SelfTune:
Tuning Cluster Managers

Microsoft Research + Microsoft [Office 365, Azure]

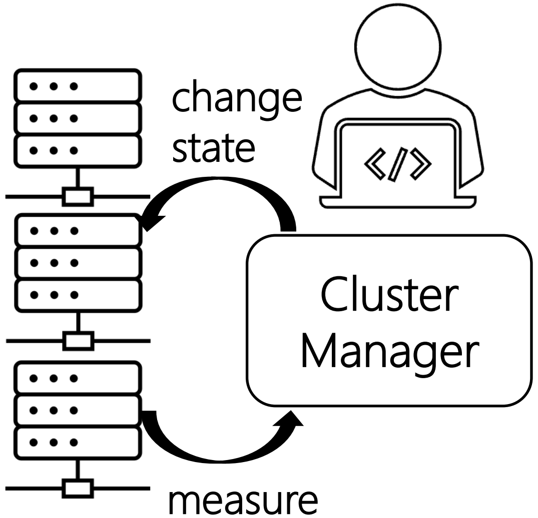
NSDI

April 2023

Cluster management frameworks

Kubernetes

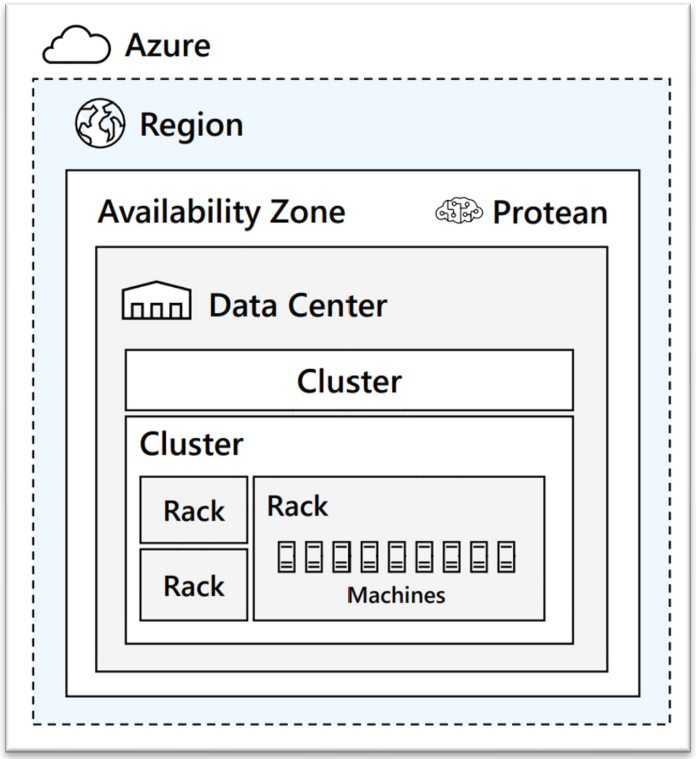
Azure **K**ubernetes **S**ervice



Borg

Twine

Azure Protean

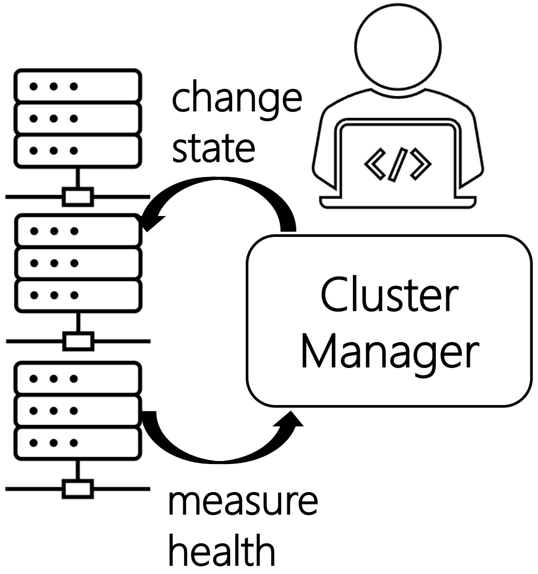


container allocation, scheduling, VM placement

EXO Workload Manager

Schedules background jobs on Substrate machines

Adjust concurrency limit of disk (#background tasks that can simultaneously access disk)



CPU utilization, disk latency, network loss, ...

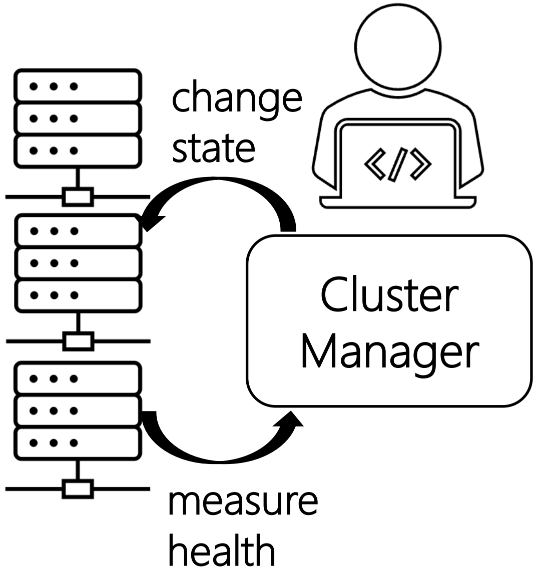
Thousands of lines of code implementing the scheduler heuristics

Hundreds of configuration parameters tweaked by experts.

(*for different forests, machine types, resource types, job types, etc.*)

Kubernetes

deploy new container,
resize container memory



CPU utilization,
memory utilization,
pod evictions
...

Thousands of lines of code implementing the control plane heuristics

Hundreds of configuration parameters

https://github.com/kubernetes/autoscaler/blob/master/vertical-pod-autoscaler/pkg/recommender/main.go

```go
var (
	metricsFetcherInterval = flag.Duration("recommender-interval", 1*time.Minute, `How often metrics should be fetched`)
	checkpointsGCInterval  = flag.Duration("checkpoints-gc-interval", 10*time.Minute, `How often orphaned checkpoints should be garbage collected`)
	prometheusAddress      = flag.String("prometheus-address", "", `Where to reach for Prometheus metrics`)
	prometheusJobName      = flag.String("prometheus-cadvisor-job-name", "kubernetes-cadvisor", `Name of the prometheus job name which scrapes the cAdvisor metrics`)
	address                = flag.String("address", ":8942", "The address to expose Prometheus metrics.")
	kubeconfig             = flag.String("kubeconfig", "", "Path to a kubeconfig. Only required if out-of-cluster.")
	kubeApiQps             = flag.Float64("kube-api-qps", 5.0, `QPS limit when making requests to Kubernetes apiserver`)
	kubeApiBurst           = flag.Float64("kube-api-burst", 10.0, `QPS burst limit when making requests to Kubernetes apiserver`)

	storage = flag.String("storage", "", `Specifies storage mode. Supported values: prometheus, checkpoint (default)`)
	// prometheus history provider configs
	historyLength       = flag.String("history-length", "8d", `How much time back prometheus have to be queried to get historical metrics`)
	historyResolution   = flag.String("history-resolution", "1h", `Resolution at which Prometheus is queried for historical metrics`)
	queryTimeout        = flag.String("prometheus-query-timeout", "5m", `How long to wait before killing long queries`)
	podLabelPrefix      = flag.String("pod-label-prefix", "pod_label_", `Which prefix to look for pod labels in metrics`)
	podLabelsMetricName = flag.String("metric-for-pod-labels", "up{job=\"kubernetes-pods\"}", `Which metric to look for pod labels in metrics`)
	podNamespaceLabel   = flag.String("pod-namespace-label", "kubernetes_namespace", `Label name to look for pod namespaces`)
	podNameLabel        = flag.String("pod-name-label", "kubernetes_pod_name", `Label name to look for pod names`)
	ctrNamespaceLabel   = flag.String("container-namespace-label", "namespace", `Label name to look for container namespaces`)
	ctrPodNameLabel     = flag.String("container-pod-name-label", "pod_name", `Label name to look for container pod names`)
	ctrNameLabel        = flag.String("container-name-label", "name", `Label name to look for container names`)
	vpaObjectNamespace  = flag.String("vpa-object-namespace", apiv1.NamespaceAll, "Namespace to search for VPA objects and pod stats. Empty means all namespaces will be used.")
)

// Aggregation configuration flags
var (
	memoryAggregationInterval      = flag.Duration("memory-aggregation-interval", model.DefaultMemoryAggregationInterval, `The length of a single interval, for which the peak mem
	memoryAggregationIntervalCount = flag.Int64("memory-aggregation-interval-count", model.DefaultMemoryAggregationIntervalCount, `The number of consecutive memory-aggregation-in
	memoryHistogramDecayHalfLife   = flag.Duration("memory-histogram-decay-half-life", model.DefaultMemoryHistogramDecayHalfLife, `The amount of time it takes a historical memory
	cpuHistogramDecayHalfLife      = flag.Duration("cpu-histogram-decay-half-life", model.DefaultCPUHistogramDecayHalfLife, `The amount of time it takes a historical CPU usage sa
```

Configuring cluster managers

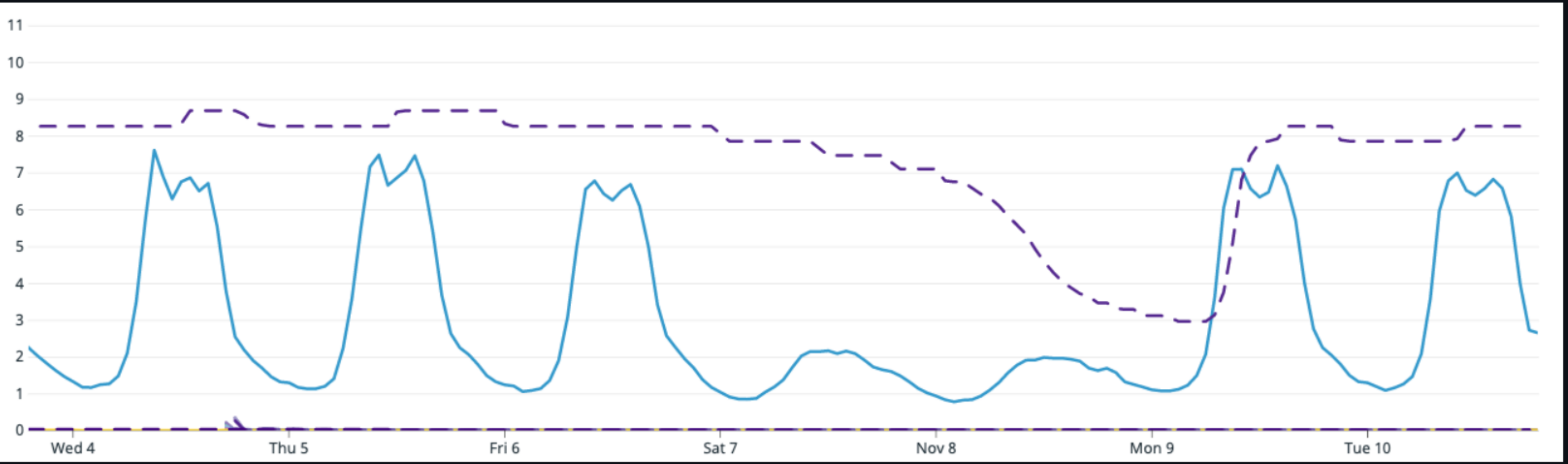
- Domain expertise, (limited) empirical evaluations; can be sub-optimal
- Global, static configuration of cluster managers; can be sub-optimal
    - What works for one cluster need not work for another
    - What works for one cluster *today* may not work *next week*, as workload patterns drift or hardware is replaced

https://github.com/kubernetes/autoscaler/issues/3684

UseCase:

I have an app with low CPU usages over the weekends and the VPA CPU recommendation dips respectively. However, is it possible to configure the recommender in a way to not be influenced by the two day dip? I'd like to keep the recommendation the same as during regular usage days.

Here is a chart of my CPU usage / VPA CPU recommendation:
CPU usage - solid line
VPA CPU recommendation - dotted line



The reason I don't want the recommended CPU limit to dip is a risk factor because of how much business value the app brings and I'm willing to "waste" the CPUs over the weekend to prepare for anything.

Expected:

VPA CPU limit recommendation to stay consistent to the majority of the usages and not dip due to the two days of low usage on weekends.

Configuring cluster managers

- Domain expertise, (limited) empirical evaluations; can be sub-optimal
- Global, static configuration of cluster managers; can be sub-optimal
  - What works for one cluster need not work for another
  - What works for one cluster *today* may not work *next week*, as workload patterns drift or hardware is replaced

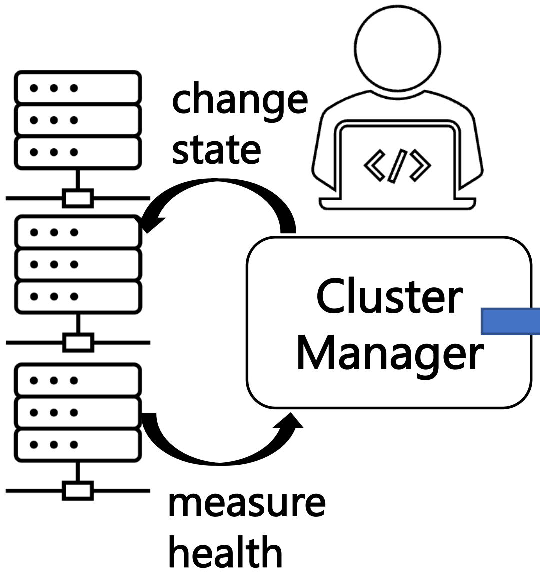
- Options for developers/infrastructure team:
  - Create and manage multiple configuration files for different environments
  - Use hyperparameter search/sampling/simulation mechanisms [CherryPick '17, BestConfig '17, Metis '18, MLOS '20, …]

None of these options is satisfactory

SelfTune: A framework to seamlessly tune

Kubernetes

deploy new container,
resize container memory



CPU utilization,
memory utilization,
pod evictions

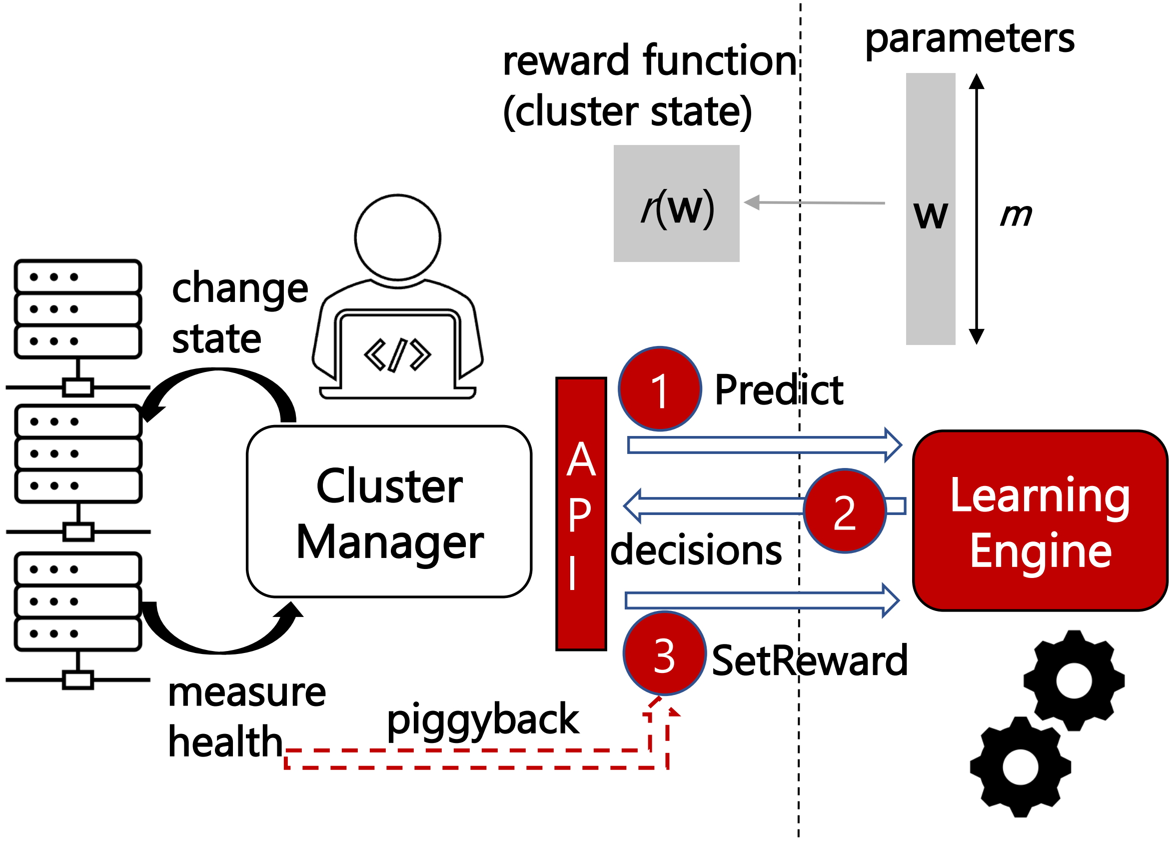
1. params = selfTune.Predict()

// re-configure Kubernetes with params

2. selfTune.SetReward (numPodEvictionsDaily)

Developer specifies *just* what is needed to drive the system towards desired states.

SelfTune Framework: API + Learner



Cluster management is classic RL setting: observe health, deploy action, and repeat.
We can piggyback on this set up to tune the underlying parameters.

Key Features

- (Almost) zero engineering overhead
  - integration, computational

- "Implicit context" – developers choose the granularity at which parameters need to be tuned
  - *Machine-level? Cluster-level?*
  - *Application-level? Workload-level?*

- Minimal assumptions

State-of-the-art RL systems for parameter tuning
[Decima '19, CDBTune '19, MS Decision Service '16]
feature engineering, complexity, categorical parameters

Environments, rewards, optimal configs vary with time

- Online tuning of parameters
- Sample complexity (# rounds it takes for the algorithm to catch up with an oracle) needs to be small

Bayesian Optimization common but not ideal in our setting

We use a simple and intuitive state-of-the-art algorithm for "continuous bandits"

Optimal regret algorithm for Pseudo-1d Bandit Convex Optimization.
A Saha, N Natarajan, P Jain, P Netrapalli. ICML 2021.

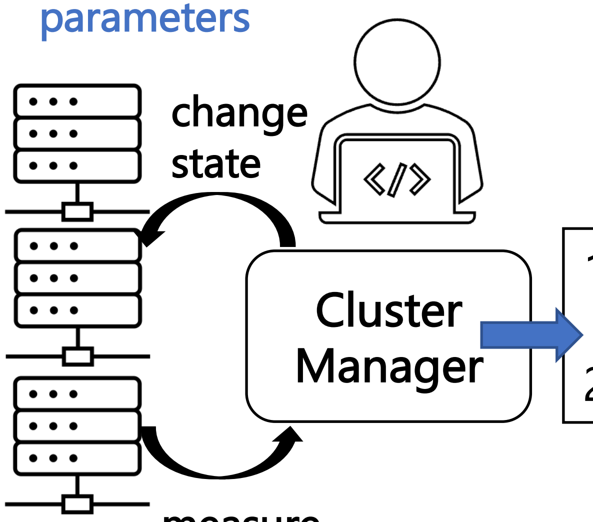
Evaluation: Three cluster management settings

- Kubernetes
  - Focus: optimal latency for containerized applications
  - Results on DeathStar benchmark with a social networking application
- EXO Workload Manager
  - Focus: optimal cluster resource utilization & throughput for workloads
  - Results from months of deployment in LAM, NAM, APAC forests
- Azure Functions (“FaaS”)
  - Focus: optimal latency for the cloud users and save costs for Azure
  - Results on full set of Azure traces (2M applications, >10M daily invocations, 4 months)

Revisiting the Kubernetes example

100 real-valued/discrete parameters

Kubernetes



p95 latency

```
1. params = SelfTune.Predict()

2. SelfTune.SetReward (P95latency)
```

Workload: HTTP requests to service endpoint
Implicit context: Workload-level tuning

Metric	SelfTune	BO	eps-greedy	Default
P95 latency (ms)	**19.5**	19.9	20.0	31.1
# Samples	**8**	41	30	-
P50 iter. cost (ms)	**20.5**	23.3	29.2	-
P75 iter. cost (ms)	**21.1**	33.0	33.2	-
P95 iter. cost (ms)	**28.3**	76541.9	67640.3	-

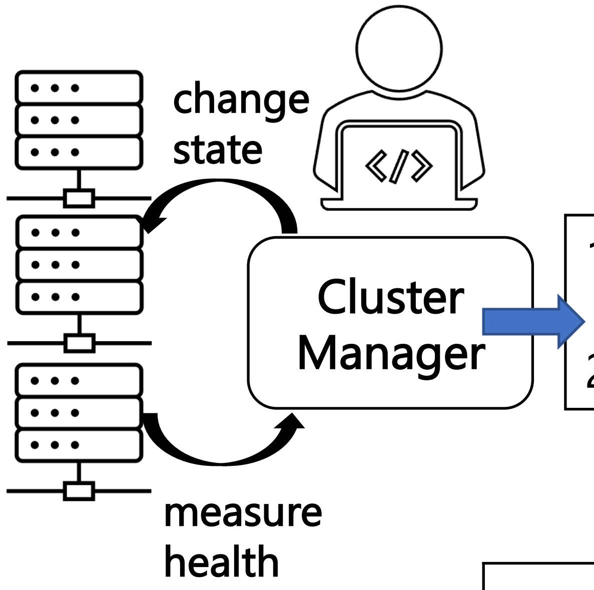
Evaluation: Three cluster management settings

- Kubernetes
  - Focus: optimal latency for containerized applications
  - Results on DeathStar benchmark with a social networking application
- EXO Workload Manager
  - Focus: optimal cluster resource utilization & throughput for workloads
  - Results from months of deployment in LAM, NAM, APAC forests
- Azure Functions (“FaaS”)
  - Focus: optimal latency for the cloud users and save costs for Azure
  - Results on full set of Azure traces (2M applications, >10M daily invocations, 4 months)

EXO Workload Manager

resource concurrency

EXO
Workload Manager



grant ratio
HUP

Workload: Live production scheduling workloads
Implicit context: Machine-level tuning

1. params = SelfTune.Predict()

2. SelfTune.SetReward (GrantRatio)

Cluster	Reward	Metric Improvement					
		Res Utilization			Throughput		
		P25	P50	P75	P25	P50	P75
1	Throughput	SI	SI	SI	214%	178%	169%
2	Throughput	SI	SI	SI	34%	37%	25%
3	Res Utilization	2%	1%	3%	18%	18%	20%

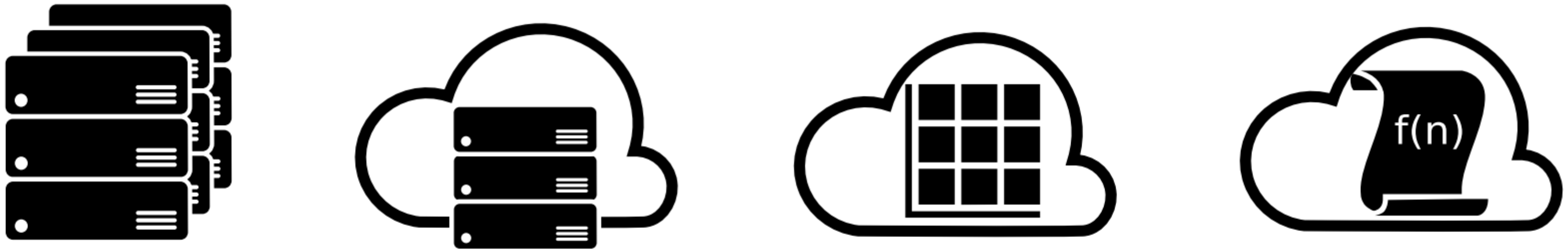
Evaluation: Three cluster management settings

- Kubernetes
  - Focus: optimal latency for containerized applications
  - Results on DeathStar benchmark with a social networking application

- EXO Workload Manager
  - Focus: optimal cluster resource utilization & throughput for workloads
  - Results from months of deployment in LAM, NAM, APAC forests

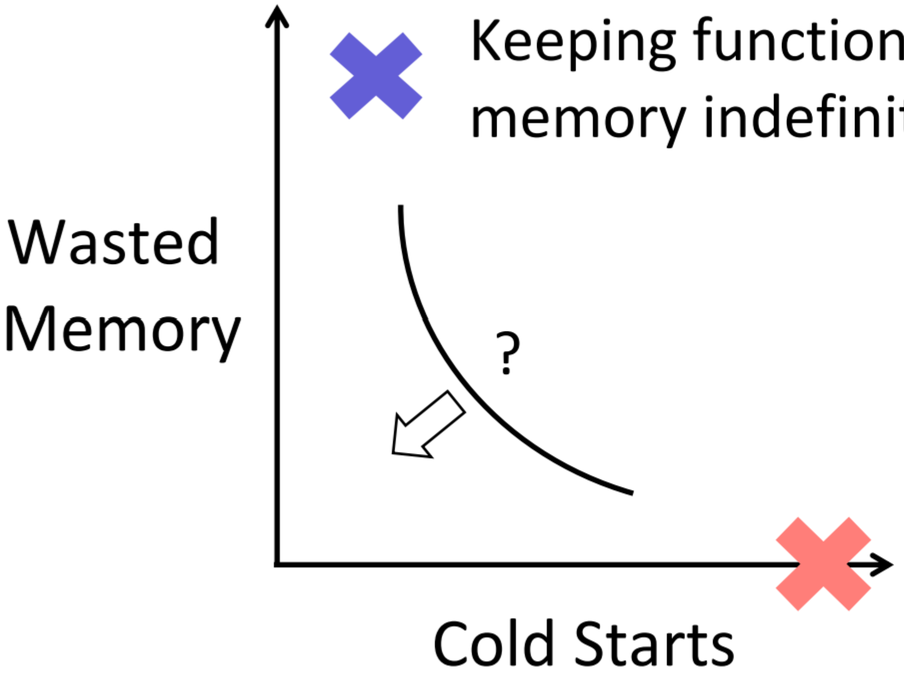
- Azure Functions (“FaaS”)
  - Focus: optimal latency for the cloud users and save costs for Azure
  - Results on full set of Azure traces (2M applications, >10M daily invocations, 4 months)

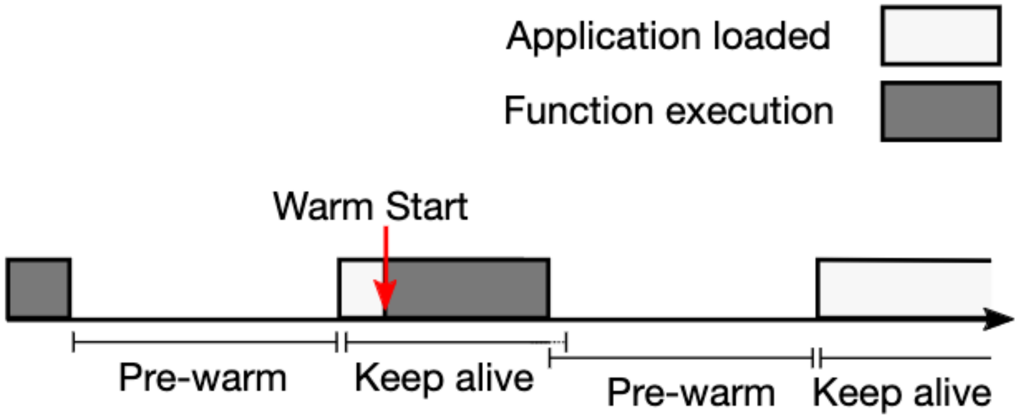
Azure Function-as-a-Service



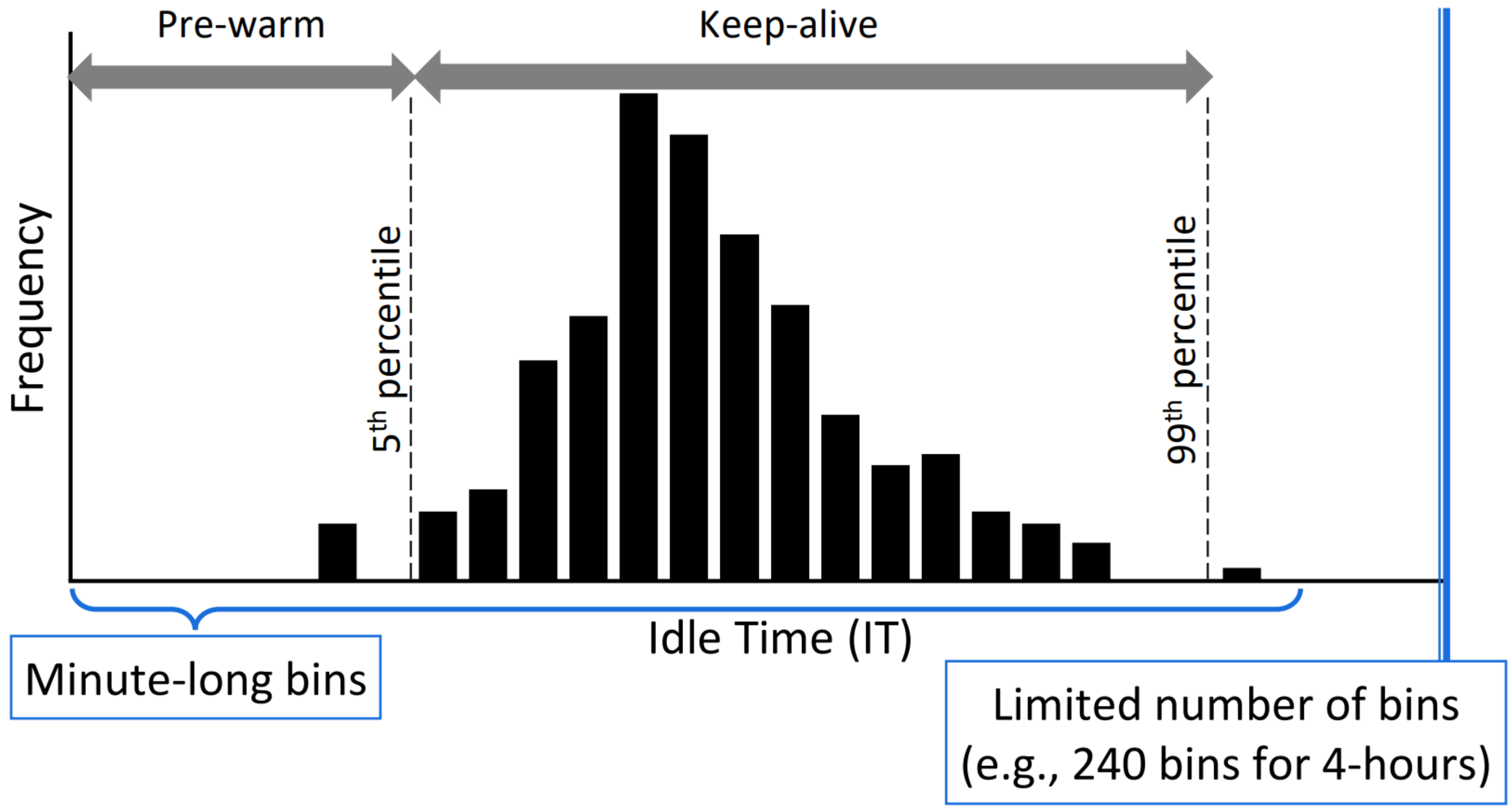
	Bare Metal	VMs (IaaS)	Containers	Functions (FaaS)
Unit of Scale	Server	VM	Application/Pod	Function
Provisioning	Ops	DevOps	DevOps	Cloud Provider
Init Time	Days	~1 min	Few seconds	Few seconds
Scaling	Buy new hardware	Allocate new VMs	1 to many, auto	0 to many, auto
Typical Lifetime	Years	Hours	Minutes	O(100ms)
Payment	Per allocation	Per allocation	Per allocation	Per use
State	Anywhere	Anywhere	Anywhere	Elsewhere

Azure Function-as-a-Service



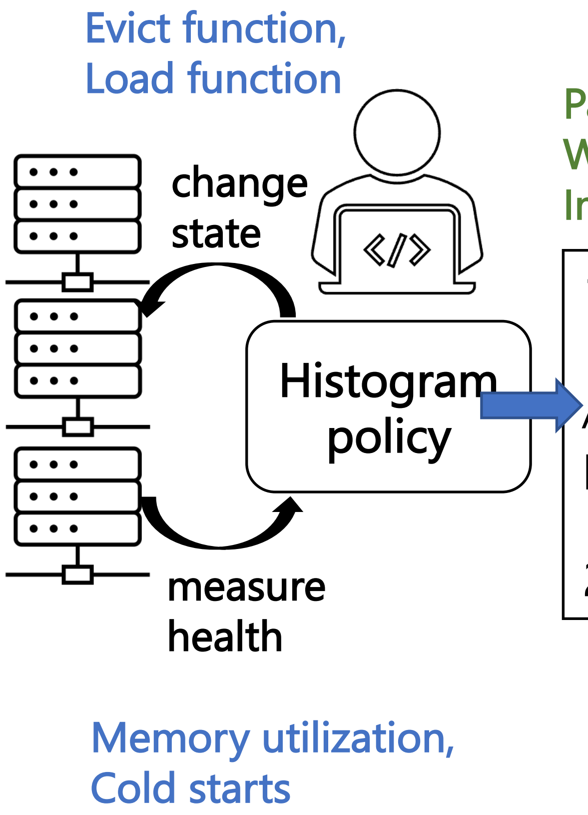


Hybrid Histogram Policy



Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. M Shahrad, R Fonseca, Í Goiri, G Chaudhry, P Batum, J Cooke, E Laureano, C Tresness, M Russinovich, R Bianchini. ATC 2020.

App-specific tuning of policy parameters with SelfTune

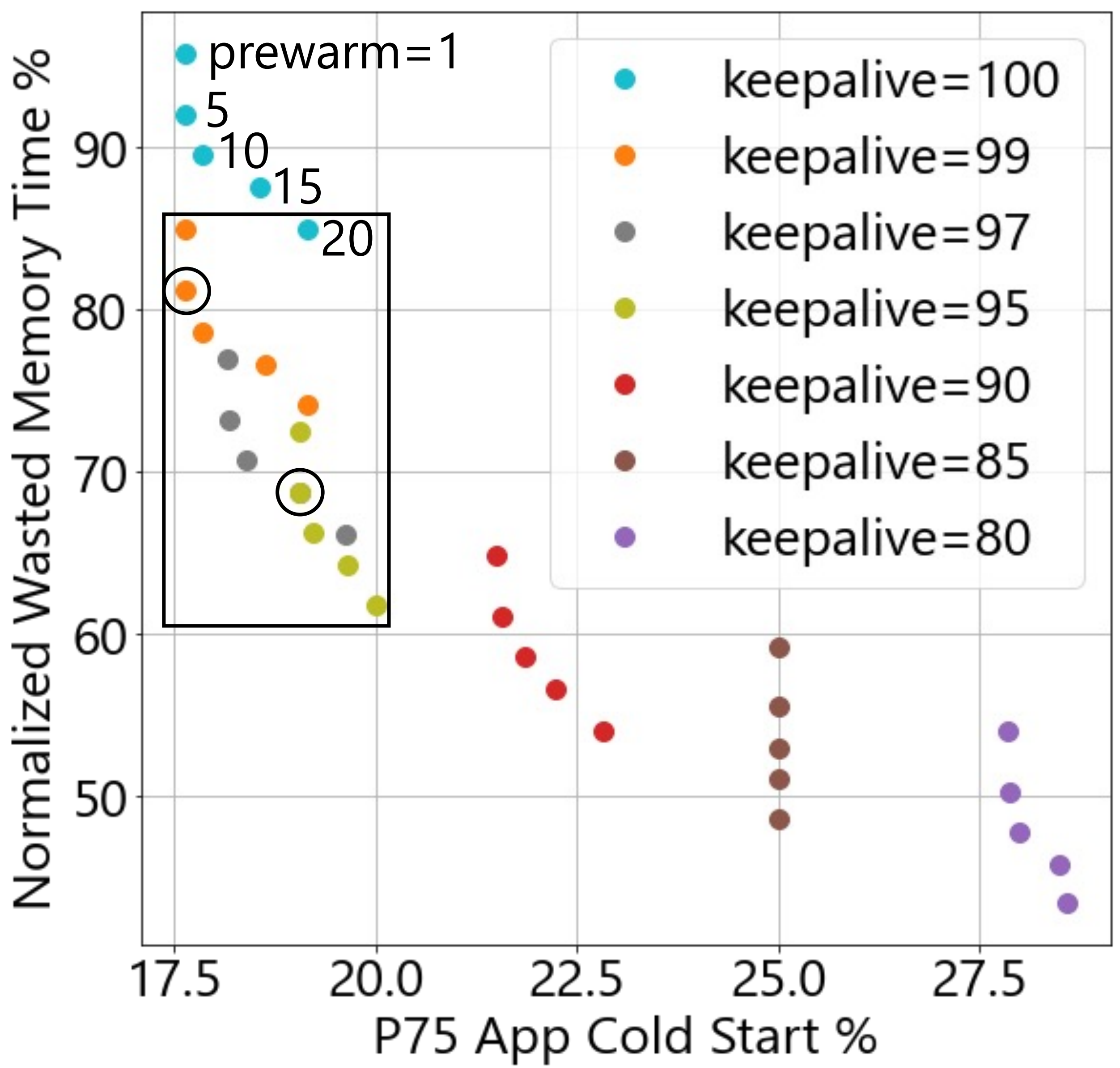


Parameters: Prewarm and Keepalive
Workload: Azure Function traces
Implicit context: Application-level tuning

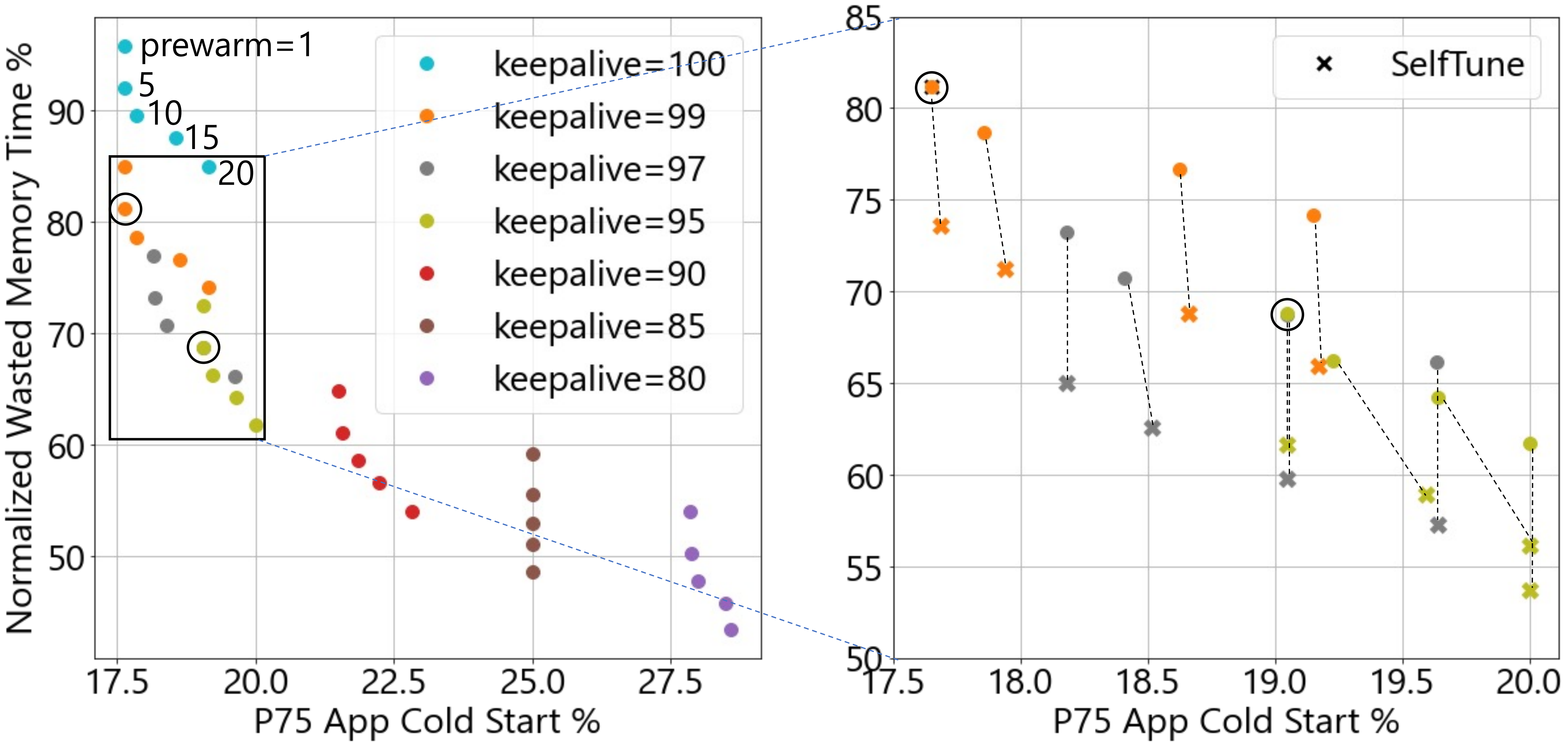
```
1. pw, ka = SelfTune.Predict()

// re-configure hybrid histogram policy with
params

2. SelfTune.SetReward (memoryWastage)
```



prewarm=1
5
10
15
20
keepalive=100
keepalive=99
keepalive=97
keepalive=95
keepalive=90
keepalive=85
keepalive=80
Normalized Wasted Memory Time %
90
80
70
60
50
17.5
20.0
22.5
25.0
27.5
P75 App Cold Start %



About 10%-12% reduction in mem wastage over 3 months of Azure traces, compared to ka=99, pw=5

Summary

- Minimal engineering overhead
- Online tuning with small sample complexity
- Versatile – implicit context
- Success on production workloads