Supplementary material for: Transfer Learning for Improving Model Predictions in Highly Configurable 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 Configuration Parameters

The Apache Storm parameters we have used in the experiments (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.

The CoBot parameters (cf. Table 1):

- num_particles: each particle representing a possible state, i.e., a hypothesis of where the robot is. This parameter specifies the number of particles the Monte Carlo localization algorithm initiate with.
- num_refinement: an integer variable such as the number of iterative refinements in the Monte Carlo localization algorithm.

Monte Carlo Localization (MCL) is an algorithm to localize a robot using a particle filter. The algorithm requires a known map and the task is to estimate the position and orientation of the robot within the map based on the motion and sensing of the robot. The algorithm starts with an initial belief of the robot pose's probability distribution, which is represented by particles distributed according to such belief. These particles are propagated following the robot's motion model each time the robot's pose changes. Upon receiving new sensor readings, each particle will evaluate its accuracy by checking how likely it would receive such sensor readings at its current pose. Next the algorithm will redistribute (resample) particles to bias particles that are more accurate. Keep iterating these moving, sensing and resampling steps, and all particles should converge to a single cluster near the true pose of robot if localization is successful.

1.2 Datasets

The details of the experimental dataset and the associated configuration parameters are presented in Table 1.

1.3 Benchmark Settings

The details regarding the cluster specification are in Table 2, regarding the Cassandra benchmark are in Table 3 and regarding the workload are in Figure 1.

1.4 Benchmark Systems Architecture

We have used three different systems with different architecture complexity (Figure 2, Figure 3) and inherent characteristics (Figure 4).

The performance gain between the worst and best settings are measured for each datasets in Table 4.

Table 1: Experimental datasets, note that this is the complete set of datasets that we experimentally collected over the course of 3 month of 9 different cloud clusters. 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 12	cass-10 cass-20	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
13	CoBot(4D)	1-odom_miscalibration:{0,0.05,...,0.45} 2-odom_noise:{0,0.05,...,0.45} 3-num_particles:{5,10,20,30,40,50,70,100, 125,150,200,250,300,350,400,450,500,750, 1000,1500,2000,3500,5000,7500,10000} 4-num_refinement::{1,3,5,8,10,15,20,35,50,70,100, 150,200,250,300,350,400,450,500,750, 1000,1500,2000,3500,5000,7500,10000}	56585	C9

Table 2: Testbed specification.

Cluster	Specification
C0	MacBook Pro, 2.5 GHz Intel Core i7, 16 GB 1600 MHz DDR3
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)
C6x	OpenNebula, 1 N(4CPU, 4GB, 2 Disks)
C6y	OpenNebula, 1 N(4CPU, 6GB, 2 Disks)
C7	Flexiant, 3 Sup, 1 ZK, 1 Nimbus, N: (1CPU, 2GB Mem)
C8	Flexiant, 2 Sup, 1 ZK, 1 Nimbus, N: (1CPU, 2GB Mem)
C9	bare metal, Intel(R) Core(TM) i5-4250U CPU @ 1.30GHz with SSD drive

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

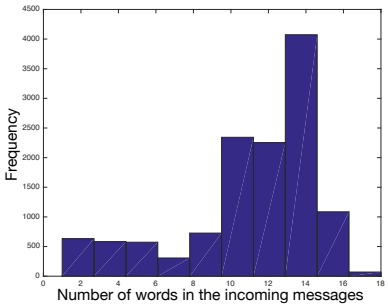

Figure 1: WordCount workload (arrival rate). For RollingSort and SOL experiments the size of the incoming messages are set to 100. The messages are created by StringBuilder which append 100 random numbers between 1-9.

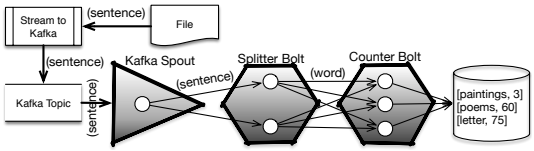

Figure 2: WordCount architecture.

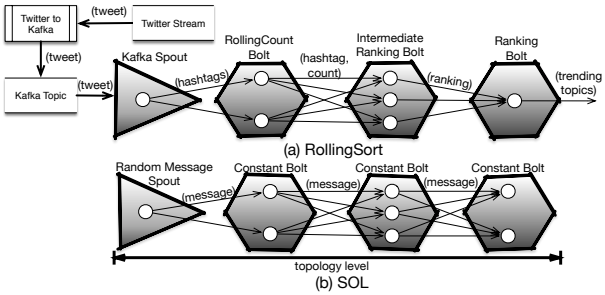

Figure 3: (a) RollingSort, (b) SOL architecture.

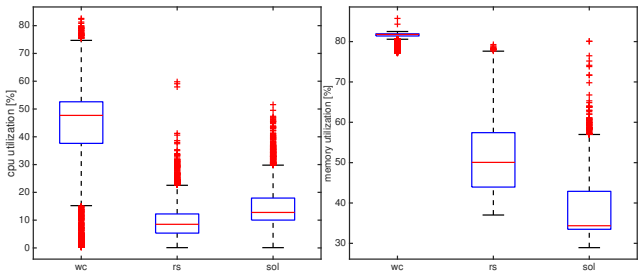

Figure 4: CPU and memory usage over 100 experiments.

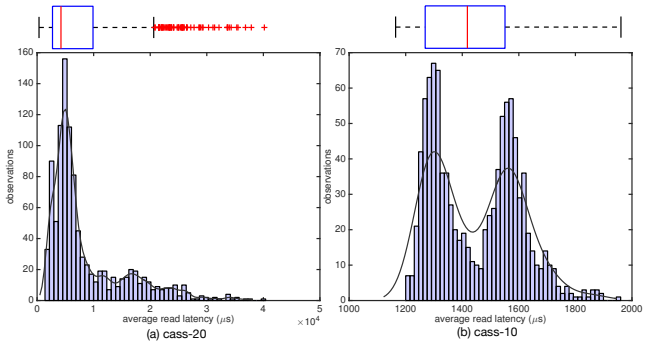

Figure 5: Average latency for (a) cass-20, (b) cass-10.

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%

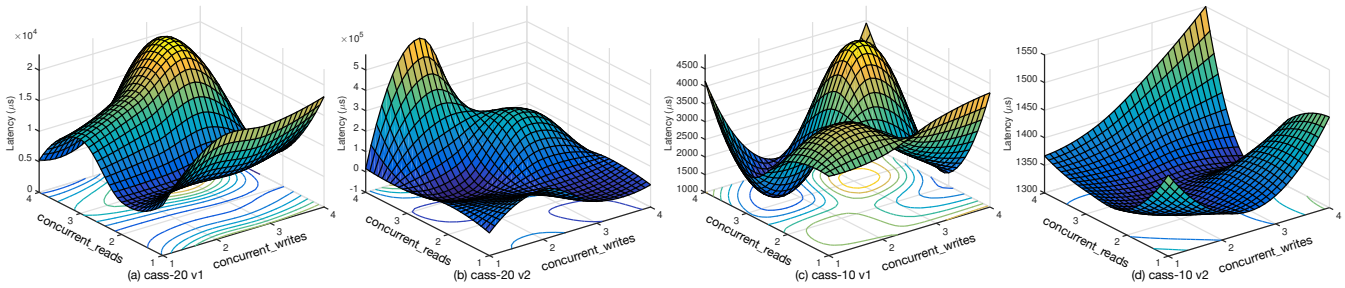

Figure 6: Response functions corresponding to the 2D subspace of cass-10,20 that are different in terms of *infrastructure*.

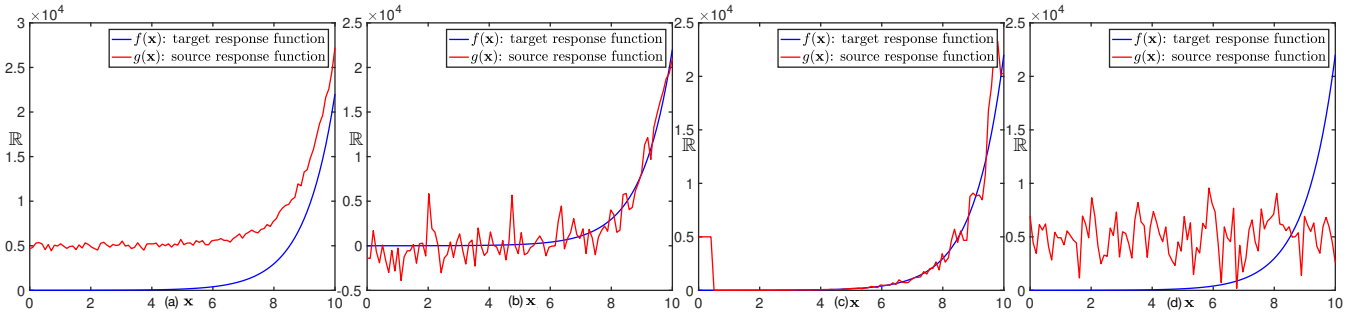

Figure 7: Relationships between source and target response functions: (a) source is highly correlated with the target function but shifted or linearly skewed, (b) contain noise, (c) have discrepancies towards the boundaries, or (d) uncorrelated.

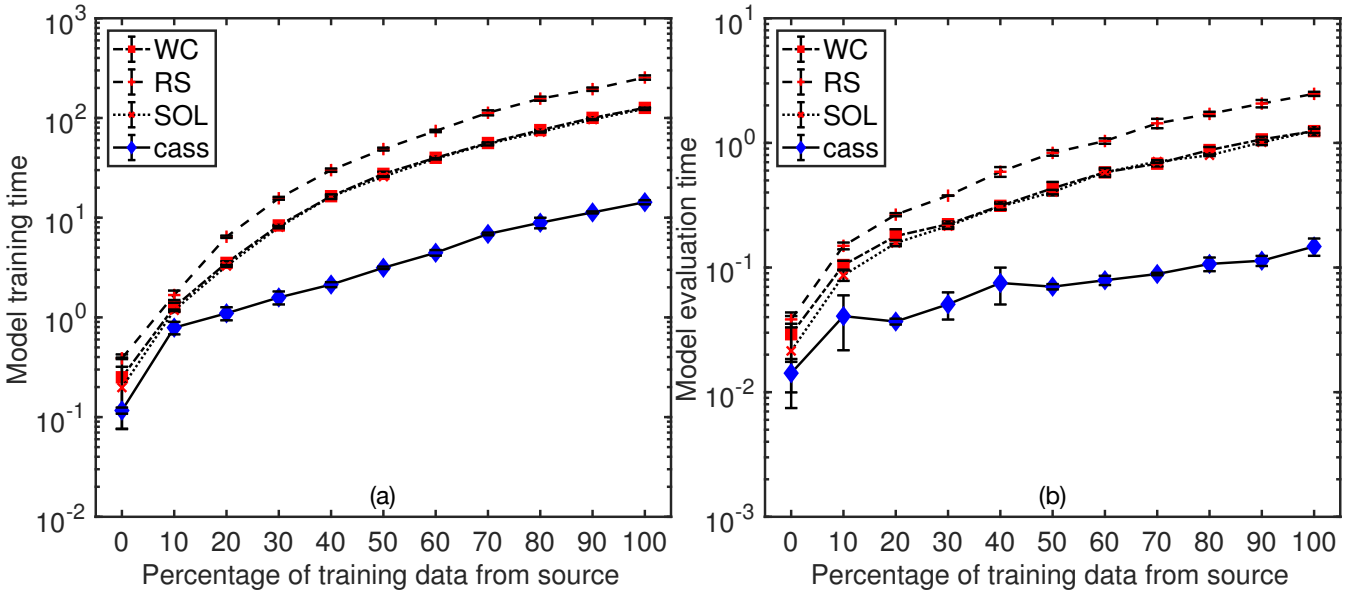

Figure 8: (a) training overhead and (b) evaluation overhead of our approach for training models using different datasets. All the model training and evaluation experiments were run on C0 (Table 2.)

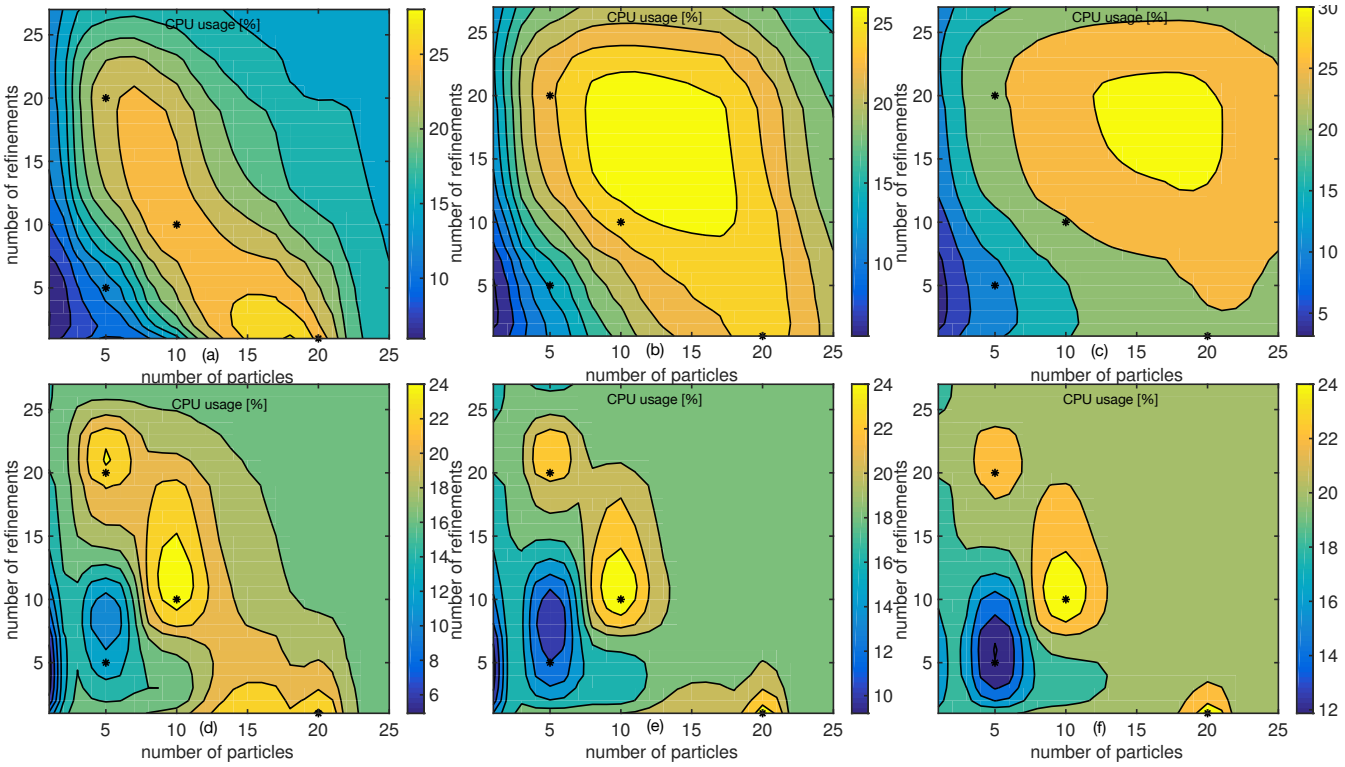

Figure 9: Prediction of the target space using source data taken from different response functions from (a) to (f) of decreasing correlation coefficients. As the source becomes less related to the target the prediction power of the model is decreased and the model is not able to get the structure of the target response correctly.

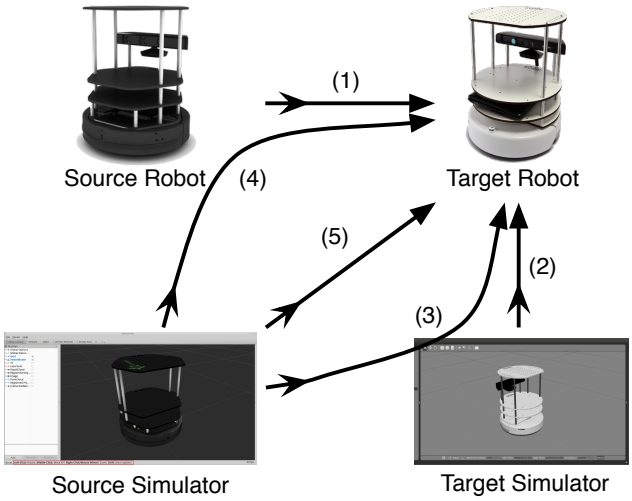

Figure 10: Possible transfer learning scenarios.