
A Dollar from 15 Cents: Cross-Platform Management for Internet Services

Christopher Stewart (Univ. of Rochester)

Terence Kelly (HP Labs)

Kai Shen (Univ. of Rochester)

Alex Zhang (HP Labs)

Motivation

1. Internet services are increasingly popular
 - Customer-facing websites— e.g., Amazon and BestBuy.com
 - Essential enterprise applications— e.g., SAP and Salesforce.com
2. Data centers are increasingly complex
 - Comprise heterogeneous processor platforms
 - Energy costs affect ability to scale

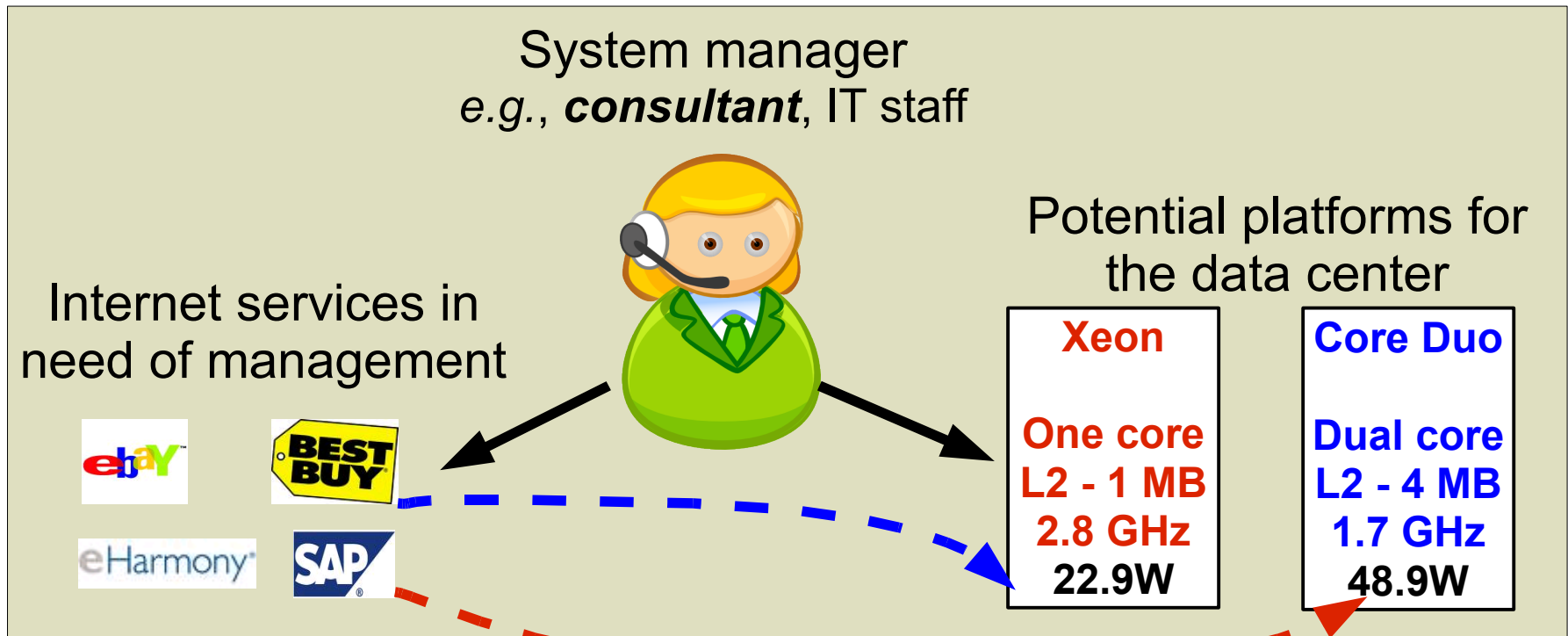
Decisions that span multiple platforms are commonplace in the data centers of almost every firm in every industry

One goal of such **cross-platform management** is to achieve *high performance at low cost*

Challenges for Cross-Platform Management in the Real-World

- Performance of an Internet service on a particular platform is hard to predict
 - Platform configurations like cache sizes and latencies, and number of cores have complex performance effects
 - Fluctuating workloads of Internet services place different demands on processors
- Data available for making decisions is constrained
 - Combination of security, trust, and politics
 - Are you a consultant? Hosting service? IT technician?
 - Invasive methods for system measurement are not allowed

Motivating Example



Challenge #1: Which processor is most cost-effective for each service (*i.e.*, maximize throughput-per-watt)?

Constraints in Production

Experiment with non-production copies of each application.
[nagaraja-osdi-2004]

Not Allowed

Instrument OS or middleware.
[barham-osdi-2004, stewart-nsdi-2005]

Not Allowed



Xeon

One core
L2 - 1 MB
2.8 GHz
22.9W

Core Duo

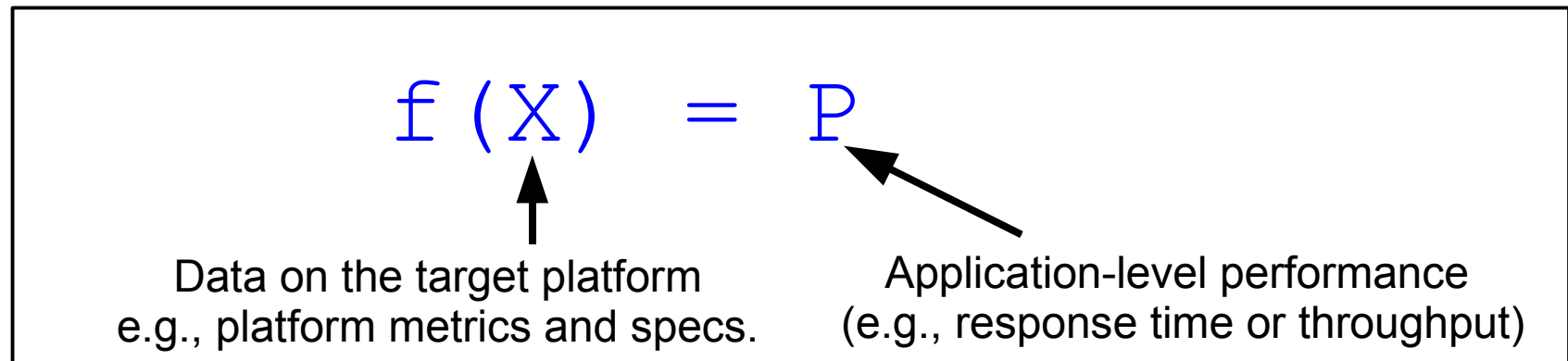
Dual core
L2 - 4 MB
1.7 GHz
48.9W

Challenge #2: Use only the limited data that can be collected non-intrusively with standard tools.

In the words of Tupac Shakur and the Digital Underground,
“trying to make a dollar from 15 cents”

Making A Dollar from 15 Cents

Develop a cross-platform performance model using only data commonly available in production



- Convert seemingly useless production data into a surprisingly accurate guide on the performance consequences of cross-platform decisions

Our Cross-Platform Performance Model

Key concepts:

Trait models— sub-models derived from empirical observations of a production system

Existing expert knowledge— textbooks and sound theory on the structure of both processors and Internet services

Key principle:

Derive trait models from production data for hard-to-characterize platform parameters and use expert knowledge to compose traits for performance prediction

Outline

1. Motivation

2. Trait models— *empirically observed building blocks*

- Identification and calibration
- Example trait models

3. Our cross-platform model— *composition of trait models*

4. A Dollar from 15 cents— *cross-platform management*

5. Future work and conclusion

Trait Models

Parsimonious sub-models that characterize only one aspect of a complex system (*like personality traits*)

Input: One platform configuration or workload property

Output: A processor metric— *e.g.*, cache misses or instructions

- Benefits of intentional simplicity
 - Easy to identify empirically— **reduce a-priori knowledge**
 - Few free parameters— **calibrate with little data**

Deriving Trait Models

A trait model is the simplest functional form that yields low error for observed pairs of input parameters and output metrics

▫ Error metric is normalized residual error: $Err = \frac{|pred - act|}{act}$

■ Empirical models can be misleading?

Rigorous validation steps

1. “Makes sense” based on system design principles
2. Observed for several applications/workloads
3. Practical calibration requirements

Trait Model #1

Input: Cache size (S)

Output: Cache misses (M)

Find the simplest functional form with low error on observed data

Empirical Data from RUBiS

Cache Size	Cache Misses per (10K instr.)
16K	10.21
32	9.36
512	5.72
2048	4.53
4096	3.26
16384	1.25

Same workload for each test. Adequate resource provisioning

Functional form	Median Error
-----------------	--------------

Linear	0.73
--------	------

Exponential	0.14
-------------	------

Logarithmic	0.09
-------------	------

Power law	0.02
-----------	------


$$M = A \times S^B$$

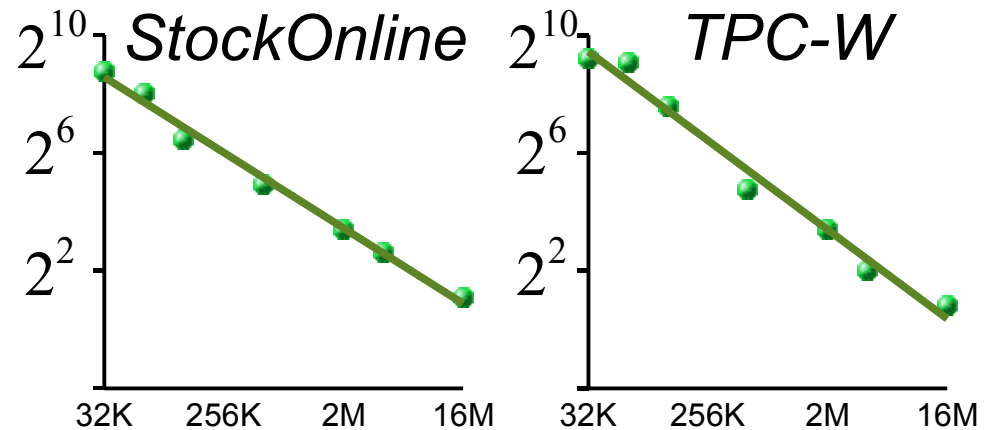
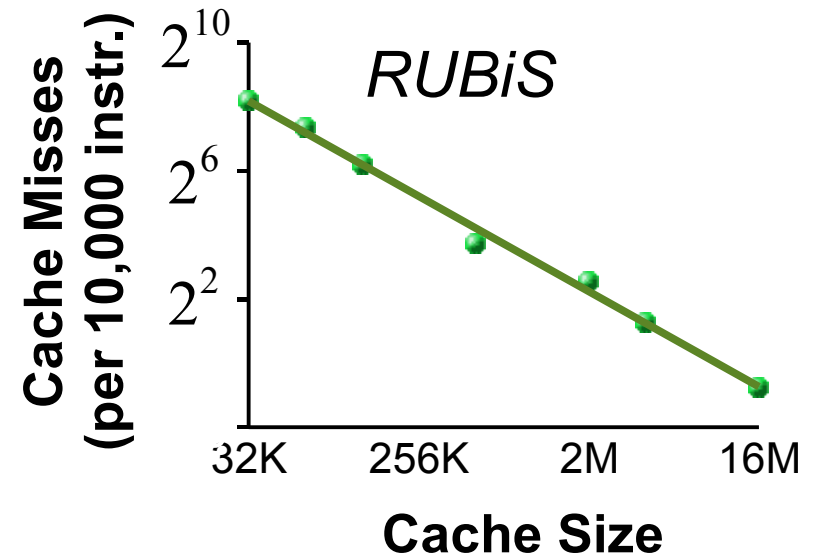
Trait Model #1 Validation

Input: Cache size (S)

Output: Cache misses (M)

Relationship: $M = A \times S^B$

- 1. General:** Observed across 3 applications
- 2. Makes Sense:** Heavy-tail reflects hard-to-remove cache misses
- 3. Practical:** Calibrate with observations of cache misses on just 2 cache sizes



Trait Model #2

Input: Request-mix vector
($\langle N_1, N_2, N_k \rangle$)

Output: Aggregate Instr. (I)

Empirical Data			
Request Mix			Aggr.
N_1	$N_2 \dots$	N_k	Instr.
18	42	Y_1	280M
33	36	Y_2	311M

Relationship: $I = \sum_{type\ j} \alpha_k \cdot N_k$

1. **General:** 3 applications and more than 500 request mixes

<i>Median Error</i>	
RUBiS	0.023
StockOnline	0.033
TPCW	0.025

2. **Makes Sense:** Request type influences code path

3. **Practical:** Nonstationarity [stewart-eurosys-2007] enables calibration from Apache logs

Outline

1. Motivation

2. Trait models— *empirically observed building blocks*

3. Our cross-platform model— *composition of trait models*

- Service time prediction
- Response time prediction
- Evaluate accuracy and compare to alternatives

4. A Dollar from 15 cents— *cross-platform management*

5. Future work and conclusion

Composition Principles

- Expert knowledge about processors and Internet services
 1. Instruction and memory-access latencies are key components of the time spent processing end-user requests
[hennessey & patterson] [karkhanis-isca-2004]
 2. Response time can be predicted using queuing models
[doyle-usits-03] [stewart-nsdi-05] [stewart-eurosys-07]
- Rigorous evaluation validates composition
 - We target the business-logic tier where processor configurations affect performance the most

Expert Knowledge on Processor Design

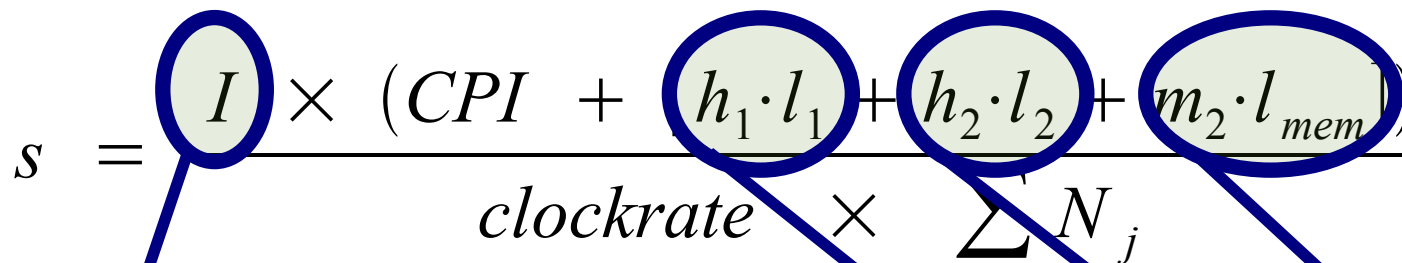
Average CPU service time for a particular mix of
business-logic requests [hennessey & patterson]

$$s = \frac{I \times (CPI + [h_1 \cdot l_1 + h_2 \cdot l_2 + m_2 \cdot l_{mem}])}{clockrate \times \sum N_j}$$

- I → Aggregate instructions count
- CPI → Average number of cycles to execute an instruction (after registers are filled)
- h and l → accesses and latency in level k cache
- Clock rate → cycles per second available on the machine
- N_j → number of requests of type j

Incorporating Trait Models

- Prediction across cache sizes and request mixes

$$s = \frac{I \times (CPI + h_1 \cdot l_1 + h_2 \cdot l_2 + m_2 \cdot l_{mem})}{clockrate \times \sum N_j}$$


Trait model #2
(in paper)

$$I = \sum_{type\ j} \alpha_j \cdot N_j$$

$$M_i = \sum_{type\ j} \alpha_{ij} \cdot N_j$$

Trait model #1

$$M = A \times S^B$$

- Accesses at level i equals misses at i-1

Expert Knowledge on Queuing Models

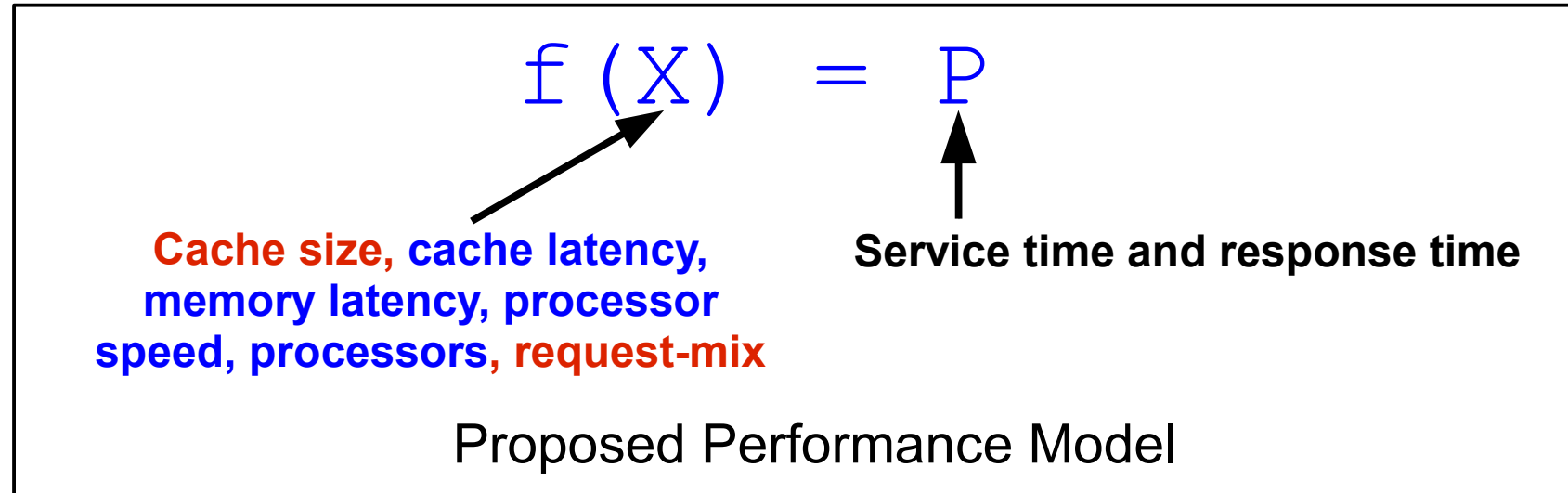
- Response time = service time + queuing delay
- Request-mix M/M/1 queuing model

[stewart-eurosys-2007]

$$y = \sum_{j=1}^n s_j N_j + \sum_r \left(\frac{1}{\lambda} \cdot \frac{U_r^2}{1 - U_r} \right) \cdot \sum_{j=1}^n N_j$$

- Accounts for request mix in waiting time
 - Intuition: other requests utilize resource r which detracts from the time available to service it
- Validated with traces from *real* Internet services

Our Performance Model (Bird's Eye)



- Prototype originally implemented in R (ported to VBA)
- Basic data collection tools: Sysstat, Oprofile, Apache
 - Formatted with ~483 lines of Perl
 - Timestamped data— ORCA Data collector (Adrian Cockcraft)

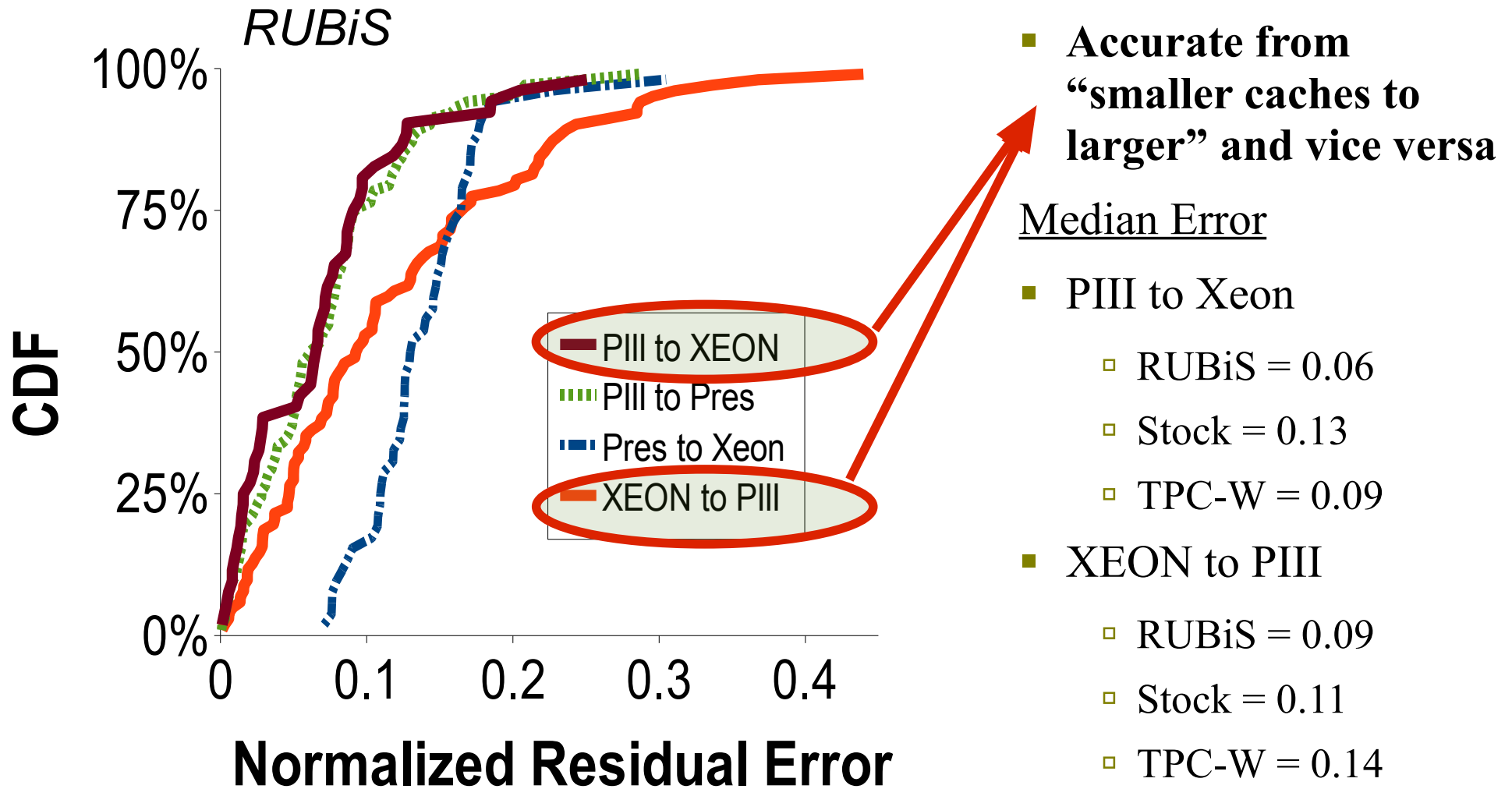
Experimental Setup

- Test platforms
 - **PIII** (Coppermine)
 - 32 KB L1
 - 128 KB L2 cache
 - **Pres** (P4 Prescott)
 - 16 KB L1
 - 512 KB L2 cache
 - **XEON** (PentiumD)
 - 32 KB L1
 - 2 MB L2 cache
 - **More in Paper**
- Request mixes
 - 10-hour nonstationary sequence of requests
 - 1 request mix equals 30 second interval
 - **First 5 hours calibrate model on one platform**
 - **Evaluate on 2nd 5 hours on a different platform**

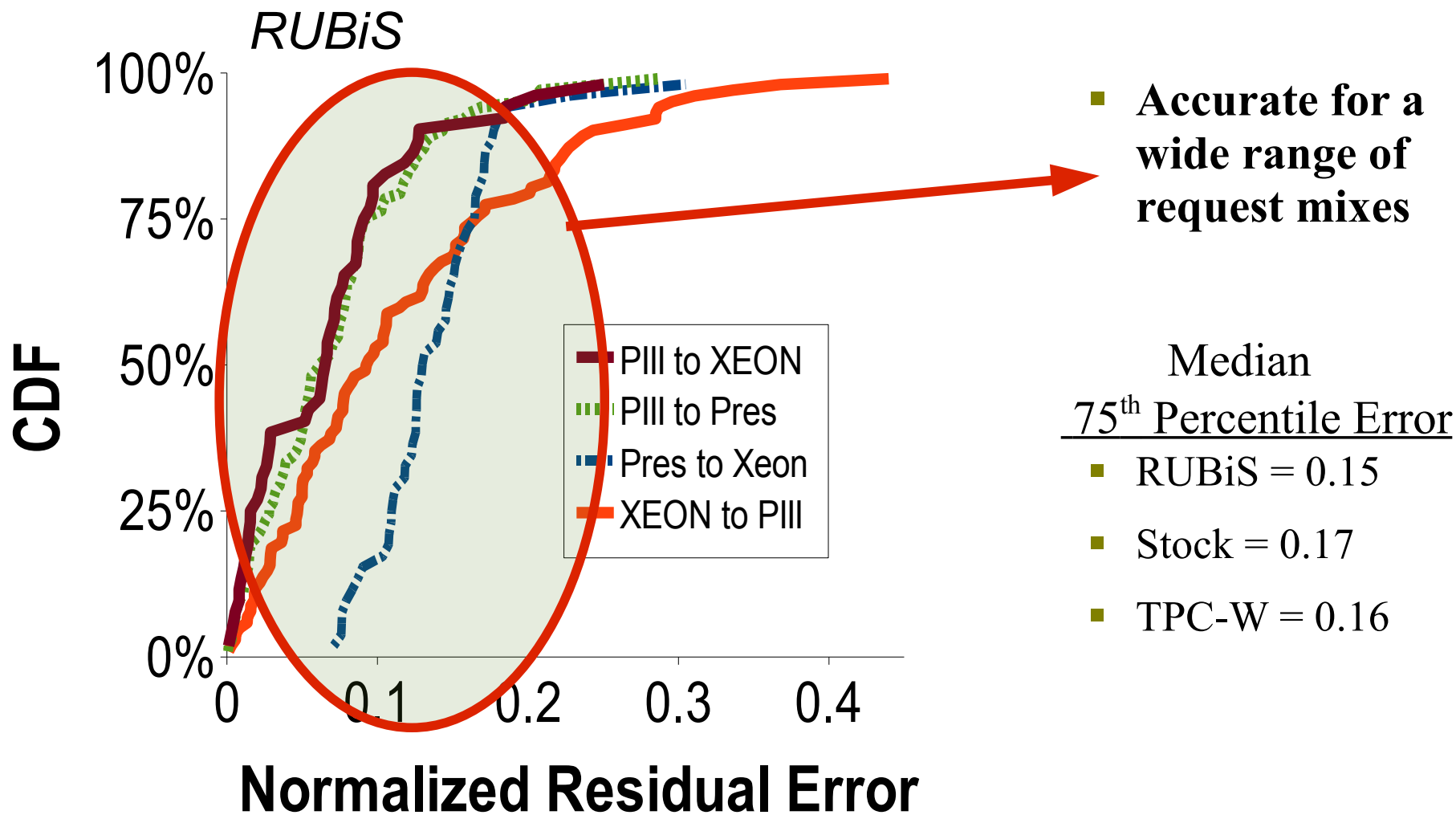
Download our workloads

www.cs.rochester.edu/u/stewart/models.html

Accuracy Across Cache Configurations



Accuracy Across Request Mixes



Outline

1. Motivation

2. Trait models— *empirically observed building blocks*

3. Our cross-platform model— *composition of trait models*

4. A Dollar from 15 cents— *cross-platform management*

- Maximum throughput-per-watt processor selection (see paper)
- Platform-aware load balancing

5. Future work and conclusion

Heterogeneity in Data Centers

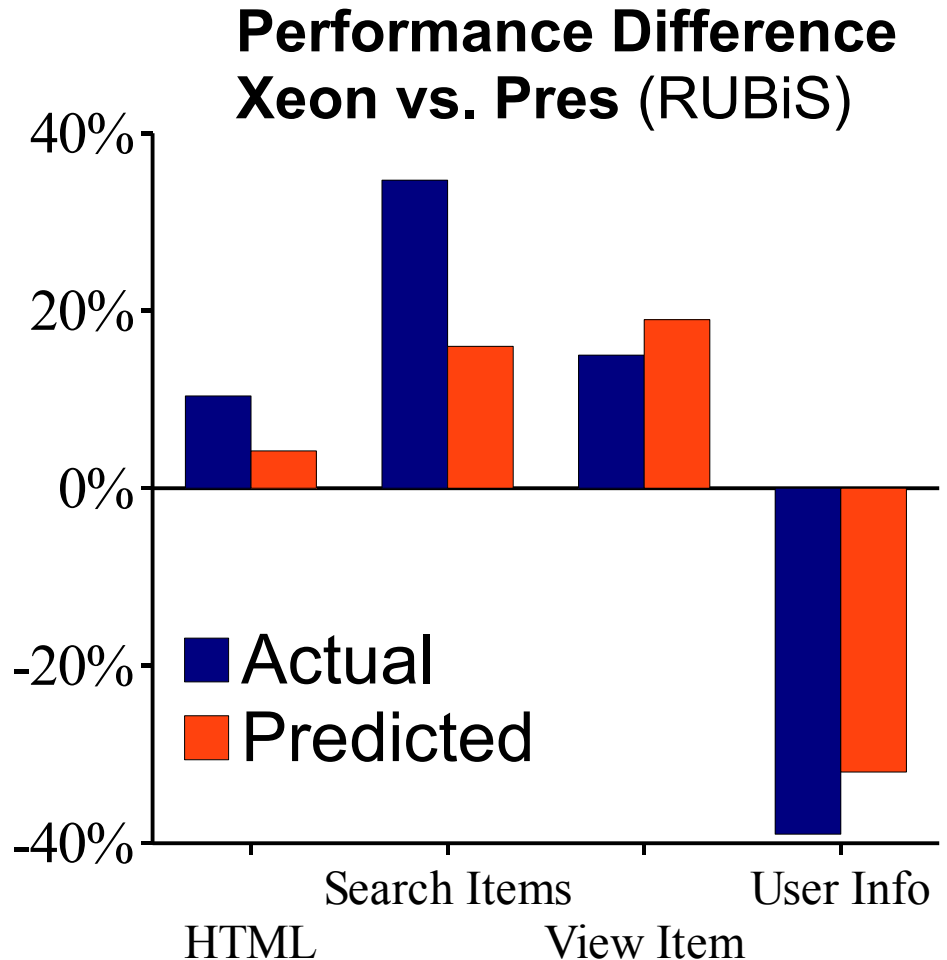
- Makeup of data centers changes over time
 - New platforms are introduced piecewise
 - Old platforms die or become too inefficient
- *Load balancing across heterogeneous platforms*
 - In practice— Weighted round robin
 - Alternatively, designate a homogeneous cluster for each application
 - Research solutions— Locality-aware scheduling
 - Issues requests with similar patterns (same type) to same machine
 - [pai-asplos-1998][amza-usits-2003][elinkety-eurosys-2007]
 - *Suit software to best utilize hardware*

Platform-Aware Load Balancing

- **Distribute requests to the platform best configured to their architectural demands**
 - Request-type granularity
- Example: Xeon vs. Pres
 - Xeon larger L1 & L2
 - Pres lower L2 latency

The best processor varies with request type

Our model can be used to predict per-type performance



Outline

1. Motivation
2. Trait models— *empirically observed building blocks*
3. Our cross-platform model— *composition of trait models*
4. A Dollar from 15 cents— *cross-platform management*
5. Future work and conclusion

Future Work

- Can single-parameter trait models capture the intentions of system designers?
 - **Intuition:** The performance difference between empirical observations that vary on only one parameter most often reflects design intentions
 - **Benefit:** Detect unintended performance degradations (*performance anomalies*) without a-priori knowledge on the intended effects of every parameter
 - [augillera-sosp-2003][chen-nsdi-2004][reynolds-nsdi-2006]
 - **Results:** Found 5 anomalies in Linux; 4 in RUBiS/JBoss

Take Away Points

1. Presented a cross-platform performance model for Internet services
 - Composition of single-parameter *trait models* that are:
 1. derived from empirical data and
 2. calibrated with data commonly available in production
2. Our model can guide the management of today's heterogeneous data centers

Nonstationary workload:

<http://www.cs.rochester.edu/u/stewart/models.html>