PARAGON: QOS-AWARE SCHEDULING FOR HETEROGENEOUS DATACENTERS

Christina Delimitrou and Christos Kozyrakis

Stanford University

ASPLOS – March 18th 2013

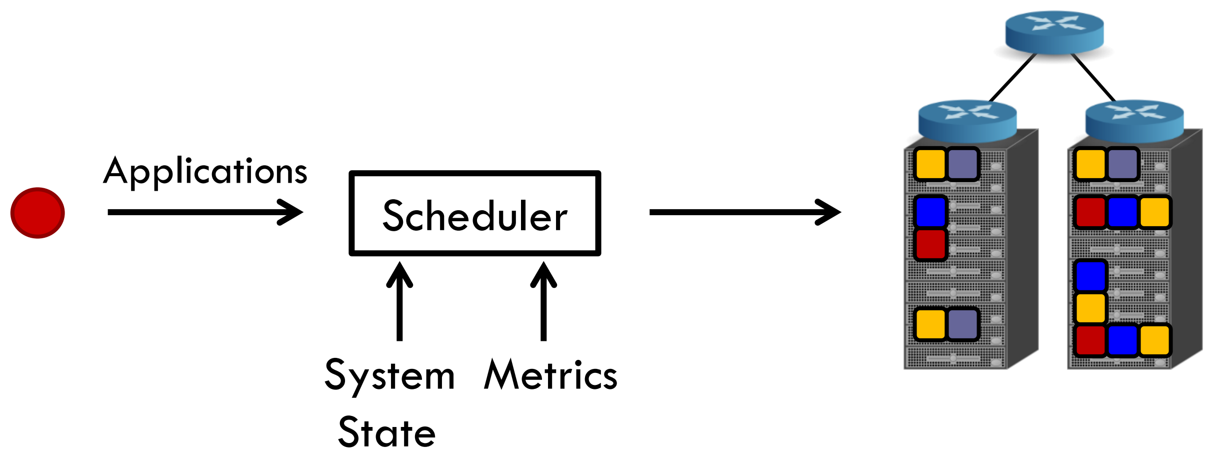
Executive Summary

- Problem: scheduling in cloud environments (e.g., EC2, Azure, etc.)
  - Heterogeneity → losses when running on wrong server
  - Interference → performance loss when interference is high
  - High rates of unknown workloads → no a priori assumptions
- How to get information for a workload?
  - Detailed profiling → intolerable overheads
  - Instead: Leverage info about previously scheduled apps → fast and accurate application classification
- Paragon is a scheduling framework that is:
  - Heterogeneity and interference-aware, app agnostic
  - Scalable & lightweight: scales to 10,000s of apps and servers
  - Results: 5,000 apps on 1,000 servers → 48% utilization increase, 90% of apps < 10% degradation

Outline

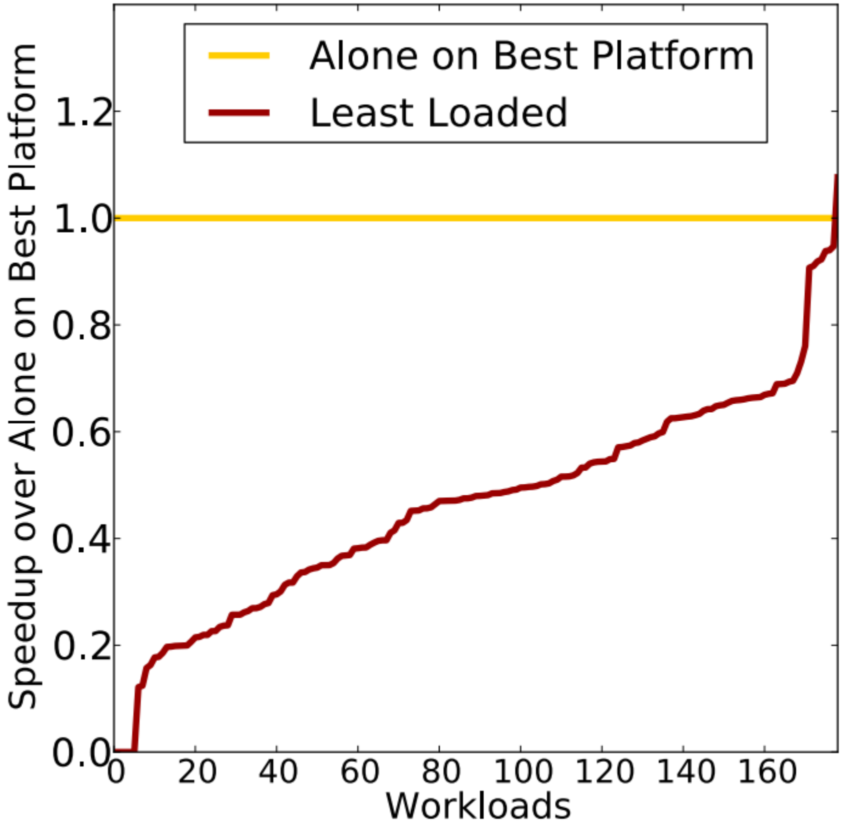
- Motivation
- Application Classification
- Paragon
- Evaluation

Cloud DC Scheduling



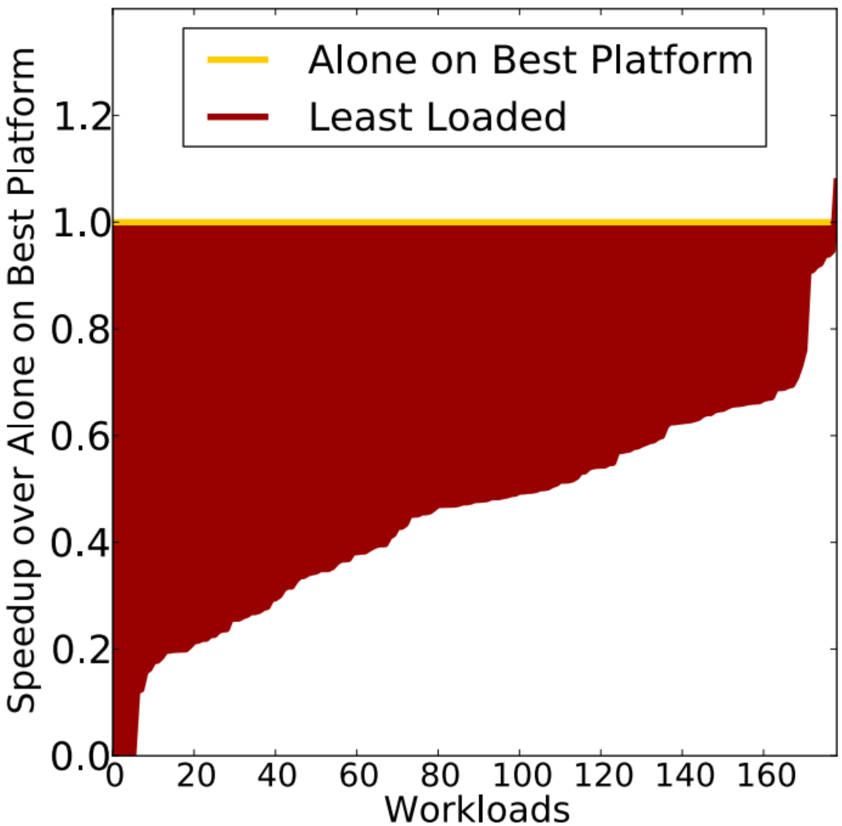
- Workloads are unknown
  - Random apps submitted for short periods, known workloads evolve
- Significant churn (arrivals/departures)
- High variability in workloads characteristics
- Decisions must be performed fast

Common Practice Today



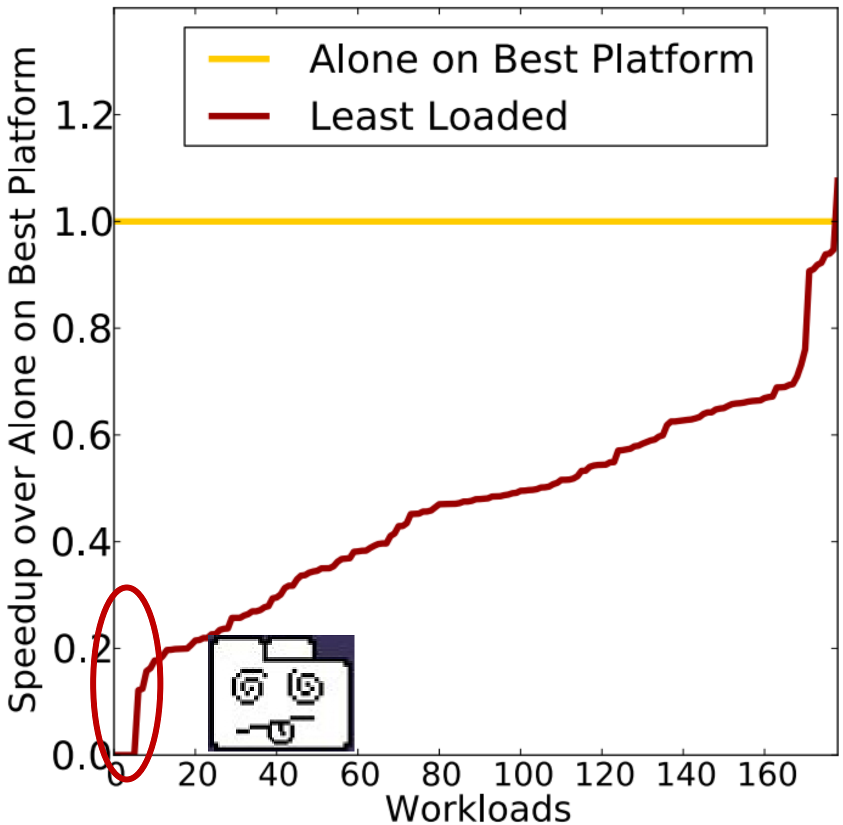
- Least-loaded scheduling
  - Using CPU & memory availability
  - Ignores heterogeneity
  - Ignores interference
- Poor efficiency
  - Over 48% degradation compared to running alone
  - Some apps won't even finish

Common Practice Today



- Least-loaded scheduling
  - Using CPU & memory availability
  - Ignores heterogeneity
  - Ignores interference
- Poor efficiency
  - Over 48% degradation compared to running alone
  - Some apps won't even finish

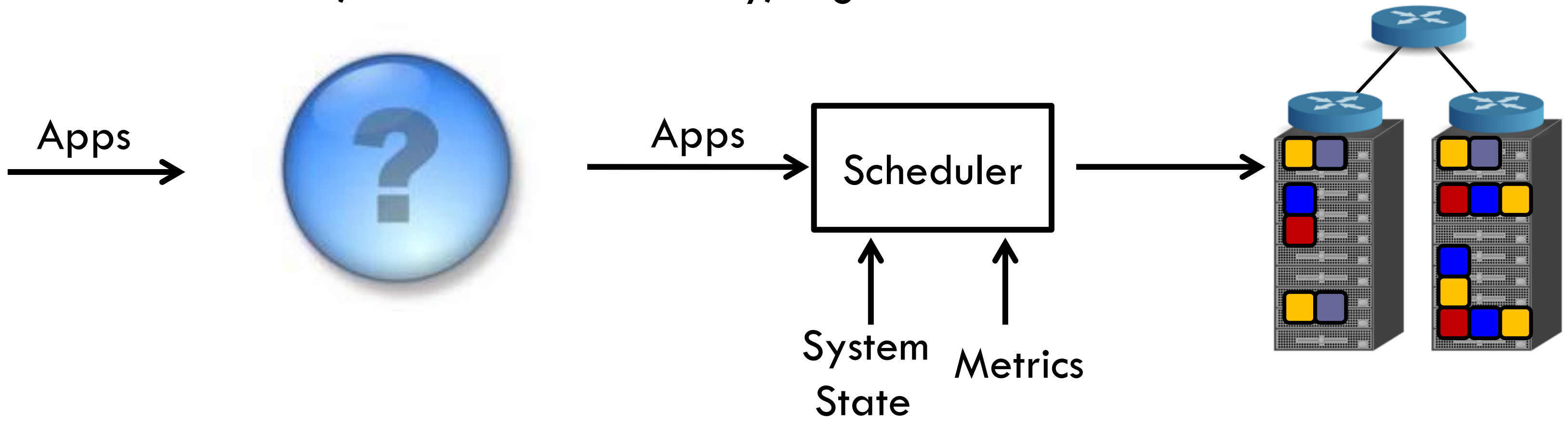
Common Practice Today



- Least-loaded scheduling
  - Using CPU & memory availability
  - Ignores heterogeneity
  - Ignores interference
- Poor efficiency
  - Over 48% degradation compared to running alone
  - Some apps won't even finish

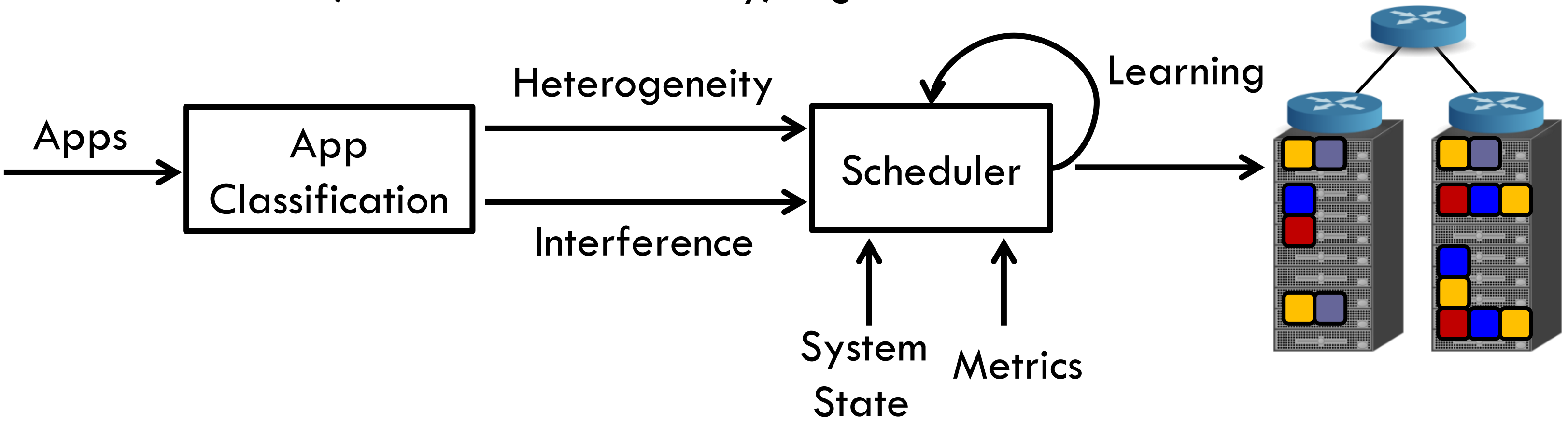
Insight

- Reason for scheduling inefficiency
  - Lack of knowledge of application behavior
  - Heterogeneity & interference characteristics
- Existing approach for app characterization: exhaustive profiling
  - High overheads, does not work with unknown apps
- Our work: Leverage knowledge about previously-scheduled apps
  - Accurate, small data Vs. noisy, big data



Insight

- Reason for scheduling inefficiency
  - Lack of knowledge of application behavior
  - Heterogeneity & interference characteristics
- Existing approach for app characterization: exhaustive profiling
  - High overheads, does not work with unknown apps
- Our work: Leverage knowledge about previously-scheduled apps
  - Accurate, small data Vs. noisy, big data

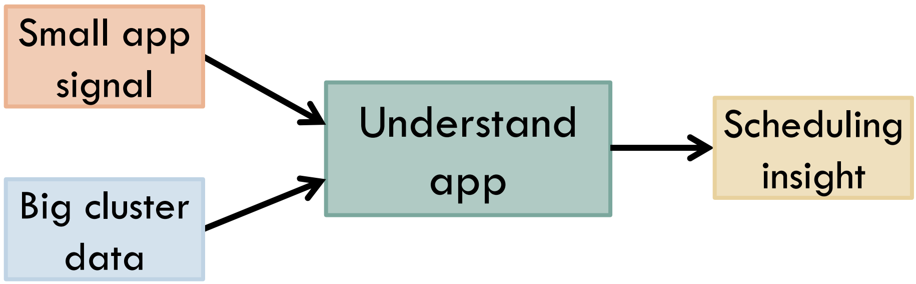


Outline

- Motivation
- Application Classification
- Paragon
- Evaluation

Understanding App Behavior

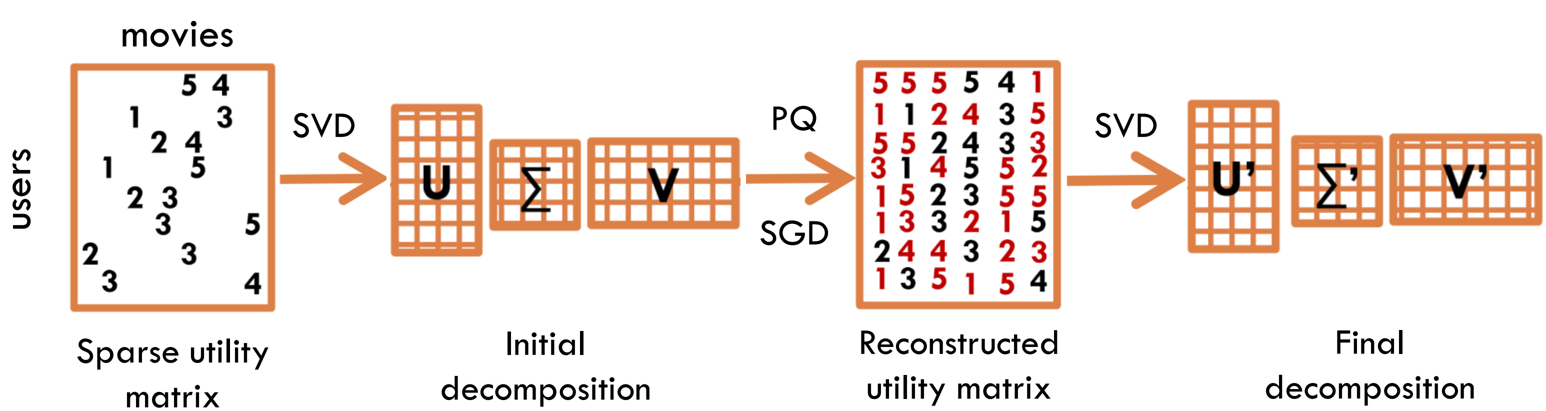
- <u>Goal</u>: quickly extract accurate info on each application to guide scheduling



- Input:
  - Small signal about a new workload
  - Large amount of information about previously-scheduled applications
- Output:
  - Understand app behavior/requirements → recommendations for scheduling
- Looks like a classification problem
  - Similar to systems used in e-commerce, Netflix, etc.

Something familiar…

- Collaborative filtering – similar to Netflix Challenge system
  - Singular Value Decomposition (SVD) + PQ reconstruction (SGD)
    - Leverage the rich information the system already has
- Extract similarities between applications on:
    - Heterogeneous platforms that benefit them
    - Interference they <u>cause</u> and <u>tolerate</u> in shared resources
- Recommendations on ***platforms*** and ***co-scheduled applications***



Classification for Heterogeneity

The Netflix Challenge	**Platform Classification**
Recommend **movies** to **users**	Recommend **platforms** to **apps**
Utility matrix rows → **users** Utility matrix columns → **movies**	Utility matrix rows → **apps** Utility matrix columns → **platforms**
Utility matrix elements → **movie ratings**	Utility matrix elements → **app scores**

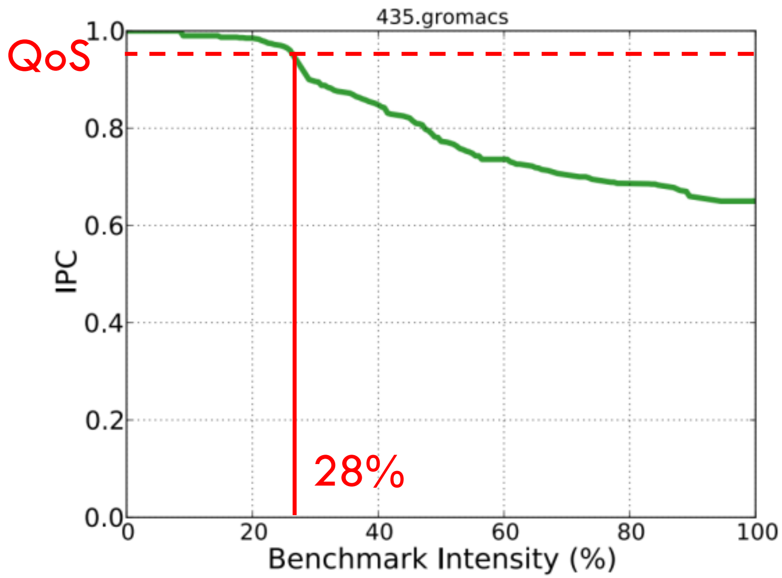
- Offline mode
  - Profile a few apps (20-30) across the different configurations
  - Assign performance scores per run (IPS, QPS, other system metric)
- Online mode
  - For each new app, run briefly on two platforms (1min)
  - Assign performance scores
  - Derive missing entries & identify similarities between apps

Classification for Interference

The Netflix Challenge	**Interference Classification**
Recommend **movies** to **users**	Recommend **minimally interfering co-runners** to **apps**
Utility matrix rows → **users** Utility matrix columns → **movies**	Utility matrix rows → **apps** Utility matrix columns → **microbenchmarks (SoIs)**
Utility matrix elements → **movie ratings**	Utility matrix elements → **sensitivity scores to interference**

- Two types of interference:
  - Interference the application tolerates
  - Interference the application causes
- Identifying sources of interference (SoIs):
  - Cache hierarchy, memory bandwidth/capacity, CPU, network/storage bandwidth

Measuring Interference Sensitivity



- Rank sensitivity of an application to each microbenchmark (0-100%)
- Increase microbenchmark intensity until the application violates its QoS
  → sensitivity to tolerated interference
- Similarly for sensitivity to caused interference

Classification Validation

- Large set of ST, MT, MP and I/O workloads
- 10 Server Configurations (SC)
- 10 Sources of Interference (SoI)

	Metric	Applications (%)			
		ST	**MT**	**MP**	**I/O**
Heterogeneity	Select best SC	86%	86%	83%	89%
	Select SC within 5% of best	91%	90%	89%	92%
Interference	Avg. error across μbenchmarks	5.3%			
	Apps with < 10% error	ST: 81%		MT: 63%	
	SoI with highest error:				
	for ST: L1 i-cache	15.8%			
	for MT: LLC capacity	7.8%			

Classification Overhead

- Time overhead:
  - Training:
    - 2x1min runs for heterogeneity (alone) + 2x1min with two microbenchmarks for interference → in parallel
  - Decision:
    - SVD + PQ reconstruction: $O(\min(n^2m, m^2n)) + O(mn)$
    - Practically: msec for 1,000s apps and servers

- Space overhead:
  - 64B per app and 64B per server

Outline

- Motivation
- Application Classification
- Paragon
- Evaluation

Greedy Server Selection

- Two step process:
  - Select servers with minimal interference
  - Select server with best hardware configuration
- Overview:
  - Start with most critical resource
  - Prune servers that would violate QoS
  - Repeat for all resources
  - Select server with best HW configuration
  - If no candidate left, backtrack and relax QoS requirement
    - Rare, but ensures convergence

Monitor & Adapt

- Sources of inaccuracy:
  - App goes through phases
  - App is misclassified
  - App is mis-scheduled

- Monitor & adapt:
  1. Reactive phase detection: upon performance degradation, reclassify the workload and searches for a more suitable server
  2. Preemptive phase detection: periodically sample a workload subset, reclassify and if heterogeneity/interference profile has changed re-schedule before QoS degrades

- Preview: application scenario with changing workloads in evaluation

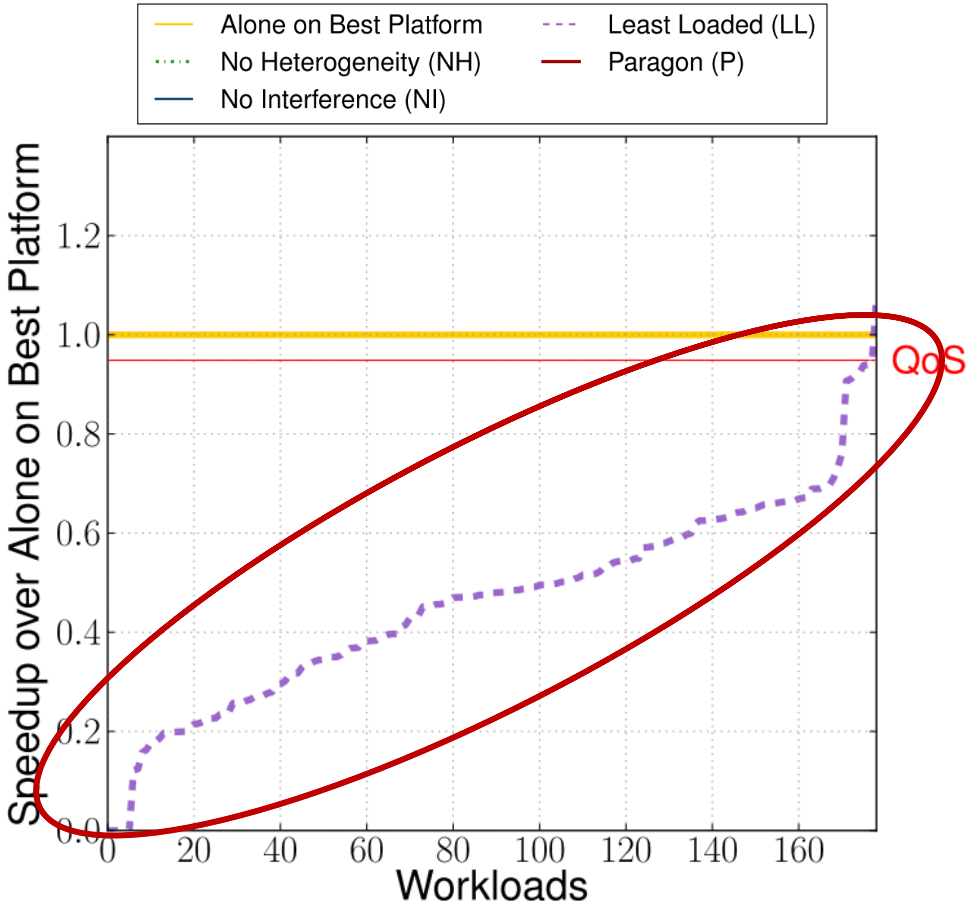
Outline

- Motivation
- Application Classification
- Paragon
- Evaluation

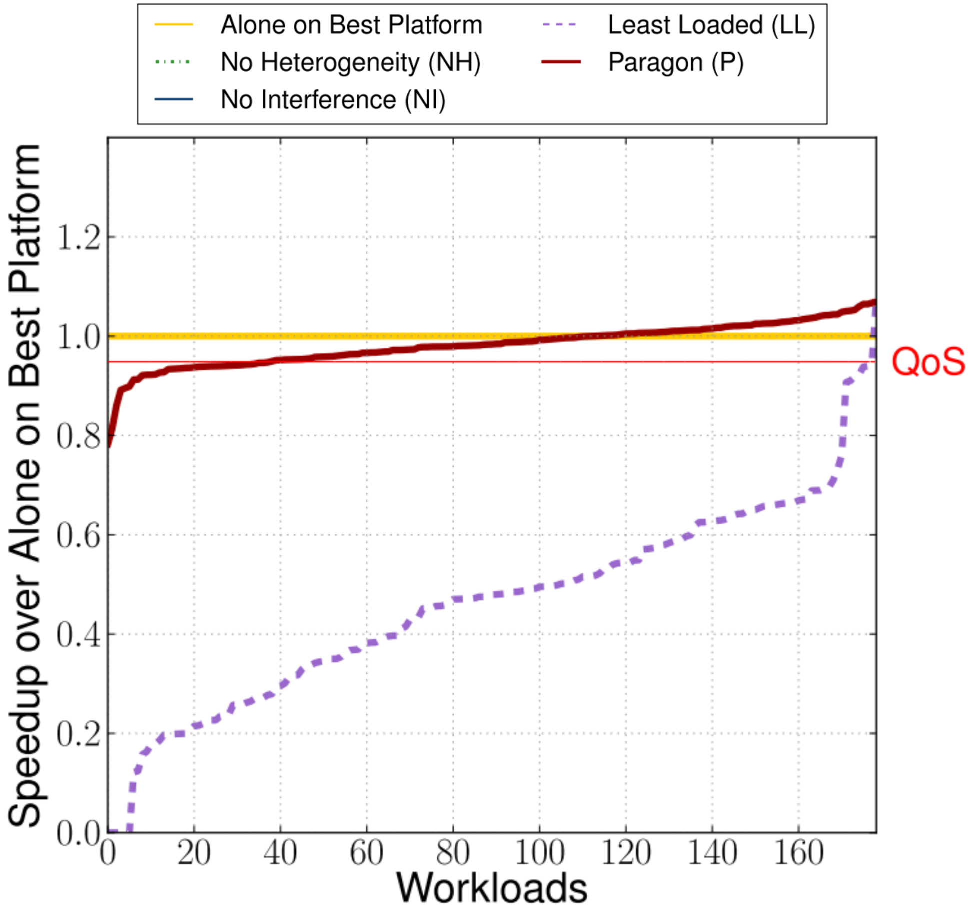
Methodology

- Workloads:
  - Single-threaded: SPEC CPU2006
  - Multi-threaded: PARSEC, SPLASH-2, BioParallel, Minebench, Specjbb
  - Multiprogrammed mixes: 350 4-app mixes of SPEC CPU2006
  - I/O: data mining, Matlab, single-node Hadoop
- Systems:
  - Small-scale → 40-machine local cluster (10 configurations)
  - Large-scale → 1,000 EC2 servers (14 configurations)
- Workload Scenarios:
  - Low load, high load, with phases and oversubscribed

Evaluation: Small Scale (high load)

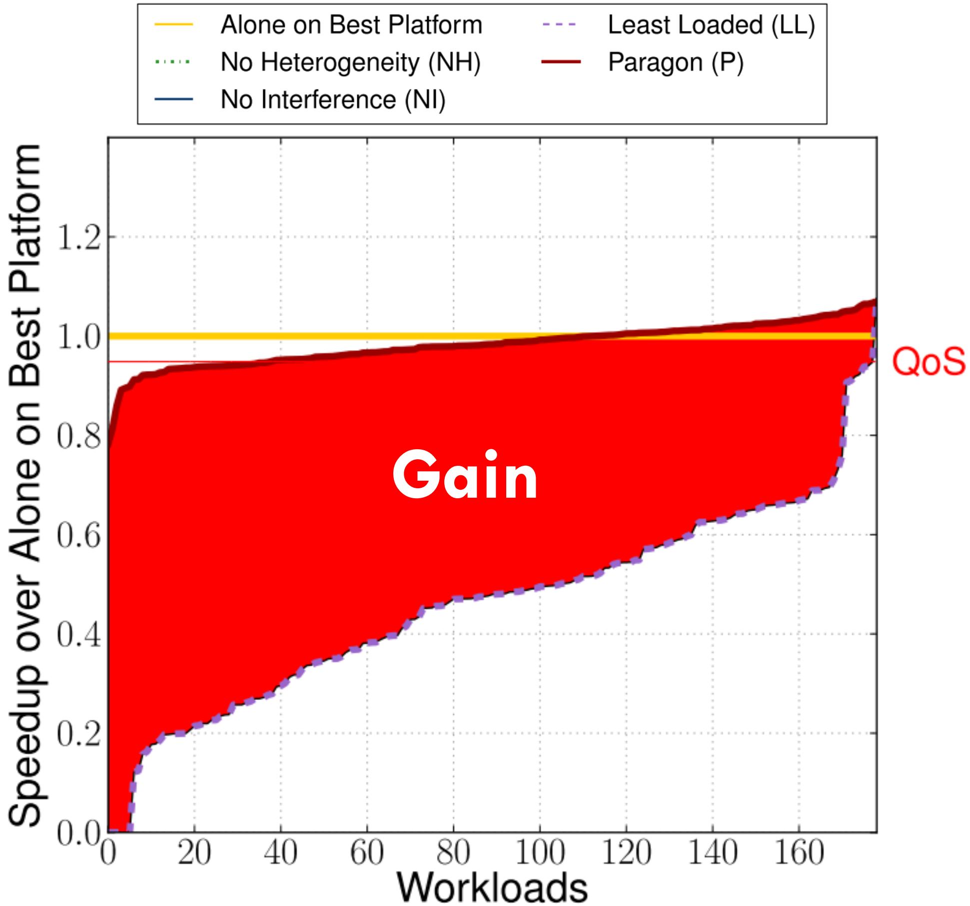


Evaluation: Small Scale (high load)



- Paragon preserves QoS for 64% of workloads
- Bounds degradation to less than 10% degradation for 90% of workloads

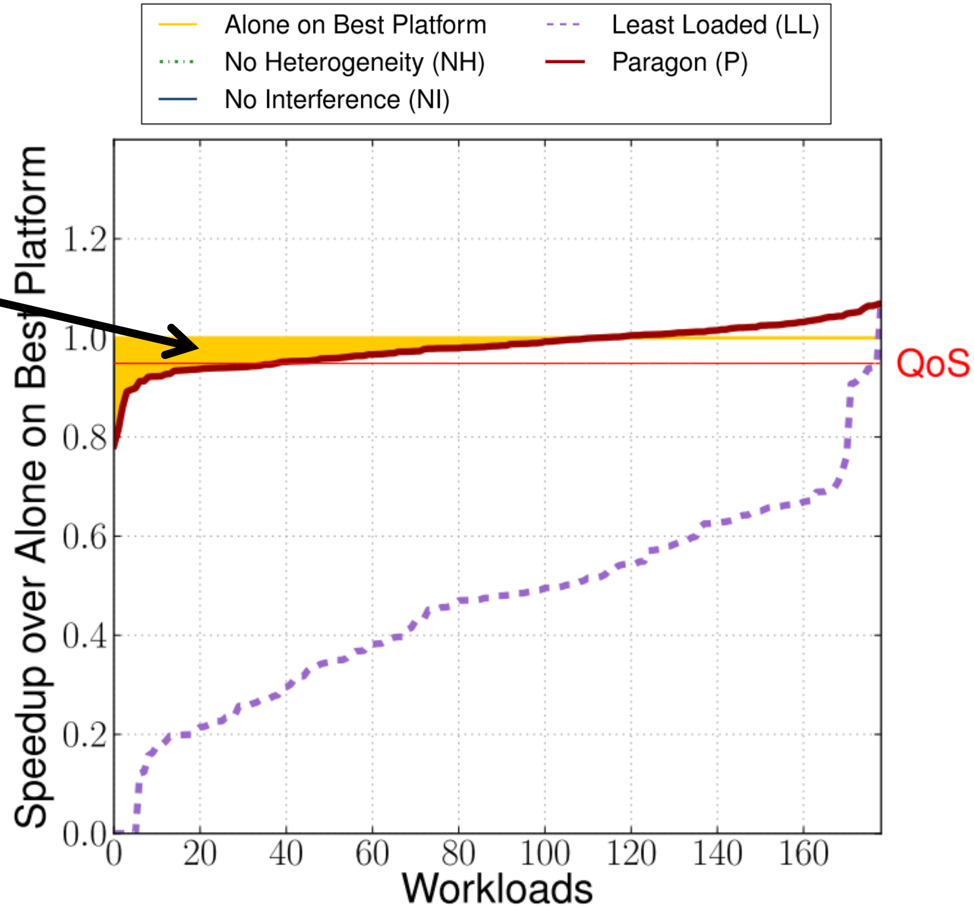
Evaluation: Small Scale (high load)



- Paragon preserves QoS for 64% of workloads
- Bounds degradation to less than 10% degradation for 90% of workloads

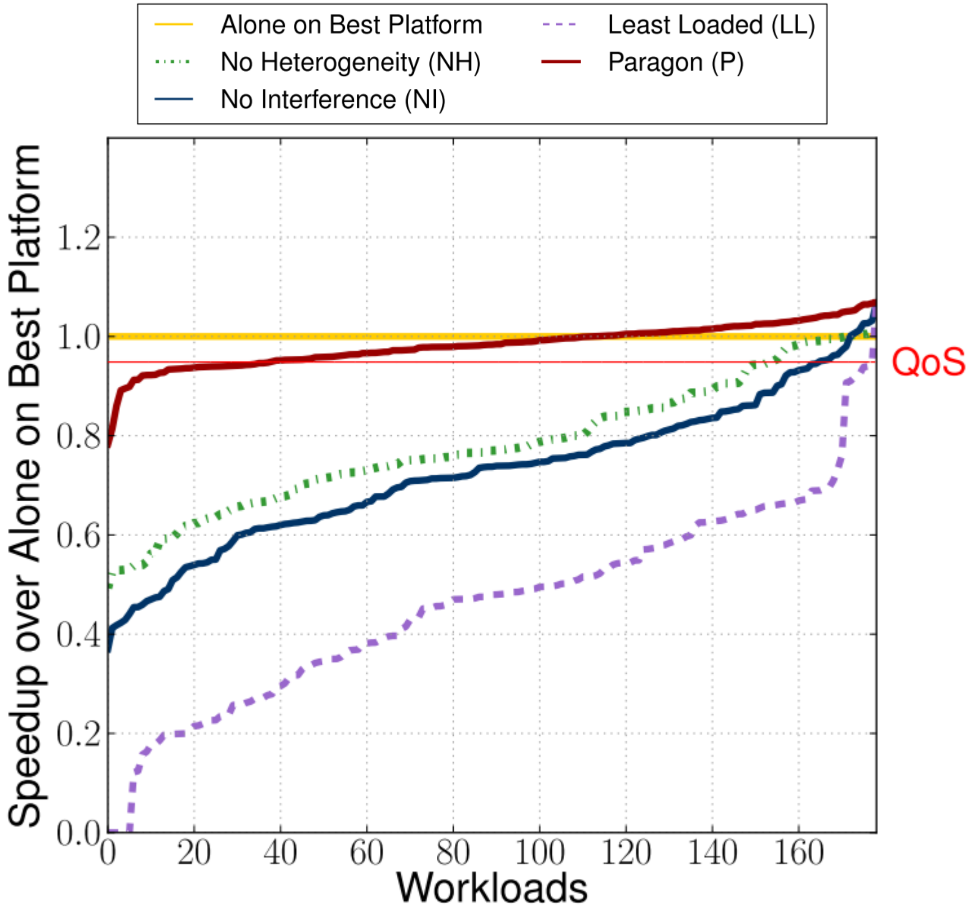
Evaluation: Small Scale (high load)

Distance from optimal



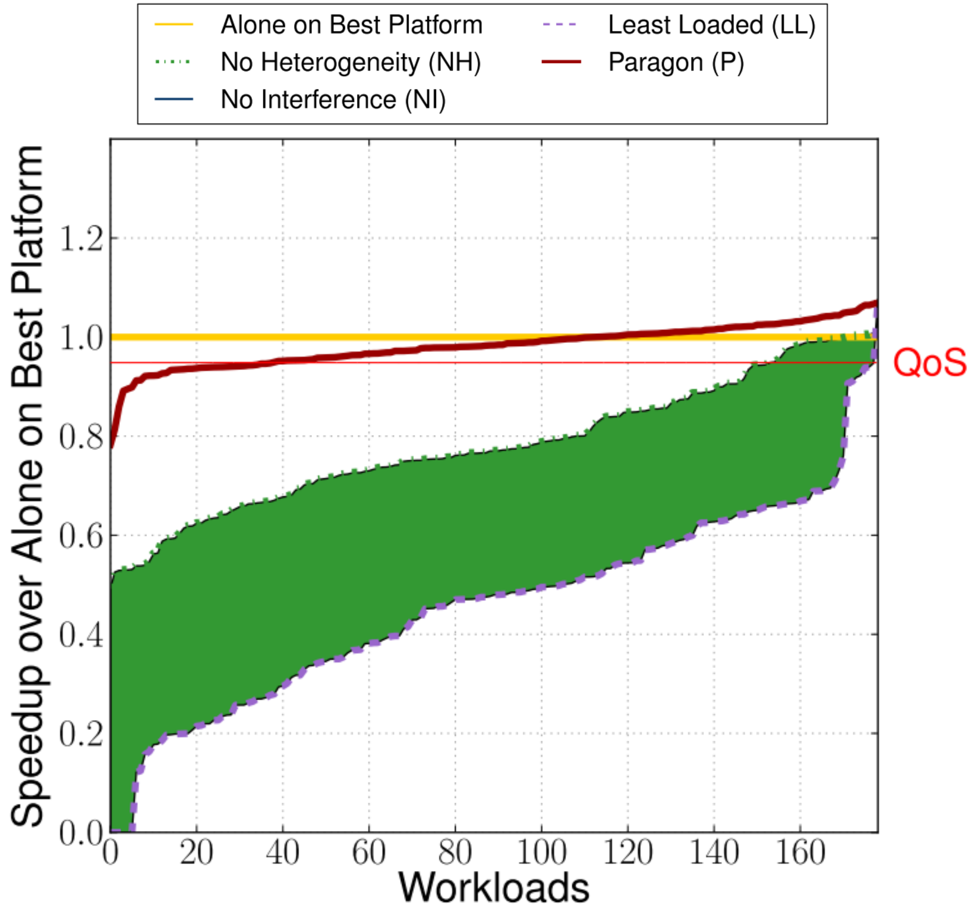
- Paragon preserves QoS for 64% of workloads
- Bounds degradation to less than 10% degradation for 90% of workloads

Evaluation: Small Scale (high load)



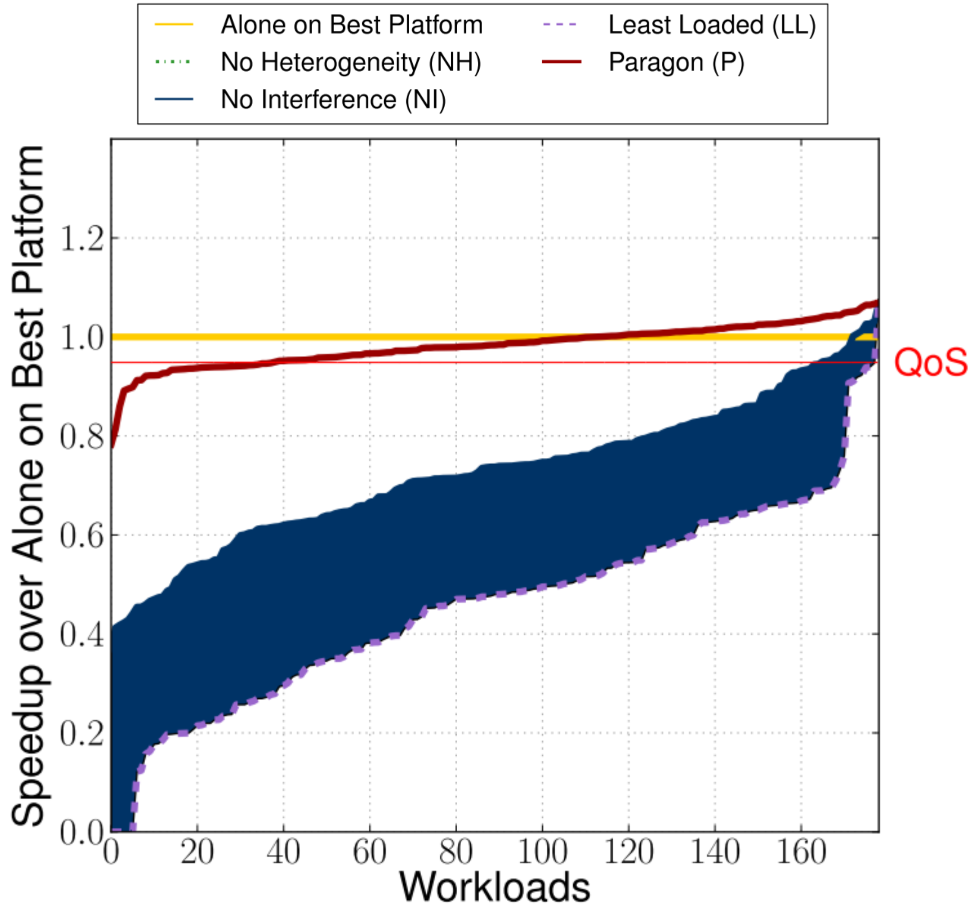
- Paragon preserves QoS for 64% of workloads
- Bounds degradation to less than 10% degradation for 90% of workloads

Evaluation: Small Scale (high load)



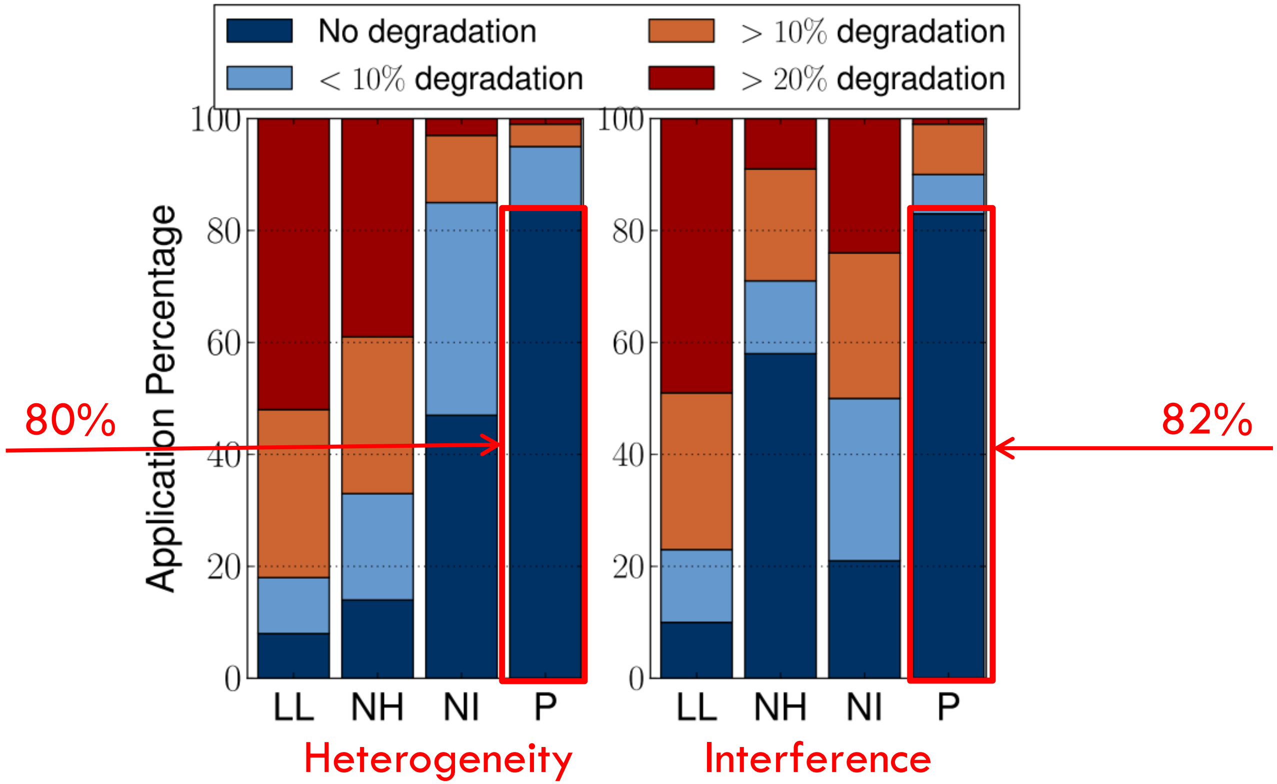
- Paragon preserves QoS for 64% of workloads
- Bounds degradation to less than 10% degradation for 90% of workloads

Evaluation: Small Scale (high load)



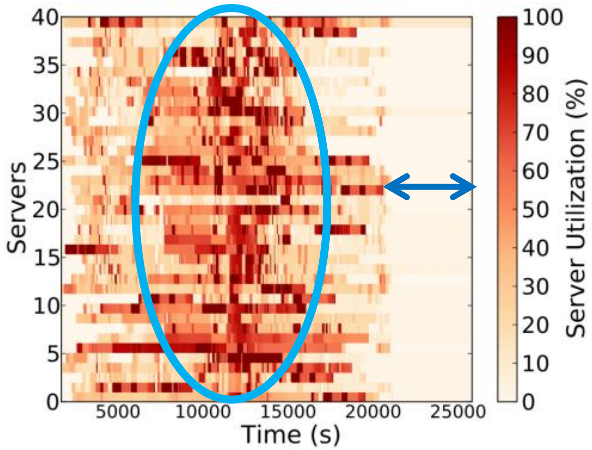
- Paragon preserves QoS for 64% of workloads
- Bounds degradation to less than 10% degradation for 90% of workloads

Decision Quality

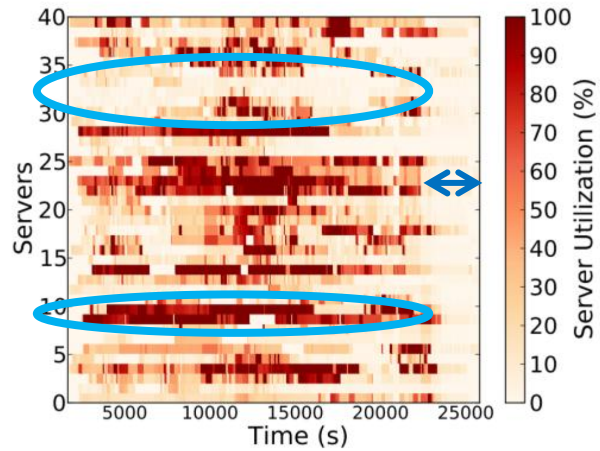


- LL: poor decision quality both for heterogeneity and interference
- NH: poor platform decisions, good interference decisions
- NI: good platform decisions, poor interference decisions
- Paragon: better than NI in heterogeneity, better than NH in interference

Increasing Utilization



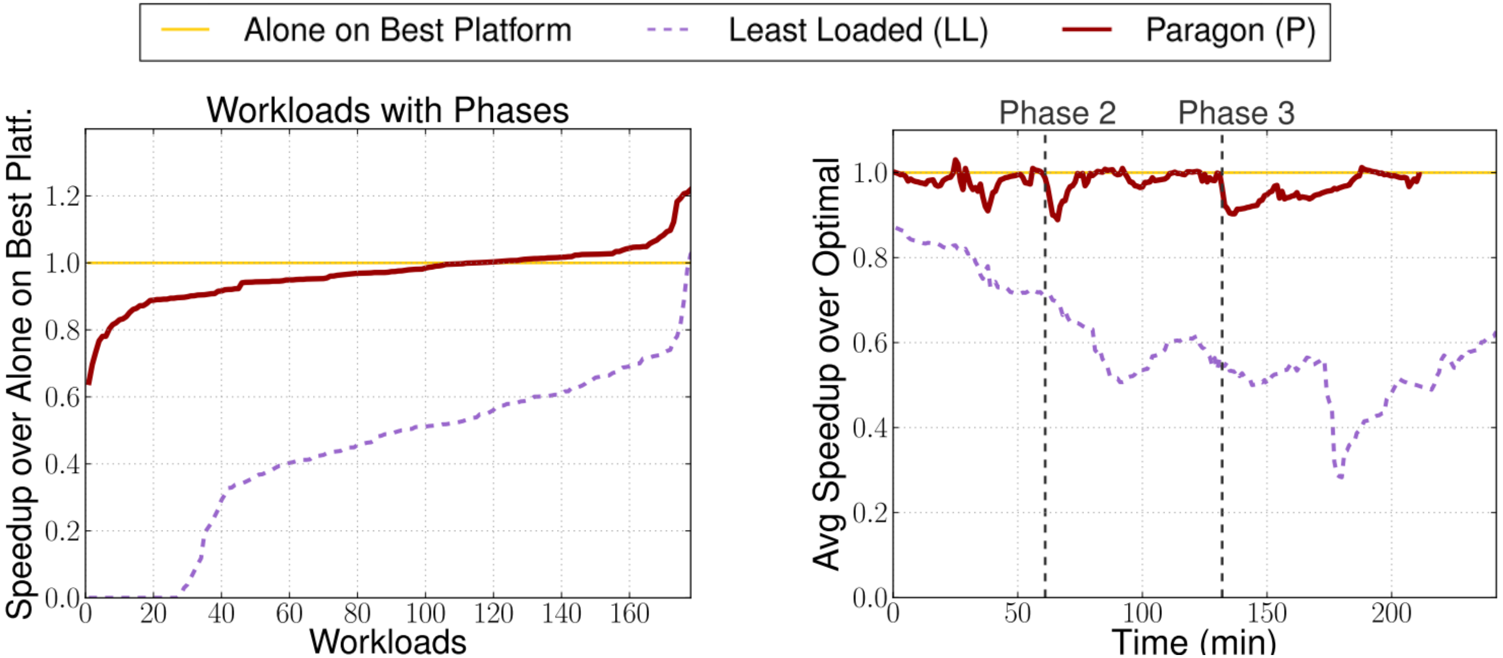
Paragon



Least-Loaded (LL)

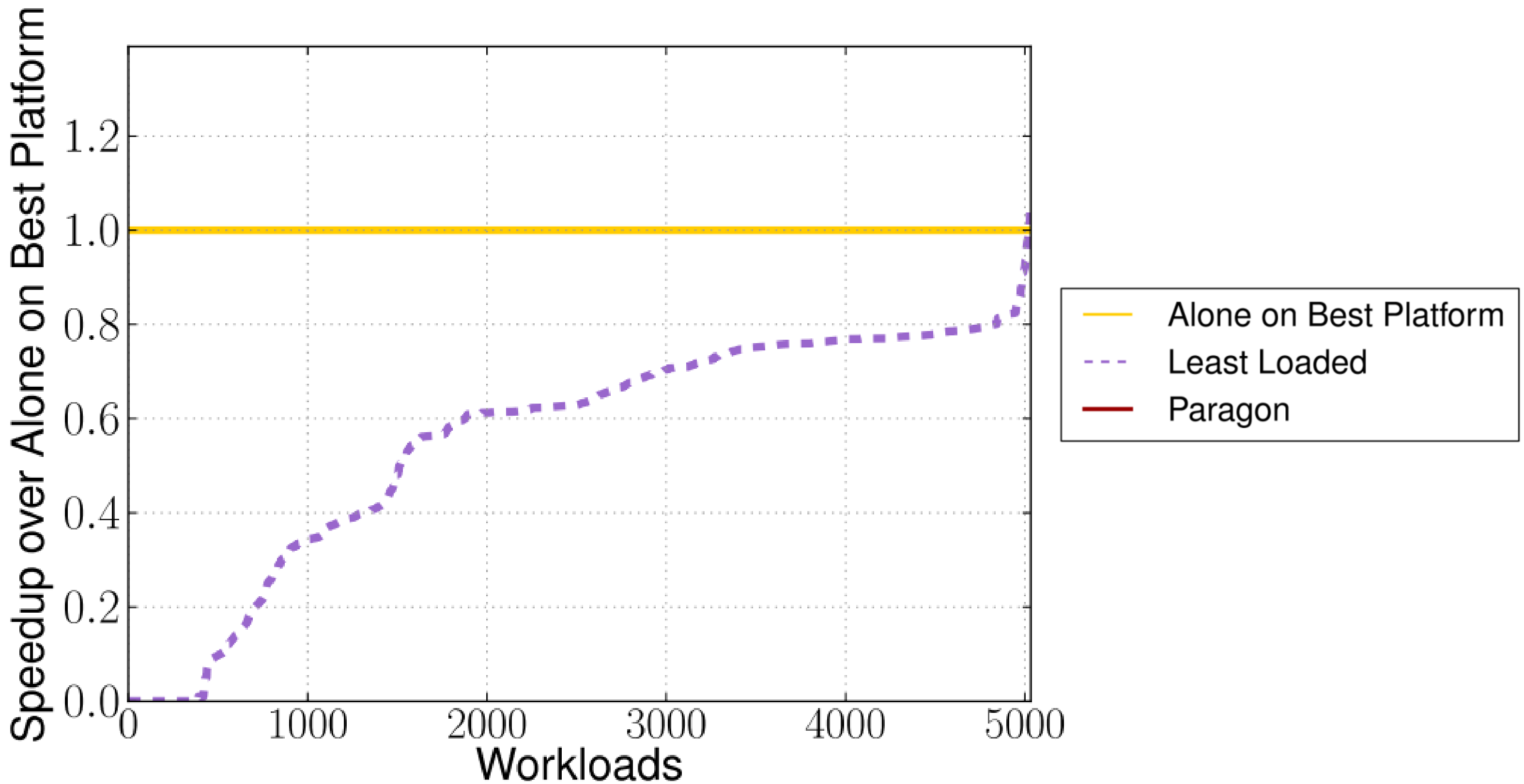
- Paragon increases server utilization by 47%:
  - Same performance for user (QoS guarantees)
  - Better utilization for the DC operator → resource efficiency
- With baseline (LL):
  - Imbalance in server utilization (too high vs. too low)
  - Per-app QoS violations + scenario execution time increase

Workloads with Phases



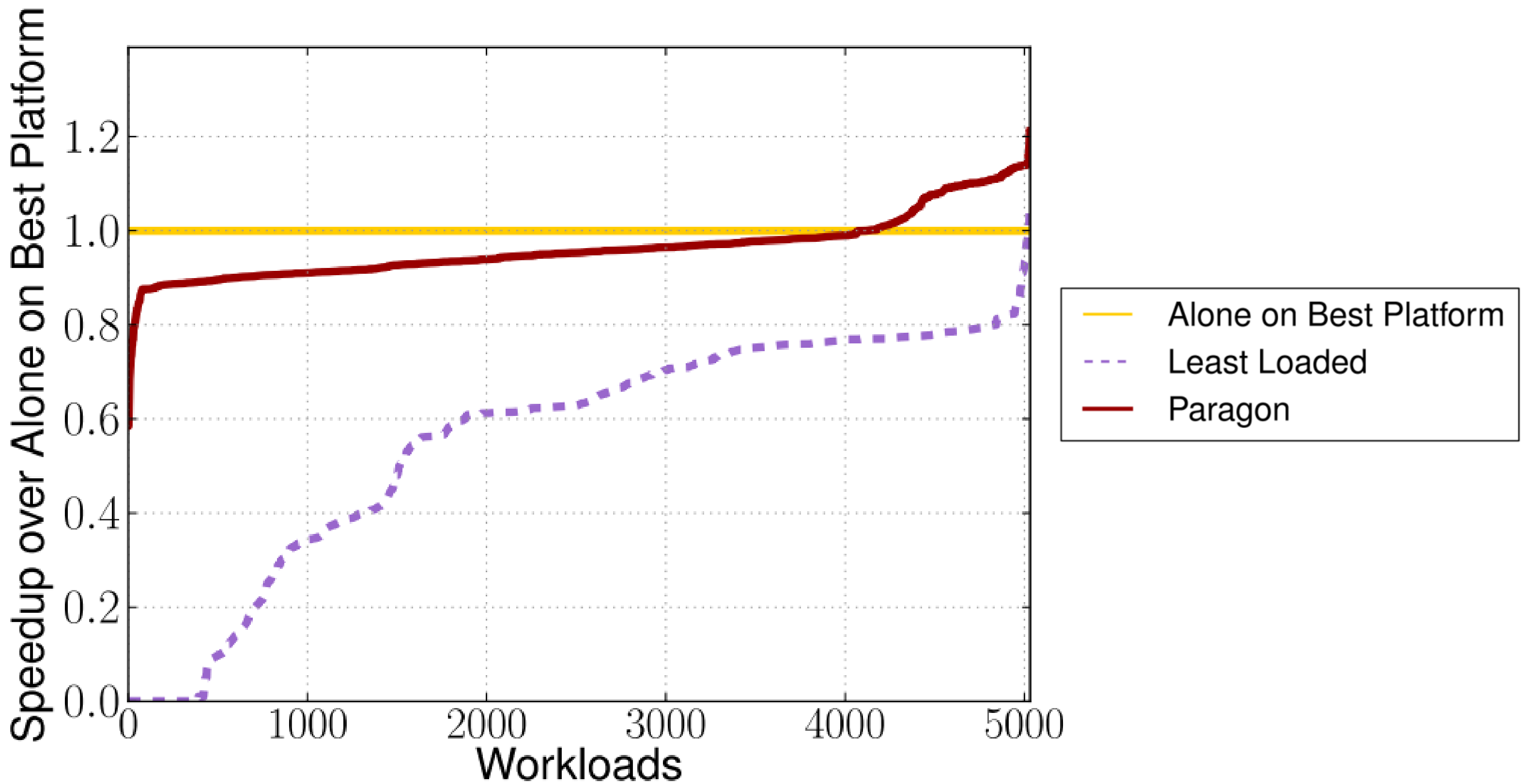
- QoS is preserved for 75% of applications
  - Using the other schedulers preserves QoS for < 10% of apps
- Paragon adapts to workload phases over time → performance recovers shortly after the phase change

Large Scale (EC2) – High Load



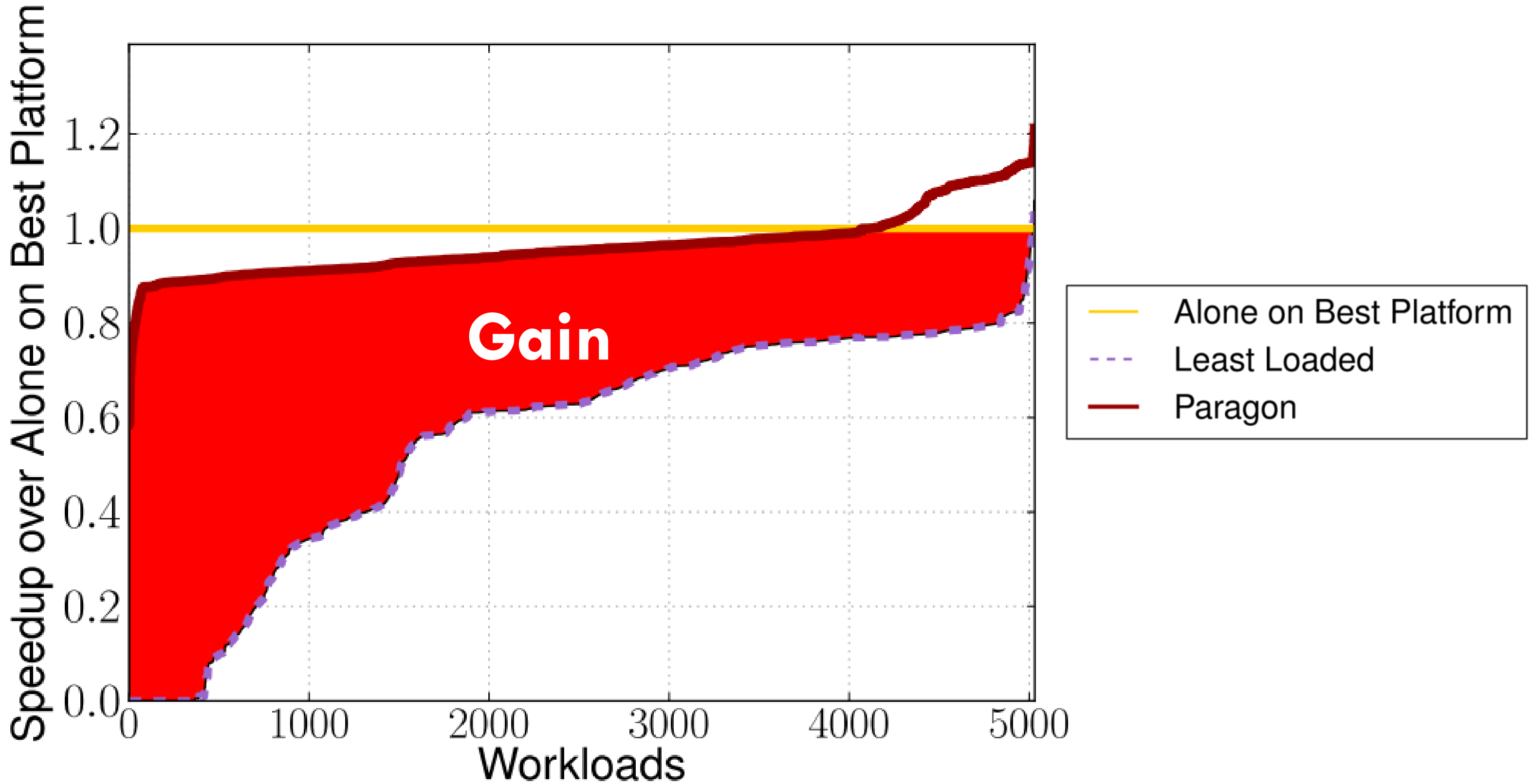
- LL: violates QoS for 99% of workloads
- NH: violates QoS for 96% of workloads
- NI: violates QoS for 97% of workloads

Large Scale (EC2) – High Load



- Paragon preserves QoS for 61% of workloads
- Bounds degradation to less than 10% for 90% of workloads.

Large Scale (EC2) – High Load



- Paragon preserves QoS for 61% of workloads
- Bounds degradation to less than 10% for 90% of workloads.

Conclusions

- A heterogeneity and interference aware DC scheduler
- Leverages robust analytical methods to quickly classify apps
- Minimizes interference and maximizes utilization
- It is scalable and lightweight

Questions?

Thank you!

cdel@stanford.edu

http://paragonDC.stanford.edu