0.857	0.792	0.821	0.657
FW & L	0.599	0.476	0.551	0.437
PW & L	0.628	0.478	0.753	0.455
R50 & LP	0.759	0.775	0.730	0.478
R200 & LP	0.855	**0.925**	**0.961**	0.564
FW & LP	0.599	0.476	-0.338	0.437
PW & LP	0.571	0.819	0.846	0.667
R50 & LS	0.781	0.835	0.797	0.655
R200 & LS	0.781	0.964	0.788	0.840
FW & LS	0.000	0.000	0.000	0.305
PW & LS	0.000	0.000	0.000	0.346
R50 & LSP	0.781	0.835	0.797	0.655
R200 & LSP	0.781	**0.964**	0.788	0.840
FW & LSP	0.000	0.000	0.000	0.305
PW & LSP	0.000	0.000	0.000	0.346
R50 & SWL	0.874	**0.919**	**0.951**	0.556
R200 & SWL	**0.948**	**0.971**	**0.977**	0.870
FW & SWL	0.534	0.456	-0.418	0.408
PW & SWL	0.870	0.877	**0.930**	0.667
R50 & EN	0.820	0.713	0.794	0.541
R200 & EN	0.781	0.807	0.808	0.664
FW & EN	0.000	0.000	-0.067	0.350
PW & EN	0.117	0.000	0.605	0.434
R50 & DT	**0.998**	**1**	**0.951**	**0.976**
R200 & DT	**0.999**	**1**	**1**	**0.995**
FW & DT	0.215	0.292	0.090	0.314
PW & DT	**0.994**	0.575	**0.902**	**0.994**
R50 & SDT	**0.999**	**1**	**0.955**	0.890
R200 & SDT	**0.999**	**1**	**0.969**	**0 900**
FW & SDT	0.216	0.294 0.097	0.313	
PW & SDT	**0.994**	0.584	0.899	**0.901**
R50 & RF	**0.995**	**0.978**	**0.969**	**0.914**
R200 & RF	**0.999**	**1**	**1**	**0.997**
FW & RF	0.369	0.413	0.408	0.347
PW & RF	0.862	0.614	0.832	0.898
R50 & NN	0.403	0.507	-0.394	0.387
R200 & NN	0.408	0.491	-0.376	0.490
FW & NN	0.397	0.006	-0.567	0.020
PW & NN	0.397	0.021	-0.548	0.381
Comprex	**0.919**	**1**	**0.996**	**0.991**

L: Simple linear regression; LS: Lasso linear regression; LR: 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; **Bolded** values in cells indicate similarly high coefficients. We excluded results for SPLatDelayed as it behaves similarly to Brute-Force.

Table 3. Number of linear model terms with coefficients > 0.3 secs.

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

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

4 DEBUGGING PARTITIONS

We used a debugging strategy to identify potential effects of inaccurate partitions. We expect that multiple executions of configurations within the same subspace of a region's partition have the same performance behavior. Observing significant performance differences for executions of different configurations within the same subspace, beyond normal measurement noise, indicate that the region's partition might be inaccurate.

Specifically, we analyzed the performance measurements of all regions, searching for regions with a significant performance influence (> 0.1ms in any observed configuration) and with a high variance among the execution times of the same subspace in a region (coefficient of variation of the execution times > 1.0). Among the 2263 method level regions that we analyzed across all systems, we found only 8 of such regions, all in Lucene.

We executed the iterative analysis with the regular workload and compared the partitions identified in those regions, to investigate whether the inaccurate partitions were caused by using a small workload. In 7 out of the 8 regions, we traced the issue to missed subspaces due to slightly different taints in the shorter workload. Correcting the partitions in those 7 regions and rebuilding the model decreased the MAPE from 8.0 to 7.4 and increased the linear correlation from 0.919 to 0.965. However, the time to run the taint analysis with the regular workload is *extremely* expensive; 11 hours instead of 29 minutes to run the same 26 configurations! We argue that the extremely high cost to obtain the missing subspaces does not outweigh the slight increase in accuracy of the generated model.

In addition to Lucene, we also executed the iterative analysis in the other systems with various workload sizes, to determine the impact of the workload size on the execution of the analysis. In all systems, except for Lucene, the iterative analysis did not finish executing with the regular workload after 24 hours, with numerous subspaces left unexplored. For instance, Berkeley DB only executed 117 configurations in 24 hours, whereas 144 configurations were executed in 11 minutes with the small workload. Similarly to Lucene, we observed that larger workloads resulted in different taints in some subspaces and, in some cases, an increase on the number of configurations to execute. Nevertheless, we argue that the extremely high cost and inability to use the regular workload to run the iterative analysis, in 3 out of our 4 subject systems, does not outweigh a potential slight increase in the accuracy of the already highly accurate models that we generated.

These results support our conjecture that inaccuracies on the regions' partition caused by using an unsound analysis with a small workload are a minor issue in practice.

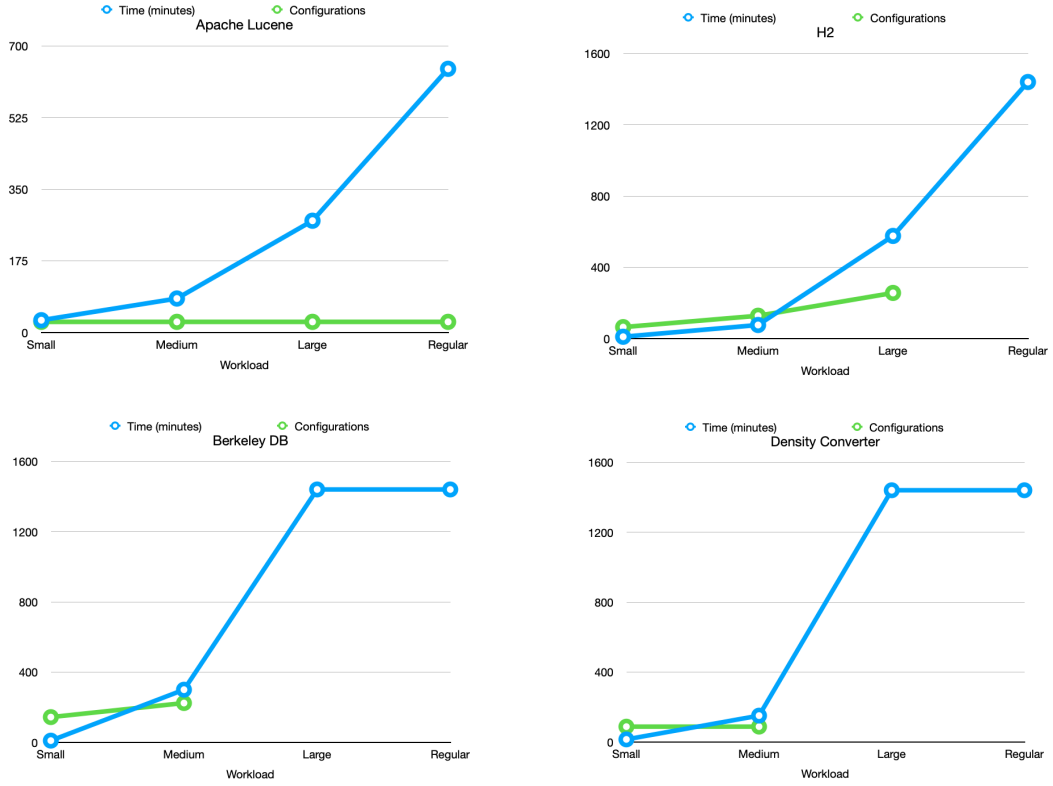

Fig. 1. Impact of workload size on the cost of the iterative dynamic analysis.