Supplementary material for: Transfer Learning for Optimal Configuration of Big Data Software

In this extra material, we briefly describe additional details about the experimental setting and complementary results that were not included in the main text.

1. FURTHER DETAILS ON SETTINGS

1.1 Code: TL4CO & BO4CO

https://github.com/dice-project/DICE-Configuration-BO4CO
https://github.com/dice-project/DICE-Configuration-TL4CO

1.2 Instructions for Storm Deployment

Since it is non trivial, we documented here how to deploy Storm on EC2, Azure and OpenNebula:
https://gist.github.com/pooyanjamshidi/e3570c275037043944fe

1.3 Configuration Parameters

The Apache Storm parameters (cf. Table 1):

- max_spout (topology.max.spout.pending). The maximum number of tuples that can be pending on a spout.
- spout_wait (topology.sleep.spout.wait.strategy.time.ms). Time in ms the SleepEmptyEmitStrategy should sleep.
- netty_min_wait (storm.messaging.netty.min_wait_ms). The min time netty waits to get the control back from OS.
- spouts, splitters, counters, bolts. Parallelism level.
- heap. The size of the worker heap.
- buffer_size (storm.messaging.netty.buffer_size). The size of the transfer queue.
- emit_freq (topology.tick.tuple.freq.secs). The frequency at which tick tuples are received.
- top_level. The length of a linear topology.
- message_size, chunk_size. The size of tuples and chunk of messages sent across PEs respectively.

The Apache Cassandra parameters (cf. Table 1):

- trickle_fsync: when enabled it forces OS to flush the write buffer at a regular time.
- auto_snapshot: enables the system to take snapshots, avoiding to lose data during truncation or table drop.
- concurrent_reads: number of threads dedicated to the read operations.
- concurrent_writes: number of threads dedicated to the write operations.
- file_cache_size_in_mb: the memory used as reading buffers.
- concurrent_compactors: number of threads dedicated to the compaction process of the data on the disk.

1.4 Datasets

Table 1: Experimental datasets, note that this is the complete set of datasets that we experimentally collected over the course of 3 month of 6 different cloud infrastructures. We only have shown part of this in the paper.

	Dataset	Parameters	Size	Testbed
1	wc(6D)	1-spouts: {1,3}, 2-max_spout: {1,2,10,100,1000,10000}, 3-spout_wait:{1,2,3,10,100}, 4-splitters:{1,2,3,6}, 5-counters:{1,3,6,12}, 6-netty_min_wait:{10,100,1000}	2880	C1
2	sol(6D)	1-spouts:{1,3}, 2-max_spout:{1,10,100,1000,10000}, 3-top_level:{2,3,4,5}, 4-netty_min_wait:{10,100,1000}, 5-message_size: {10,100,1e3,1e4,1e5,1e6}, 6-bolts: {1,2,3,6}	2866	C2
3	rs(6D)	1-spouts:{1,3}, 2-max_spout:{10,100,1000,10000}, 3-sorters:{1,2,3,6,9,12,15,18}, 4-emit_freq:{1,10,60,120,300}, 5-chunk_size:{1e5,1e6,2e6,1e7}, 6-message_size:{1e3,1e4,1e5}	3840	C3
4	wc(3D)	1-max_spout:{1,10,100,1e3, 1e4,1e5,1e6}, 2-splitters:{1,2,3,4,5,6}, 3-counters:{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18}	756	C4
5	wc+rs	1-max_spout:{1,10,100,1e3, 1e4,1e5,1e6}, 2-splitters:{1,2,3,6}, 3-counters:{1,3,6,9,12,15,18}	196	C4
6	wc+sol	1-max_spout:{1,10,100,1e3, 1e4,1e5,1e6}, 2-splitters:{1,2,3,6}, 3-counters:{1,3,6,9,12,15,18}	196	C4
7	wc+wc	1-max_spout:{1,10,100,1e3, 1e4,1e5,1e6}, 2-splitters:{1,2,3,6}, 3-counters:{1,3,6,9,12,15,18}	196	C4
8	wc(5D)	1-spouts:{1,2,3}, 2-splitters:{1,2,3,6}, 3-counters:{1,2,3,6,9,12}, 4-buffer-size:{256k,1m,5m,10m,100m}, 5-heap:{"-Xmx512m", "-Xmx1024m", "-Xmx2048m"}	1080	C5
9	wc-c1	1-spout_wait:{1,2,3,4,5,6,7,8,9,10,100,1e3,1e4}, 2-splitters:{1,2,3,4,5,6}, 3-counters:{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18}	1343	C1
10	wc-c3	1-spout_wait:{1,2,3,4,5,6,7,8,9,10,100,1e3,1e4,6e4}, 2-splitters:{1,2,3,4,5,6}, 3-counters:{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18}	1512	C3
11	cass-10	1-trickle_fsync:{false,true}, 2-auto_snapshot:{false,true}, 3-con._reads:{16,24,32,40}, 4-con._writes:{16,24,32,40}, 5-file_cache_size_in_mb:{256,384,512,640}, 6-con._compactors:{1,3,5,7}	1024	C6
12	cass-20			

1.5 Benchmark Settings

Table 2: Cluster specification

Cluster	Specification
C1	OpenNebula, 3 Sup, 1 ZK, 1 Nimbus, N: (1CPU, 4GB Mem)
C2	EC2, 3 Sup, 1 ZK, 1 Nimbus, N: m1.medium (1 CPU, 3.75GB)
C3	OpenNebula, 3 Sup: (3CPU,6GB Mem), 1 ZK: (1CPU,4GB Mem), 1 Nimbus: (2CPU,4GB Mem)
C4	EC2, 3 Sup, 1 ZK, 1 Nimbus, N: m3.large (2CPU, 7.5GB)
C5	Azure, 3 Sup: Standard_D1(1CPU, 3.5GB) , 1 ZK, 1 Nimbus, N: Standard_A1(1CPU, 1.75GB)
C6	OpenNebula, 1 N(4CPU, 4GB, 2 Disks)

Table 3: Cassandra benchmark specification.

Parameter	Value
Column family fields	10
Column family data size	1K
Read requests	50%
Update requests	50%
Read all fields	yes
Requests distribution	uniform
YCSB threads	2

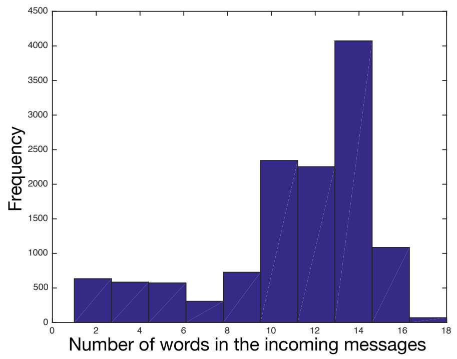


Figure 1: WordCount workload (arrival rate). Note that for RollingSort and SOL experiments the size of the incoming messages are set to 100. The message itself is randomely created by StringBuilder procedure which append 100 integer number between 1-9 randomly.

1.6 Apache Storm Architecture

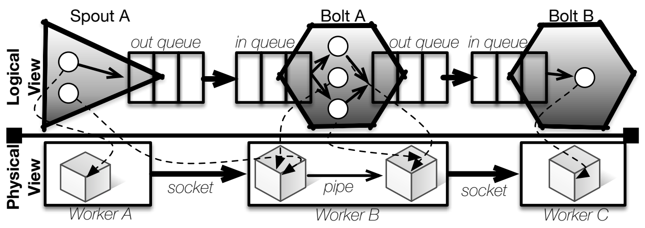


Figure 2: System model of a SPS with three worker nodes.

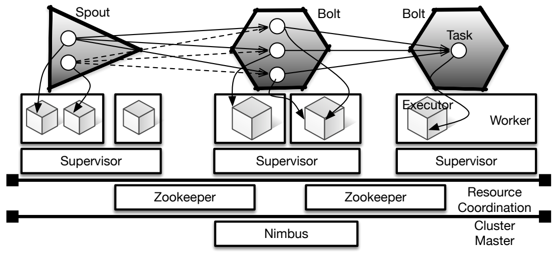
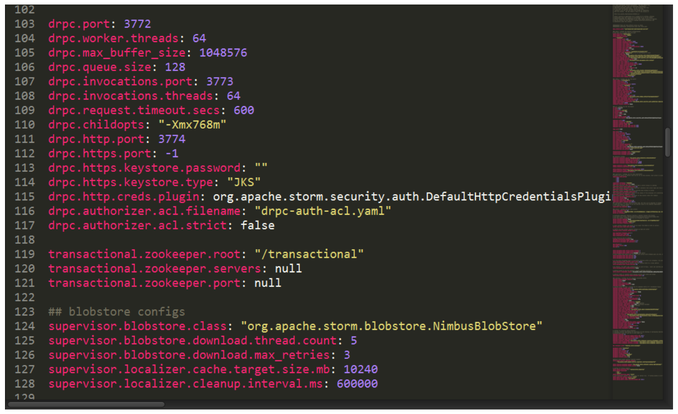


Figure 3: Apache Storm architecture.



```
drpc.port: 3772
drpc.worker.threads: 64
drpc.max_buffer_size: 1048576
drpc.queue.size: 128
drpc.invocations.port: 3773
drpc.invocations.threads: 64
drpc.request.timeout.secs: 600
drpc.childopts: "-Xmx768m"
drpc.http.port: 3774
drpc.https.port: -1
drpc.https.keystore.password: ""
drpc.https.keystore.type: "JKS"
drpc.http.creds.plugin: org.apache.storm.security.auth.DefaultHttpCredentialsPlugi
drpc.authorizer.acl.filename: "drpc-auth-acl.yaml"
drpc.authorizer.acl.strict: false

transactional.zookeeper.root: "/transactional"
transactional.zookeeper.servers: null
transactional.zookeeper.port: null

## blobstore configs
supervisor.blobstore.class: "org.apache.storm.blobstore.NimbusBlobStore"
supervisor.blobstore.download.thread.count: 5
supervisor.blobstore.download.max_retries: 3
supervisor.localizer.cache.target.size.mb: 10240
supervisor.localizer.cleanup.interval.ms: 600000
```

Figure 4: An example of a YAML file for Storm containing configuration parameters that we automatically change.

1.7 Additional Response Surfaces

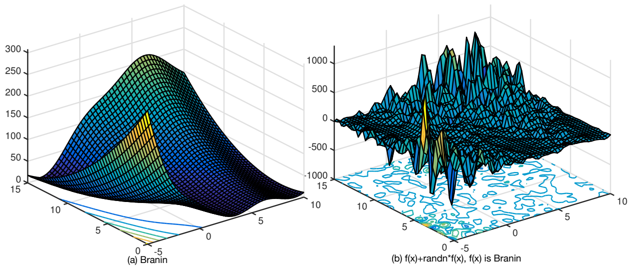


Figure 5: (a) Branin(2D), (b) $f(\boldsymbol{x}) + randn \times f(\boldsymbol{x})$.

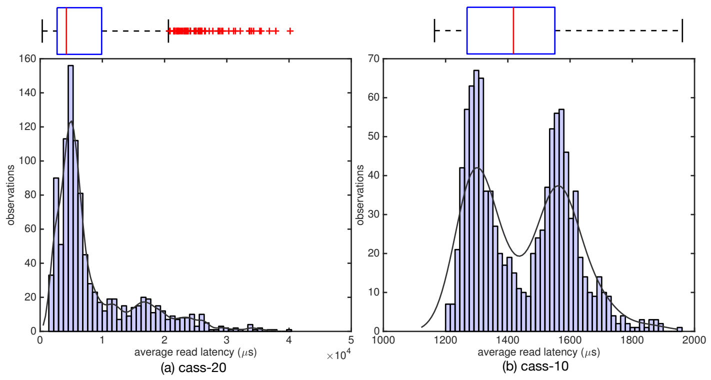


Figure 6: Average latency for (a) cass-20, (b) cass-10.

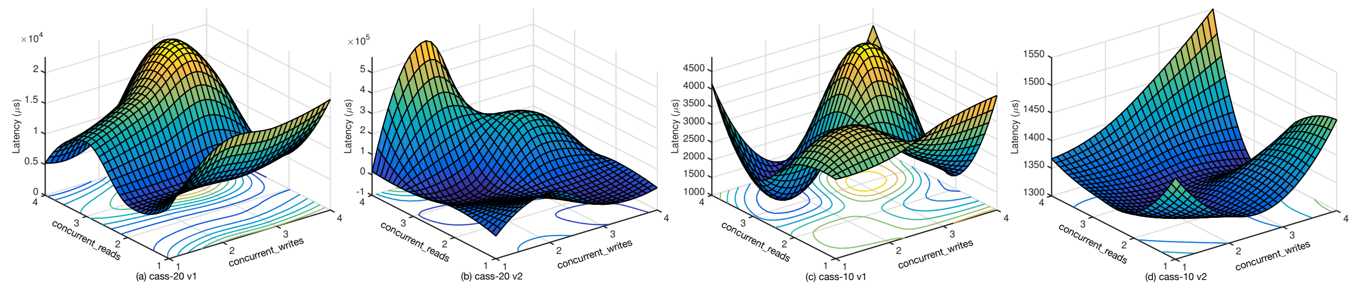


Figure 7: The response functions corresponding to the 2D subspace of two versions of cass-10,20.

2. RESEARCH QUESTIONS AND INSIGHTS

The extensive experimental data that we collected to evaluate TL4CO could be used to address several research questions. Due to the space limitation, we did not specified the question explicitly in the paper. Instead, here we define the questions and provide evidences based on the results reported in the paper. Here, we only focus on the research questions that we believe are most important for *software engineering* community.

***RQ1**: How large is the potential difference and as a result potential impact of a wrong choice of parameter settings?*

We demonstrated for both SPS in Figure 2 (*in the paper*) and for NoSQL systems in Figure 6 and also in Figure 7 that the performance difference between different configurations is significant. The error measures in the experimental results in the paper also demonstrate this large difference. The performance gain between the worst and best settings are measured for each datasets in Table 4.

Table 4: Performance gain between best and worst settings.

	Dataset	Best (ms)	Worst(ms)	Gain (%)
1	wc(6D)	55209	3.3172	99%
2	sol(6D)	40499	1.2000	100%
3	rs(6D)	34733	1.9000	99%
4	wc(3D)	94553	1.2994	100%
5	wc(5D)	405.5	47.387	88%
6	cass-10	1.955	1.206	38%
7	cass-20	40.011	2.045	95%

Different parameter settings cause very large variance in the performance of the system under test.

***RQ2**: How does a "default" setting compare to the worst and optimum configuration in terms of performance?*

Figure 16 reports the relative performance for the default setting, in terms of throughput and latency with the worst and best configuration as well as the one prescribed by NoSQL experts. As opposed our expectations, a default configuration is worst that majority of other configurations, it performs better than only 10% of other settings for read operations. However, for write operations the default configurations performs better than half of the other settings. This was expected because Apache Cassandra is designed to be write efficient. Even the configuration that is prescribed by practitioners and NoSQL experts is far from the optimum setting TL4CO found after only 100 iterations.

Default parameter settings perform poorly. The expert's prescription is also far from optimal on individual problem instances.

***RQ3**: How is the performance of a configuration optimization on a version when it has been tuned on another version?*

If one makes tuning on a specific version of the system under test, then we would expect it would be optimum for other versions of the system after it has gone under some changes. The results for both SPS and NoSQL in Figures 2 and 7 show that the optimum setting will be different for different versions. We also observed that parameter settings that should work well on average can be particularly inefficient on new versions compared to the best configuration for those versions. In other words, there is a very high variance in the performance of parameter settings. This has been also observed by another study in search-based software engineering community [1]. One interesting observations that has not reported previously is that the performance data have correlations among different versions.

Tuned parameters can improve upon default values on average, but they are far from optimal on individual problem instances.

***RQ4**: How much can transfer learning improve tuning with a larger number of observations from other versions?*

The larger the training set is, the more accurate the model predictions will likely be. In the context of configuration tuning, carrying over a larger observation set will result in better tuning, but it would be more expensive to carry out the optimization process as we have shown in Section 4.6. The results in Figure 18(b) and Figure 19(b) show that although the models become more accurate slightly, the gain in performance tuning does not pay off the costs.

Although transferring more observations improves performance and reduce the entropy, the improvement is low.

3. REFERENCES

[1] A. Arcuri and G. Fraser. Parameter tuning or default values? an empirical investigation in search-based software engineering. *Empirical Software Engineering*, 18(3):594–623, 2013.