Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence

Transfer Learning-Based Co-Run Scheduling for Heterogeneous Datacenters

Wei Kuang, Laura E. Brown and **Zhenlin Wang**
Department of Computer Science, Michigan Technological University, Houghton, MI 49931
{wkuang,lebrown,zlwang}@mtu.edu

Abstract

Today's data centers are designed with multi-core CPUs where multiple virtual machines (VMs) can be co-located into one physical machine or distribute multiple computing tasks onto one physical machine. The result is co-tenancy, resource sharing and competition. Modeling and predicting such co-run interference becomes crucial for job scheduling and Quality of Service assurance. Co-locating interference can be characterized into two components, *sensitivity* and *pressure*, where sensitivity characterizes how an application's own performance is affected by a co-run application, and pressure characterizes how much contentiousness an application exerts/brings onto the memory subsystem. Previous studies show that with simple models, sensitivity and pressure can be accurately characterized for a single machine. We extend the models to consider cross-architecture sensitivity (across different machines).

Introduction

Each year more computation moves to data centers deploying thousands of heterogeneous, modern machines to provide customers with computing and storage services. Most modern machines have multiple cores on a single chip; thus, multiple tasks/applications can be executed simultaneously so that resource utilization can be optimized. Though each CPU core can have a dedicated task, co-locating does require resource sharing, such as last level cache and memory bandwidth. The diversity of application behavior will have some programs be more aggressive towards the shared resources resulting in interference. Co-locating interference can be characterized into two components, *sensitivity* and *pressure*, where sensitivity characterizes how an application's own performance is affected by a co-run application, and pressure characterizes how much contentiousness an application exerts/brings to the system. Currently, the focus is on the shared memory subsystem, but other shared resources can be considered such as memory bandwidth and network I/O. Without modeling and predicting this interference, the data center scheduler may have to dedicate a whole multi-core machine to a single time sensitive application, leaving some cores idle, in order to guarantee Quality of Service.

Previous studies have shown that an application's sensitivity and pressure can be obtained by profiling; an application is co-run with a carefully designed probe program to generate both metrics (Mars et al. 2011; Yang et al. 2013). The performance degradation due to co-run interference can be predicted with relative low error. However, this method has two limitations: the sensitivity curve is interpolated from the profiling results rather than fit to a model and the prediction from the sensitivity and pressure is only for the machine on which the profiling was run. Collaborative filtering was used to predict interference of programs on machines with different hardware configurations, yet the latent model needs to be further explored (Delimitrou and Kozyrakis 2013).

Our motivation is to develop a method to accurately characterize the sensitivity curves for different applications via a general, mathematical model, and also ensure that the models continue be effective from architecture to architecture. The process will be initiated with initial offline profiling and training of the model, yet it must be able to evolve or adapt for new architectures becoming available; employing ideas from transfer learning (Pan and Yang 2010).

Preliminary Results

The sensitivity curve of an application on a specific machine is found by using the "Bubble-up" methodology (Mars et al. 2011). A program called bubble is co-run with a target application on the machine. The bubble is a simple program that can exert different levels of pressure onto the memory subsystem. By careful design, the bubble can take up increasing levels (MBs) of cache space, thus making the co-run application slow down (degraded performance). The target application's performance, often referred to as normalized execution time, is recorded at different level of bubble pressure. From a collection of bubble size versus performance degradation value pairs, a model can be fit (e.g., polynomial or logistic function). In a pending submission, we collect the sensitivity curves of several sets of benchmarks including SPEC CPU2006 integer and floating point programs and a subset of Parsec 3.0 and CloudSuite 2.0 programs. We found that a logistic function achieves the lowest generalizable error in modeling such sensitivity curves.

The profiling process of each individual program was repeated on machines with different hardware. Hence, each application has multiple sensitivity curves corresponding to

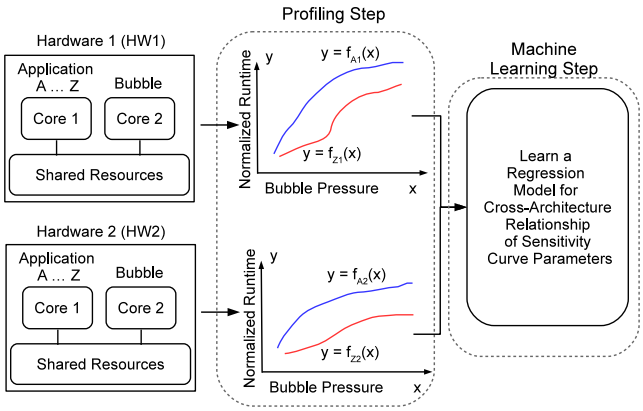

Figure 1: Framework for cross-architecture prediction

different hardware configurations. The program pressure was also profiled across machines. By observing the collected data, we found that both sensitivity curve and pressure of a program will behave differently on different machines. Therefore, we cannot predict performance interference for a specific machine from the model generated by another machine with dissimilar hardware.

Nevertheless, we found that sensitivity curve and pressure of applications can be estimated on a new machine without extensive profiling and training. A relationship between the parameters of the sensitivity curves was estimated. Therefore, the framework of our co-run performance prediction system is as follows (see Fig. 1). With a base hardware configuration, and an abundance of benchmark programs representing different types of applications, we could do exhaustive off-line profiling and training. Whenever a new hardware configuration is added, we only do profiling for a small subset of benchmark programs and find the cross-architecture relations between parameters. Thus, any application's sensitivity curve can be recovered through the parameters of function obtained from the baseline machine. The same design is followed to find relations of pressure.

We test this approach with SPEC CPU2006 benchmarks as the training set, and Parsec 3.0 and CloudSuite 2.0 as the test set with a Core2 Duo as the base hardware and Intel i5 as the new machine. The predicted co-run performance degradation using the cross-architecture functions for sensitivity curves and pressure has a mean relative error of $<2\%$.

Current Focus

The preliminary work piloted the idea of co-run sensitivity learning across architectures, yet generalizations and extensions to this idea need exploration. For example, the cross-architecture relationship between sensitivity curves can be learned for specific machines, but is not a general model that is based on hardware configuration information, e.g., last level cache size, L2 cache, etc. Additionally, an application's pressure will change between different machines. A uniform model (polynomial, logistic) may not closely characterize these variations. What other features may be incorporated into the models to explain these variations?

Our group has done significant preliminary work on locality modeling, cache partitioning, memory management, migration and remote memory in the system side. We also have extensive experience in machine learning. This thesis work examines the use and design of machine learning methods to the systems domain to predict memory resource demand and cache pressure. In particular, for the cross-architecture prediction problems, transfer learning is examined.

Two initial frameworks for the problem are to be explored. For the first, detailed source domain(s) data will be available, that is extensive profiling will be collected on older hardware configurations. The focus is to improve prediction accuracy and require limited profiling of applications on any new target architectures added to the data center. For this framework, initial inductive transfer learning methods that employ a parameter-transfer approach will be considered. The second framework will consider an active, or lifelong learning design, so that new applications and/or architectures will be continually added to the system (Eaton and Ruvolo 2013).

Both frameworks will make use of an extensive library of profiled benchmarks on different machine architectures. The estimated generalization performance of the models will be confirmed with tests on a realistic environment in a data center. The models will be incorporated into the Xen virtual machine hypervisors and a comprehensive set of benchmarks will be co-run to validate the results and refine our models.

Conclusion

We describe a method to help a data center assign tasks with high utilization while maintain QoS requirements. Based on our experimental results, an application's sensitivity curve can be characterized by a logistic function. Furthermore, the parameters of the sensitivity functions for different machines are related. Transfer learning will be explored to help shorten both profiling and training time and online model updating examined as there will always be new applications and architectures arriving in a data center.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 1422342 and 0643664.

References

Delimitrou, C., and Kozyrakis, C. 2013. Paragon: Qos-aware scheduling for heterogeneous datacenters. In *ASPLOS '13*, 77–88.

Eaton, E., and Ruvolo, P. L. 2013. Ella: An efficient lifelong learning algorithm. In *ICML-13*, volume 28, 507–515.

Mars, J.; Tang, L.; Hundt, R.; Skadron, K.; and Souffa, M. L. 2011. Bubble-up increasing utilization in modern warehouse scale computers via sensible co-locations. In *MICRO-44*.

Pan, S. J., and Yang, Q. 2010. A survey on transfer learning. *IEEE TKDE* 22(10):1345–1359.

Yang, H.; Breslow, A.; Mars, J.; and Tang, L. 2013. Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers. In *ISCA '13*, 607–618.