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

Supplementary Material

MIGUEL VELEZ, Carnegie Mellon University, USA
POOYAN JAMSHIDI, University of South Carolina, USA
NORBERT SIEGMUND, Leipzig University, Germany
SVEN APEL, Saarland University, Germany
CHRISTIAN KÄSTNER, Carnegie Mellon University, USA

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 and 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. The options that we considered are:
  - `RAM_BUFFER_SIZE_MB`
  - `MERGE_POLICY`
  - `MERGE_SCHEDULER`
  - `COMMIT_ON_CLOSE`
  - `CHECK_PENDING_FLUSH_UPDATE`
  - `READER_POOLING`
  - `MAX_BUFFERED_DOCS`
  - `CODEC`
  - `USE_COMPOUND_FILE`
  - `INDEX_DELETION_POLICY`
  - `MAX_TOKEN_LENGTH`
  - `MAX_CFS_SEGMENT_SIZE_MB`
  - `NO_CFS_RATIO`
- H2[3] 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. The options that we considered are:
  - `FILE_LOCK`
  - `AUTOCOMMIT`
  - `ACCESS_MODE_DATA`
  - `CIPHER`
  - `CACHE_TYPE`
  - `CACHE_SIZE`

[1] https://lucene.apache.org/
[2] https://home.apache.org/ mikemccand/lucenebench/
[3] https://h2database.com/html/main.html

- MV_STORE
- ANALYZE_AUTO
- DEFRAG_ALWAYS
- IF_EXISTS
- FORBID_CREATION
- IGNORE_UNKNWON_SETTING
- PAGE_SIZE
- ANALYZE_SAMPLE
- COMPRESS
- OPTIMIZE_DISTINCT

- 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. The options that we considered are:
  - DUPLICATES
  - SEQUENTIAL
  - MAX_MEMORY
  - ENV_SHARED_CACHE
  - REPLICATED
  - ENV_IS_LOCKING
  - CACHE_MODE
  - TEMPORARY
  - JE_FILE_LEVEL
  - ENV_BACKGROUND_READ_LIMIT
  - LOCK_DEADLOCK_DETECT
  - TXN_SERIALIZABLE_ISOLATION
  - JE_DURABILITY
  - ADLER32_CHUNK_SIZE
  - CHECKPOINTER_BYTES_INTERVAL
  - LOCK_DEADLOCK_DETECT_DELAY
- 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 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. The options that we considered are:
  - SCALE
  - SCALE_IS_HEIGHT_DP
  - COMPRESSION_QUALITY
  - OUT_COMPRESSION
  - PLATFORM
  - UPSCALING_ALGO
  - DOWNSCALING_ALGO
  - ROUNDING_MODE

[4] https://www.oracle.com/database/berkeley-db/java-edition.html

[5] https://github.com/patrickfav/density-converter

  - `SKIP_UPSCALING`
  - `SKIP_EXISTING`
  - `ANDROID_INCLUDE_LDPI_TVDPI`
  - `VERBOSE`
  - `ANDROID_MIPMAP_INSTEAD_OF_DRAWABLE`
  - `ANTI_ALIASING`
  - `POST_PROCESSOR_PNG_CRUSH`
  - `POST_PROCESSOR_WEBP`
  - `DRY_RUN`
  - `POST_PROCESSOR_MOZ_JPEG`
  - `KEEP_ORIGINAL_POST_PROCESSED_FILES`
  - `IOS_CREATE_IMAGESET_FOLDERS`
  - `CLEAN`
  - `HALT_ON_ERROR`

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.

3 EVALUTION

The following list details the learners that we used in our evaluation, including the hyperparameters that we changed from the default values.

- 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.
- Shallow decision tree (SDT):. We used the same decission tree regressor function above, but set `max_depth=6`.
- Decision tree (DT): We used Scikit learn's `tree.DecisionTreeRegressor` function. The function implements a decision tree regressor.
- Random forest (RF): We used Scikit learn's `ensemble.RandomForestRegressor` function. The function implements a random forest regressor.
- Multi-layer perceptron (NN): We used Scikit learn's `neural_network.MLPRegressor` function. The function implements a multi-layer Perceptron regressor.

Similar to linear models, decision trees are usually considered as inherently interpretable. However, if trees grow too large, humans have more difficulty understanding the model's behavior. Especially. when trying to understand not only individual predictions, but the influence of options or interactions. Unconstrained decision trees learned in our experiments often are 10 to 20 levels deep and have numerous internal decision nodes. This complexity will likely be difficult for most humans to understand the tree's behavior. As a contrast, we also report results on shallow decision trees (SDT), were we intentionally cap the maximum depth at level 6 (i.e., 63 decisions maximum).

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 & SDT	**1.6**	**2.8**	21.8	151.5
R200 & SDT	**0.6**	**1.1**	16.1	18.8
FW & SDT	**8.7**	103.5	123.1	406.3
PW & SDT	11.5	123.8	43.9	1779.1
R50 & DT	**2.4**	**3.0**	**6.8**	57.7
R200 & DT	**0.5**	**1.0**	16.0	18.2
FW & DT	11.5	125.1	143.1	1768.7
PW & DT	**1.0**	127.7	36.9	109.2
R50 & RF	**0.4**	**1.1**	16.4	59.9
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	46.9	27.3
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	**3.2**	**2.9**	**5.0**	**9.4**

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; **Bolded** values in cells indicate similarly low errors.

Table 2. Number of linear model terms with coefficients > 0.1 secs.

Approach	Apache Lucene	H2	Berkeley DB	Density Converter
R50 & SLR	41	25	37	45
R200 & SLR	26	13	28	30
FW & SLR	5	6	15	20
PW & SLR	23	20	21	83
Comprex	24	52	87	91

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

Table 3. 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
Comprex	19	16	32	72

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

Table 4. Number of linear model terms with coefficients > 0.7 secs.

Approach	Apache Lucene	H2	Berkeley DB	Density Converter
R50 & SLR	19	25	19	42
R200 & SLR	10	13	19	30
FW & SLR	3	3	15	8
PW & SLR	5	6	10	38
Comprex	10	12	17	68

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

Table 5. Number of linear model terms with coefficients > 1.0 secs.

Approach	Apache Lucene	H2	Berkeley DB	Density Converter
R50 & SLR	15	25	15	40
R200 & SLR	7	13	13	30
FW & SLR	3	3	15	7
PW & SLR	5	6	9	37
Comprex	2	8	17	58

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

4 DEBUGGING PARTITIONS

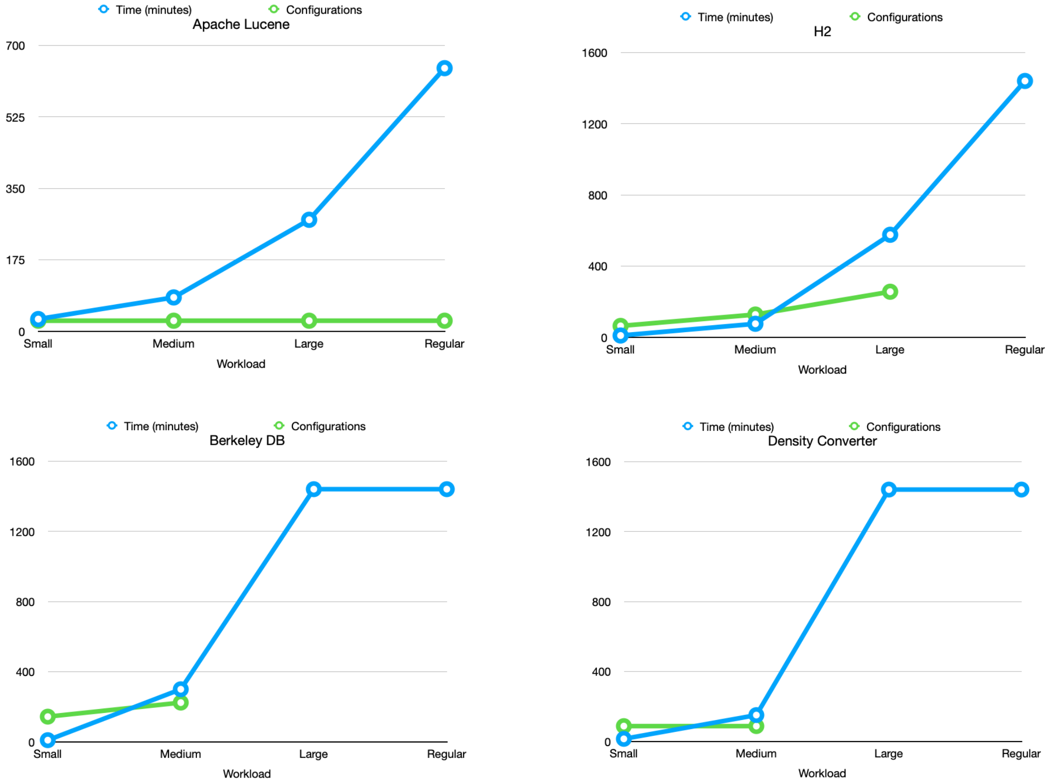

Fig. 1. Impact of workload size on the cost of the iterative dynamic analysis.