


# Instances of Reinforcement Learning, Supervised Learning and Unsupervised Learning with Application to Storage Systems

**Nima Mohammadi** <nima.mohammadi@ipm.ir>
Data Storage, Networks & Processing (DSN) Laboratory
Digital Media Laboratory (DML)

Under supervision of Prof. Asadi and Prof. Rabiee

# Four Topics

1. Disk Performance Prediction
2. Dynamic Resource Allocation
3. Proactive Failure Prediction
4. Representative Sampling of IO Traces

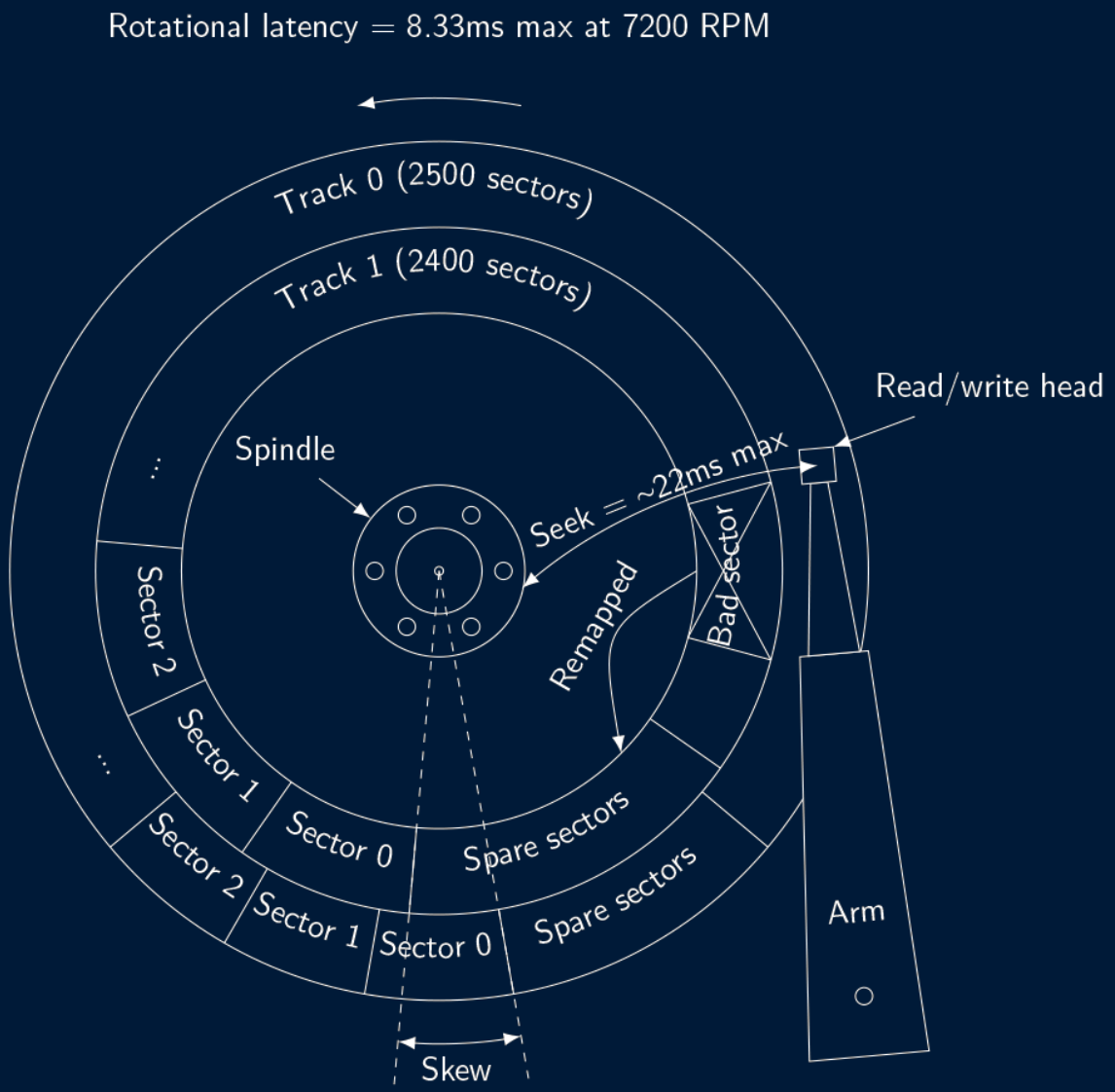
Disk Performance Prediction

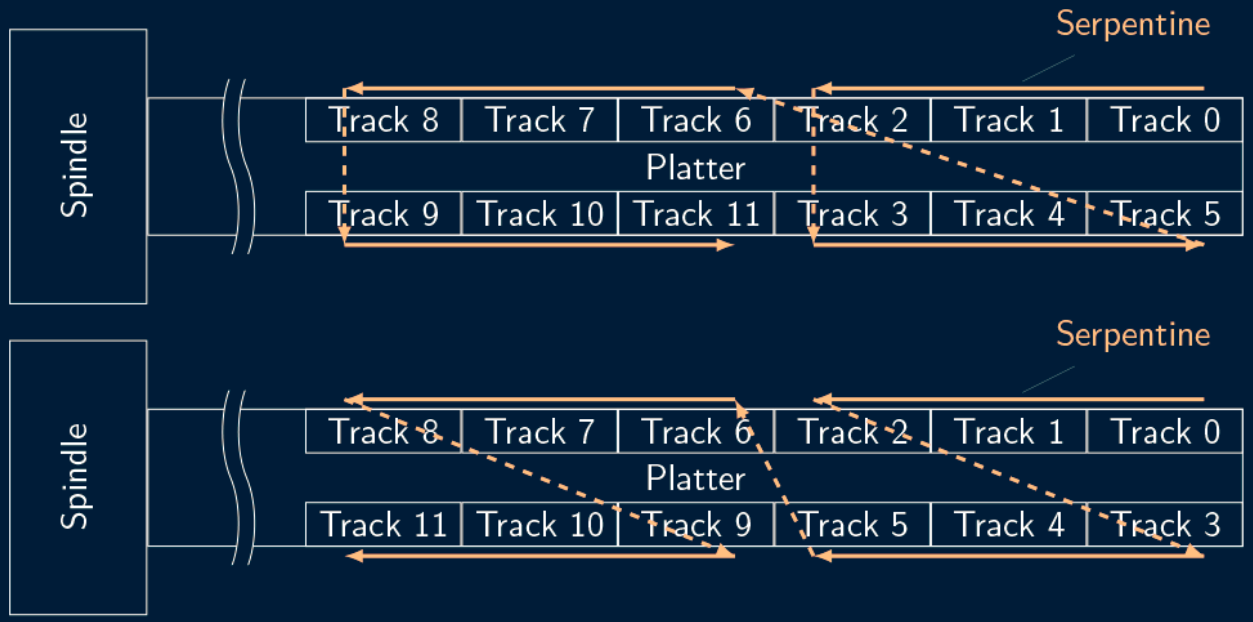
**Fourier-Assisted Machine Learning of Hard Disk Drive Access Time Models** (by Adam Crume et al.)

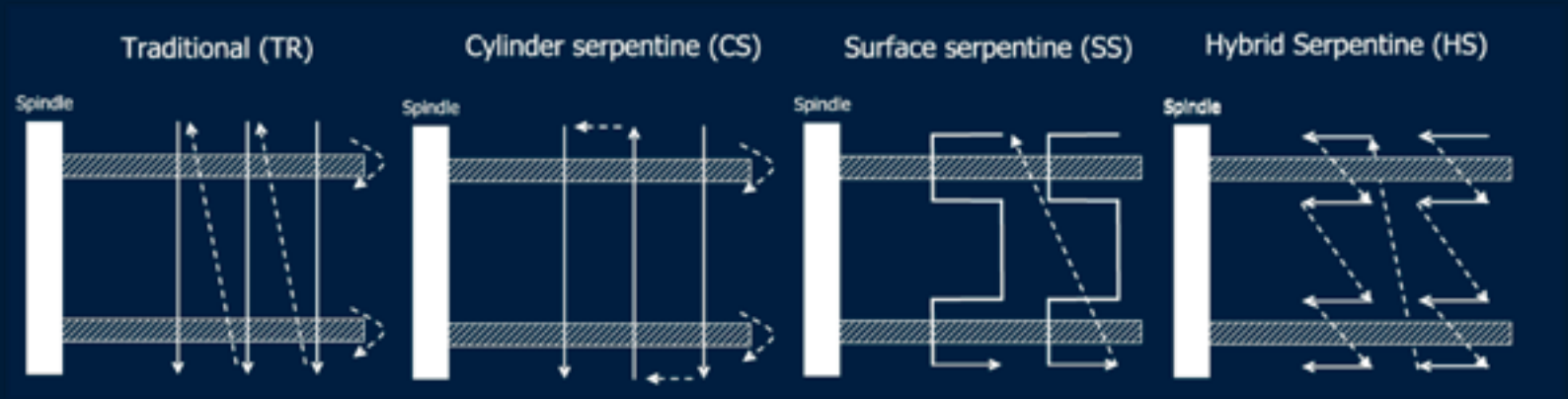
# Outline

- Use cases of predicting disk drive performance
- Hard disk internals
- Using machine learning techniques
- Online vs. offline prediction
- Existing approaches
- Workload characteristics
- Access time breakdown
- Difficulties of predicting access time
- Input augmentation
- Fourier transform
- Neural Nets
- Results and Conclusions

- **Use cases:**
  - **System simulations**
  - **File system design**
  - **Quality of service / real-time guarantees**


Rotational latency = 8.33ms max at 7200 RPM
Track 0 (2500 sectors)
Track 1 (2400 sectors)
Read/write head
Spindle
Seek = ~22ms max
Bad sector
Remapped
Sector 2
Sector 1
Sector 0
Spare sectors
Sector 2
Sector 1
Sector 0
Spare sectors
Arm
Skew

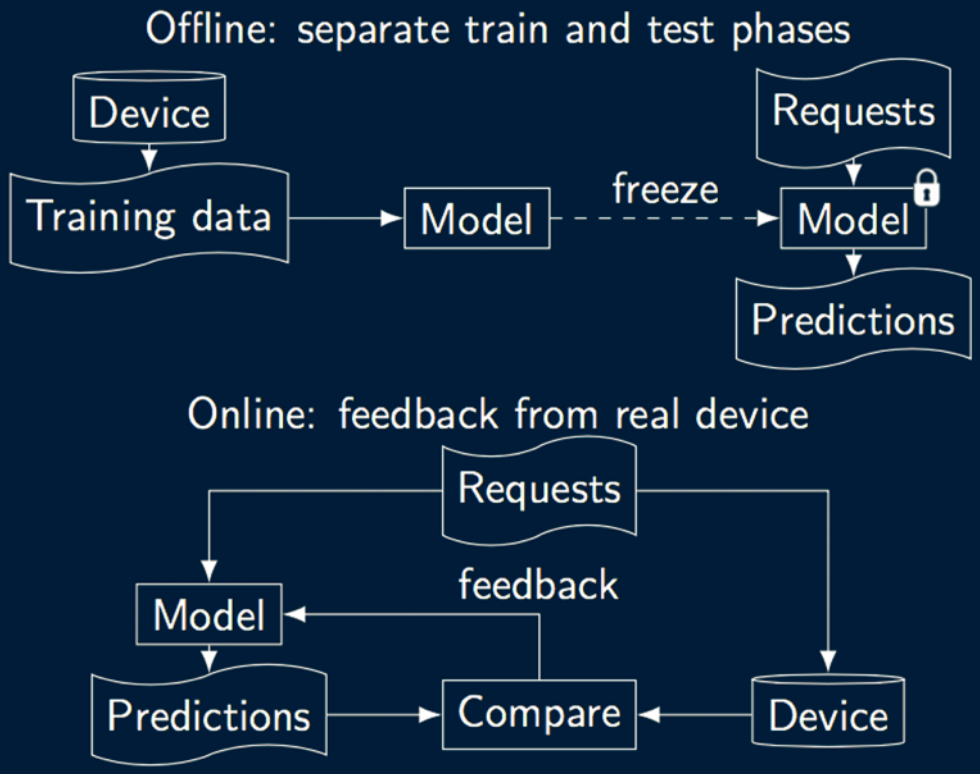

Serpentine
Spindle
Track 8
Track 7
Track 6
Track 2
Track 1
Track 0
Platter
Track 9
Track 10
Track 11
Track 3
Track 4
Track 5
Serpentine
Spindle
Track 8
Track 7
Track 6
Track 2
Track 1
Track 0
Platter
Track 11
Track 10
Track 9
Track 5
Track 4
Track 3


Traditional (TR)
Cylinder serpentine (CS)
Surface serpentine (SS)
Hybrid Serpentine (HS)
Spindle
Spindle
Spindle
Spindle

- **What makes this task complicated:**
  - **Queueing**
  - **Scheduling**
  - **Caching**
  - **Readahead**
  - **Write-back**

- Short term goals (this presentation):
  - Automated
  - Fast
- Long term goals (future work):
  - Flexible
  - Future-proof
  - Device-independent

- **System simulations (offline)**
- **File system design (offline)**
- **Quality of service / real-time guarantees (offline/online)**




- **For each time slice:**

$r_0 \rightarrow prediction_0$

$r_1 \rightarrow prediction_1$

. .

. .

. .

$r_n \rightarrow prediction_n$

} → average → compare with real average


- **For each time slice:**

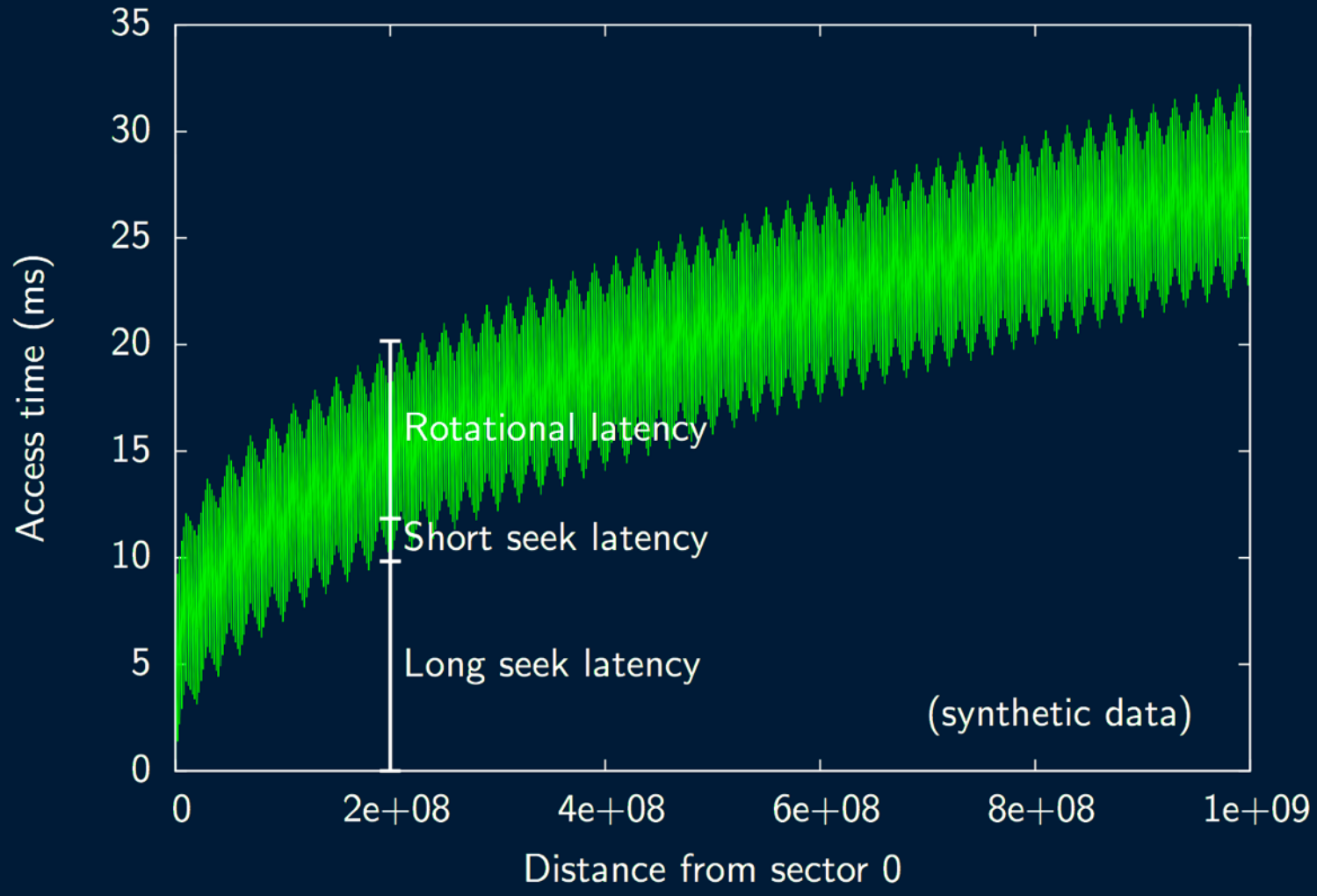
$r_0$
$r_1$
.
.
.
$r_n$

**→ aggregate → predict average → compare with real average**

- **Characteristics:**
  - **Random**
  - **Read-only**
  - **Single-sector**
  - **Full utilization**
  - **First serpentine**
- **Minimizes:**
  - **Caching**
  - **Readahead**
  - **Write-back**
  - **Transfer time**
  - **Request arrival time sensitivity**
  - **Track length variation**

**Workload emphasizes access time (which is a hard problem by itself) and de-emphasizes everything else. Other workloads are future work.**

## Access Time Breakdown



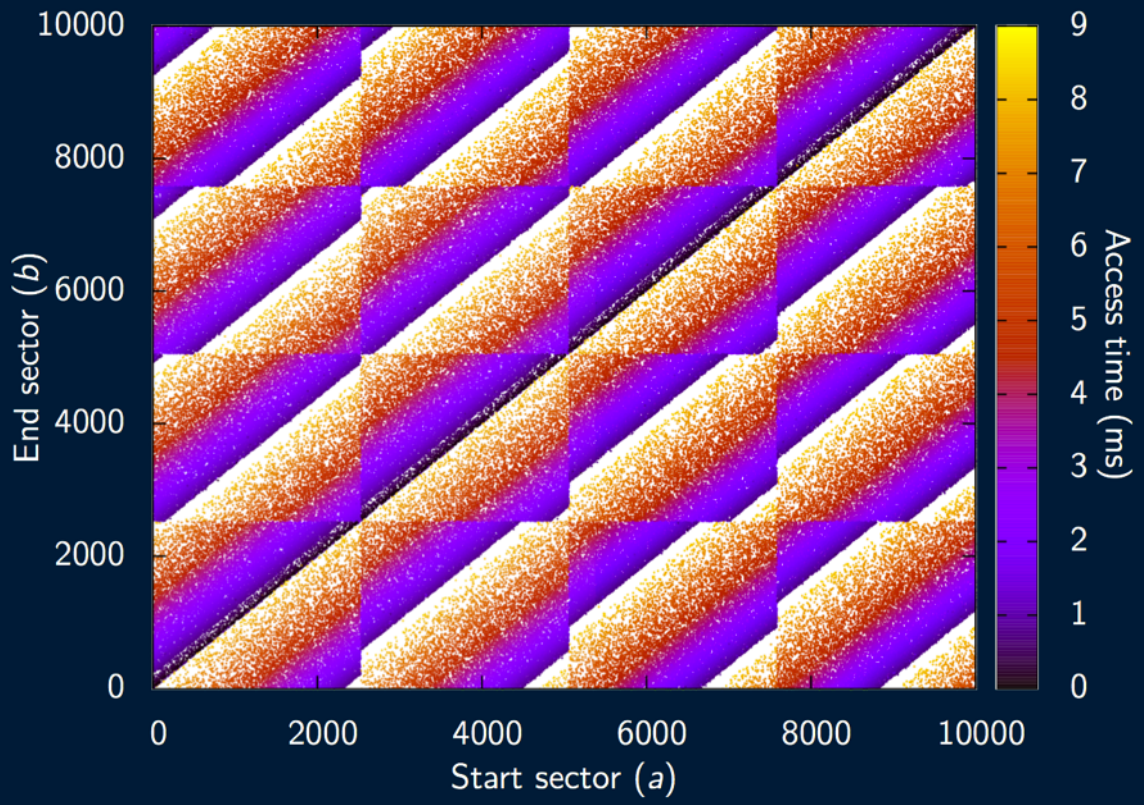
- Rotational layout
- Serpentines
- Sector sparing
- Skew

They all have periodicity in common!

Most of machine learning techniques cannot directly address the problem of predicting periodic functions well.

# Modeling Hard Disk Drive Access Time

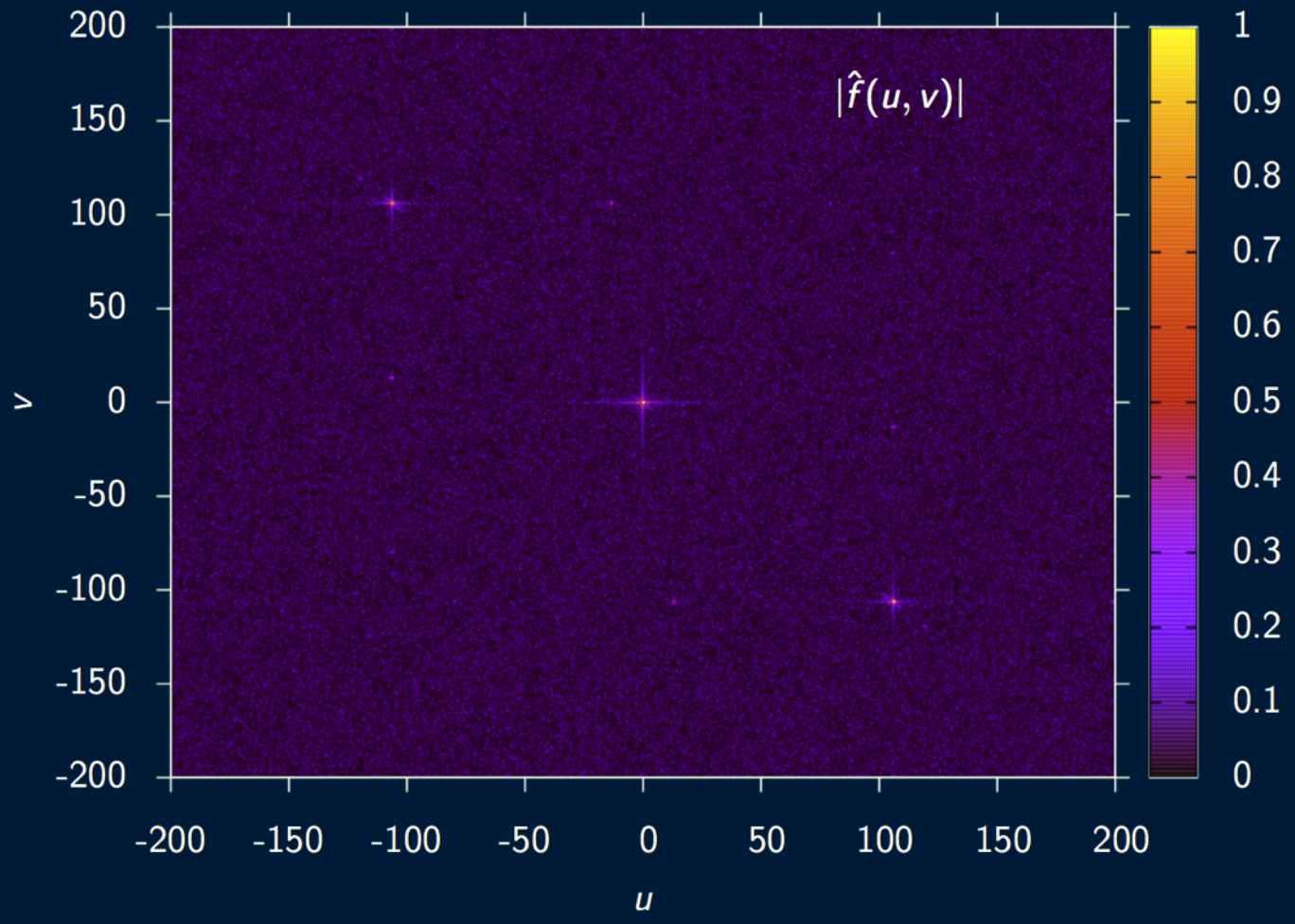
## Access Time



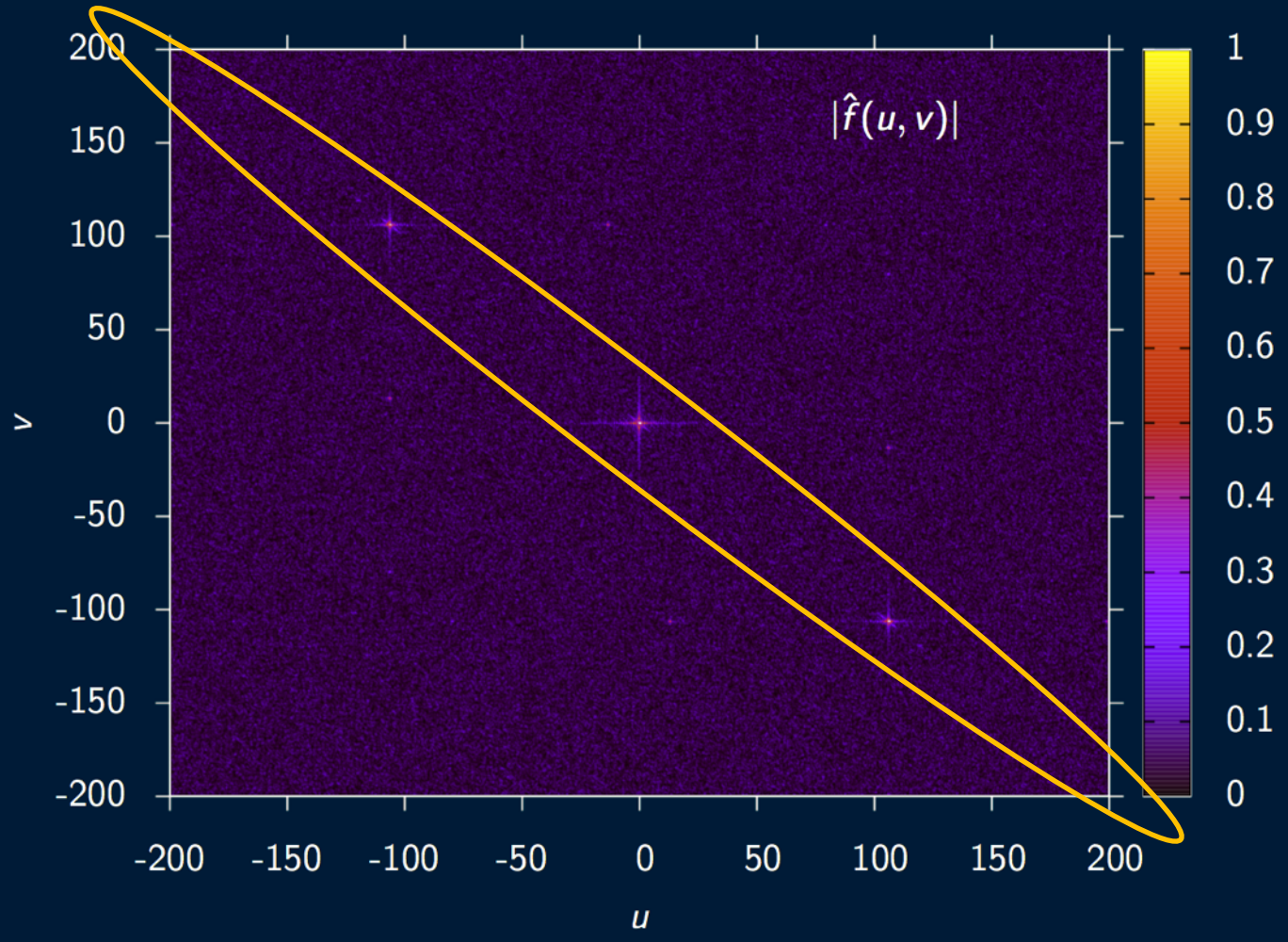

**Full table is 1 billion by 1 billion entries, would take approximately 500 million years to capture data and 3.5 exabytes to store it. Extremely sparse sampling is required, must compute on the fly.**

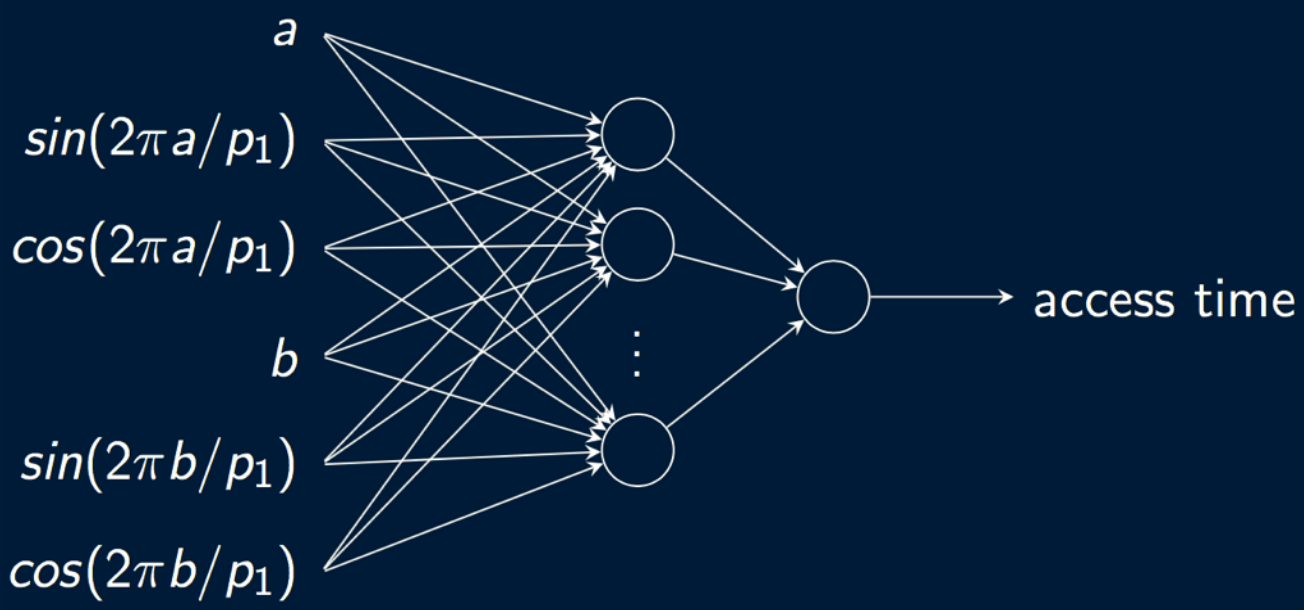
**Augmenting features by adding sines and cosines to input**

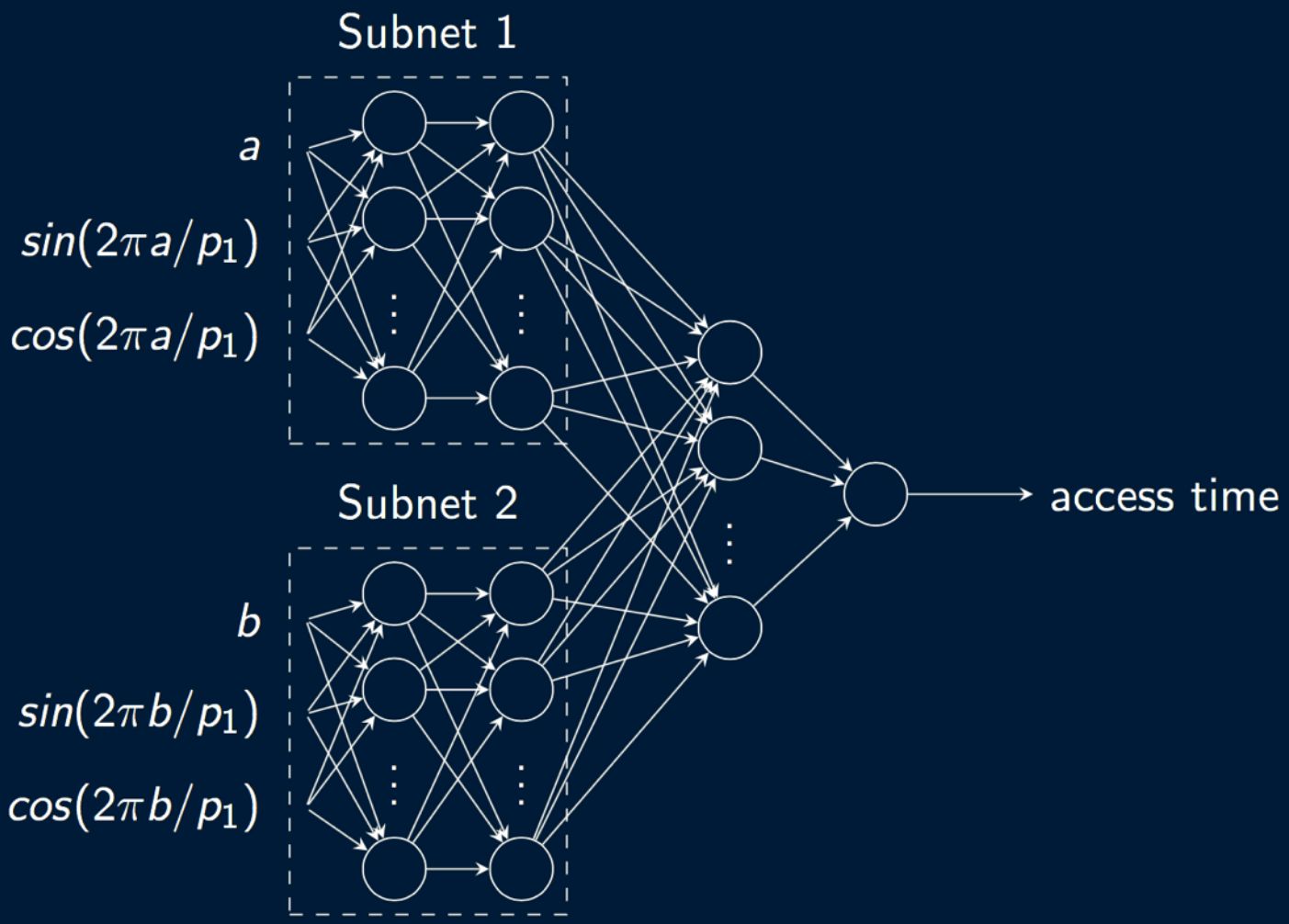
$$\begin{pmatrix} a \\ b \end{pmatrix} \rightarrow \begin{pmatrix} a \\ \sin(2\pi a/p_1) \\ \cos(2\pi a/p_1) \\ \sin(2\pi a/p_2) \\ \cos(2\pi a/p_2) \\ \vdots \\ b \\ \sin(2\pi b/p_1) \\ \cos(2\pi b/p_1) \\ \sin(2\pi b/p_2) \\ \cos(2\pi b/p_2) \\ \vdots \end{pmatrix}$$


$|\hat{f}(u,v)|$
200
150
100
50
0
-50
-100
-150
-200
v
-200
-150
-100
-50
0
50
100
150
200
u
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

**Search on diagonal to limit computations.**



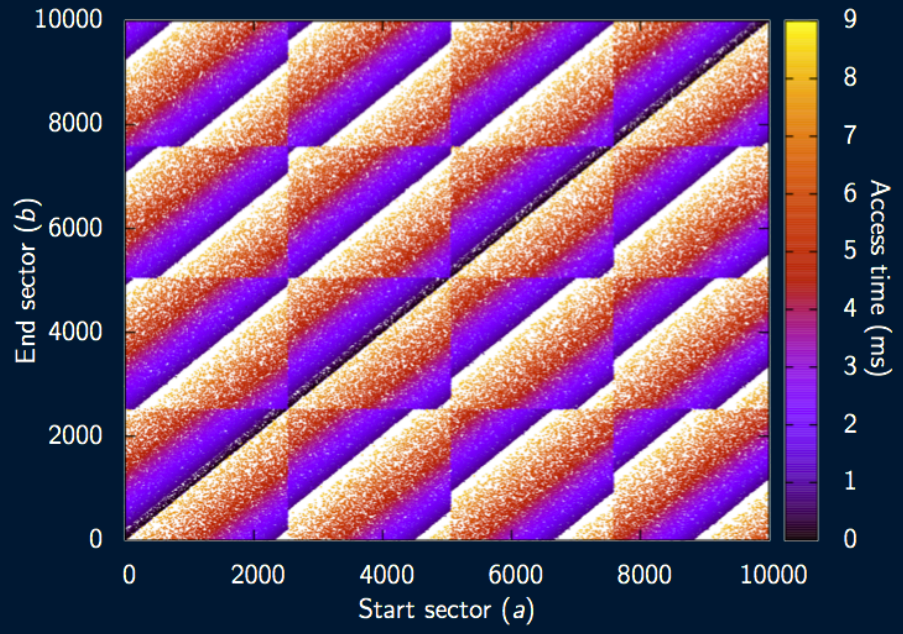

a
sin(2πa/p1)
cos(2πa/p1)
b
sin(2πb/p1)
cos(2πb/p1)
access time

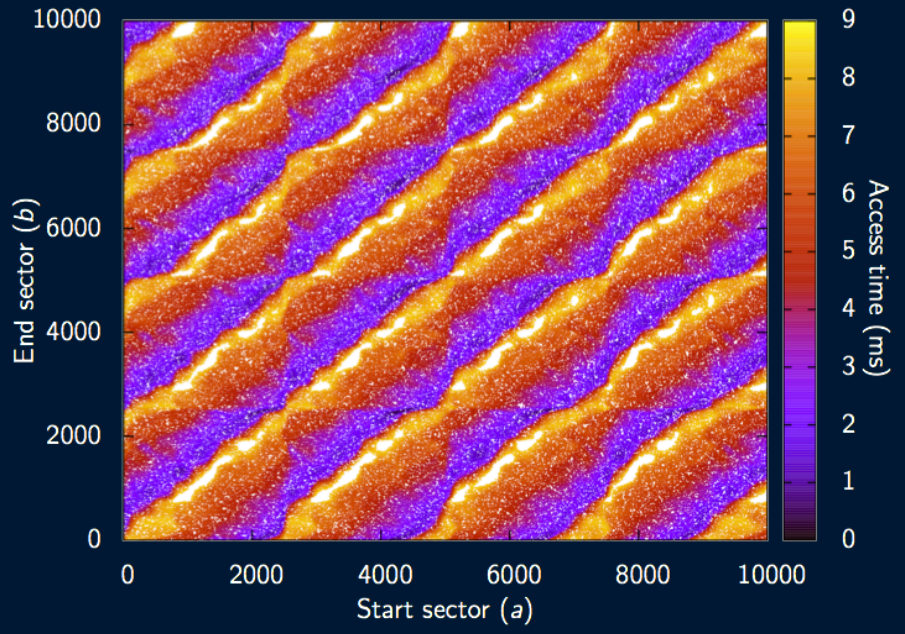

Subnet 1
a
$sin(2\pi a/p_1)$
$cos(2\pi a/p_1)$
Subnet 2
b
$sin(2\pi b/p_1)$
$cos(2\pi b/p_1)$
access time

# Modeling Hard Disk Drive Access Time

Neural Nets

Weight Sharing



# Modeling Hard Disk Drive Access Time

## Results

| | Configuration | Error (ms) |
|---|---|---|
| | constant value | $2.013 \pm 0.024$ |
| Decision trees | no periods, w/o bagging | $2.075 \pm 0.014$ |
| Decision trees | no periods, with bagging | $2.067 \pm 0.001$ |
| Decision trees | 6 random periods, w/o bagging | $2.019 \pm 0.013$ |
| Decision trees | 6 random periods, with bagging | $2.015 \pm 0.013$ |
| Decision trees | 6 periods, w/o bagging | $1.649 \pm 0.154$ |
| Decision trees | 6 periods, with bagging | $1.123 \pm 0.009$ |
| Neural nets | no periods, w/o subnets | $2.014 \pm 0.034$ |
| Neural nets | no periods, with subnets | $2.012 \pm 0.019$ |
| Neural nets | 6 random periods, w/o subnets | $1.924 \pm 0.176$ |
| Neural nets | 6 random periods, with subnets | $1.992 \pm 0.059$ |
| Neural nets | 6 periods, w/o subnets | $0.954 \pm 0.052$ |
| Neural nets | 6 periods, with subnets | $0.830 \pm 0.031$ |

- Periodicity information improves multiple algorithms
- High-level assumption, likely to apply to many devices
- Machine learning of per-request latencies is possible

Dynamic Resource Allocation

**VCONF: a reinforcement learning approach to virtual machines auto-configuration.** (by Jia Rao et al.)

- Reinforcement learning (RL) is a process of learning by interactions with dynamic environment, which generates the optimal control policy for a given set of states.
  - It requires no domain knowledge of the controlled system and is able to generate policies optimizing a long-term goal.

- VCONF: using model-based RL algorithms for scalability and adaptability.
  - By maximizing the long run reward (based on summarized performance of each VM), VCONF automatically directs each virtual machine configuration to a good (if not optimal) configuration.

# RL and VM Auto-configuration

- RL is concerned with how an agent ought to take actions in a dynamic environment so as to maximize long term rewards defined on a high level goal

- Two advantages:
  1. No need for a model of either the system in consideration or the environment dynamics.
  2. RL is able to capture the delayed effect in a decision-making task.

- RL Output: a policy that maps the current state of the agent to the best action it could take.
  - The "goodness" of an action in a state is measured by a value function which estimates the future accumulated rewards by taking this action.

- The reward info is propagated backward temporally, eventually leading to an approximation of the value function. The optimal policy is essentially choosing the action that maximizes the value function in each state.

# RL and VM Auto-configuration (cont.)

- How it fits our problem?
  - Agent would be a controller in dom0.
  - States are VM resource allocations
  - Possible changes to the allocations form the set of Actions
  - Each time the controller adjusts the VM configurations, it receives performance feedback from individual VMs.
  - With sufficient interactions, the controller obtains good estimations of the "goodness" of decisions.

  ❖ Starting from an arbitrary initial setting, the controller is able to lead the VMs to optimal configurations by following the optimal policy.

# Markov Decision Process

- A RL problem is usually modeled as a Markov Decision Process (MDP)
  - A set of environment states $S$
  - A set of actions $A$
  - MDP is defined by
    - the transition probability

$$P_a(s, s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$$

    - and an immediate reward function

$$R = \mathbb{E}[r_{t+1} | s_t = s, a_t = a, s_{t+1} = s']$$

  - The value function of taking action $a$ in state $s$ can be defined as:

$$Q(s, a) = \mathbb{E}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\}$$

# Formulation of VM Conf. as RL Task

- The VM configuration task is naturally formulated as a continuing discounted MDP.
- The goal is to optimize the overall VM(s) performance.
  - We define the reward function based on individual VM's application level performance.
  - The state spaces are the hardware configuration of VMs which are fully observable in the driver domain.
  - Actions are the combination of the change to the configurable parameters.

- The performance of individual VM is measured by a score which is the ratio of current throughput (thrpt) to a reference throughput plus possible penalties when response time (resp) based SLAs (Service Level Agreement) are violated.

$$score = \frac{thrpt}{ref\_thrpt} - penalty$$

$$penalty = \begin{cases} 0 & \text{if } resp \leq SLA \\ \dfrac{resp}{SLA} & \text{if } resp > SLA \end{cases}$$

- A low score indicates either lack of resource or interference between VMs.
- Immediate reward is the summarized scores over all VMs:

$$reward = \begin{cases} \sqrt[n]{\prod_{i=1}^{n} w_i * score_i} & \text{if for all } score_i > 0 \\ -1 & \text{otherwise} \end{cases}$$

- The state space is defined to be the set of possible VM configurations.
- In the driver domain (dom0), VM configurations are fully observable. States defined on the configurations are deterministic in that $P_a(s, s') = 1$, which simplifies the RL problem.
- We define the RL state as the global resource allocations:

$$(mem_1, time_1, vcpu_1, \ldots, mem_n, time_n, vcpu_n)$$

- For each of the three configurable parameters, possible operations can be either *increase*, *decrease* or *nop*.
- The actions for the RL task are defined as the combinations of the operations on each parameter.
- For parameters like time and mem that are continuous, they are quantified by defining change steps. CPU number is incremented or decremented by one at a time.

# Solutions to the RL Task

- The solution to a RL task is an optimal policy that maximizes the cumulative rewards at each state.
- Equivalent to finding an estimation of $Q(s,a)$ which approximates its actual value.
- The basic RL algorithms in experience-based solution are called temporal-difference (TD) methods, which update $Q(s,a)$ at each time a sample $(s_t, a_t, r_{t+1})$ is collected:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha * [r_{t+1} + \gamma * Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

# Adaptability and Scalability

- **Adaptability** is the ability of RL algorithms to revise the existing policy in response to the change of the environment.

- **Scalability** issues refer to the problem that the number of Q values grows exponentially with the state variables.

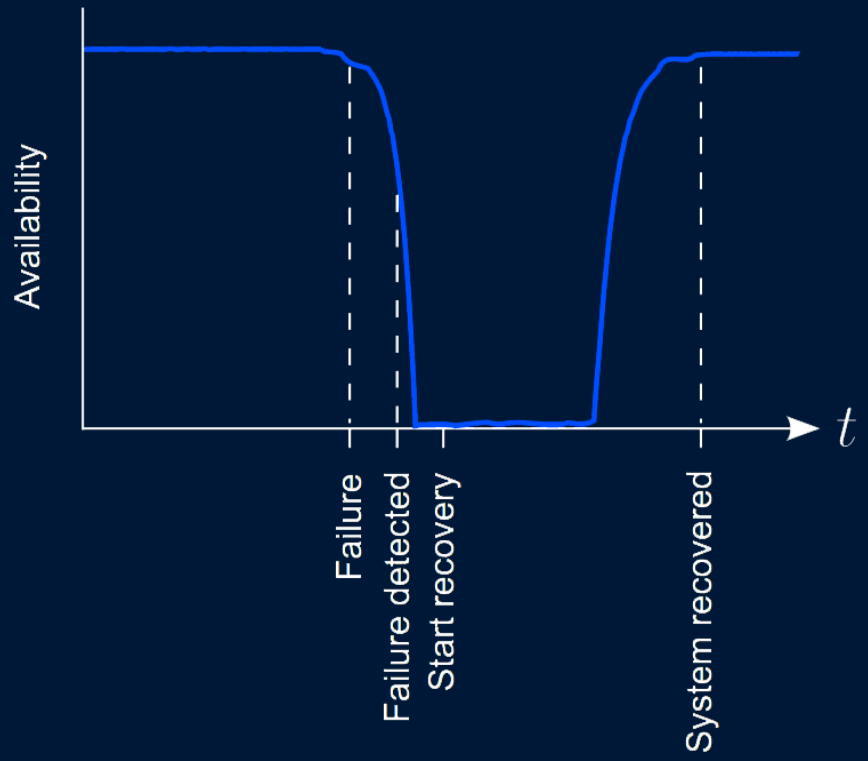
- **Adaptability** is the ability of RL algorithms to revise the existing policy in response to the change of the environment.
- To **adapt** current policy to a new one, the RL agent needs to perform a certain amount of **exploration** actions, which are believed to be **suboptimal** actions leading to **bad rewards**.
- The RL algorithm usually requires **a long time** before a new policy can be derived. This is **not acceptable** for online policy generation tasks like VM auto-configuration **in production systems**.
- **Scalability** issues refer to the problem that the number of Q values grows exponentially with the state variables.

- Scalability issues refer to the problem that the number of Q values grows exponentially with the state variables.
- The convergence of the optimal policy depends critically on the assumption that each table entry be visited at least once.
- Even if the storage and computation complexity for a large Q table are not a concern, the time required to collect sample rewards to populate the Q table is prohibitively long.
- Instead of updating each $Q(s,a)$ value directly from the immediate reward recently collected, VCONF employs environment models to generate simulated experiences for value function estimation.
- The authors have selected standard multi-layer feed-forward back propagation neural network (MLP) with sigmoid activations and linear output to represent the environment model.

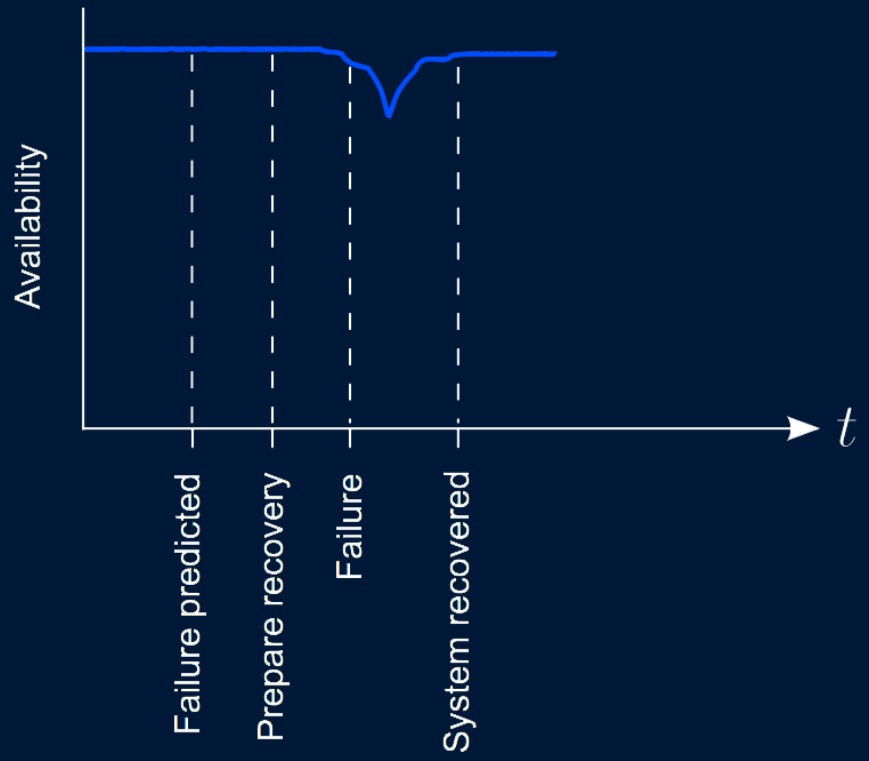
Proactive Failure Prediction

**Proactive drive failure prediction for large scale storage systems.** (by Zhu et al.)

# Failure Management

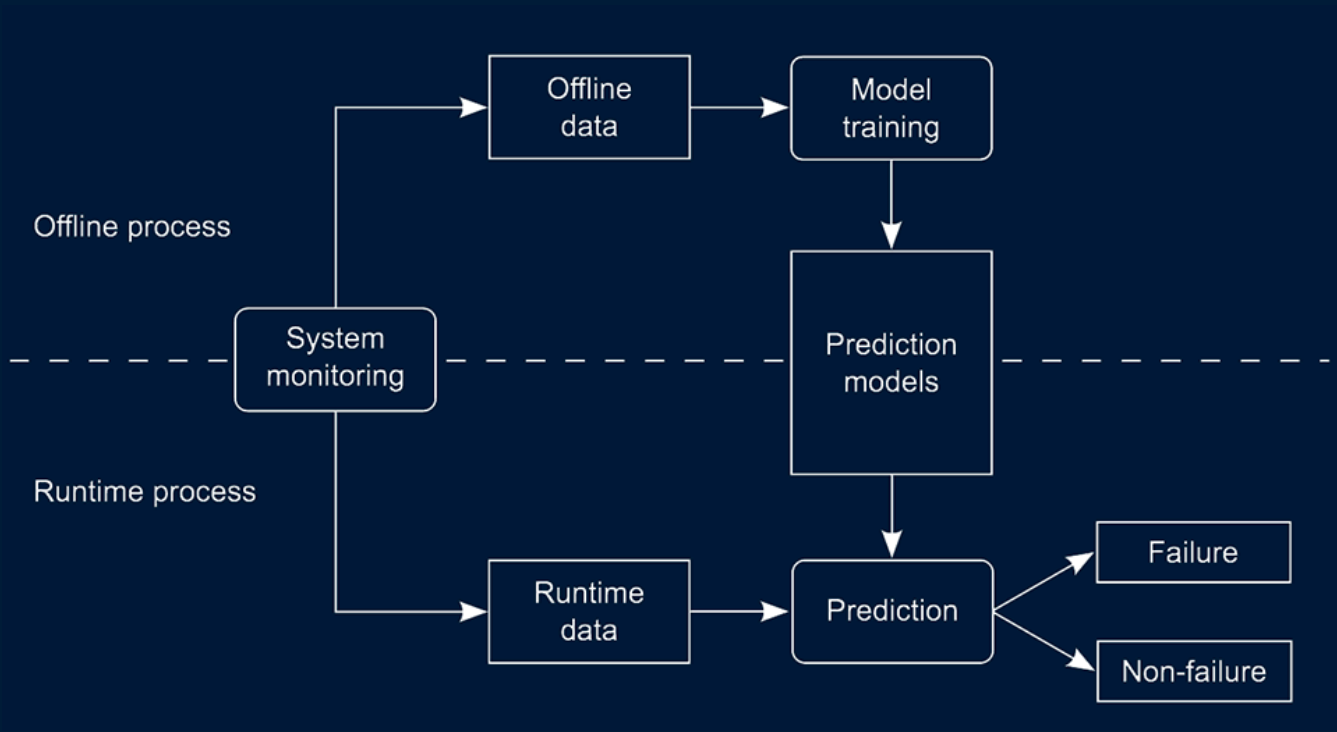



Reactive approach

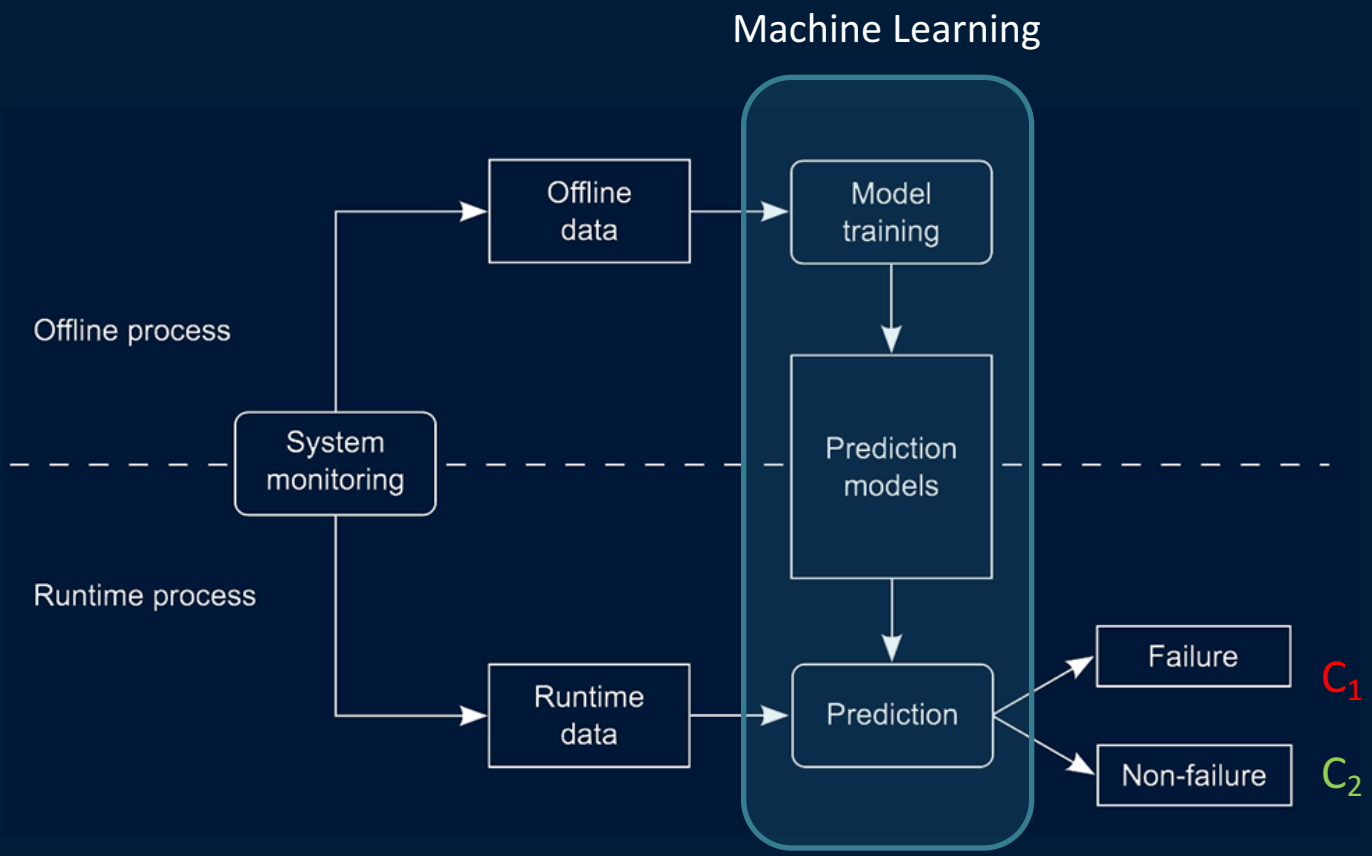



Proactive approach

# Failure Management Framework



# Failure Management Framework



# Failure Management

## Healthy drive

| Serial no. | Temp1 | FlyHeight1 | Servo8 | ReadError17 | WriteError | ··· |
|---|---|---|---|---|---|---|
| 100192 | 29 | 7958 | 0 | 0 | 6 | ··· |
| 100192 | 29 | 7957 | 0 | 0 | 13 | ··· |
| 100192 | 58 | 7971 | 0 | 0 | 36 | ··· |
| 100192 | 57 | 7916 | 0 | 0 | 36 | ··· |
| 100192 | 35 | 7962 | 0 | 0 | 36 | ··· |
| 100192 | 37 | 7969 | 0 | 0 | 37 | ··· |

$C_1$

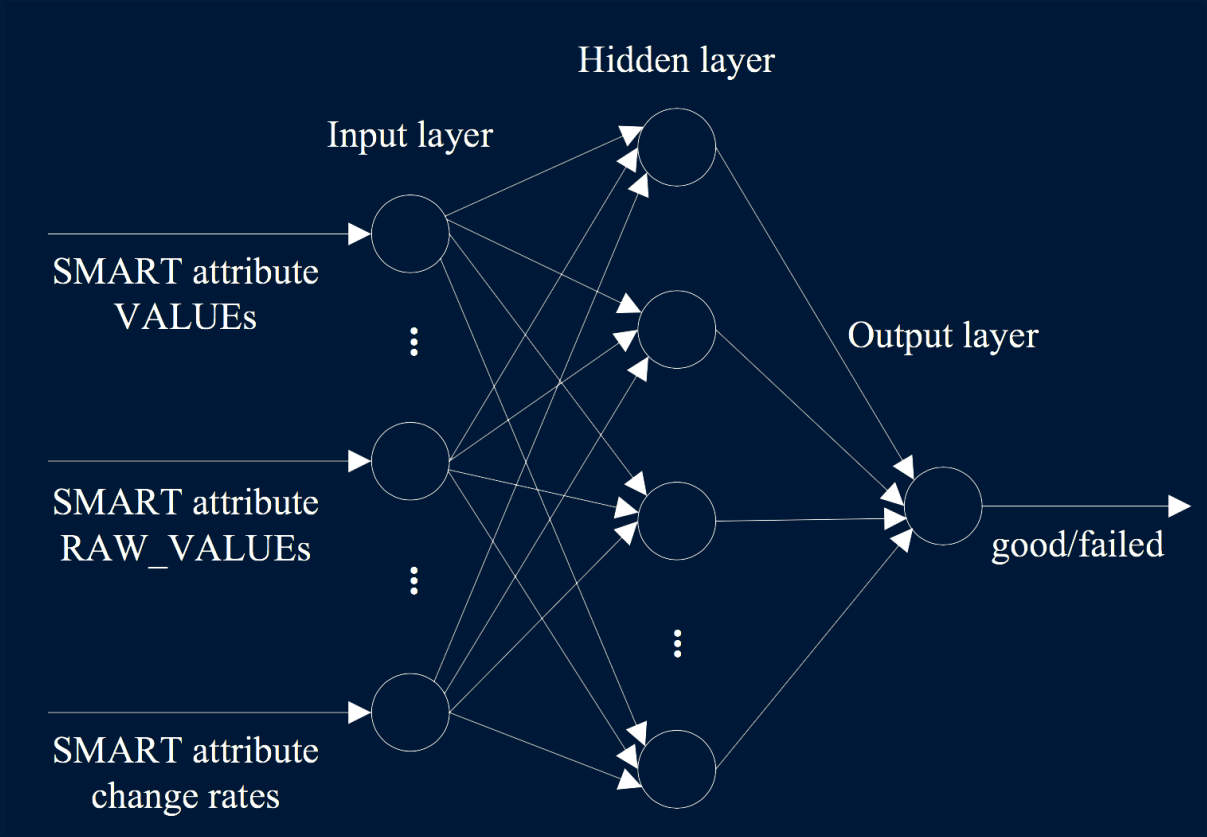
## Failed drive

| Serial no. | Temp1 | FlyHeight1 | Servo8 | ReadError17 | WriteError | ··· |
|---|---|---|---|---|---|---|
| 100001 | 10 | 7962 | 0 | 0 | 57005 | ··· |
| 100001 | 12 | 7972 | 0 | 0 | 57005 | ··· |
| 100001 | 11 | 7949 | 0 | 8 | 57005 | ··· |
| 100001 | 9 | 7955 | 0 | 1280008 | 57005 | ··· |
| 100001 | 8 | 7955 | 0 | 1280544 | 57075 | ··· |
| 100001 | 15 | 7952 | 0 | 1280548 | 57098 | ··· |
| 100001 | 23 | 7972 | 0 | 1280548 | 57227 | ··· |

$C_1$

$C_2$

| ID # | Attribute Name |
|---|---|
| 1 | Raw Read Error Rate |
| 3 | Spin Up Time |
| 5 | Reallocated Sectors Count |
| 7 | Seek Error Rate |
| 9 | Power On Hours |
| 187 | Reported Uncorrectable Errors |
| 189 | High Fly Writes |
| 194 | Temperature Celsius |
| 195 | Hardware ECC Recovered |
| 197 | Current Pending Sector Count |


Hidden layer
Input layer
SMART attribute VALUEs
SMART attribute RAW_VALUEs
SMART attribute change rates
Output layer
good/failed

# Introduction

- Storage devices have long been the subject of many performance optimizations such as prefetching, request scheduling, caching, layout optimizations, and hybrid storage.
- Evaluating such design refinements requires accurate experimentation, which is often very time-consuming.
- Software-based simulators do not suffice. As storage devices are opaque and contain many sophisticated algorithms.
- Previous studies have resorted to run traces as fast as possible [50], speed up inter-arrival time [51, 39], manually select portions of the trace [61, 14, 57], and manual selection along with speed-ups [60]. These methods either lack reproducibility or do not preserve key parameters of trace behavior.

Representative Sampling of IO Traces

**DiskAccel: Accelerating Disk-Based Experiments by Representative Sampling.** (by Mojtaba Tarihi et al.)

# Introduction (cont.)

- We present a methodology called DiskAccel to efficiently select key intervals of a trace as representatives and to replay them to estimate the response time of the whole workload.
  - Uses a variety of properties to select a representative subset of the workload.
  - Does not use performance numbers obtained from other hardware devices (hardware independent)
  - Focuses on real hardware

➢ Divides each trace into multiple intervals, extracts a variety of features from each interval, and then performs clustering.

- Also developed a tool to replay representative intervals, avoiding pitfalls:
  a) Warm-up of hardware cache by replaying an appropriate number of preceding requests
  b) Enforcing inter-request dependency by constructing dependency trees
  c) Accurate control of request arrival time by avoiding scheduling delays

# Introduction (cont.)

- To validate DiskAccel, we use the Microsoft Research Cambridge (MSRC) workloads.

- After clustering is performed, validation in two steps:
  1. Use the performance numbers embedded in the traces
  2. Perform full runs on some of the selected traces

- Finally, we perform partial runs on our hardware testbed with an average speed-up of 577x versus full one-week runs and compare the weighted response time against the average response time of the numbers yielded from full one-week runs. This yields an average and maximum relative errors of 7.6% and 17.2%, respectively.

- Benchmark tools such as Postmark, Bonnie++, and Iozone, are publicly available and have been widely used
- However, I/O benchmarks have difficulties in replicating burstiness and auto-correlation, despite these behaviors being exhibited widely by I/O workloads

- While real-world traces contain key spatio-temporal properties, their long duration must be addressed appropriately.

- Two studies are well-known on the subject of sampling CPU traces:
  - SMARTS which performs systematic sampling of intervals for detailed simulation
  - SimPoint which uses per-interval feature extraction and clustering to pick representative intervals

- 36 MSRC traces
- Each trace is approximately a week long and records requests to a single volume.
- Challenging for replay on real hardware

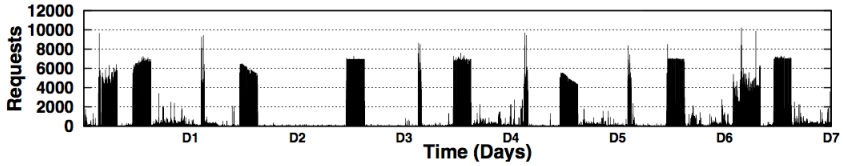



Figure 1: Average per-minute requests of usr_1 trace from [28] (Note the diurnal patterns).

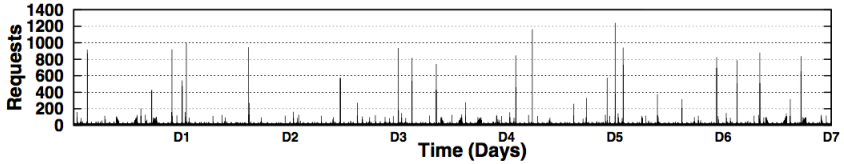



Figure 2: Average per-minute requests of rsrch_0 trace from [28] (Non-diurnal, very bursty, and having no well-defined patterns).

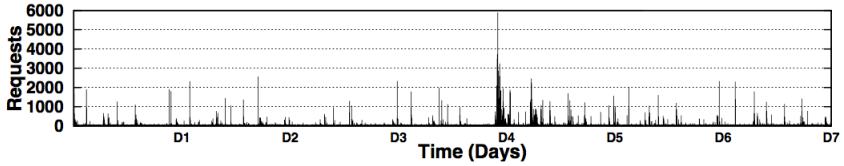



Figure 3: Average per-minute requests of hm_0 trace from [28] (Bursty, activity peaks on a certain day).

- Developed a tool to
  a) Perform whole one-week runs of a selected number of traces to serve as new references
  b) Perform partial runs including only the representative intervals and compare their WRT against the newly obtained reference numbers

- Steps to prevent interference:
  a) No filesystem
  b) O_DIRECT ag to prevent kernel caching
  c) RT_PREEMPT real-time capability patch
  d) Linux's Native Asynchronous I/O which allows getting the exact request finish times while being able to have multiple requests in flight.
  - Switching to busy-waiting [which reduced average drift time by hundreds of microseconds even when the real-time patch was active]

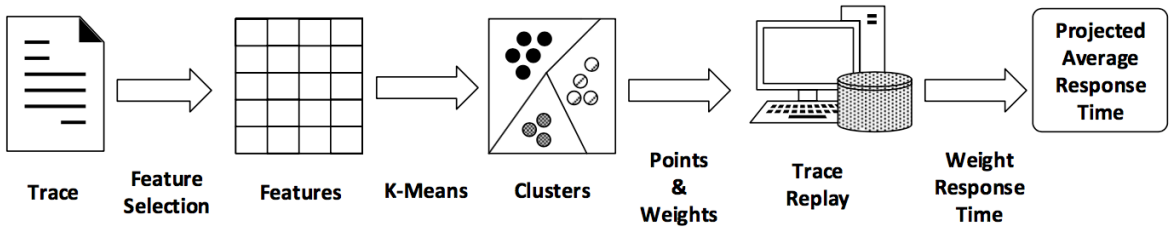



Figure 4: DiskAccel workflow.

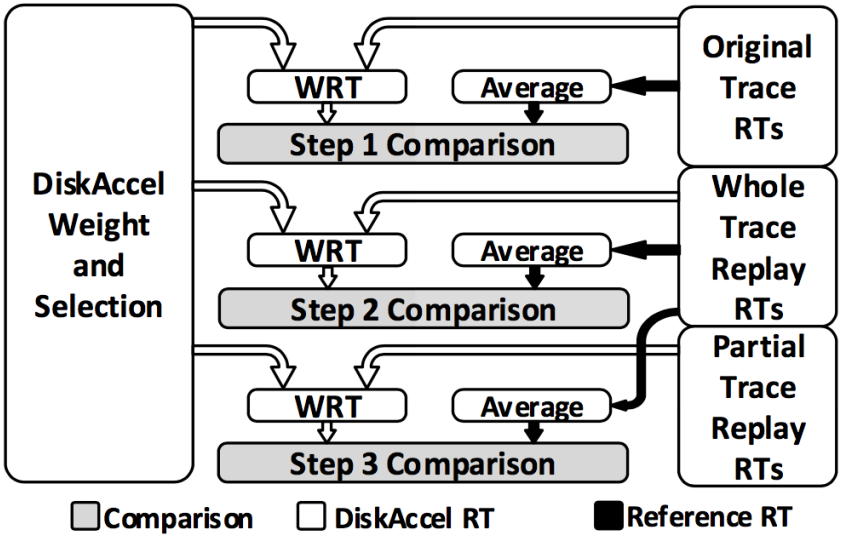
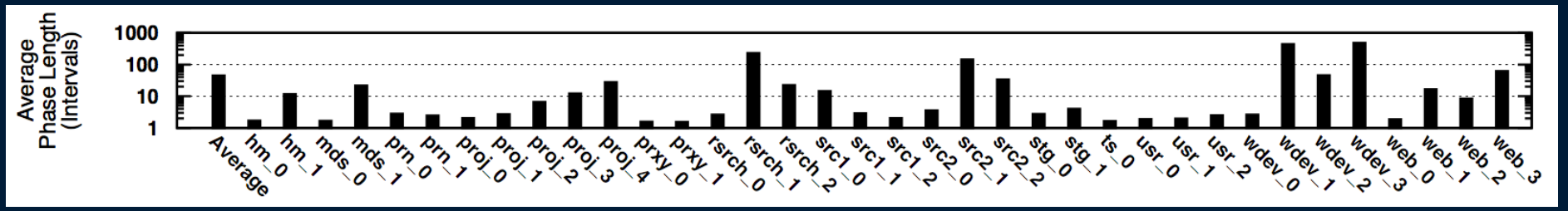



Figure 5: Evaluation steps (RT: Response Time, WRT: Weighted Response Time).

# TRACE SAMPLING

- The methodology first performs clustering to choose the representative intervals and then replays them while performing warm-up and enforcing request dependency.

- CPUs have much higher activity levels, meaning an I/O trace of the same activity is going to have much lower number of requests.
- I/O traces are very bursty and the response time fluctuates wildly between request intervals, while in CPU traces Cycles per Instruction (CPI) remains constant for many different intervals
- Burstiness and auto-correlation of storage workloads means that requests cannot be considered independently and any sampling must capture the temporal context of requests.
  - Group requests into 10-second intervals
  - We define a phase as a set of successive intervals where performance fluctuates by less than 10%.



- The effectiveness of using systematic sampling at representing the performance of the whole trace is evaluated.
- **SMARTS** determines the number of sampled intervals which must run in a detailed manner by using confidence intervals. It then uses systematic sampling with a fixed period to select these intervals and run them in a detailed manner.
- The paper concludes that this strategy is **unsuitable** for reducing storage traces.

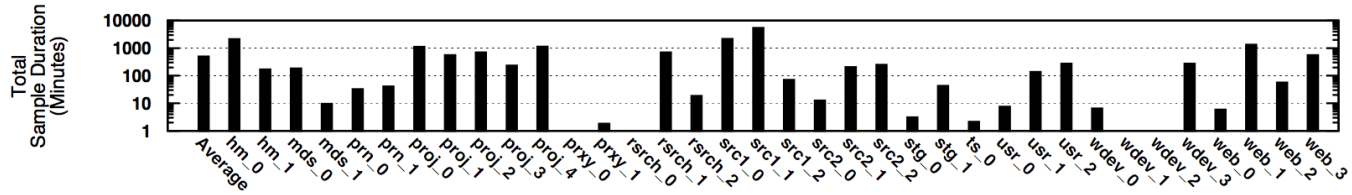



Figure 7: Total duration of intervals selected by confidence-based sampling with a maximum error of 15% and a confidence of 90%. The full duration of the traces is one week (~10000 minutes).

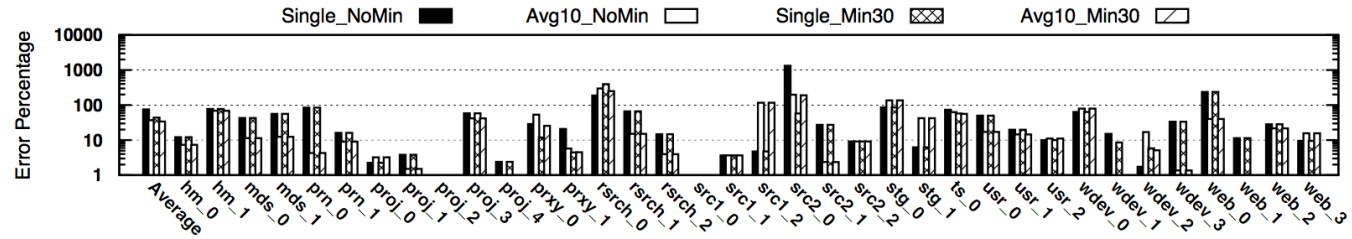



Figure 8: Error percentage of using a SMARTS-like [58] systematic sampling policy on 10-second intervals. AvgN averages the N runs with the highest total request count (Single = Avg1). MinK places a minimum bound of K on the sample count (NoMin = Min1).

| Feature Name | Short Form | Description | Related Property |
|---|---|---|---|
| Random Ratio | *Rnd* | Percentage of Queue-Based Random Requests (See [2]) | Temporal and Spatial Locality |
| Read Ratio | *Rd* | Percentage of Read Requests | Read Frequency |
| Total Traffic | *Mss* | Sum of Request Sizes | Device Load |
| Total Request Count | *Cnt* | Total Number of Requests | Device Load |
| Address Entropy | *Ant* | Multi-level Address Shannon Entropy | Temporal and Spatial Locality |
| Arrival Time Entropy | *Ent* | Multi-level Arrival Time Shannon Entropy | Burstiness |
| Working Set | *Wst* | Sum of Unique Locations Accessed | Device Load, Temporal Locality |
| Working Set Locality | *Wsl* | Ratio of Total Traffic to Working Set | Temporal Locality |
| Average Request Size | *Arq* | Ratio of Total Traffic to Request Count | Device Load |
| Travel Distance | *Tre* | Queue-Based Request Offset Travel (See [2]) | Spatial Locality, Seek Count |
| Average Travel Distance | *Ate* | Average Queue-Based Request Offset Travel (See [2]) | Spatial Locality, Seek Count |

- Aims:
  a) group trace intervals into similar clusters and
  b) choose a suitable representative for each cluster.
- The process of clustering and selecting representative intervals must take interval density into account, as dense request intervals have much more impact on performance

- Weighted variation of K-Means is used. Data is normalized beforehand.
- Value of K has a great impact on RT estimation error.
- BIC and $F_k$ were implemented to find the value of K. [K ranging from 2 to 50]
- Unified feature vector used across all traces: ArqWslRndAntEntTreAte

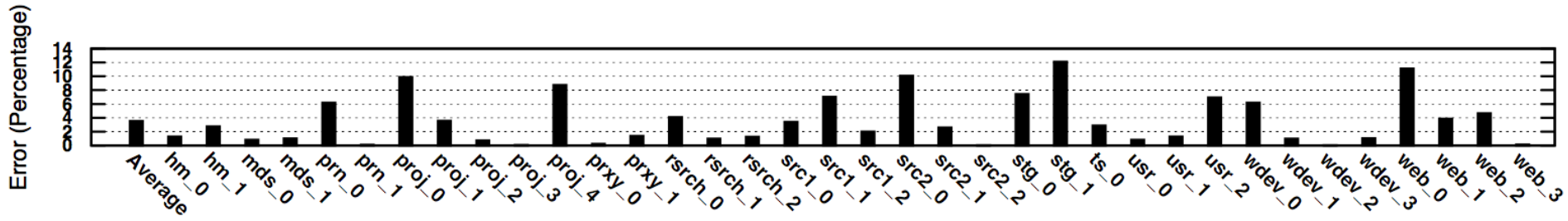



**Figure 10: Per-trace WRT error of our chosen feature vector *ArqWslRndAntEntTreAte* with original response times (Step 1 in Figure 5).**

- Ignoring inter-request dependency is referred to as open-loop replaying which causes significant error, making closed-loop replays essential.
- The replay tool replays traces in a closed-loop manner by constructing I/O request dependency trees.
- While it is possible to replay every request as blocking, it is not believed to represent real system behavior.
- Reads are considered to be blocking and writes to be non-blocking.

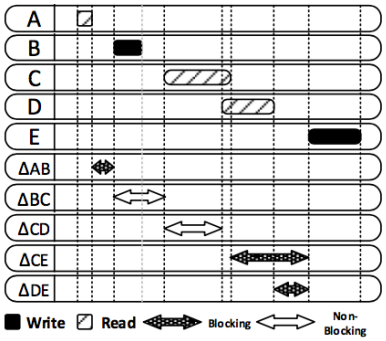



Figure 12: Inferring dependencies by treating reads as blocking and writes as non-blocking.

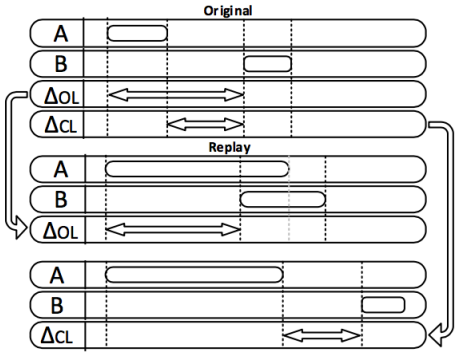



Figure 11: Comparison of open-loop (Middle) and closed-loop (Bottom) replays of a recorded workload (Top). Originally (Top), B depends on A and its start is $\Delta_{OL}$ and $\Delta_{CL}$ units of time after the start and finish time of A, respectively. On the destination, A takes longer than original and unlike the open-loop replay (Middle), close-loop (Bottom) replay avoids potential conflicts.

- Note that due to operations such as read-ahead, much fewer requests must be selected to warm up the cache but we choose a pessimistic policy to ensure the accuracy of warm-up.

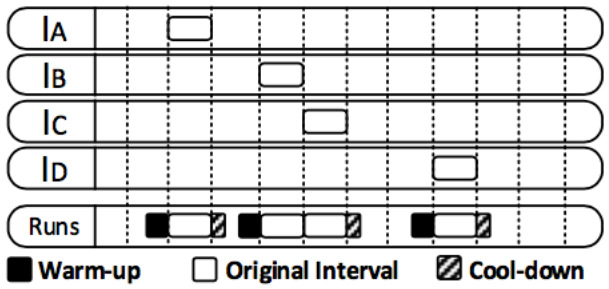



**Figure 13: Runs generated (including warm-up and cool-down) for the representative intervals $I_A$...$I_D$. Since $I_B$ and $I_C$ are continuous, a singular run can perform measurements for both of them.**

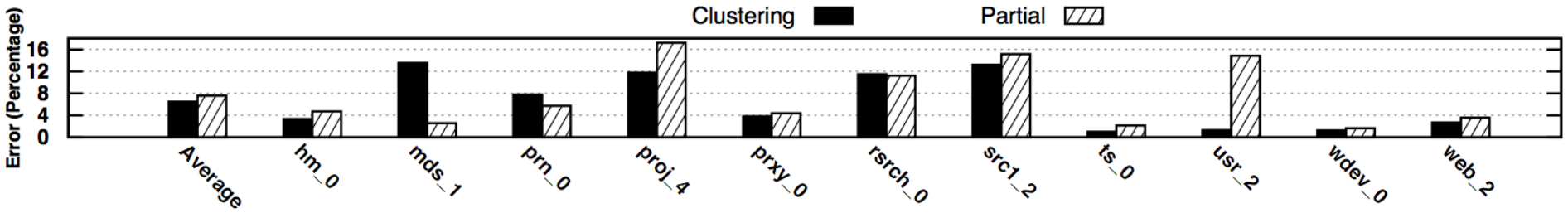



**Figure 14: Per-trace WRT error of our chosen feature vector *ArqWslRndAntEntTreAte*, compared against the reference. The "clustering" statistic is the WRT error by clustering whole trace runs (Step 2 in Figure 5) and "partial" refers to the WRT error of partial trace runs (Step 3 in Figure 5). The reference for both cases is the average response time of a whole trace replay.**

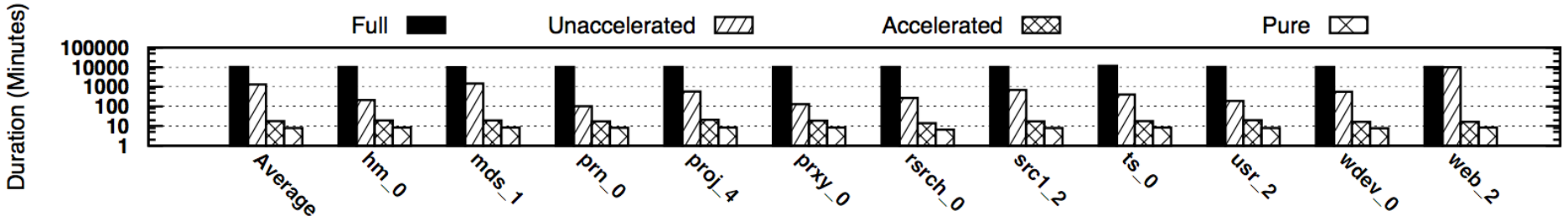



**Figure 15: Run time durations of different modes, Whole trace (Full), Partial runs without accelerated warm-up (Unaccelerated), Partial runs with accelerated warm-up (Accelerated), and without warm-up (Pure).**

Thanks for your attention :)

Any Questions?!

**Contact me at nima.mohammadi@ut.ac.ir**