2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)

Performance Prediction of NUMA Placement: a Machine-Learning Approach

Fanourios Arapidis, Vasileios Karakostas, Nikela Papadopoulou, Konstantinos Nikas,
Georgios Goumas, Nectarios Koziris
Computing Systems Laboratory
School of Electrical and Computer Engineering
National Technical University of Athens, ICCS
{farap, vkarakos, nikela, knikas, goumas, nkoziris}@cslab.ece.ntua.gr

Abstract—**In this paper we present a machine-learning approach to predict the impact on performance of core and memory placement in non-uniform memory access (NUMA) systems. The impact on performance depends on the architecture and the application's characteristics. We focus our study on features that can be easily extracted with hardware performance counters that are found in commodity off-the-self systems. We run various single-threaded benchmarks from Spec2006 and Parsec under different placement scenarios, and we use this benchmarking data to train multiple regression models that could serve as performance predictors. Our experimental results show notable accuracy in predicting the impact on performance with relatively simple prediction models.**

Index Terms—**performance, modeling, NUMA, placement**

I. Introduction

NUMA systems are present in today's data centers as they provide the necessary infrastructure for scale-up workloads. The ubiquity of NUMA systems owes to their ability to provide large amounts of shared memory to applications over physically distributed memory nodes. However, the varying memory access latency and bandwidth that arise from the multiple memory modules renders the placement of application threads and memory critical to the application performance. Common wisdom mainly calls for memory locality, namely an application thread and its memory should reside on the same NUMA node, while also taking into consideration memory bandwidth and contention. Hence, many prior works have focused on characterizing and/or modeling the impact of NUMA on multi-threaded applications [9]–[11], [14], [15], focusing mainly on selecting the optimal placement of the application's multiple threads on the NUMA nodes.

Our work in this paper is triggered by a different problematic scenario that may arise in NUMA systems and that is exemplified in the ACTiCLOUD project [6], which seeks to improve utilization and resource efficiency in data centers. Instead of optimizing the placement of a single multi-threaded application that can use as many resources as possible, or the placement of many applications that run concurrently and hence need to have their cores and memory close to each other, we target the case in which it is impossible to supply all applications with (mostly) local memory and some need to run with remote memory. Hence, effective modeling is necessary to select which applications should run remotely.

For example, consider a 4-node NUMA system in which a single memory-hungry application, e.g., an in-memory database, occupies the cores of a single NUMA node and the memory of two NUMA nodes. That scenario leaves spare resources, i.e., the cores of the second NUMA node. However, there is no local memory available. In that case, when the resource manager decides which applications should be scheduled on the spare cores, it should be able to select applications that show little performance degradation when running with remote memory. Our experiments show that indeed not all applications experience the same performance degradation due to remote placement (Section II). Hence, to make correct decisions, the resource manager should have some prior knowledge on the impact of memory placement on performance, by making use of effective prediction models.

In this paper, we present a machine-learning approach to predict the impact on performance of core and memory placement in NUMA systems (Section III). We focus our study on features that can be easily extracted using common hardware performance counters, i.e., misses that occur in the last-level cache (LLC) and the TLB, and instructions per cycle (IPC). We use benchmarking data from a wide set of single-threaded benchmarks from Spec2006 and Parsec, to train multiple regression models that could serve as performance predictors. Our experimental results show notable accuracy in predicting the impact on performance, i.e., 5.31% mean absolute error and 0.93 coefficient of determination (R^2), with relatively simple, yet non-linear, prediction models (Section IV).

II. Motivation

In this section we provide background information about non-uniform memory access (NUMA) systems and then we quantify the impact of NUMA placement on various benchmarks, motivating the need for effective models that predict performance under different NUMA placement scenarios.

A. Background on NUMA systems

A NUMA system typically consists of many cores and large amount of main memory by assembling together multiple sockets and memory modules. This large computing system is unified in the sense that a single operating system is

978-1-5386-7899-2/18/$31.00 ©2018 IEEE
DOI 10.1109/CloudCom2018.2018.00064

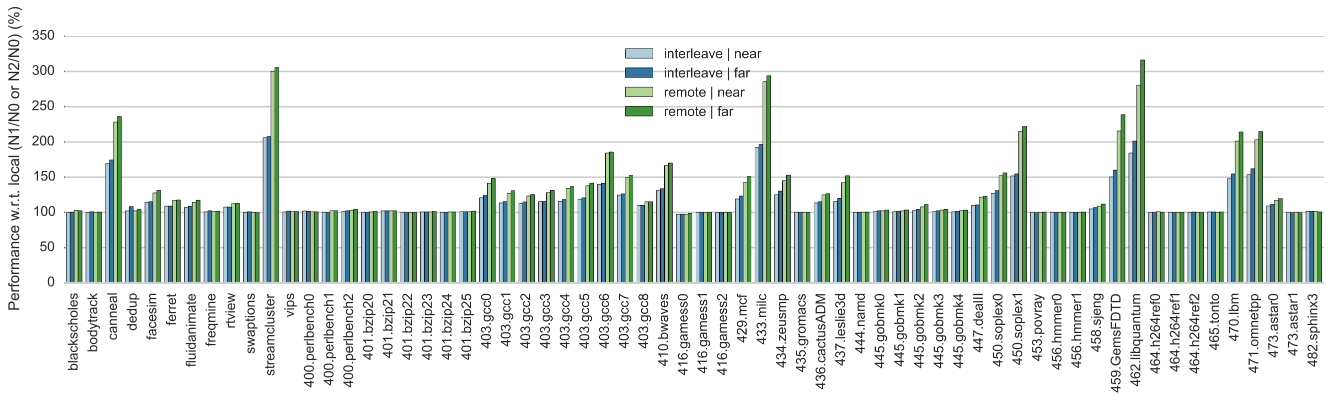

Fig. 1: Percentage of performance degradation for interleave and remote placement scenarios, normalized to the local placement.

responsible for managing the resources, while the programmer sees a large shared address space.

A NUMA system classifies cores and memory into NUMA nodes [8]. All memory available in one node has the same access characteristics (i.e., latency and bandwidth) for a particular core. Memory is called *local* or *remote* if it is allocated from the same or different NUMA node where the core is allocated, respectively. The process of assigning cores and memory from the various NUMA nodes that are available in the system is called *NUMA placement*.

B. Impact of NUMA placement

We want first to quantify the impact of NUMA placement on performance. We run benchmarks from Spec2006 [7] and Parsec [4] benchmark suites using the `numactl` [1] command utility under three NUMA placement scenarios:

- *local*: Both core and memory are allocated from the same NUMA node. This is the best scenario in terms of memory latency.
- *interleave*: Core is allocated from one NUMA node (e.g., N0), while memory is allocated from two nodes, the local and a remote one (e.g., N0 and N1), in interleave fashion. In this scenario, applications experience worse memory latency; however, the memory bandwidth doubles.
- *remote*: Core and memory are allocated from different NUMA nodes. This is the worst scenario in terms of memory latency and has the same memory bandwidth with the local scenario.

We use a 4-node NUMA system in which the latency differs when accessing a remote NUMA node. Section IV provides more details regarding the system. In our experimental platform, the latency for accessing N1's memory from N0's core is lower compared to when accessing N2's memory from N0's core. Hence, we define two sets of the above interleave and remote placement scenarios: in the first set the core is allocated from N0 and the memory from N1, while in the second scenario the core is again allocated from N0 but the memory is allocated from N2.

Figure 1 shows the impact of NUMA placement on performance for the various configurations. The bars marked as *near* refer to the case when the remote node is close (N1), and the bars marked as *far* refer to the case when the remote node is *far* (N2). For both near and far scenarios, we plot results for the interleave placement and the the remote placement. The results are normalized to the local placement.

We observe that applications can be classified into three categories depending on the impact of NUMA placement on their performance. The first set consists of applications that exhibit negligible performance degradation when executed under the interleave and placement scenarios, such as `perlbench`, `bzip2`, `hmmer`, and `freqmine`. The second set consists of applications that experience up to 50% performance degradation, such as `gcc`, `mcf`, `zeusmp`, and `facesim`. Finally, the third set consists of applications that experience more than 50% performance degradation and reaches up to 200%, such as `milc`, `soplex`, `GemsFDTD`, and `streamcluster`.

In summary, we observe that different applications experience different impact on performance due to NUMA placement. Hence, in this work we aim at identifying the characteristics that correlate with performance and use them in order to predict that performance degradation.

III. NUMA Performance Prediction

The goal of our work is to create performance models that associate the impact of NUMA placement on performance with a set of runtime metrics, transparently to the running applications. The target metric of the performance models is the performance degradation (or improvement) that an application experiences when part or all of its memory is allocated on a remote NUMA node, in comparison to its performance when all of its memory is resident on the local NUMA node. As we want our modeling approach to operate transparently to the running applications, we use IPC as the performance indicator of single-threaded applications. Thus, our target metric is the relative performance in terms of IPC, namely $IPC_{interleave}/IPC_{local}$ or IPC_{remote}/IPC_{local}.

A. Metrics as Features

We base our modeling approach on metrics that can be easily extracted at runtime using generic hardware performance

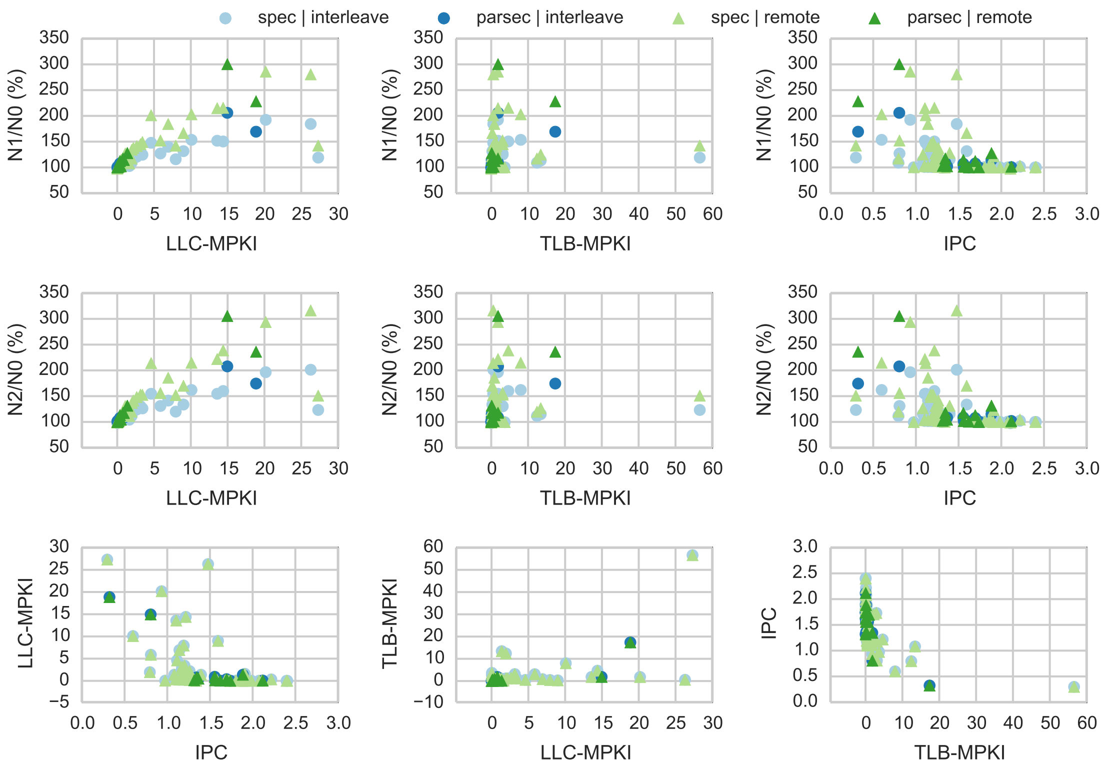

Fig. 2: Correlation results for the LLC-MPKI, TLB-MPKI, and IPC with respect to performance degradation (first two rows) and with respect to each other (last row). In the first row, the remote node is near (N1). In the second row, the remote node is far (N2).

counters and tools (e.g., `perf` [2] in Linux). All the listed metrics depend on both the benchmarks' characteristics and the platform's architecture. More specifically, the metrics that we consider as features are the following:

- *Last level Cache Misses per Kilo Instructions (LLC-MPKI)*: This metric counts the number of LLC misses that occur during the execution of thousand instructions. It indicates how often the benchmark needs to fetch data from memory, and hence provides a good indicator of whether a certain NUMA placement scenario would affect its performance. This metric is related to the cache organization of the platform.
- *TLB Misses per Kilo Instructions (TLB-MPKI)*: This metric counts the number of TLB misses that occur during the execution of thousand instructions. It indicates how often an application needs to fetch an address translation and provides also a good indicator of whether an application is memory intensive. This metric is related to the TLB organization of the platform.
- *Instructions per Cycle (IPC)*: This metric indicates the current performance of the application, counting the number of instructions that are committed in every cycle. This metric is related to the application's available instruction level parallelism, indicates whether the application is compute or memory intensive, and depends on the platform's hardware resources that exploit that parallelism.

B. Benchmarking

We use benchmarks from Spec2006 [7] and Parsec [4] benchmark suites to: (i) identify any correlation with the aforementioned metrics and the impact of NUMA placement on performance under various execution scenarios, and (ii) build prediction models that use those metrics as inputs and predict performance for various NUMA placement scenarios.

C. Model Building

The task of building performance models for predicting the impact of NUMA placement on performance falls into the category of supervised learning. The training set consists of (some of) the benchmarks, executed under the various placement scenarios that were described in Section II.

The features play an important role in modeling. Because not all features affect to the same extent the performance degradation of NUMA placement, we need to understand their correlation. Figure 2 plots the correlation of the various metrics with performance degradation. The columns correlate performance degradation with LLC-MPKI, TLB-MPKI, and IPC, respectively. The first row of results considers the case when the remote node is near (N1), and the second row considers the case when the remote node is far (N2). The results are normalized to the local placement scenario.

We observe a strong correlation and a monotonic relation with the LLC-MPKI metric, and a weaker correlation with the TLB-MPKI metric. Correlation with IPC is less clear.

To understand whether some of the features are redundant, we also plot the correlation of pairs of the three metrics, LLC-MPKI, TLB-MPKI and IPC, as shown in the third row of Figure 2. We observe that there is no clear correlation between the individual features. Hence, we conclude that an ideal model should include all three features. Since there are no evident linear relationships between the three features and the target, we opt for a non-linear model.

To avoid the manual effort of selecting the model hypothesis, given the complexity of searching the space of non-linear combinations of the three metrics, we automatically generate models following a model generation approach similar to the one presented in [5]. To generate multiple model hypotheses, we perform non-linear transformations of the three metrics and also combine them in interaction terms, namely products of two or more metrics. The generated models may include one or more of the following transformations of the three metrics as additive terms, along with coefficients:

$\{a_{i0} \times x, a_{i0} \times log(x_i), a_{i0} \times log^{n_{i0}}(x_i), a_{i0} \times {x_i}^{n_{i0}}, a_{i0} \times x_i + a_{i1} \times log(x_i), a_{i0} \times x_i + a_{i1} \times log^{n_{i0}}(x_i), a_{i0} \times x^{n_{i0}} + a_{i1} \times log(x_i), a_{i0} \times {x_i}^{n_{i0}} + a_{i1} \times log^{n_{i1}}(x_i)\}, x_i \in \{LLC\_MPKI, TLB\_MPKI, IPC\}$.

Our model hypotheses may also include interactions of the three metrics, and in particular, one of the following two terms:

$\{a_{ij} \times {x_i}^{m_i} \times {x_j}^{m_j}, a_{ijk} \times {x_i}^{m_i} \times {x_j}^{m_j} \times {x_k}^{m_k}\}, x_i, x_j, x_k \in \{LLC\_MPKI, TLB\_MPKI, IPC\}$.

The coefficients a_{i0}, a_{i1}, a_{ij}, a_{ijk} and the exponents $n_{i0}, n_{i1}, m_i, m_j, m_k$, where they exist, are computed during the model estimation. Finally, our models include an intercept term b, also computed during model estimation.

We have created a python tool for the automatic model generation, generating 1360 model hypotheses. We use Python's `curve_fit` function from the `scikit` package to fit all the generated models to our training set. We then select the model that best fits our data using the mean absolute error (MAE) and the coefficient of determination R^2.

D. Discussion

The approach we propose opts for simple, yet accurate, models. Given the limited number of NUMA nodes in our experimental setup, we have built different models for the various placement scenarios, using only the three features related to the application characteristics and have trained four models separately for each one of the four different NUMA placement scenarios. However, for a system with more nodes and asymmetry in memory latency and bandwidth, building different models for each possible placement is an efficient solution. A more suitable approach would be to build a single model for all possible placements, using additional, explanatory features to reflect the memory placement and the nodes in use. This approach could also yield a model with improved accuracy due to the larger, merged training set.

Benchmarking is currently the most time-consuming part in our modeling approach, as it is based on collecting measurements for applications in standard benchmark suites. An alternative approach is to construct a micro-benchmark that will be able to sweep the parameter space and provide a large space of possible values for the features in use and performance under different NUMA placement scenarios. This would significantly reduce the time to collect measurements for the training set, and would also make our modeling approach more generic and easily applicable to larger systems.

Model generation and training, for our current setup, is not lengthy in time, taking only a few hours, and only needs to be applied once per system. However, if the number of features increases, the number of generated models can become orders of magnitude higher, making the training time prohibitively long. A solution is to prune the space of generated models, by constraining the powers of the exponents to specific values, as in [5]. That would also decrease the per model training time, as it decreases the search space for the curve fitting algorithm.

IV. Evaluation

In this section, we evaluate the accuracy of our approach for predicting the impact of NUMA placement on performance. We report the training methodology for our model and demonstrate prediction results for the relative performance of applications for the target NUMA placement.

TABLE I: Platform details.

Model	Intel Xeon E5-4620
Micro-architecture	Sandy Bridge
Core frequency	2.20GHz
L1 cache (I/D)	32KB/32KB
L2 cache	256KB per core
L3 cache	16MB per core
Cores per socket	8
Sockets	4
NUMA nodes	4
Memory	256GB

A. Methodology

All the reported experiments are performed on an Intel Xeon E5-4620 platform that consists of 4 NUMA nodes. Table I includes details about the platform.

To train and evaluate the accuracy of the generated prediction models, we perform cross-validation. In particular, we implement random subsampling validation, repeated 5 times. Each time, we form a randomly-generated test set, consisting of 20 benchmarks, out of 64 in total. We use the rest of the benchmarks that do not belong in the test set as the training set. We train the model and then measure its accuracy by using the benchmarks in the testing set and computing the mean absolute error for all predicted values, as well as the coefficient of determination R^2. We repeat the same process for all five test sets, and compute and report the average of the aforementioned metrics, across the five repetitions.

B. Results

Table II presents the 10 best models that were generated by our tool and that were evaluated for their accuracy using our methodology. The models are sorted based on the mean absolute error and the coefficient of determination R^2. We observe that all 10 best models include all features, but each

TABLE II: The 10 best generated performance models. x_1, x_2, and x_3 correspond to LLC-MPKI, IPC, and TLB-MKPI, respectively.

No.	***Model***	R^2	***Avg. Error (%)***
1	$a_1 * x_1^{n_1} + a_2 * log(x_2+1)^{n_2} + a_3 * x_3 + b$	0.9310	5.3153
2	$a_1 * x_1^{n_1} + a_2 * x_2^{n_2} + a_3 * x_3 + b$	0.9303	5.3562
3	$a_1 * log(x_1+1) + a_2 * x_1 + a_3 * x_2^{n_1} + a_4 * x_3 + b$	0.9291	5.4004
4	$a_1 * x_1 + a_2 * log(x_2+1) + a_3 * x_3 + a_4 * (x_1^{n_1}) * (x_2^{n_2}) + b$	0.9283	5.3818
5	$a_1 * x_1 + a_2 * x_2 + a_3 * x_3 + a_4 * (x_1^{n_1}) * (x_2^{n_2}) + b$	0.9278	5.4169
6	$a_1 * log(x_1+1)^{n_1} + a_2 * x_1 + a_3 * x_2^{n_2} + a_4 * x_3 + b$	0.9274	5.4801
7	$a_1 * log(x_1+1) + a_2 * x_1 + a_3 * x_2 + a_4 * x_3 + b$	0.9253	5.6393
8	$a_1 * x_1^{n_1} + a2 * log(x_2+1)^{n_2} + a_3 * x_2 + a_4 * x_3 + b$	0.9253	5.3471
9	$a_1 * log(x_1+1) + a_2 * x_1^{n_1} + a_3 * x_2 + a_4 * x_3 + b$	0.9245	5.6528
10	$a_1 * x_1^{n_1} + a_2 * log(x_2+1) + a_3 * x_3 + b$	0.9235	5.8177

model includes different terms. Additionally, some of the models include interaction terms between the three variables. We also observe that all achieve very similar results, with low average error and high R^2. This validates our approach to auto-generate multiple model hypotheses, to cope with the complexity of searching the vast space of non-linear models. Moreover, it makes our methodology applicable to different architectures without requiring extensive tuning. The low mean absolute error is indicative of the accuracy of our model, while the high R^2 score (over >0.92 for the 10 best models) shows that the trained models fit well to our data.

Finally, Figure 3 shows the actual and predicted results for the most accurate performance model. Each row contains results for the five test sets that were used. The top two rows consider the case when the remote node is near (N1) with *interleave* and *remote* placement policies, and the bottom two rows consider the case when the remote node is far (N2) with *interleave* and *remote* placement policies. We observe that the performance model provides accurate estimations for the various test sets with a mean absolute error of 5.31% across the predicted points. The plots also allow us to observe that we do not encounter any high prediction errors in cases where performance degradation/improvement is lower than 5%, i.e., the mean absolute error (e.g. `perlbench, bzip, bodytrack, swaptions, deall` and others). Finally, there are no severe or misleading mispredictions that could wrongly indicate, for example, performance improvement instead of performance degradation. This observation validates the quality and usefulness of our modeling approach.

V. Related Work

The complexity of NUMA systems demands efficient placement of application threads and memory onto the system to harness the system performance. As a result, significant research has been done towards modeling performance of NUMA systems or applications running on NUMA systems. Majo et al. [10] have developed a model to characterize the sharing of local and remote memory bandwidth in NUMA systems. McCormick et al. [11] have proposed a memory-access model to improve task-scheduling decisions by scheduling tasks near the data they need. However, their model depends only on the platform's characteristics and do not take into consideration the application's characteristics. Wang et al. [15] have proposed a model that predicts both memory bandwidth usage and optimal core allocations for multi-threaded applications on NUMA systems. Finally, Luo et al. [9] proposed a compositional model for selecting the best core and memory placement scenario in NUMA systems. However, their model targets multi-threaded applications and requires profiling information that could affect the performance of the applications.

Machine-learning has seen many applications as an empirical method to model systems/applications performance in the past decade. Indicatively, Barnes et al. [3] have used multivariate regression to extrapolate the performance of parallel applications on higher core counts and predict their scalability. Shudler et al. [13] have used single-variable regression for scalability prediction, auto-generating non-linear models. Calotoiu et al. [5] use the same technique for automatic model generation to model the performance of complex parallel applications. Papadopoulou et al. [12] have used multivariate regression to predict communication time of MPI applications. Regarding NUMA systems, Su et al. [14] have used machine-learning techniques to predict the walltime of multi-threaded applications on NUMA systems, for different thread placements. Their approach is orthogonal to ours, predicting performance in relation to thread instead of memory placement.

VI. Summary and Future work

In this paper we described a machine-learning approach to predict the impact on performance of core and memory placement in NUMA systems. While our methodology and models are the first step towards effective modeling that could drive the decision-making logic of a resource manager, there are still many directions for improving our work. First, we would like to apply our methodology to multi-threaded applications and to applications with larger working sets that stress more the memory system. We would also like to apply our methodology to other platforms that have more nodes and expose asymmetry in memory access. Furthermore, we would like to explore correlation with more performance counters (e.g., memory bandwidth). Finally, we would like to test our methodology with micro-benchmarks to decouple the creation of models from the execution of particular real-world applications.

Acknowledgment

This research has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement no. 732366 (ACTiCLOUD).

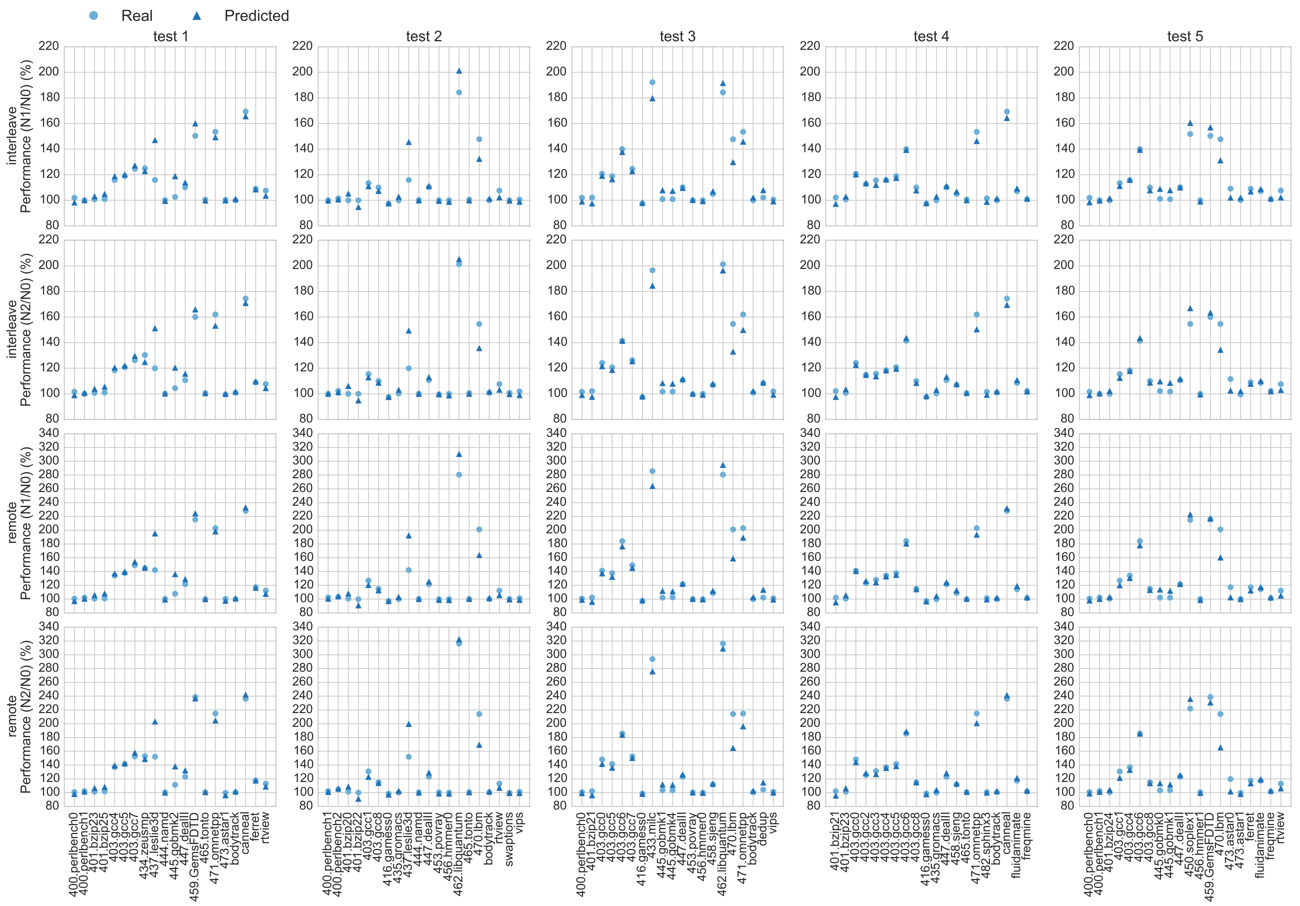

Fig. 3: Actual and predicted results for the most accurate performance model. Each row contains results for the five test sets that were used.

References

[1] "numactl - Control NUMA policy for processes or shared memory." https://linux.die.net/man/8/numactl, [Online].

[2] "perf: Linux profiling with performance counters." https://perf.wiki.kernel.org/index.php/Main_Page, [Online].

[3] B. J. Barnes, B. Rountree, D. K. Lowenthal, J. Reeves, B. De Supinski, and M. Schulz, "A regression-based approach to scalability prediction," in *Proceedings of the 22nd annual International Conference on Supercomputing*, 2008, pp. 368–377.

[4] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, January 2011.

[5] A. Calotoiu, D. Beckinsale, C. W. Earl, T. Hoefler, I. Karlin, M. Schulz, and F. Wolf, "Fast multi-parameter performance modeling," in *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, 2016, pp. 172–181.

[6] G. I. Goumas, K. Nikas, E. B. Lakew, C. Kotselidis, A. Attwood, E. Elmroth, M. Flouris, N. Foutris, J. Goodacre, D. Grohmann, V. Karakostas, P. Koutsourakis, M. L. Kersten, M. Luján, E. Rustad, J. Thomson, L. Tomás, A. Vesterkjaer, J. Webber, Y. Zhang, and N. Koziris, "ACTiCLOUD: Enabling the Next Generation of Cloud Applications," in *37th IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 1836–1845.

[7] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006.

[8] C. Lameter, "NUMA (Non-Uniform Memory Access): An Overview," *Queue*, vol. 11, no. 7, pp. 40:40–40:51, Jul. 2013.

[9] H. Luo, J. Brock, P. Li, C. Ding, and C. Ye, "Compositional model of coherence and NUMA effects for optimizing thread and data placement," in *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2016, pp. 151–152.

[10] Z. Majo and T. R. Gross, "Memory System Performance in a NUMA Multicore Multiprocessor," in *Proceedings of the 4th Annual International Conference on Systems and Storage*, ser. SYSTOR '11. ACM, 2011, pp. 12:1–12:10.

[11] P. S. McCormick, R. K. Braithwaite, and W.-c. Feng, "Empirical Memory-Access Cost Models in Multicore NUMA Architectures," Los Alamos National Lab. (LANL), Los Alamos, NM (United States), , 2011.

[12] N. Papadopoulou, G. I. Goumas, and N. Koziris, "A Machine-Learning Approach for Communication Prediction of Large-Scale Applications," in *2015 IEEE International Conference on Cluster Computing (CLUSTER)*, 2015, pp. 120–123.

[13] S. Shudler, A. Calotoiu, T. Hoefler, A. Strube, and F. Wolf, "Exascaling your library: Will your implementation meet your expectations?" in *Proceedings of the 29th ACM on International Conference on Supercomputing*, 2015, pp. 165–175.

[14] C. Su, D. Li, D. S. Nikolopoulos, K. W. Cameron, B. R. de Supinski, and E. A. León, "Model-based, memory-centric performance and power optimization on NUMA multiprocessors," in *2012 IEEE International Symposium on Workload Characterization (IISWC)*, 2012, pp. 164–173.

[15] W. Wang, J. W. Davidson, and M. L. Soffa, "Predicting the memory bandwidth and optimal core allocations for multi-threaded applications on large-scale NUMA machines," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016, pp. 419–431.