# An Empirical Model for Predicting Cross-Core Performance Interference on Multicore Processors


Jiacheng Zhao

Institute of Computing Technology, CAS

In Conjunction with Prof. Jingling Xue,

UNSW, Australia


Sep 11, 2013

# Problem – Resource Utilization in Datacenters

- How?

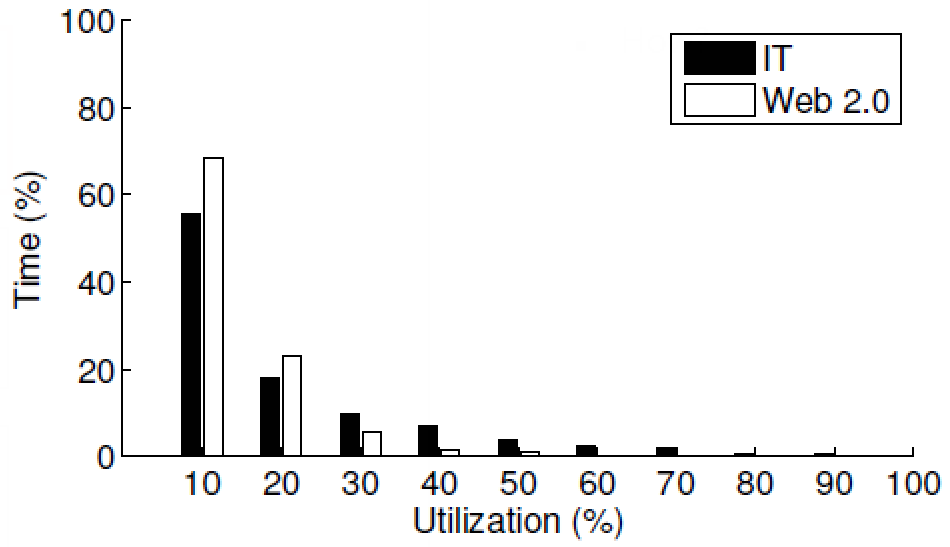
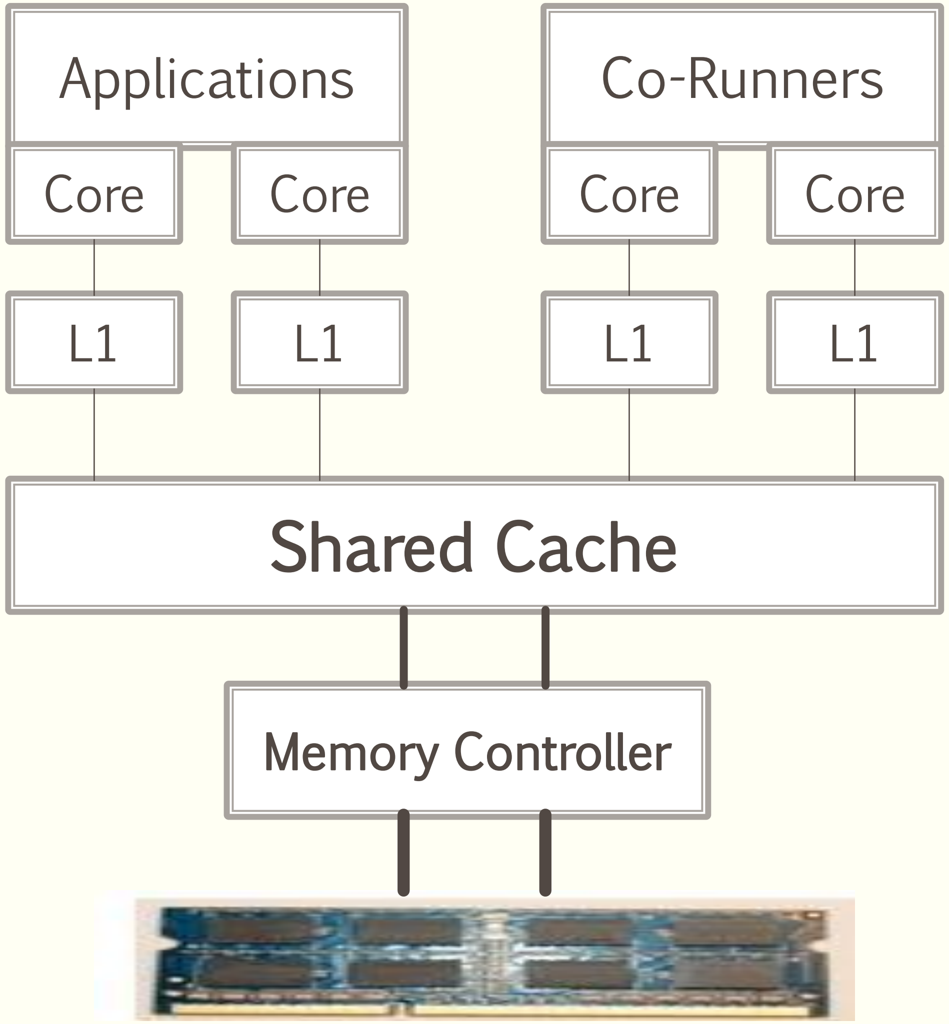



**Figure 1: Server Utilization Histogram.** Real data centers are under 20% utilized.

ASPLOS'09 by David Meisner+

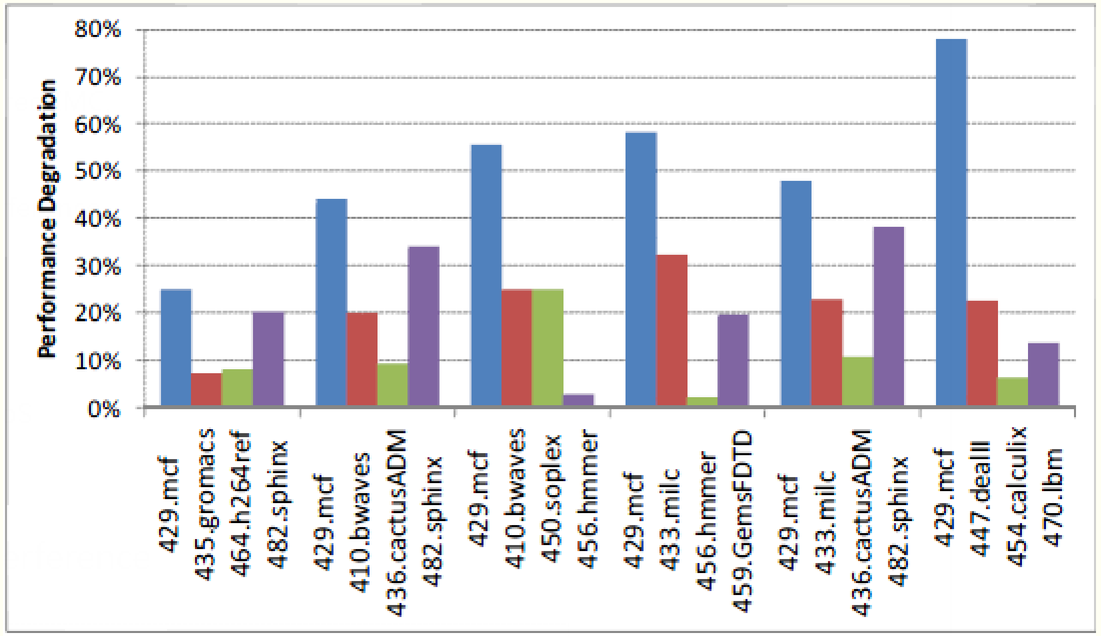
# Problem – Resource Utilization in Datacenters

- Co-located applications
  - Contention for shared cache, shared IMC, etc.
  - Negative and unpredictable interference
- Two types of applications
  - Batch – No QoS guarantees
  - Latency Sensitive - Attain high QoS
- Co-location is disabled
  - Low server utilization
- Lacking the knowledge of interference



# Problem – Resource Utilization in Datacenters

- Co-located applications
  - Contention for shared cache, shared IMC, etc.
  - Negative and unpredictable interference
- Two types of applications
  - Batch – No QoS guarantees
  - Latency Sensitive - Attain high QoS
- Co-location is disabled
  - Low server utilization
- Lacking the knowledge of interference



# Problem – Resource Utilization in Datacenters

- Co-located applications
  - Contention for shared cache, shared IMC, etc.
  - Negative and unpredictable interference
- Two types of applications
  - Batch – No QoS guarantees
  - Latency Sensitive - Attain high QoS
- Co-location is disabled
  - Low server utilization
- Lacking the knowledge of interference

[Micro’11 by Jason Mars+]

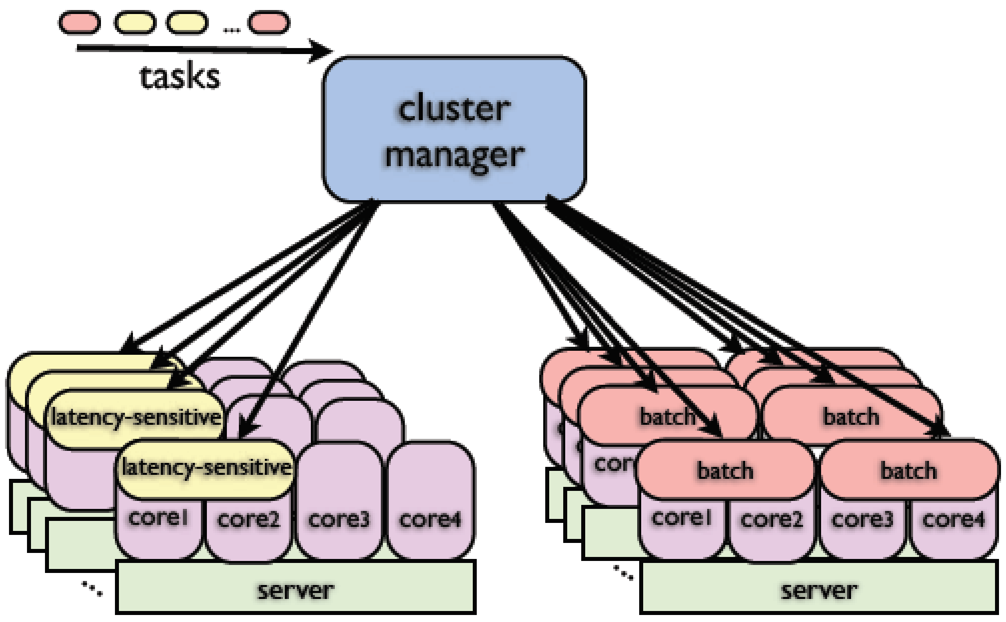



Figure: Task placement in datacenters

# Our Goals: Predicting the interference

- **Quantitatively** predict the cross-core performance interference
- Applicable for **arbitrarily** co-locations
- Identify any “safe” co-locations
- Deployable for datacenters

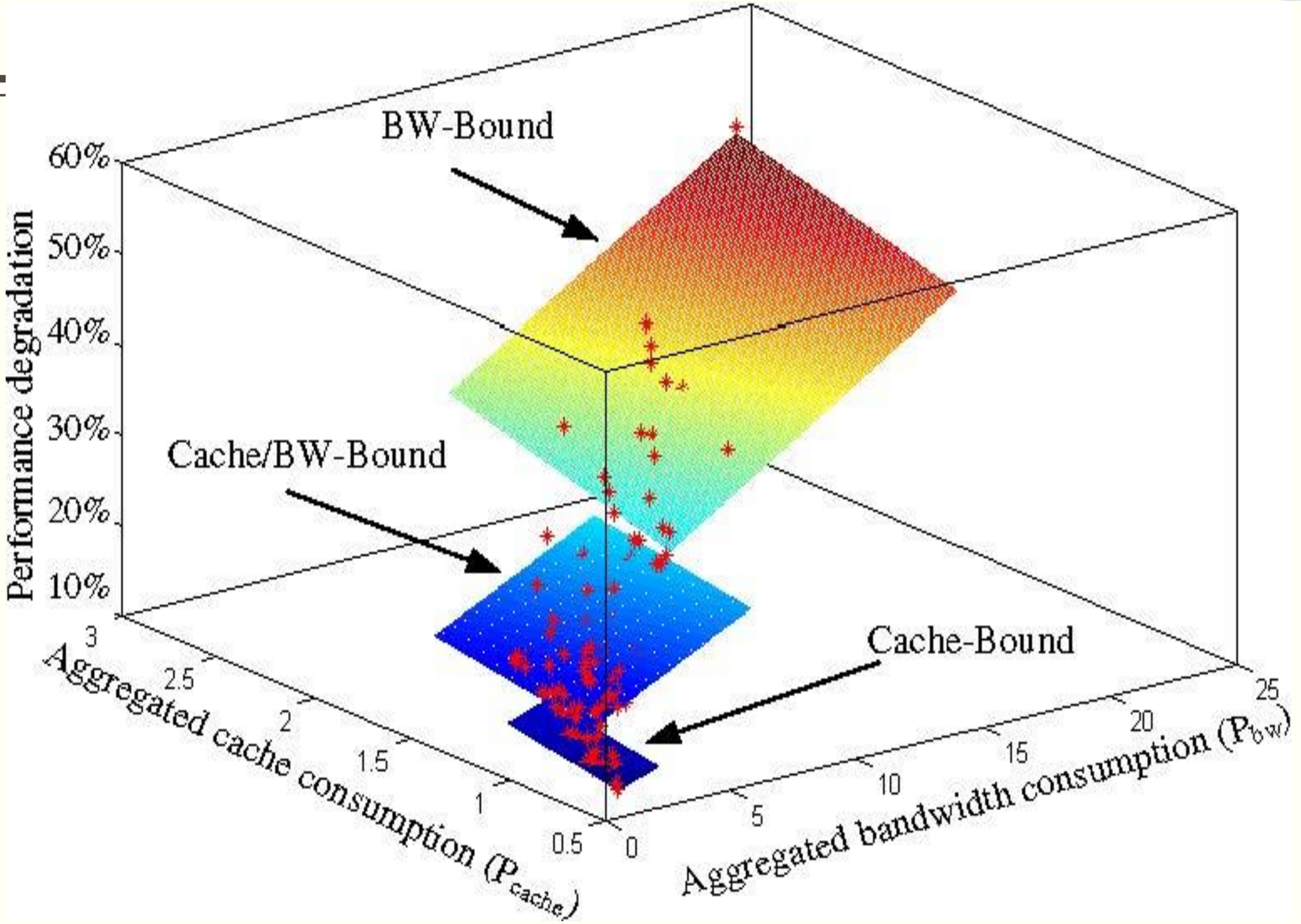
# Our Intuition – Mining a model from large training data

Training Set

| Application | Co-Runners | $A_i'$s Performance Degradation |
|---|---|---|
| $A_1$ | $W_{A_1,1}$ | $PD_{A_1,W_{A_1,1}}$ |
| | $\cdots$ | |
| $A_1$ | $W_{A_1,Q}$ | $PD_{A_1,W_{A_1,Q}}$ |
| $A_2$ | $W_{A_2,1}$ | $PD_{A_2,W_{A_2,1}}$ |
| | $\cdots$ | |
| $A_2$ | $W_{A_2,Q}$ | $PD_{A_2,W_{A_2,Q}}$ |
| | $\cdots$ | |

✓ Using machine learning approaches

# Motivation example




$$PD_{mcf} = \begin{cases} 0.485P_{bw} + 0.183P_{cache} - 0.138, & if\ P_{bw} < 3.2 \\ 0.706P_{bw} + 1.725P_{cache} - 0.220, & if\ 3.2 \le P_{bw} \le 9.6 \\ 0.907P_{bw} + 3.087P_{cache} - 0.561, & if\ P_{bw} > 9.6 \end{cases}$$

# Outline

- Introduction
- **Our Key Observations**
- Our Approach – Two-Phase Approach
- Experimental Results
- Conclusion

# Our Key Observations

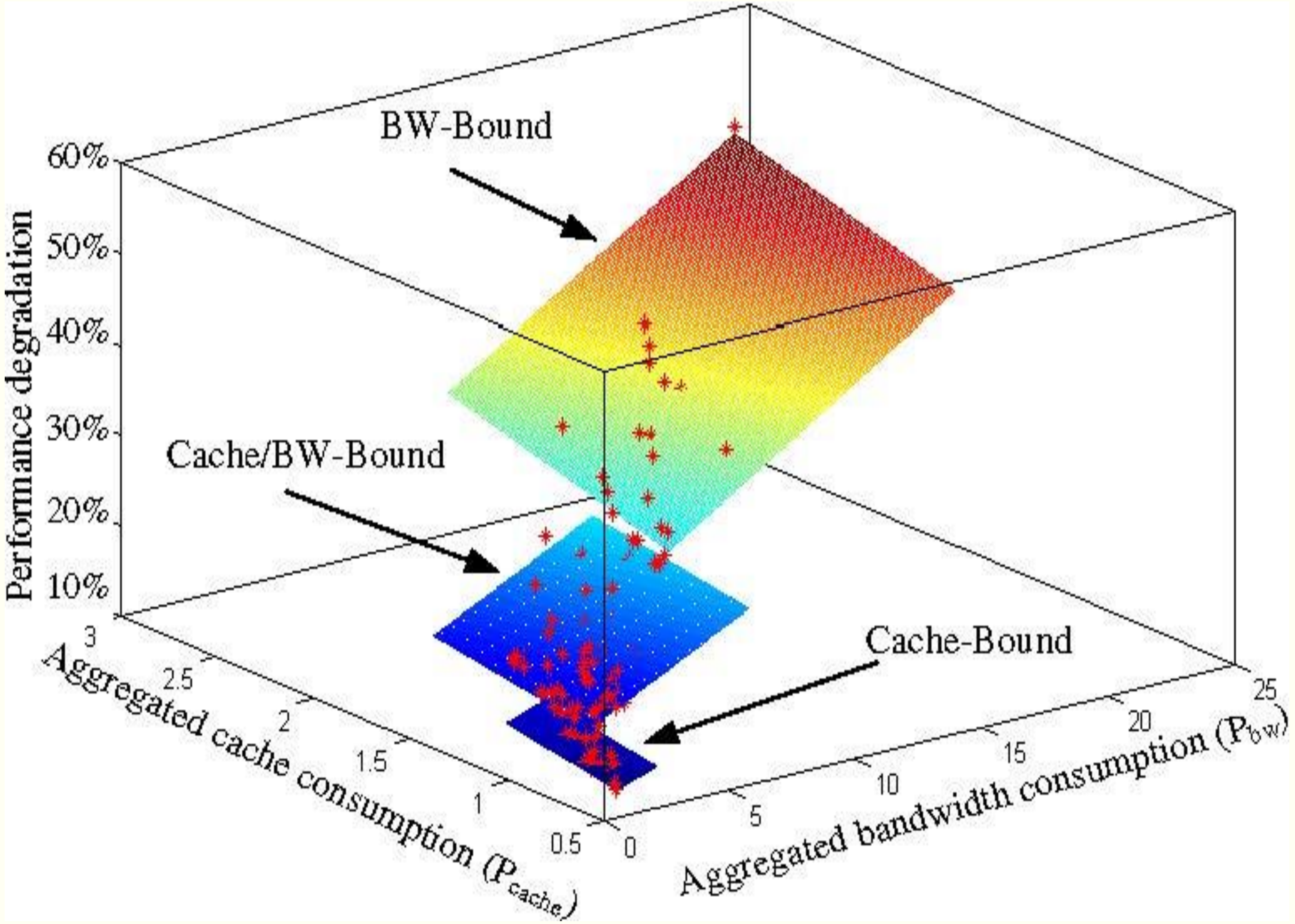
- **Observation 1:** The function depends only on the pressure on shared resources, regardless of individual pressures from one co-runner.

  For an application A, $PD_A = f(P_{cache}, P_{bw})$

  $(P_{cache}, P_{bw}) = g(A_1, A_2, \ldots, A_m)$

# Our Key Observations

- **Observation 2:**
  - The function f is piecewise.



# Our Key Observations

- Naively, we can create A's prediction model using brute-force approach
- **BUT,** we can **NOT** apply brute force approach for each application!
  - Thousands of applications in one datacenter
  - Frequent software updates
  - Different generations of processors
  - Even steps for one application is expensive
- **Observation 3:**
  - The function **form** is platform-dependent and application independent
  - Only the coefficients are application-dependent

# Outline

- Introduction
- Our Key Observations
- **Our Approach - Two-Phase Approach**
- Experimental Results
- Conclusion

# Our Approach - Two-Phase Approach

## Phase 1: Get the abstract model

- Find a function form best suitable for all applications on a given platform

Training Applications → Co-running Trainer

- Heavy – many training workloads
- Run once for one platform

$$PD = \begin{cases} a_{11}P_{bw} + a_{12}P_{cache} + a_{13}, subdomain1 \\ a_{21}P_{bw} + a_{22}P_{cache} + a_{23}, subdomain2 \\ a_{31}P_{bw} + a_{32}P_{cache} + a_{33}, subdomain3 \end{cases}$$

## Phase 2: Instantiate the abstract model

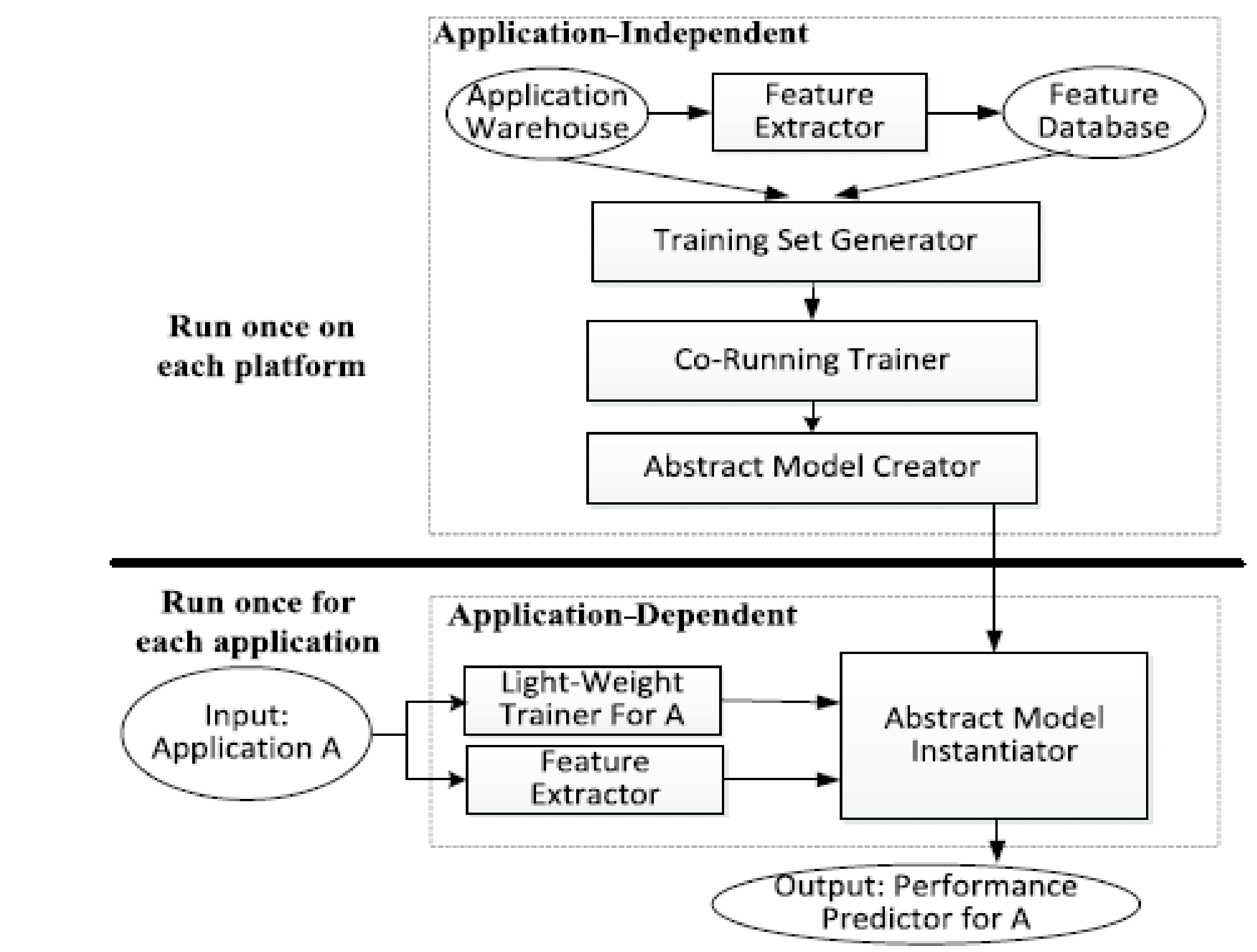
- Determine the application-specific coefficients (a11, etc.)

One Application → Co-running Trainer

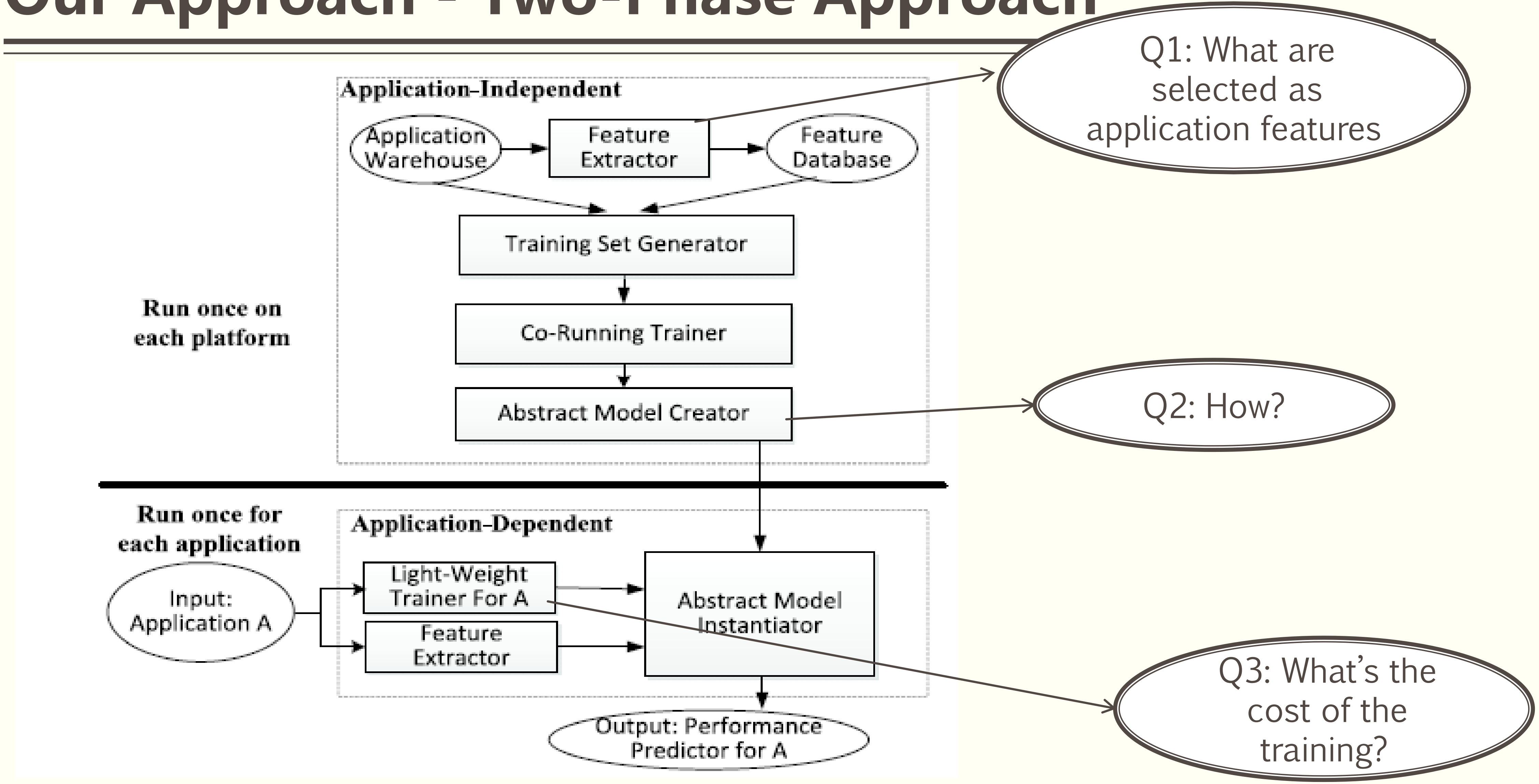
- Light-weighted, with a small number of trainings
- Run once for one application

$$PD_{mcf} = \begin{cases} 0.49P_{bw} + 0.18P_{cache} - 0.13, P_{bw} < 3.2 \\ 0.71P_{bw} + 1.73P_{cache} - 0.22, others \\ 0.91P_{bw} + 3.09P_{cache} - 0.56, P_{bw} > 9.6 \end{cases}$$

# Our Approach - Two-Phase Approach

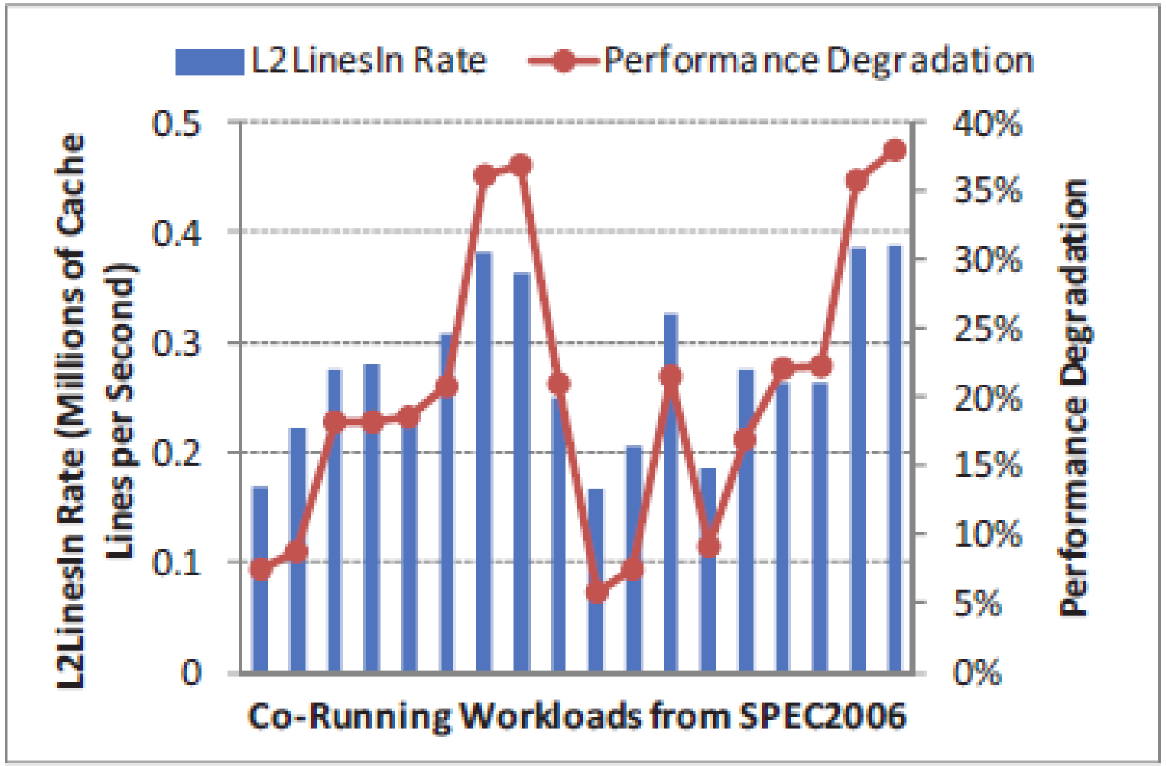


# Our Approach - Two-Phase Approach



# Our Approach – Some Key Points

- Q1: What are selected as application features?
  - Runtime profiles
    - Shared cache consumption
    - Bandwidth consumption



# Our Approach – Some Key Points

- Q2: How to create the abstract model?
  - Regression analysis
  - Configurable
    - Each configuration binding to a function form
  - Searching for the best function form for all applications in the training set

| Application | Co-Runners | $A_i's$ Performance Degradation |
|---|---|---|
| $A_1$ | $W_{A_1,1}$ | $PD_{A_1,W_{A_1,1}}$ |
| | ... | |
| $A_1$ | $W_{A_1,Q}$ | $PD_{A_1,W_{A_1,Q}}$ |
| $A_2$ | $W_{A_2,1}$ | $PD_{A_2,W_{A_2,1}}$ |
| | ... | |
| $A_2$ | $W_{A_2,Q}$ | $PD_{A_2,W_{A_2,Q}}$ |
| | ... | |

**#Aggregation**
  **#Pre-Processing**: none/exp(p)/log(p)/pow(p)
  **#Mode**: add/mul
**#Domain Partitioning**: (shared-resource$_1$, condition$_1$), ...
**#Function**: linear/polynomial(p)/*user-defined*

# Our Approach – Some Key Points

- Q3: What's the cost of the training when instantiation
  - Cover all sub-domains of the piecewise function, say S
  - Constant points for each sub-domain, say C
    - The constant depends on the form of abstraction model
  - C*S training runs in total
  - Usually C and S are small, our experience: C=4, S=3

# Outline

- Introduction
- Our Key Observations
- Our Approach - Two-Phase Approach
- **Experimental Results**
- Conclusion

# Experimental Results

- Accuracy of our two-phase regression approach
  - Prediction precision
  - Error analysis
- Deployment in a datacenter
  - Utilization gained
  - QoS enforced and violated

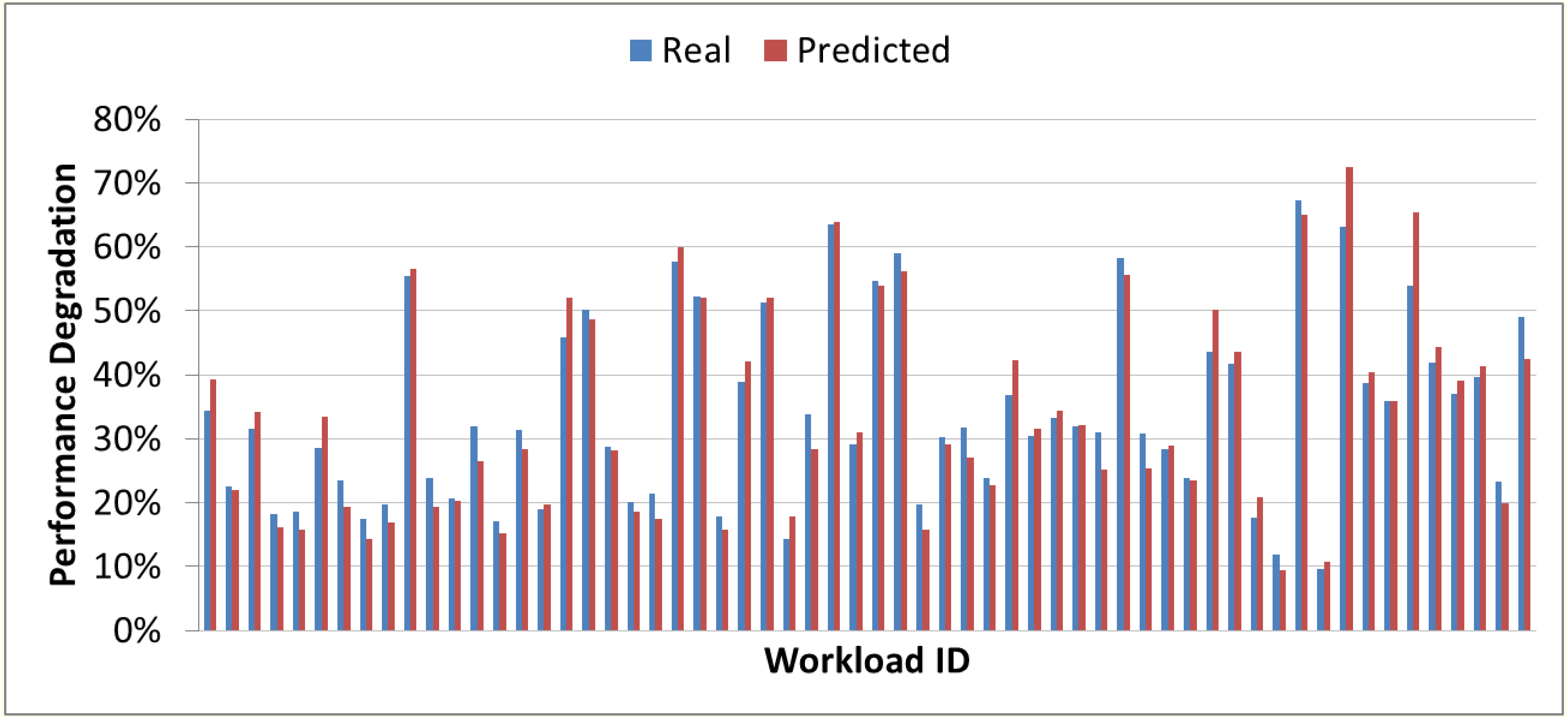
# Experimental Results

- Benchmarks:
  - SPEC2006
  - Nine real-world datacenter applications
    - Nlp-mt, openssl, openclas, MR-iindex, etc.
- Platforms:
  - Intel quad-core Xeon E5506 (main)
- Datacenter:
  - 300 quad-core Xeon E5506

# Some Predictor Function

| | | |
|---|---|---|
| 400.perlbench | $0.108*P_{bw}+0.484*P_{cache}+0.003$<br>$0.115*P_{bw}+0.460*P_{cache}+0.001$<br>$0.176*P_{bw}+0.336*P_{cache}-0.026$ | $(P_{bw} < 3.2)$<br>$(3.2 <= P_{bw} <= 9.6)$<br>$(P_{bw} > 9.6)$ |
| 401.bzip2 | $0.422*P_{bw}+1.337*P_{cache}-0.007$<br>$0.438*P_{bw}+0.714*P_{cache}+0.018$<br>$0.445*P_{bw}+1.240*P_{cache}-0.046$ | $(P_{bw} < 3.2)$<br>$(3.2 <= P_{bw} <= 9.6)$<br>$(P_{bw} > 9.6)$ |
| 433.milc | --<br>$0.403*P_{bw}+0.752*P_{cache}-0.154$<br>$0.935*P_{bw}+1.124*P_{cache}-0.719$ | $(P_{bw} < 3.2)$<br>$(3.2 <= P_{bw} <= 9.6)$<br>$(P_{bw} > 9.6)$ |
| 435.gromacs | $0.093*P_{bw}+0.430*P_{cache}-0.015$<br>$0.129*P_{bw}+0.405*P_{cache}-0.028$<br>$0.154*P_{bw}+0.297*P_{cache}-0.033$ | $(P_{bw} < 3.2)$<br>$(3.2 <= P_{bw} <= 9.6)$<br>$(P_{bw} > 9.6)$ |
| 471.omnetpp | $0.355*P_{bw}+2.044*P_{cache}-0.080$<br>$0.648*P_{bw}+1.280*P_{cache}-0.126$<br>$0.843*P_{bw}+1.012*P_{cache}-0.222$ | $(P_{bw} < 3.2)$<br>$(3.2 <= P_{bw} <= 9.6)$<br>$(P_{bw} > 9.6)$ |

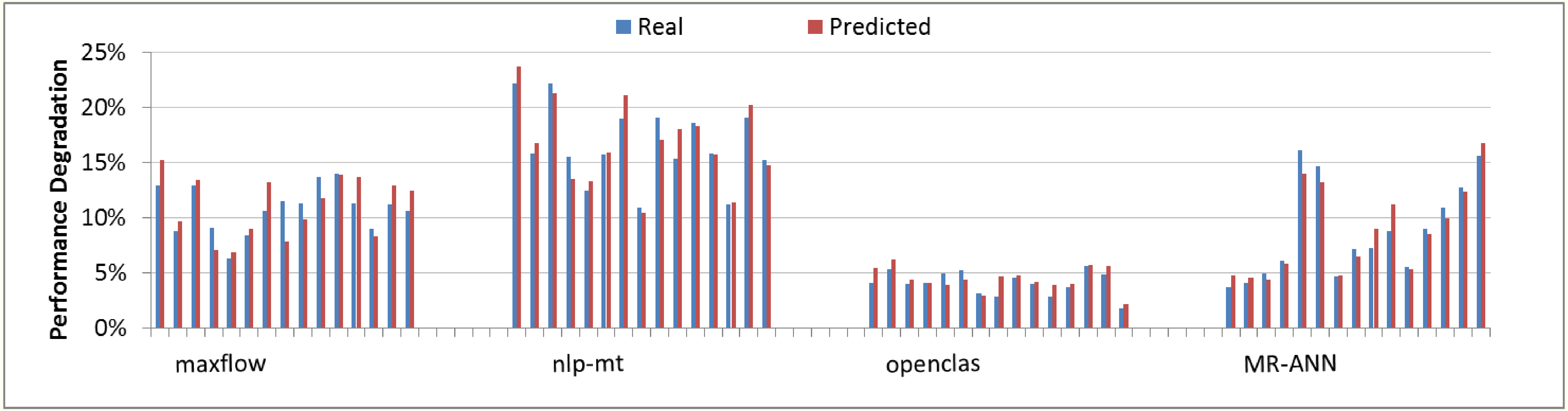
# Prediction precision for SPEC Benchmarks




- Prediction Error: Average **0.2%**, from 0.0% to 8.6%.

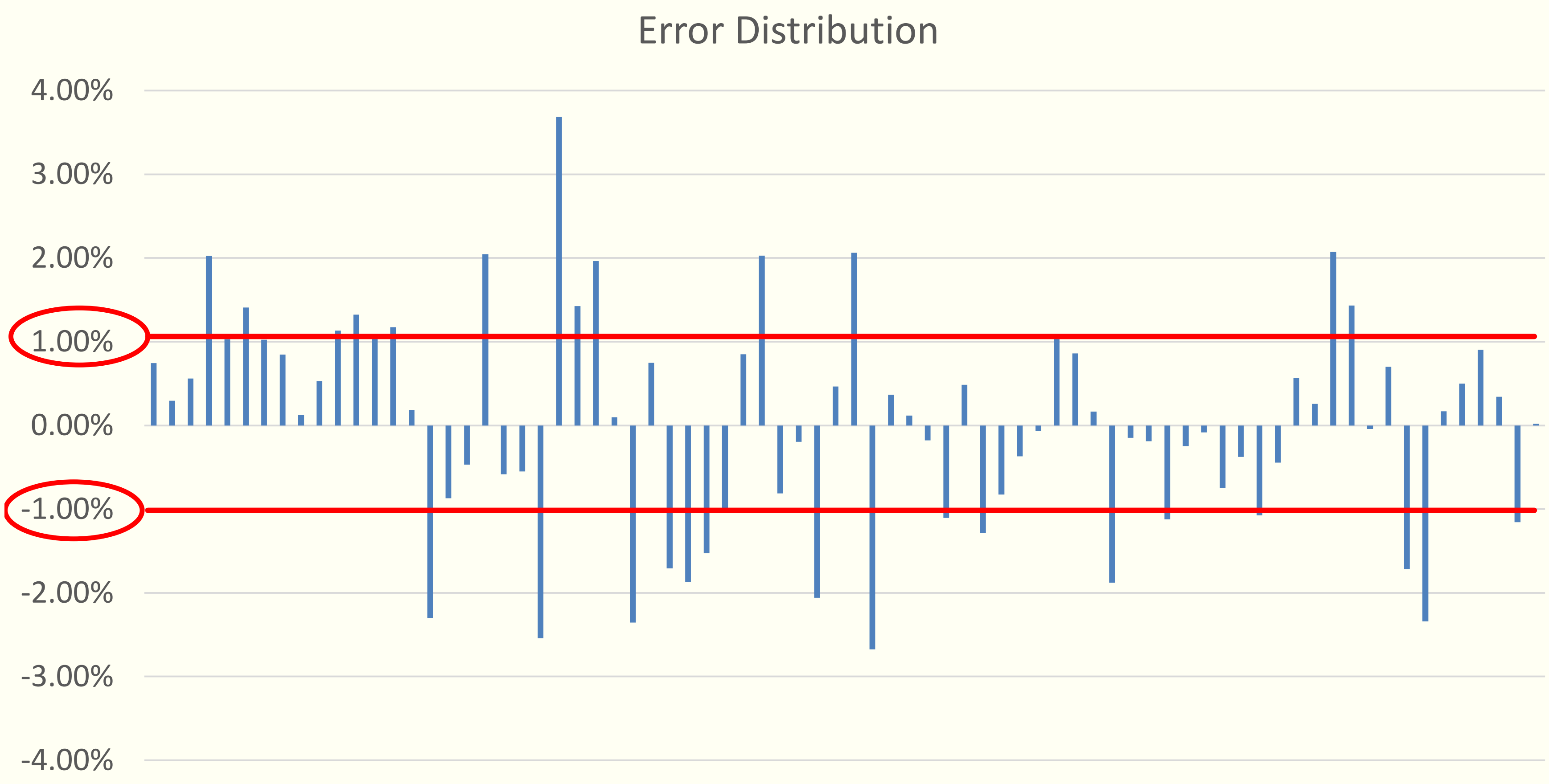
# Prediction precision for datacenter applications

- 15 workloads for each datacenter applications




- Prediction Error: Average **0.3%**, from 0.0% to 5%.

# Error Distribution

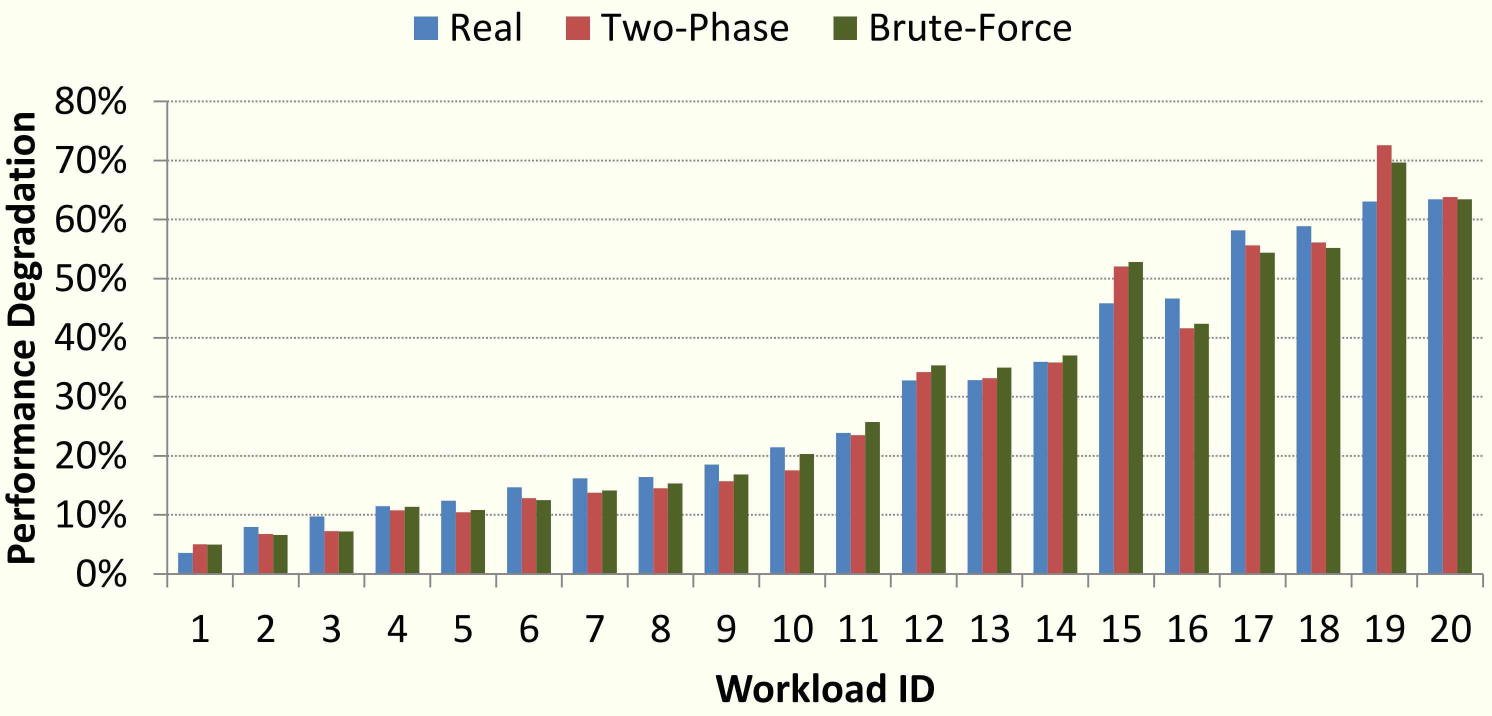


# Prediction Efficiency

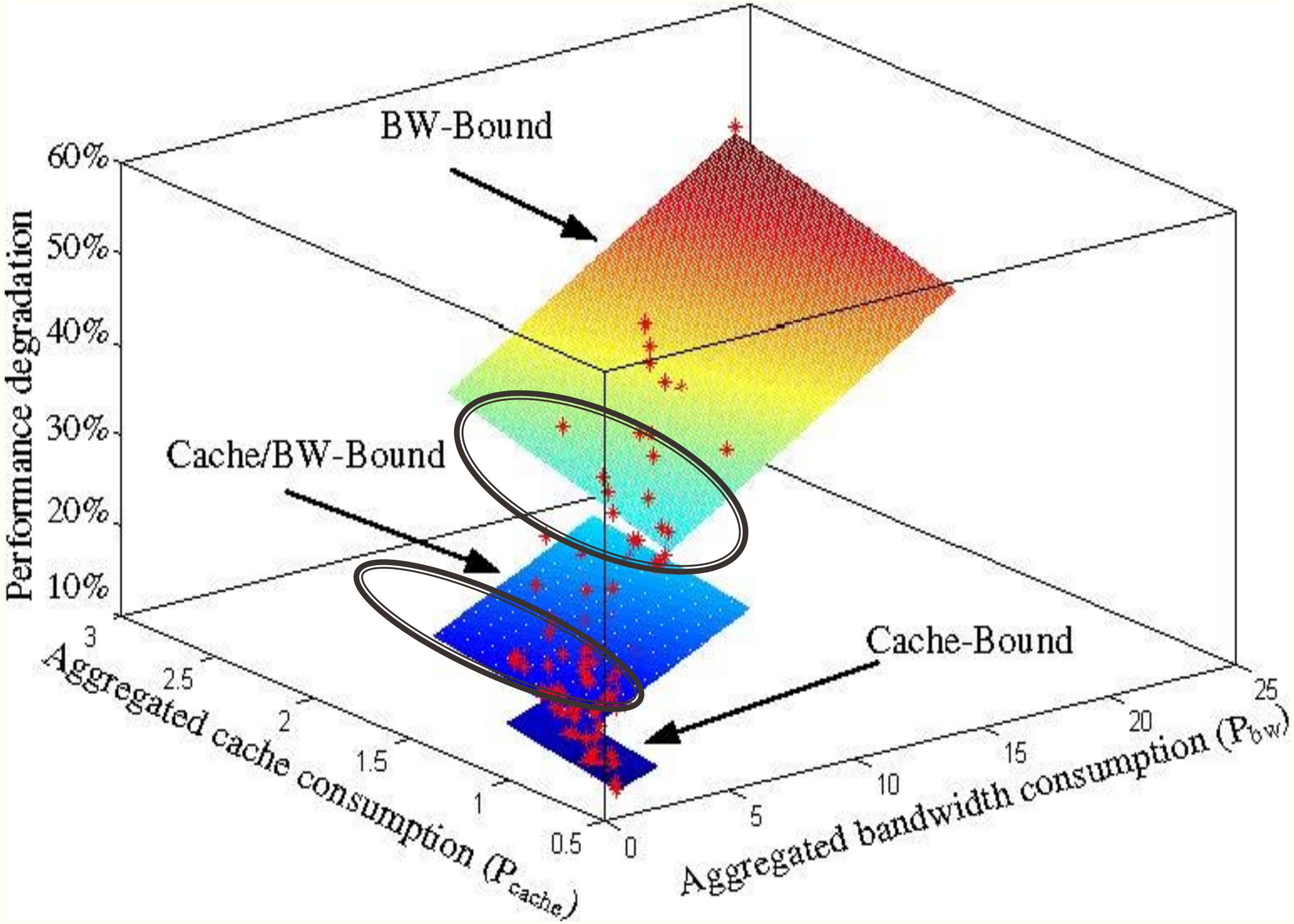
- Precision
  - Two-Phase:

    0.0~11.7%, Average: **0.40%**
  - Brute-Force

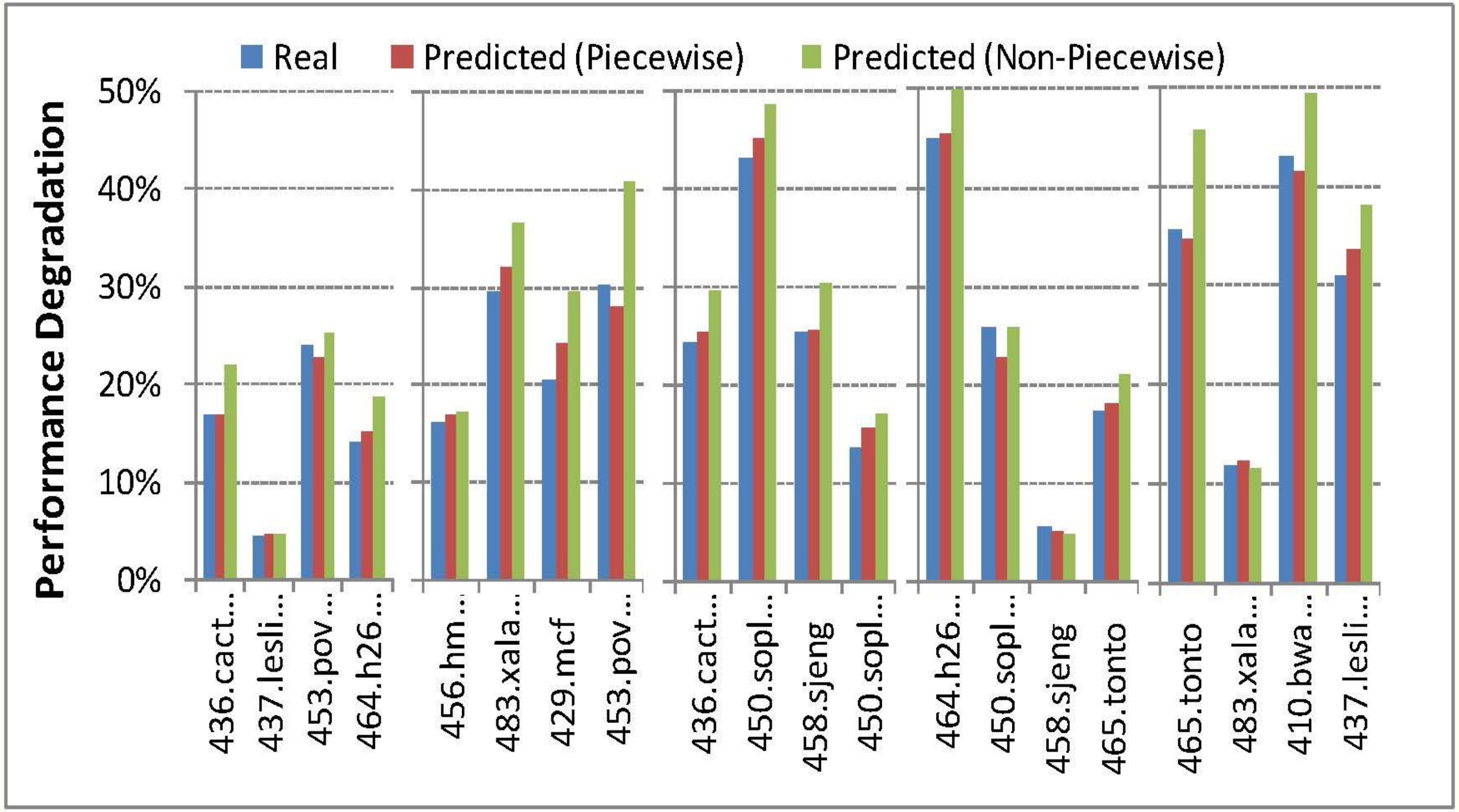
    0.0~10.1%, Average: **0.23%**
- Efficiency
  - co-running: ~200 → 12



# Benefits of piecewise predictor functions

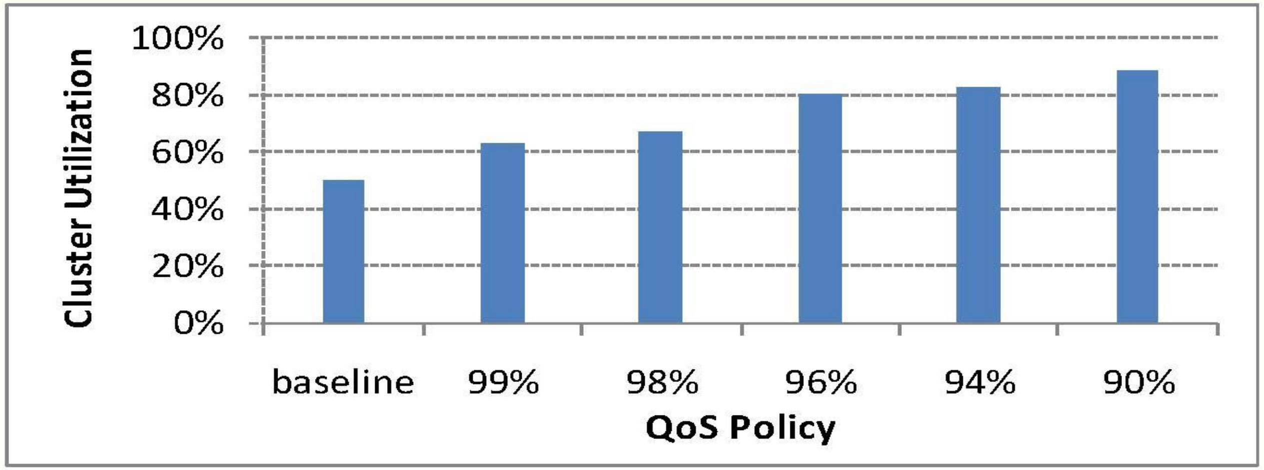


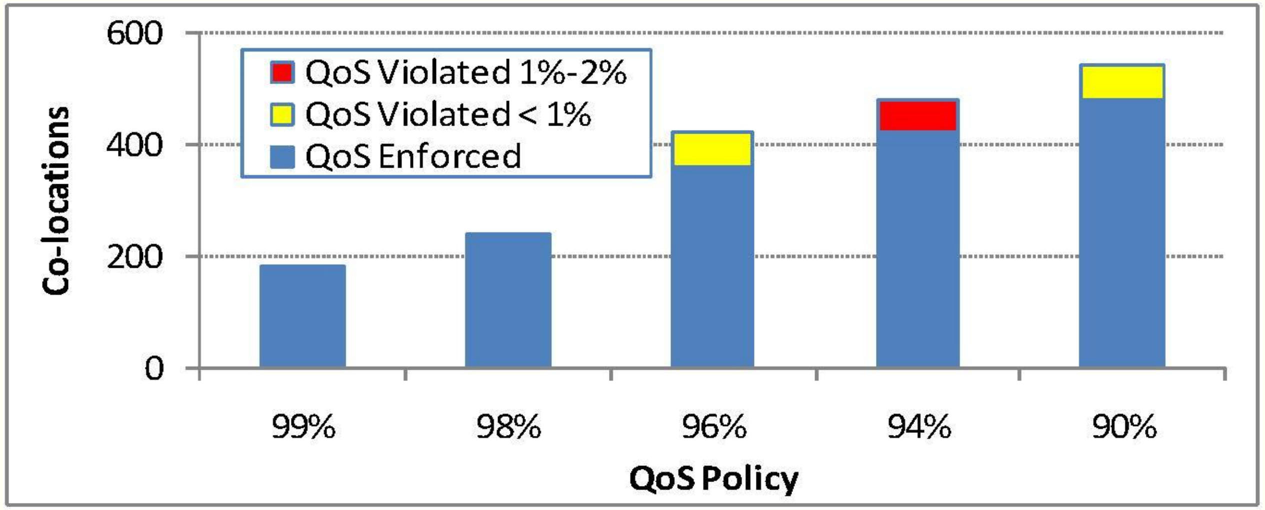
# Benefits of piecewise predictor functions



# Deployment in a datacenter

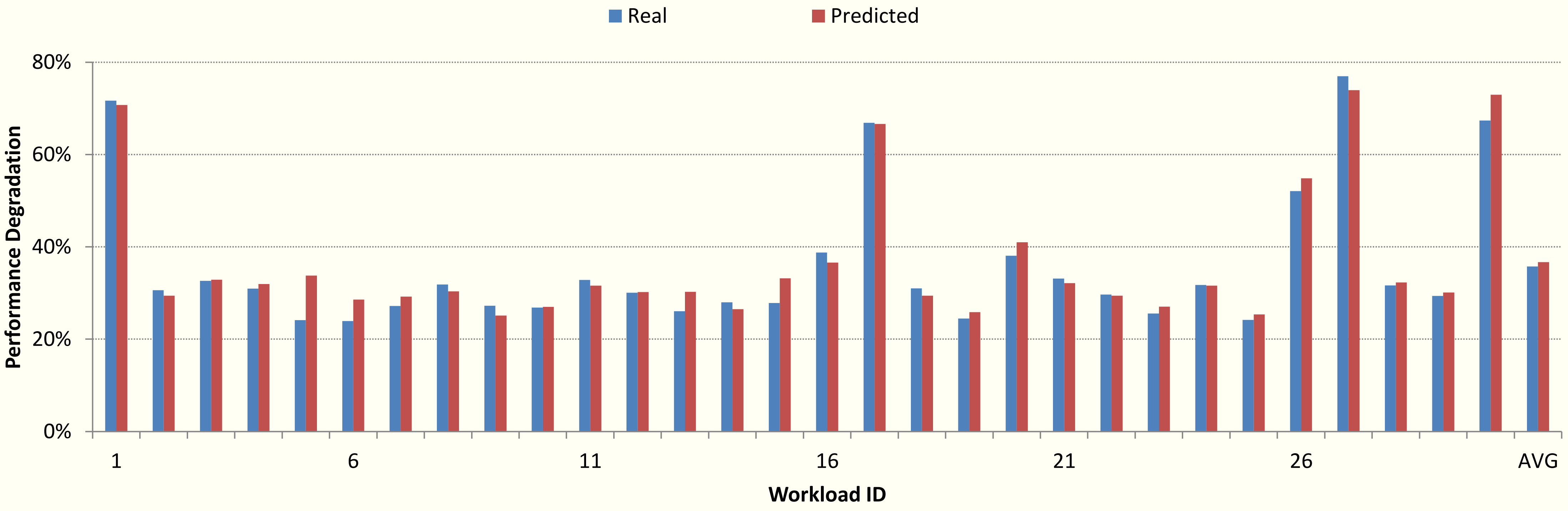
- 300 quad-core Xeon
  - 1200 tasks when fully occupied
- Applications
  - Latency sensitive: Nlp-mt
    - machine translation
    - 600 dedicated cores, 2/chip
  - Batch job
    - 600 tasks, kmeans, MR
- Our Purpose
  - QoS policy
  - Issue batch jobs to idle cores



# ➢Cross-platform applicability

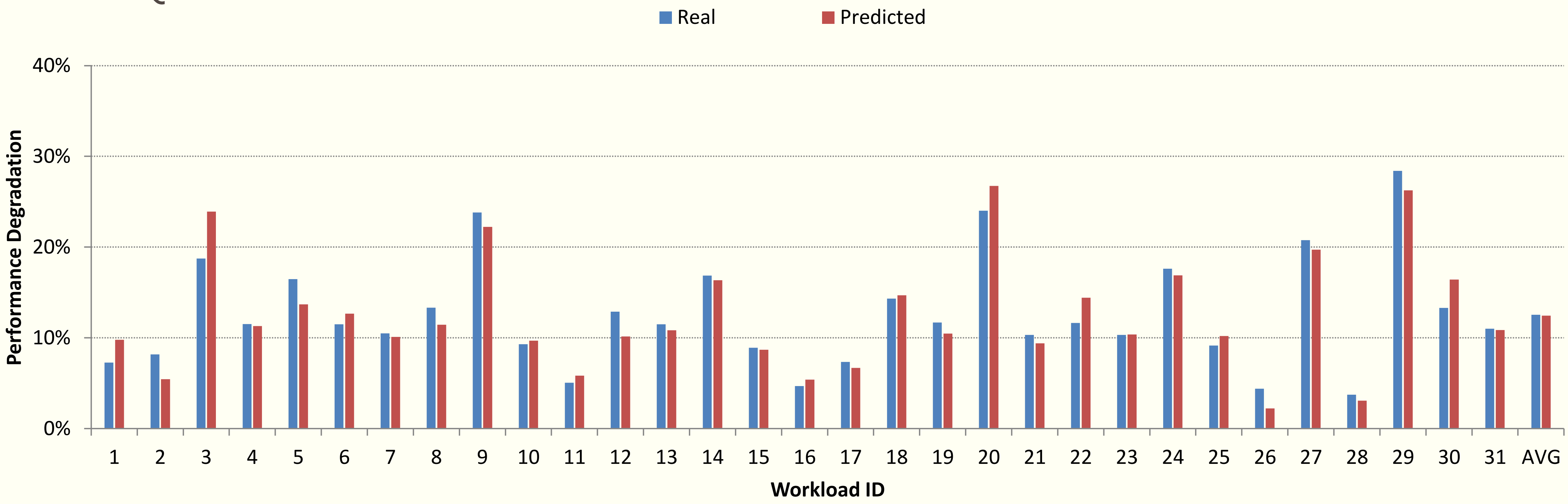
- Six-core Intel Xeon




- Prediction Error: Average **0.1%**, range from 0.0% to 10.2%

# ➢Cross-platform applicability

➢ Quad-core AMD




➢ Prediction Error: Average **0.3%**, range from 0.0% to 5.1%

# Outline

- Introduction
- Our Key Observations
- Our Approach - Two-Phase Approach
- Experimental Results
- **Conclusion**

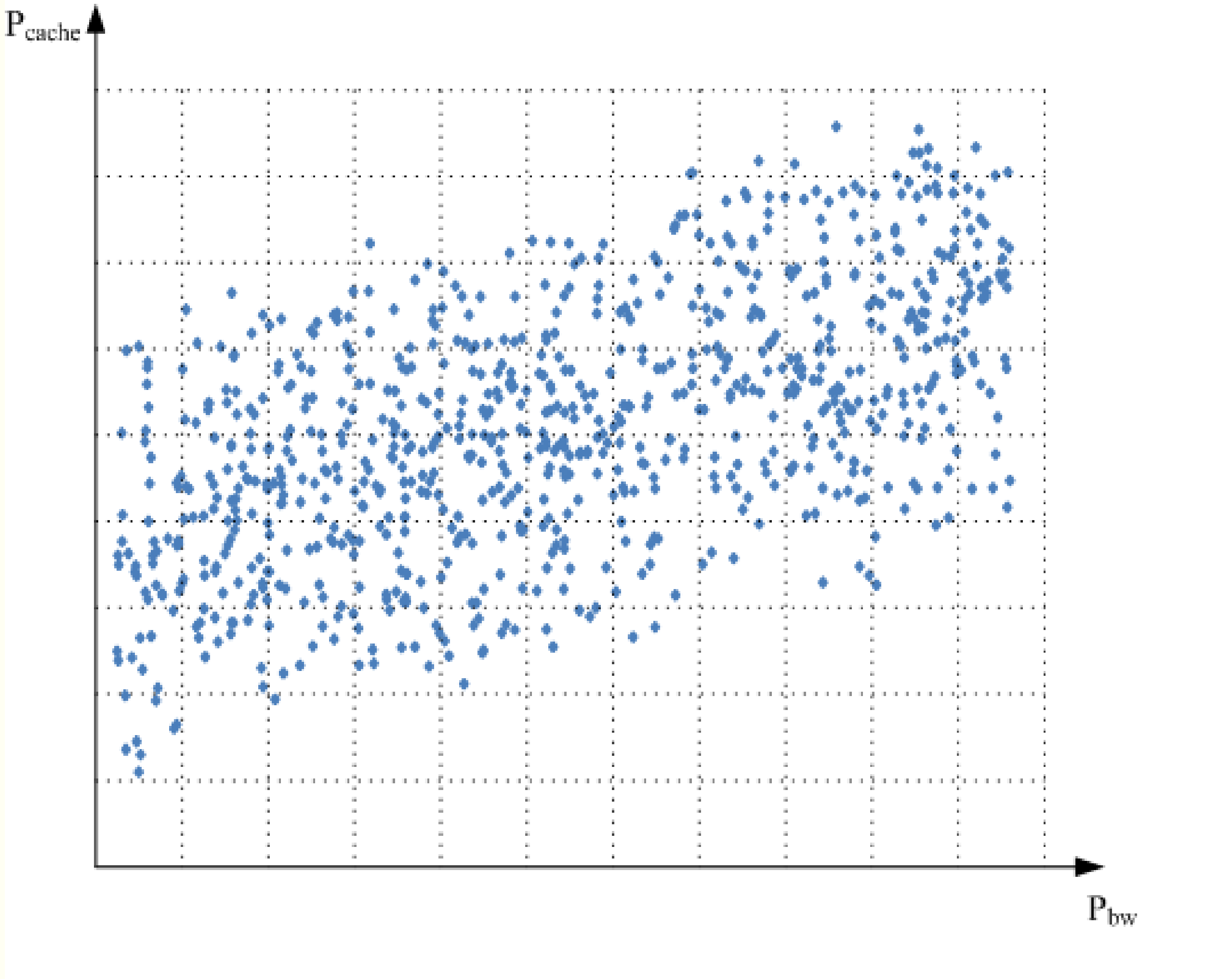
# Conclusion

- An empirical model, based on our key observations
- Using aggregated resource consumptions to create the predictor function, thus working for **arbitrarily** co-locations
- Piecewise is reasonable and effective
- Breaking the model creation into two phases, for efficiency

# Thanks

# Backup slides

- How to make the training set representative?
  - Partition the space into grids
  - Sample for each grid



# Backup slides

- How to do domain partitioning?
  - Specified in configuration file
  - Syntax: (shared $resource_i$, $condition_i$), e.g. ($P_{bw}$, equal(4))
  - Empirical knowledge to perform this task

```
#Aggregation
    #Pre-Processing: none, exp(2), log(2), pow(2)
    #mode: add, mul
#Domain Partitioning: {((Pbw), equal(4)), ((Pcache), equal(4)), ((Pcache, Pbw), equal(4, 4))},
#Function: linear, polynomial(2)
```

# Backup slides

- Two sources of error:
  - Estimation for shared resources consumption
    - L2 LinesIn
  - Phase behavior of applications