

Appendix: Collection of Base Models

Jin Chen, Gokul Soundararajan[†], Cristiana Amza[†]

Department of Electrical and Computer Engineering[†]
Department of Computer Science
University of Toronto

1 Introduction

Our *hybrid* performance model leverages the predictions made by several base performance models. We classify the models into three major classes: (i) analytical models – that mathematically model the performance of the system, gray-box models – that use domain-specific knowledge and curve-fitting, and black-box models – that use statistical regression to predict performance.

2 Analytical Performance Models

Analytical models are mathematical models built after a thorough analysis of the underlying system. The models work as follows: they use information about the workload and the underlying system to predict the performance for different configurations. For example, an analytical model of the cache requires a trace of cache accesses, the cache replacement policy, and the cost of cache hit and miss to predict the access for any cache size. In the following, we describe two such analytical models that we develop to predict the performance of a multi-tier storage system consisting of two-levels of caches (i.e., the DBMS buffer pool and the storage cache) and a disk scheduler.

2.1 Modeling Storage System Performance

The performance of a multi-tier storage system, e.g., *Akash* [3], is affected by the two-levels of caches (i.e., the DBMS buffer pool and the storage cache), and the disk I/O scheduler. In the following, we explain the derivation of the analytical model to predict the performance of this system.

Multi-level caching model: In a multi-level cache hierarchy using the standard (uncoordinated) LRU replacement policy at all levels, any cache miss from cache level i will result in bringing the needed block into all lower

levels of the cache hierarchy, before providing the requested block to cache i . It follows that the block is *redundantly* cached at all cache levels, which is called the *inclusiveness* property [4]. Therefore, if an application is given a certain cache quota ρ_i at a level of cache i , any cache quotas ρ_j given at any lower level of cache j , with $\rho_j < \rho_i$ will be mostly wasteful. Based on these observations, we make the following simplifications to approximate the overall miss-ratio of a two-level cache, i.e., $\widehat{\mathcal{M}}(\rho_c, \rho_s)$, based on a single-level cache model.

In an uncoordinated LRU cache hierarchy, only the maximum size quota given at any level of cache matters; therefore, we approximate the miss-ratio of a two level cache, consisting of a buffer pool (with quota ρ_c) and a storage cache (with quota ρ_s) by the following formula

$$\widehat{\mathcal{M}}(\rho_c, \rho_s) \approx \mathcal{M}_c(\max[\rho_c, \rho_s]) \quad (1)$$

We can further approximate the fraction of accesses that miss in both levels of cache, hence reach the disk, i.e., $\mathcal{M}_c(\rho_c)\mathcal{M}_s(\rho_c, \rho_s)$ as

$$\mathcal{M}_c(\rho_c)\mathcal{M}_s(\rho_c, \rho_s) = \widehat{\mathcal{M}}(\rho_c, \rho_s) \quad (2)$$

I/O Scheduler Model: Our storage system testbed, *Akash* [3], uses the quanta-based scheduler to divide the storage bandwidth among several virtual volumes. The quanta-based scheduler partitions the bandwidth by allocating a time quantum where one of the workload obtains exclusive access to the underlying disk. For modeling the quanta latency, we observe that the typical server system is an *interactive*, closed-loop system. This means that, even if incoming load may vary over time, at any given point in time, the rate of serviced requests is roughly equal to the incoming request rate. Then, according to the *interactive response time law* [2]:

$$L_d = \frac{N}{X} - Z \quad (3)$$

where L_d is the response time of the storage server, including both I/O request scheduling and the disk access latency, N is the number of application threads, X is the throughput, and Z is the think time of each application thread issuing requests to the disk. We then use this formula to derive the average disk access latency for each application, when given a certain fraction of the disk bandwidth. We assume that think time per thread is negligible compared to request processing time, i.e., we assume that I/O requests are arriving relatively frequently, and disk access time is significant. Then, through a simple derivation, we arrive at the following formula

$$L_d(\rho_d) = \frac{L_d(1)}{\rho_d} \quad (4)$$

where $L_d(1)$ is the *baseline disk latency* for an application, when the entire disk bandwidth is allocated to that application. This formula is intuitive. For example, if the entire disk was given to the application, i.e., $\rho_d = 1$, then the storage access latency is equal to the underlying disk access latency. On the other hand, if the application is given a small fraction of the disk bandwidth, i.e., $\rho_d \approx 0$, then the storage access latency is very high (approaches ∞).

Overall Storage Performance – A-STOR: Assuming that the hit access latency in the buffer pool is negligible, the overall latency is determined by the accesses that miss in the buffer pool and either (i) hit in the storage cache or (ii) miss in the storage cache, hence access the disk. We also assume that the access latency for a hit/miss in the storage cache is approximately the network/disk latency, i.e., L_{net}/L_d , respectively, then the average data access latency is

$$\begin{aligned} \mathcal{R}_{page}(\rho_c, \rho_s, \rho_d) &= \underbrace{\mathcal{M}_c(\rho_c)\mathcal{H}_s(\rho_c, \rho_s)L_{net}}_{\text{I/Os satisfied by the storage cache}} \quad (5) \\ &+ \underbrace{\mathcal{M}_c(\rho_c)\mathcal{M}_s(\rho_c, \rho_s)L_d(\rho_d)}_{\text{I/Os satisfied by the disk}} \end{aligned}$$

where the miss (and hit) ratio at the storage cache, i.e., $\mathcal{M}_s(\rho_c, \rho_s)$, is a function of both the quota at the first level cache (ρ_c), and the quota at the second level cache (ρ_s), while the miss-ratio of the buffer pool, $\mathcal{M}_c(\rho_c)$, is only a function of ρ_c . More details of this derivation can be found in [3].

2.2 Modeling DBMS Performance

Next, we describe the analytical models predicting the performance of DBMS systems. A DBMS is a complex system consisting of many software modules making it difficult to model all aspects of the system. Thus,

in our approach, we build lightweight approximations of the components critical to the performance of the application in specific operating modes. We classify the operating mode of the database application into either I/O-intensive or CPU-intensive and build simpler analytical models for each mode as follows.

I/O Intensive Query Model – A-DISK: The I/O-intensive query model is an analytical model designed for I/O intensive workloads. Thus the model does not consider the time at the CPU because if a workload is I/O-bound then, most of the query processing time is in fact spent on waiting for I/O completion. In this case, we predict the query latency to be

$$\mathcal{R}_{query}(\rho_p, \rho_c, \rho_s, \rho_d) = N_{acc} * \mathcal{R}_{page}(\rho_c, \rho_s, \rho_d) \quad (6)$$

where N_{acc} is the average number of page accesses made for each query in the workload and \mathcal{R}_{page} is the average data access latency as derived in Equation 5.

CPU-intensive Query Model – A-CPU: Conversely, the CPU-intensive query model is designed for CPU intensive workloads. It does not consider the I/O cost with the belief that if a workload is CPU intensive then, it either has very few data accesses or most of data accesses hit in the memory (i.e., the DBMS buffer pool). As a result, we can safely ignore the disk cost under this case. The model predicts the query latency as

$$\mathcal{R}_{query}(\rho_p, \rho_c, \rho_s, \rho_d) = \frac{\mathcal{R}_{query}(1, 1, 1, 1)}{\rho_p} \quad (7)$$

where $\mathcal{R}_{query}(1, 1, 1, 1)$ is the baseline query latency for an application, when all resources are fully allocated to that application.

3 Gray-box Performance Models

Gray-box models exploit high-level domain knowledge about the system, and combine them with experimental data samples to perform their prediction. Unlike analytical models, there is no need to have a complete knowledge about the system, partial knowledge or even high-level observations can be used.

Gray-box Region Model – G-RGN: The region-based model exploits the fact that while the performance models of applications are complex in general, they are simple within an operating region, i.e., constant, linear, or polynomial. Hence, we can easily model the performance using simple curve-fitting within a region where the experimental samples are used to find the best-fit for parameters of a given curve. In our evaluation, we use a simple average function to fit our curve.

Gray-box Inverse Model – G-INV: The inverse model is generalized from the behavior we have seen in

proportional-share schedulers, i.e., quanta-based schedulers as shown in Equation 4. It describes a function

$$\hat{y}_{\alpha,\beta}(\rho_d) = \frac{\alpha}{\rho_d^\beta} \quad (8)$$

where y is data access time, and ρ_d is the disk bandwidth fraction allocated to the application. The parameters α and β are estimated using performance samples. We further observe that if a database workload is operating in an I/O intensive mode, then this *inverse* relationship can be extended to between DBMS query latency and the disk bandwidth proportion as well.

4 Black-box Performance Models

We also build black-box models to predict application performance; the advantage of these models is that they do not need to know any information of the internals of the system and thus can be easily applied to model many systems. One such method is the use of statistical regression-based models that use a set of training points to train their model parameters for prediction. In particular, we use *support vector machine regression* [1] (B-SVM), to estimate the performance for configuration settings we have not actuated, through interpolation between a given set of sample points.

Black-box SVM Model – B-SVM: Support vector machine regression is a nonlinear regression algorithm that scales well for highly-dimensional, non-linear data and works as follows. Given a set of training points

$$S = \{(\vec{x}_1, y_1), \dots, (\vec{x}_m, y_m)\}$$

SVR finds a smooth function $\hat{f}(\vec{x})$ that has a small deviation (ϵ) from the targets y_i for all training data. The estimated function $\hat{f}(\vec{x})$ takes the form

$$\hat{f}(\vec{x}) = \sum_{i=1}^m \alpha_i y_i K(\vec{x}_i, \vec{x}) \quad (9)$$

where each training point \vec{x}_i is associated with a variable α_i that represents the strength with which the training point is embedded in the final function. The points which lie closest to the hyperplane, denoting $\hat{f}(\vec{x})$, are called the *support vectors*. $K(\vec{x}_i, \vec{x})$ is a kernel function which maps the input into a high dimensional space, called feature space, where linear support vector regression is applied. We use radial basis functions (RBFs) as our kernel functions. Once the data is transformed using the kernel functions, the training of SVR consists of solving a convex optimization problem using quadratic programming.

Black-box Constant Model – B-CNST: This is a simple model; it has no knowledge of the underlying system,

and simply returns the average value of training samples to predict performance for unknown configurations. Due to this, it usually performs poorly since the actual performance surface is not constant. We build this model to evaluate that if we have a *bad* model with high errors in our collection affects the performance of the *hybrid* model.

References

- [1] H. Drucker, C. J. C. Burges, L. Kaufman, A. J. Smola, and V. Vapnik. Support Vector Regression Machines. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS'96)*, pages 155–161, 1996.
- [2] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling*. John Wiley & Sons, 1991.
- [3] G. Soundararajan, D. Lupei, S. Ghanbari, A. D. Popescu, J. Chen, and C. Amza. Dynamic Resource Allocation for Database Servers Running on Virtual Storage. In *FAST*, pages 71–84, 2009.
- [4] T. M. Wong and J. Wilkes. My Cache or Yours? Making Storage More Exclusive. In *Proceedings of the 2002 USENIX Annual Technical Conference (USENIX'02)*, pages 161–175, 2002.