Predicting and Optimizing System Utilization and Performance via Statistical Machine Learning

Archana Sulochana Ganapathi

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2009-181
http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-181.html

December 17, 2009

Predicting and Optimizing System Utilization and Performance via Statistical Machine Learning

by

Archana Sulochana Ganapathi

A dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor David A. Patterson, Chair
Professor Armando Fox
Professor Michael Jordan
Professor John Chuang

Fall 2009

Predicting and Optimizing System Utilization and Performance via Statistical Machine Learning

by

Archana Sulochana Ganapathi

Abstract

Predicting and Optimizing System Utilization and Performance via Statistical Machine Learning
by
Archana Sulochana Ganapathi
Doctor of Philosophy in Computer Science
University of California, Berkeley
Professor David A. Patterson, Chair

The complexity of modern computer systems makes performance modeling an invaluable resource for guiding crucial decisions such as workload management, configuration management, and resource provisioning. With continually evolving systems, it is difficult to obtain ground truth about system behavior. Moreover, system management policies must adapt to changes in workload and configuration to continue making efficient decisions. Thus, we require data-driven modeling techniques that auto-extract relationships between a system's input workload, its configuration parameters, and consequent performance.

This dissertation argues that statistical machine learning (SML) techniques are a powerful asset to system performance modeling. We present an SML-based methodology that extracts correlations between a workload's pre-execution characteristics or configuration parameters, and post-execution performance observations. We leverage these correlations for performance prediction and optimization.

We present three success stories that validate the usefulness of our methodology on storage and compute based parallel systems. In all three scenarios, we outperform state-of-the-art alternatives. Our results strongly suggest the use of SML-based performance modeling to improve the quality of system management decisions.

Dedicated to
the Dancing Enchanter of Brindavan

and

to my family,
for being pillars of support throughout my life.

Contents

List of Figures iv

List of Tables vii

1 Introduction 1

2 Formal Methods for System Performance Modeling 4

2.1 Challenges in System Performance Modeling 4

2.2 Statistical Machine Learning for Performance Modeling 6

2.3 A KCCA-based Approach to System Performance Modeling 8

2.4 Related Work in Formal System Modeling 13

2.5 Summary 15

3 Predicting Performance of Queries in a Parallel Database System 16

3.1 Motivation 16

3.2 Experimental Setup 18

3.3 Addressing Design Challenges of Using KCCA 20

3.4 Results 28

3.5 Lessons and Limitations 38

3.6 Related Work 42

3.7 Summary 45

4 Predicting Performance of Production Hadoop Jobs **46**

4.1 Motivation . . . 46
4.2 Experimental Setup . . . 48
4.3 Addressing Design Challenges of Using KCCA . . . 50
4.4 Results . . . 53
4.5 Lessons and Limitations . . . 56
4.6 Related Work . . . 58
4.7 Summary . . . 59

5 Auto-tuning High Performance Computing Motifs on a Multicore System **60**

5.1 Motivation . . . 60
5.2 Experimental Setup . . . 63
5.3 Addressing Design Challenges of Using KCCA . . . 66
5.4 Results . . . 70
5.5 Lessons and Limitations . . . 73
5.6 Related Work . . . 80
5.7 Summary . . . 80

6 Reflections and Future Work **81**

6.1 Most promising future directions for our three case studies . . . 81
6.2 Useful side-effects of KCCA-based modeling . . . 82
6.3 Beyond performance modeling . . . 83
6.4 New directions for SML research . . . 84

7 Conclusions **86**

Bibliography **88**

List of Figures

2.1 Our goal is to mimic the behavior of a real system using a statistical model. 5
2.2 KCCA projects workload and performance features onto dimensions of maximal correlation. 9
2.3 Training: From vectors of workload features and performance features, KCCA projects the vectors onto dimensions of maximal correlation across the data sets. Furthermore, its clustering effect causes "similar" workload instances to be collocated. 12

3.1 Schematic architecture of HP's Neoview database. This image is borrowed from: http://h20195.www2.hp.com/v2/GetPDF.aspx/4AA2-6924ENW.pdf . 19
3.2 KCCA-predicted vs. actual elapsed times for 61 test set queries, using SQL text statistics to construct the query feature vector. We use a log-log scale to accommodate the wide range of query execution times. The R^2 value for our prediction was -0.10, suggesting a very poor model. 23
3.3 Using the query plan to create a query vector: vector elements are query operator instance counts and cardinality sums. 24
3.4 Testing: KCCA projects a new query's feature vector, then looks up its neighbors from the performance projection and uses their performance vectors to derive the new query's predicted performance vector. 26
3.5 KCCA-predicted vs. actual elapsed times for 61 test queries. We use a log-log scale to accommodate the wide range of query execution times from milliseconds to hours. The R^2 value for our prediction was 0.55 due to the presence of a few outliers (as marked in the graph). Removing the top two outliers increased the R^2 value to 0.95. 29
3.6 KCCA-predicted vs. actual records used. We use a log-log scale to accommodate wide variance in the number of records used. The R^2 value for our prediction was 0.98. (R^2 value close to 1 implies near-perfect prediction). . . 30
3.7 KCCA-predicted vs. actual message count. We use a log-log scale to accommodate wide variance in the message count for each query. The R^2 value for our prediction was 0.35 due to visible outliers. 31

3.8 Regression-predicted vs. actual elapsed times for 1027 training and 61 test queries. The graph is plotted on a log-log scale to account for the wide range of query runtimes. 176 datapoints are not included in the plot as their predicted times were negative numbers. The triangles indicate datapoints that were in our test set (not used to build the regression model). 32
3.9 Regression-predicted vs. actual records used for 1027 training and 61 test queries. Note that 105 datapoints had negative predicted values going as low as -1.18 million records. The triangles indicate datapoints that were in our test set (not used to build the regression model). 33
3.10 KCCA-predicted vs. actual elapsed times. The axes are log-log scale to accommodate wide variance of elapsed times in the 61 queries of our test set. 34
3.11 Two-model KCCA-predicted elapsed time vs. actual elapsed times for the 61 queries in our test set. The first model classifies the query as a feather, golf ball or bowling ball; the second model uses a query type-specific model for prediction. A log-log scale was used to accommodate wide variance in query elapsed times. The R^2 value for our prediction was 0.82. 35
3.12 A comparison of one-model KCCA-prediction, two-model KCCA-prediction and actual value of elapsed times for a test set of customer queries. 36
3.13 Comparison of KCCA-predicted elapsed time for workloads running at multi-programming level = 4, elapsed times for the same workloads running at multi-programming level = 1 on the simulator, and elapsed time for test workloads run at multi-programming level = 4 in the simulator. 38
3.14 Optimizer-predicted costs vs. actual elapsed times for a set of queries, the same test queries used later for Experiment 1 in Section 3.4. The optimizer costs and measured times came from Neoview's commercial query optimizer. The horizontal line marks a computed linear best-fit line. 39
3.15 Optimizer-predicted costs vs. actual elapsed times for a set of queries, the same test queries used later for Experiment 1 in Section 3.4. The optimizer costs and measured times came from Neoview's commercial query optimizer. 40

4.1 Architecture of MapReduce from [Dean and Ghemawat, 2004]. 49
4.2 Predicted vs. actual execution time for Hive queries, modeled using Hive operator instance counts as job features. The model training and test sets contained 5000 and 1000 Hive queries respectively. The diagonal line represents the perfect prediction scenario. Note: the results are plotted on a log-log scale to accommodate variance in execution time. 52

4.3 Prediction results for Hive queries, modeled using job configuration and input data characteristics as job features. The model training and test sets contained 5000 and 1000 Hive queries respectively. The diagonal lines represent the perfect prediction scenario. Note: the results are plotted on a log-log scale to accommodate the variance in data values. (a) Predicted vs. actual execution time, with R^2 value of 0.87. (b) Predicted vs. actual map time, with R^2 value of 0.84. (c) Predicted vs. actual reduce time, with R^2 value of 0.71. (d) Predicted vs. actual bytes written, with R^2 value of 0.86. 54
4.4 Prediction results for ETL jobs, modeled using job configuration and input data characteristics as job features. The model training and test sets contained 5000 and 1000 ETL jobs respectively. The diagonal lines represent the perfect prediction scenario. Note: the results are plotted on a log-log scale to accommodate the variance in data values. (a) Predicted vs. actual execution time, with R^2 value of 0.93. (b) Predicted vs. actual map time, with R^2 value of 0.93. (c) Predicted vs. actual reduce time, with R^2 value of 0.85. (d) Predicted vs. actual bytes written, with R^2 value of 0.61 due to visible outliers. 55
4.5 Hadoop job performance metrics measured using Ganglia. 57

5.1 This figure shows schematics of the Intel Clovertown and AMD Barcelona architectures. Diagrams courtesy of Sam Williams. 64
5.2 3D 7-point stencil: (a) Pseudo-code. (b) Visualization. 65
5.3 PGEMM pseudo-code. 66
5.4 Performance results for the 7-point stencil on Intel Clovertown and AMD Barcelona. 71
5.5 Performance results for the 27-point stencil on Intel Clovertown and AMD Barcelona. 72
5.6 Performance results for PGEMM on Intel Clovertown and AMD Barcelona. . 73
5.7 Energy efficiency on Clovertown for the two stencils. 75
5.8 Performance vs. energy efficiency for the 7-point stencil training data on Clovertown. Note that the slope indicates Watts consumed. The oval highlights configurations with similar performance but different energy efficiencies. 76
5.9 Performance results for the 7-point and 27-point stencils on (a) Intel Clovertown and (b) AMD Barcelona as we vary the number of counters we use to train the model. The error bars represent the standard deviation of peak performance. 78

List of Tables

3.1 We created pools of candidate queries, categorized by the elapsed time needed to run each query on the HP Neoview 4 processor system. 20
3.2 Comparison of R^2 values for using Euclidean Distance and Cosine Distance to identify nearest neighbors. Euclidean Distance has better prediction accuracy. 27
3.3 Comparison of R^2 values produced when varying the number of neighbors. Negligible difference is noted between the various choices. 27
3.4 Comparison of R^2 values produced when using various relative weights for neighbors 28
3.5 Comparison of R^2 values produced for each metric on various configurations of the 32-node system (we show the number of nodes used for each configuration). Null values for Disk I/O reflect 0 disk I/Os required by the queries due to the large amount of memory available on the larger configuration. 35

5.1 Attempted optimizations and the associated parameter spaces explored by the auto-tuner for a 256^3 stencil problem ($NX, NY, NZ = 256$) using 8 threads ($NThreads = 8$). All numbers in the parameter tuning range are in terms of double precision numbers. 67
5.2 Attempted optimizations and the associated parameter spaces explored by the auto-tuner for a 768^3 pgemm problem ($M, N, K = 768$) using 8 threads ($NThreads = 8$). All numbers in the parameter tuning range are in terms of doubles. 68
5.3 Measured performance counters on the Clovertown and Barcelona architectures. 69
5.4 Code optimization categories. 74
5.5 Results for using best Clovertown configurations on Barcelona. 78
5.6 Results for using best Barcelona configurations on Clovertown. 79

Acknowledgements

I am grateful for the time, efforts and insights of several individuals who helped shape the ideas presented in this dissertation.

First and foremost, I thank David Patterson, my advisor and dissertation committee chair. Throughout my undergraduate and graduate career at UC Berkeley, Dave has been an incredible source of inspiration, encouragement and wisdom. His mentorship and unique insights were an immense asset to my research. I am truly fortunate to have had the opportunity to learn from and work with him.

I am grateful to Armando Fox for his guidance on my research. He has co-advised me for a majority of my graduate career, and I have learned a lot from my interactions with him, about systems research as well as musical theater.

I am indebted to Michael Jordan for my statistical machine learning education. He was instrumental to formulating the statistical techniques used in this dissertation. As members of my qualifying exam committee, Eric Brewer, Joe Hellerstein and John Chuang provided invaluable suggestions that substantially improved the content of this dissertation.

Our case studies and experiments benefited from the collaboration and guidance of several people who I gratefully acknowledge. From my collaborators at HP labs, Harumi Kuno, Umesh Dayal and Janet Wiener of HP Labs, I learned a lot about database technology. The success of our parallel database case study is in no small part also due to their tireless efforts to facilitate our experiments. Dhruba Borthakur of Facebook and Matei Zaharia of RADLab helped refine our Hadoop prediction problem set up with timely answers to technical questions. They took significant initiative and efforts to enable access to Facebook's production Hadoop job logs. Kaushik Datta and Marghoob Mohiyuddin provided code generators for the auto-tuning case study. Along with Sam Williams, they also helped interpret and validate our performance results. Furthermore, the members of the BeBOP group provided insightful comments on early results. Jon Kuroda provided unparalleled technical support for various experiments presented in this dissertation.

I have enjoyed many intellectually stimulating conversations with the faculty and students of the RADLab and ParLab. I am thankful for their willingness to provide input on various drafts of papers and presentations of my research. I would also like to acknowledge the RADLab and ParLab retreat attendees for providing valuable feedback.

I am blessed to have a family that is instrumental to my successes and an infallible source of moral support. My father, mother, grandparents, great-grandmother and aunt Uma were my first teachers and role models. My sister Gauri and my husband Nishanth have been steady sources of companionship and patience through the peaks and troughs. I express my heartfelt gratitude to them.

This research is supported in part by a graduate fellowship and grants from the National Science Foundation, the University of California Industry/University Cooperative Research

Program (UC Discovery) grant COM07-10240, and generous gifts from Sun Microsystems, Google, Microsoft, Amazon Web Services, Cisco Systems, Cloudera, eBay, Facebook, Fujitsu, Hewlett-Packard, Intel, Network Appliance, SAP, VMWare and Yahoo!.

Chapter 1

Introduction

Most modern systems are built from a plethora of components that interact in complex ways. For example, decision support databases consist of several homogeneous processors, large capacity disks, and an inter-node connection network. Web services are built from one or more of these databases, several user-facing front-end web servers, and networking equipment such as routers and load balancers. Recently, with the software as a service model in industry, cloud computing infrastructures have risen to make "the datacenter as a computer" [Hoelzle and Barroso, 2009]. These infrastructures typically comprise of a heterogeneous selection of multicore nodes, large-scale storage infrastructures and a variety of networking equipment. Such large multi-component and continuously evolving systems make it difficult to understand the underlying topology of and interactions between components.

There are practical details that make it difficult to model these systems' performance and thus non-trivial to operate and maintain them:

- There is high variability in workload and usage patterns based on diurnal patterns and popular trends. For example, traffic to Apple.com website was 110% higher on black Friday 2009 compared to Thanksgiving day, and 39% higher compared to black Friday 2008 [Albanesius, 2009].

- With 24×7 availability requirements of web services, failures must be handled transparently and thus frequent hardware reconfiguration is inevitable [Oppenheimer *et al.*, 2003]. Storage and/or compute nodes must be provisioned on-demand to minimize downtime.

- Clusters are often built using commodity hardware to minimize cost and improve scalability [Barroso *et al.*, 2003]. Consequently, the available resources and configuration vary among cluster nodes and result in asymmetric node reliability and performance characteristics.

- Web services frequently add features to stay competitive and attract users. The resultant high software churn rate makes these systems more susceptible to bugs, however, and thus requires even more software changes to fix them [Nagappan and Ball, 2007].

- When building multi-component systems, open-source or *out-of-the-box* software is popular. However, different components often have conflicting hardware/software requirements that are difficult to resolve, and the effects of these configuration changes are difficult to identify a priori.

- Lastly, a major advantage of using the Cloud infrastructure is the elasticity it provides. As a result, it is expected that a service can grow the number of nodes to handle unexpected load, and reduce nodes used to save money during low load periods. Decisions to grow and shrink capacity must be made frequently and accurately to save money while meeting service level agreements [Armbrust *et al.*, 2009].

To address the above these challenges, infrastructure service providers must model workloads well to anticipate spikes and account for diurnal patterns, efficient resource provisioning to maximize utilization and minimize cost and contention, and lastly, fair and efficient scheduling across multiple users of multiple services. The above decisions would benefit from tools that accurately capture causal relationships between a system's workload, its configuration, and consequent performance characteristics. All these tools require the ability to ask *what-if* questions regarding workload and configuration changes.

A redeeming feature of modern systems is that most of them seamlessly integrate with instrumentation for tracing requests and collecting performance measurements. Such instrumentation facilitates the use of data-driven modeling techniques including *statistical machine learning* (SML). These data-driven techniques treat the system as a black box and require minimal knowledge about system internals and implementation. As a result, *what-if* questions can be answered without recreating the system's hardware/software stack or waiting for a long running job to complete before decision making.

This dissertation addresses the problem of performance modeling for a variety of multi-component parallel systems. We propose a SML-based technique for extracting relationships between workload and system performance as well as between configuration and performance. We demonstrate the usefulness of this technique for performance prediction and performance optimization of a variety of systems.

We summarize the contributions of this dissertation below:

1. We introduce a general performance modeling methodology for storage and compute based parallel systems. Our proposed methodology is based on statistical machine learning techniques and is easy to adapt to a wide variety of systems modeling problems.

2. We show examples of how our methodology can be used for both performance prediction and performance optimization.

3. We demonstrate how to map elements of systems problem to elements of our SML-based methodology. As a result, our techniques reduce the amount of domain knowledge and expertise to get good prediction/optimization results. To use our techniques, one simply needs to identify important features of the input to and output from the system.

4. We present three case studies with good results that demonstrate the usefulness of methodology for real problems on three different parallel systems. Using our methodology, we are able to obtain results that outperform current state-of-the-art techniques in these three domains.

The remainder of this dissertation is organized as follows. In Chapter 2, we formalize the problem of system performance modeling and present our SML-based methodology. This chapter also surveys related statistical machine learning techniques and non-SML alternatives to formal systems modeling. We then introduce three case studies that demonstrate the use of this technique on a decision support database system in Chapter 3, a production MapReduce cluster in Chapter 4, and high performance computing code on multicore processors in Chapter 5. Each of these case studies detail how we adapt our general methodology to the specific problem at hand, compare our results to state-of-the-art technology in that area, and discusses open questions that result from our contribution. Chapter 6 presents lessons learned and future directions, and Chapter 7 concludes.

Chapter 2

Formal Methods for System Performance Modeling

In this chapter, we describe the performance modeling framework that we use in subsequent chapters of this dissertation. We also discuss challenges of using mathematical modeling techniques for systems research, requirements to address these challenges, relevance and shortcomings of other statistical machine learning techniques, and related work.

2.1 Challenges in System Performance Modeling

Figure 2.1 shows a simplistic view of our performance modeling problem. We can observe a system's input workload and configuration parameters, and measure its performance and resource utilization. Our goal is the auto-extraction of relationships between the input parameters and measured performance and utilization metrics. Using statistical machine learning to identify these relationships, we can extrapolate "what-if" workload scenarios using our statistical model in place of a functional instance of the entire system. However, there are several constraints and challenges we must overcome to successfully construct a performance model for a system. Goldszmidt and his colleagues pose three challenges in the intersection of machine learning and systems pertaining to 1) model lifecycle management, 2) human interpretability/visualization, and 3) ease of looking up prior results [Goldszmidt *et al.*, 2005]. We enumerate several additional constraints and derive goals to address these challenges below.

- **Challenge 1:** Systems operators often do not design and construct the system. Consequently, the operator lacks domain knowledge to monitor and debug system performance issues. When components are used *out-of-the-box*, source code may be unavailable or poorly documented.

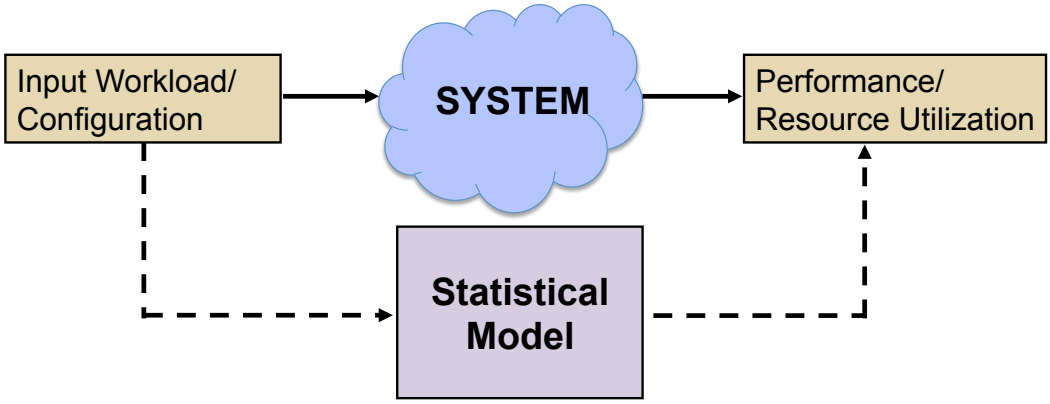

Figure 2.1: Our goal is to mimic the behavior of a real system using a statistical model.

Solution 1: Use a black-box modeling approach with pre-execution parameters and post-execution performance as non-overlapping datasets.

- **Challenge 2:** System workload can be characterized by hundreds of parameters, not all of which impact performance. It is useful to automatically extract a small set of knobs with which to understand and manipulate system performance [Cohen *et al.*, 2005].
  Solution 2: Use techniques that can reduce the dimensionality of input data by automatically selecting features of interest.

- **Challenge 3:** System performance and utilization are measured by several interdependent metrics. Building a separate model per metric does not capture these interdependencies [Cohen *et al.*, 2005]. For example, optimizing performance using one metric's model may adversely impact another metric, and the trade-offs may not be evident without a unified performance model.
  Solution 3: Use a single performance model that captures multiple performance metrics simultaneously.

- **Challenge 4:** System workload input parameters can be non-numeric and/or categorical variables[1]. Some parameter values denote distinct categories and there is no particular ordering between these categories. For example, a code variant can use either *fixed* or *moving* pointer type and thus pointer type is a categorical variable. Many mathematical techniques are specifically designed for scalar data and accounting for

[1]In the statistics literature, the term 'categorical variables' denotes variables whose values range over categories. See http://www.ats.ucla.edu/stat/mult_pkg/whatstat/nominal_ordinal_interval.htm for more details.

these non-scalar parameters can be non-trivial.
Solution 4: Transform the categorical and non-numeric data into scalar features.

- **Challenge 5:** Performance and resource utilization are often measured qualitatively, relative to prior or expected outcome. For example, a database query could have different performance characteristics depending how saturated the system is; comparing the query's execution to prior instances would reveal these differences. While some parameters are easy to quantify, others are purely based on qualitative observations. It is important to define a *closeness* or *similarity* between observations.
  Solution 5: Define distance metrics to capture both quantitative and qualitative parameters.

Recasting the performance modeling problem with these challenges in mind, and given a black-box system, our goal is to find relationships between two multivariate datasets. The first dataset represents the workload and configuration parameters input to the system, and the second dataset represents the measured performance and resource utilization metrics available upon executing the input workload. Statistical machine learning provides a variety of algorithms that can be used for modeling multivariate datasets.

The next section discusses some of the techniques we considered, their relevance for performance prediction and optimization, and shortcomings for addressing the above challenges.

2.2 Statistical Machine Learning for Performance Modeling

Statistical machine learning techniques first derive a model based on a training set of previously executed workload and their measured performance. They then predict performance for unknown workload based on this model. Each workload consists of many *features*, that is individually measureable properties. With the challenges and constraints presented in Section 2.1, we discuss the appropriateness of each algorithm presented below.

2.2.1 Regression

Regression is the simplest machine learning technique with a sound statistical basis. Although we knew that single variable regression would be unlikely to work due to the high dimensionality of workload features, we decided to try multivariate regression to predict each performance metric of interest. We can define independent variables $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ for each workload feature and treat each performance metric as a separate dependent variable y. The goal of regression is to solve the equation $a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + \ldots + a_n\mathbf{x}_n = y$ for the coefficients a_i. Generating a regression curve per dependent variable makes it difficult to account for

interdependencies among the various dependent variables, thus not resolving Challenge 3 as discussed above.

2.2.2 Clustering Techniques

Clustering techniques partition a dataset based on the similarity of multiple features. Typically, clustering algorithms work on a single dataset by defining a distance measure between points in the dataset. While partition clustering algorithms such as Kmeans [Macqueen, 1967] can be used to identify the set of points "nearest" to a test datapoint, it is difficult to leverage these algorithms for performance modeling, as clustering would have to be performed independently on the workload features and the performance features. The points that cluster together with respect to workload features do not necessarily reflect the points that cluster together with respect to performance. Thus, modeling cross-dataset relationships is difficult. In other words, we are looking for a relationship *between* pair-wise datasets.

2.2.3 Principal Component Analysis (PCA)

The oldest technique for finding relationships in a multivariate dataset is Principal Component Analysis (PCA) [Hotelling, 1933]. PCA identifies dimensions of maximal variance in a dataset and projects the raw data onto these dimensions. However, PCA produces projections of each dataset in isolation. Dimensions of maximal variance in workload do not necessarily reflect dimensions that most affect performance. Although PCA addresses the challenge of dimensionality reduction, it does not allow us to identify *correlations* between the two datasets, and thus suffers from the same drawback as clustering.

2.2.4 Canonical Correlation Analysis (CCA)

Canonical Correlation Analysis (CCA) [Hotelling, 1936], is a generalization of PCA that considers pair-wise datasets and finds dimensions of maximal correlation across both datasets. However, CCA violates Challenge 5 above; since it does not have a notion of similarity between the instances being correlated, it is unable to identify which *known* workload instances are qualitatively similar to an *unknown* instance.

2.2.5 Kernel Canonical Correlation Analysis (KCCA)

Kernel Canonical Correlation Analysis (KCCA) [Bach and Jordan, 2003], is a variant of CCA that captures similarity using a *kernel function*. The correlation analysis is on pairwise distances, not the raw data itself. This approach provides much more expressiveness in capturing similarity and its correlations can then be used to quantify performance similarity

of various workloads. KCCA is the statistical machine learning technique we use in this dissertation.

In the next section, we describe KCCA in more detail along with an adaptation of it for performance modeling of systems.

2.3 A KCCA-based Approach to System Performance Modeling

Given two multivariate datasets, KCCA computes basis vectors for subspaces in which the projections of the two datasets are maximally correlated. Figure 2.2 shows a schematic of the transformation KCCA imposes on the datasets. Seemingly dissimilar points in the raw input space may end up clustering together in the projections due to correlation maximization that spans both datasets.

The projection resulting from KCCA provides two key properties:

1. The dimensionality of the raw datasets is reduced based on the number of useful correlation dimensions.

2. Corresponding datapoints in both projections are collocated. Thus, there is a clustering effect that preserves neighborhoods across projections.

KCCA thus addresses all five challenges presented in Section 2.1. We next describe the operational procedure for using KCCA for extracting correlations between system workload and performance.

2.3.1 Constructing feature vectors

The first step is to create feature vectors for all points in the two data sets that we want to correlate. Specifically, this step entails constructing workload feature vectors for each input workload and constructing a performance feature vector for each corresponding observation of system resource utilization and performance. This step must be customized based on available system workload and performance traces.

We denote the set of workload feature vectors $\mathbf{x}_k$ and corresponding performance feature vectors $\mathbf{y}_k$ as $\{(\mathbf{x}_k, \mathbf{y}_k) : k = 1, \ldots, N\}$. Each workload feature vector $\mathbf{x}_k$ has a corresponding performance feature vector $\mathbf{y}_k$. We then combine these vectors into a workload matrix X with one row per workload feature vector and a performance matrix Y with one row per performance feature vector. Each row in the performance matrix represents observations from the workload captured in the corresponding row of the workload matrix.

The next step in customizing KCCA involves representing the similarity between any two workload feature vectors, and between any two performance feature vectors.

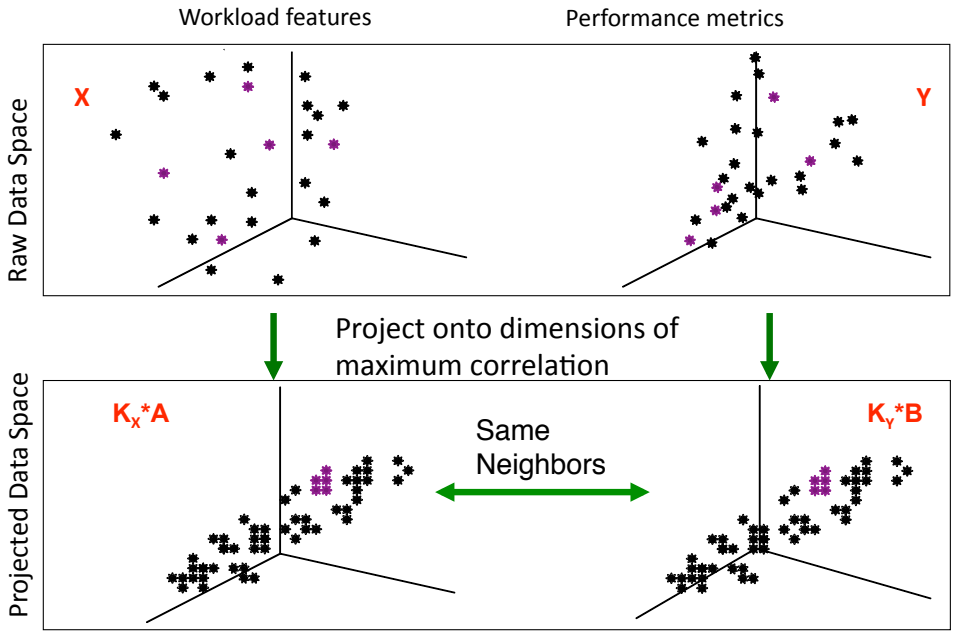

Figure 2.2: KCCA projects workload and performance features onto dimensions of maximal correlation.

2.3.2 Kernel functions to define similarity

KCCA uses kernel functions to compute "distance metrics" between all pairs of workload vectors and pairs of performance vectors. A kernel function allows us to transform non-scalar data into scalar vectors, allowing us to use algorithms that require input vectors in the form of scalar data [Bishop, 2006]. Since our workload and performance features contain categorical and non-numeric data, we create custom kernel functions to transform our datasets.

Given N workload instances, we form an $N \times N$ matrix K_x whose (i,j)th entry is the kernel evaluation $k_x(\mathbf{x}_i, \mathbf{x}_j)$. We also form an $N \times N$ matrix K_y whose (i,j)th entry is the kernel evaluation $k_y(\mathbf{y}_i, \mathbf{y}_j)$. Since $k_x(\mathbf{x}_i, \mathbf{x}_j)$ represents similarity between $\mathbf{x}_i$ and $\mathbf{x}_j$, and similarly for $k_y(\mathbf{y}_i, \mathbf{y}_j)$, the kernel matrices K_x and K_y are symmetric and their diagonals are equal to one.

There are well-defined restrictions on what constitutes a valid kernel and what properties a kernel function must have [Shawe-Taylor and Cristianini, 2004]. The Gram matrices K_x and K_y above must be symmetric and *positive semidefinite*. Specifically, the eigenvalues of K_x and K_y must be real and non-negative and therefore $w^T K_x w >= 0$ and $w^T K_y w >= 0$ for all values of w. A useful property for building new kernels is that a kernel can be constructed from a composition of simpler kernels [Bishop, 2006]. We leverage this property to create a separate kernel function per feature in our datasets, and use a weighted average of these

individual kernel evaluations to generate the kernel function for our workload and performance feature vectors. For example, the workload kernel k_x can be created by combining a kernel function for its numeric features and a kernel function for its non-numeric features.

The specific kernel functions we use depend on the representative workload and performance features for a given system. Once K_x and K_y are created, the next step in our methodology is to maximize the correlation between these two matrices.

2.3.3 Projecting kernel matrices onto dimensions of correlation

The KCCA algorithm takes the kernel matrices K_x and K_y and solves the following generalized eigenvector problem[2]:

$$\begin{bmatrix} 0 & K_xK_y \\ K_yK_x & 0 \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} = \lambda \begin{bmatrix} K_xK_x & 0 \\ 0 & K_yK_y \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix}.$$

This procedure finds subspaces in the linear space spanned by the eigenfunctions of the kernel functions such that projections onto these subspaces are maximally correlated. A kernel can be understood in terms of clusters in the original feature space [Shawe-Taylor and Cristianini, 2004]. Thus, intuitively, KCCA finds correlated pairs of clusters in the workload vector space and the performance vector space.

Operationally, KCCA produces a matrix A consisting of the basis vectors of a subspace onto which K_x may be projected, giving $K_x \times A$, and a matrix B consisting of basis vectors of a subspace onto which K_y may be projected, such that $K_x \times A$ and $K_y \times B$ are maximally correlated. We call $K_x \times A$ and $K_y \times B$ the workload projection and performance projection, respectively.

2.3.4 Tackling the pre-image problem with Nearest Neighbors

We would need to perform the reverse mapping from the post-KCCA projections to the original raw data space to make actionable suggestions for performance prediction and optimization. This problem is an active research area in machine learning, referred to as the *pre-image problem* [Kwok and Tsang, 2003]. Finding a reverse mapping from the projected feature space back to the input space is a known hard problem, both because of the complexity of the mapping algorithm and also because the dimensionality of the feature space can be much higher or lower than the input space (based on the goal of the transformation function).

To address the pre-image problem, we leverage the collocated clustering effect of KCCA projections to approximate a reverse mapping. The mapping between workload and per-

[2]In practice, we use a regularized version of KCCA to prevent overfitting. See [Shawe-Taylor and Cristianini, 2004] for details.

formance is straightforward in the projection space because a workload's projection and its corresponding performance projection lie in similar neighborhoods. This clustering behavior is a useful side effect of using kernel functions in correlation analysis. To map back from a workload or performance projection to the input space, we can leverage these neighborhoods, identify a datapoint's nearest neighbors in the projected space, and look up the raw feature vectors for these nearest neighbors. Depending on the performance modeling goal, we can use various mathematical or heuristic techniques for making actionable prediction or optimization suggestions for the system at hand.

Using nearest neighbor approximations comes with a set of customization parameters. Based on the use case and empirical analysis, one must select the distance metric for measuring the nearness of a neighbor, identify how many neighbors to use for the approximation, and lastly, decide how much influence each neighbor has on the decision/suggestion for performance prediction and optimization. Each of these decisions must be customized based on system-specific features and constraints.

2.3.5 Discussion

Figure 2.3 pictorially summarizes the steps involved in using KCCA for performance modeling, which addresses all the challenges described in Section 2.1. KCCA allows us to build a model without requiring domain knowledge about system internals. The kernel functions used to transform input data to kernel matrices allow us to represent numeric, non-numeric and categorical features and define custom similarity metrics for them. With KCCA, we build a single model for multiple performance metrics and reduce the dimensionality of both the workload and performance datasets by projecting them onto dimensions of correlation. The projections allow us to visualize the similarity of datapoints, and nearest neighbor approaches enable us to provide actionable suggestions for predicting or optimizing various performance metrics.

To maintain the freshness and relevance of our model, we can retrain a model using recent and representative data. Although the training time for KCCA is exponential with respect to the number of datapoints, a modern multicore computer can build a model of thousands of datapoints within minutes. Prediction or optimization using nearest neighbors is inexpensive and linear in cost. Thus, the overhead of building the KCCA model can be amortized by reusing it for quickly making several decisions.

Since KCCA is a generic algorithm, the most challenging aspect of our approach is to formalize the problem of system performance modeling and map it onto the data structures and functions used by KCCA. In particular, we need to make the following three design decisions:

1. How to summarize the pre-execution information about each input workload into a vector of "workload features," and how to summarize the post-execution metrics of a

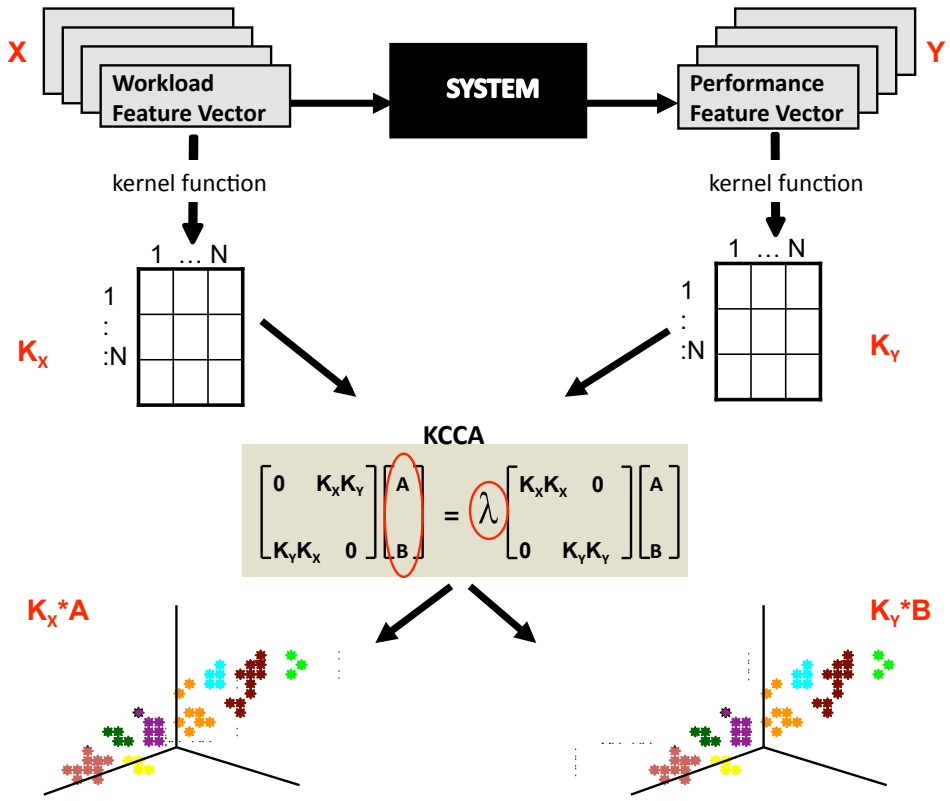

Figure 2.3: Training: From vectors of workload features and performance features, KCCA projects the vectors onto dimensions of maximal correlation across the data sets. Furthermore, its clustering effect causes "similar" workload instances to be collocated.

workload instance into a vector of "performance features".

2. Which kernel functions to use for quantifying the similarity between pairs of workload vectors and pairs of performance vectors.

3. How to leverage the projection neighborhoods and nearest neighbors to make actionable suggestions.

We address each of these issues in the three case studies we present in subsequent chapters of this dissertation.

2.4 Related Work in Formal System Modeling

2.4.1 Manually Constructed Models and Simulators

Traditionally, performance modeling techniques use *expectations* or knowledge of a system's internals for performance prediction and optimization. Analytical models and simulations of system components have been successfully used to model and predict performance of a variety of systems including monolithic operating systems [Shen *et al.*, 2005], I/O subsystems [Ganger and Patt, 1998], disk arrays [Uysal *et al.*, 2001], databases [Sevcik, 1981] and static web services [Doyle *et al.*, 2003]. These models are constructed with a notion of *typical usage* or *expected behavior*, and are not adaptable to subsequent versions or changes in system usage.

2.4.2 Queueing Models for Multi-Component Systems

When a system is constructed from several components, it is difficult to build a single analytical model of the system as a whole. In such systems, queueing theory is used to represent each component as a queue and inter-component interactions as dependencies between the queues [Lazowska *et al.*, 1984]. For example, [Thereska and Ganger, 2008] uses queues to represent resources in a multi-tier distributed system and [Urgaonkar *et al.*, 2005] uses queues to model tiered Internet service applications. While such queueing models capture dependencies between workload and system performance, they rely on extensive knowledge of the system's architecture. Although this knowledge can be incorporated during the system design phase by using assertion checks [Perl and Weihl, 1993] and expectations [Reynolds *et al.*, 2006], it is difficult to enforce such design practices. It is unlikely that assertions and expectations will be diligently updated when systems are synthesized from pre-existing components, designed by multiple entities, and evolve rapidly as is typical for web services.

2.4.3 Dynamic Control Models

Component-specific analytic and queueing models have been augmented with control theory to build models that adapt to changes in system workload and behavior. Such reactive models have been used for a variety of systems research including server resource allocation [Xu *et al.*, 2006], admission control [Diao *et al.*, 2001], database memory tuning [Storm *et al.*, 2006], providing performance guarantees during data migration [Lu *et al.*, 2002], and scheduling in distributed systems [Stankovic *et al.*, 2001].

A major drawback in using control theory for performance modeling is that these models rely on cost functions to decide how to react to workload input parameters [Hellerstein *et al.*, 2004]. These cost functions are user-defined, require domain knowledge to construct, and typically vary in complexity based on the size and variability of workload and system sub-components.

2.4.4 Data Driven Models

With the decline in storage costs and prevalence of tracing and instrumentation frameworks, data driven performance modeling techniques have become an attractive alternative to analytic and cost-based modeling techniques. Large scale distributed systems such as Google's MapReduce cluster process up to 20 petabytes of data per day [Dean and Ghemawat, 2008]. Event log data alone accounts for 1 GB to 1 TB per day [Lemos, 2009]. Such large volumes of data make it impossible to manually perform trend analysis and anomaly detection.

Several researchers have adopted statistical techniques to mine data and build probabilistic models of workload and performance. These data driven models are used for a variety of purposes including workload characterization, anomaly detection, failure diagnosis, resource provisioning and performance tuning. For example, Liblit and others use logistic regression for localizing bugs that cause program behavioral errors [Liblit *et al.*, 2005] while Bodík and his colleagues use the same technique for building signatures of datacenter operational problems [Bodík *et al.*, 2008]. Wang and others use CART models for predicting performance in storage systems [Wang *et al.*, 2004]. Xu and Huang use PCA to detect anomalous patterns in console logs [Xu *et al.*, 2009] and network traffic [Huang *et al.*, 2007], respectively. Xu also builds decision trees from his PCA model to perform root cause analysis for software operational problems [Xu *et al.*, 2009]. While the above examples use a single model for analysis, Zhang and others use ensembles of probabilistic models of system workload to extract correlations between low-level system metrics and violations of service level objectives [Zhang *et al.*, 2005b].

We diverge from prior applications of statistical modeling techniques in that we derive a single model to simultaneously predict or optimize multiple performance metrics.

2.5 Summary

In this chapter, we introduced a statistical machine learning methodology for system modeling. Our methodology uses kernel canonical correlation analysis (KCCA) to extract correlations between system workload and performance, and uses nearest neighbors to leverage these correlations to make actionable performance prediction and optimization decisions. This methodology requires minimal system domain knowledge, serving as an effective black-box modeling technique.

The next three chapters evaluate the usefulness of this methodology for modeling performance in three modern systems — a parallel database, a Hadoop cluster, and various multicore platforms. Each chapter describes the specific performance prediction or optimization problem, constraints imposed by the system being modeled, a customization of our KCCA-based methodology to tackle the modeling problem, and an evaluation of how well the goals and challenges were met.

Chapter 3

Predicting Performance of Queries in a Parallel Database System

3.1 Motivation

Data warehousing is an integral part of business operation in most major enterprises. Many decision support/business intelligence services rely on data warehousing for answering questions ranging from inventory management to accounting and sales reporting. There are several parallel databases that are specifically designed for data warehousing purposes such as Oracle, Teradata, IBM DB2, Microsoft SQL Server, and HP Neoview, to name a few. While the specific implementation of these databases are different – for example, shared-nothing vs. shared-memory/shared-disk – the goals of these databases are similar: i) store and serve large quantities of data, typically terabytes to petabytes; ii) provide tools to load data from various sources and compute useful metrics for decision support; and, consequently, iii) efficiently serve a variety of queries ranging from short-running online transaction processing (OLTP) queries to long-running batch queries with varying deadlines and priorities.

From a consumer perspective, there are many considerations for purchasing a parallel database, including cost, number of processors, memory per node, disk capacity per node, and so on. Many of these decisions are directly impacted by the size and complexity of the anticipated workload in addition to desired concurrency in terms of users and queries and expected response time guarantees. Business intelligence workloads are difficult to predict due to the complexity of query plans involving hundreds of query operators and the high variance of query times ranging from milliseconds to days.

Performance does not in general scale linearly with system size and is highly dependent on the mix of queries and data that constitute a given workload. Predicting behavior not only involves estimating query runtime but also requires being able to anticipate the query's resource requirements, for example, whether a query is CPU-bound, memory bound, disk I/O bound, or communication bound. Identifying query performance characteristics *before*

the query starts executing is at the heart of several important decisions:

- *Workload management*: Should we run this query? If so, when? How long do we wait for it to complete before deciding that something went wrong so that we should kill it? Which queries can be co-scheduled to minimize resource contention?

- *System sizing*: For a new installation of a system, an approximate idea of expected sets of queries, and time constraints to meet, what is the most cost-effective system configuration that can execute this workload while satisfying service level agreements? A new system that is too small will be too slow and the customer will be unhappy. A new system that is too powerful will have better performance, but the price might still leave the customer unhappy. In the worst case, the new system may be both expensive and inappropriate for the customer's workload.

- *Capacity management*: Given an expected change to a workload, should we upgrade the existing system? Can we downgrade to a cheaper configuration without significant adverse effects and performance degradation? For example, purchasing one hundred times more disks and partitioning data across them will not help if poor performance is actually due to insufficient CPU resources. The customer will spend more money than he wanted *and their performance will actually degrade.*

With accurate performance predictions, we would not even start to run the queries that we would later kill if they cause too much resource contention or do not complete by a deadline. Sources of uncertainty, however, such as skewed data distributions and erroneous cardinality estimates, combined with complex query plans and terabytes or petabytes of data, make performance estimation hard. Understanding the progress of an executing query is notoriously difficult, even with complete visibility into the work done by each query operator [Chaudhuri *et al.*, 2005].

This chapter is about predicting query and workload performance and resource use so both businesses and database vendors can decide which system configuration is "good enough." We describe here our efforts to use machine learning to predict, a priori, multiple aspects of query performance and resource usage for a wide variety of queries. We set the following goals:

- Predict elapsed time so that we can identify long-running queries and workload lengths.

- Predict processor usage, memory usage, disk usage, and message bytes sent so that we can anticipate the degree of resource contention and to make sure the system is adequately configured.

- Predict performance using only information available before query execution starts, such as the query's SQL statement or the query plan produced by the query optimizer.

- Predict accurately for both short and long-running queries. The queries that run in a data warehouse system can range from simple queries that complete in milliseconds to extremely complex decision support queries that take hours to run.
- Predict performance for queries that access a different database with a different schema than the training queries. This case is common for a vendor choosing the right system to sell to a new customer.

To meet these goals, we generated a wide range of queries and ran them on an HP Neoview four-processor database system to gather training and testing data. We evaluated the ability of several machine learning techniques to predict the desired performance characteristics. We then validated the effectiveness of the "winning" technique by training and testing on four other HP Neoview configurations.

Our approach finds correlations between the query properties and query performance metrics on a training set of queries and then uses these correlations to predict the performance of new queries. We are able to predict elapsed time within 20% of the actual time, at least 85% of the time, for single queries. Predictions for the other characteristics are similarly accurate and are very useful for explaining the elapsed time predictions. They also enable choosing the right amount of memory and other resources for the target system. We also present some preliminary results for predicting workload performance.

In the remainder of this chapter, we describe our experimental setup, explain our adaptation of the KCCA algorithm as described in Section 2.3 for predicting database performance, present results of predicting many different queries and workloads, discuss our design tradeoffs and some open questions, and lastly review related work.

3.2 Experimental Setup

In this section, we describe the database system configurations and the queries used in our experiments. Subsequent sections reveal our prediction results for queries run on these various configurations.

3.2.1 Database configurations

All of our experiments were run on HP Neoview database systems. HP Neoview is a descendant of the Tandem NonStop [Non, 1989] database system. A successful and competitive commercial decision support database product, Neoview is architecturally representative of commercial databases in the market and is thus a suitable testbed for our experiments. See Figure 3.1.

Our research system is a four-processor machine. Each processor is allotted 262144 physical pages of 16 KB resulting in 4 GB of memory. The machine also has four disks,

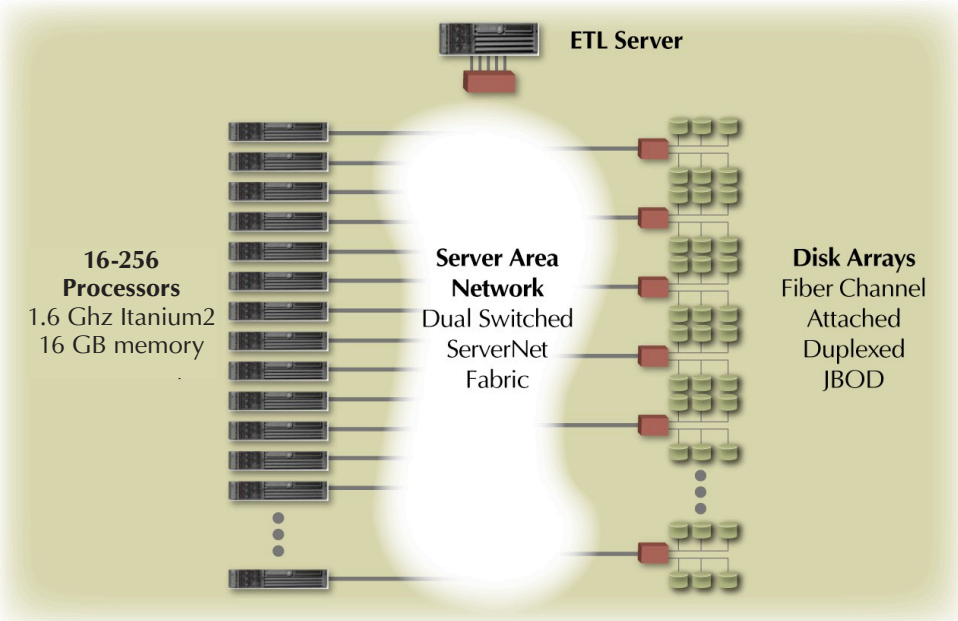

Figure 3.1: Schematic architecture of HP's Neoview database. This image is borrowed from: http://h20195.www2.hp.com/v2/GetPDF.aspx/4AA2-6924ENW.pdf

each with a capacity of 144 GB. Data is partitioned roughly evenly across all four disks. Most query operators run on all four processors; the main exceptions are the operators that compose the final result. We used this system extensively for gathering most of our training and test query performance metrics.

We were also able to get limited access to a 32-processor system with 8 GB of memory per processor and 32 disks of 144 GB each. HP Neoview allows users to configure one or more queries to use a subset of the processors. We therefore ran our queries on four different configurations of this system, using 4, 8, 16, and all 32 processors. In all cases, the data remained partitioned across all 32 disks.

3.2.2 Training and test queries

Our experiments are intended to predict the performance of both short and long-running queries. We therefore decided to categorize queries by their runtimes, so that we could control their mix in our training and test query workloads. Table 3.1 shows the three categories of queries: *feather*, *golf ball*, and *bowling ball*. (We also defined *wrecking ball* queries as queries that were too long to be bowling balls.) We then describe our training and test workloads in terms of the numbers of queries of each type.

While we defined boundaries between the query types based on their elapsed times,

query type	number of instances	elapsed time		
		mean	minimum (hh:mm:ss)	maximum
feather	2807	30 secs	0:0:00.03	0:02:59
golf ball	247	10 mins	0:03:00	0:29:39
bowling ball	48	1 hour	0:30:04	1:54:50

Table 3.1: We created pools of candidate queries, categorized by the elapsed time needed to run each query on the HP Neoview 4 processor system.

these boundaries are arbitrary because there were no clear clusters of queries. They simply correspond to our intuition of "short" and "long." Therefore, they can cause confusion in classifying queries near boundaries, as we show later.

To produce the queries summarized in Table 3.1, we started with the standard decision support benchmark TPC-DS [Othayoth and Poess, 2006]. However, most of the queries we generated from TPC-DS templates (at scale factor 1) were feathers; there were only a few golf balls and no bowling balls. In order to produce longer-running queries, we wrote additional custom templates against the TPC-DS database. We based the new templates on real "problem" queries from a production enterprise data warehouse system that had run for at least four hours before they completed or were killed. Database administrators gave us these queries as examples of queries whose performance they would have liked to predict.

We used these templates, in addition to the original TPC-DS query templates, to generate thousands of queries. We ran the queries in single query mode on the four processor system we were calibrating and then sorted them into query pools based on their elapsed times.

It took significant effort to generate queries with appropriate performance characteristics, and it was particularly difficult to tell a priori whether a given SQL statement would be a feather, golf ball, bowling ball, or wrecking ball. For example, depending on which constants were chosen, the same template could produce queries that completed in three minutes or that ran for over an hour and half.

With training and test set queries created we focused our efforts on identifying and customizing an appropriate machine learning algorithm for performance prediction.

3.3 Addressing Design Challenges of Using KCCA

We adapted the KCCA machine learning algorithm from Chapter 2 to predict query performance. Since KCCA is a generic algorithm, the most challenging aspect of our approach was to formalize the problem of performance prediction and map it onto the data structures and functions used by KCCA.

Our goal was to extract relationships between a query's pre-execution features and post-execution performance metrics, and use these relationships to predict query performance. KCCA was the best match for our goal because it finds correlations between paired datasets and we can leverage these correlations for prediction.

Figure 2.3 summarizes the steps in using KCCA to build a model.

Given N query vectors $\mathbf{x}_k$ and corresponding performance vectors $\mathbf{y}_k$, we form an $N \times N$ matrix K_x whose (i,j)th entry is the kernel evaluation $k_x(\mathbf{x}_i, \mathbf{x}_j)$. We also form an $N \times N$ matrix K_y whose (i,j)th entry is the kernel evaluation $k_y(\mathbf{y}_i, \mathbf{y}_j)$. Recall from Section 2.3 that the KCCA algorithm takes the matrices K_x and K_y and solves the following generalized eigenvector problem:

$$\begin{bmatrix} 0 & K_xK_y \\ K_yK_x & 0 \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} = \lambda \begin{bmatrix} K_xK_x & 0 \\ 0 & K_yK_y \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix}.$$

Intuitively, KCCA finds correlated pairs of clusters in the query vector space and the performance vector space.

In the process of customizing KCCA for query prediction, we needed to make the following three design decisions:

1. How to summarize the pre-execution information about each query into a vector of *query features*, and similarly, how to summarize the performance statistics from executing the query into a vector of *performance features*.

2. How to define a similarity measure between pairs of query vectors so that we can quantify how similar any two queries are, and likewise, how to define a similarity measure between pairs of performance vectors – that is, define the kernel functions – so that we can quantify the similarity of the performance characteristics of any two queries.

3. How to use the output of the KCCA algorithm to predict the performance of new queries.

We next describe these design decisions and evaluate the best choice for each of them. We evaluate the choices for each decision by training a KCCA model with 1027 queries and testing the accuracy of performance predictions for a separate set of 61 queries. The training and test set queries do not overlap. Section 3.2 describes the queries, which contain a mix of feathers, golf balls, and bowling balls. All of them were run and performance statistics were collected from our four processor HP Neoview system.

We use the R^2 metric to compare the accuracy of our predictions. R^2 is calculated using the equation:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(predicted_i - actual_i)^2}{\sum_{i=1}^{N}(actual_i - actual_{mean})^2}$$

An R^2 value close to 1 indicates near-perfect prediction. Negative R^2 values are possible in our experiments since the training set and test set are disjoint. Note that this metric is sensitive to outliers, and in several cases, the R^2 value for predictions improved significantly by removing the top one or two outliers.

By the end of this section, we will identify the adaptation of KCCA that provides the best results for query performance prediction.

3.3.1 Query and performance feature vectors

Many machine learning algorithms applied to systems problems require feature vectors, but there is no simple rule for defining them. Typically, features are chosen using a combination of domain knowledge and intuition.

Selecting features to represent each query's performance metrics was fairly straightforward task: we gave KCCA all of the performance metrics that we could get from the HP Neoview database system when running the query, then allowed it to select the best features with which to predict. The metrics we use for the experiments in this chapter are elapsed time, number of disk I/Os, message count, message bytes, records accessed (the input cardinality of the file_scan operator), and records used (the output cardinality of the file_scan operator). The performance feature vector therefore has six elements. We could easily add other metrics, such as memory used.

Identifying features to describe each query was less trivial. One of our goals was to predict using only data that is available prior to execution. We also wanted to identify features likely to be influential in the behavior of the system because such features are likely to produce more accurate predictions. We evaluated two potential feature vectors: one based on the SQL text of the query and one that condenses information from the query execution plan.

3.3.1.1 SQL feature vector

The first feature vector we tried consisted of statistics on the SQL text of each query. The features for each query were number of nested sub-queries, total number of selection predicates, number of equality selection predicates, number of non-equality selection predicates, total number of join predicates, number of equijoin predicates, number of non-equijoin predicates, number of sort columns, and number of aggregation columns.

Figure 3.2 shows the prediction results for elapsed time. While the SQL text is the most readily available description of a query and parsing it to create the query feature vector is simple, the prediction accuracy was fairly poor for elapsed time as well as all other performance metrics. One reason for the poor accuracy is that often two textually similar queries may have dramatically different performance due simply to different selection predicate constants.

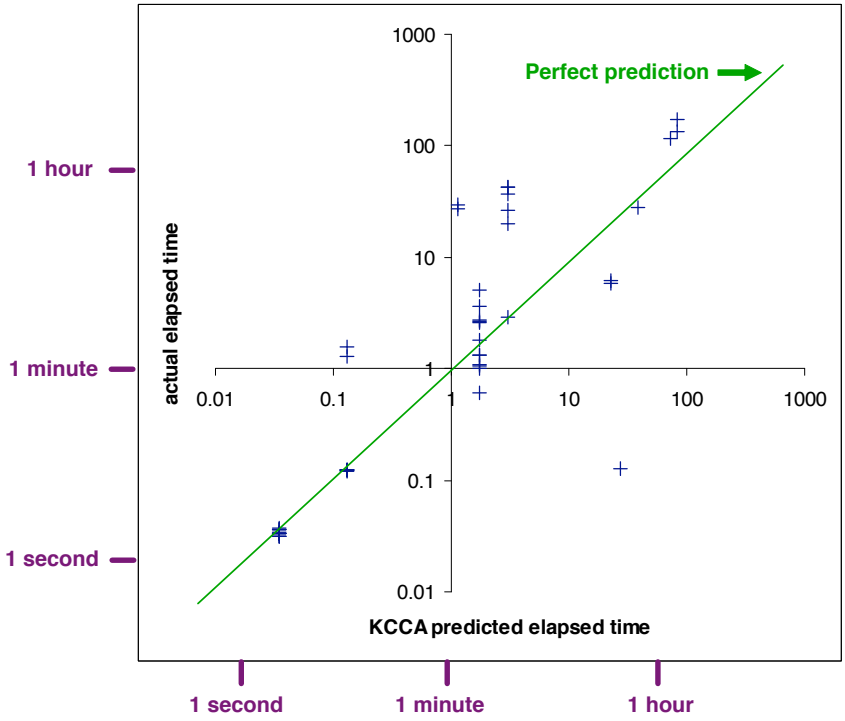

Figure 3.2: KCCA-predicted vs. actual elapsed times for 61 test set queries, using SQL text statistics to construct the query feature vector. We use a log-log scale to accommodate the wide range of query execution times. The R^2 value for our prediction was -0.10, suggesting a very poor model.

3.3.1.2 Query plan feature vector

Before running a query, the database query optimizer produces a query plan consisting of a tree of query operators with estimated cardinalities. We use this query plan instead to create a query feature vector. The query optimizer produces this plan in milliseconds or seconds, so it is not hard to obtain, and we hoped that the cardinality information, in particular, would make it more indicative of query performance. Furthermore, because poor query plan choices contribute to a substantial fraction of telephone support calls, most commercial databases, including Neoview, provide tools that can be configured to simulate a given system and obtain the same query plans as would be produced on the target system.

Our query plan feature vector contains an instance count and cardinality sum for each possible operator. For example, if a sort operator appears twice in a query plan with cardinalities 3000 and 45000, the query plan vector includes a "sort instance count" element containing the value 2 and a "sort cardinality sum" element containing the value 48000.

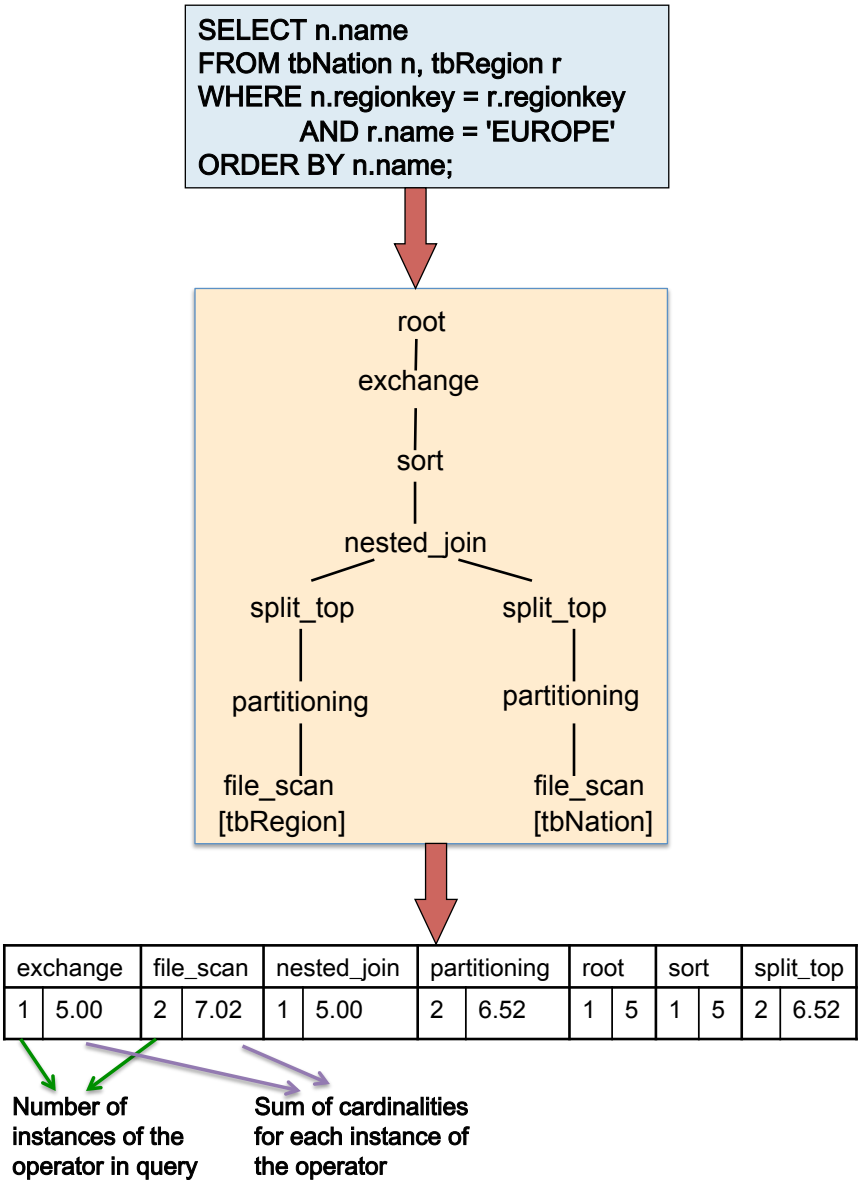

Figure 3.3: Using the query plan to create a query vector: vector elements are query operator instance counts and cardinality sums.

Figure 3.3 shows the query plan and resulting feature vector for a simple query, although it omits operators whose count is 0 for simplicity. The intuition behind this representation is that each operator "bottlenecks" on some particular system resource, such as processor or memory, and the cardinality information roughly captures expected resource consumption.

This feature vector proved to be a better choice than SQL text statistics for predicting the performance of each query; see results in Section 3.4. In all subsequent experiments, we used the query plan feature vector.

3.3.2 Kernel functions

We chose the Gaussian kernel [Shawe-Taylor and Cristianini, 2004] to quantify the similarity between two query feature vectors or two performance feature vectors. This kernel assumes only that the raw feature values follow a simple Gaussian distribution. For a pair of query vectors, $\mathbf{x}_i$ and $\mathbf{x}_j$, we define the Gaussian kernel as follows:

$$k_x(\mathbf{x}_i, \mathbf{x}_j) = \exp\{-\|\mathbf{x}_i - \mathbf{x}_j\|^2/\tau_x\},$$

where $\|\mathbf{x}_i - \mathbf{x}_j\|$ is the Euclidean distance and τ_x is a scale factor. We also define a Gaussian vector on pairs of performance vectors, $\mathbf{y}_i$ and $\mathbf{y}_j$, using a scale factor τ_y. We set the scaling factors τ_x and τ_y to be a fixed fraction of the empirical variance of the norms of the datapoints. Specifically, τ_x for our query feature vectors was 0.1 and τ_y for our performance feature vectors was 0.2. Although our results are quite good with this simple kernel, it may be worth investigating more complicated feature-specific kernels for features whose values follow a different distribution.

3.3.3 Predicting performance from a KCCA model

KCCA produces correlated projections of the query and performance features. However, it is not obvious how to leverage these projections for prediction. Figure 3.4 shows the three steps we use to predict the performance of a new query from the query projection and performance projection in the KCCA model. First, we create a query feature vector for the new query and use the KCCA model to find its coordinates on the query projection $K_x \times A$. Since we have not executed this query, we do not have a corresponding coordinate on the performance projection $K_y \times B$. We infer the query's coordinates on the performance projection by using the k nearest neighbors in the query projection. This inference step is enabled because KCCA projects the raw data onto dimensions of maximal correlation, thereby collocating points on the query and performance projections. We evaluate choices for k in Section 3.3.3.2. Finally, we must map from the performance projection back to the metrics we want to predict.

As described in Section 2.3.4, finding a reverse mapping from the feature space back to the input space is a known hard problem, both because of the complexity of the mapping algorithm and also because the dimensionality of the feature space can be much higher or lower than the input space based on the goal of the transformation function.

Accurate predictions of a previously unseen query's performance depend on identifying

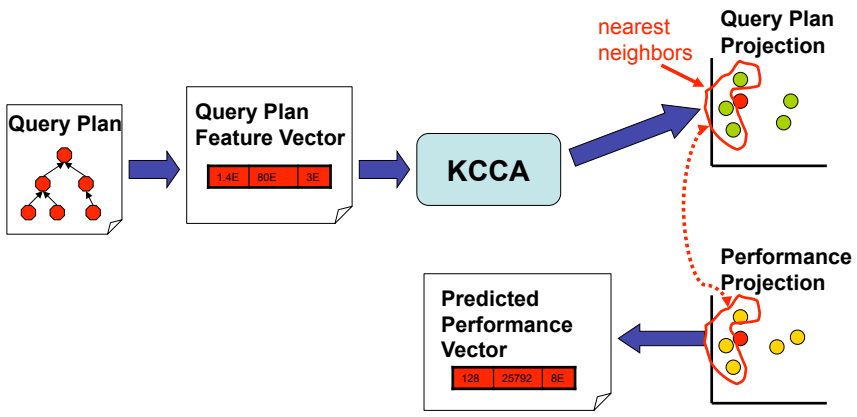

Figure 3.4: Testing: KCCA projects a new query's feature vector, then looks up its neighbors from the performance projection and uses their performance vectors to derive the new query's predicted performance vector.

the query's nearest neighbors, or at least neighbors that are "similar enough." We address three important issues in using nearest neighbors to predict a test query's performance:

1. What is the metric to determine neighbors and their nearness?
2. How many neighbors do we need to consider when calculating our prediction?
3. How do we weigh the different neighbors in our prediction?

We consider each issue in turn.

3.3.3.1 What makes a neighbor "nearest"?

We considered both Euclidean distance and cosine distance as a metric to determine the nearest neighbors. Since we are projecting our vectors onto a higher dimensional subspace, it seemed useful to consider the directional nearness. While Euclidean distance captures the magnitude-wise closest neighbors, cosine distance captures direction-wise nearest neighbors. Table 3.2 compares the cosine distance function with the Euclidean distance function for computing nearest neighbors in the query projection. Using Euclidean distance yielded better prediction accuracy than using cosine distance, as shown by the consistently higher R^2 value, so we use Euclidean distance to determine nearest neighbors in our experiments.

3.3.3.2 How many neighbors?

Table 3.3 shows the result of varying the number of neighbors to use to predict query performance. $k = 3$ seems to work as well as $k = 4$ or 5. The difference between the R^2 values

Metric	Euclidean Distance	Cosine Distance
Elapsed Time	0.55	0.51
Records Accessed	0.70	-0.09
Records Used	0.98	0.19
Disk I/O	-0.06	-0.07
Message Count	0.35	0.24
Message Bytes	0.94	0.68

Table 3.2: Comparison of R^2 values for using Euclidean Distance and Cosine Distance to identify nearest neighbors. Euclidean Distance has better prediction accuracy.

Metric	k=3	k=4	k=5	k=6	k=7
Elapsed Time	0.55	0.59	0.57	0.61	0.56
Records Accessed	0.70	0.63	0.67	0.70	0.71
Records Used	0.98	0.98	0.98	0.98	0.98
Disk I/O	-0.06	-0.01	-0.003	-0.01	-0.01
Message Count	0.35	0.34	0.34	0.31	0.31
Message Bytes	0.94	0.94	0.94	0.94	0.94

Table 3.3: Comparison of R^2 values produced when varying the number of neighbors. Negligible difference is noted between the various choices.

is negligible for most of the metrics. The thousands of feather queries listed in Table 3.1 all fit in memory, so disk I/Os were 0 for most queries. Thus, the R^2 value appears to be very poor, perhaps because the number of disk I/Os is a function of the amount of memory and whether the tables can reside in-memory, which is more of a configuration issue than a query feature. $k = 3$ performs better for predicting records accessed relative to $k = 4$, and $k = 4$ performs better for predicting elapsed time compared to $k = 3$. We therefore chose $k = 3$ with the intuition that for queries with few close neighbors, a smaller value of k would be better.

3.3.3.3 How do we map from the neighbors to performance metrics?

Another consideration in our methodology was how to map from the set of k neighbors to a vector of predicted performance metrics. We decided to combine the performance elements of the neighbors and tried different ways to weight the neighbors' values: equal weight for all three neighbors, a 3:2:1 ratio for the weight of the three neighbors in order of nearness, and weight proportional to the magnitude of distance from the test query feature vector. Table 3.4 summarizes results for all three scenarios. None of the three weighting functions

Metric	Equal	3:2:1 Ratio	Distance Ratio
Elapsed Time	0.55	0.53	0.49
Records Accessed	0.70	0.83	0.35
Records Used	0.98	0.98	0.98
Disk I/O	-0.06	-0.09	-0.04
Message Count	0.35	0.37	0.35
Message Bytes	0.94	0.94	0.94

Table 3.4: Comparison of R^2 values produced when using various relative weights for neighbors

yielded better predictions consistently for all of the metrics. We therefore choose the simplest function: weight all neighbors equally. Our prediction for each performance metric is the average of the three neighbors' metrics.

3.4 Results

After finding a good adaptation of KCCA for query prediction, we evaluated its performance prediction accuracy for multiple query sets and system configurations. We first present results from training our model on the four-node HP Neoview system using queries to TPC-DS tables and predict performance for a different set of queries to TPC-DS as well as queries to a customer database. We then validated our approach by training a model for and predicting performance of queries on various configurations of a 32-node Neoview system. Lastly, we discuss how our predictions compare to the query optimizer's cost estimates. We note that a substantial subset of results in this section were previously presented as a conference publication [Ganapathi *et al.*, 2009c].

3.4.1 Performance prediction using KCCA on a 4-node system

3.4.1.1 Experiment 1: Train model with realistic mix of query types

The first experiment shows our results from using 1027 queries for our training set, including 30 bowling balls, 230 golf balls and the 767 feathers. The test set includes 61 queries (45 feathers, 7 golf balls and 9 bowling balls) that were not included in our training set.

Figure 3.5 compares our predictions to actual elapsed times for a range of queries. As illustrated by the closeness of nearly all of the points to the diagonal line (perfect prediction), our predictions were quite accurate. Note that six of the nine bowling balls in the test set were run on a more recent version of the operating system than our training dataset. The accuracy of our predictions for those queries was not as good as accuracy for bowling ball

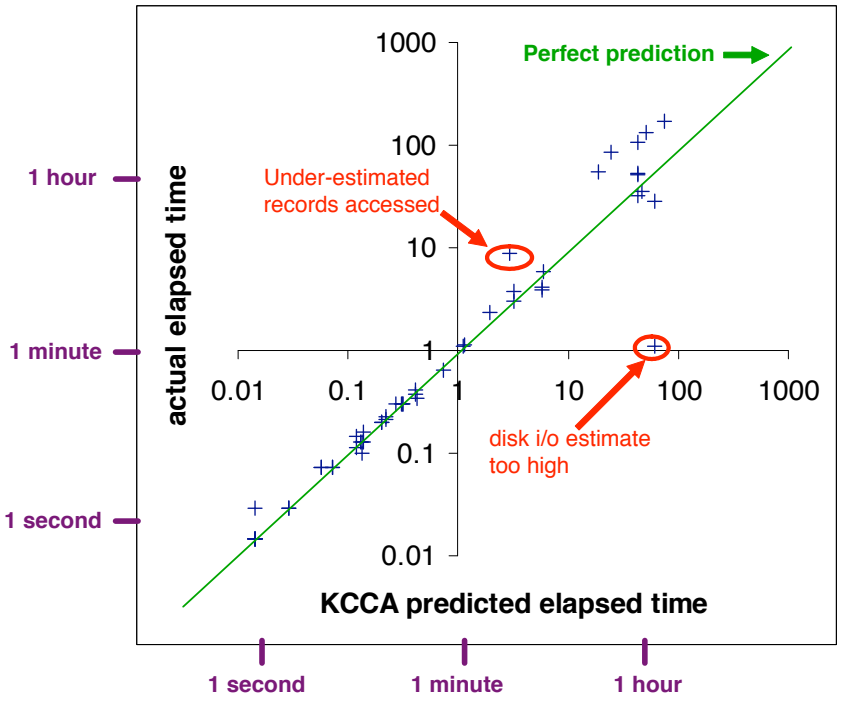

Figure 3.5: KCCA-predicted vs. actual elapsed times for 61 test queries. We use a log-log scale to accommodate the wide range of query execution times from milliseconds to hours. The R^2 value for our prediction was 0.55 due to the presence of a few outliers (as marked in the graph). Removing the top two outliers increased the R^2 value to 0.95.

queries run on the same version of the operating system as the original training set. When the two circled outliers and the more recent bowling balls were eliminated, then the R^2 value jumps to 0.95.

We show similar graphs for records used in Figure 3.6 and message counts in Figure 3.7; other metrics are omitted for space reasons. The simultaneous predictability of multiple performance metrics using our approach enabled us to better understand inaccurate predictions. For example, for one prediction in which elapsed time was much too high, we had greatly overpredicted the disk I/Os. This error is likely due to our parallel database's methods of cardinality estimation. When we underpredicted elapsed time by a factor of two, it was due to under-predicting the number of records accessed by a factor of three.

We are often confronted with the question, "Why not try a simpler machine learning technique for prediction?" Regression is the simplest machine learning technique that has a sound statistical basis. While we knew that single variable regression would not work, we decided to try multivariate regression to predict each performance metric of interest.

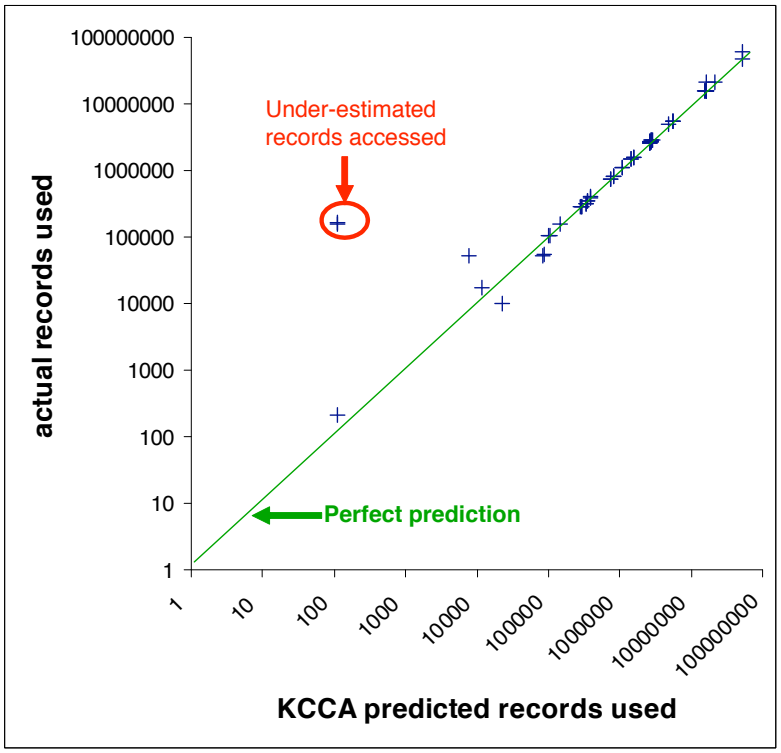

Figure 3.6: KCCA-predicted vs. actual records used. We use a log-log scale to accommodate wide variance in the number of records used. The R^2 value for our prediction was 0.98. (R^2 value close to 1 implies near-perfect prediction).

We defined independent variables $\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_n$ for different query plan features, such as join operator counts and cardinality sums. We use the same features later in the example in Figure 3.3. Each performance metric was considered as a separate dependent variable y. The goal of regression is to solve the equation $a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + \ldots + a_n\mathbf{x}_n = y$ for the coefficients a_i.

Figures 3.8 and 3.9, compare the predicted and actual values for elapsed time and records used respectively. The regression models do a poor job predicting these metrics of interest and results for predicting other metrics were equally poor. Many of the predictions are orders of magnitude off from the actual value of these metrics for each query. Furthermore, the regression for elapsed time predicts that several of the queries will complete in a negative amount of time; for example, -82 seconds!

One interesting fact we noticed is that the regression equations did not use all of the independent variables. For example, even though the *hashgroupby* operator had actual cardinalities greater than zero for many queries, regression gave it a coefficient of zero. However,

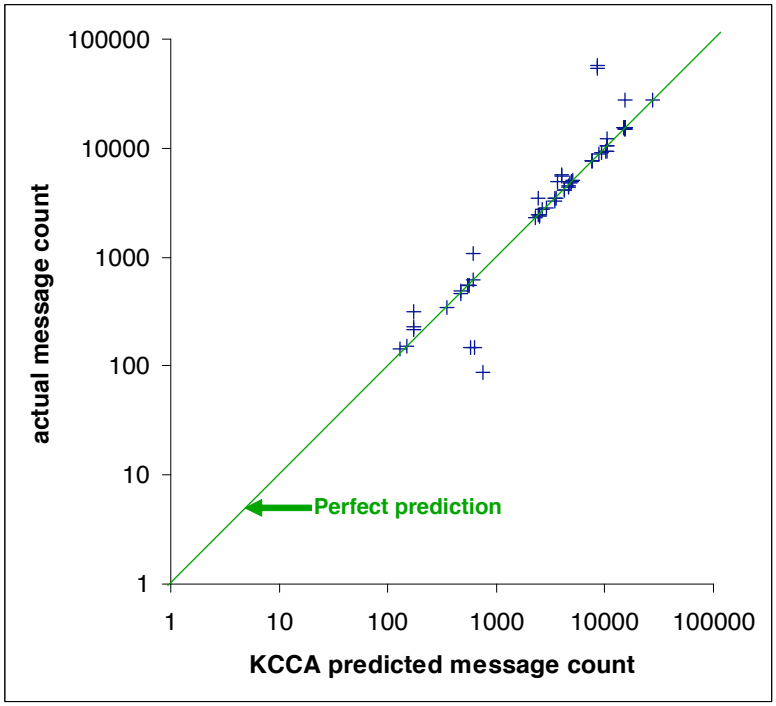

Figure 3.7: KCCA-predicted vs. actual message count. We use a log-log scale to accommodate wide variance in the message count for each query. The R^2 value for our prediction was 0.35 due to visible outliers.

the features discarded were not consistent across each of the dependent variables considered. Thus, it is important to keep all query features when building each model. Furthermore, since each dependent variable is predicted from a different set of chosen features, it is difficult to unify the various regression curves into a single prediction model. This fact makes it difficult to group queries with respect to all the different performance metrics simultaneously.

3.4.1.2 Experiment 2: Train model with 30 queries of each type

For our second experiment, to balance the training set with equal numbers of feathers, golf balls, and bowling balls, we randomly sampled 30 golf balls and 30 feathers to include in the training set and predicted performance of the same set of 61 test set queries as in Experiment 1. Figure 3.10 compares predicted and actual elapsed times for this experiment. Our predictions were not as accurate as in our previous experiment, which included a larger training set of 1027 queries. We considered constructing more bowling balls to add to the training set to have a few hundred queries of each type included in the model. However, these

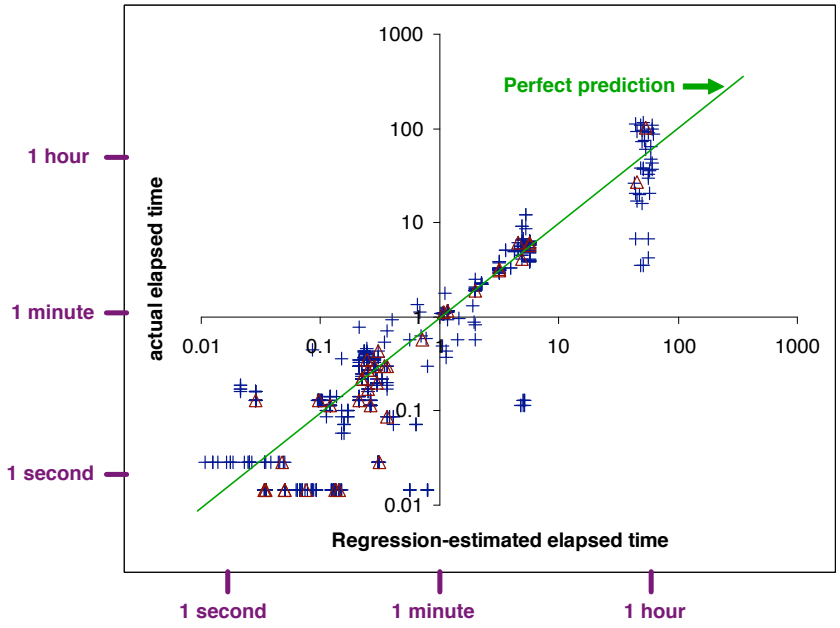

Figure 3.8: Regression-predicted vs. actual elapsed times for 1027 training and 61 test queries. The graph is plotted on a log-log scale to account for the wide range of query runtimes. 176 datapoints are not included in the plot as their predicted times were negative numbers. The triangles indicate datapoints that were in our test set (not used to build the regression model).

proportions are unrealistic in business intelligence workloads. Our results for this experiment indicate that it is better to build a model with a realistic workload mix.

3.4.1.3 Experiment 3: Two-model prediction with query type-specific models

So far, we have described a single model for predicting query performance. We also tried a two-model approach using the same feature vectors. We use the neighbors from the first model to predict simply whether the query is a "feather," "golf ball," or "bowling ball." For example, if a test query's neighbors are two feathers and a golf ball, the query is classified as a feather. We then predict its performance using a second model that was trained *only* on "feather" (or "golf ball" or "bowling ball") queries. Since we had many fewer examples of golf ball and bowling ball queries in our training sets, we thought this approach might do better for predicting their performance.

Figure 3.11 shows our results for predicting elapsed time of queries using this two-model approach. Our predictions were more accurate than in Experiment 1, as evidenced by the fewer number of outliers with respect to the perfect prediction line. In comparing Figure 3.5

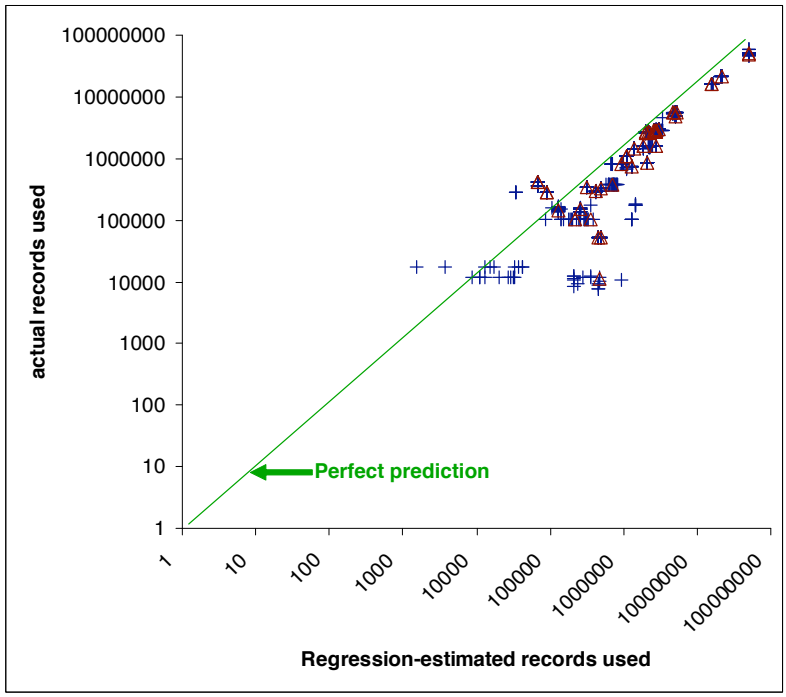

Figure 3.9: Regression-predicted vs. actual records used for 1027 training and 61 test queries. Note that 105 datapoints had negative predicted values going as low as -1.18 million records. The triangles indicate datapoints that were in our test set (not used to build the regression model).

and Figure 3.11, we notice a few instances where the one-model prediction was more accurate than the two-model prediction. For the most part, this result was because the test query was too close to the temporal threshold separating feathers from golf balls and forcing it into one category made the prediction marginally worse. The R^2 statistics for this experiment favor using two-model prediction to single-model prediction.

3.4.1.4 Experiment 4: Training and testing on queries to different data tables

We also evaluated how well we predict performance when the training set queries and test set queries use different schemas and databases. Figure 3.12 shows our one-model and two-model KCCA predictions for a set of 45 queries to a customer's database where the training was done on queries to the TPC-DS tables. When compared to actual elapsed time, the two-model KCCA prediction method performed better than the one-model prediction method. Most of the one-model KCCA-predictions were one to three orders of magnitude longer than the actual elapsed times. One caveat of this experiment is that the customer queries we had access to were all extremely short-running (mini-feathers). As a result, even when our

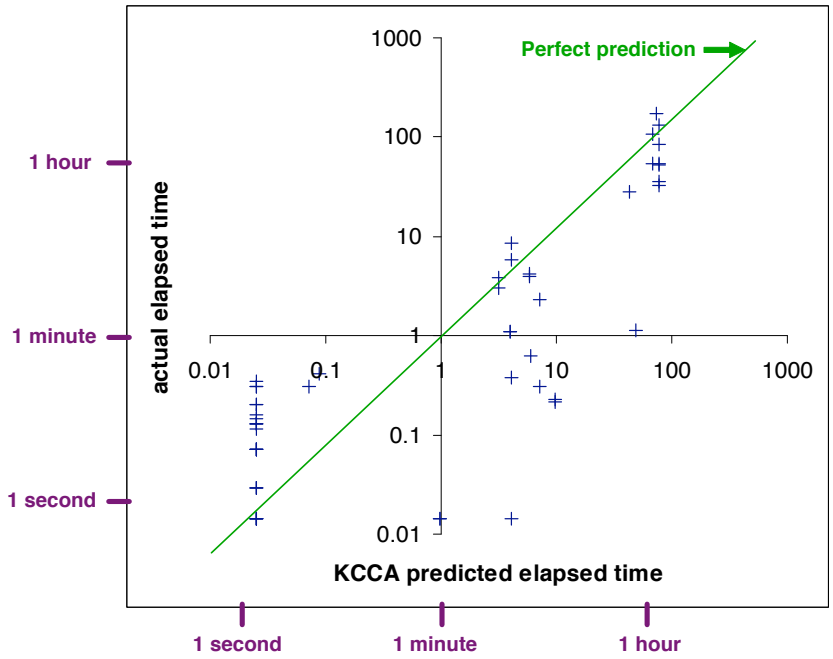

Figure 3.10: KCCA-predicted vs. actual elapsed times. The axes are log-log scale to accommodate wide variance of elapsed times in the 61 queries of our test set.

predicted elapsed times were within seconds of the actual elapsed times, the prediction error is very large relative to the size of the query. This experiment would have benefited from access to longer-running queries for the same customer as well as a variety of queries from other customers. We learn from this experiment that it is difficult to predict performance for a test set of queries to a different set of tables than the training set.

3.4.2 Prediction on a 32-node system

We further validated our prediction technique by training and testing on more and larger system configurations. We had limited access to a 32-node parallel database system, which we could configure to process queries using only subsets of nodes.

We reran the TPC-DS scale factor 1 queries on the 32-node system and used 917 of these queries for training the model and 183 queries for testing the model. Table 3.5 shows the R^2 values for each metric predicted for each of four configurations of the 32-node system. The configuration was varied by using only a subset of the nodes on the system, thus reducing the number of processors and consequent memory available for use. The training set of queries was rerun on each configuration.

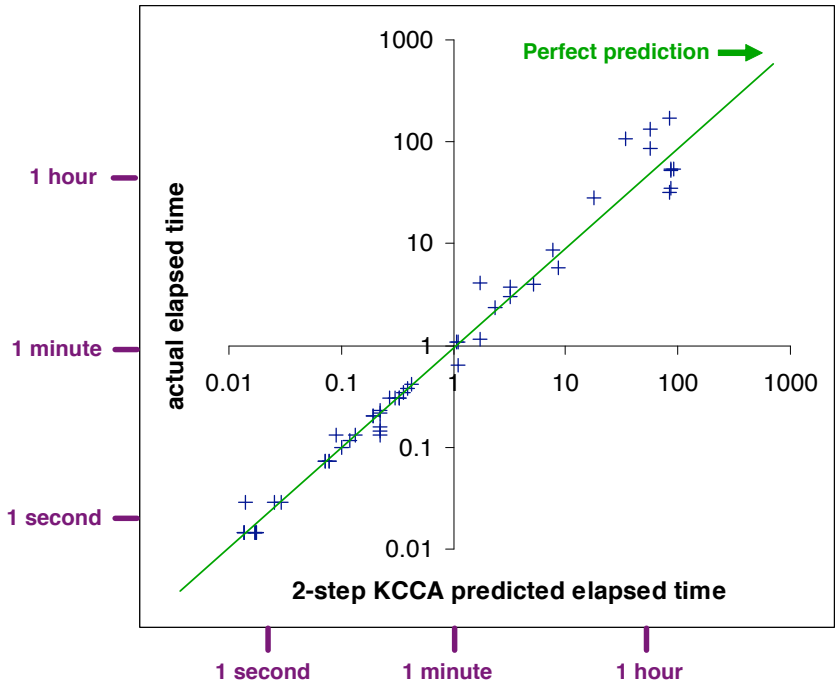

Figure 3.11: Two-model KCCA-predicted elapsed time vs. actual elapsed times for the 61 queries in our test set. The first model classifies the query as a feather, golf ball or bowling ball; the second model uses a query type-specific model for prediction. A log-log scale was used to accommodate wide variance in query elapsed times. The R^2 value for our prediction was 0.82.

Metric	4 nodes	8 nodes	16 nodes	32 nodes
Elapsed Time	0.89	0.94	0.83	0.68
Records Accessed	1.00	1.00	1.00	0.57
Records Used	0.99	1.00	1.00	0.99
Disk I/O	0.72	-0.23	Null	Null
Message Count	0.99	0.99	0.99	0.94
Message Bytes	0.99	0.99	1.00	0.97

Table 3.5: Comparison of R^2 values produced for each metric on various configurations of the 32-node system (we show the number of nodes used for each configuration). Null values for Disk I/O reflect 0 disk I/Os required by the queries due to the large amount of memory available on the larger configuration.

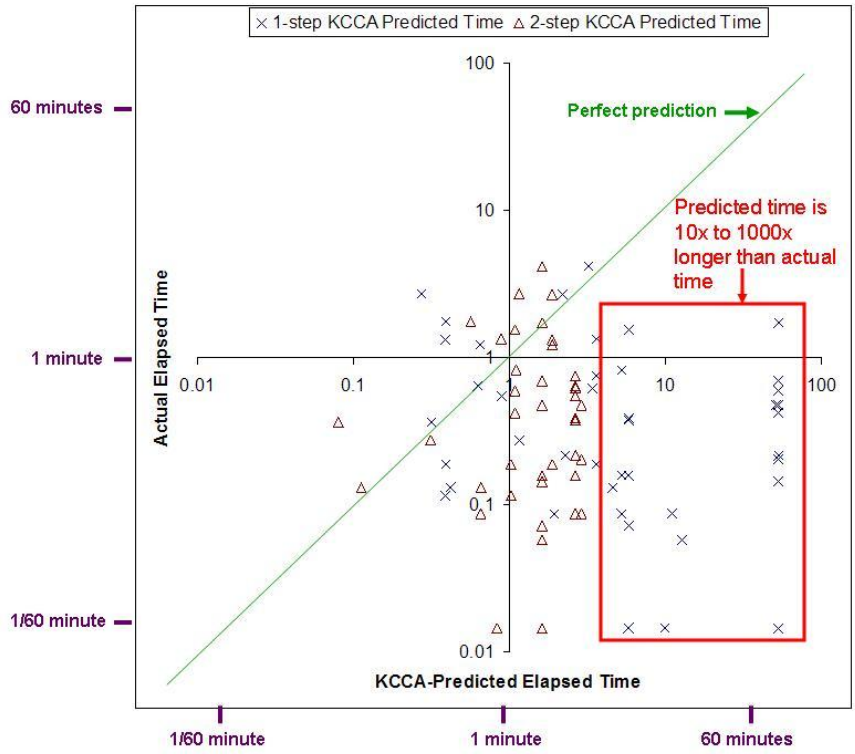

Figure 3.12: A comparison of one-model KCCA-prediction, two-model KCCA-prediction and actual value of elapsed times for a test set of customer queries.

Our results are encouraging and demonstrate that we can effectively predict the performance of TPC-DS queries on various system configurations. We made several observations in the course of re-running our queries on various configurations of the 32-node system.

First, while we had characterized our queries as feathers, golf balls, and bowling balls on the 4-node system, when the same queries were run on the 32-node system, they all ran very quickly. The queries were short-running regardless of the number of nodes used on the 32-node system. We would have liked to run queries that took longer than 10 minutes on the 32-node system, but were unable due to restrictions on our access to this machine.

Another noteworthy point is that the scenario in which only 4 of the 32 nodes were used showed reasonable prediction accuracy of Disk I/Os whereas all other configurations of the 32-node system seemed to exclude disk I/Os in the model. We note that the likely cause for this special case is that the number of disk I/Os for each query was significantly higher for the 4-node configuration where the amount of available memory was not enough to hold many of the TPC-DS tables in memory. Since increasing the number of nodes used also proportionally increases the amount of memory, the 8, 16 and 32-node configurations had

very few queries that required disk I/Os.

Lastly, the query execution plans for the 4-node system were different from the same queries' plans on the 32-node system. Also, plans for various configurations of the 32-node system differed from one another. This is because although we limited the number of processors used to process queries, we did not change the physical layout of the data on disk, so more than 4-nodes were involved in the data access operations. All plans produced on the 32-node system access and combine results from more disk drives than those on the 4-node system. Plans that limit the number of nodes used for processing on the 32-node system have fewer resources (especially memory) than those that run with all 32 nodes.

3.4.3 Multi-query workload prediction on a simulator

While it is difficult to accurately predict the performance of individual queries running in isolation, it even more difficult to predict the performance of a set of queries running concurrently. Often, running a query concurrently with other queries delays its completion due to conflicting resource requirements of the concurrently running queries.

To explore multi-query workloads without waiting hours for each workload to execute, we used a Neoview simulator [Krompass *et al.*, 2007]. The simulator calculates query and workload costs using estimated cardinalities and prior knowledge of dominant resource requirements for each operator. Since the simulator has been successfully used for studying workload management policies, we believe it would be a suitable alternative to running query workloads on a real system.

Our final experiment exhibits our attempt at predicting the performance of multi-query workloads. We built our workload feature vector as a sum of the processor, disk and message costs of each query running in isolation on the simulator. We then applied the KCCA methodology on the workload feature vectors and workload performance metrics. Our training set contained 50 distinct multi-query workloads, each with a different mix of feathers, golf balls, and bowling balls. Our test set contained 10 multi-query workloads, which were not in the training set.

Figure 3.13 compares predicted and actual workload elapsed times on a simulator configured to run up to four queries concurrently, that is multi-programming level (MPL) = 4. It is common practice in the business intelligence community to estimate a concurrent workload's elapsed time using the elapsed time for a sequential run of the queries in the workload, that is, sum of the elapsed times of the queries contained in the workload. Figure 3.13 also shows the elapsed times for our ten test workloads running in sequential mode, where MPL = 1.

Our prediction of workload elapsed time is not always closer to actual simulator runs of the workload with multi-programming level = 4 compared to the sequential execution approximation determined by multi-programming level = 1 runs. However, we are able to deduce that for the given set of test workloads, we either overpredict elapsed times, or when we underpredict, we perform better than the sequential execution approximation of elapsed

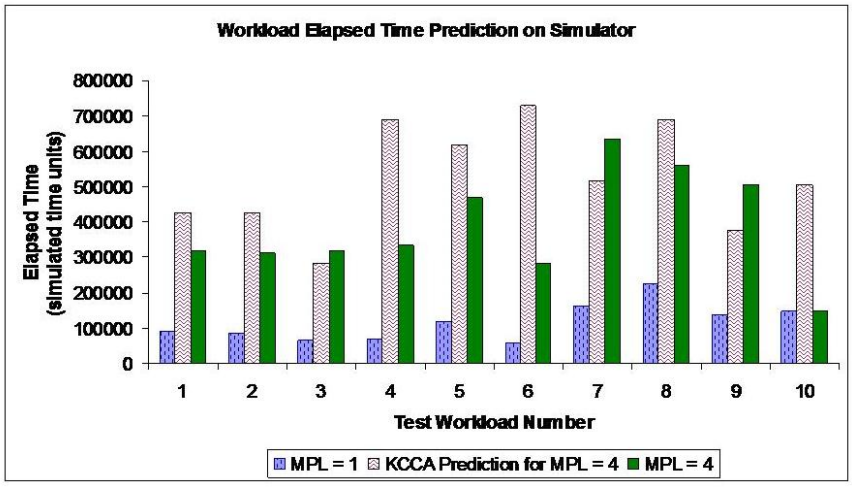

Figure 3.13: Comparison of KCCA-predicted elapsed time for workloads running at multi-programming level = 4, elapsed times for the same workloads running at multi-programming level = 1 on the simulator, and elapsed time for test workloads run at multi-programming level = 4 in the simulator.

time. As most database administrators prefer an upperbound on time rather than a lower-bound, our predictions seem more relevant than the sequential execution approximation, which consistently underestimates time for these 10 workloads.

3.5 Lessons and Limitations

Our methodology has been successfully adopted for Neoview workload management and is currently being incorporated into the production system. In this section, we address some of the open questions about our results and its applicability in production environments.

3.5.1 Comparing our predictions to query optimizer cost estimates

Our predictions and the query optimizer's estimates serve different purposes. The optimizer's estimates do *not* predict actual performance characteristics. The query optimizer's estimates are used to select the most efficient way to execute a query. The optimizer needs to produce a cost estimate for every query plan it considers for a given query. In contrast, our predictions are geared towards estimating query resource requirements given a particular query execution plan.

Figures 3.14 and 3.15 compare the 4-node system's query optimizer cost estimates to the actual elapsed times for running the 61 test queries. Since the optimzer's cost units do not

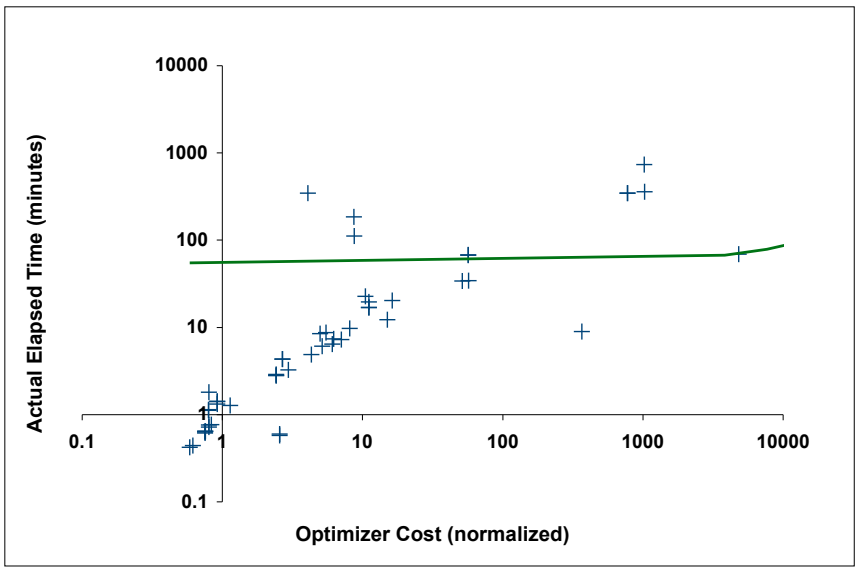

Figure 3.14: Optimizer-predicted costs vs. actual elapsed times for a set of queries, the same test queries used later for Experiment 1 in Section 3.4. The optimizer costs and measured times came from Neoview's commercial query optimizer. The horizontal line marks a computed linear best-fit line.

directly correspond to time units, we draw a linear best fit line in Figure 3.14. Evidently, the relationship between the optimizer-predicted cost and actual elapsed time is not linear; therefore we draw a hypothetical perfect prediction line in Figure 3.15. We generated similar graphs for other optimizer metrics, such as records used, which were consistent with the one shown: the optimizer's estimates do not correspond to actual resource usage for many queries, especially ones with elapsed times of over a minute. When compared to our results in Figure 3.11, it is apparent that our model's predictions are more accurate than the optimizer's query cost estimation.

However, our predictions are created in completely different circumstances than the query optimizer's. First, we have the luxury of being able to train to a *specific configuration*, whereas the optimizer's cost models must accommodate all possible configurations. Second, we can spend orders of magnitude more time calculating our estimates than the optimizer. The optimizer must produce multiple estimates in milliseconds. Third, we have access to historical information about the actual performance metrics for various plans. By design, the optimizer has no access to this information. In fact, our performance estimates are actually based upon the optimizer's cost estimates, since we use the optimizer's cost estimates plus historical performance data in our feature vectors.

We believe our predictions can be complementary to the query optimizer. Given static

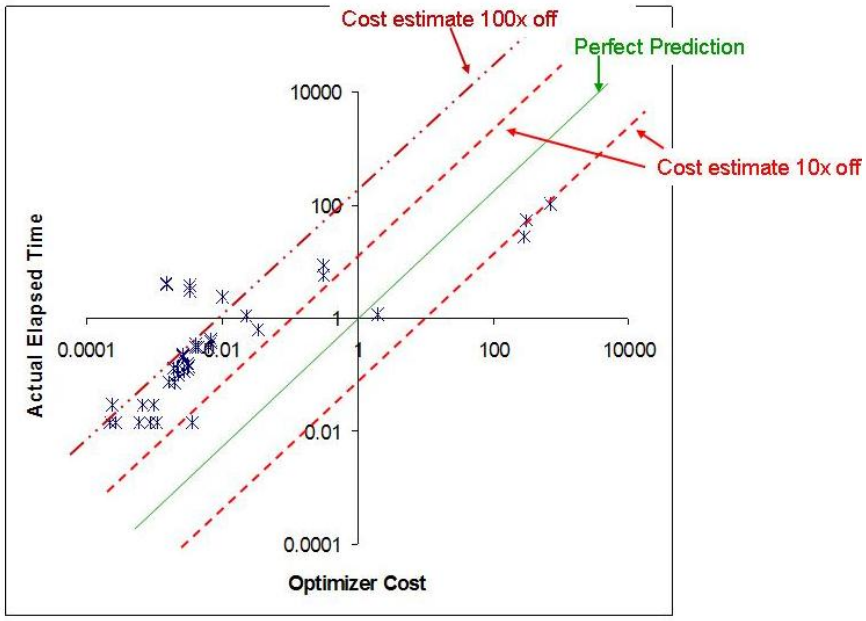

Figure 3.15: Optimizer-predicted costs vs. actual elapsed times for a set of queries, the same test queries used later for Experiment 1 in Section 3.4. The optimizer costs and measured times came from Neoview's commercial query optimizer.

cost functions, the optimizer can use our prediction results to adapt its rules and thresholds to the production environment. It can develop custom cost functions that optimize query plans in response to patterns in the workloads. We believe this is a promising new area for query optimization research.

3.5.2 Why do erroneous cardinality estimates produce good prediction results?

It is common for decision support databases to maintain inaccurate table statistics due to the volume and frequency of churn in data. We have empirically observed several orders of magnitude difference between estimated and actual cardinalities for various query operators. The use of kernel functions in our prediction methodology inherently adjusts for these cardinality estimation errors. To demonstrate by example, if $query_1$ has a join cardinality sum of $3000 + joinEstimationError$ and $query_2$ has a join cardinality sum of $2000 + joinEstimationError$, the kernel function calculates the difference between join cardinality sums for $query_1$ and $query_2$, i.e. $(3000 + joinEstimationError) - (2000 + joinEstimationError)$, consequently eliminating the $joinEstimationError$ component altogether. The fact that estimation errors

tend to be operator-dependent and table-specific explains why our experiments with training and test queries to the same dataset perform better than experiments with test set queries to a different dataset than the training set queries.

3.5.3 How can we improve multi-query workload predictions?

Our preliminary multi-query workload prediction results provide reasonable bounds on execution times on a simulator. However, we believe a variant of our methodology would provide more accurate predictions. Each query is further deconstructed into fragments by the query execution engine. A query fragment is the granularity at which node scheduling (and consequently resource scheduling) decisions are made. Instead of representing a workload's feature vectors as the sum of features of its constituent queries, a more insightful representation would treat each workload as a sum of its query fragments. Such a representation would account for overhead imposed by query fragments contending for operators implemented on a specific subset of the database's nodes, and thus would mask potential wait times imposed by concurrent query ordering and scheduling decisions.

3.5.4 How can our results inform database development?

We believe our prediction techniques can help with open questions that might be of interest to a database engine development team. For example, the development team would be interested in knowing which query operators, and in what conditions, have the greatest impact on query performance. However, in our model, when the raw query plan data is projected by KCCA, the dimensions of the projection do not necessarily correspond to features of the raw data; it is computationally difficult to invert KCCA's projection to determine which features are considered for each dimension. As an alternative to estimating the role of each feature, we compared the similarity of each feature of a test query with the corresponding features of its nearest neighbors. At a cursory glance, it appears that the counts and cardinalities of the join operators (e.g., nested loops join) contribute the most to our performance prediction model.

3.5.5 How can our results benefit database customers?

We believe our prediction methodology can directly improve customer experience by boosting the quality of system sizing, capacity planning, and workload management. Since most database vendors support a limited number of configurations, it is feasible to build a model per supported configuration, predict performance for customer workload on each supported configuration, and make recommendations for system size based on price-performance tradeoffs. Capacity planning can be more efficient as models can be built using representative samples of current workload, and predictions of performance of expected future workload

can be used to determine bottleneck resources that could be expanded. Lastly, workload management can benefit from the insight of query resource requirements. The workload manager can use resource predictions to decide if a query should be scheduled to run immediately (there is no resource contention with currently running queries) or placed on hold until potential conflicting queries complete. The prediction framework is currently being beta tested for use as part of a dashboard for Neoview's workload manager.

3.5.6 Can we predict anomalous queries?

As is common with most machine learning techniques, our predictions are limited to the range of performance metrics reflected by our training set, and our prediction results are only as good as the proximity of our test data to its neighbors in the projected space. Initial results indicate that we can use Euclidean distance from the three neighbors as a measure of confidence – and that we can thus identify queries whose performance predictions may be less accurate. This approach could potentially identify anomalous queries. For example, the anomalous bowling balls in Figure 3.5 were not as close to their neighbors as the better-predicted ones.

3.5.7 How often should the KCCA model be refreshed?

Given a KCCA model, prediction of a single query can be done in under a second, which makes it practical for queries that take many minutes or hours to run, but not for all queries. Training, on the other hand, takes minutes to hours because each training set datapoint is compared to every other training set datapoint, and computing the dimensions of correlation takes exponential time with respect to the number of datapoints. While it feasible to use a dedicated machine for training the model, the frequency of retraining depends on the churn rate and variability of a customer's workload. A reasonable compromise would test against both a stale model built from a substantial dataset and a more recent model built quickly from a smaller and more recent dataset. The most suited prediction result can be based on the distance of the queries from their nearest neighbors.

This limitation presents a research opportunity to investigate techniques to make KCCA more amenable to continuous retraining (e.g., to reflect recently executed queries). Such an enhancement would allow us to maintain a sliding training set of data with a larger emphasis on more recently executed queries.

3.6 Related Work

According to experts we interviewed who perform system sizing and capacity planning for major commercial database systems including Oracle, Teradata, HP Neoview, IBM DB2, and

Microsoft SQL Server, current state-of-the-art approaches to workload prediction are manual, time consuming, often inaccurate, and seldom validated. Current research approaches for single queries focus on predicting a single metric (typically elapsed time). Often, the techniques estimate the percentage of work done or relative cost of a query rather than producing actual performance metrics. Furthermore, several existing solutions require access to runtime performance statistics such as tuples processed. As a result, a significant amount of instrumentation is required to the core engine to accommodate usable statistics. In contrast, we address the problem of predicting multiple characteristics of query performance *prior* to run-time.

While the use of statistical learning techniques has been prevalent in database research, their full potential has not been explored for more challenging customer-facing research problems until recently. Below, we discuss relevant work in traditional database research areas in which performance analysis and statistical techniques are essential components.

3.6.1 Query Planning and Optimization

The primary goal of the database query optimizer is to choose a good query plan. As part of evaluating different plans, the optimizer produces a rough cost estimate for each plan. However, the units used by most optimizers do not map easily onto time units. Furthermore, the optimizer has little time to produce a cost estimate and its logic must generalize across multiple configurations. Query optimization has been proven to be an NP-complete problem [Ibaraki and Kameda, 1984], and is typically addressed using a series of thresholds and cost functions, randomized techniques [Swami and Iyer, 1993; Galindo-Legaria *et al.*, 1994], rule-based approaches [Graefe and McKenna, 1993], heuristics [Yoo and Lafortune, 1989; Shekita *et al.*, 1993], or a hybrid of these various approaches [Ioannidis, 1996]. These estimates are often wildly inaccurate, but as long as they guide the optimizer to a good plan, accuracy is not needed. While we rely on the same (often inaccurate) cardinality estimates as the optimizer for input to our prediction model, our model bases its predictions on the relative similarity of the cardinalities for different queries, rather than their absolute values.

A few papers use machine learning to predict a relative cost estimate for use by the query optimizer. In their work on the COMET statistical learning approach to cost estimation, [Zhang *et al.*, 2005a] use transform regression to produce a self-tuning cost model for XML queries. Because they can efficiently incorporate new training data into an existing model, their system can adapt to changing workloads, a very useful feature that we have not yet addressed. COMET, however, focuses on producing a single cost value for comparing query plans to each other as opposed to a metric that could be used to predict resource usage or runtime. Similarly, IBM's LEO learning optimizer compares the query optimizer's estimates with actuals at each step in a query execution plan, and uses these comparisons from previously executed queries to repair incorrect cardinality estimates and statistics [Stillger *et al.*, 2001; Markl and Lohman, 2002]. Like COMET, LEO focuses on producing a better

cost estimate for use by the query optimizer, as opposed to attempting to predict actual resource usage or runtime. Although a query optimizer enhanced with LEO can be used to produce relative cost estimates prior to executing a query, it does require instrumentation of the underlying database system to monitor actual cost values. Also, LEO itself does not produce any estimates; its value is in repairing errors in the statistics underlying the query optimizer's estimates.

3.6.2 Query Progress Indication

Query progress indicators attempt to estimate a running query's degree of completion. Existing approaches to this problem all assume that the progress indicator has complete visibility into the number of tuples already processed by each query operator [Chaudhuri *et al.*, 2004; Luo *et al.*, 2004; Luo *et al.*, 2005; Chaudhuri *et al.*, 2005]. Such operator-level information can be prohibitively expensive to obtain, especially when multiple queries are executing simultaneously.

Luo and his peers [Luo *et al.*, 2006] leverage an existing progress indicator to estimate the remaining execution time for a running query (based on how long it has taken so far) in the presence of concurrent queries. They do not address the problem of predicting the performance characteristics of a query that has not yet begun execution.

3.6.3 Workload Characterization

A number of papers [Yu *et al.*, 1992; Lo *et al.*, 1998; Keeton *et al.*, 1998] discuss how to characterize database workloads with an eye towards validating system configuration design decisions such as the number and speed of disks, the amount of memory, and so on. These papers analyze features such as how often indexes were used, or the structure and complexity of their SQL statements, but they do not make actual performance predictions. For instance, [Elnaffar *et al.*, 2002] observes performance measurements from a running database system and uses a classifier (developed using machine learning) to identify OLTP vs. DSS workloads, but does not attempt to predict specific performance characteristics.

We believe that workload characterization is a side effect of using KCCA to predict query performance. The collocation of points on the projected query and performance spaces naturally lends itself to clustering techniques for characterizing queries by similarity. A useful avenue of future research would determine labels for such clusters and use them as feedback for benchmark creation.

3.7 Summary

In this chapter, we presented an adaptation of KCCA that allows us to accurately predict the performance metrics of database queries whose execution times range from milliseconds to hours. The most promising technique, based on KCCA and nearest neighbors, was not only the most accurate, but also predicted metrics such as elapsed time, records used, and message count simultaneously and using only information available prior to query execution. We believe our methodology can greatly improve the effectiveness of critical data warehousing tasks that rely on accurate predictions, including system sizing, capacity planning, and workload management.

The custom adaptation of KCCA we use combines relative differences between operator cardinalities and interpolation to build its model. This powerful combination allows us to make very accurate performance predictions for specific query plans. The predictions can be used to custom-calibrate optimizer cost estimates for a customer site at runtime. They also give us a quantitative comparison of different query plans for the same query.

Our methodology needs improvement to handle the use of queries to different tables in the training and test sets. Changes to the feature vectors, for example, incorporating data layout information, could help improve prediction results in this category. We also show scope for improving our multi-query workload prediction results. While we are able to provide an upperbound on workload elapsed time, we must re-evaluate the methodology to provide more accurate predictions.

A natural extension to the parallel database study is to validate the techniques on Map-Reduce environments [Dean and Ghemawat, 2004]. Recently, there has been a phase shift from large databases to Map-Reduce frameworks on cloud computing platforms to facilitate large scale data mining and analysis [Pavlo *et al.*, 2009]. In these environments, not only is the workload diverse, but also the churn rate of the underlying hardware/software configuration is quite high. The next chapter elaborates on performance prediction in the presence of configuration changes.

Chapter 4

Predicting Performance of Production Hadoop Jobs

In the previous chapter, we demonstrated our ability to predict performance of SQL queries to a parallel database system. In this chapter, we extend our prediction capabilities to a distributed computing environment with larger scale workloads. While this case study differs from the previous chapter in that the underlying system is built on heterogeneous and distributed hardware, our prediction methodology continues to be useful.

4.1 Motivation

The computing industry is recently experiencing a paradigm shift towards large-scale data-intensive computing. Internet companies such as Google, Yahoo!, Amazon, and others rely on the ability to process large quantities of data to drive their core business. Traditional decision support databases no longer suffice because they do not provide adequate scaling of compute and storage resources. To satisfy their data-processing needs, many Internet services turn to frameworks like Hadoop, an open-source implementation of a big-data computation paradigm [Dean and Ghemawat, 2004] complementary to parallel databases. At the same time, the advent of cloud computing infrastructures, such as Amazon EC2, makes large-scale cluster computing accessible even to small companies [Armbrust *et al.*, 2009]. The prevalence of SQL-like interfaces such as Hive [Thusoo *et al.*, 2009] and Pig [Olston *et al.*, 2008] further eases the migration of traditional database workloads to the Cloud.

These cloud computing environments present a new set of challenges for system management and design. One of the biggest challenges is resource provisioning and management. Given the heterogeneity introduced by building ad-hoc clusters from commodity hardware, it is difficult to simultaneously minimize resource contention and maximize data placement and locality. In these environments, heuristics and cost functions traditionally used for query

optimization no longer suffice. Furthermore, the large number of simultaneous users and the volume of data amplify the penalty of poor allocation and scheduling decisions. There is also an increasing need for resource management strategies that allow multiple metrics of success such as performance and energy efficiency.

Cloud providers must address several challenging questions to efficiently handle their workload:

- Which jobs should be scheduled together to avoid resource contention? This challenge is similar to our parallel database problem of scheduling queries only if they do not contend with currently running queries for available resources.
- Should some jobs be queued until a later time when the cluster is less loaded? The same problem surfaced in the context of our parallel database case study in the form of workload management.
- What is the optimal number of cluster nodes to allocate to a job given its resource requirements and deadlines? Since parallel databases do not provide abstractions to select the number of nodes to run each query on, this challenge is new in the context of cloud computing environments.
- Given observed behavior of a job run at small scale, how will the job behave when scaled up? Are the performance bottlenecks the same at larger scale? We explored similar issues in the previous chapter when we built performance models of various configurations, varying the number of nodes used, in a 32-node parallel database.

The above resource provisioning and scheduling decisions greatly benefit from the ability to accurately predict multiple performance characteristics. The goal of this chapter is to extend and apply our KCCA-based prediction methodology to Hadoop, a big-data computation framework complementary to parallel databases. Specifically, our methodology must address the following goals:

- Predict job execution time and resource requirements using a single model.
- Predict performance of the same job run under multiple configurations.
- Predict equally well for jobs that are expressed using SQL-like syntax and regular Hadoop jobs.
- Only use information/features available before job starts executing in a cluster.

This technique is an extension to our work in the previous chapter, where we have demonstrated the effectiveness of the KCCA-based technique for predicting query performance in parallel databases. With the right choice of predictive features, our KCCA-based methodology leads to highly accurate predictions that improve with the quality and coverage of

performance data. We evaluate our prediction methodology using production traces from Facebook's Hadoop cluster. We are able to accurately predict performance for a variety of data analytics jobs.

In the rest of this chapter, we provide an overview of Hadoop and the details of its deployment at Facebook, explain how we adapt the KCCA algorithm for predicting performance of Hadoop jobs, present prediction results for two types of data analytics workloads, and lastly discuss open questions and use cases for our methodology.

4.2 Experimental Setup

In this section, we provide an overview of the Map-Reduce abstraction and the Hadoop implementation that was used on a production cluster from which we obtain data for evaluating our prediction methodology.

4.2.1 Hadoop Overview

The Map-Reduce computation pattern has been around since the LISP language. Google chose the name MapReduce to refer to their implementation of this abstraction for parallel processing of large datasets [Dean and Ghemawat, 2004] on very large clusters. Today, MapReduce powers Google's flagship web search service. Several open-source implementations of this abstraction have emerged, including Hadoop, and are widely used on clusters at Yahoo!, Facebook, and other Internet services [Had, b]. Henceforth, we use *Map-Reduce* to refer to the computational pattern, *MapReduce* to refer to Google's implementation, and *Hadoop* to refer to the open source implementation. Programs written using Map-Reduce are automatically executed in parallel on the cluster. The fact that Map-Reduce can run on clusters of commodity machines provides significant cost-benefit compared to using specialized clusters. Furthermore, Map-Reduce scales well, allowing petabytes of data to be processed on thousands or even millions of machines. Most relevant to our work, Map-Reduce is especially suitable for the KCCA prediction methodology because it has a homogenous execution model, and production MapReduce and Hadoop workloads often perform repetitive computations on similar or identical datasets.

Figure 4.1 shows a schematic of Google's MapReduce architecture, which is also implemented by Hadoop. MapReduce requires the user to specify two functions. The *Map* function takes a key-value pair, and generates a set of intermediate key-value pairs. The *Reduce* function selects the subset of intermediate key-value pairs pertaining to a specific key, and emits the output key-value pairs based on user-specified constraints. Users can additionally specify a *Combine* function that minimizes network traffic by merging multiple intermediate key-value pairs before propagating them to the Reducer. Both the Map input pairs and Reduce output pairs are stored in an underlying distributed file system (DFS).

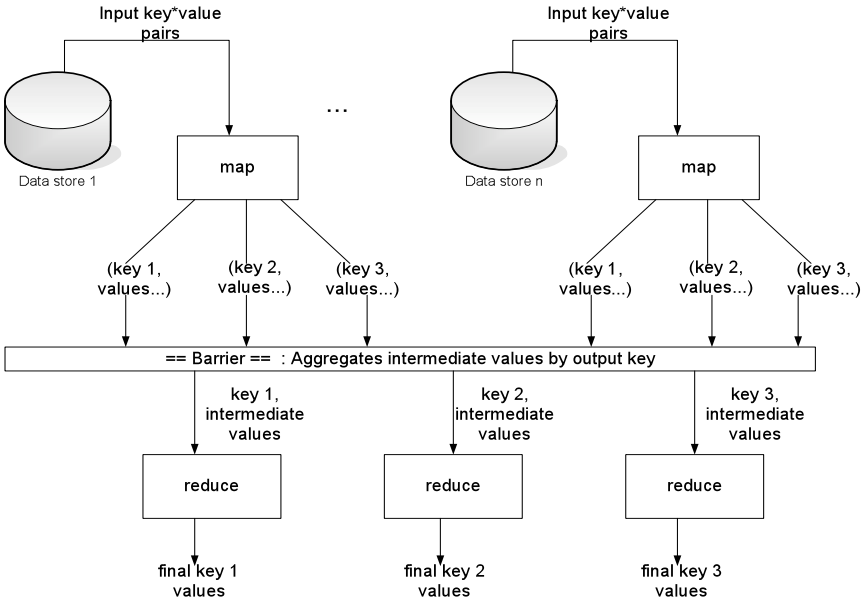

Figure 4.1: Architecture of MapReduce from [Dean and Ghemawat, 2004].

The run-time system takes care of retrieving from and outputting to the DFS, partitioning the data, scheduling parallel execution, coordinating network communication, and handling machine failures.

A MapReduce execution occurs in several stages. A MapReduce master daemon coordinates a cluster of worker nodes. The master divides the input file, resident in DFS, into many *splits*, and each split is read and processed by a Map worker. Intermediate key-value pairs are periodically written to the Map worker's local disk, and the master is updated with locations of these pairs. The Reduce workers collect these pairs' location information from the master, and subsequently read these pairs from the Map workers using a remote procedure call. Next, the Reduce worker sorts the data by its intermediate key, applies the user-specified Reduce function, and appends the output pairs to an output file, which resides in the DFS. Backup executions are launched for lagging Map or Reduce executions to potentially accelerate completion. An entire MapReduce computation is called a *job*, and the execution of a Map or Reduce function on a worker node is referred to as a *task*. For example, Figure 4.1 shows one job that consists of 6 map tasks and 3 reduce tasks. Each worker node allocates resources in the form of *slots*, and therefore, each map task or reduce task uses one *slot*.

For our work, we select the Hadoop implementation of MapReduce [Had, a]. In Hadoop, the user-supplied Map and Reduce functions can be written in many languages. The Hadoop distributed file system (HDFS) implements many features of the Google DFS [Ghemawat

et al., 2003]. There have been several efforts to extend Hadoop to accommodate different data processing paradigms. Most relevant to our work, Hive [Thusoo *et al.*, 2009] is an open source data warehouse infrastructure built on top of Hadoop. Users write SQL-style queries in a declarative language called HiveQL, which is compiled into MapReduce jobs and executed on Hadoop. Since Hive's query interface is similar to that of commercial parallel databases, it is a natural extension to evaluate the prediction accuracy of Hive queries in Hadoop using the KCCA-based technique as described in the previous chapter.

4.2.2 Cluster Configuration

Our data was collected from a production deployment of Hadoop at Facebook. This deployment was on a multi-user environment comprising 600 cluster nodes. Half of the nodes contained 16 GB memory, 5 map slots and 5 reduce slots each, and the remaining 300 nodes each contained 8 GB memory, 5 map slots and no reduce slots. We collected Hadoop job history logs over a six-month period and used a subset of the data for our analysis.

4.3 Addressing Design Challenges of Using KCCA

Similar to the parallel database query performance prediction case study in the previous chapter, our goal is to predict Hadoop job performance by correlating pre-execution features and post-execution performance metrics. KCCA allows us to simultaneously predict multiple performance metrics using a single model. This property captures interdependencies among multiple metrics, a significant advantage over more commonly used techniques such as Regression, which model a single metric at a time.

In the process of customizing KCCA for Hadoop job prediction, recall from previous chapters that we need to make the following three design decisions:

1. How to summarize the pre-execution information about each Hadoop job configuration into a vector of *job features*, and similarly, how to summarize the performance statistics from executing the job into a vector of *performance features*.

2. How to define a similarity measure between pairs of job feature vectors so that we can quantify how similar any two Hadoop jobs are, and likewise, how to define a similarity measure between pairs of performance vectors so that we can quantify the similarity of the performance characteristics of any two queries.

3. How to use the output of the KCCA algorithm to predict the performance of new Hadoop jobs.

The only difference between our Hadoop performance prediction methodology, compared to our parallel database performance prediction set up in Chapter 3, is the feature vectors

we used, and is described in Section 4.3.1. Similar to the database performance prediction case study, we chose the Gaussian kernel [Shawe-Taylor and Cristianini, 2004] to quantify the similarity between two query feature vectors or two performance feature vectors. Once we build the KCCA model, performance prediction is as follows. Beginning with a Hive query whose performance we want to predict, we create job feature vector $\hat{\mathbf{x}}$ and calculate its coordinates in subspace $K_x \times A$. We infer the job's coordinates on the performance projection subspace $K_y \times B$ by using its three nearest neighbors in the job projection. This inference step is possible because KCCA projects the raw data onto dimensions of maximal correlation, thereby collocating points on the job and performance projections. Finally, our performance prediction is calculated using a weighted average of the three nearest neighbors' raw performance metrics.

We next evaluate design decisions for defining our job and performance feature vectors using job history logs from a production Hadoop deployment at a major web service. We specifically focus on predicting performance of Hive queries since they provide a SQL-like interface to Hadoop through which users can run data analytics queries.

4.3.1 Hadoop job and performance feature vectors

A crucial step in using KCCA-based modeling is to represent each Hadoop job as a feature vector of job characteristics and a corresponding vector of performance metrics. This step is the only place where we deviate from the winning methodology in the previous chapter.

From the Hadoop job logs, we construct performance feature vectors to include map time, reduce time, and total execution time. These metrics represent the sum of the completion times of all the tasks corresponding to a specific job. These metrics are central to any scheduling decisions. We also include data metrics such as map output bytes, HDFS bytes written, and locally written bytes.

There are several possible options for job feature vectors. The choice greatly affects prediction accuracy. We evaluate two options for constructing the job feature vectors below.

4.3.1.1 Hive plan feature vector

The first choice of job feature vectors is an extension of the feature vectors in the previous chapter, shown to be effective for parallel databases.

Like relational database queries, Hive queries are translated into execution plans involving sequences of operators. We observed 25 recurring Hive operators, including Create Table, Filter, Forward, Group By, Join, Move and Reduce Output, to name a few. Our initial job feature vector contained 25 features – corresponding to the number of occurrences of each operator in a job's execution plan.

Figure 4.2 compares the predicted and actual execution time using Hive operator instance counts as job features. The prediction accuracy was very low, with a negative R^2

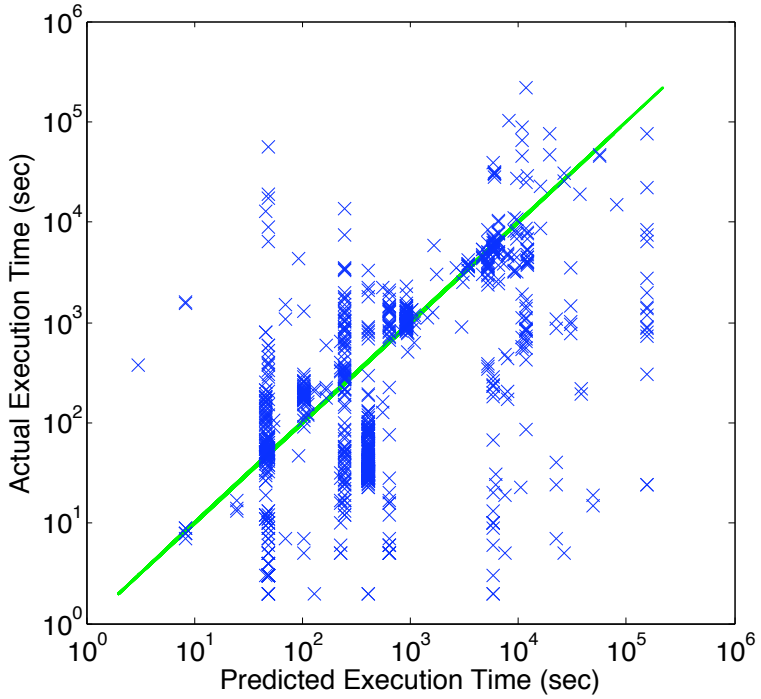

Figure 4.2: Predicted vs. actual execution time for Hive queries, modeled using Hive operator instance counts as job features. The model training and test sets contained 5000 and 1000 Hive queries respectively. The diagonal line represents the perfect prediction scenario. Note: the results are plotted on a log-log scale to accommodate variance in execution time.

value, indicating poor correlation between predicted and actual values[1]. We interpret the results to mean that Hive operator occurrence counts are insufficient for modeling Hive query performance.

This finding is somewhat unsurprising. Unlike relational databases, Hive execution plans are an intermediate step before determining the number and configuration of maps and reduces to be executed as a Hadoop job. Job count and configuration are likely to form more effective job feature vectors, since they describes the job at the lowest level of abstraction visible prior to executing the job.

4.3.1.2 Hadoop job log feature vector

Our next choice of job feature vector used Hive query's configuration parameters and input data characteristics. We included the number and location of maps and reduces required by

[1]Negative R^2 values are possible since the training data and test data are disjoint.

all Hadoop jobs generated by each Hive query, and data characteristics such as bytes read locally, bytes read from HDFS, and bytes input to the map stage. This feature vector choice provides much better results and also generalizes to non-Hive queries, as demonstrated in the subsequent section.

4.4 Results

In this section, we evaluate how well we were able to predict performance of Hive Jobs as well as a set of non-Hive data warehousing Hadoop jobs. For each experiment, we build the model using 5000 jobs in the training set, and evaluate our prediction accuracy on 1000 different jobs sampled exclusively for the test set.

4.4.1 Prediction for Hive jobs

Figure 4.3 shows our prediction results for the same training and test set of Hive queries as in Figure 4.2. Our R^2 prediction accuracy is now 0.87 ($R^2 = 1.00$ signifies perfect prediction). Using the same model, our prediction accuracy was 0.84 for map time, 0.71 for reduce time, and 0.86 for bytes written. Our prediction accuracy was lower for reduce time since the reduce step is fundamentally exposed to more variability due to data skew and uneven map finishing times. These results convincingly demonstrate that feature vectors with job configuration and input data characteristics enable effective modeling of Hive query performance.

4.4.2 Prediction for other Hadoop jobs

A significant advantage of our chosen job and performance feature vectors is that they contain no features that limit their scope to Hive queries. As a natural extension, we evaluate our performance prediction methodology on another class of Hadoop jobs that mimic data warehouse Extract Transform Load (ETL) operations. ETL involves extracting data from outside sources, transforming it to fit operational needs, then loading it into the end target data warehouse. KCCA prediction is especially effective for Hadoop ETL jobs because the same jobs are often rerun periodically with varying quantities/granularities of data. Also, KCCA prediction can bring great value because ETL jobs are typically long running. Thus, it is important to anticipate the job execution times so that system administrators can plan and schedule the rest of their workload.

Figure 4.4 shows the predicted vs. actual execution times of ETL jobs at the same Hadoop deployment, as well as prediction results for map time, reduce time and local bytes written. Our R^2 prediction accuracy for job execution time was 0.93. Prediction results for other metrics were equally good, with R^2 values of 0.93 for map time and 0.85 for reduce time. Although the R^2 values are better than Hive predictions, there are some visible prediction

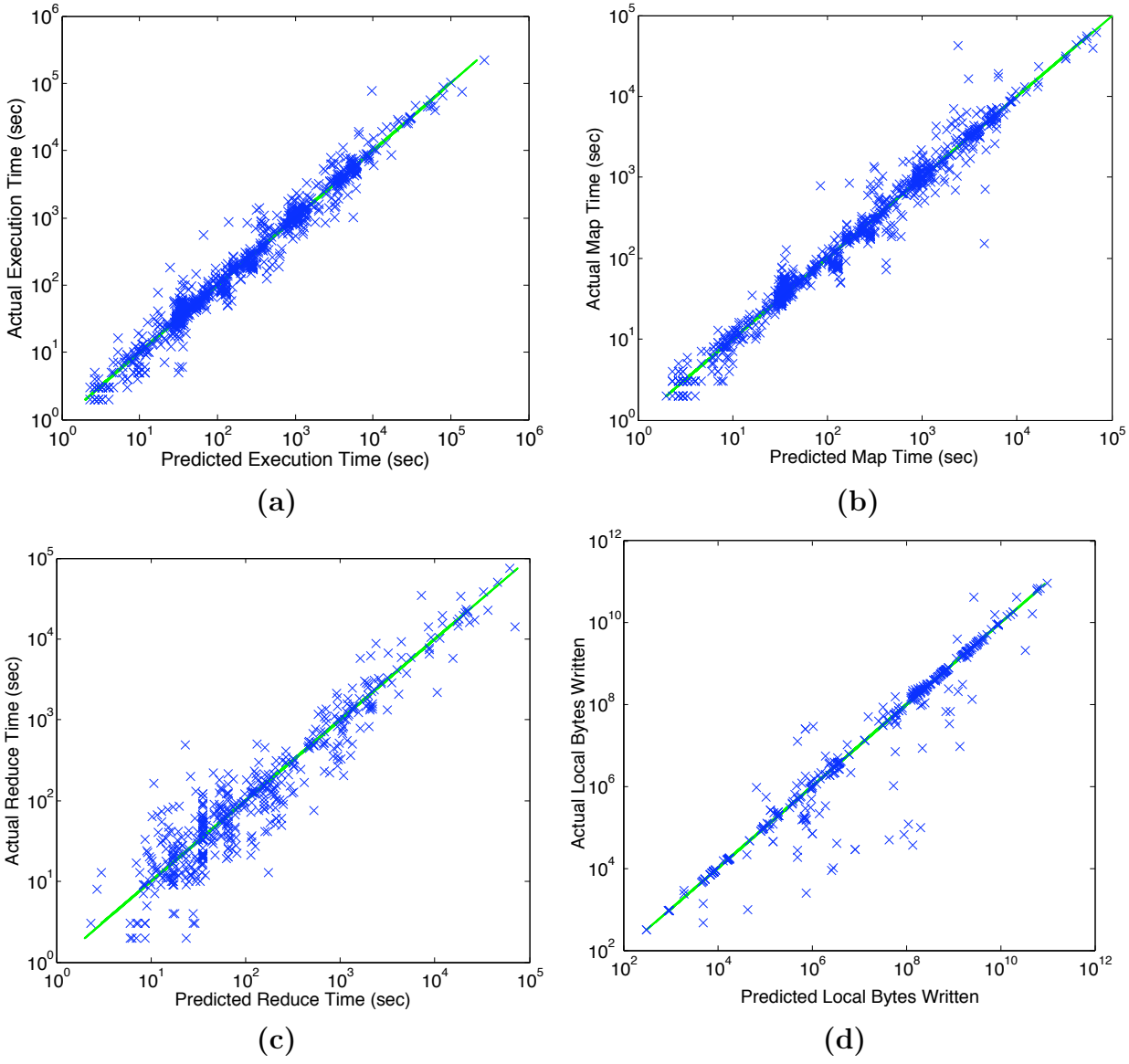

Figure 4.3: Prediction results for Hive queries, modeled using job configuration and input data characteristics as job features. The model training and test sets contained 5000 and 1000 Hive queries respectively. The diagonal lines represent the perfect prediction scenario. Note: the results are plotted on a log-log scale to accommodate the variance in data values. (a) Predicted vs. actual execution time, with R^2 value of 0.87. (b) Predicted vs. actual map time, with R^2 value of 0.84. (c) Predicted vs. actual reduce time, with R^2 value of 0.71. (d) Predicted vs. actual bytes written, with R^2 value of 0.86.

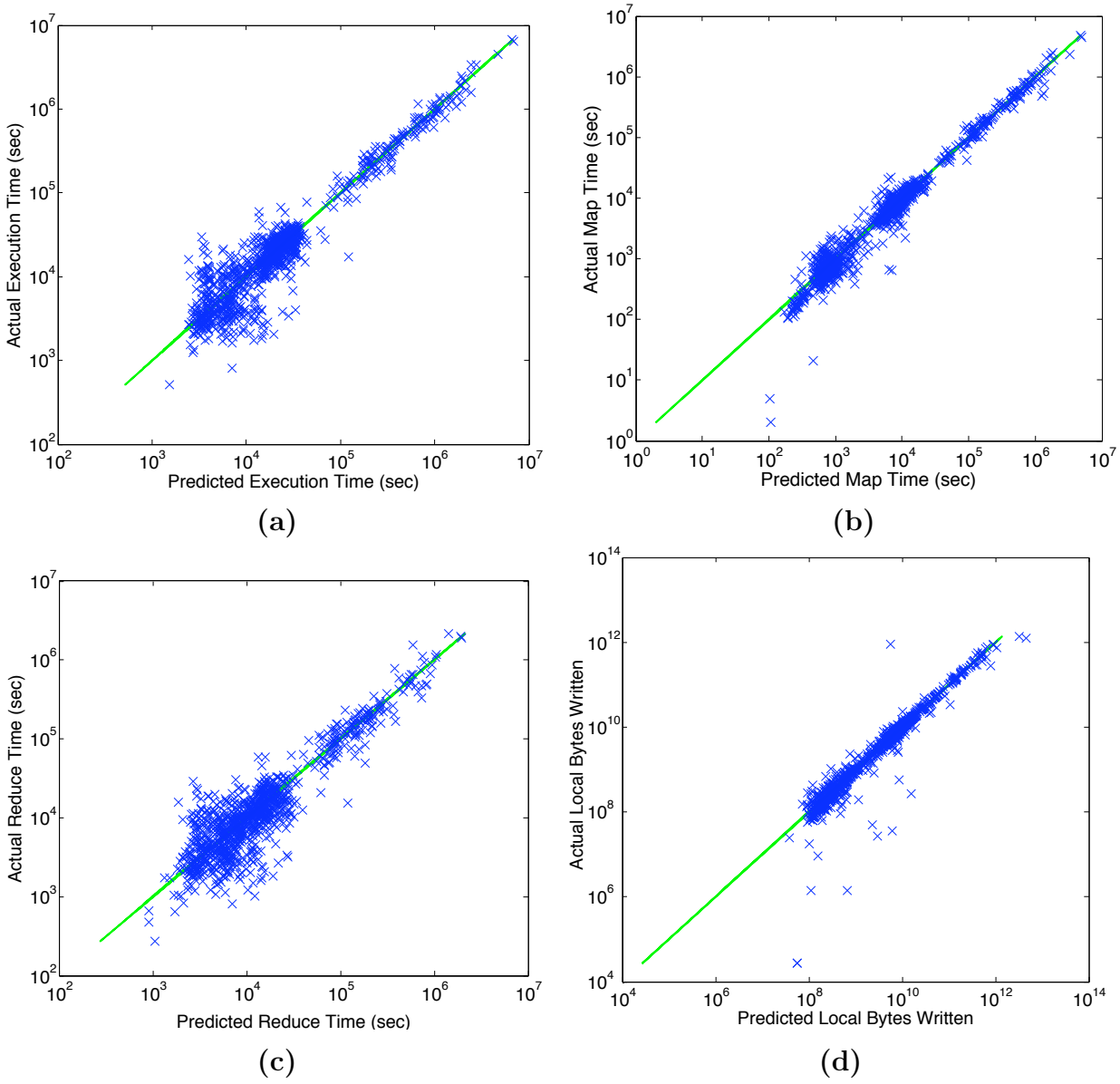

Figure 4.4: Prediction results for ETL jobs, modeled using job configuration and input data characteristics as job features. The model training and test sets contained 5000 and 1000 ETL jobs respectively. The diagonal lines represent the perfect prediction scenario. Note: the results are plotted on a log-log scale to accommodate the variance in data values. (a) Predicted vs. actual execution time, with R^2 value of 0.93. (b) Predicted vs. actual map time, with R^2 value of 0.93. (c) Predicted vs. actual reduce time, with R^2 value of 0.85. (d) Predicted vs. actual bytes written, with R^2 value of 0.61 due to visible outliers.

errors for jobs with short execution times. These jobs are inherently more difficult to predict because the setup overhead for big data Hadoop jobs is high, contributing to increased job execution time. Nevertheless, the results indicate that the KCCA-based methodology is also effective for predicting Hadoop ETL jobs.

4.5 Lessons and Limitations

To realize the full potential of our prediction methodology, there are several concerns to address before integrating KCCA with scheduling and resource management infrastructures.

4.5.1 Can we predict well for other job types?

In Section 4.4, we show prediction results only for Hive queries and ETL jobs. However, we believe the prediction methodology is applicable for a more generic set of Hadoop jobs. The features we used for our predictions are common to all Hadoop jobs. Given that resource demands vary by the specific map and reduce functions, we can augment the feature vectors with specific map and reduce function identifiers and/or the language in which these functions were implemented.

4.5.2 Can we use this methodology to predict task completion times?

Recall that a Hadoop job is composed of several map tasks and reduce tasks. Although each map task for a job performs identical operations, the task completion times can vary based on which nodes the tasks run on. The completion time variability is even more pronounced in among reduce tasks for a job because there is added uncertainty due to data locality. With task-level execution logs, we can use the same modeling framework to model relationships between various tasks and their performance characteristics. For example, accurate, statistics-driven prediction of task finishing times can help us better identify lagging tasks. These lagging tasks may indicate slow or faulty cluster nodes or asymmetric load conditions.

4.5.3 How can we include other resources in our model?

Given fine-grained instrumentation, we can monitor the utilization of various resources including CPU, disk, and network. Current state-of-the-art tools such as Ganglia [Massie *et al.*, 2003] measure per-node resource utilization on regular intervals. However, the granularity of measurement does not decouple resources consumed per-task.

Figure 4.5 provides an example of measuring performance metrics using Ganglia for two types of Hadoop jobs run on a single node. The results show two runs of *pi* and two runs

Trial: Job Type	time	bytes_in	bytes_out	cpu_idle	cpu_user	mem_cached	mem_free
T1: Grep	1:38	26437	30768	98.01	1.62	63704	10354957
T2: Grep	1:24	32113	33918	97.61	1.93	65360	10334010
T1: Pi	53	32542	35048	97.51	2.01	64209	10353230
T2: Pi	55	34669	34646	97.68	1.86	65397	10339415
T1: Simultaneous Grep+Pi	1:55	31237	35909	97.38	2.11	63987	10334261
T2: Simultaneous Grep+Pi	2:22	37989	37819	97.09	2.37	67124	10307272

Figure 4.5: Hadoop job performance metrics measured using Ganglia.

of *grep* in isolation, and two trials of simultaneously running *pi* and *grep*. Based on our timing results alone, it is unclear whether running the two job types simultaneously provides any gain or loss over running them in isolation. Such ambiguity in resource requirements is exacerbated when jobs are run simultaneously across multiple nodes in a cluster. The same trend was observed for co-scheduled queries in a parallel database system.

Per-job performance monitoring facilities would alleviate this problem and allow more accurate resource measurements. Such instrumentation would allow us to augment the job feature vector with resource consumption levels, providing more accurate predictions to the scheduler. Furthermore, we can "rerun" the workload on different hardware and cluster configurations, turning the KCCA framework into an even more powerful tool that can directly compare resource consumption between different hardware/cluster configurations.

4.5.4 Can we predict and prevent Hadoop master node overload?

A major insight we learned from presenting our results at Facebook is that the Hadoop master node, by design, is a single point of failure. In production, the master node consistently crashes due to exhausted memory resources from tracking too many jobs simultaneously. Our methodology could be adapted to predict how much memory is required for the master node to track a new Hadoop job given its anticipated number of tasks and splits. If the node's available memory does not accommodate the new job's resource requirements, we can retire old jobs from the master node's job-tracking list.

4.5.5 How can this methodology be used for cluster resource provisioning?

Given predicted resource requirements and desired finishing time, one can evaluate whether there are enough resources in the cluster or if more nodes should be pooled in. If predicted execution time is too long, one can assign nodes for other computations. Conversely, if

predicted time is very short, one can turn off unnecessary nodes to reduce power, or cost, in the public cloud, without impacting performance.

4.5.6 How can we use this prediction methodology to drive job scheduling decisions?

A good scheduler should understand whether particular jobs have conflicting or orthogonal resource demands. However, accurate finishing time predictions alone enable several scheduler policies. One such example is a shortest-job-first scheduler, where we can minimize per-job wait time by executing jobs in order of increasing predicted execution time. We can also implement a deadline-driven scheduler that allows jobs with approaching deadlines to jump the queue "just in time."

4.5.7 Can we use models from one organization to predict well for Hadoop workloads from other organizations?

Different organizations have different computational needs. We need to evaluate the prediction effectiveness for different workloads to increase confidence in the KCCA framework. In the absence of a representative Hadoop benchmark, we must rely on case studies on production logs. However, access to production logs involves data privacy and other logistical issues. Thus, we need a way to anonymize sensitive information while preserving a statistical description of the Hadoop jobs and performance. We can create a workload generator to replay this *necessary and sufficient* workload summary to compare resource requirements for workloads across multiple organizations.

4.6 Related Work

The open source nature of Hadoop has made it easy to augment the implementation with new optimizations. Our high prediction accuracy suggests that related work on Hadoop optimization should consider augmenting their mechanisms using KCCA predictions.

Morton and others adapt traditional query progress indication techniques to estimate the progress of Pig queries [Morton *et al.*, 2010]. They estimate remaining execution time for these queries by using heuristics that compare the number of tuples processed versus the number of remaining tuples to process. Our job execution time prediction results can replace these heuristics and provide more accurate completion time predictions using only information available prior to executing these queries.

Zaharia and his colleagues use similar heuristics to determine task completion times for scheduling decisions. They implement a way to speculatively execute backup tasks that are estimated to have a completion time far in the future [Zaharia *et al.*, 2008]. The LATE

scheduler could be augmented with KCCA-based task execution time predictions instead of the finishing time estimation heuristics to make even better scheduling decisions.

For scheduling jobs in a multi-user environment, Zaharia and his colleagues implemented the Hadoop FAIR scheduler [Zaharia *et al.*, 2009]. This scheduler can improve its resource allocation using predicted execution time in addition to using the number of slots as a resource consumption proxy. This combination could lead to better resource sharing and decrease the need for killing or preempting tasks.

In addition, recent work on pipeline task execution and streaming queries [Condie *et al.*, 2009], as well as resource managers from multiple computational frameworks that include Map-Reduce [Hindman *et al.*, 2009] would benefit from our ability to predict at various granularities for a variety of jobs.

4.7 Summary

In this chapter, we described an adaptation of the KCCA-based prediction methodology for predicting performance of Hadoop jobs. Our technique accurately predicted multiple performance metrics – including total execution time, map time, reduce time, and bytes written – using a single model. We validated our approach on two types of data analytics Hadoop workloads in a production environment.

Our prediction methodology is currently being used to implement a statistics-driven Hadoop job scheduler. Furthermore, to validate the prototype with realistic workload without privacy concerns of obtaining real production logs, we are also building a Hadoop workload generator [Ganapathi *et al.*, 2009a]. This workload generator uses distributions for each job input and configuration feature we used for our prediction, and generates jobs that have similar input - shuffle - output ratios.

Although accurate performance predictions are powerful, we would like to adapt our KCCA-based methodology for performance optimization as well. Configuration management is a challenging task for many systems and identifying the appropriate configuration settings to optimize performance is often an art rather than a science. In the next chapter, we tackle the problem of performance tuning on multicore platforms.

Chapter 5

Auto-tuning High Performance Computing Motifs on a Multicore System

In the previous two chapters, we demonstrated the power of our KCCA-based technique for accurately predicting the performance of diverse workloads on two different systems. In this chapter, we further generalize the usefulness of this technique by adapting it for performance tuning in a parallel computing environment. In contrast to the prediction problem, where we use previously seen workloads to predict performance of future workloads, this case study finds an optimal configuration to run a fixed workload on a specific platform.

5.1 Motivation

The multicore revolution has produced several complex processor architectures. As new generations of hardware are continually released, their complexity increases with higher core counts, varying degrees of multithreading, and heterogeneous memory hierarchies. This rapidly evolving landscape has made it difficult for compilers to keep pace. As a result, compilers are unable to maximally utilize system resources. This architectural trend has also made it impossible to perform effective hand-tuning for every new release within and across processor families.

Auto-tuning has emerged as an effective solution that can be applied across platforms for performance optimization. Auto-tuning first requires a performance expert to identify a set of useful optimizations and acceptable parameter ranges using hardware expertise and application-domain knowledge. Then an expert programmer generates appropriate code variants corresponding to these optimizations. Lastly, the auto-tuner searches the parameter space for these optimizations to find the best performing configuration. In the last decade, auto-tuning has been successfully used to tune scientific kernels for serial [Frigo and Johnson, 2005; Vuduc *et al.*, 2005] and multicore processors [Williams *et al.*, 2007; Datta *et al.*, 2008].

Auto-tuning is scalable, automatic, and can produce high-quality code that is several times faster than a naïve implementation [Demmel *et al.*, 2005]. Unfortunately it suffers from two major drawbacks. The first drawback is the size of the parameter space to explore: state-of-the-art auto-tuners that consider only a single application, a single compiler, a specific set of compiler flags, and homogeneous cores may explore a search space of nearly 40 million configurations [Datta *et al.*, 2008]. This search would take about 180 days to complete on a single machine [Ganapathi *et al.*, 2009b]. If the auto-tuner considers multiple compilers, multichip non-uniform memory access (NUMA) systems, or heterogeneous hardware, the search becomes prohibitively expensive. Even parallel exploration of multiple configurations (e.g. in a supercomputing environment) achieves only linear speedup in the search. Furthermore, cloud computing environments would not help parallel configuration explorations because tuning must occur on dedicated nodes with a specific architecture. Therefore, most auto-tuners prune the space by using extensive domain knowledge about the platform as well as the code being tuned. Such domain knowledge is accumulated over months/years of experience and is neither easily portable to new platforms nor easily documentable for subsequent use.

The second drawback with auto-tuning is that most auto-tuners only minimize overall running time. Given that power consumption is a proximate cause of the multicore revolution, and a vital metric for tuning embedded devices, it is important to tune for energy efficiency as well. A performance-optimal configuration is not necessarily energy-optimal, and vice versa. Furthermore, good or bad performance is often a second order effect of caching effects and memory traffic. Thus, capturing these dependencies is important in identifying performance optimization strategies.

In this chapter, we show how we can adapt the same SML methodology used in previous chapters to address the problem of efficiently exploring the auto-tuning configuration space. SML algorithms allow us to draw inferences from automatically constructed models of large quantities of data [Goldszmidt *et al.*, 2005]. We leverage these models for identifying a subspace of configuration parameters that produces optimal performance. We set our goals as follows:

- Require minimal micro-architecture and application domain knowledge. This goal is similar to our database and Hadoop case studies' goals of using a methodology that is not implementation-dependent and requires minimal understanding of the system's software architecture.

- Simultaneously optimize for multiple metrics of success including performance and energy efficiency. This goal resembles the previous case studies' goals of predicting multiple metrics simultaneously using a single model.

- Efficiently prune the vast parameter space and traverse a subspace that produces good configurations in less than a day. This time constraint follows the previous case studies'

constraints that the prediction time must be a fraction of the query/job's execution time.

- Easily extend to handle problems with even larger search spaces. This goal allows the auto-tuning methodology to be portable to new architectures with more tunable parameters.

To meet these goals, we use an adaptation of KCCA as described in Section 2.3. Recast from a SML perspective, auto-tuning leverages relationships between a set of optimization parameters and a set of resultant performance metrics to explore the search space. Several hurdles to applying SML in our previous two case studies from the systems domain are absent in the high performance computing (HPC) domain. We enumerate several examples below:

- In contrast to the technical and political difficulty of obtaining data from a production deployment of the parallel database and Hadoop cluster, real data for multicore/architecture problems can be obtained by using commodity hardware to run a problem instance. While representative Hadoop jobs and decision support workloads require hours or days-long experimental runs, multicore/architecture experimental runs can complete in minutes or hours.

- Systems vary widely in the quantity, granularity, and uniformity of instrumentation available. For example, the granularity of measuring resource utilization in Hadoop does not match the specific time window during which a job executes. Therefore, getting "ground truth" from systems datasets—i.e., understanding what "really happened" during a failure scenario—can be elusive and complex. In contrast, the architecture community has a long-established culture of designing for testability and measurability via mechanisms such as programmer-visible performance counters so we can repeat the experiment *ad nauseam* and collect instrumentation to any desired degree of detail.

- Lastly, since the architecture community tends to optimize applications with exclusive access to hardware, the SML models need not adapt to varying machine usage patterns and externally-imposed load conditions. This advantage is significant compared to the multi-user environments in which multiple Hadoop jobs or multiple data warehousing queries must simultaneously execute. Thus, models built once for auto-tuning should be reusable without change.

As a result of these advantages, we see a great opportunity for using SML for optimizing performance of HPC code.

We evaluate our methodology by optimizing three different HPC motifs – a 7-point stencil, a 27-point stencil, and a matrix multiply problem – on two modern multicore platforms. For

each of these motifs, we collect training data to build our KCCA-based models and identify new configurations that perform well with respect to various metrics of interest. Our results reveal that we are able to reduce the half-year long exhaustive search to a few minutes while achieving performance at least within 1% of and up to 50% better than that achieved by a human expert. Similar to the parallel database prediction results, our SML-based auto-tuner results are comparable to the state-of-the-art techniques' results.

In the remainder of this chapter, we describe the three HPC motifs and the two multicore platforms we optimize them for, explain our adaptation of the KCCA algorithm for performance optimization, present results comparing the performance of an expert's configuration parameters to our SML-guided parameters, and lastly review open questions and related work.

5.2 Experimental Setup

In this section, we describe the platforms we ran our experiments on and the three HPC motifs we tuned. Subsequent sections show the results of optimizing each individual motif on our multicore platforms.

5.2.1 Multicore Platforms

We conduct our performance optimization experiments on two multicore platforms: the Intel Clovertown and the AMD Barcelona. Figure 5.1 shows architectural schematics for both these platforms. Both the Clovertown and Barcelona are superscalar out-of-order machines based on the x86 architecture.

Both our experimental platforms are both dual-socket and quad-core with one hardware thread per core. Clovertown has a clock rate of 2.66 GHz and offers DRAM bandwidth of up to 10.7 GB/s, with a peak performance of 85.3 GFLOPs per second. Barcelona, on the other hand, has a clock rate of 2.30 GHz and offers 9.2 GB/s peak bandwidth with peak performance of 73.6 GFLOPs per second. We compiled our code using the icc compiler on Clovertown and gcc on Barcelona.

To measure performance metrics on these platforms, we use PAPI performance counters [PAPI, 2009]. PAPI provides a uniform interface to the platforms' in-built performance counter mechanisms. Table 5.3, on page 69, shows the PAPI performance counters we use for the Clovertown and Barcelona. The performance metrics we measure include total cycles (time), cache misses, TLB misses, and cache coherency traffic. While total cycles is the ultimate measure of performance, we felt it was equally important to capture counters that heavily influence this metric.

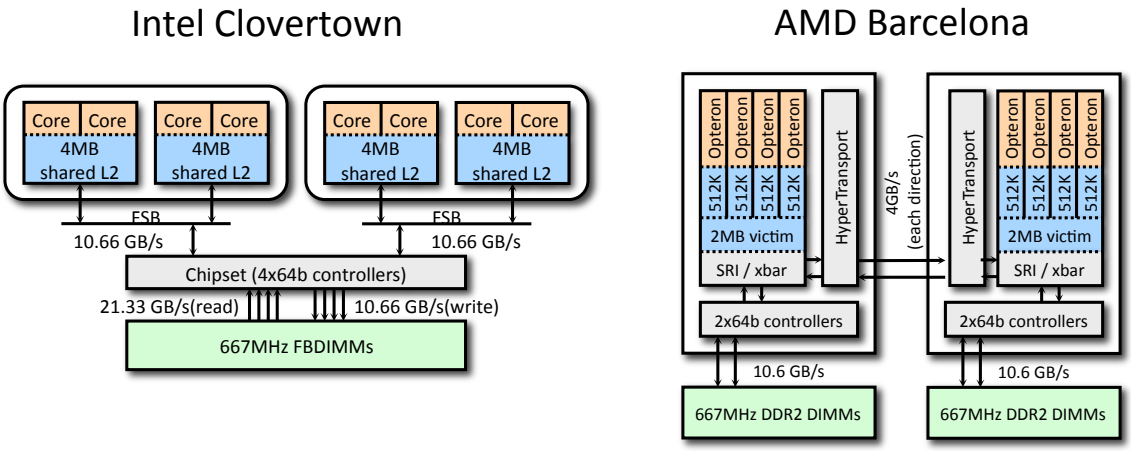

Figure 5.1: This figure shows schematics of the Intel Clovertown and AMD Barcelona architectures. Diagrams courtesy of Sam Williams.

5.2.2 HPC Motifs and Optimizations

Asanovic and others [Asanovic *et al.*, 2006] have identified thirteen common communication and computation patterns in parallel computing, of which seven are fundamental to HPC. These *motifs* are common targets of performance optimization. We next describe two such motifs – stencil codes and PGEMM – and possible techniques to optimize their performance.

5.2.2.1 Stencil Codes

Stencil (nearest-neighbor) computations are used extensively in partial differential equation (PDE) solvers involving structured grids [Berger and Oliger, 1984; Applied Numerical Algorithms Group, 2009]. In addition to their importance in scientific calculations, stencils are interesting as an architectural evaluation benchmark because they have abundant parallelism and low computational intensity, offering a mixture of opportunities for on-chip parallelism and challenges for associated memory systems.

In a typical stencil code, each element of a grid receives a new value based on the elements adjacent to it; the number of such elements varies with the dimensions of the grid and size of the stencil. Our work utilizes 3D 7-point and 27-point stencils arising from finite difference calculations.

Figure 5.2 provides a visualization of the 7-point stencil and corresponding pseudo-code. For each stencil run, we perform a single sweep over a 256^3 grid where the read and write arrays are distinct. This grid size ensures that the problem does not fit into cache. For the 7-point stencil each point in the grid is updated by considering its 6 adjacent neighbors while the 27-point stencil considers its 26 neighbors in a $3 \times 3 \times 3$ inner-grid. The 27-point stencil

$$
\begin{aligned}
B[i,j,k] \quad &= \quad C_0(A[i,j,k]) + \\
&C_1(\quad A[i+1,j,k] + A[i-1,j,k] \\
&+ \quad A[i,j+1,k] + A[i,j-1,k] \\
&+ \quad A[i,j,k+1] + A[i,j,k-1])
\end{aligned}
$$

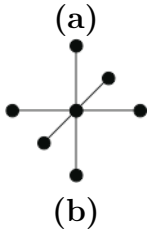

Figure 5.2: 3D 7-point stencil: (a) Pseudo-code. (b) Visualization.

performs significantly more FLOPs per byte transferred than the 7-point stencil (1.25 vs. 0.33 FLOPs/byte), and thus is more likely to be compute-bound compared to the bandwidth-bound 7-point stencil.

Prior work on auto-tuning of stencil codes for multicore architectures [Datta *et al.*, 2008] identified a set of possibly effective optimizations, along with an appropriate choice of parameters. Table 5.1, on page 67, shows the four basic categories of the optimizations. The first one, *domain decomposition*, specifies how the grid is broken into blocks and allocated to individual cores. This allocation is necessary for parallelization. The second optimization, *software prefetching*, was employed to mask memory latency. Next, we performed *padding* to reduce cache conflict misses by adding extra elements onto the contiguous dimension of the array. These extra elements were not used in computation, however. Finally, the inner loop optimizations rewrote the inner loop to expose instruction-level parallelism. The *register block size* indicates the number of loop unrollings performed in each of three dimensions. For more details on the optimizations and parameters, we refer the reader to [Datta *et al.*, 2008]. Currently, the full parameter space produces nearly 40 million combinations, which is intractable to search over.

5.2.2.2 PGEMM

Matrix multiplication is an essential linear algebra building block for many HPC applications. *PGEMM* is an implementation of the matrix multiply operation that forms the foundation of BLAS (Basic Linear Algebra Subprograms) [Whaley *et al.*, 2001]. Figure 5.3 shows the pseudo-code that PGEMM implements. C is a matrix of dimensions $M \times N$, and is updated

```
for i = 1 to M do
  for j = 1 to N do
    for k = 1 to K do
      C[i,j] = C[i,j] + A[i,k]*B[k,j]
    end for
  end for
end for
```

Figure 5.3: PGEMM pseudo-code.

by multiplying matrix A of dimensions $M \times K$ with matrix B of dimensions $K \times N$. To ensure our problem size exceeds the cache size, to make the problem more interesting, we set $M = N = K = 768$ for our experiments.

Depending on the chosen cache block size, PGEMM performs as few as 4 and up to 32 FLOPs per byte transferred, which makes it significantly more computation bound and significantly less bandwidth bound compared to the two stencils in Section 5.2.2.1.

Whaley and others [Whaley *et al.*, 2001] enumerate several optimizations and corresponding parameters for improving the performance of matrix multiplication. Table 5.2, on page 68, elaborates on these parameters, which overlap significantly with the optimization parameters for stencils. The only new addition to the set of optimizations is *simd*, which allows the code to leverage single instruction-multiple data facilities in the underlying x86 instruction set. The full parameter space produces over 8.6 million combinations, which is intractable to search over.

The next section addresses the issue of how to navigate the parameter space quickly with minimal domain knowledge. Using machine learning to guide the auto-tuning process, we can reduce the search space to a tractable scale.

5.3 Addressing Design Challenges of Using KCCA

Auto-tuning in the context that we are proposing explores a much larger search space than the previous work in earlier chapters, thereby exercising the full potential of newer SML algorithms. Recast from a SML perspective, auto-tuning leverages relationships between a set of optimization parameters and a set of resultant performance metrics to explore the search space. We used the KCCA-based methodology presented in Section 2.3 to identify these relationships. Since KCCA finds multivariate correlations between optimization parameters and performance metrics on a training set of data, we can leverage these relationships to optimize for performance and energy efficiency.

Optimization	Parameter Type	Name	Parameter Range	Number of Configurations
Domain Decomposition	Block Size	CX	$\{2^7 \ldots \text{NX}\}$	36
		CY	$\{2^1 \ldots \text{NY}\}$	
		CZ	NZ	
	Chunk Size		$\{1 \ldots \frac{NX \times NY \times NZ}{CX \times CY \times CZ \times NThreads}\}$	
Software Prefetching	Prefetching Type		{register block, plane, pencil}	3
	Prefetching Distance		$\{0, 2^5 \ldots 2^9\}$	6
Padding	Padding Size		$\{0 \ldots 31\}$	32
Inner Loop	Register Block Size	RX	$\{2^0 \ldots 2^1\}$	10
		RY	$\{2^0 \ldots 2^1\}$	
		RZ	$\{2^0 \ldots 2^3\}$	
	Statement Type		{complete, individual}	2
	Read From Type		{array, variable}	2
	Pointer Type		{fixed, moving}	2
	Neighbor Index Type		{register block, plane, pencil}	3
	FMA-like Instructions		{yes, no}	2

Table 5.1: Attempted optimizations and the associated parameter spaces explored by the auto-tuner for a 256^3 stencil problem ($NX, NY, NZ = 256$) using 8 threads ($NThreads = 8$). All numbers in the parameter tuning range are in terms of double precision numbers.

Recall that there are three design decisions to be considered when customizing KCCA for auto-tuning:

1. How to represent optimization parameters as *configuration feature vectors* and performance metrics as *performance feature vectors*?
2. How to define kernel functions for pairs of configuration vectors so that we can quantify how similar two configurations are, and similarly, how to define kernel functions between pairs of performance features vectors?
3. How to use the output of the KCCA algorithm to identify optimal configurations?

We discuss our choices for the above design decisions in turn.

5.3.1 Configuration and Performance Feature Vectors

For each stencil code run, we construct one configuration vector and one performance vector. Selecting features to represent the configurations was straightforward, given the optimization parameters described in Tables 5.1 and 5.2. Configuration feature vectors for stencils

Optimization	Parameter Type	Name	Parameter Range	Number of Configurations
Domain Decomposition	Register Tile Size	rr	$\{2^0 \dots 2^3\}$	4
		rc	$\{2^0 \dots 2^3\}$	4
	Cache Block Size	nb	$\{2^4 \dots 2^7\}$	4
		kb	$\{2^4 \dots 2^7\}$	4
Software Prefetching	Prefetch Type		{NTA, T0, T1, T2}	4
	Prefetch Distance for A	pA	{0, 1, 2, 4, 8}	5
	Prefetch Distance for B	pB	{0, 1, 2, 4, 8}	5
Inner Loop	K Loop Unroll Factor		$\{2^0 \dots$ kb$\}$	7
	Loop Order		{IJK, JIK}	2
SIMD	Use SIMD Layout		{0, 1}	2
	SIMD Dimension		{M, N, K}	3

Table 5.2: Attempted optimizations and the associated parameter spaces explored by the auto-tuner for a 768^3 pgemm problem ($M, N, K = 768$) using 8 threads ($NThreads = 8$). All numbers in the parameter tuning range are in terms of doubles.

contained a different set of parameters than the configuration feature vectors for PGEMM code because of the differences in the optimizations exposed by the code generator for these two motifs. However, the 7-point and 27-point stencil configurations were represented using the same set of configuration features.

Constructing performance feature vectors for these motifs was straightforward given the performance counters we measured for each platform. The specific features we used for these vectors varied between Clovertown and Barcelona to account for differences in available performance counters. Table 5.3 shows the list of performance counters we used as features. We augmented the performance feature vector with an energy efficiency metric, which was calculated based on power meter readings of Watts consumed during experimental runs.

For each motif, we randomly choose 1500 configurations to build our training data sets. The random sampling was performed independently for each motif on each platform. We limit our sample size to 1500 datapoints as the KCCA algorithm is exponential with respect to the number of datapoints. We run the motif's code variant reflecting each chosen configuration and collect low-level performance counter data and power measurements. For the 7-point and 27-point stencils, N=1500 configuration vectors of K=16 features are combined into a $N \times K$ configuration matrix; the N performance vectors of L features (L=8 for Clovertown and L=6 for Barcelona) produce a $N \times L$ performance matrix. The corresponding rows in each of the two matrices describe the same stencil run. Similarly, for PGEMM, the N configuration vectors of K=12 features are combined into a $N \times K$ configuration matrix; the N performance vectors of L features (L=8 for Clovertown and L=6 for Barcelona) produce

Counter Name	Counter Description	Clovertown	Barcelona
PAPI_TOT_CYC	Clock cycles per thread	✓	✓
PAPI_L1_DCM	L1 data cache misses per thread	✓	✓
PAPI_L2_DCM	L2 data cache misses per thread	✓	✓
PAPI_L3_TCM	L3 total cache misses per thread		✓
PAPI_TLB_DM	TLB data misses per thread	✓	✓
PAPI_CA_SHR	Accesses to shared cache lines	✓	
PAPI_CA_CLN	Accesses to clean cache lines	✓	
PAPI_CA_ITV	Cache interventions	✓	
	Power meter (Watts/sec)	✓	✓

Table 5.3: Measured performance counters on the Clovertown and Barcelona architectures.

a $N \times L$ performance matrix. The corresponding rows in each of the two matrices describe the same PGEMM run.

5.3.2 Defining Kernel Functions

Recall that a useful aspect of the KCCA algorithm is that it produces neighborhoods of similar data with respect to configuration features as well as performance features. However, to achieve this result, KCCA uses *kernel functions*[1] to define the similarity of any two configuration vectors or any two performance vectors.

Since our performance vector contains solely numeric values, we use the Gaussian kernel function [Shawe-Taylor and Cristianini, 2004] below:

$$k_{Gaussian}(y_i, y_j) = exp\{-\|y_i - y_j\|^2/\tau_y\}$$

where $\|y_i - y_j\|$ is the Euclidian distance and τ_y is calculated based on the variance of the norms of the data points. We derive a symmetric matrix K_y such that $K_y[i,j] = k_{Gaussian}(y_i, y_j)$. If y_i and y_j are identical, then $K_y[i,j] = 1$.

Since the configuration vectors contain both numeric and non-numeric values, we construct a kernel function using a combination of two other kernel functions. For numeric features, we use the Gaussian kernel function. For non-numeric features we define:

$$k_{binary}(x_i, x_j) = \begin{cases} 1 & \text{if } x_i = x_j, \\ 0 & \text{if } x_i \neq x_j \end{cases}$$

[1]Our use of the term *kernel* in this chapter refers to the SML *kernel function* and not HPC scientific *kernels*.

We define our symmetric matrix K_x such that

$$K_x[i,j] = average(k_{binary}(x_i, x_j) + k_{Gaussian}(x_i, x_j))$$

Thus, given the $N \times K$ configuration matrix and the $N \times L$ performance matrix, we form a $N \times N$ matrix K_x and a $N \times N$ matrix K_y, which are used as input to the KCCA algorithm.

5.3.3 Identifying Optimal Configurations

Upon running KCCA on K_x and K_y, we obtain projections K_xA and K_yB that are maximally correlated. We leverage these projections to find an optimal configuration. We first identify the best performing point in our training set, called P_1. We look up its coordinates on the K_yB projection, and find its two nearest neighbors, called P_2 and P_3, on the projected space. We then construct new configuration vectors using a genetic algorithm to determine all combinations of the optimizations in P_1, P_2, and P_3. We do not vary the parameters within each optimization. We run these new configurations to identify the best performing configuration.

5.4 Results

We evaluate our adaptation of KCCA for multicore performance optimization by comparing results from using our methodology to results from several alternate approaches:

- *No Optimization*: This approach assigns baseline parameter values for the optimization parameters, mimicking the scenario that generates unoptimized code.
- *Expert Optimized*: For both the stencils, this approach is derived from techniques suggested by Datta and others [Datta *et al.*, 2008], where the application/architecture domain expert ranks optimizations by likely effectiveness and applies them consecutively to determine a good configuration. For PGEMM, this approach exhaustively searches the parameter space for a subset of optimizations using a small matrix size that fits into cache. Then, the best resultant configuration is used on the 768^3 problem while exhaustively exploring the prefetch parameter space.
- *Random Raw Data*: This approach takes the best performing configuration from our randomly generated raw training data points.
- *Genetic on Raw Data*: This approach represents the results of using the genetic algorithm in Section 5.3.3 on the raw data. Given the top three best performing training data points, this approach permutes the parameters for each optimization, and identifies the best performing configuration from the permuted subset.

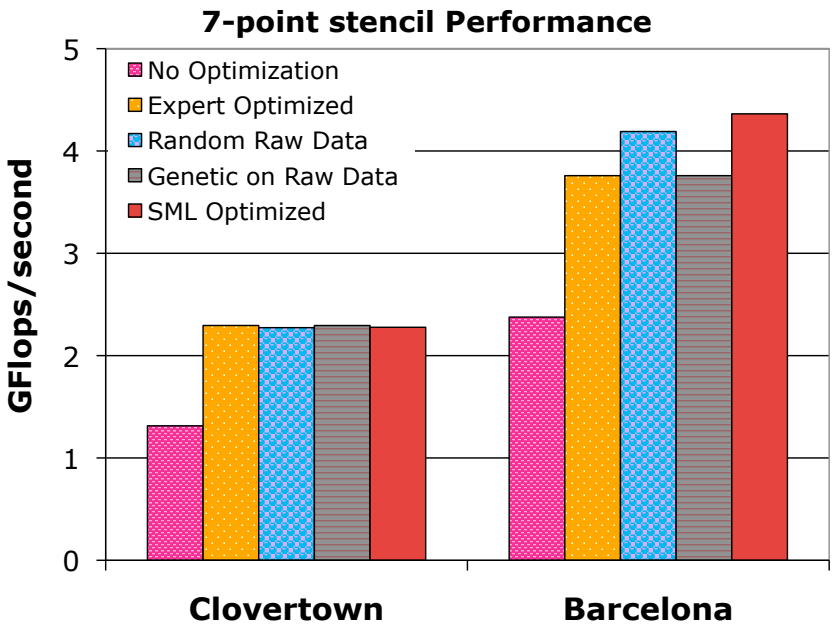

Figure 5.4: Performance results for the 7-point stencil on Intel Clovertown and AMD Barcelona.

- *SML optimized*: This approach uses our KCCA methodology, as described in Section 5.3, to determine the best configuration.

Our primary metric of success is performance, measured in GFLOPs per second. Our performance counters inform us of cycles used for each run; we convert this number to GFLOPs/sec using the following equation:

$$\text{GFLOPs/sec} = \frac{(\text{ClockRate} \times \text{FLOPs})}{\text{Cycles}}$$

Note that the clock rate is 2.66 GHz for Clovertown and 2.30 GHz for Barcelona. The number of FLOPs varies per motif. The 7-point stencil performs 8 FLOPs per point on the 256^3 points and the 27-point stencil performs 30 FLOPs per point on the 256^3 point grid. PGEMM, on the other hand, performs just 2 FLOPs per point on the 768^3 points.

5.4.1 7-Point Stencil Performance

Figure 5.4 compares the results of the various performance optimization techniques on the 7-point stencil on both Clovertown and Barcelona. On Clovertown, our technique provides performance within .02 GFLOPs/sec (1%) of expert optimized. Because the 7-point stencil

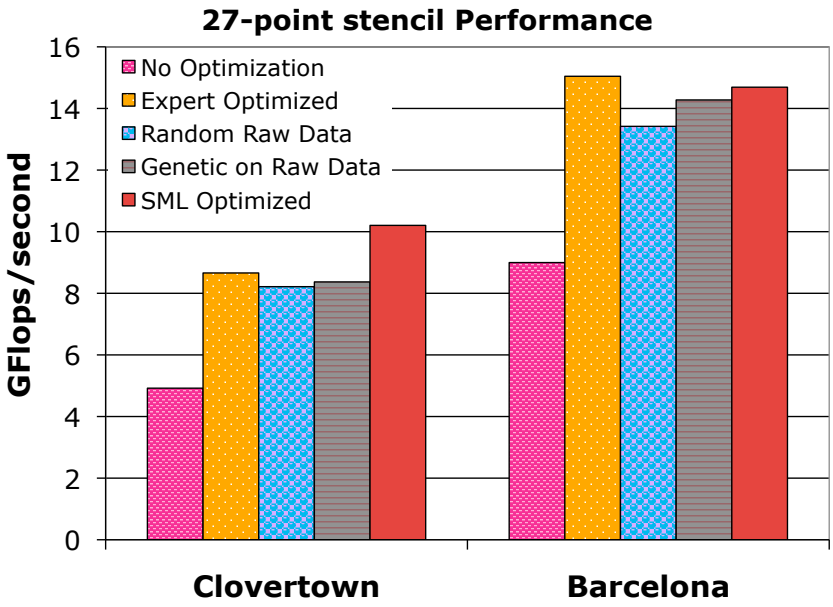

Figure 5.5: Performance results for the 27-point stencil on Intel Clovertown and AMD Barcelona.

is bandwidth bound on the Clovertown, none of the techniques show significant performance gains.

On Barcelona, our performance is 0.6 GFLOPs/sec (16%) better than that achieved by expert optimized for the 7-point stencil. Our inner loop optimization parameters unroll along a different dimension than expert optimized. We also prefetch further ahead than the expert optimized configuration.

5.4.2 27-Point Stencil Performance

Figure 5.5 shows the 27-point stencil's performance optimization results. On Clovertown, our technique provides performance 1.5 GFLOPs/sec (18%) better than expert for the 27-point stencil. This significant performance gain can be attributed to two factors: (i) our domain decomposition parameters more efficiently exploit the 27-point stencil's data locality; (ii) we likely use registers and functional units more efficiently as a result of our inner loop parameter values.

On Barcelona, our performance is within 0.35 GFLOPs/sec (2%) for the 27-point stencil. The dominant factor causing the performance gap is the smaller padding size in the unit stride dimension used by the expert optimized configuration. Furthermore, the expert's configuration uses more efficient domain decomposition.

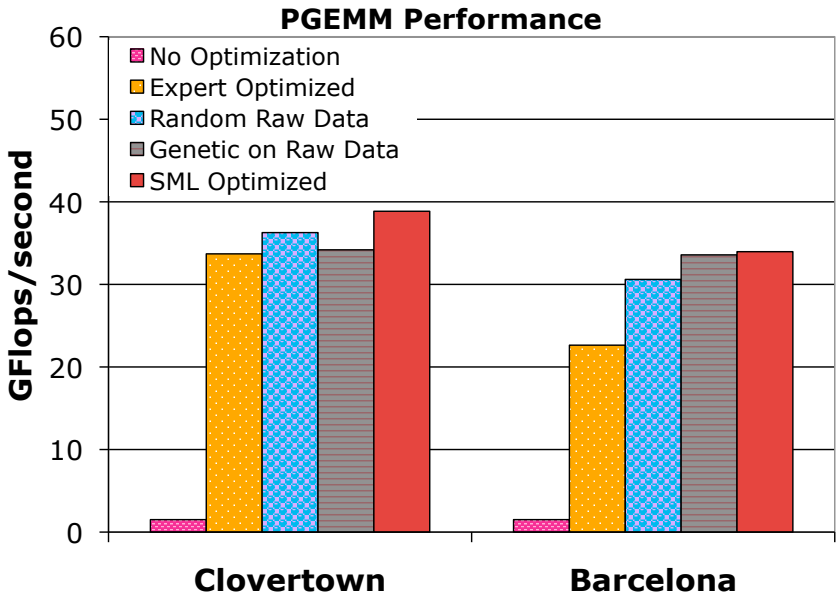

Figure 5.6: Performance results for PGEMM on Intel Clovertown and AMD Barcelona.

5.4.3 PGEMM Performance

Figure 5.6 shows the performance optimization results for running PGEMM on Clovertown and Barcelona. Our technique provides performance 5.2 GFLOPs/sec (15%) better than the expert's configuration on Clovertown. Our choice of register tile size has twice as many rows as the expert's configuration. We also choose a smaller cache block size for the K dimension. Our inner loop parameters unroll the loop fewer times and along a different dimension than the expert. We SIMD-ize the code and prefetch while the expert's configuration performed neither.

On Barcelona, our performance is 11.3 GFLOPs/sec (50%) better than the expert's configuration. Our technique chose the same domain decomposition parameters as the expert's configuration but performed significantly more prefetching compared to the expert's configuration.

5.5 Lessons and Limitations

In this section, we address several open questions about our results and its generalizeability to other motifs and platforms.

Optimization	Stencils		PGEMM	
	Parameters	Total Configs	Parameters	Total Configs
Thread Count	1	4	1	4
Domain Decomposition	4	36	4	256
Software Prefetching	2	18	3	100
Padding	1	32	n/a	n/a
Inner Loop	8	480	2	14
SIMD	n/a	n/a	2	6
Total	**16**	**39,813,120**	**12**	**8,601,600**

Table 5.4: Code optimization categories.

5.5.1 How long does our methodology take?

The last row in Table 5.4 summarizes the total number of configurations in the exhaustive search space for both stencils and PGEMM. At 5 trials per configuration and about 0.08 seconds to run each trial, the total time amounts to 180 days for stencil parameter exhaustive search. Our case study only requires running 1500 randomly chosen configurations. Given our previous assumptions, our runs would complete in 10 minutes; however, we must add the time it takes to build the model (approximately 10 minutes for the 1500 data points) and the time to compute the heuristic and run the suggested configurations (under one minute) - adding up to just over twenty minutes! The timesavings is of comparable scale for PGEMM. Obviously, domain knowledge would help eliminate areas of the search space, which is reflected by our expert-optimized results. However, our methodology is easier to scale to other architectures as well as other optimization problems, such as FFTs and sparse matrix multiplication [Williams *et al.*, 2007], and it can produce superior results.

5.5.2 Can we use the same model to optimize Energy Efficiency?

On most platforms, optimizing for performance also optimizes energy efficiency. However, we want to verify that our model does not produce configurations that result in poor energy efficiency. Based on power meter wattage readings, we calculate energy efficiency with the following formula:

$$\text{Energy Efficiency} = \frac{\text{MFLOPs/sec}}{\text{Watts}}$$

As seen in Figure 5.7, on Clovertown we achieve within 1% of the expert optimized energy efficiency for the 7-point stencil and 13% better than expert optimized for the 27-point stencil. For both our stencil codes on Clovertown and the 27-point stencil code on Barcelona, the best performing configuration is also the most energy efficient. For the 7-

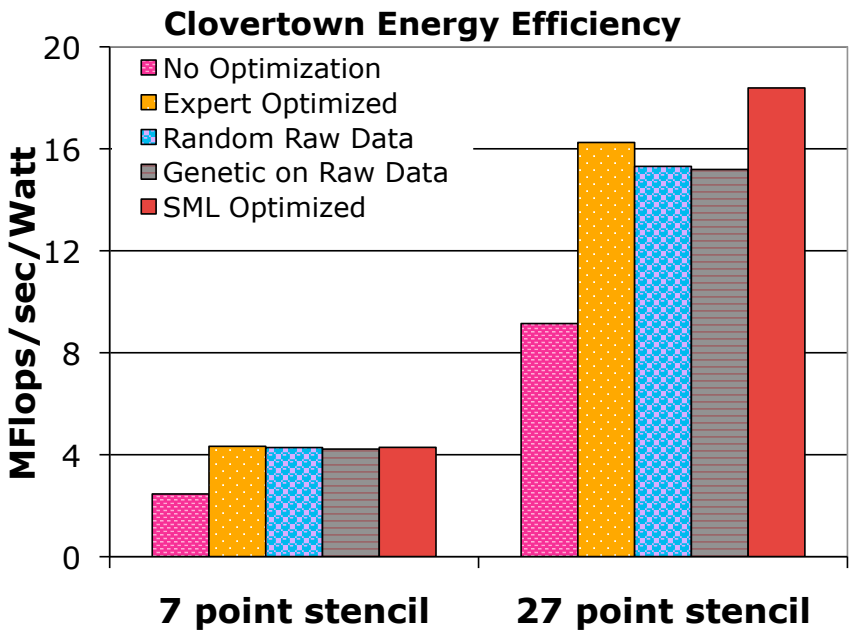

Figure 5.7: Energy efficiency on Clovertown for the two stencils.

point stencil on Barcelona, the energy efficiency of our fastest run differs from the most energy efficient run by a mere 0.3%. As a result, we have omitted the Barcelona energy efficiency graphs. For PGEMM, the relative energy efficiency the SML-optimized and expert-optimized configurations were the same as their relative performance on both Clovertown and Barcelona.

Figure 5.8 compares performance against energy efficiency on the Clovertown. The slope of the graph represents Watts consumed, and since marker shape/color denote thread count, we see that the number of threads used dictates power consumption. We observe configurations with identical performance but differing energy efficiency and vice versa, as highlighted by the oval. In environments with real-time constraints (e.g., portable devices), there is no benefit to completing well before the real-time deadline; but there is significant benefit to conserving battery power. In such environments, performance can be sacrificed for energy efficiency, and thus we expect a wider gap between the two metrics.

5.5.3 Can we measure success without comparing to an expert?

Human experts incorporate architectural and application-specific domain knowledge to identify a good configuration. However, it is increasingly difficult to find people with both types of expertise to guide all our tuning decisions. For instance, our expert optimized config-

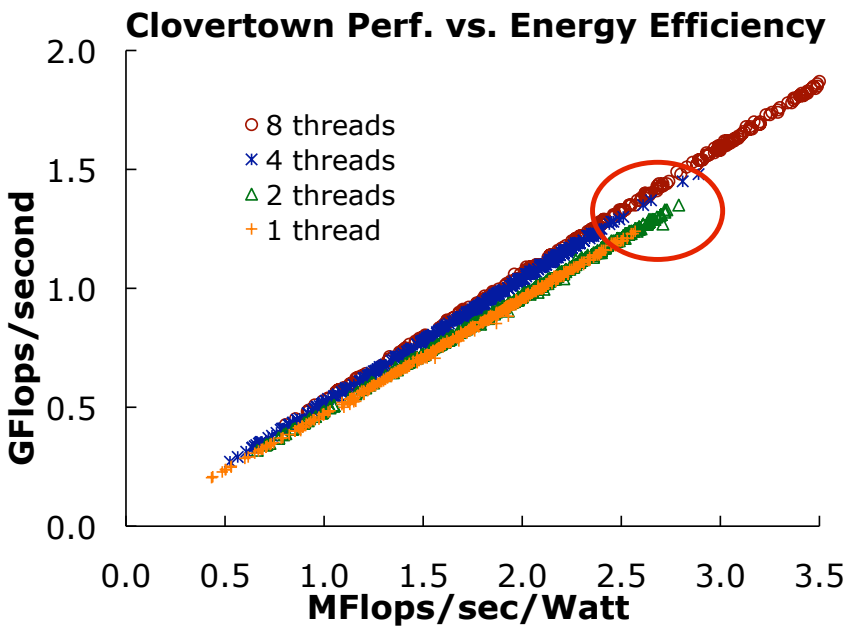

Figure 5.8: Performance vs. energy efficiency for the 7-point stencil training data on Clovertown. Note that the slope indicates Watts consumed. The oval highlights configurations with similar performance but different energy efficiencies.

urations for both the 7-point and 27-point stencils were derived from heuristics suggested by a human expert with extensive knowledge of stencils as well as both Clovertown and Barcelona. However, our PGEMM expert optimized configuration was suggested by a human expert with extensive knowledge about PGEMM but not platform-specific expertise.

Furthermore, the expert's configuration may not reflect the system's true performance upper bound. The Roofline model [Williams *et al.*, 2009] uses architectural specifications and microbenchmarks to calculate peak performance. We can use this model to gauge how close we are to fully exhausting system resources. For example, the Roofline model considers memory bandwidth as a potential bottleneck. This metric is particularly relevant to stencil codes since stencils are relatively bandwidth bound compared to other motifs. Since the PGEMM motif is overwhelmingly compute-bound, Stream bandwidth is not a useful metric for measuring success.

Considering only compulsory misses in our stencil codes, we achieve 95.4% of Stream bandwidth for the 7-point stencil and 13.8% *more* than the Stream bandwidth for the 27-point stencil on the Clovertown platform. We exceed Stream bandwidth for the 27-point stencil because the Stream benchmark is unoptimized. On Barcelona, we achieve 89.2% of Stream bandwidth for the 7-point stencil and 80.1% for the 27-point stencil. Since Barcelona

has a lower FLOP to byte ratio than Clovertown, the stencils are more likely to be compute bound on this platform. As a result, we are unable to achieve a higher fraction of stream bandwidth.

5.5.4 Do simpler algorithms work?

One of the common criticisms of our methodology is that a simpler algorithm would have worked just as well. To address this concern, Figures 5.4, 5.5 and 5.6 include results for two simpler alternatives. The *Random Raw Data* column shows the best performing point in our training data set. The *Genetic on Raw Data* column shows the best-case performance achieved by using a genetic algorithm (combinations) on the top three best-performing points in our training data set. We do not vary the parameters within each optimization. While these two techniques are building blocks of our methodology, individually they do not perform as well as our SML optimized results.

For stencils, we also considered an alternate statistical algorithm in place of KCCA. We built a regression curve on the training data and calculated regression coefficients t_{fit} by solving $K_x * t_{fit} = totalCycles$. We then generated 50,000 configurations and calculated their predicted cycle times using the regression curve. We then ran the top 1000 configurations with shortest predicted cycle times. We evaluated this alternate technique on both the 7-point and 27-point stencils on Clovertown and Barcelona. These regression predicted results did not outperform our KCCA-based configuration for either the 7-point or 27-point configuration on both platforms. For the 7-point stencil on Clovertown and the 27-point stencil on Barcelona, our regression predicted results performed even worse than our *Random Raw Data* configuration. Not only did this technique perform consistently worse than our KCCA-based technique, but it also took longer to run due to running 1000 additional configurations to validate predictions.

5.5.5 Can we achieve good results with fewer counters?

For the 7-point and 27-point stencils, we conducted experiments that built the KCCA model using a subset of the available performance counters on each platform. Figure 5.9 compares the number of counters used to the percentage of peak performance we observed. We always included total cycles in our model so the other counters were never used in isolation. On Clovertown, the 7-point stencil did not show significant advantages to adding several counters. However, varying features to train the 27-point stencil model revealed very unexpected behavior. Training with some subsets of two counters produced better configurations than training with more counters; however, the model trained using all 8 counters demonstrated near-optimal performance. On Barcelona, using total cycles alone performed better than training the model with 2 or 3 counters. However, the peak performance was achieved when using all 6 counters.

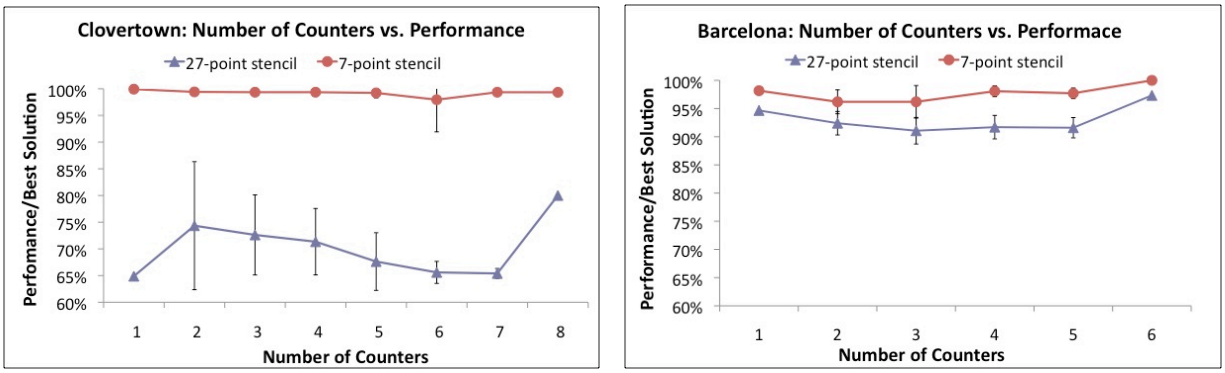

Figure 5.9: Performance results for the 7-point and 27-point stencils on (a) Intel Clovertown and (b) AMD Barcelona as we vary the number of counters we use to train the model. The error bars represent the standard deviation of peak performance.

These results indicate that the choice of an appropriate subset of counters to train with is non-trivial. When in doubt, it is best to use all available counters since the best counters for one motif do not necessarily reflect the best counters for another.

5.5.6 How portable are the best configurations across platforms?

Performance tuning experts often make the claim that code tuned for one platform does not guarantee that it will perform well when run on another platform. To empirically evaluate this claim, we ran each of the three motifs with their best configuration setting for Clovertown on Barcelona, and vice versa. Tables 5.5 and 5.6 show results for this experiment. The results are conclusive that tuning the code for one platform does not guarantee cross-platform performance portability.

Motif	% of best observed Barcelona performance
7-point stencil	98%
27-point stencil	52%
PGEMM	65%

Table 5.5: Results for using best Clovertown configurations on Barcelona.

Motif	% of best observed Clovertown performance
7-point stencil	95%
27-point stencil	79%
PGEMM	71%

Table 5.6: Results for using best Barcelona configurations on Clovertown.

5.5.7 Can our technique benefit from fine-grained power profiling?

One of the limitations of our study is that we measured whole-machine power consumption due to the lack of easily available lower-level power measurements. We can improve energy efficiency by considering each component's individual power consumption rather than only considering whole-machine power consumption. Bird and others [Bird *et al.*, 2010] describe a framework to standardize and expose low-level power and performance metrics. Architectures with dynamic voltage and frequency scaling of either the whole chip or individual cores would also increase the number of tunable parameters we consider.

5.5.8 Can we optimize performance for a composition of motifs?

While we optimized a single scientific motif, complex applications often consist of a composition of these motifs. Optimal configurations for a particular motif do not correspond to optimal configurations for composing that motif with others. The only way to get good performance for a multi-motif application is by co-tuning its motifs on each platform [Mohiyuddin *et al.*, 2009]. We can explore the composition space using the techniques in this chapter by merely changing the training data set.

5.5.9 Can we tune applications for multi-chip servers?

The SML based technique presented in this chapter can be used to tune applications for multi-chip servers. The methodology can be extended to simultaneously optimize both computation on individual (possibly heterogeneous) nodes as well as communication efficiency across the network.

5.5.10 Can our results inform architecture design?

A key issue in system/architecture design is the integration of appropriate instrumentation to expose counters that facilitate decision-making. Evaluating the usefulness of KCCA or other SML algorithm-guided decisions on an experimental system provides valuable feedback on potential areas for improving measurement granularity.

Our successes in using KCCA for auto-tuning have inspired several new use cases for instrumentation-dependent decision making. Building a model of resource availability and application performance can inform resource allocation and cache placement strategies on experimental platforms. Exposing performance counters to the operating system enables research on SML model driven adaptable scheduling algorithms.

5.6 Related Work

Statistical machine learning has been used by the high-performance computing (HPC) community in the past to address performance optimization for simpler problems: Brewer [Brewer, 1995] used linear regression over three parameters to select the best data partitioning scheme for parallelization; Vuduc [Vuduc, 2003] used support vector machines to select the best of three optimization algorithms for a given problem size; and Cavazos and his colleagues [Cavazos *et al.*, 2007] used a logistic regression model to predict the optimal set of compiler flags. All three examples carve out a small subset of the overall tuning space, leaving much of it unexplored. In addition, the above research was conducted prior to the multicore revolution, thus ignoring metrics of merit like energy efficiency.

Recently, Liao and others [Liao *et al.*, 2009] used a variety of machine learning techniques to tune processor prefetch configurations for Datacenters. While they evaluate their optimization configurations on several Datacenter applications, their tunable parameter space contains 4 prefetchers, each of which can be enabled or disabled, resulting in a total possible of 16 configurations. In contrast, our tunable parameter space is orders of magnitude larger.

5.7 Summary

In this chapter, we have demonstrated that SML can quickly identify configurations that simultaneously optimize running time and energy efficiency. SML based auto-tuning methodologies are agnostic to the underlying architecture as well as the code being optimized, resulting in a scalable alternative to human expert-optimization.

As an example, we optimized two stencil codes and a matrix multiplier on two multicore architectures, either nearly matching or outperforming a human expert by up to 50%. The optimization process took about twenty minutes to explore a space that would take half a year to explore exhaustively on a single computer using conventional techniques. This result gives us reason to believe that SML effectively handles the combinatorial explosion posed by the optimization parameters, potentially allowing us to explore some previously-intractable research directions including tuning for a composition of motifs and on multi-chip servers.

Our results to date, and the promise of mapping difficult research problems such as those above onto approaches similar to our own, give us confidence that SML will open up exciting new opportunities to advance the state-of-the-art in multicore performance optimization.

Chapter 6

Reflections and Future Work

Our success stories in the previous three chapters encourage us to explore other use cases for our methodology. This chapter reflects on our experiences and provides a roadmap of future directions for SML-based modeling techniques.

6.1 Most promising future directions for our three case studies

In our case studies in previous chapters, we identified several opportunities for using our prediction/optimization methodology to advance state of the art tools. We revisit a few of the most promising of these opportunities below:

- **Database query optimization:** Query optimizers struggle with predicting the relative performance of different query plans to ultimately choose the best query execution plan. Historically, query optimizers are built using static cost functions that are determined by the estimated selectivity at each phase of a query execution plan. However, predicting selectivity at each step of a query's execution is error-prone, and errors in prediction propagate through the query plan [Ioannidis and Christodoulakis, 1993]. Furthermore, data skew can thwart attempts to parallelize query processing, also compounding prediction errors in parallel databases.

  Our execution time and resource predictions can be complementary to the query optimizer. The query optimizer can use our prediction results to dynamically customize its cost functions and thresholds to the production environment. As a result, the optimizer can make decisions that are informed by workload patterns, hardware configuration and data layout. We believe this research area can advance the state of the art in query optimization.

- **Configuration management in a heterogeneous environment:** In a study of Internet service failures, Oppenheimer and his colleagues identified configuration management to be a key cause of user-visible failures [Oppenheimer *et al.*, 2003]. With the growing complexity and heterogeneity of components in modern systems, automatic configuration management is a key property that these systems must possess to allow seamless scaling and avoid performance degradation. The ability to catalog *working configuration sets*, and consequent system behavior is an indispensible tool to facilitate such configuration management.

  With historical information, we can use KCCA to build a model of components' behavior in reaction to various configuration changes. We can use this model predict whether a hypothetical configuration change is likely to result in degraded performance or erroneous system state.

- **Performance tuning multi-motif applications on multicore platforms:** The multicore computing era has made application performance tuning an essential precursor to achieving good performance. While we demonstrated techniques to efficiently tune single high performance computing motifs per platform, applications are typically constructed from multiple such motifs. Our results from Chapter 5 as well a recent study by Mohiyuddin and others [Mohiyuddin *et al.*, 2009] suggest that performance is not portable across motifs on a particular platform, let alone across different platforms. Consequently, to achieve good performance for an application, we must co-tune the building block motifs of that application on each platform.

  Using our KCCA-based methodology, we can explore the space of optimizations for multi-motif applications by merely changing the data training data and configuration feature vectors. The remainder of our methodology can be reused to identify optimal configuration settings to use for the application.

6.2 Useful side-effects of KCCA-based modeling

In Section 2.3.4, we discussed several advantages of using KCCA for system performance modeling. Recall that KCCA projects kernel matrices of the raw data onto dimensions of maximal correlation. This projection step enables the two additional benefits:

- **Workload characterization:** The co-clustering effect caused by projecting raw data onto dimensions of correlation naturally lends itself to workload characterization. We can run clustering algorithms on the projections of the data to characterize workload simultaneously by correlated workload and performance characteristics. However, the clusters of points in the projection spaces must be interpreted manually or programmatically to provide workload characterization insights. For instance, one must identify

common characteristics among points in a cluster to determine what defines the cluster as a unique subset of the workload.

- **Anonymization of sensitive data:** We discussed the pre-image problem and alternatives to overcome it in Section 2.3.4. A positive outcome of the projection step is that the raw data cannot be reverse engineered from the projections alone. This property allows us to distinguish between *model creation* and *model publishing*. Such a distinction facilitates sharing models across organizational boundaries. With privacy concerns eliminated, it is easier for researchers to use these privacy-preserving models for obtaining high impact results based on realistic and representative data.

6.3 Beyond performance modeling

With minor changes to problem formulation, our modeling technique can be adapted to new use cases:

- **Failure prevention and diagnosis:** The three case studies in this dissertation leverage our SML-based technique for performance prediction and optimization. However, we believe our technique can be reframed to predict system failures and diagnose their root cause. We can build a correlation model between load characteristics of components in a system and observed failures. Such a model can be coupled with our performance prediction framework to anticipate failures a priori and pinpoint potentially failure prone nodes.

- **Forecasting system trends:** We can compare different generations of models for a system, built at different time points or different system deployment versions for instance, to identify system trends over time. Such comparisons can be leveraged to extrapolate future behavior of the system. This ability to *forecast* future trends will prove extremely useful to system developers and operators.

- **Distributed data placement:** Data locality has become a crucial component in distributed system performance. Significant research has gone into moving computation closer to data storage locations. With concise representation of data partitioning and replication policies, we can build performance models that capture relationships between data layout and consequent performance on a variety of workloads. Such models will enable us to determine the best data layout strategy for new system workloads.

- **Statistics-driven code generation:** A compiler must choose the best code variant for a given platform based on architecture-specific resource models. Our statistical modeling techniques can be used to adaptively generate optimal code variants for a

platform based on observed performance of previously executed code variants and currently available resources for code execution. This technique would provide significant improvements in multicore systems where available resources vary based on simultaneously scheduled processes. The compiler can generate several code variants, each with its optimal resource requirements explicitly parameterized, and the OS scheduler can select the code variant to run based on current resource availability.

- **Designing efficient systems:** Our SML-based technique can be used to iteratively improve system design. One example use case for system design is cache placement in a parallel architecture. Typically, architects and system designers explore alternatives using simulators to model performance impact of various design decisions. However, simulators are often as slow as, or slower than, running jobs in a real system. With large design spaces it would save a significant amount of time to explore a subset of the design space and use performance models to extrapolate to un-explored design regions. With instrumentation in place for observing resource consumption and performance of various system modules, our SML techniques can also be used to identify system bottlenecks. We can iteratively increase instrumentation granularity of bottleneck modules to eliminate such inefficiencies.

6.4 New directions for SML research

While the combination of KCCA and nearest neighbors produces several powerful tools for systems, we believe there are unexplored avenues for SML research that could significantly benefit the systems community. We suggest several research directions below:

- **Farthest strangers:** Our performance modeling used kernel functions to determine similarity between data points. Consequently, we are able to use nearest neighbor techniques for prediction and optimization. Several systems problems would benefit from techniques that allow the use of the furthest neighbors, or in other words, *farthest strangers*, to guide resource management decisions. For example, scheduling two jobs that are nearest to each other is likely to produce conflicting resource requirements. However, scheduling two jobs that are distant from one another could produce efficient resource utilization. It would be useful to investigate the statistical merits of such *farthest stranger* techniques.

- **Projecting data onto dimensions of minimal correlation:** KCCA projects data onto dimensions of maximal correlation to produce neighborhoods of similar datapoints. It would be interesting to explore the effects of projecting data onto dimensions of minimal correlation, that is, project data onto correlation dimensions that are represented by lower eigenvalues in KCCA. While such a mechanism would produce a

sparse projection map, it could shed light on how noisy our data is. Such information may prove complementary to existing anomaly detection techniques.

- **Correlating more than two datasets:** In automatically reconfiguring systems, workload patterns affect the chosen configuration, and both of them together impact performance. Recall that KCCA only models paired datasets and we end up grouping workload and configuration together as system input parameters. It would be useful to model three-way relationships among system workload, system configuration, and system performance, preserving dataset isolation.

- **Online KCCA:** To maintain freshness of the KCCA model, it would be useful to make the technique amenable to continuous retraining. This modification would require significant changes to the underlying algorithm. Adding new datapoints perturbs the eigenvectors that form the bases of our projection spaces. It is conceivable that these perturbations can be approximated to reduce the cost of recomputing the entire model.

Chapter 7

Conclusions

In this dissertation, we have shown that several problems in modeling the performance of computer systems can be recast as problems in correlation analysis. To that end, we use Kernel Canonical Correlation Analysis to model the system as a black box, and extract relationships between system workload or configuration parameters and multiple metrics of performance. These relationships are leveraged for performance prediction and optimization. We demonstrated the power of this technique on three different parallel systems. Despite being quite distinct problem classes, our three case studies prove that our statistical machine learning (SML) based methodology provides a high degree of automation, does not require SML expertise to be used, and often, outperforms the state-of-the-art alternatives.

In Chapter 3, we achieved very high prediction accuracy for a variety of Business Intelligence queries on a commercial parallel database system. Using information available prior to query execution, we were able to simultaneously predict multiple metrics including query execution time, records used and message count using a single performance model, with R^2 prediction accuracy values as high as 0.98 where 1.00 would indicate perfect prediction. Our technique predicted equally well for long and short running queries, thus providing more actionable predictions than the built-in query optimizer's cost estimates. Our prediction mechanisms are currently being incorporated into HP's Neoview parallel database to better inform workload management decisions.

Our second proof of concept, in Chapter 4, used the same methodology for predicting performance of Hadoop jobs in a production cluster at Facebook. We achieved prediction accuracy as high as 0.93 for multiple metrics when simultaneously predicting execution time, map time, reduce time and bytes written for two data analytics workloads at Facebook. Our prediction results are currently being used to drive a statistics-driven Hadoop job scheduler and workload generator.

In Chapter 5, the third success story for our methodology was for optimizing performance of high performance computing motifs on multicore platforms. Using our SML-based technique, we matched within 2% of, and in many cases, outperformed performance results of an

architecture and application domain expert by up to 50% for tuning a variety of motifs on two multicore architectures. Our technique required minimal domain knowledge and completed in approximately 20 minutes, which is orders of magnitude faster than a six month long exhaustive search. The scalability and efficiency of our technique opens up opportunities for previously intractable research directions for multicore performance tuning.

Our successes to date suggest that our SML-based technique provides powerful tools for managing and analyzing state-of-the-art systems. We expect our methodology to have scope beyond the success stories presented in this dissertation. We believe that SML should join queueing theory in the essential toolkit that systems researchers and practitioners should know.

Bibliography

[Albanesius, 2009] Chloe Albanesius. Amazon Tops Black Friday Retail Web Traffic. http://www.pcmag.com/article2/0,2817,2356397,00.asp, 2009.

[Applied Numerical Algorithms Group, 2009] Applied Numerical Algorithms Group. Chombo: Adaptive Mesh Refinement Library. http://seesar.lbl.gov/ANAG/software.html, 2009.

[Armbrust *et al.*, 2009] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[Asanovic *et al.*, 2006] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick. The Landscape of Parallel Computing Research: A View from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.

[Bach and Jordan, 2003] Francis R. Bach and Michael I. Jordan. Kernel Independent Component Analysis. *Journal of Machine Learning Research*, 3:1–48, 2003.

[Barroso *et al.*, 2003] Luiz Andr Barroso, Jeffrey Dean, and Urs Hlzle. Web Search for a Planet: The Google Cluster Architecture. *IEEE Micro*, 23(2):22–28, 2003.

[Berger and Oliger, 1984] M. Berger and J. Oliger. Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations. *Journal of Computational Physics*, 53:484–512, 1984.

[Bird *et al.*, 2010] Sarah Bird, Kaushik Datta, Karl Fuerlinger, Archana Ganapathi, Shoaib Kamil, Rajesh Nishtala, David Skinner, Andrew Waterman, Samuel Williams, Krste Asanović, and David Patterson. Software Knows Best: A Case for Hardware Transparency and Measurability (in submission). In *37th International Symposium on Computer Architecture (ISCA 2010)*, 2010.

[Bishop, 2006] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[Bodík *et al.*, 2008] Peter Bodík, Moisés Goldszmidt, and Armando Fox. HiLighter: Automatically Building Robust Signatures of Performance Behavior for Small- and Large-Scale Systems. In *SysML*, 2008.

[Brewer, 1995] Eric A. Brewer. High-Level Optimization via Automated Statistical Modeling. In *PPOPP '95: Proceedings of the fifth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 80–91, New York, NY, USA, 1995. ACM.

[Cavazos *et al.*, 2007] John Cavazos, Grigori Fursin, Felix Agakov, Edwin Bonilla, Michael F. P. O'Boyle, and Olivier Temam. Rapidly Selecting Good Compiler Optimizations using Performance Counters. In *CGO '07: Proceedings of the International Symposium on Code Generation and Optimization*, pages 185–197, Washington, DC, USA, 2007. IEEE Computer Society.

[Chaudhuri *et al.*, 2004] Surajit Chaudhuri, Vivek Narasayya, and Ravishankar Ramamurthy. Estimating Progress Of Execution For SQL Queries. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 803–814, New York, NY, USA, 2004. ACM Press.

[Chaudhuri *et al.*, 2005] Surajit Chaudhuri, Raghav Kaushik, and Ravishankar Ramamurthy. When Can We Trust Progress Estimators For SQL Queries? In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pages 575–586, New York, NY, USA, 2005. ACM Press.

[Cohen *et al.*, 2005] Ira Cohen, Steve Zhang, Moisés Goldszmidt, Julie Symons, Terence Kelly, and Armando Fox. Capturing, Indexing, Clustering, and Retrieving System History. In *SOSP*, pages 105–118, 2005.

[Condie *et al.*, 2009] Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, Khaled Elmeleegy, and Russell Sears. MapReduce Online. Technical Report UCB/EECS-2009-136, EECS Department, University of California, Berkeley, Oct 2009.

[Datta *et al.*, 2008] Kauskik Datta, Mark Murphy, Vasily Volkov, Samuel Williams, Jonathan Carter, Leonid Oliker, David Patterson, John Shalf, and Katherine Yelick. Stencil Computation Optimization and Auto-tuning on State-of-the-Art Multicore Architectures. In *Supercomputing*, November 2008.

[Dean and Ghemawat, 2004] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[Dean and Ghemawat, 2008] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM (CACM)*, 51(1):107–113, 2008.

[Demmel *et al.*, 2005] James Demmel, Jack Dongarra, Victor Eijkhout, Erika Fuentes, Antoine Petitet, Richard Vuduc, R. Clint Whaley, and Katherine Yelick. Self Adapting Linear Algebra Algorithms and Software. In *Proceedings of the IEEE: Special Issue on Program Generation, Optimization, and Adaptation*, volume 93, February 2005.

[Diao *et al.*, 2001] Yixin Diao, Joseph L. Hellerstein, and Sujay Parekh. Stochastic Modeling of Lotus Notes with a Queueing Model. In *Proc. Computer Measurement Group Int. Conference (CMG01)*, 2001.

[Doyle *et al.*, 2003] Ronald P. Doyle, Jeffrey S. Chase, Omer M. Asad, Wei Jin, and Amin M. Vahdat. Model-Based Resource Provisioning in a Web Service Utility. In *USITS'03: Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*, pages 5–5, Berkeley, CA, USA, 2003. USENIX Association.

[Elnaffar *et al.*, 2002] Said Elnaffar, Pat Martin, and Randy Horman. Automatically Classifying Database Workloads. In *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management*, pages 622–624, New York, NY, USA, 2002. ACM.

[Frigo and Johnson, 2005] Matteo Frigo and Steven G. Johnson. The Design and Implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[Galindo-Legaria *et al.*, 1994] C. Galindo-Legaria, J. Pellenkoft, M. L. Kersten, Arjan Pellenkoft, and Martin Kersten. Fast, Randomized Join-Order Selection - Why Use Transformations? In *In Proc. of the Conf. on Very Large Data Bases (VLDB*, pages 85–95, 1994.

[Ganapathi *et al.*, 2009a] Archana Ganapathi, Yanpei Chen, Armando Fox, Randy H. Katz, and David A. Patterson. Statistics-Driven Workload Modeling for the Cloud. Technical Report UCB/EECS-2009-160, EECS Department, University of California, Berkeley, Nov 2009.

[Ganapathi *et al.*, 2009b] Archana Ganapathi, Kaushik Datta, Armando Fox, and David Patterson. A Case for Machine Learning to Optimize Multicore Performance. In *HotPar '09: Proceedings of the Workshop on Hot Topics in Parallelism*. USENIX, March 2009.

[Ganapathi *et al.*, 2009c] Archana Ganapathi, Harumi Kuno, Umeshwar Daval, Janet Wiener, Armando Fox, Michael Jordan, and David Patterson. Predicting Multiple Performance Metrics for Queries: Better Decisions Enabled by Machine Learning. 2009.

[Ganger and Patt, 1998] Gregory R. Ganger and Yale N. Patt. Using System-Level Models to Evaluate I/O Subsystem Designs. *IEEE Transactions on Computers*, 47(6):667–678, 1998.

[Ghemawat *et al.*, 2003] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. *SIGOPS Operating Systems Review*, 37(5):29–43, 2003.

[Goldszmidt *et al.*, 2005] Moises Goldszmidt, Ira Cohen, Armando Fox, and Steve Zhang. Three Research Challenges at the Intersection of Machine Learning, Statistical Induction, and Systems. In *HOTOS'05: Proceedings of the 10th conference on Hot Topics in Operating Systems*, pages 10–10, Berkeley, CA, USA, 2005. USENIX Association.

[Graefe and McKenna, 1993] Goetz Graefe and William J. McKenna. The Volcano Optimizer Generator: Extensibility and Efficient Search. In *Proceedings of the Ninth International Conference on Data Engineering, April 19-23, 1993, Vienna, Austria*, pages 209–218. IEEE Computer Society, 1993.

[Had, a] Hadoop. http://hadoop.apache.org.

[Had, b] Hadoop Power-By Page. http://wiki.apache.org/hadoop/PoweredBy.

[Hellerstein *et al.*, 2004] Joseph L. Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[Hindman *et al.*, 2009] Benjamin Hindman, Andrew Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Scott Shenker, and Ion Stoica. Nexus: A Common Substrate for Cluster Computing. HotCloud 2009: Workshop on Hot Topics in Cloud Computing, 2009.

[Hoelzle and Barroso, 2009] Urs Hoelzle and Luiz Andre Barroso. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 2009.

[Hotelling, 1933] H. Hotelling. Analysis of a Complex of Statistical Variables into Principal Components. *Journal of Educational Psychology*, 24:417–441; 498–520, 1933.

[Hotelling, 1936] H. Hotelling. Relations Between Two Sets of Variates. *Biometrika*, 28:321–377, 1936.

[Huang *et al.*, 2007] Ling Huang, Xuanlong Nguyen, Minos Garofalakis, Michael Jordan, Anthony D. Joseph, and Nina Taft. In-Network PCA and Anomaly Detection. Technical Report UCB/EECS-2007-10, EECS Department, University of California, Berkeley, Jan 2007.

[Ibaraki and Kameda, 1984] Toshihide Ibaraki and Tiko Kameda. On the Optimal Nesting Order for Computing N-Relational Joins. *ACM Transactions on Database Systems*, 9(3):482–502, 1984.

[Ioannidis and Christodoulakis, 1993] Yannis E. Ioannidis and Stavros Christodoulakis. Optimal Histograms for Limiting Worst-Case Error Propagation in the Size of Join Results. *ACM Transactions on Database Systems*, 18(4):709–748, 1993.

[Ioannidis, 1996] Yannis E. Ioannidis. *Query Optimization*. 1996.

[Keeton *et al.*, 1998] Kimberly Keeton, David A. Patterson, Yong Qiang He, Roger C. Raphael, and Walter E. Baker. Performance Characterization of a Quad Pentium Pro SMP using OLTP Workloads. In *ISCA*, pages 15–26, 1998.

[Krompass *et al.*, 2007] Stefan Krompass, Harumi Kuno, Umeshwar Dayal, and Alfons Kemper. Dynamic Workload Management for Very Large Data Warehouses: Juggling Feathers and Bowling Balls. In *in Proceedings of the 33rd Conference on Very Large Database (VLDB)*, 2007.

[Kwok and Tsang, 2003] James T. Kwok and Ivor W. Tsang. The Pre-Image Problem in Kernel Methods. In *ICML*, pages 408–415, 2003.

[Lazowska *et al.*, 1984] Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1984.

[Lemos, 2009] Robert Lemos. Another Data Center Headache: Log Data Exploding. *CIO Magazine*, April 2009.

[Liao *et al.*, 2009] S. Liao, T. Hung, D. Nguyen, C. Chou, C. Tu, and H. Zhou. Machine Learning-Based Prefetch Optimization for Data Center Applications. In *Supercomputing*, November 2009.

[Liblit *et al.*, 2005] Ben Liblit, Mayur Naik, Alice X. Zheng, Alex Aiken, and Michael I. Jordan. Scalable Statistical Bug Isolation. In *PLDI '05: Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, pages 15–26, New York, NY, USA, 2005. ACM.

[Lo *et al.*, 1998] Jack L. Lo, Luiz André Barroso, Susan J. Eggers, Kourosh Gharachorloo, Henry M. Levy, and Sujay S. Parekh. An Analysis of Database Workload Performance on Simultaneous Multithreaded Processors. In *ISCA*, pages 39–50, 1998.

[Lu *et al.*, 2002] Chenyang Lu, Guillermo A. Alvarez, and John Wilkes. Aqueduct: Online Data Migration with Performance Guarantees. In *FAST '02: Proceedings of the 1st USENIX Conference on File and Storage Technologies*, page 21, Berkeley, CA, USA, 2002. USENIX Association.

[Luo *et al.*, 2004] Gang Luo, Jeffrey F. Naughton, Curt J. Ellmann, and Michael W. Watzke. Toward a Progress Indicator For Database Queries. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 791–802, New York, NY, USA, 2004. ACM Press.

[Luo *et al.*, 2005] Gang Luo, Jeffrey F. Naughton, Curt J. Ellmann, and Michael W. Watzke. Increasing The Accuracy and Coverage of SQL Progress Indicators. In *ICDE '05: Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*, pages 853–864, Washington, DC, USA, 2005. IEEE Computer Society.

[Luo *et al.*, 2006] Gang Luo, Jeffrey F. Naughton, and Philip S. Yu. Multi-Query SQL Progress Indicators. In *Advances in Database Technology - EDBT 2006, 10th International Conference on Extending Database Technology*, pages 921–941, 2006.

[Macqueen, 1967] J. Macqueen. Some Methods for Classification and Analysis of Multivariate Observations. In L. M. Le Cam and J. Neyman, editors, *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1 of *Berkeley: University of California Press*, pages 281–297, 1967.

[Markl and Lohman, 2002] Volker Markl and Guy Lohman. Learning Table Access Cardinalities With LEO. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 613–613, New York, NY, USA, 2002. ACM.

[Massie *et al.*, 2003] Matthew L. Massie, Brent N. Chun, and David E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation And Experience. *Parallel Computing*, 30:2004, 2003.

[Mohiyuddin *et al.*, 2009] Marghoob Mohiyuddin, Mark Murphy, Leonid Oliker, John Shalf, John Wawrzynek, and Samuel Williams. A Design Methodology for Domain-Optimized Power-Efficient Supercomputing. In *Supercomputing*. ACM, 2009.

[Morton *et al.*, 2010] Kristi Morton, Abram Friesen, Magdalena Balazinska, and Dan Grossman. Estimating the Progress of MapReduce Pipelines. In *Proc International Conference on Data Engineering (ICDE)*, March 2010.

[Nagappan and Ball, 2007] Nachiappan Nagappan and Thomas Ball. Using Software Dependencies and Churn Metrics to Predict Field Failures: An Empirical Case Study. *Empirical Software Engineering and Measurement, International Symposium on*, 0:364–373, 2007.

[Non, 1989] Tandem Database Group - NonStop SQL: A Distributed, High-Performance, High-Availability Implementation of SQL. In *Proceedings of the 2nd International Workshop on High Performance Transaction Systems*, pages 60–104, London, UK, 1989. Springer-Verlag.

[Olston *et al.*, 2008] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig Latin: A Not-So-Foreign Language for Data Processing. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110, New York, NY, USA, 2008. ACM.

[Oppenheimer *et al.*, 2003] David Oppenheimer, Archana Ganapathi, and David A. Patterson. Why Do Internet Services Fail, And What Can Be Done About It? In *USITS'03: Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*, pages 1–1, Berkeley, CA, USA, 2003. USENIX Association.

[Othayoth and Poess, 2006] Raghunath Othayoth and Meikel Poess. The Making of TPC-DS. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 1049–1058. VLDB Endowment, 2006.

[PAPI, 2009] PAPI. Performance Application Programming Interface. http://icl.cs.utk.edu/papi/, 2009.

[Pavlo *et al.*, 2009] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. Dewitt, Samuel Madden, and Michael Stonebraker. A Comparison of Approaches to Large-Scale Data Analysis. In *SIGMOD '09: Proceedings of the 2009 ACM SIGMOD International Conference*. ACM, June 2009.

[Perl and Weihl, 1993] Sharon E. Perl and William E. Weihl. Performance Assertion Checking. *SIGOPS Operating Systems Review*, 27(5):134–145, 1993.

[Reynolds *et al.*, 2006] Patrick Reynolds, Charles Killian, Janet L. Wiener, Jeffrey C. Mogul, Mehul A. Shah, and Amin Vahdat. Pip: Detecting the Unexpected in Distributed Systems. In *NSDI'06: Proceedings of the 3rd conference on Networked Systems Design & Implementation*, pages 9–9, Berkeley, CA, USA, 2006. USENIX Association.

[Sevcik, 1981] Kenneth C. Sevcik. Database System Performance Prediction Using an Analytical Model (invited paper). In *VLDB '1981: Proceedings of the seventh international conference on Very Large Data Bases*, pages 182–198. VLDB Endowment, 1981.

[Shawe-Taylor and Cristianini, 2004] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004.

[Shekita *et al.*, 1993] Eugene J. Shekita, Honesty C. Young, and Kian-Lee Tan. Multi-Join Optimization for Symmetric Multiprocessors. In *VLDB '93: Proceedings of the 19th International Conference on Very Large Data Bases*, pages 479–492, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[Shen *et al.*, 2005] Kai Shen, Ming Zhong, and Chuanpeng Li. I/O System Performance Debugging Using Model-Driven Anomaly Characterization. In *FAST'05: Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies*, pages 23–23, Berkeley, CA, USA, 2005. USENIX Association.

[Stankovic *et al.*, 2001] John A. Stankovic, Tian He, Tarek Abdelzaher, Mike Marley, Gang Tao, Sang Son, and Cenyan Lu. Feedback Control Scheduling in Distributed Real-Time Systems. In *RTSS '01: Proceedings of the 22nd IEEE Real-Time Systems Symposium*, page 59, Washington, DC, USA, 2001. IEEE Computer Society.

[Stillger *et al.*, 2001] Michael Stillger, Guy M. Lohman, Volker Markl, and Mokhtar Kandil. LEO - DB2's LEarning Optimizer. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, pages 19–28, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[Storm *et al.*, 2006] Adam J. Storm, Christian Garcia-Arellano, Sam S. Lightstone, Yixin Diao, and M. Surendra. Adaptive Self-Tuning Memory in DB2. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 1081–1092. VLDB Endowment, 2006.

[Swami and Iyer, 1993] Arun N. Swami and Balakrishna R. Iyer. A Polynomial Time Algorithm for Optimizing Join Queries. In *Proceedings of the Ninth International Conference on Data Engineering*, pages 345–354, Washington, DC, USA, 1993. IEEE Computer Society.

[Thereska and Ganger, 2008] Eno Thereska and Gregory R. Ganger. Ironmodel: Robust Performance Models in the Wild. *SIGMETRICS Performance Evaluation Review*, 36(1):253–264, 2008.

[Thusoo *et al.*, 2009] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive - A Warehousing Solution Over a MapReduce Framework. In *VLDB '09: Proceedings of the 35th International Conference on Very Large Data Bases*, New York, NY, USA, 2009. ACM.

[Urgaonkar *et al.*, 2005] Bhuvan Urgaonkar, Giovanni Pacifici, Prashant Shenoy, Mike Spreitzer, and Asser Tantawi. An Analytical Model for Multi-tier Internet Services and its Applications. *SIGMETRICS Performance Evaluation Review*, 33(1):291–302, 2005.

[Uysal *et al.*, 2001] Mustafa Uysal, Guillermo A. Alvarez, and Arif Merchant. A Modular, Analytical Throughput Model for Modern Disk Arrays. In *MASCOTS '01: Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, page 183, Washington, DC, USA, 2001. IEEE Computer Society.

[Vuduc *et al.*, 2005] Richard Vuduc, James W. Demmel, and Katherine A. Yelick. OSKI: A Library of Automatically Tuned Sparse Matrix Kernels. In *Proceedings of SciDAC 2005*, Journal of Physics: Conference Series, San Francisco, CA, USA, June 2005. Institute of Physics Publishing.

[Vuduc, 2003] Richard Vuduc. *Automatic Performance Tuning of Sparse Matrix Kernels*. PhD thesis, UC Berkeley, 2003.

[Wang *et al.*, 2004] Mengzhi Wang, Kinman Au, Anastassia Ailamaki, Anthony Brockwell, Christos Faloutsos, and Gregory R. Ganger. Storage Device Performance Prediction with CART Models. *SIGMETRICS Perform*, 32(1):412–413, 2004.

[Whaley *et al.*, 2001] R. Clint Whaley, Antoine Petitet, and Jack Dongarra. Automated Empirical Optimization of Software and the ATLAS Project. *Parallel Computing*, 27(1-2):3–35, 2001.

[Williams *et al.*, 2007] Samuel Williams, Leonid Oliker, Richard Vuduc, John Shalf, Katherine Yelick, and James Demmel. Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms. In *Supercomputing*, November 2007.

[Williams *et al.*, 2009] S. Williams, Andrew Waterman, and David A. Patterson. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Communications of the ACM (CACM)*, 52(4):65–76, April 2009.

[Xu *et al.*, 2006] Wei Xu, Xiaoyun Zhu, S. Singhal, and Zhikui Wang. Predictive Control for Dynamic Resource Allocation in Enterprise Data Centers. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, pages 115–126, 2006.

[Xu *et al.*, 2009] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. Large-Scale System Problem Detection by Mining Console Logs. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles 2009, SOSP 2009*, 2009.

[Yoo and Lafortune, 1989] H. Yoo and S. Lafortune. An Intelligent Search Method for Query Optimization by Semijoins. *IEEE Transactions on Knowledge and Data Engineering*, 1(2):226–237, 1989.

[Yu *et al.*, 1992] Philip S. Yu, Ming-Syan Chen, Hans-Ulrich Heiss, and Sukho Lee. On Workload Characterization of Relational Database Environments. *Software Engineering*, 18(4):347–355, 1992.

[Zaharia *et al.*, 2008] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, and Ion Stoica. Improving MapReduce Performance in Heterogeneous Environments. In *OSDI 2008: 8th USENIX Symposium on Operating Systems Design and Implementation*. USENIX, 2008.

[Zaharia *et al.*, 2009] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Job Scheduling for Multi-User MapReduce Clusters. Technical Report UCB/EECS-2009-55, EECS Department, University of California, Berkeley, Apr 2009.

[Zhang *et al.*, 2005a] Ning Zhang, Peter J. Haas, Vanja Josifovski, Guy M. Lohman, and Chun Zhang. Statistical Learning Techniques for Costing XML Queries. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 289–300. VLDB Endowment, 2005.

[Zhang *et al.*, 2005b] Steve Zhang, Ira Cohen, Julie Symons, and Armando Fox. Ensembles of Models for Automated Diagnosis of System Performance Problems. In *DSN '05: Proceedings of the 2005 International Conference on Dependable Systems and Networks*, pages 644–653, Washington, DC, USA, 2005. IEEE Computer Society.