Applied Statistical Inference for System Design and Management

Benjamin C. Lee
Electrical and Computer Engineering, Duke University
benjamin.c.lee@duke.edu

***Abstract*—We review strategies for applying statistical inference to system design and management. In design, inferred models act as surrogates for expensive simulators and enable qualitatively new studies. In management, inferred models predict outcomes from allocation and scheduling decisions, and identify conditions that make performance stragglers more likely.**

I. Introduction

In 1965, Gordon Moore applied statistical inference to translate data into a decision [1]. The data included four measurements—the number of integrated components per chip—collected for several consecutive years. A linear fit on the data in log scale produced a trend and motivated a decision. Specifically, the number of integrated components should double periodically to reduce cost and increase capability in electronics design. Hence, the birth of Moore's Law.

Today, statistical inference is even more important when using data to meet computer architects' dual mandates in hardware design and management. Unfortunately, methods for analysis and interpretation have lagged even as data sources have become increasingly prolific. Design data flows from diverse frameworks such as RTL simulation, cycle-level models, or full-system emulation. Management data flows from system profilers and hardware counters. The data deluge requires inference, and its cousins in machine learning and data mining, to produce intelligent decisions and efficient outcomes.

Sophisticated decisions are needed for varied outcomes in modern computer systems. In the future, the trade-off space must accommodate nuanced measures of performance—e.g., latency, throughput, percentiles—and the design space must accommodate diverse approaches to power efficiency—e.g., heterogeneity, customization, computational sprinting. The trend towards complex systems and measures of their performance requires corresponding advances in a broad array of inferred models that drive design and management.

In this paper, we review strategies for applying statistical inference to system design and management. We describe when the strategies have proven effective and identify system settings that motivate new ones. Much prior work has focused on one setting, processor design [2]–[5], and we face many challenges in less explored settings. New strategies are needed when design is coordinated across multiple hardware components or across the hardware-software interface. Furthermore, new strategies are needed when using inferred models for runtime management.

II. Translating Data into Decisions

Computer architects suffer from the Goldilocks problem. During design, architects rely on slow and unwieldy simulators that produce too little data and preclude sophisticated design space exploration. During management, architects instrument running systems with pervasive profilers that provide too much data without revealing key, domain-specific insights. The problem of having too little or too much data will only grow.

Design and the Data Deficit. The quantitative approach to computer architecture heralded an era of simulator-driven design. Simulators estimate performance and other figures of merit by tracking activity in a candidate design. However, software runs three orders of magnitude more slowly in cycle-level simulation. MARSSx86 and Gem5 commit 200-300K instructions per second [6], [7]. Simulation costs are exacerbated by two trends. First, server designers target workloads that require full-system simulation; e.g., a Spark simulation must include the OS and run-time engine. Second, embedded designers pursue efficiency with accelerators, which require custom timing models or simulation with synthesized RTL.

Unwieldy simulators constrain design space exploration. Enumerating and simulating a design space is prohibitively expensive, especially as the number of hardware parameters and software benchmarks increases. Fundamental design methods, such as optimizing power efficiency with heuristic search or constructing a Pareto frontier, are intractable when the inner loop of an iterative heuristic invokes a simulator. These challenges multiply when optimizing several hardware components (e.g., processor and memory) in coordination.

Management and the Data Glut. At the other end of the spectrum, deployed systems produce a glut of data. System profilers report processor, memory, and network activity while hardware counters report microarchitectural events. At datacenter scale, profiles for thousands of servers and millions of tasks produce huge traces. For example, the Google-wide Profiler produces a database of software task parameters, hardware platform parameters, system utilization measurements, and scheduling events [8]. A 180GB trace for 29 days of datacenter activity covers 11K nodes, 925 users, 650K jobs and 25M tasks [9].

Large, comprehensive system profiles supply so much data that they obscure actionable insight. A database query for a specific profile fails to yield broader insight for management policies. Yet we need policies for diverse management

goals such as navigating machine heterogeneity, co-locating software tasks on shared hardware platforms, and quantifying hardware-software interactions during resource allocation and task scheduling. In each of these settings, profiles can supply data to support decisions.

III. Applying Statistical Inference

Statistical inference and its related methods in machine learning and data mining provide a rich toolset for design and management. We describe a few representative methods that highlight relevant capabilities.

Regression. First, consider a simple regression model that estimates response y from a linear combination of parameters x with some random error ϵ [10]. For design, y might measure instruction throughput and $x_1, x_2, \ldots$ might describe datapath resources and cache geometries.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \epsilon$$

Although the regression model is linear with respect to its fitted coefficients, non-linear transformations on x and y can produce the flexible models. For example, splines partition the parameter range and fit different coefficients to each range, producing a piecewise polynomial model—see spline $S(x)$ on x. Non-linearity is helpful when modeling diminishing returns such as Amdahl's Law or cache performance when data locality is limited.

$$S(x) = \alpha_1 x + \alpha_2 x^2 + \alpha_3 x^3 + \alpha_4 (x-a)_+^3 + \alpha_5 (x-b)_+^3$$

$$\text{where} (x-a)_+^3 = \max\{(x-a)^3, 0\}$$

Finally, regression captures scenarios in which two parameters interact to affect the outcome. A product term in the model ensures that the impact from one parameter depends on the value of the other.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2 + \epsilon$$

Neural Networks. Neural networks are comprised of neurons connected by weighted edges [11]. One neuron's outputs feed another's inputs to create a multi-layer network—see Figure 1. A neuron consumes parameters $x_1, x_2, \ldots$ and produces a value v, which is calculated by a dot product of parameters x and weights w followed by an activation function f such that $v = f(\sum x_i w_i)$. For example, input neurons use identity $f(x) = x$ and hidden neurons often use sigmoid $f(x) = (1 + e^{-x})^{-1}$. Deep neural networks extend these concepts to many layers, composing features and model outputs from earlier layers to capture their sophisticated, non-linear impact on outcomes.

Recommenders and Classifiers. Finally, classifiers are models that model and predict discrete outcomes. Suppose a computer architect wishes to know whether a software task will be a performance straggler given hardware conditions. Logistic regression models the probability that an event occurs as a linear combination of input parameters. By comparing the modeled probability against a threshold, we can construct a classifier that predicts whether an event will occur.

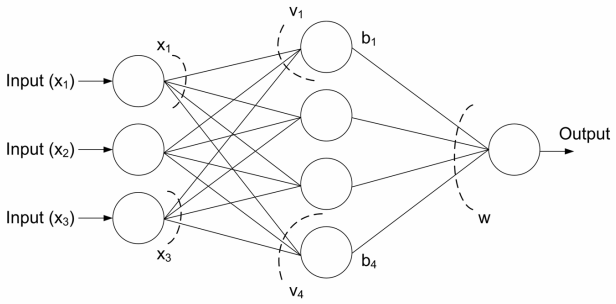

Fig. 1. Neural network with one hidden layer.

$$\log \frac{\mathrm{P}[y=1]}{1-\mathrm{P}[y=1]} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots$$

Related techniques include collaborative filtering, which estimates each software task's preference for hardware. Collaborative filtering employs a software-hardware matrix in which $M_{i,j}$ reflects task i's preference for platform j. The matrix is large and sparse with many unobserved values. To learn missing values, collaborative filtering performs a regularized least squares fit, estimating matrix values to minimize sum of square errors while penalizing solutions that use many large values, solutions that increase risk of over-fitting [12].

Limitations and Challenges. Inference faces challenges in feature selection, the process of identifying inputs to model the output. The architect must choose among candidate features, a combinatorial number of interactions between them, and non-linear transformations. Heuristics, such as stepwise regression, help by searching for statistically significant predictors of the response. Combined with domain-specific expertise, heuristics can produce effective models. But more research in feature selection and cost-effective training is needed.

IV. Exploring Design Spaces

Architects have successfully applied inference to model processor performance as a function of design parameters, creating surrogates for expensive simulators. These surrogates have enabled previously intractable design space studies. Yet, open questions remain for other hardware components, more complex systems, and interactions with tunable software.

Creating Surrogates for Simulation. Flexible, spline-based regression models are capable of capturing sophisticated, non-linear relationships in the processor design space. Computer architects can define an ambitious design space with 15-20 parameters and tens of thousands of designs, sample and simulate a few hundred points from the space, and fit accurate regression models [2]. Input parameters describe the pipeline depth, datapath width, and cache hierarchy. Model outputs estimate performance (i.e., instruction throughput) and power. In this setting, regression models are accurate and estimate figures of merit with median errors between 5-10% when validated against industrial strength simulators [2], [4].

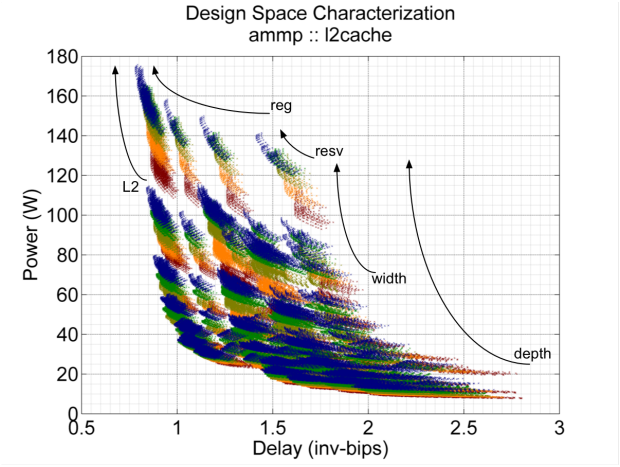

Fig. 2. Pareto frontier for design space that varies pipeline depth, issue width, register file, reservation stations, and L2 cache [3]. Colors correspond to varied L2 cache values. Power reported for 180nm process.

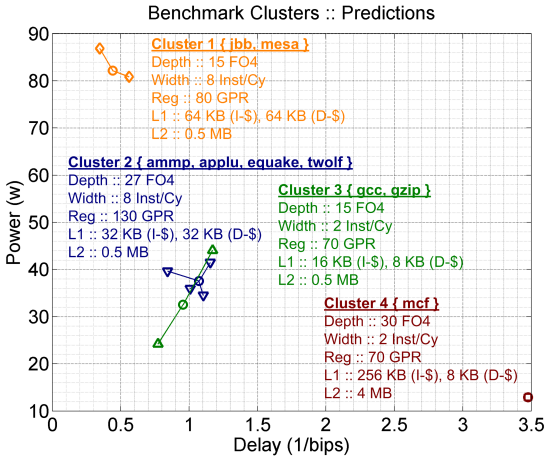

Fig. 3. Heterogeneous design clusters for nine applications [3]. Circles denote designs identified by K-means clustering. Radial points, attached to circles, denote most efficient designs for applications in cluster.

As surrogates for simulators, models can estimate performance and power for thousands of designs in seconds, enabling previously intractable design studies [3]. First, consider Pareto frontiers in performance-power coordinates. A Pareto optimal design is such that an architect cannot improve performance without increasing power or reduce power without decreasing performance. In other words, the frontier reveals the best design for a given power budget or the least expensive design for a given performance target. Searching for Pareto optima is intractable with design simulation, but is trivial with statistically inferred models.

Second, models permit sophisticated design optimization [3]. Suppose an architect wished to design multiple, heterogeneous processor cores to maximize efficiency for an application suite. With regression models, the architect can identify the most efficient design for each application. Then, she can use K-means to cluster applications with similar design preferences. Each cluster produces a core type well suited to applications in the cluster. By tuning the number of clusters, the architect tunes the degree of heterogeneity in the system. Thus, architects can optimize the number of core types and organize those types in a system to maximize efficiency.

Finally, computer architects should look beyond hardware design and consider software parameters. Software performance varies with input data, compiler optimizations, and data structure choices. An architect can characterize software behavior with microarchitecture-independent measures of performance that are portable across core designs (e.g., re-use distance, not cache miss rate). A performance model $z = \beta_0 + \beta_1 x + \beta_2 y + \beta_3 xy$ can account for interactions between hardware parameters (x) and software parameters (y). Such models can predict performance for previously unseen hardware-software pairs [5].

Looking Beyond the Processor. Statistical inference for hardware components beyond the processor core is less common yet well motivated. For evolving memory technologies, such as PCM and STT-MRAM, architects can use statistically inferred models to parameterize their assumptions about technology trends and assess implications for system architecture. Alternatively, architects could design a system with ideal parameters and then determine the mix of memories that produce the desired properties.

Domain-specific accelerators present even greater challenges for design space exploration. High-level synthesis can translate an algorithmic description into RTL to support performance and power simulation. Statistically inferred surrogates for this accelerator design flow would permit qualitatively new studies. However, identifying parameters that define the design space may require domain-specific expertise.

As we apply statistical inference to progressively more system components, architects could turn to composable models with cleanly defined interfaces. For example, separate models for the processor core and memory system could be linked by a measure of memory access time. Composable models would reduce the costs of data collection as each component model would train on a modest number of data points from its design space [4].

V. Driving Management Decisions

Architects have only recently turned to statistical inference for difficult questions in systems management. Inferred models can estimate software performance under operating conditions that vary according to hardware heterogeneity and software contention for shared resources. Beyond efforts in collaborative filtering and straggler classification, however, further research is needed in run-time models that drive allocation and scheduling.

Collaborative filtering naturally fits the resource assignment problem [13]. A system can profile performance for sparsely sampled hardware-software pairs to construct a matrix of software ratings for hardware — $M_{i,j}$ describes task i's performance on platform j. The matrix M can be factored into matrices P and Q, which describe the platform's features and a

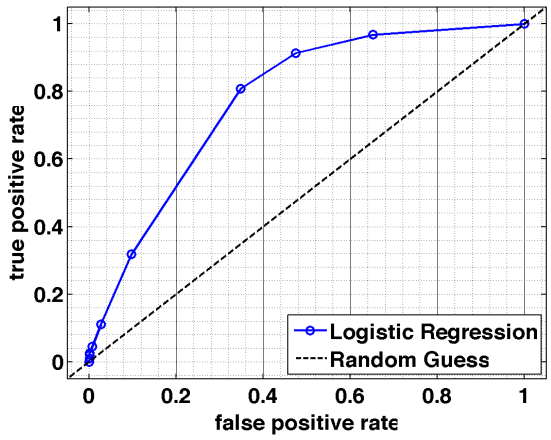

Fig. 4. Each threshold value, shown by data point, produces a TPR and FPR. When threshold is 0.3, TPR is 0.81 and FPR is 0.35.

task's preferences for each platform, respectively. Factorization produces $M' = PQ$ and provides a dense matrix that estimates every task's preferences for every platform.

Beyond predicting average performance, architects must understand stragglers in distributed systems, which lengthen the critical path and harm service quality. Yet, identifying system conditions that make stragglers more likely is difficult because they occur so rarely. Logistic regression can model the probability of a straggler with task, system, and architecture parameters. Such a model can help diagnose root causes.

We build a logistic regression model for stragglers in a Google datacenter. The binary response y is false for nominal tasks and true for outliers; tasks that report instruction throughput within the worst 25% of all profiles are considered outliers. Input parameters include the task's requests for hardware, its allocation of hardware, and system utilization. Regression coefficients are fit using 10K tasks from the datacenter trace [9]. The following model estimates the odds of an outlying tasks, where $x\beta = \log(\mathrm{P}[y=1]/P[y=0])$.

$$\begin{aligned} X\hat{\beta} =& -0.63 - 0.46\,\mathrm{cpuMean} - 0.04\,\mathrm{cpuMax} \\ &+0.62\,\mathrm{memSpace} + 0.10\,\mathrm{memMax} - 0.70\,\mathrm{memAssigned} \\ &+2.60\,\mathrm{pgCacheUnmapped} - 1.71\,\mathrm{pgCacheTotal} \\ &+0.17\,\mathrm{diskMean} - 0.15\,\mathrm{diskMax} + 8.08\,\mathrm{diskSpace} \\ &-0.02\,\mathrm{cpuReq} + 0.03\,\mathrm{memReq} + 1.11\,\mathrm{diskReq} \\ &-0.30\{1\} - 0.23\{2\} + 0.21\{3\} + 0.12\,\mathrm{priority} \end{aligned}$$

This regression model is a good fit and reports a Brier score of 0.15. Note that the Brier score uses N task profiles to compute $(1/N)\sum_{t=1}^{N}(f_t - o_t)^2$ where $0 \le f_t \le 1$ is the forecasted probability that task t is an outlier and $o_t \in \{0,\ 1\}$ is the actual outcome.

Classifier and model inputs are known prior to computation or soon after it begins. The classifier invokes the regression model to predict outlier probability and compares this probability against a threshold. The true positive rate (TPR) is the number of outliers classified as such and the false positive rate (FPR) is the frequency of false alarms for nominal tasks.

Figure 4 plot TPR versus FPR as the threshold varies. A good classifier that detects most outliers and rarely raises false alarms occupies the upper-left corner of the figure. Classifying a task an outlier when $\mathrm{P}[y = 1] > 0.3$ strikes an attractive balance — 81% of outliers are classified correctly and 35% of positives are false alarms. For perspective, logistic regression is far more accurate than random guesses, which produce true and false positives in equal measure. Thus, we successfully mine datacenter profiles for predictive relationships.

VI. Conclusions and Future Directions

Methods in statistical inference can drive management decisions. However, new strategies for training these models in dynamic systems will be required. When a system continuously profiles system behavior, models should continuously train and adapt to new data. Architects might turn to Bayesian methods, which specify priors on the model based on existing data and compute posteriors for the model based on new data. Dynamic models blur boundaries between training and prediction.

Interpreting models and separating causal relationships from correlation continue to be challenging. Although regression model and neural networks predict outcomes accurately, diagnosing root causes is challenging. Yet causality analysis would help architects design and manage the system for better outcomes. Design for manageability means anticipating management challenges during design and producing systems that are more likely to meet quality-of-service targets.

Acknowledgments

This work is supported by NSF grants CCF-1149252, CCF-1337215, and AF-1408784, as well as by STARnet, a Semiconductor Research Corporation Program, sponsored by MARCO and DARPA. Any findings in this material are those of the author(s) and do not reflect the views of these sponsors.

References

[1] G. Moore, "Cramming more components onto integrated circuits," *Electronics Magazine*, 1965.
[2] B. Lee and D. Brooks, "Accurate and efficient regression modeling for microarchitectural performance and power prediction," in *ASPLOS*, 2006.
[3] ——, "Illustrative design space studies with microarchitectural regression models," in *HPCA*, 2007.
[4] B. Lee *et al.*, "CPR: Composable performance regression for scalable multiprocessor models," in *MICRO*, 2008.
[5] W. Wu and B. Lee, "Inferred models for dynamic and sparse hardware-software spaces," in *MICRO*, 2012.
[6] A. Patel *et al.*, "MARSSx86: A full system simulator for x86 CPUs," in *DAC*, 2011.
[7] N. Binkert *et al.*, "The gem5 simulator," 2011.
[8] G. Ren *et al.*, "Google-wide profiling," *IEEE Micro*, 2010.
[9] C. Reiss *et al.*, "Heterogeneity and dynamicity at scale: Google trace analysis," in *SOCC*, 2012.
[10] F. Harrell, *Regression modeling strategies*. Springer, 2001.
[11] T. Mitchell, *Machine learning*. McGraw Hill, 1997.
[12] Y. Koren *et al.*, "Matrix factorization techniques for recommender systems," *IEEE Computer*, 2009.
[13] C. Delimitrou and C. Kozyrakis, "Paragon: QoS-aware scheduling for heterogeneous datacenters," in *HPCA*, 2013.