LoadIQ: Learning to Identify Workload Phases from a Live Storage Trace

Pankaj Pipada, Achintya Kundu, K. Gopinath, Chiranjib Bhattacharyya[†]

Sai Susarla, P. C. Nagesh[‡]

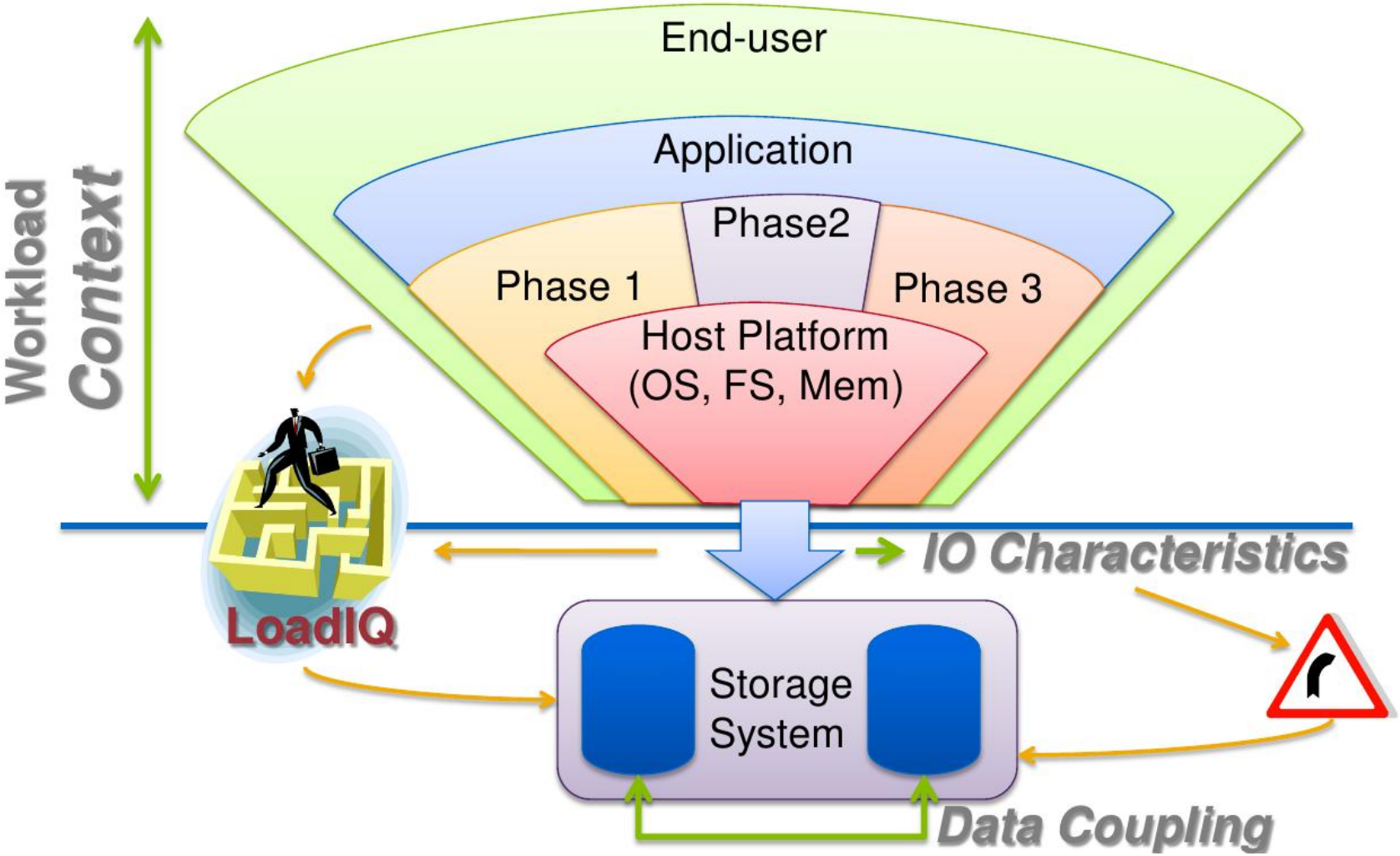
[†]Indian Institute of Science
Bangalore

[‡]NetApp

13^{th} June 2012

Outline

Motivation


End-user
Application
Phase2
Phase 1
Phase 3
Host Platform
(OS, FS, Mem)
Workload Context
LoadIQ
IO Characteristics
Storage System
Data Coupling

Why Context?

- **Application:**
  - E-commerce: browsing vs. shopping phase [Zhang, Riska, and Riedel 2008]
    - Customize storage SLOs to the workload characteristics at hand.
  - DB: OLTP vs. backup/maintenance phase
    - Tune storage-level read-ahead.

- **Host:**
  - Cache type (DRAM or Flash)? Size?
    - Use to avoid wasted caching on shared storage.
  - SNFS, HDFS, Lustre and GPFS
    - Use file system layout knowledge to optimize storage.
  - Differentiated storage services [Mesnier and Akers 2011 SOSP].

Detecting such phase transitions within an application has been problematic [Gu and Verbrugge 2006].

Related work

- Large number of studies to get aggregate information about file systems through trace analysis [Baker et al. 1991; Leung et al. 2008; Roselli, Lorch, and Anderson 2000].
  - As we need to detect specific patterns within an application, aggregate information about file systems is not useful.

Related work

- Large number of studies to get aggregate information about file systems through trace analysis [Baker et al. 1991; Leung et al. 2008; Roselli, Lorch, and Anderson 2000].
  - As we need to detect specific patterns within an application, aggregate information about file systems is not useful.

- Strong correlation between high-level application context and the IO patterns generated [Riska and Riedel 2006; Zhang, Riska, and Riedel 2008].
  - Need such correlation for application phase detection.

Related work

- Large number of studies to get aggregate information about file systems through trace analysis [Baker et al. 1991; Leung et al. 2008; Roselli, Lorch, and Anderson 2000].
  - As we need to detect specific patterns within an application, aggregate information about file systems is not useful.

- Strong correlation between high-level application context and the IO patterns generated [Riska and Riedel 2006; Zhang, Riska, and Riedel 2008].
  - Need such correlation for application phase detection.

- Inferring the sequentiality of workloads and access patterns using block traces [Madhyastha and Reed 1997].
  - Dynamically drives prefetching and caching decisions.

Related work

- Large number of studies to get aggregate information about file systems through trace analysis [Baker et al. 1991; Leung et al. 2008; Roselli, Lorch, and Anderson 2000].
  - As we need to detect specific patterns within an application, aggregate information about file systems is not useful.
- Strong correlation between high-level application context and the IO patterns generated [Riska and Riedel 2006; Zhang, Riska, and Riedel 2008].
  - Need such correlation for application phase detection.
- Inferring the sequentiality of workloads and access patterns using block traces [Madhyastha and Reed 1997].
  - Dynamically drives prefetching and caching decisions.
- The work closest in spirit to this work: Identifying workloads from NFS traces [Yadwadkar et al. 2010].
  - Uses opcode sequence for classification.
  - Limited applicability in VM environments where most requests are reads and writes only.

Our approach

- Identify any set of specific patterns based on past training.
  - Not just sequential or any particular access pattern

Our approach

- Identify any set of specific patterns based on past training.
  - Not just sequential or any particular access pattern
- A generic technique that can work for a variety of applications and is robust against variations in environment and configuration.
  - No dependence on specific heuristics for a specific application

Our approach

- Identify any set of specific patterns based on past training.
  - Not just sequential or any particular access pattern
- A generic technique that can work for a variety of applications and is robust against variations in environment and configuration.
  - No dependence on specific heuristics for a specific application
- Applicable in VM environments.

Problem definition

- Build a tool to track workload phase shifts in real-time (every minute) from a live trace feed and perform trace annotation.

Problem definition

- Build a tool to track workload phase shifts in real-time (every minute) from a live trace feed and perform trace annotation.

- Desired properties:

Problem definition

- Build a tool to track workload phase shifts in real-time (every minute) from a live trace feed and perform trace annotation.
- Desired properties:
  - Non-intrusive

Problem definition

- Build a tool to track workload phase shifts in real-time (every minute) from a live trace feed and perform trace annotation.
- Desired properties:
  - Non-intrusive
  - Dependable: Accurately discriminate among known classes of workload phases.

Problem definition

- Build a tool to track workload phase shifts in real-time (every minute) from a live trace feed and perform trace annotation.
- Desired properties:
  - Non-intrusive
  - Dependable: Accurately discriminate among known classes of workload phases.
  - Extensible: Support augmenting new phase types.

Problem definition

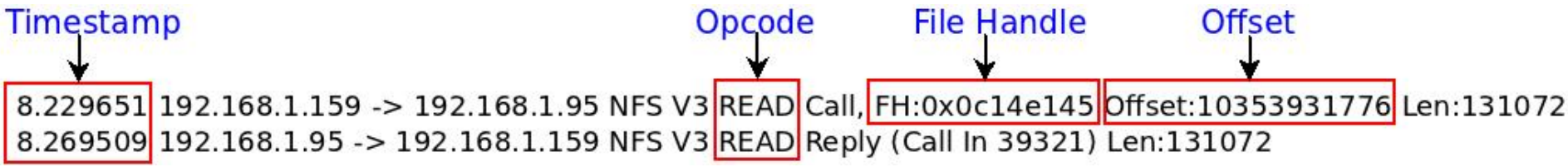
- Build a tool to track workload phase shifts in real-time (every minute) from a live trace feed and perform trace annotation.
- Desired properties:
  - Non-intrusive
  - Dependable: Accurately discriminate among known classes of workload phases.
  - Extensible: Support augmenting new phase types.
  - Automated: Identify phases in near real-time to support online adaptation, where manual intervention is impractical.

Problem definition

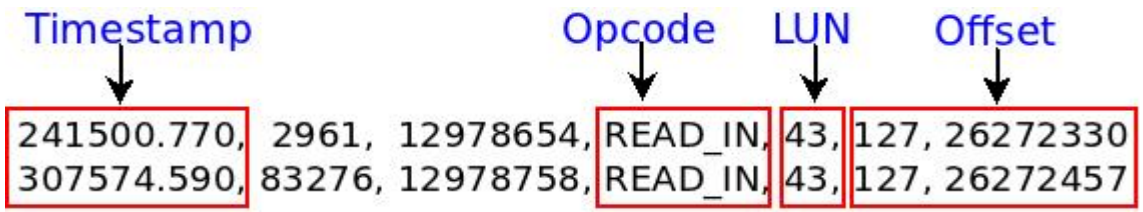
- Build a tool to track workload phase shifts in real-time (every minute) from a live trace feed and perform trace annotation.
- Desired properties:
  - Non-intrusive
  - Dependable: Accurately discriminate among known classes of workload phases.
  - Extensible: Support augmenting new phase types.
  - Automated: Identify phases in near real-time to support online adaptation, where manual intervention is impractical.
  - Robust against inevitable flux in real-world workload patterns due to variations in intensity, time spread and client-side or network environment.

Components of a trace

NFS Trace:



SAN Trace:



Similarity based classification

Set of objects: $\mathcal{X}$

Similarity function: $s : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$

Training Data: $\{ \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7 \}$

Test Input: $\mathbf{x}$

Given: $s(\mathbf{x}, \mathbf{x}_1), \ldots, s(\mathbf{x}, \mathbf{x}_4), \; s(\mathbf{x}, \mathbf{x}_5), \ldots, s(\mathbf{x}, \mathbf{x}_7)$

Q1: How to define $s(.,.)$ for storage traces ?
Q2: How to predict the class of $\mathbf{x}$?

Similarity using offset shift histograms

- Extract offset fields from the NFS trace's `READ` and `WRITE` requests.
- Compute a histogram out of the absolute difference between each successive offset fields (i.e, offset shift).
- Quantize the offset shifts into their nearest *bin* sizes in powers of 2, i.e., sizes of 2^1, 2^2, 2^3, ... bytes.
- Normalize the histograms to eliminate unwanted effects due to different trace lengths.

Similarity using offset shift histograms

- Given two histograms H_1 and H_2, a similarity score is computed as follows:

$$S(H_1, H_2) = c - \sum_{i=1}^{L} \frac{[H_1(i) - H_2(i)]^2}{H_1(i) + H_2(i)}$$

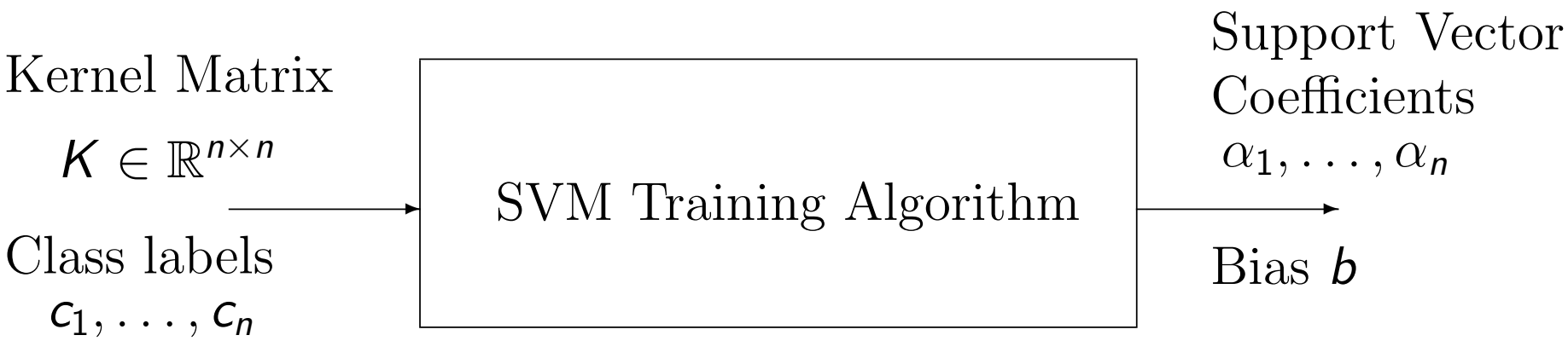
  where L is the number of bins and c is a constant representing the average similarity across all training traces.

- Given a similarity score between any two traces, a similarity matrix is constructed across all the representative traces.

Support Vector Machine (SVM)

Training Data: $\{ (\mathbf{x}_1, c_1), (\mathbf{x}_2, c_2), \ldots (\mathbf{x}_n, c_n) \}$

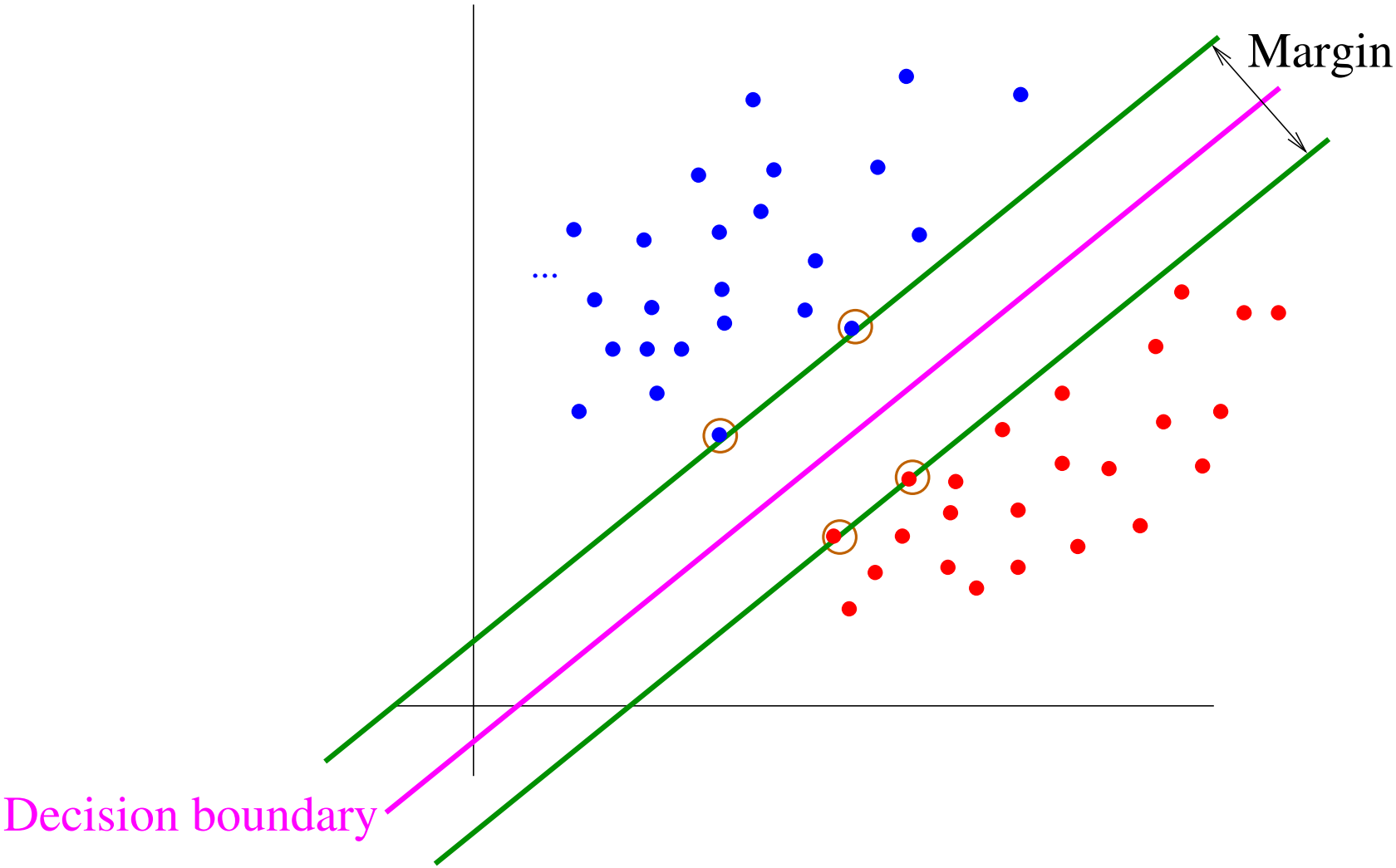
Test Data: $\mathbf{x}$



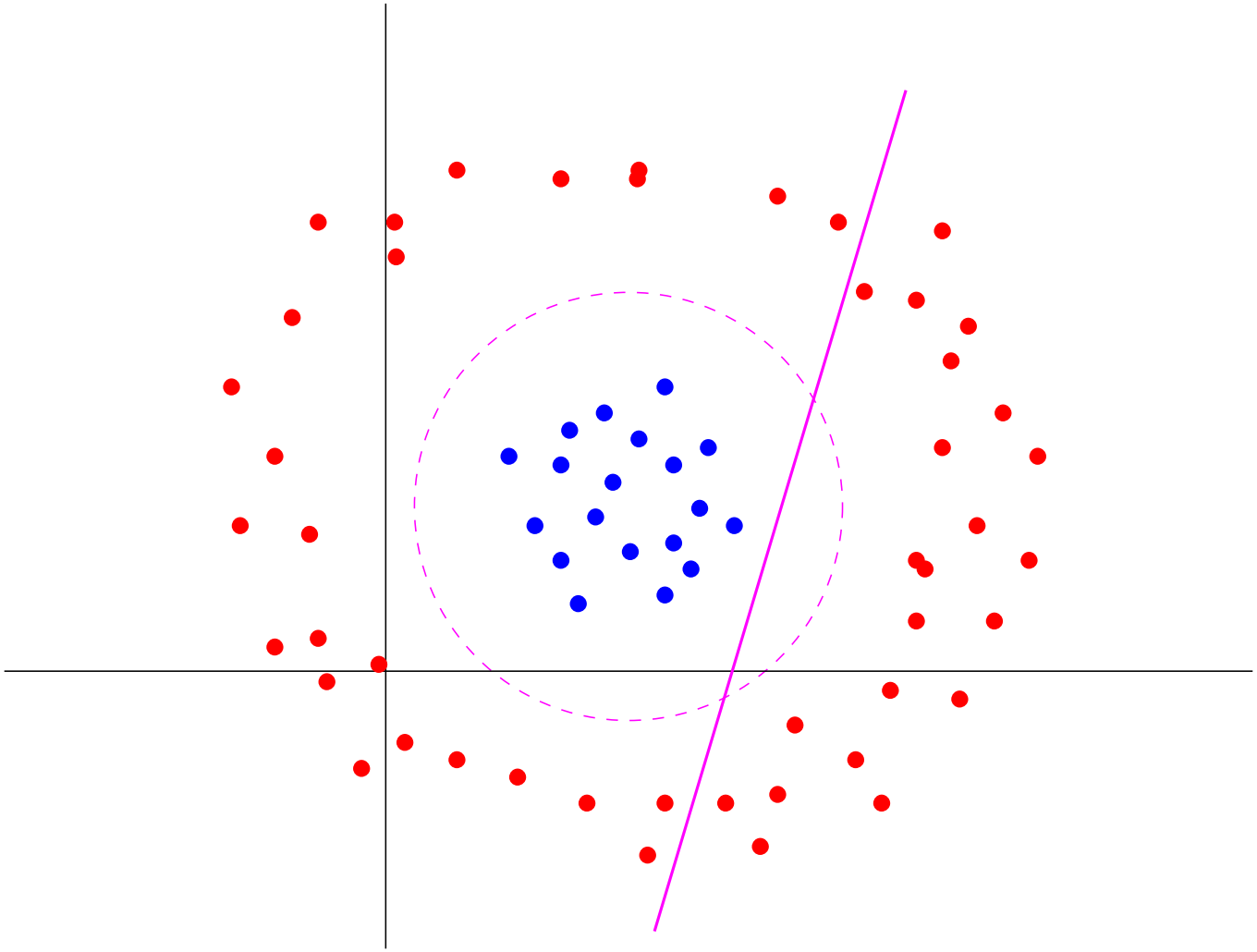
Decision function: $f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i \, c_i \, k(\mathbf{x}, \mathbf{x}_i) + b, \quad \alpha_i \geq 0, \; b \in \mathbb{R}$

SVM Classifier: $sign\,(\, f(\mathbf{x})\,)$

Linear classification using SVM



Nonlinear classification using SVM



Kernel

Similarity matrix: Symmetric but not guaranteed to be positive semidefinite.

$$S = \begin{bmatrix} s(\mathbf{x}_1, \mathbf{x}_1) & s(\mathbf{x}_1, \mathbf{x}_2) & \dots & s(\mathbf{x}_1, \mathbf{x}_n) \\ s(\mathbf{x}_2, \mathbf{x}_1) & s(\mathbf{x}_2, \mathbf{x}_2) & \dots & s(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \dots \\ s(\mathbf{x}_n, \mathbf{x}_1) & s(\mathbf{x}_n, \mathbf{x}_2) & \dots & s(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix} \in \mathbb{R}^{n \times n}$$

Kernel matrix: A PSD matrix achieved by setting the negative eigen-values of the similarity matrix to zero [Chen, Gupta, and Recht 2009].

$$K = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \dots & k(\mathbf{x}_1, \mathbf{x}_n) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \dots & k(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \dots \\ k(\mathbf{x}_n, \mathbf{x}_1) & k(\mathbf{x}_n, \mathbf{x}_2) & \dots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix} \in \mathbb{R}^{n \times n}$$

Workflow

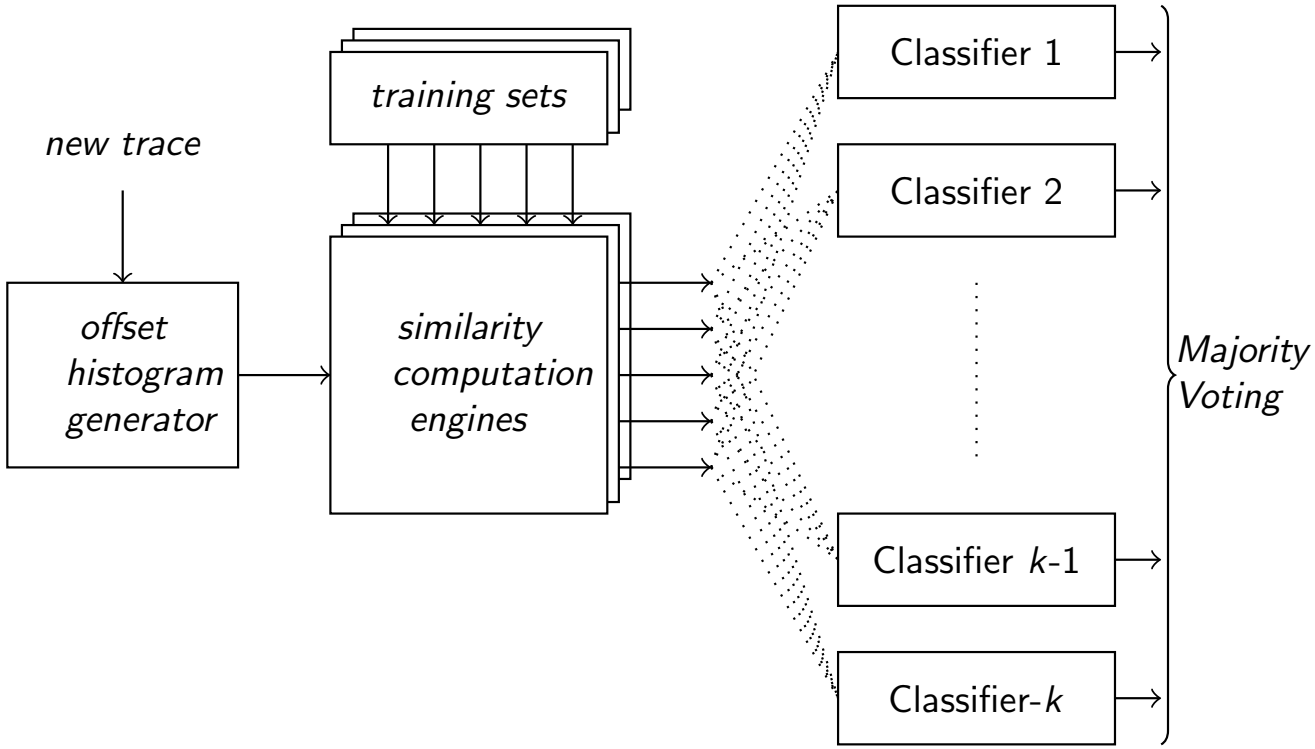


Figure: Block diagram for classifying m phases. Number of classifiers $k = \frac{1}{2}m(m-1)$.

- A trace belongs to a class if and only if number of votes in its favor is exactly $m-1$; otherwise it belongs to class *Unknown*.

Online self correction

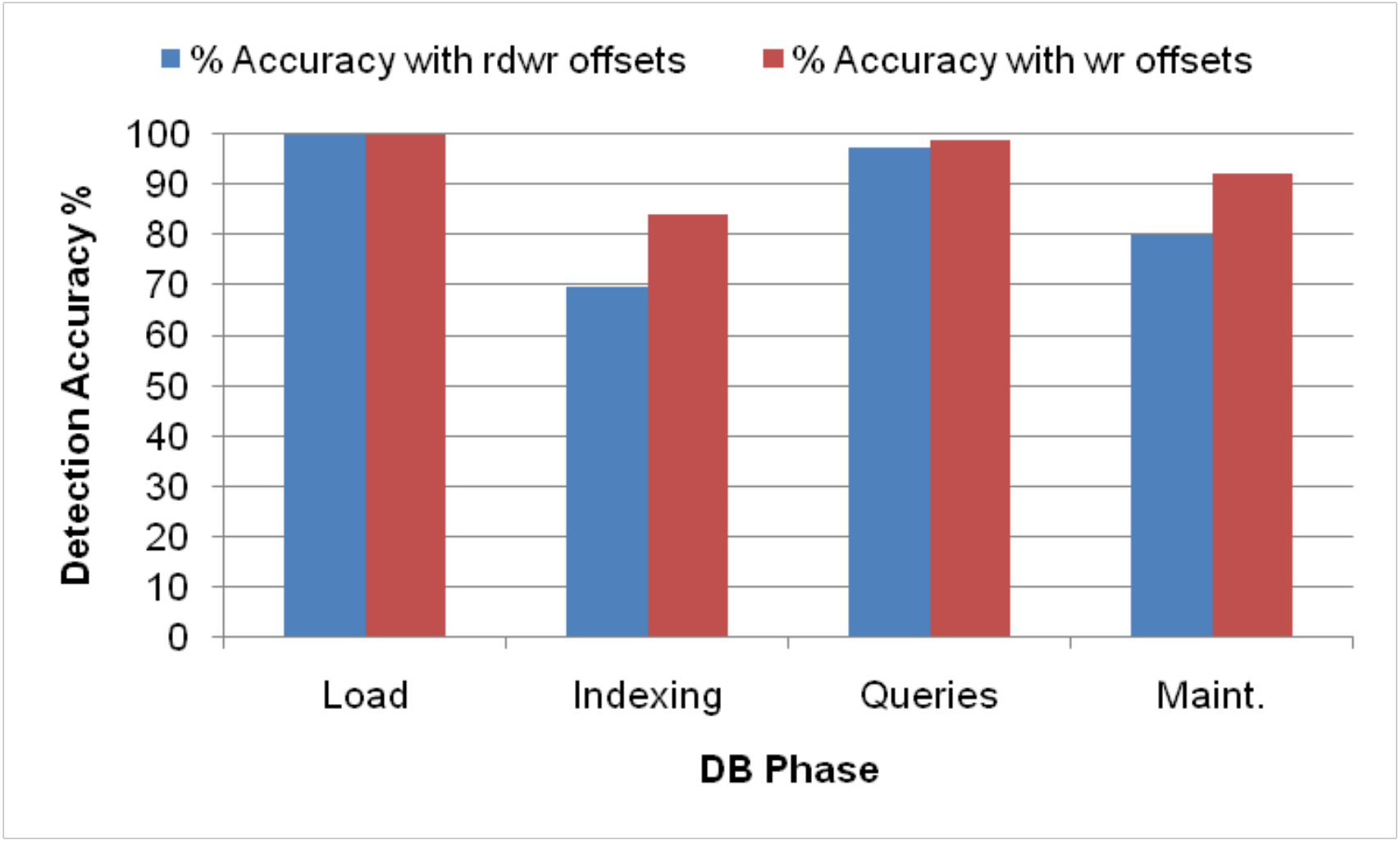
- In an online deployment, over time, the trace snippets that the SVM based multi-class classifier flags as 'Unknown' are collected.
- These are labeled with a special 'Unknown' class label and the system is re-trained by augmenting it with this class.
- Past snippets are re-classified to see if any of them join this class.
- This works well in practice.

Distinguishing phases in a database workload

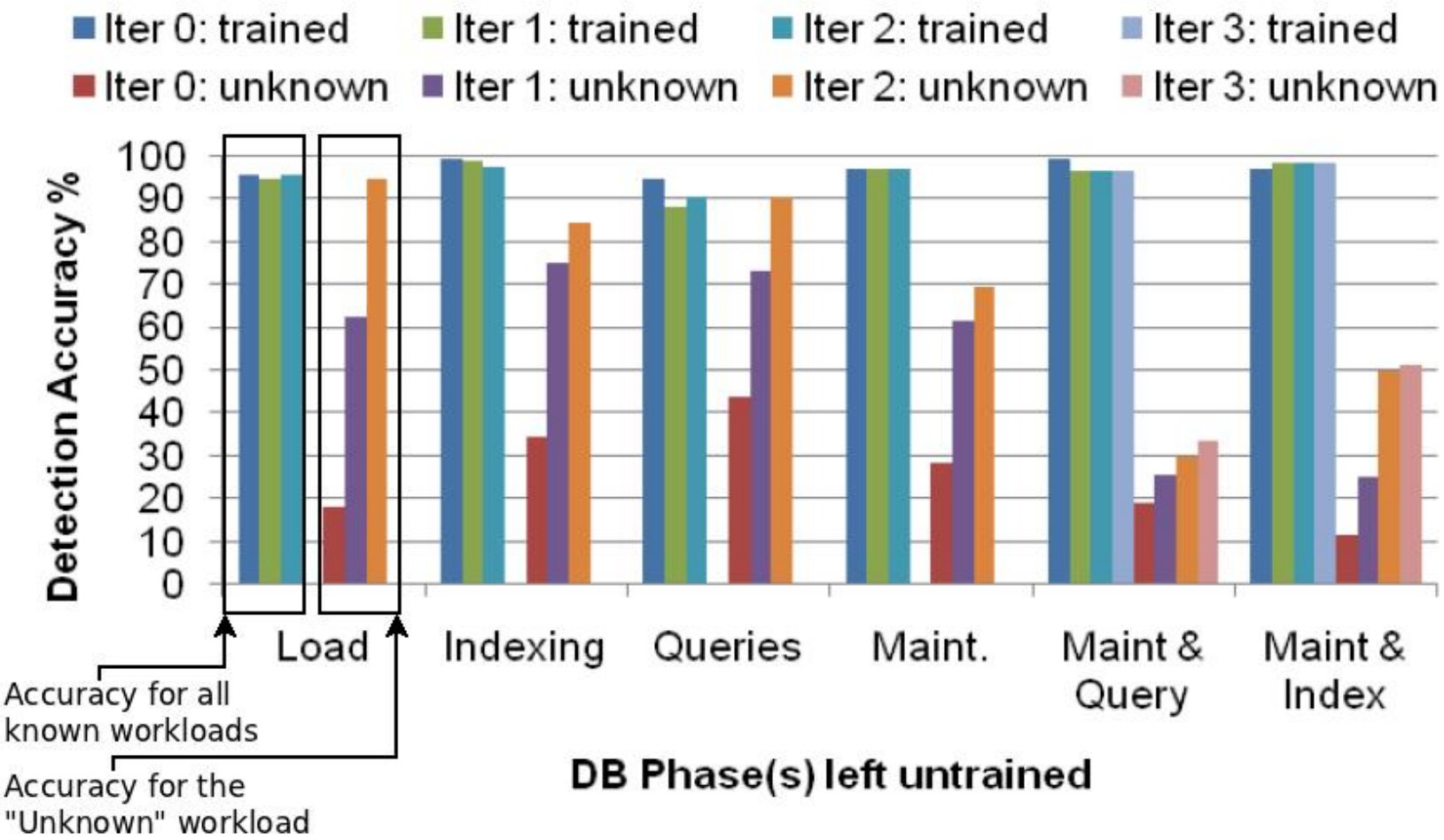
- Workload used: TPC-DS
  - phases considered: Load, Query, Indexing, Maintenance.

- PostgreSQL database runs inside a VM with 4GB RAM available and the image residing on a NFS server.
- The VM's host machine is an 8-core Xeon-5520 with 8GB RAM.
- For training LoadIQ, traces are collected while the database goes through various phases and each trace is labeled with its phase name.
- The collected traces are divided into 60-second snippets and read-write histograms are generated for each.

Results: Fully trained system



Results: Iterative training over Unknown traces



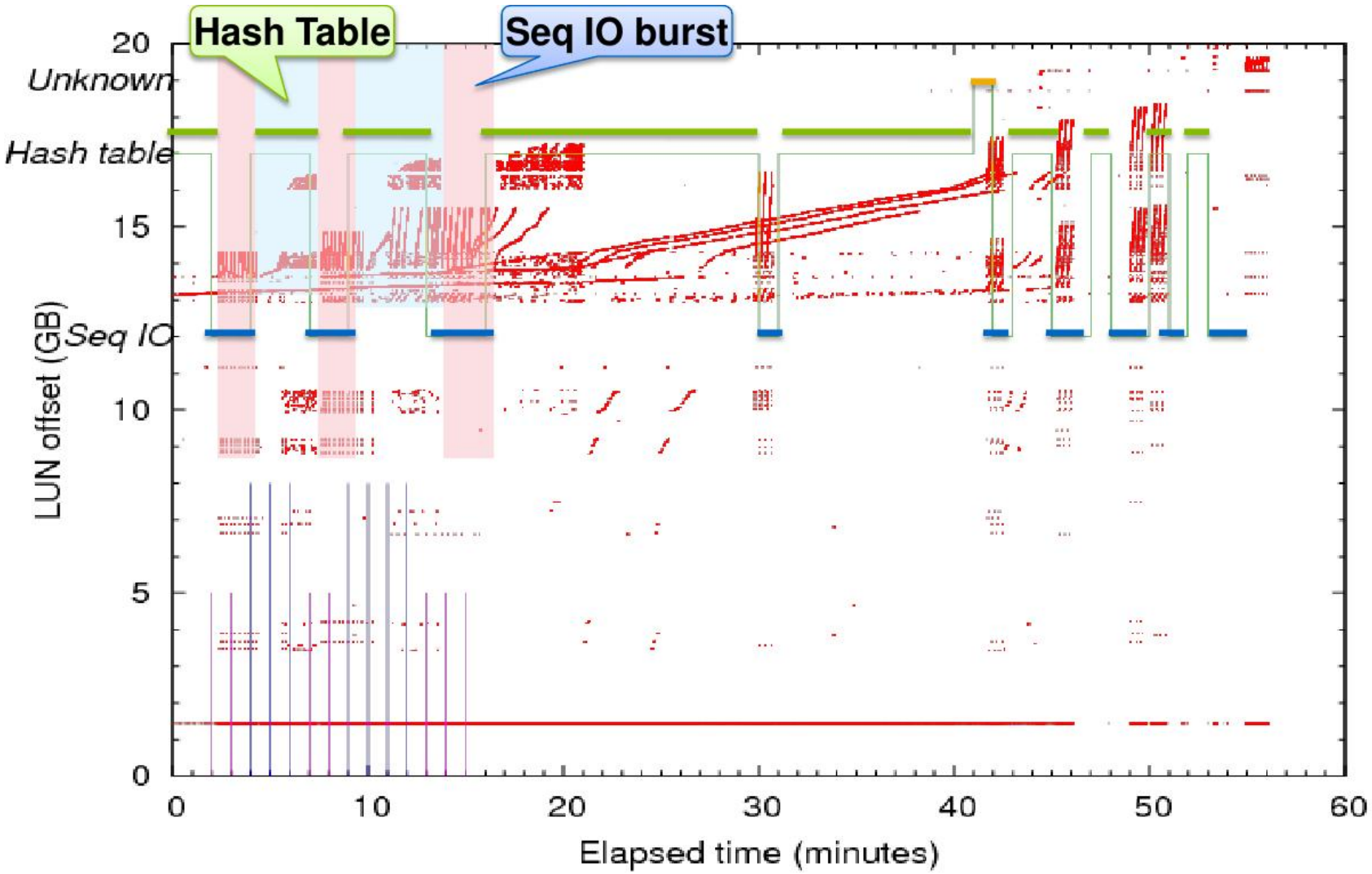
Phases in a production OLAP workload

- Aim: Use LoadIQ to automate detecting the recurrence of special/anomalous workload behaviors in a production environment.

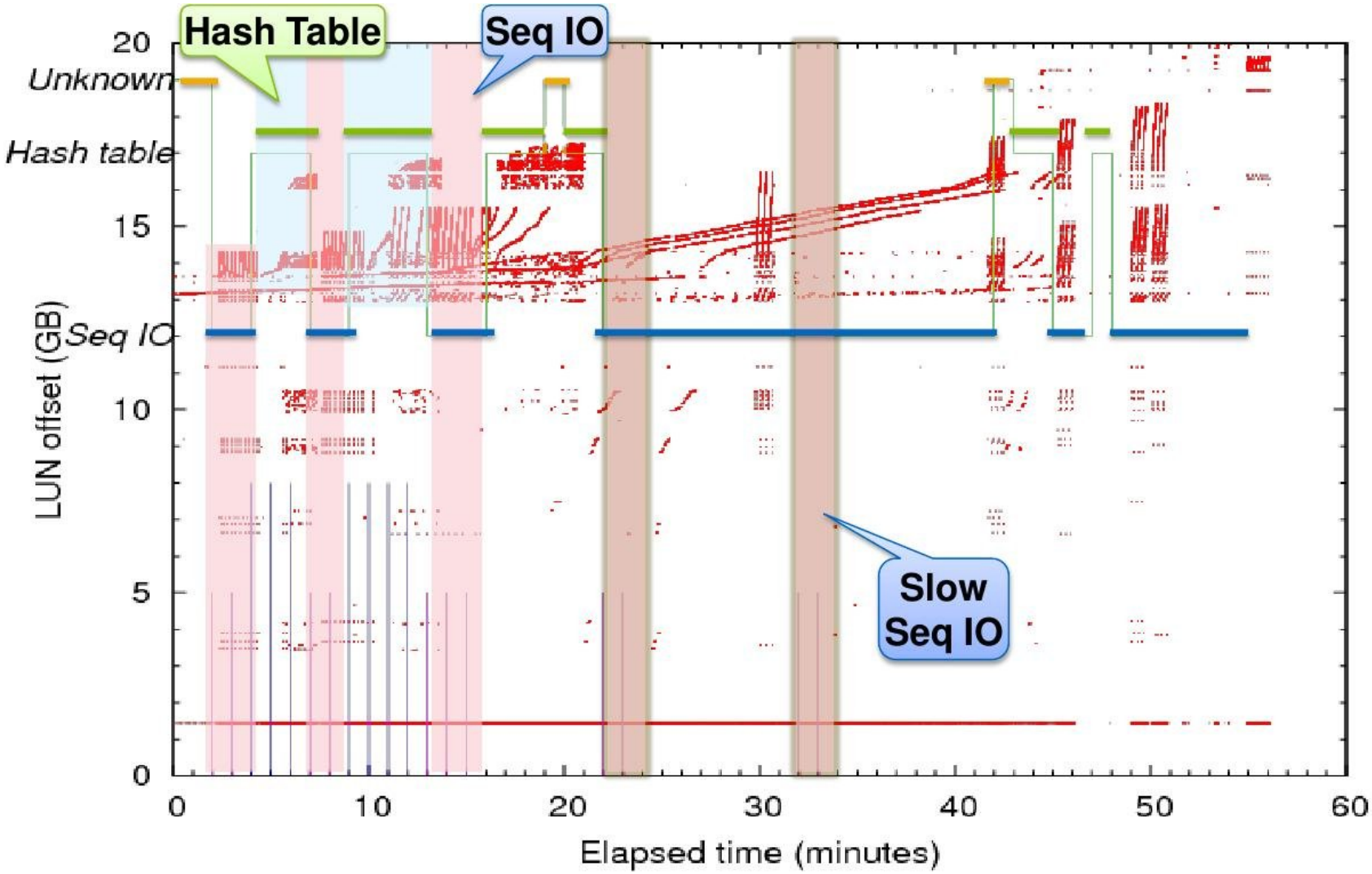
- Workload: A production enterprise data warehousing application in a 10-node cluster configured to use a SAN backend.
- 50 LUNs each of size 20GB.
- Traces: Post-host-cache SCSI request trace on all LUNs
  - 188K reads and 250K writes per LUN spread over 56 minutes.
- Phases considered: Hash table accesses and sequential IO bursts.

- Trace collection time: 60secs
- Analysis time: 4secs
- Retraining for "Unknown" class: 4secs

Spotting special workload behavior: OLAP



Interleaved sequential IO: OLAP



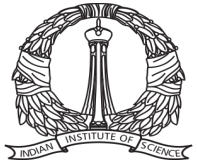
Conclusions

- A ML-based tool for identifying various phases in an application, from its live storage trace.
  - Accuracy $> 93\%$ in many cases.
  - Can flag certain traces as 'Unknown'. Retraining can be used to improve accuracy.

Conclusions

- A ML-based tool for identifying various phases in an application, from its live storage trace.
  - Accuracy $> 93\%$ in many cases.
  - Can flag certain traces as 'Unknown'. Retraining can be used to improve accuracy.

- Open questions:
  - How to separate concurrent IO patterns in a combined trace?
  - A quantifiable confidence measure of the classification output is needed. Can this be provided?
  - How to exploit phase knowledge in system design?



References

Baker, M. et al. (1991). "Measurements of a Distributed File System". In: *Proceedings of the 13th Symposium on Operating Systems Principles*.

Chen, Y., M. R. Gupta, and B. Recht (2009). "Learning Kernels from Indefinite Similarities". In: *International Conference on Machine Learning*.

Gu, D. and C. Verbrugge (2006). *A survey of phase analysis: Techniques, evaluation and applications*. Tech. rep. Citeseer.

Leung, A. et al. (2008). "Measurement and Analysis of Large-Scale File System Workloads". In: *Proceedings of the USENIX 2008 Annual Technical Conference*.

Madhyastha, T. and D. Reed (1997). "Input/Output Access Pattern Classification Using Hidden Markov Models". In: *Workshop on Input/Output in Parallel and Distributed Systems*.

Mesnier, Michael P. and Jason B. Akers (2011). "Differentiated storage services". In: *SIGOPS Oper. Syst. Rev.* 45.1 (Feb. 2011), pp. 45–53. ISSN: 0163-5980. DOI: 10.1145/1945023.1945030. URL: http://doi.acm.org/10.1145/1945023.1945030.

Riska, A. and E. Riedel (2006). "Disk Drive Level Workload Characterization". In: *Proceedings of the USENIX 2006 Annual Technical Conference*.

Roselli, D., J. Lorch, and T. E. Anderson (2000). "A comparison of file system workloads". In: *Proceedings of the USENIX 2000 Annual Technical Conference*. San Diego, California.

Yadwadkar, N. et al. (2010). "Discovery of Application Workloads from Network File Traces". In: *Proceedings of the Eighth USENIX Conference on File and Storage Technologies (FAST 2010)*.

Zhang, Xi, Alma Riska, and Erik Riedel (2008). "Characterization of the E-commerce Storage Subsystem Workload". In: *QEST*, pp. 297–306.

Grateful acknowledgments

- IISc GARP funds
- USENIX student grant program
- NetApp research grant

Thank You !!!