ASCAR

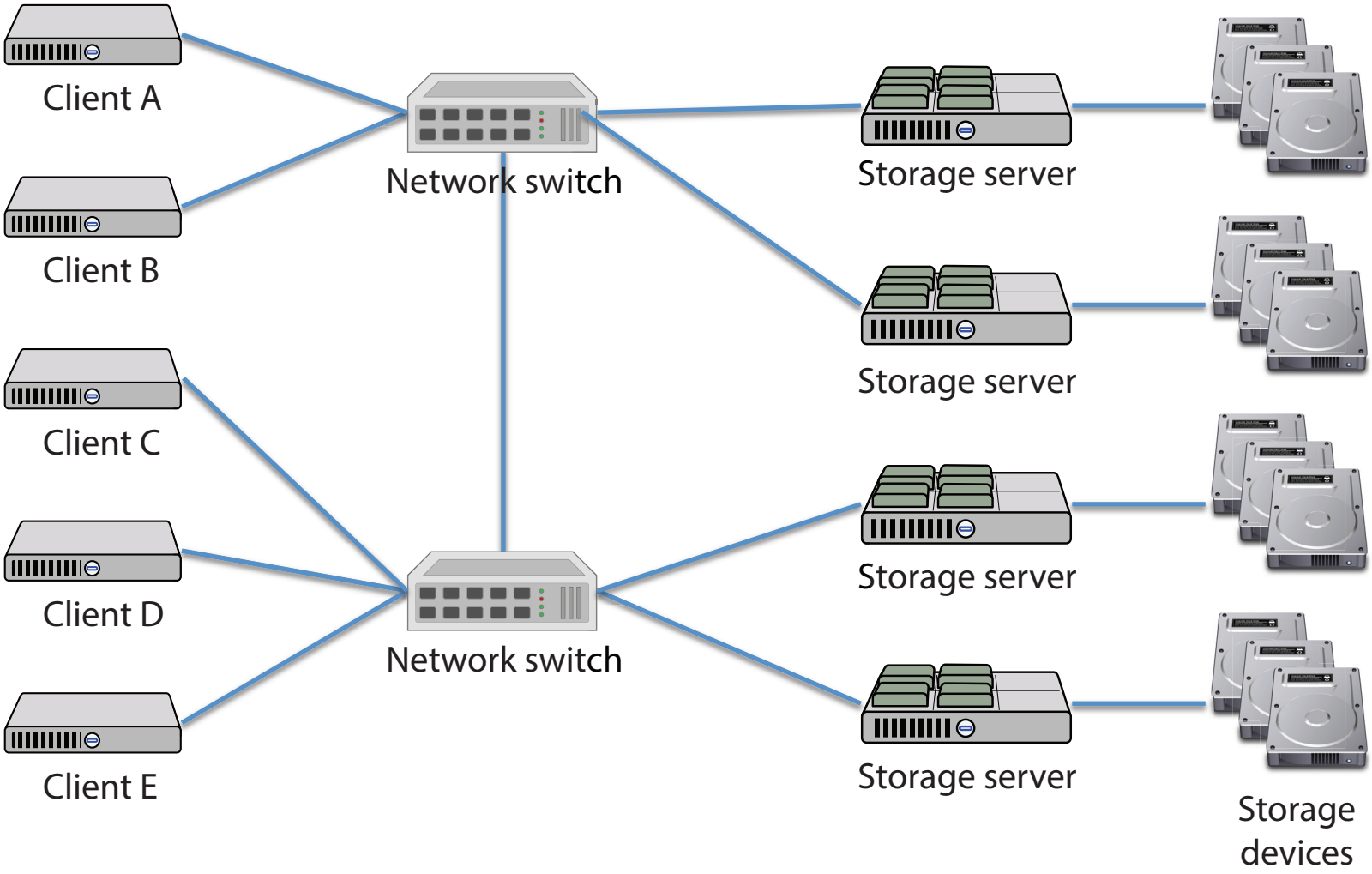
Automating Contention Management for High-Performance Storage Systems

(MSST '15, June 5th, 2015)

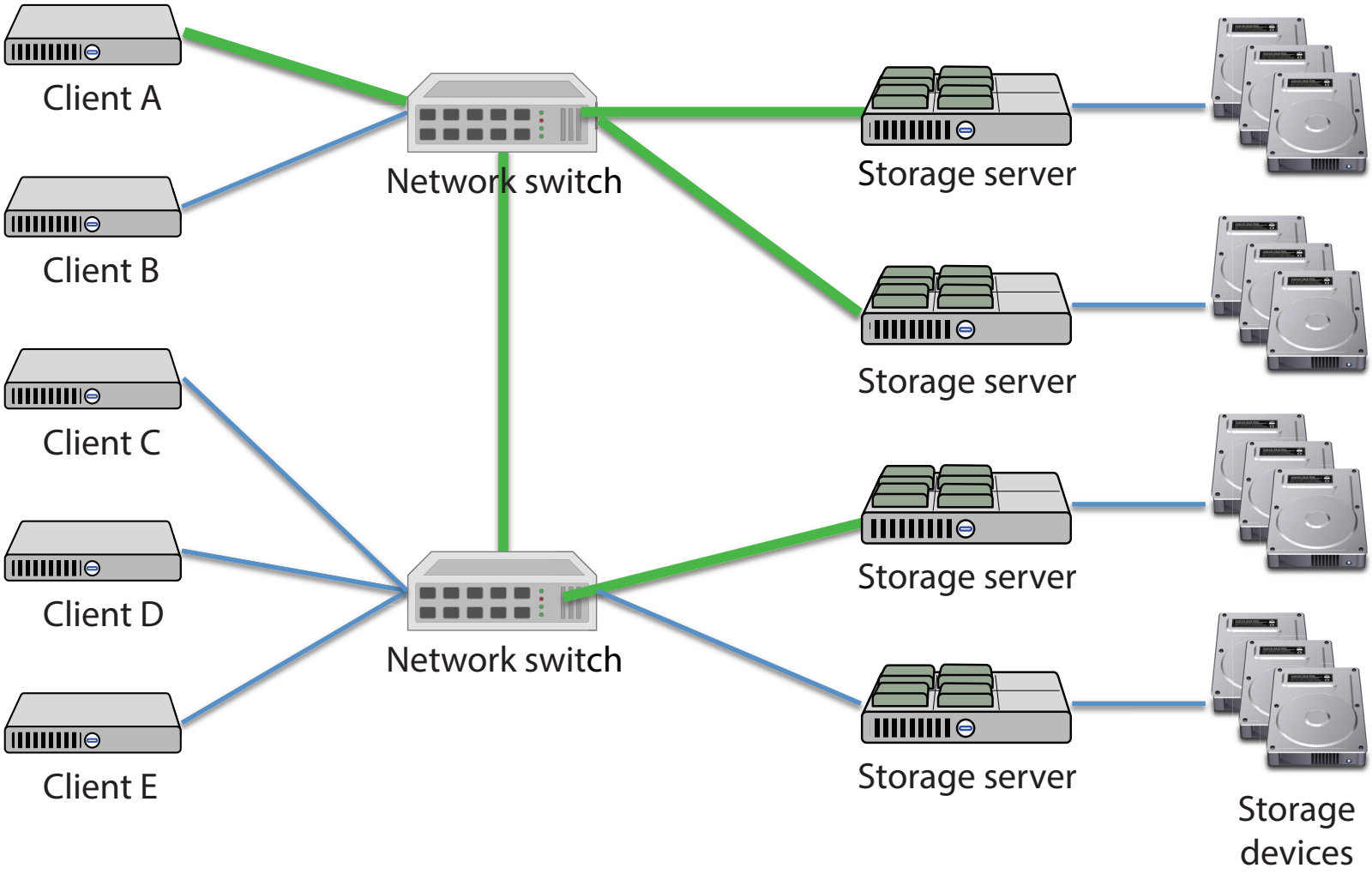
Yan Li, Xiaoyuan Lu,
Ethan Miller, Darrell Long
Storage Systems Research Center (SSRC)
University of California, Santa Cruz

Baskin Engineering
UC SANTA CRUZ

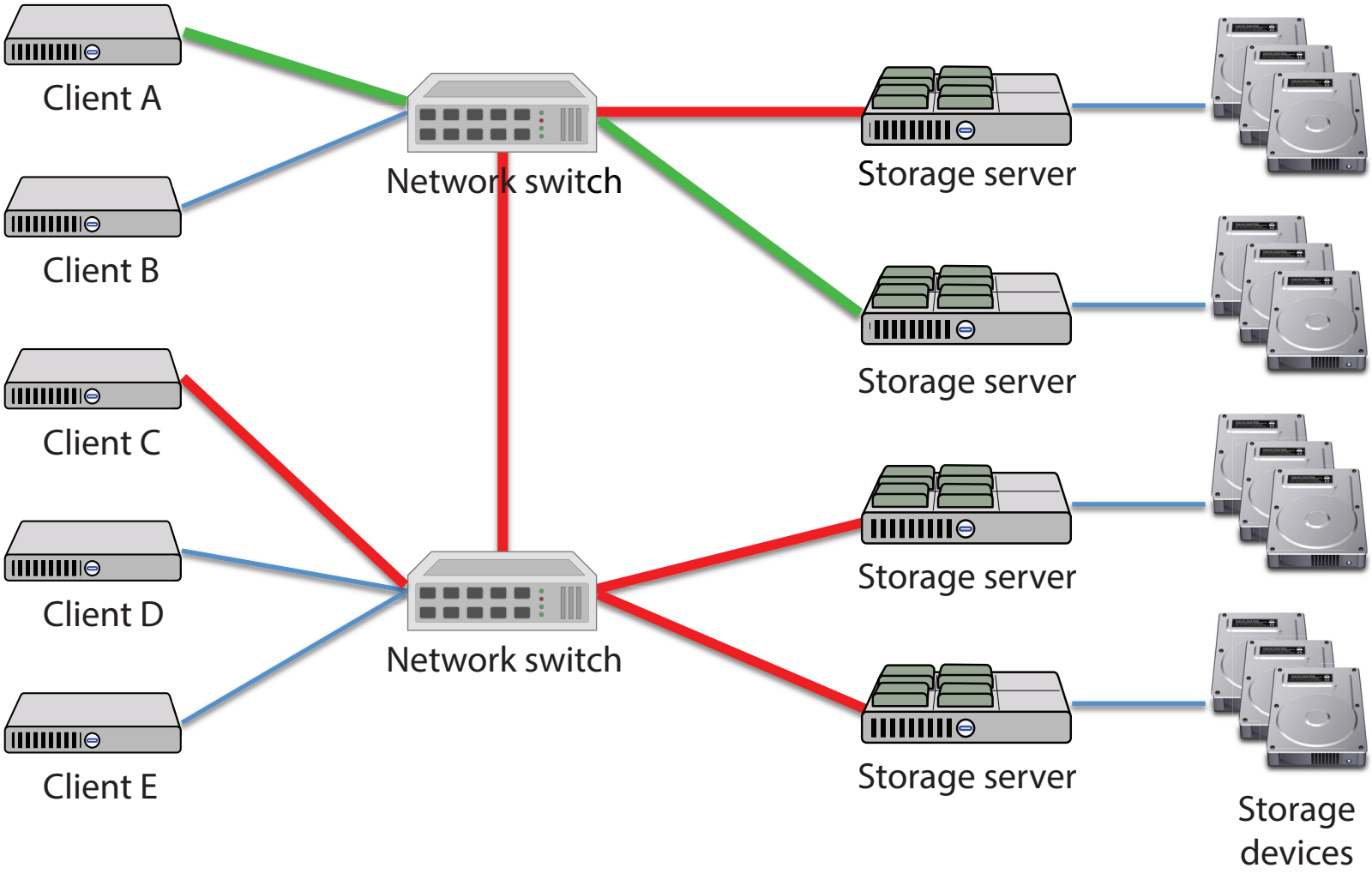
Contention in modern shared-storage system



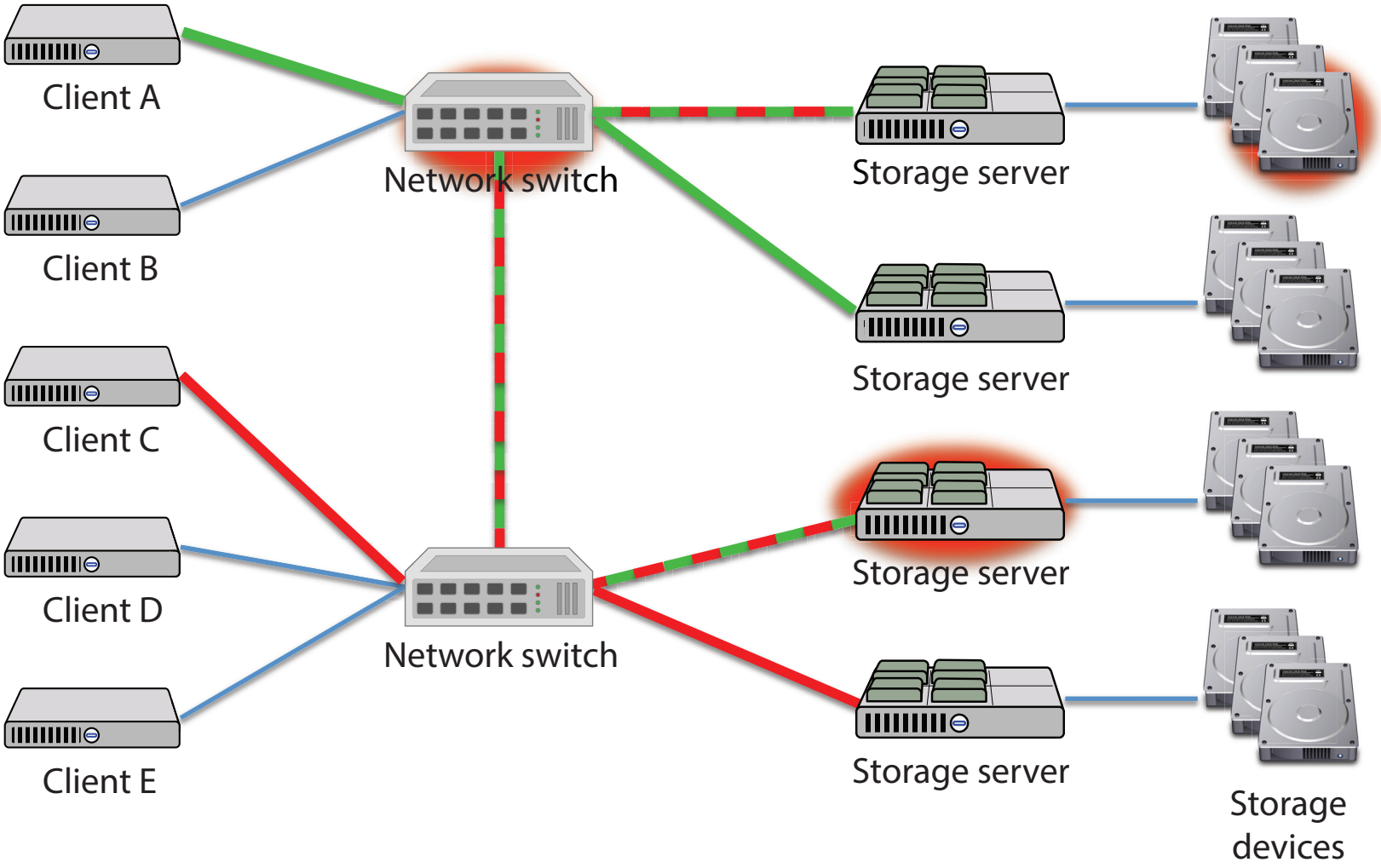
Contention in modern shared-storage system



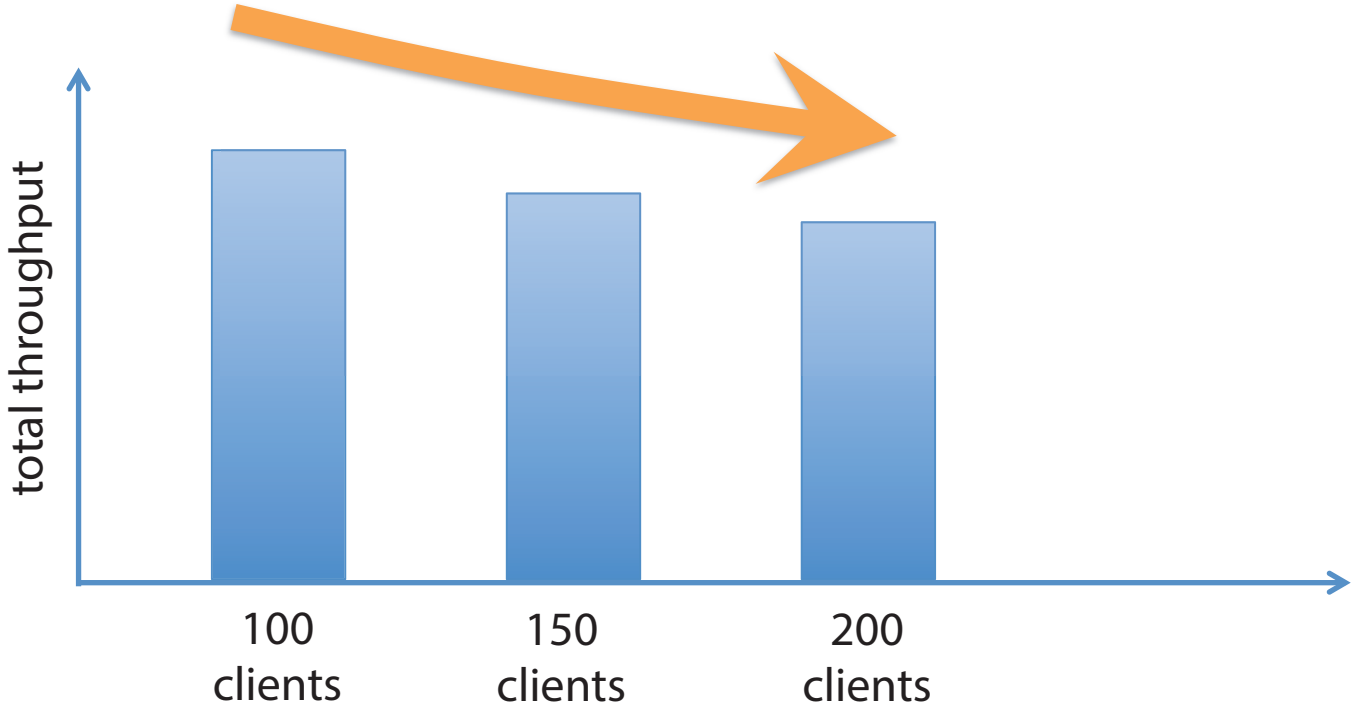
Contention in modern shared-storage system



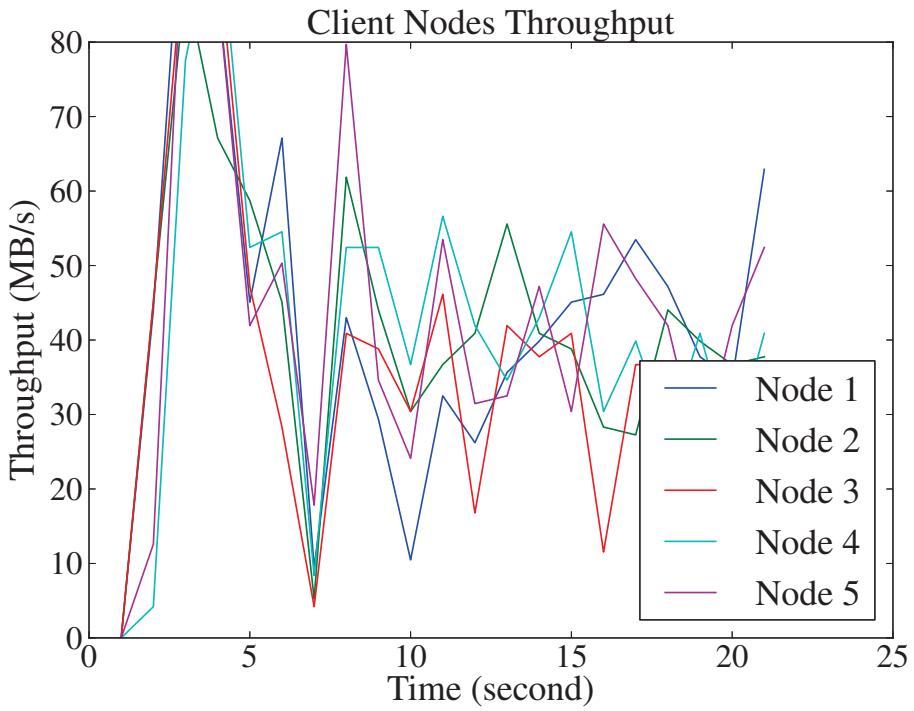
Contention in modern shared-storage system



Contention harms efficiency



Contention causes fluctuation



Client throughput of a random write workload

5 nodes accessing 5 servers

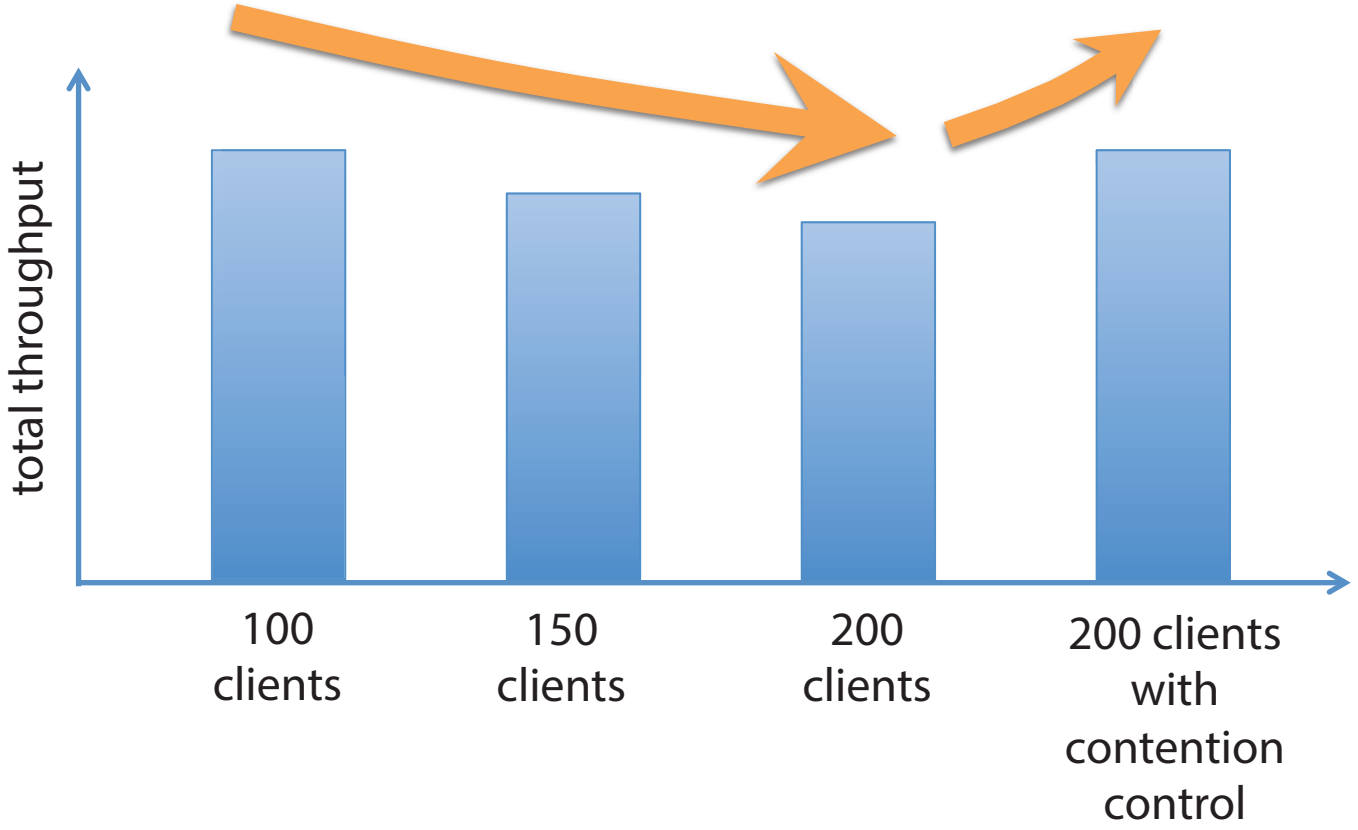
High temporal and spatial I/O speed fluctuation

This is what's happening at each server node

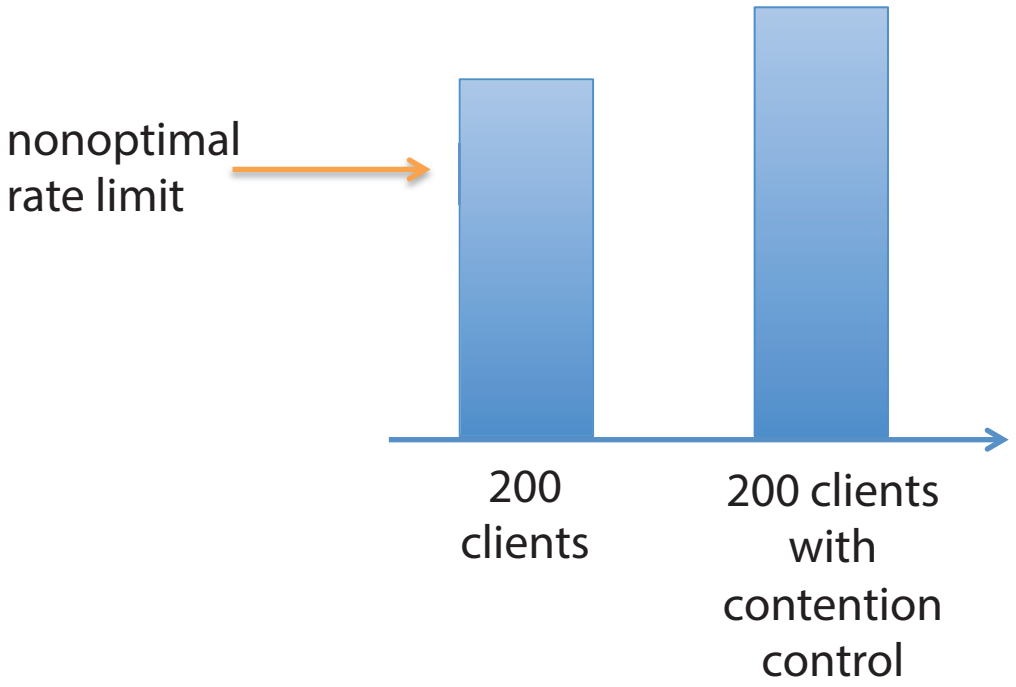
Picture credit:
https://www.checkfelix.com/reiseblog/10-argumente-urlaub-schottland/

Limiting client I/O rates can improve performance

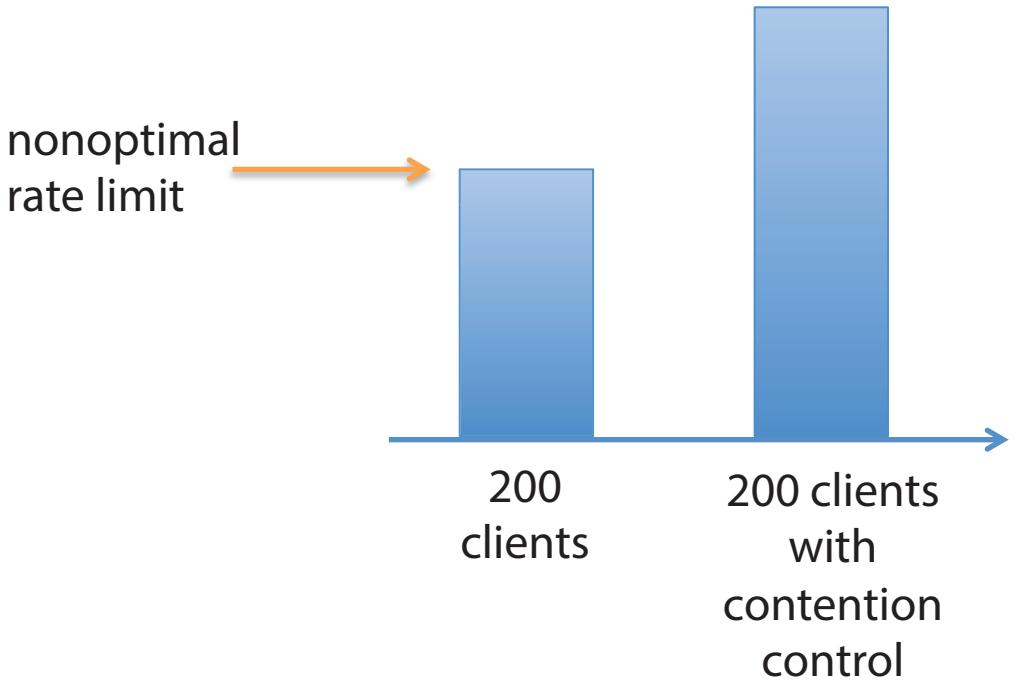
… if done properly



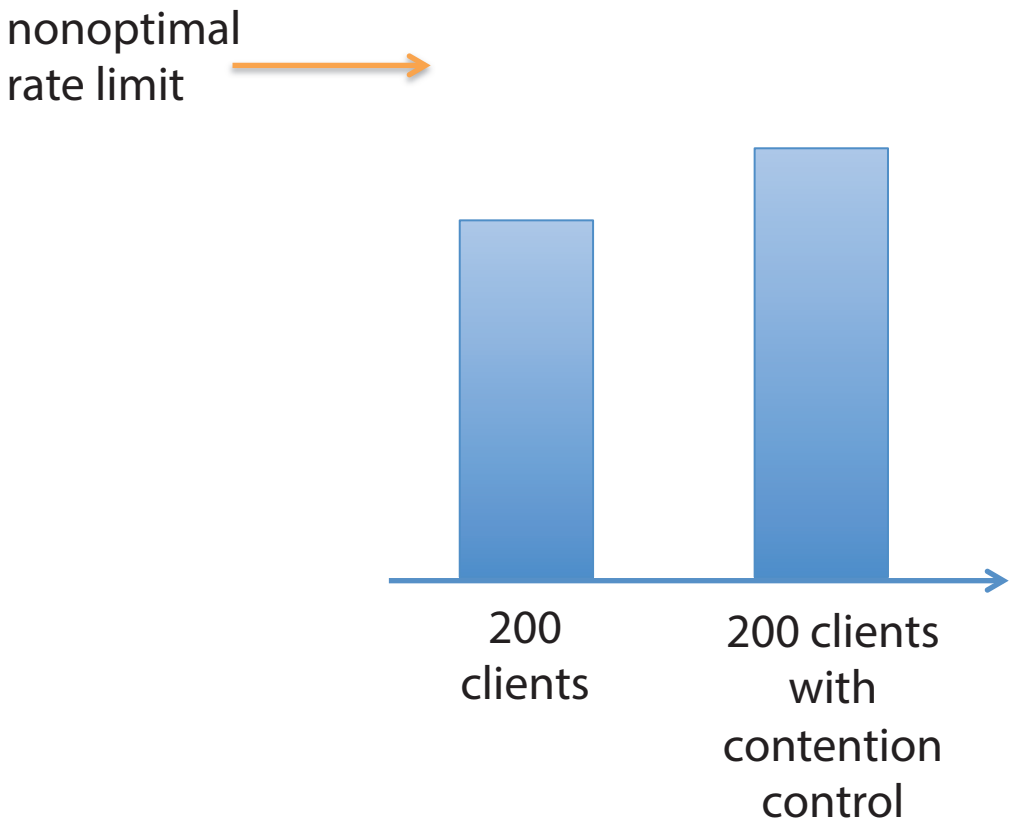
Challenges of distributed I/O rate control: 1. **capability discovery**



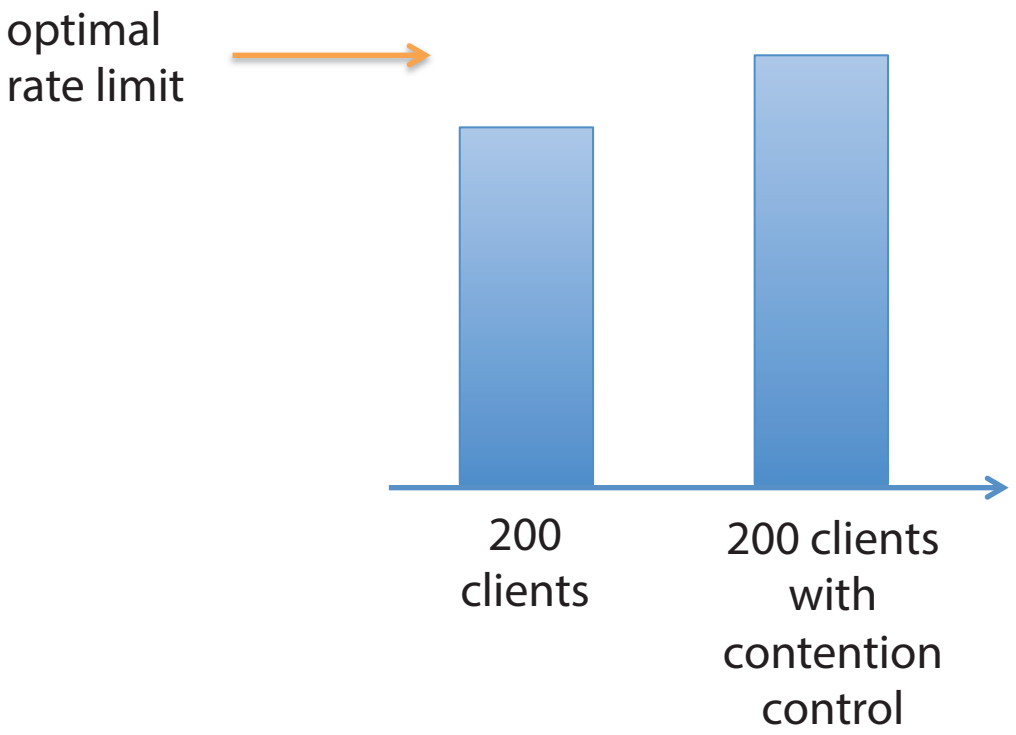
Challenges of distributed I/O rate control:
1. **capability discovery**



Challenges of distributed I/O rate control: 1. **capability discovery**



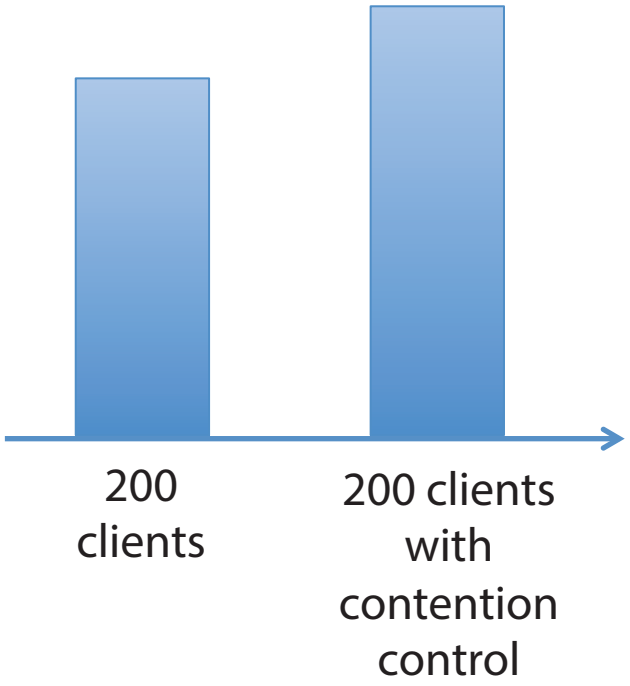
Challenges of distributed I/O rate control: 1. **capability discovery**



Challenges of distributed I/O rate control: 1. **capability discovery**

Capability discovery usually involves communication:

- between clients
- with a central controller



Challenges of distributed I/O rate control: 2. **scalability**

Capability discovery usually involves communication:

- between clients
- with a central controller

Intra-node communication grows at **$O(n^2)$**

Adds **overhead** to **congested** network

Low **responsiveness** for highly **dynamic** workload

Our goal

Reduce congestion and improve overall system efficiency

General purpose

Scalability

Reduce human involvement

Reduce management cost

Target environments

High Performance Computing

Homogeneous I/O among clients; bandwidth sensitive; requires fair bandwidth allocation

Virtual Machines accessing a shared image

Performance isolation

Enterprise computing

Interactive workloads, backup, indexing, data sharing

Future data center-wise storage system that serves multiple customers

Performance isolation

Core ideas

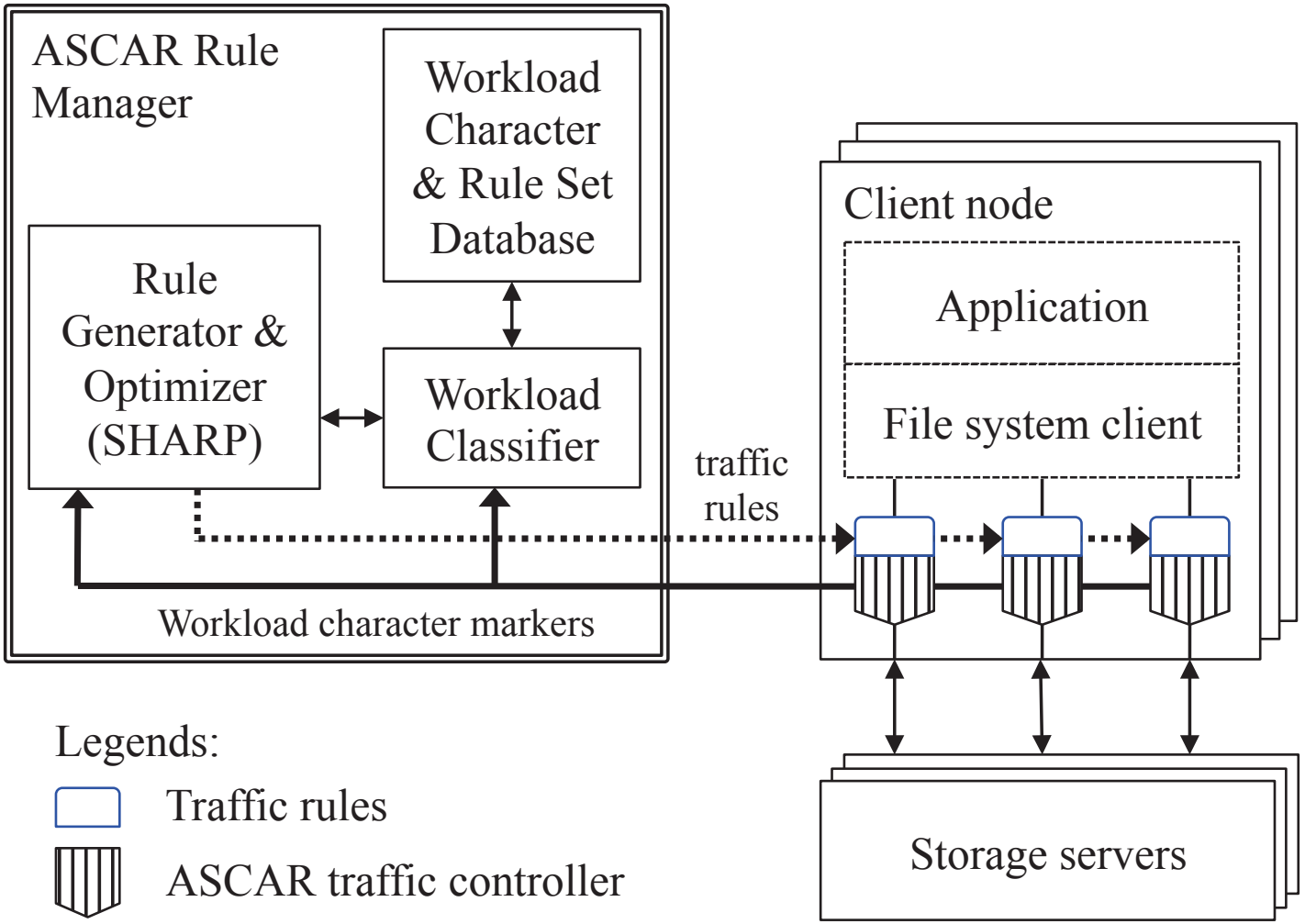
Client-side rule-based I/O rate control

1. no need for central scheduling or coordination, nimble and highly responsive
2. no need to change server software or hardware
3. no scale-up bottleneck

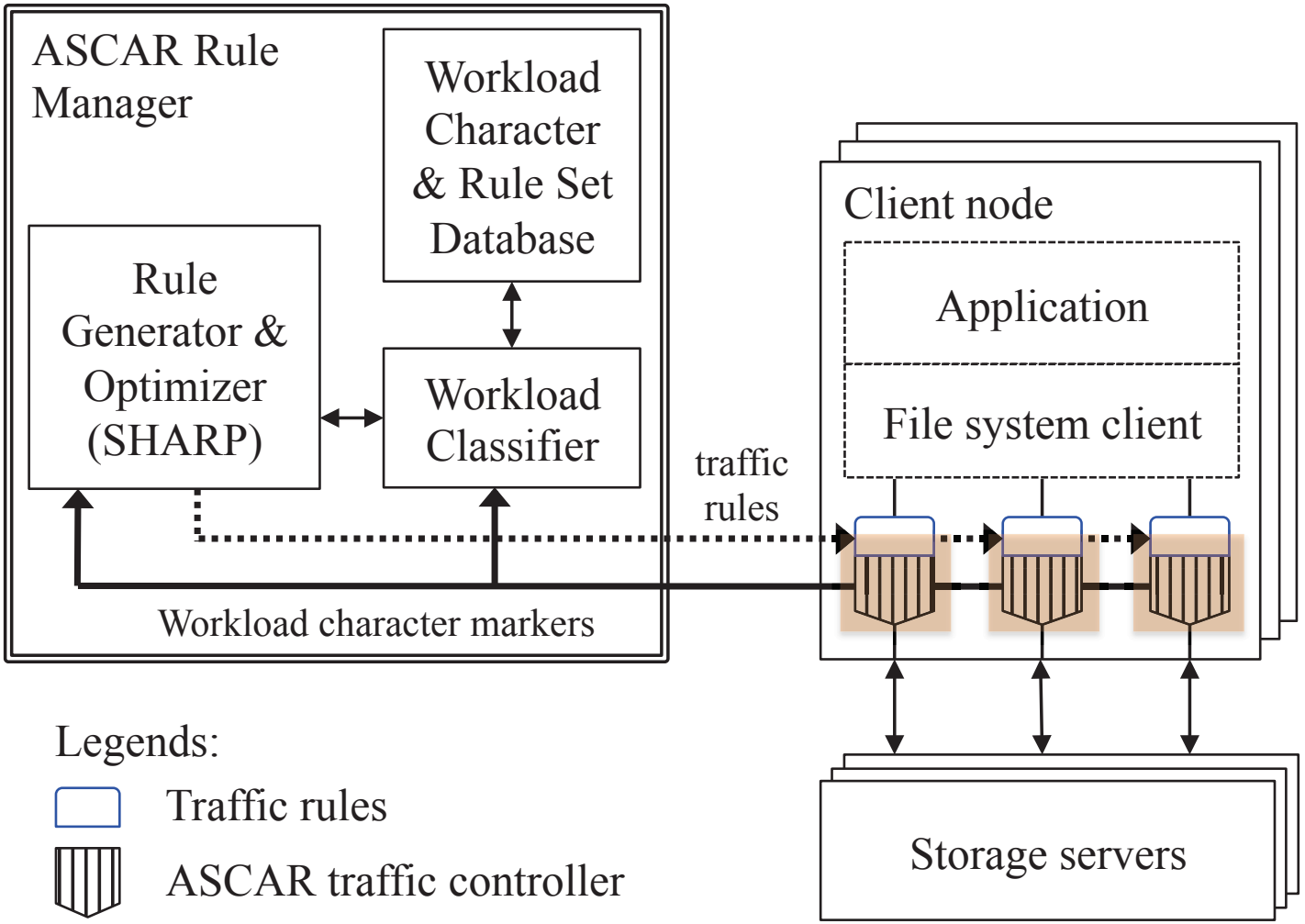
Use machine learning and heuristics for rule generation and optimization

no prior knowledge of the system or workload is required

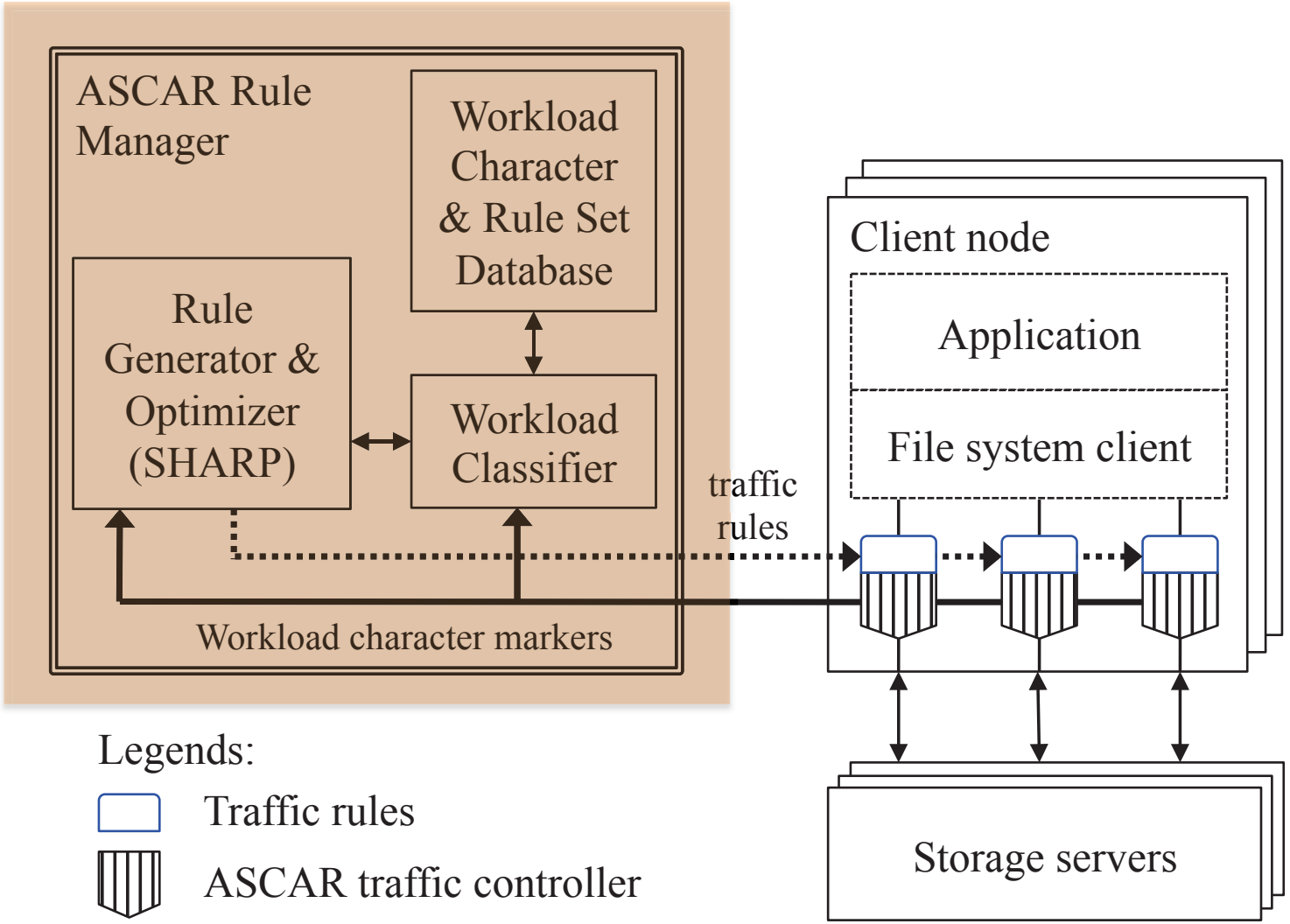
Components of the ASCAR prototype



Components of the ASCAR prototype



Components of the ASCAR prototype



Rule-based Contention Control

Rules tell the controller how to react to congestions
shrink the congestion window if congestion is building up

enlarge the congestion window if speed is increasing

Each rule maps a congestion state to an action
(Congestion State (CS) statistics) → <action>

Each client tracks three congestion state statistics
(ack_ewma, send_ewma, pt_ratio)

An action describes how to change the congestion window and rate limit
<m, b, τ>

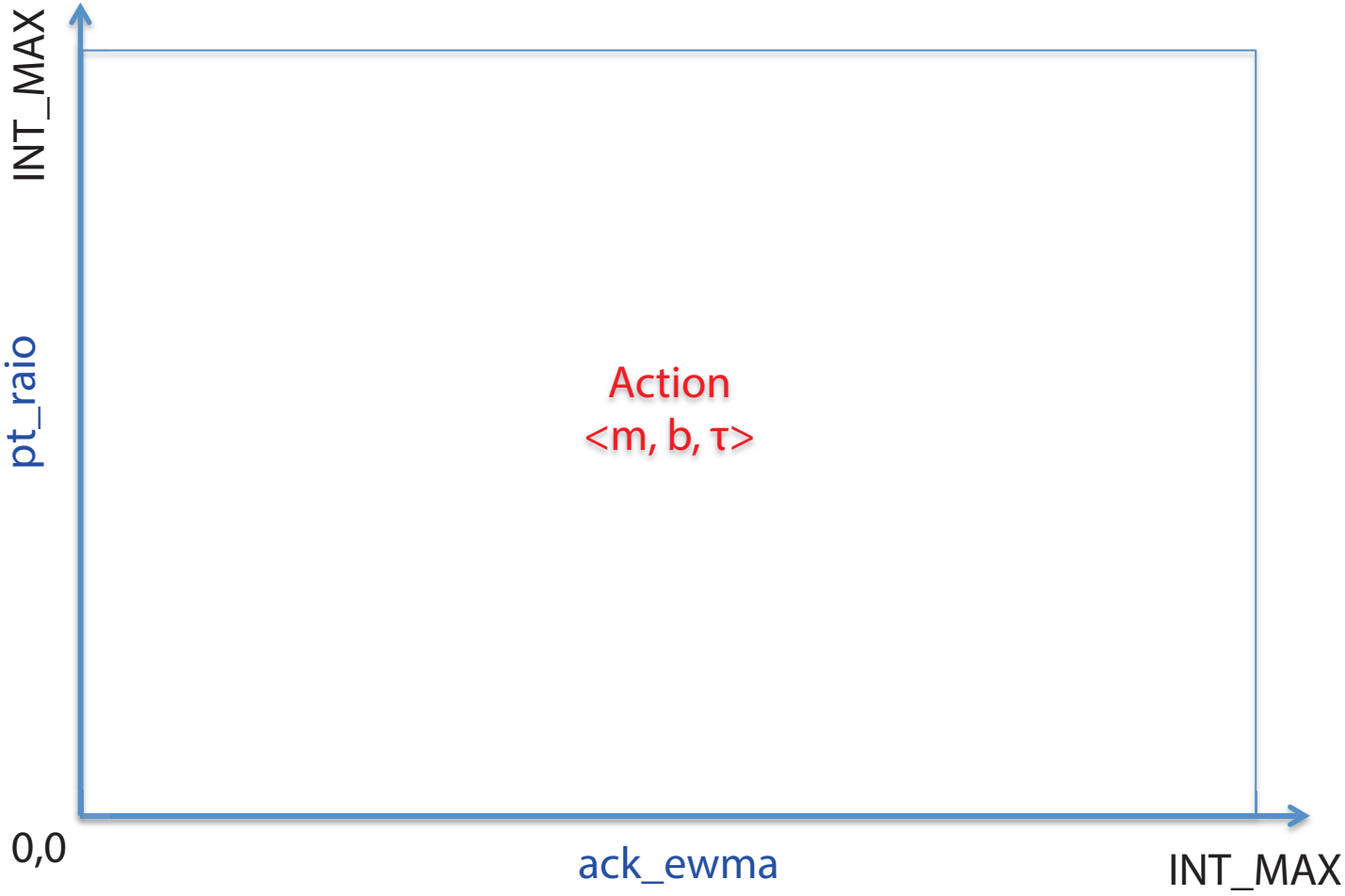
new_cong_window = m × cong_window + b

Rule set

A rule set contains many rules

A rule set covers the whole congestion state space

Rule set



Generating a rule set for a certain application

Extract a short signature workload that covers the important features of the application's I/O workload

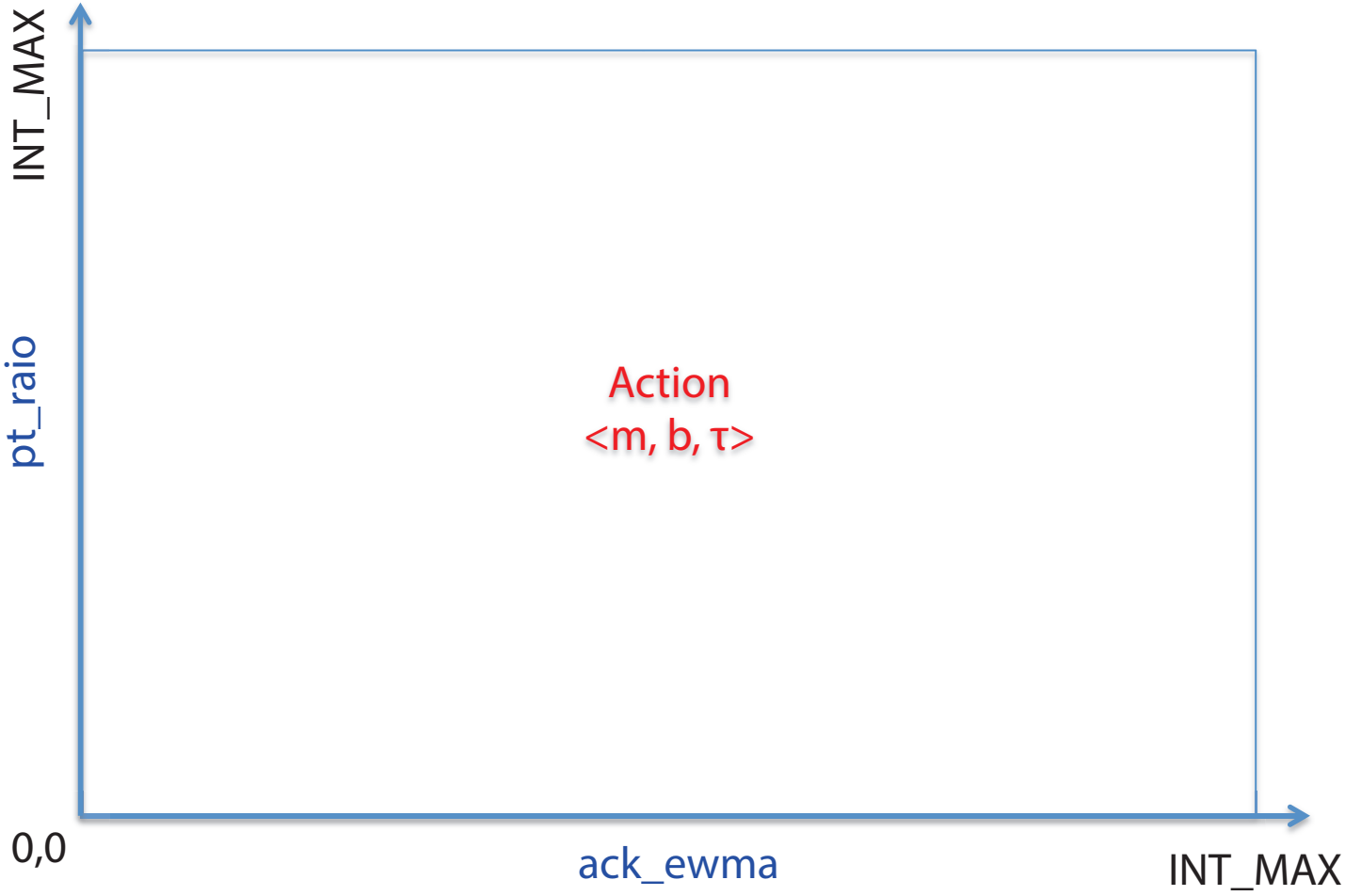
a signature workload is usually 20 ~ 30 seconds long

Generate candidate rules and benchmark them with the signature workload on the real system

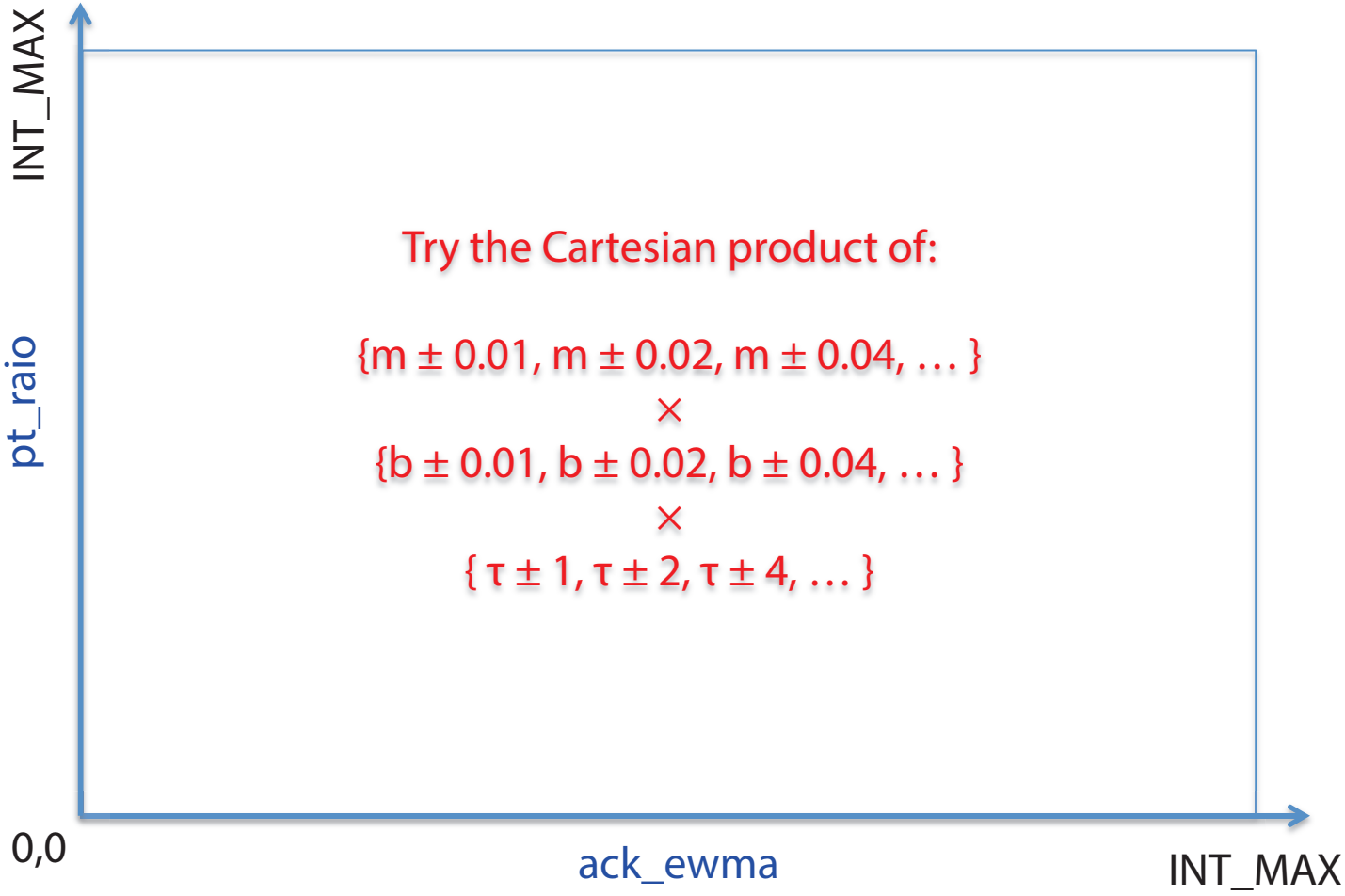
test possible combinations of action variables

Split the hottest rule in the set to generate more rules

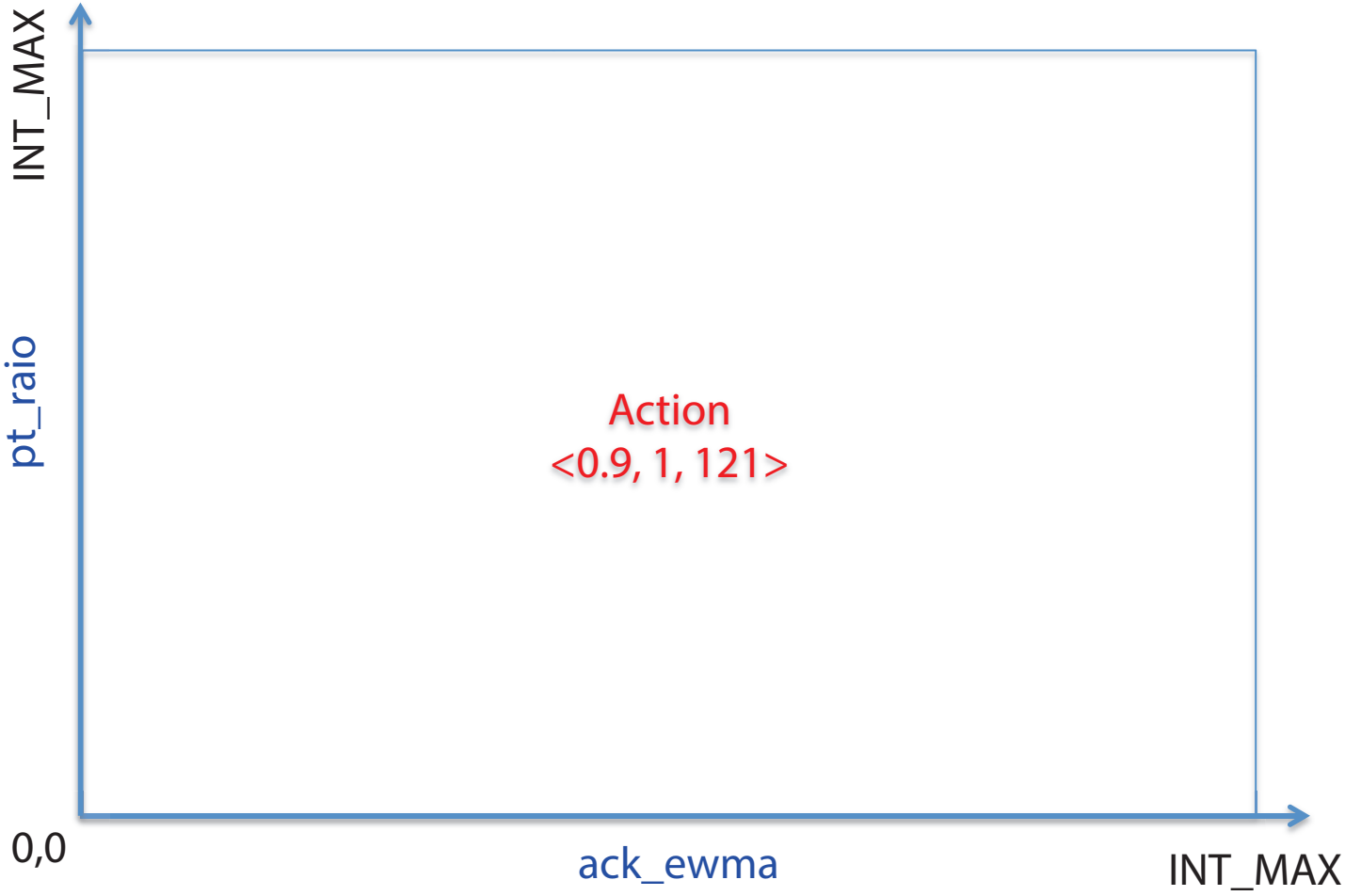
Begin with one rule: the whole state space maps to one action



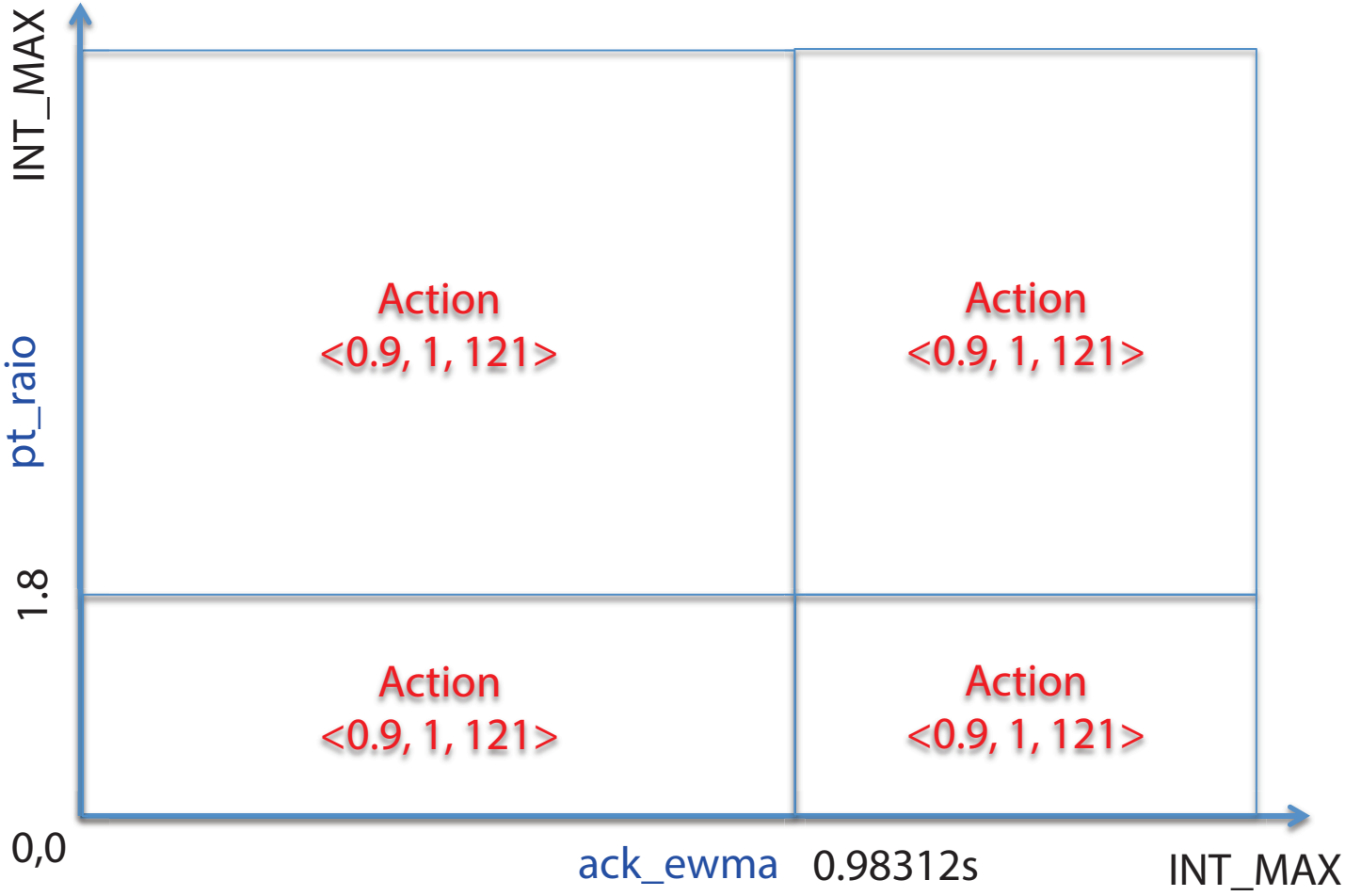
Try different values of <m, b, τ> with the workload



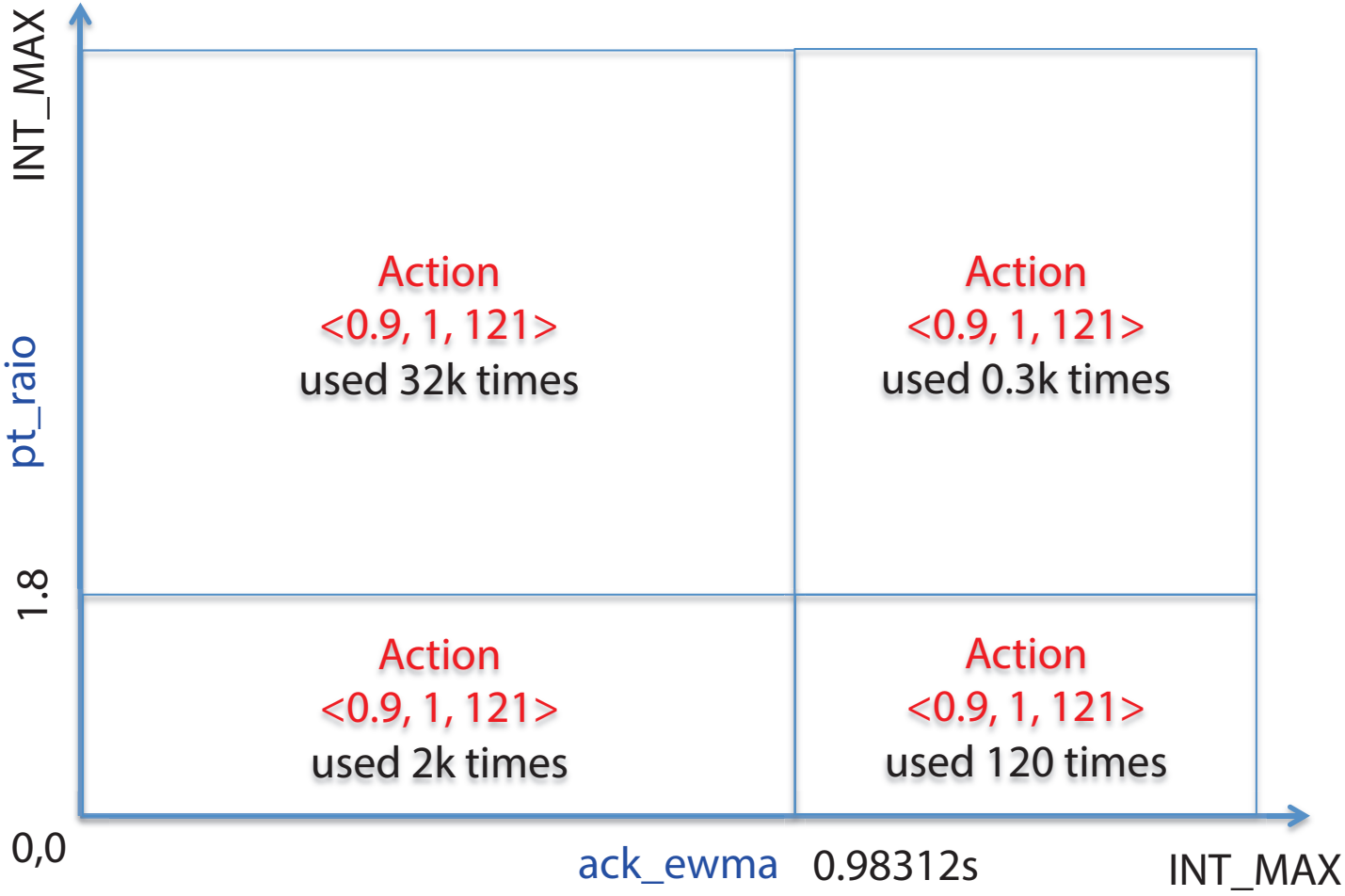
Find the rule that yields highest performance



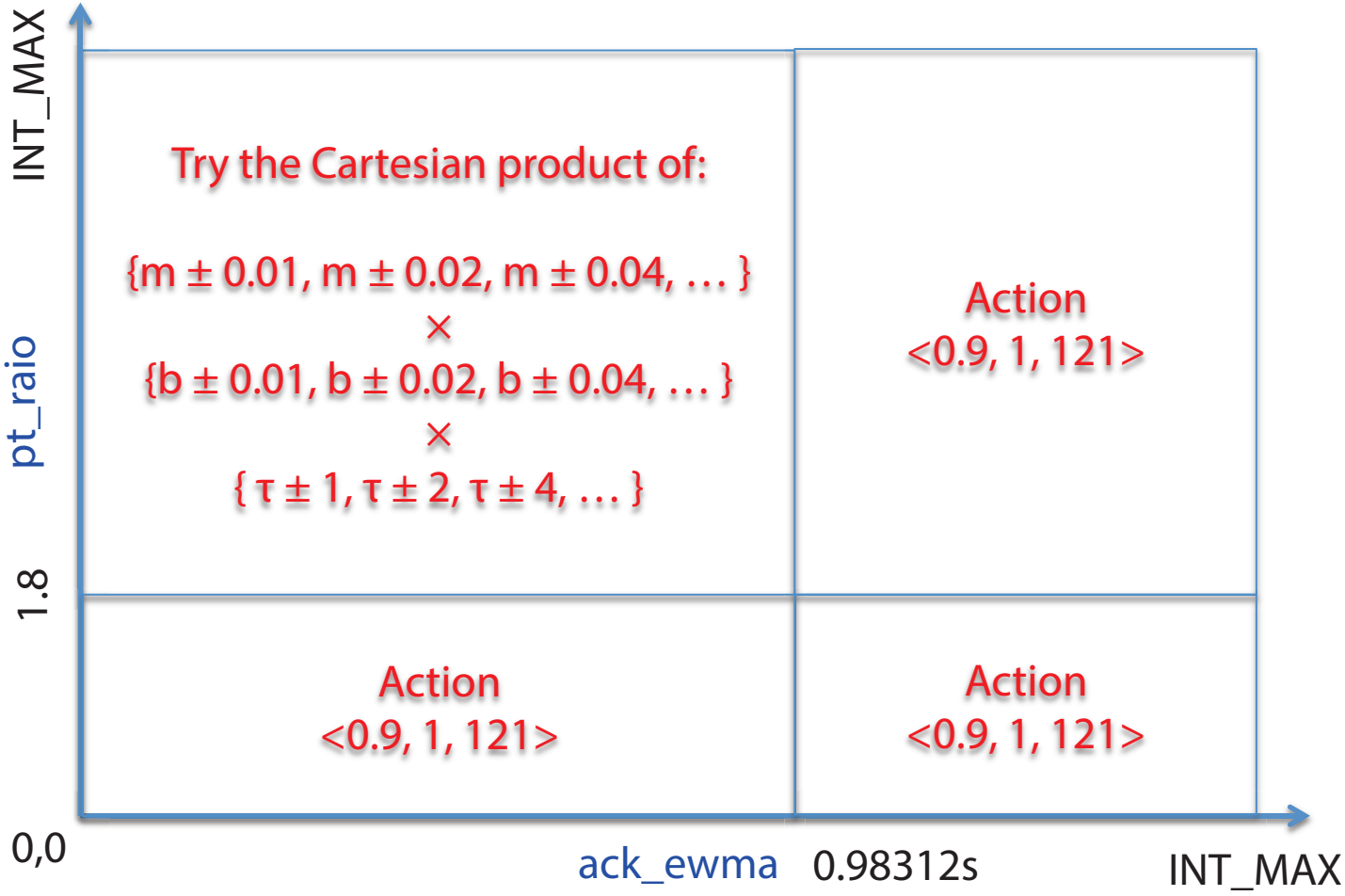
Split the state space at the most observed state values



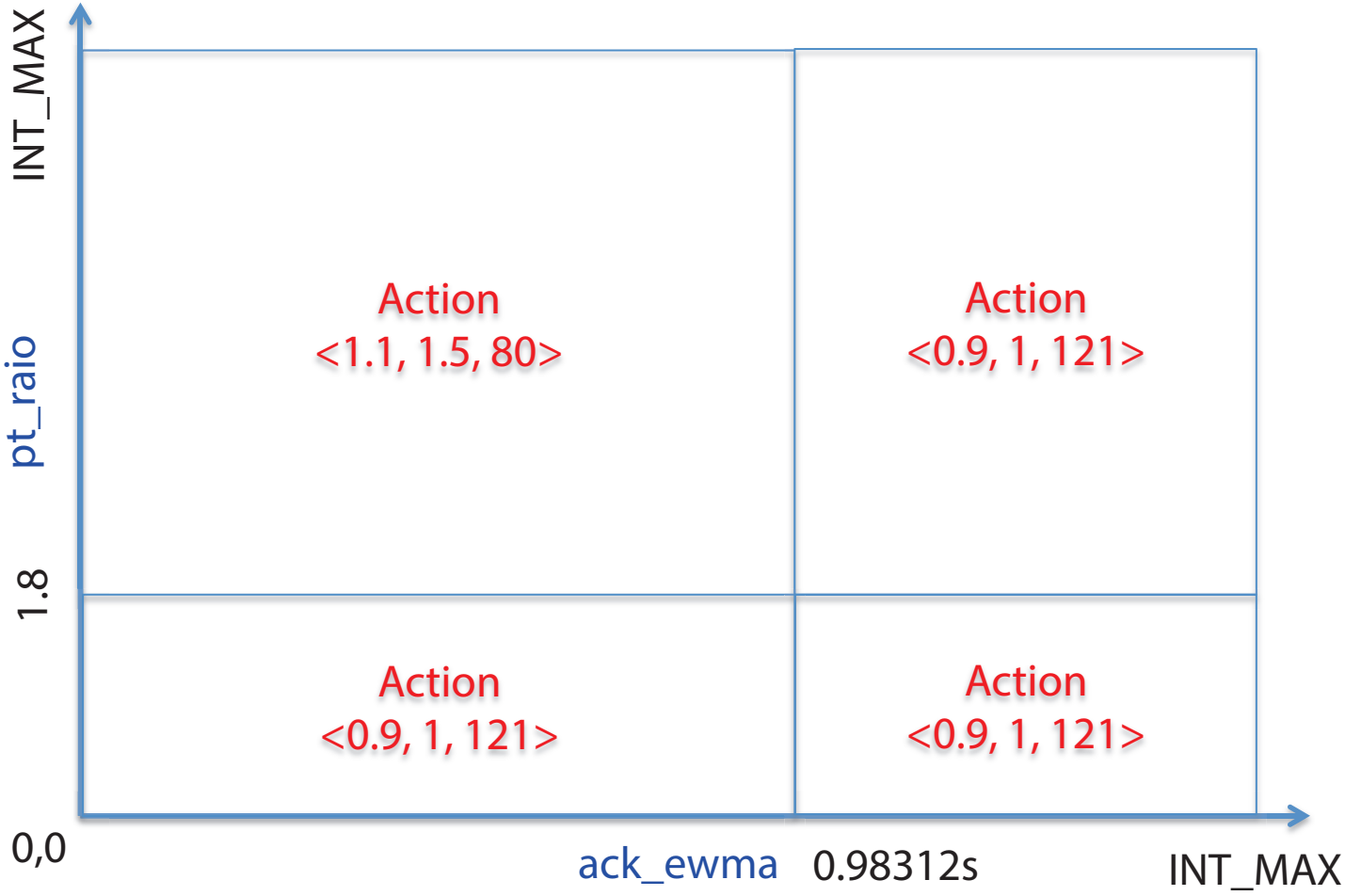
Run the workload, find out the rules that was triggered most often



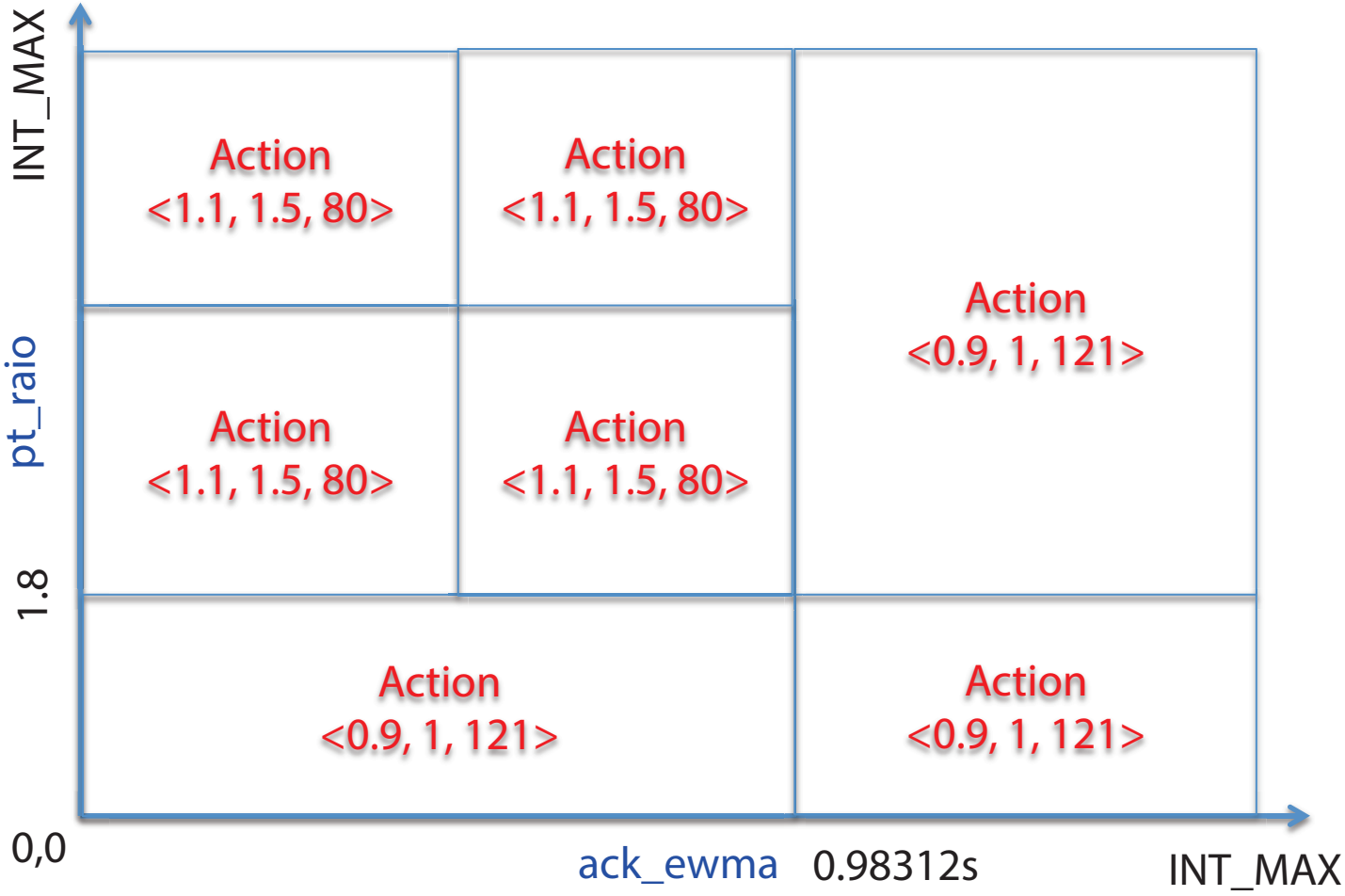
Improve the most used rules by sweeping all possible values of <m, b, τ>



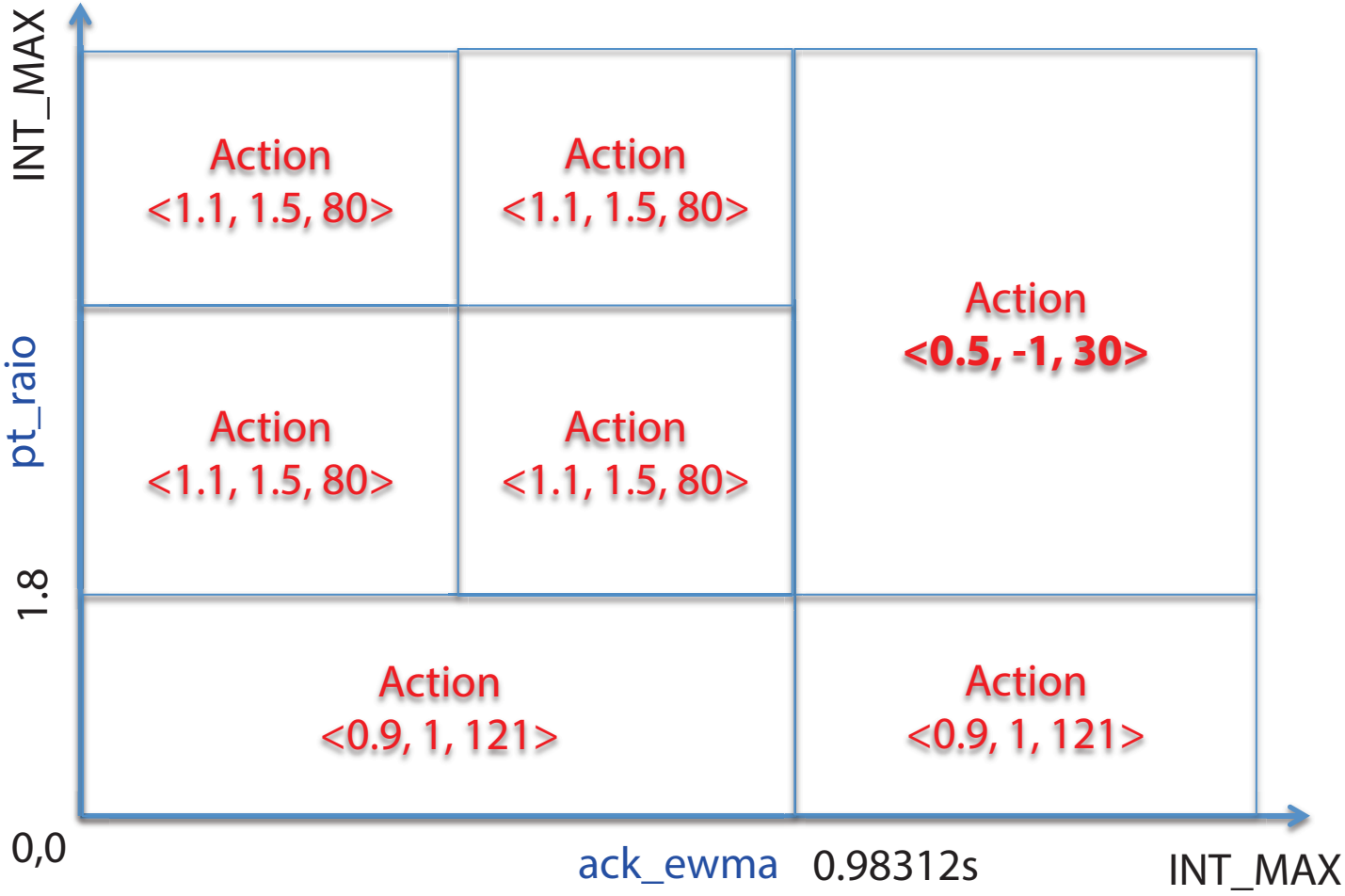
Find the action that works best



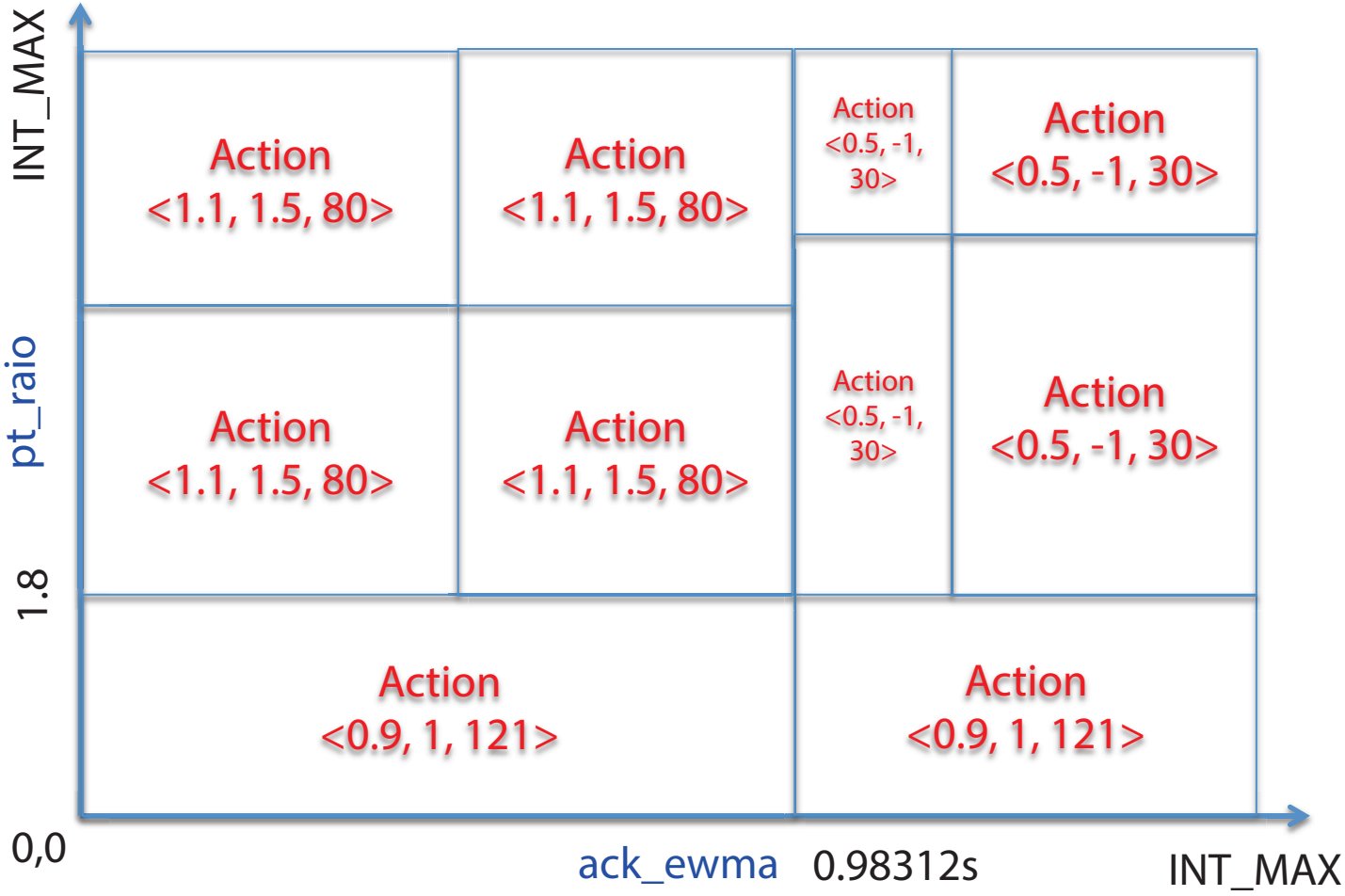
Split the most used rule's state space at the most observed state values



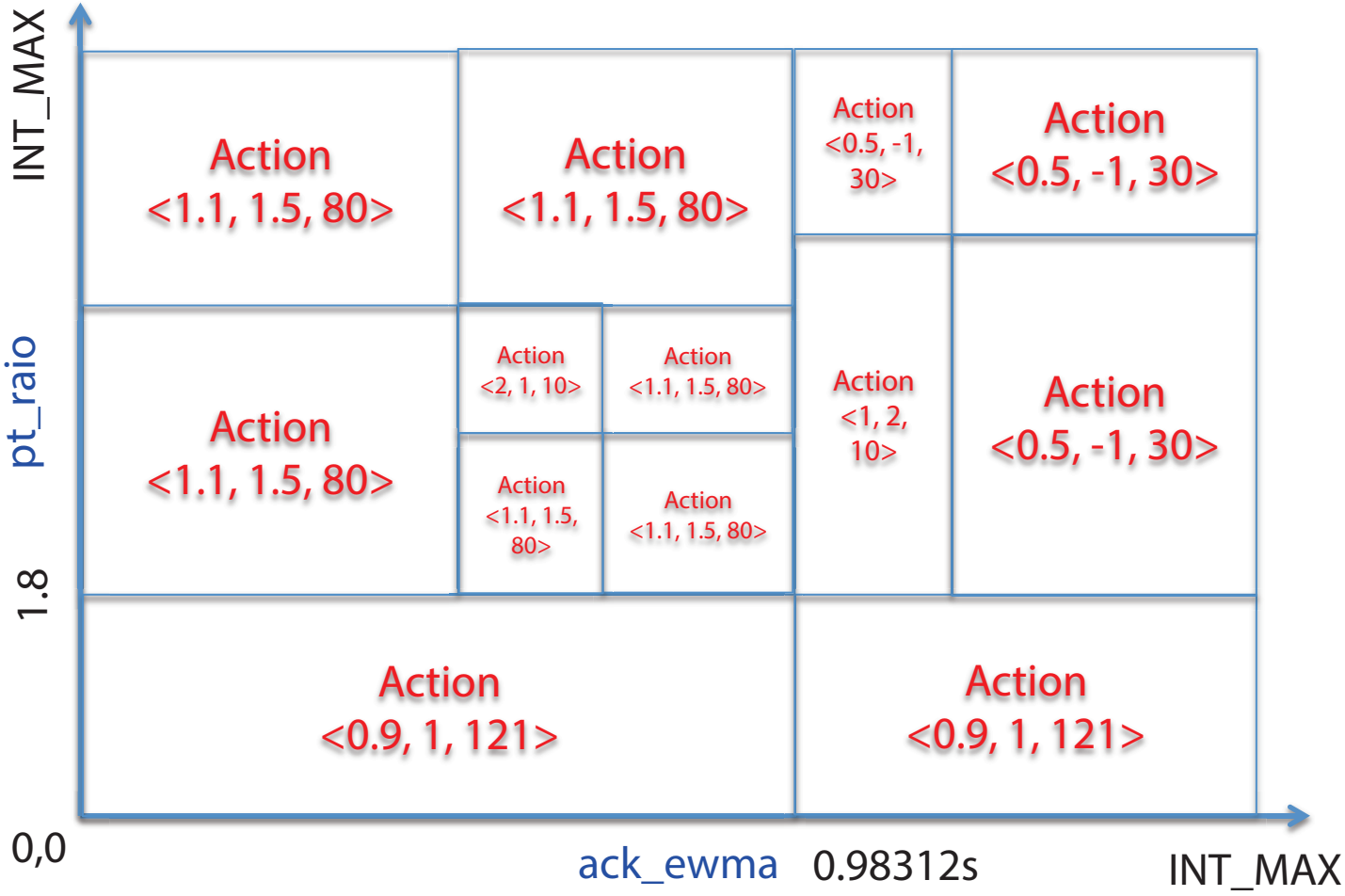
Run workload, find the most used rule, and improve it



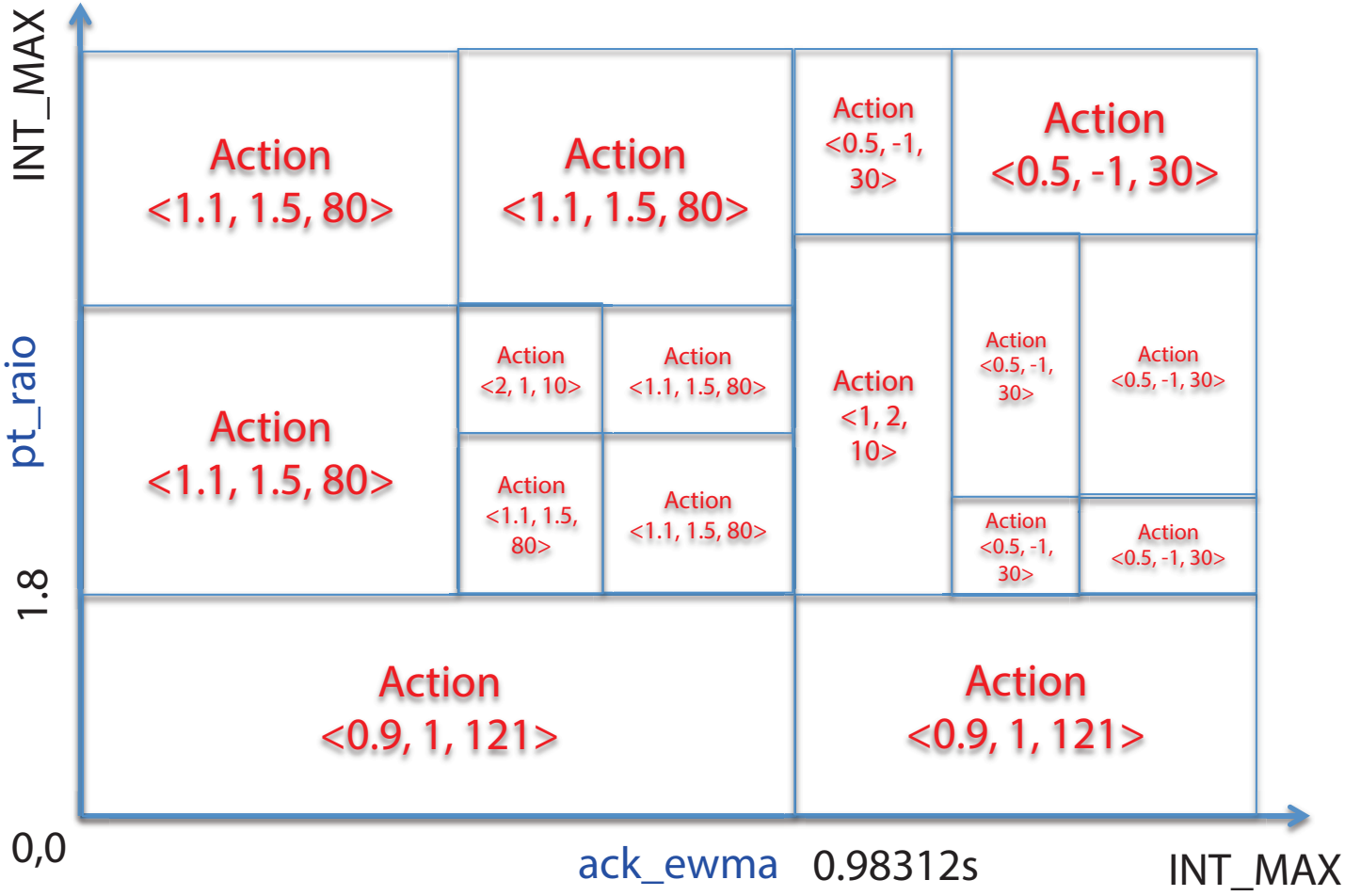
After find the best action, split the most used rule



Repeat this process



Until either we are satisfied with the results or failed to find a good rule



Prototype and Evaluation

An ASCAR prototype works with Lustre

Lustre is a popular high-performance file system used in HPC

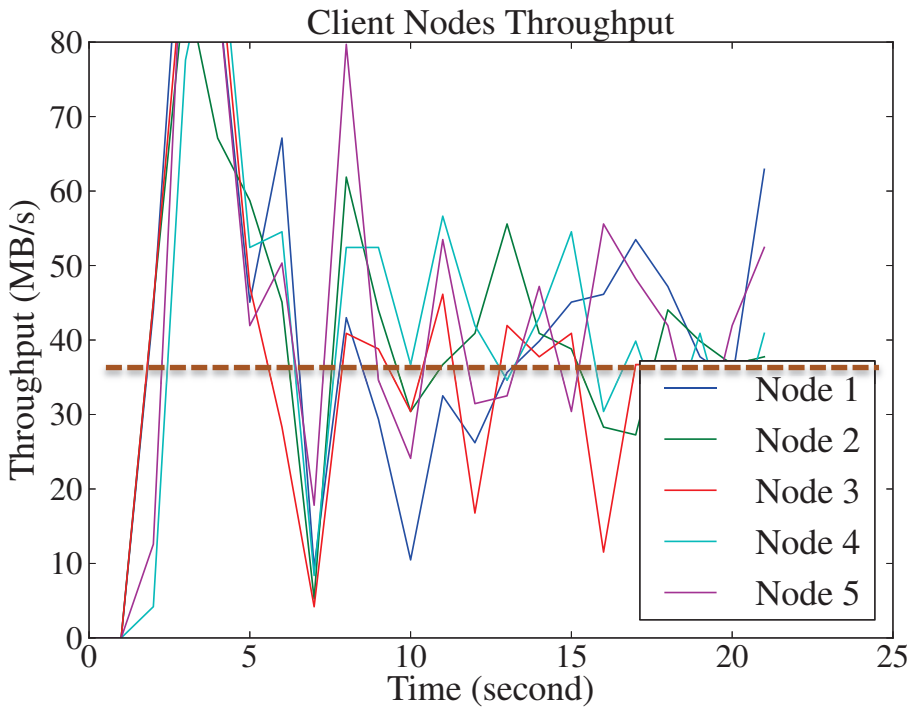
Client controller works within the Lustre client

Highly responsive and low overhead

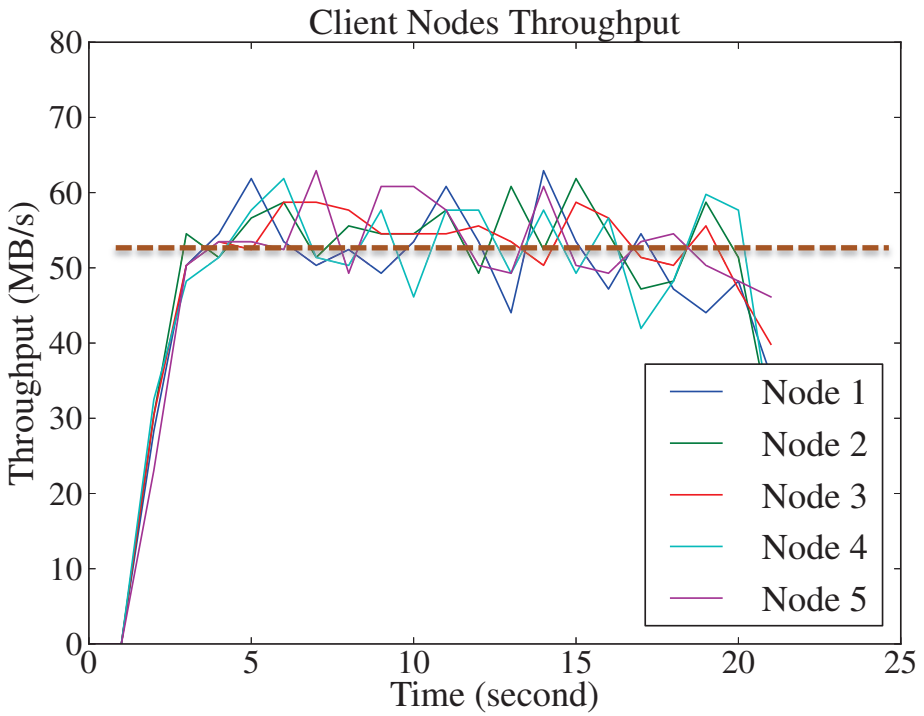
Hardware: 5 servers, 5 clients

Intel Xeon CPU E3-1230 V2 @ 3.30GHz, 16 GB RAM,
Intel 330 SSD for the OS,
dedicated 7200 RPM HGST Travelstar Z7K500 hard drive for Lustre,
Gigabit Ethernet

ASCAR is good at increasing throughout and decreasing speed variance

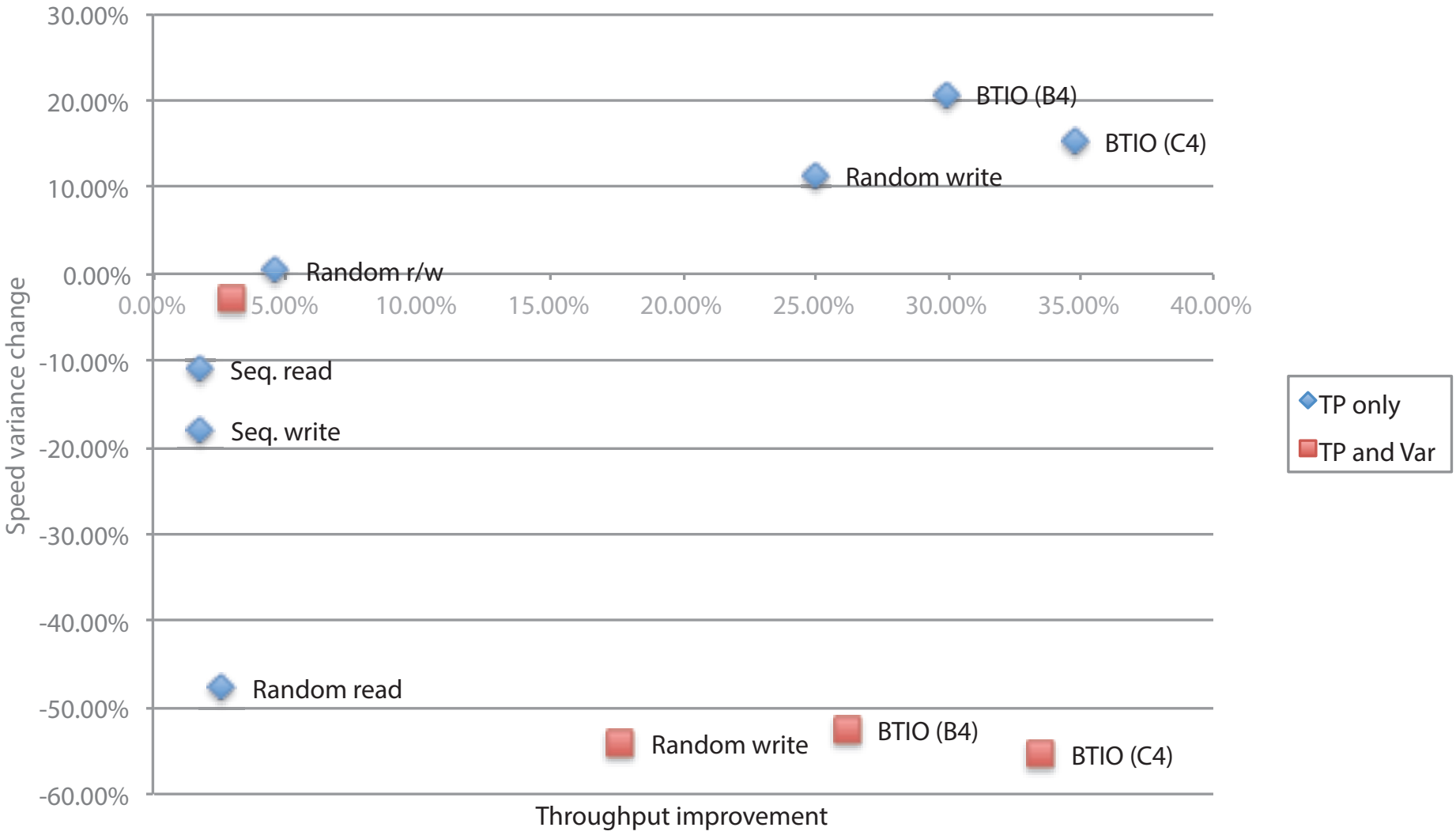


Without ASCAR

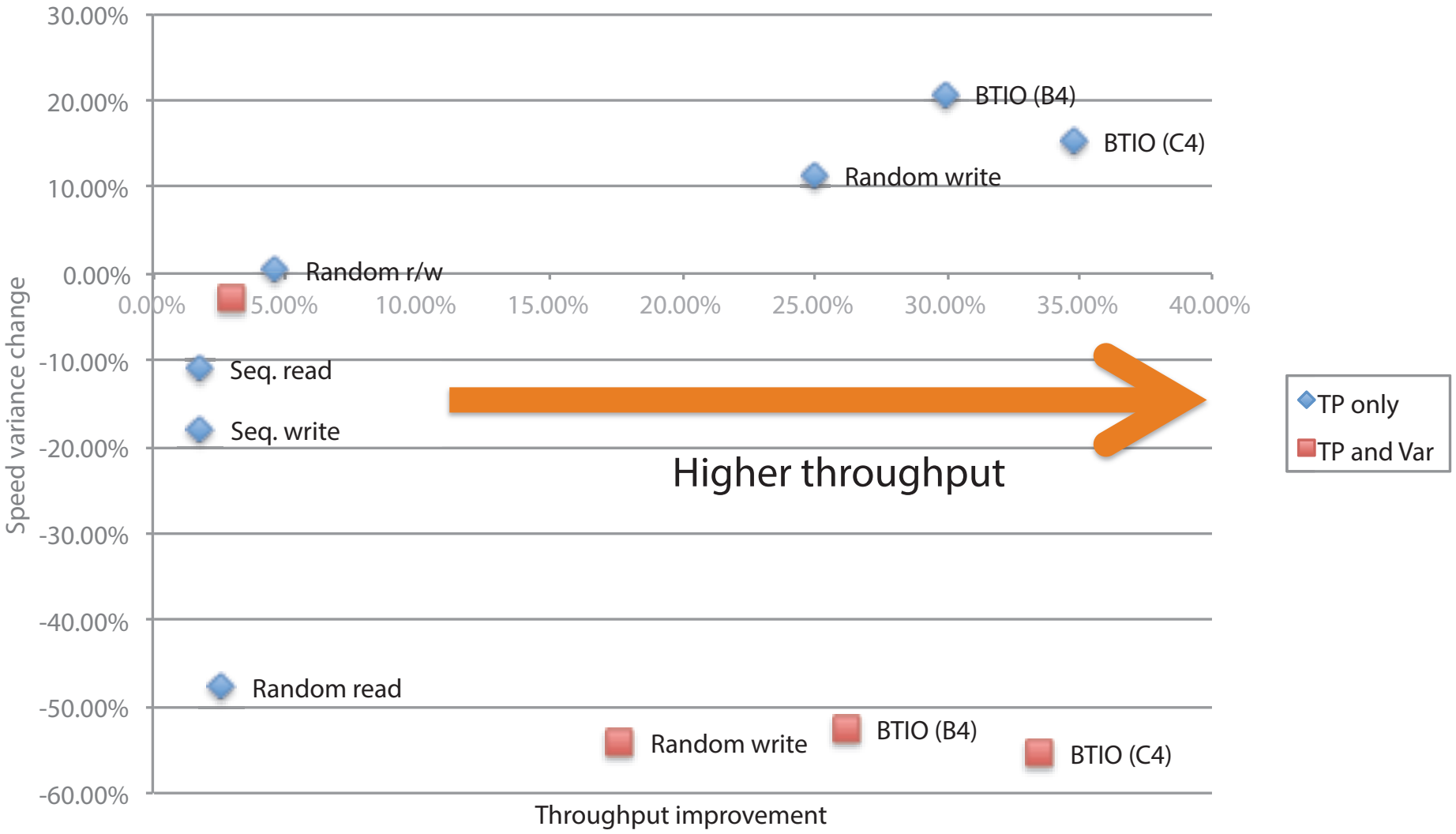


With ASCAR

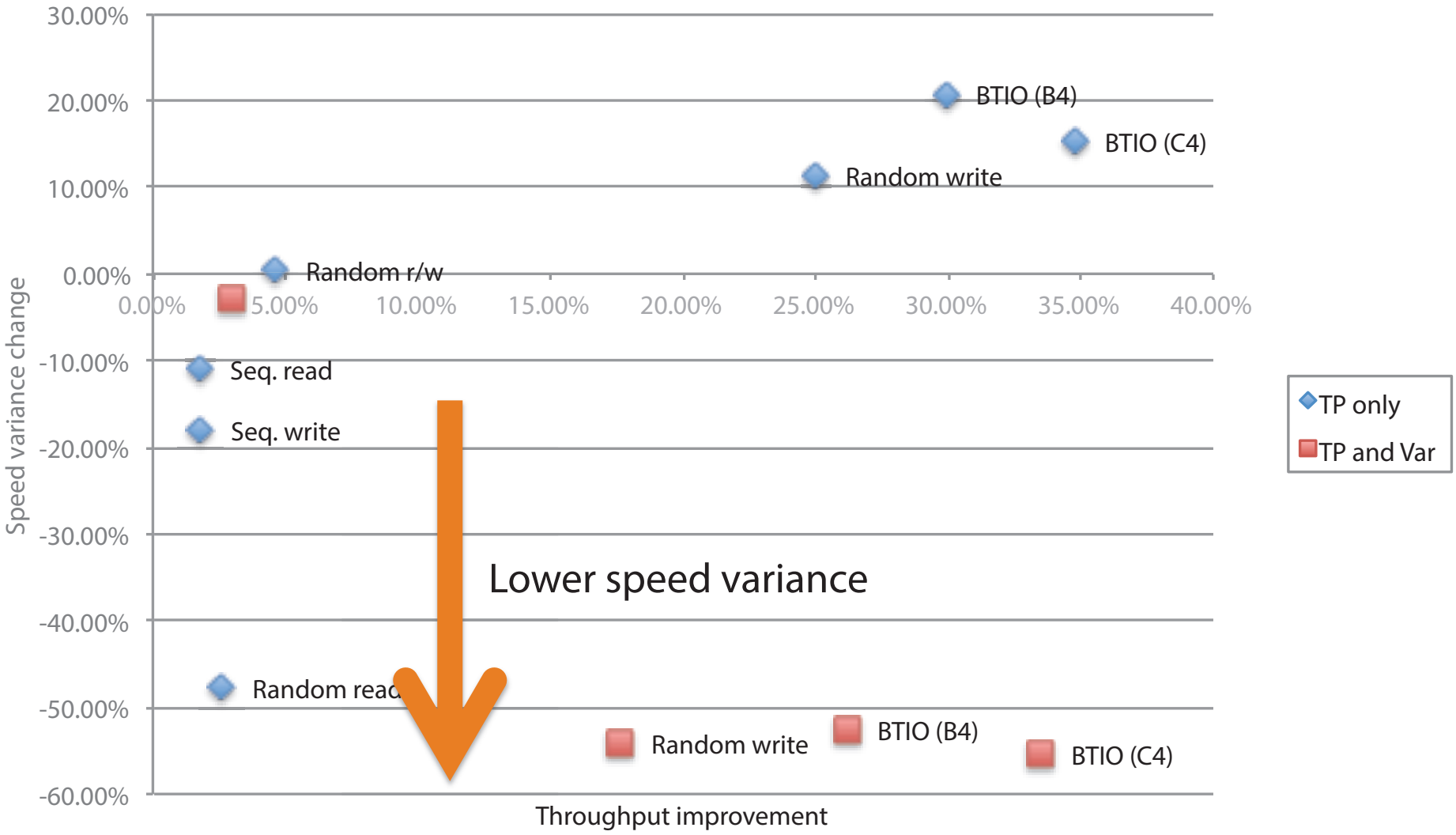
Workload Throughput Improvements



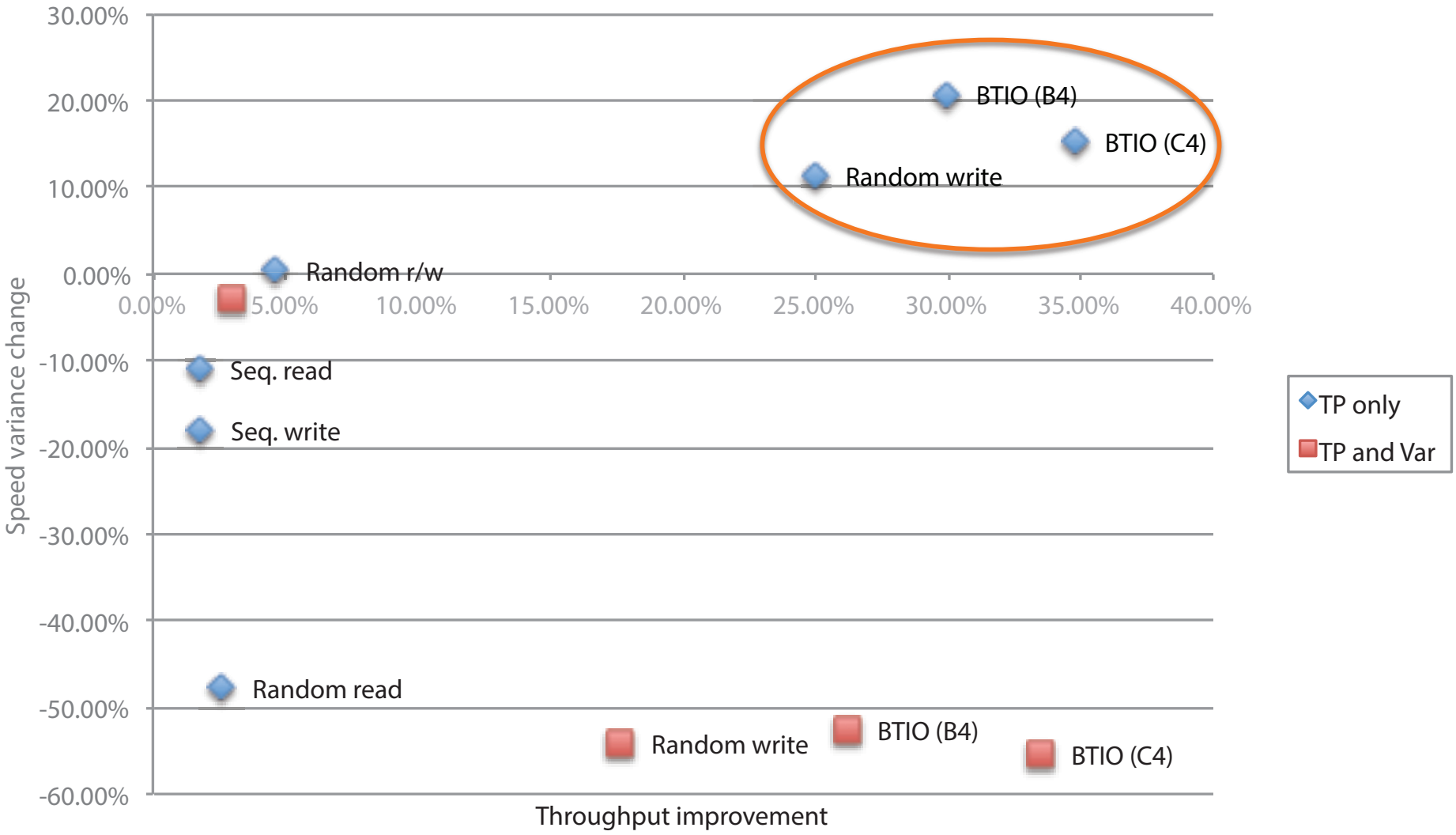
Workload Throughput Improvements



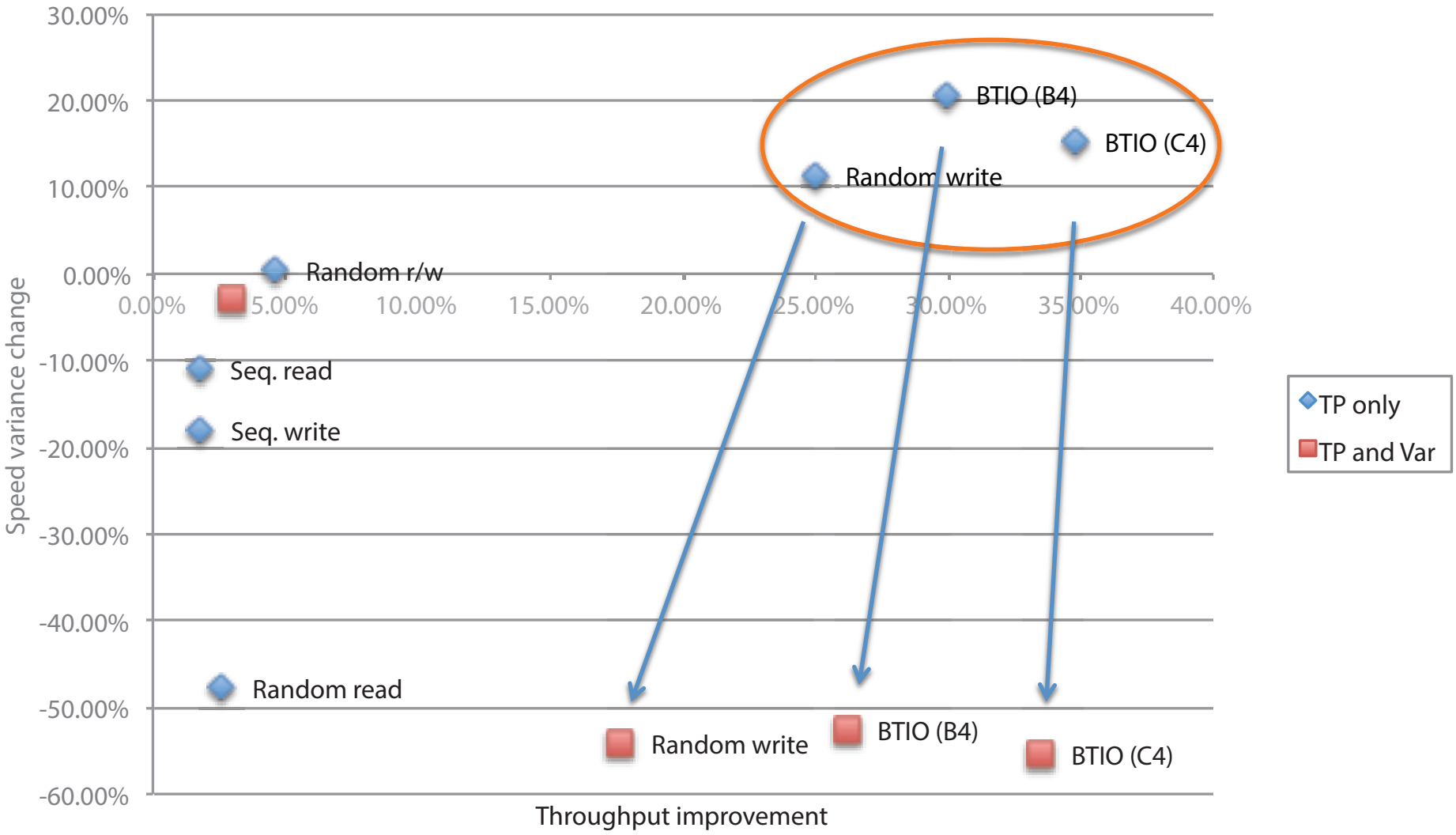
Workload Throughput Improvements



Workload Throughput Improvements



Workload Throughput Improvements



Our prototype shows that …

ASCAR increases the performance of all workloads

2% to 36%

ASCAR works best to boost write-heavy workloads

25% to 36%

and it can lower speed variance at the same time

by more than 50%

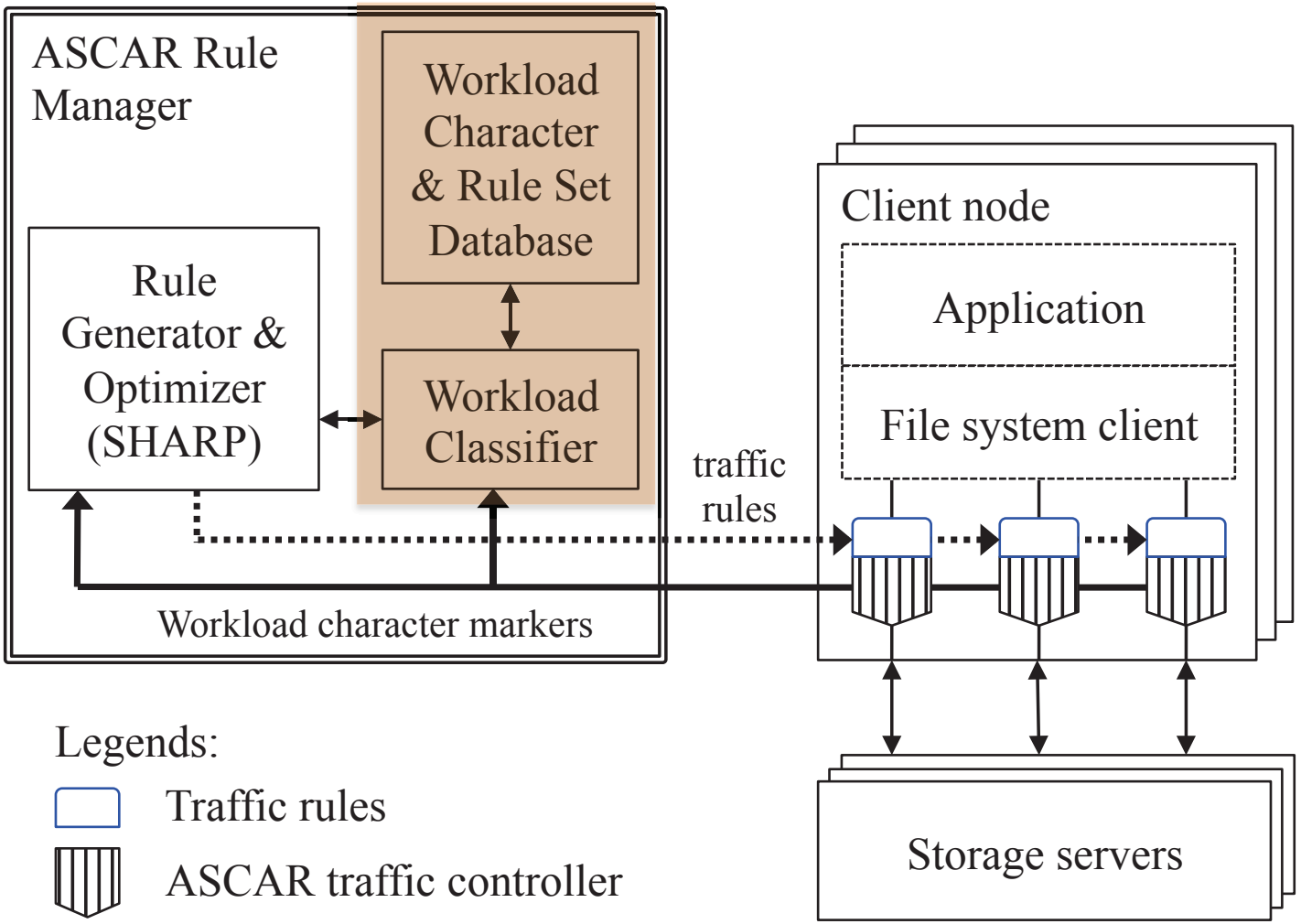
Handling a new workload without offline training

Measure the workload's features

Compare features with known workloads

Find the most similar known workload and use its contention control rules

Components of the ASCAR prototype



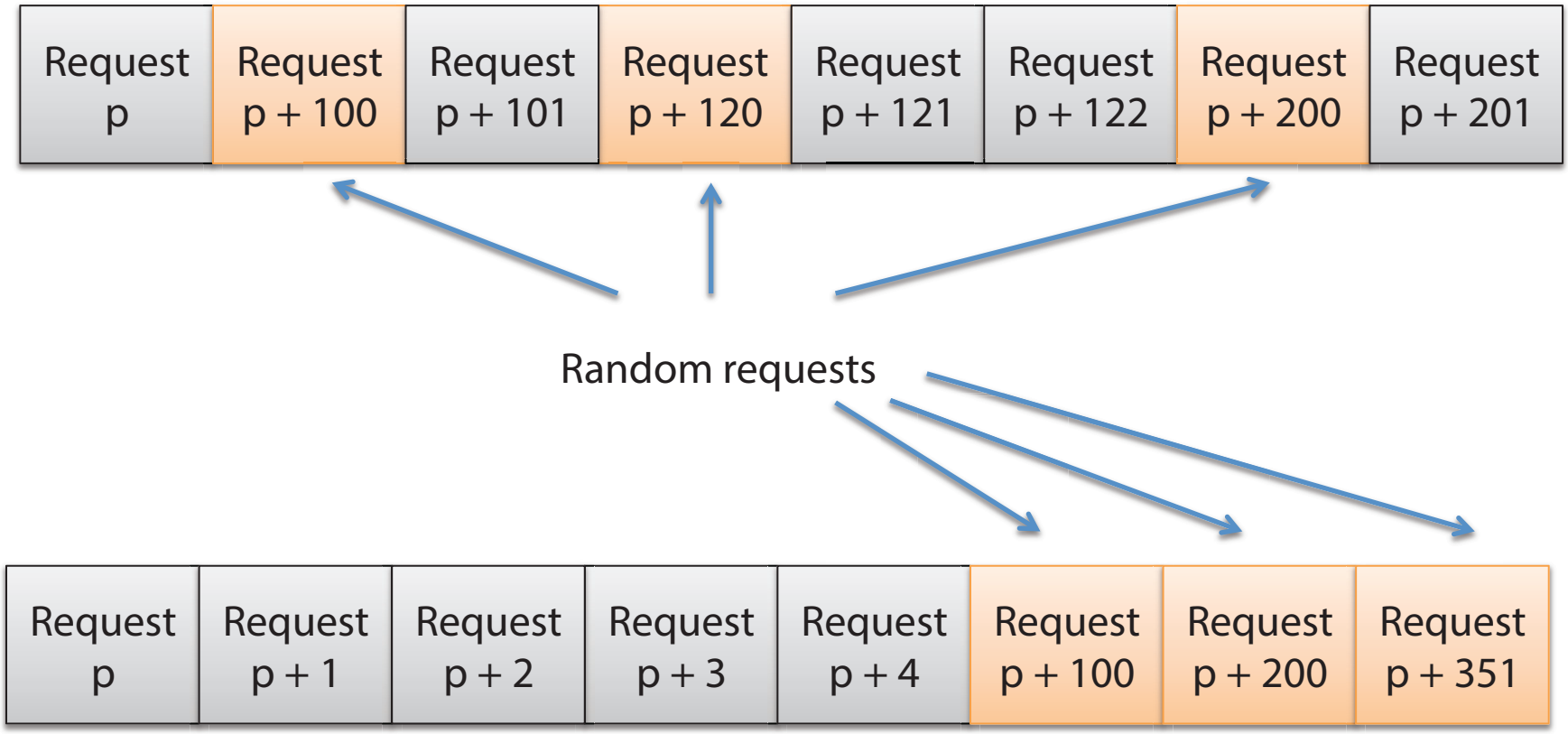
Measuring workloads' pressure on the system

Workloads can have radically different pressure on the underlying system

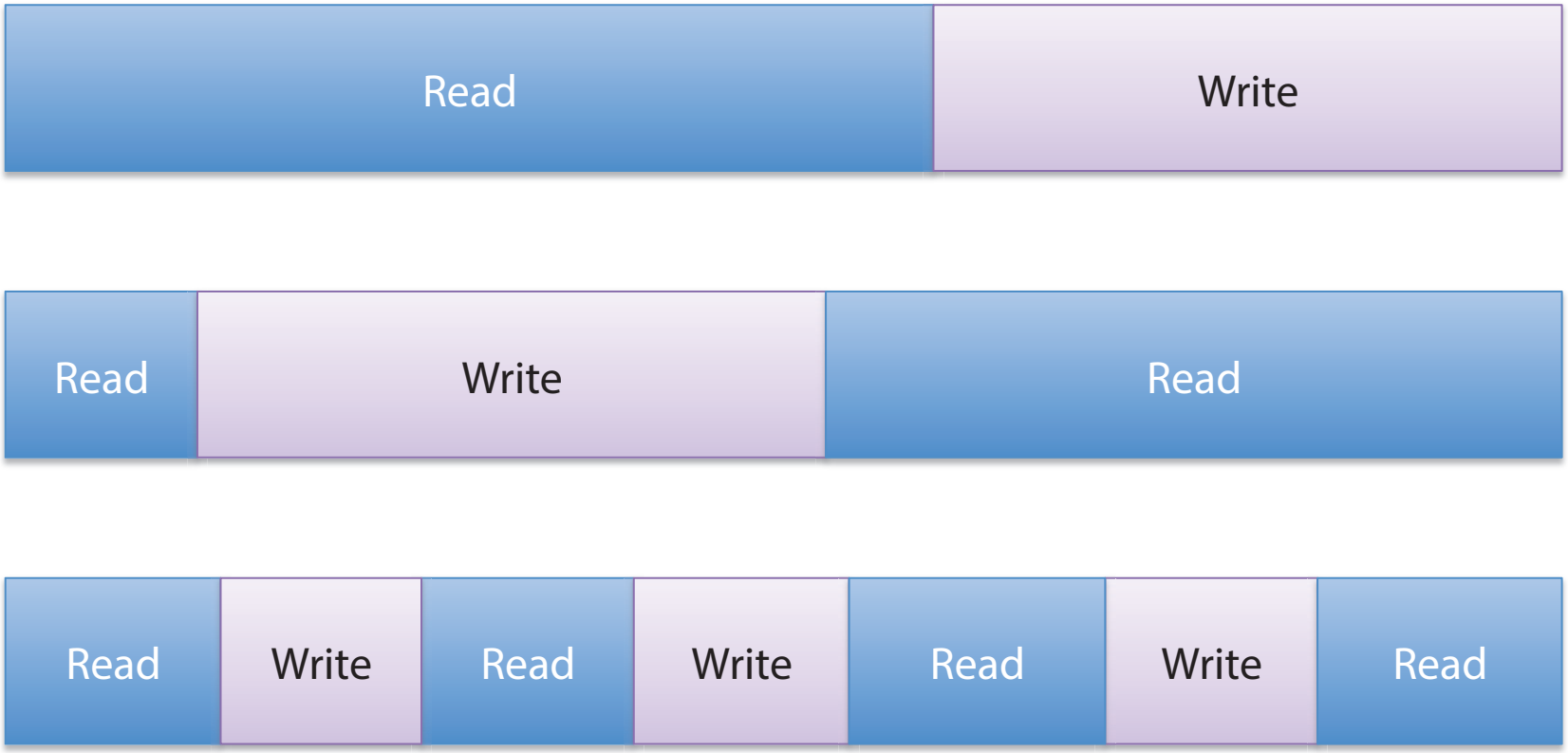
And they require different contention management rules

The combined workload from many applications is a mix of read/write and random/sequential

A 75% sequential + 25% random workload can be very different from another



And there are many different 60% read + 40% write workloads out there



The feature set we are using now

Op type: read/write/metadata

Ratios between ops (read to write, read/write to metadata, etc.)

For each type of op, we measure the following features:

1. average size of sequential ops
2. location between seq. ops: random or sequential
3. average temporal gaps between seq. ops

Sample:
Different 60% read + 40% write workloads

Read to write: 60/40
Avg. size of sequential read: 60 MB
Avg. size of sequential write: 40 MB

Read to write: 60/40
Avg. size of sequential read: 15 MB
Avg. size of sequential write: 13 MB

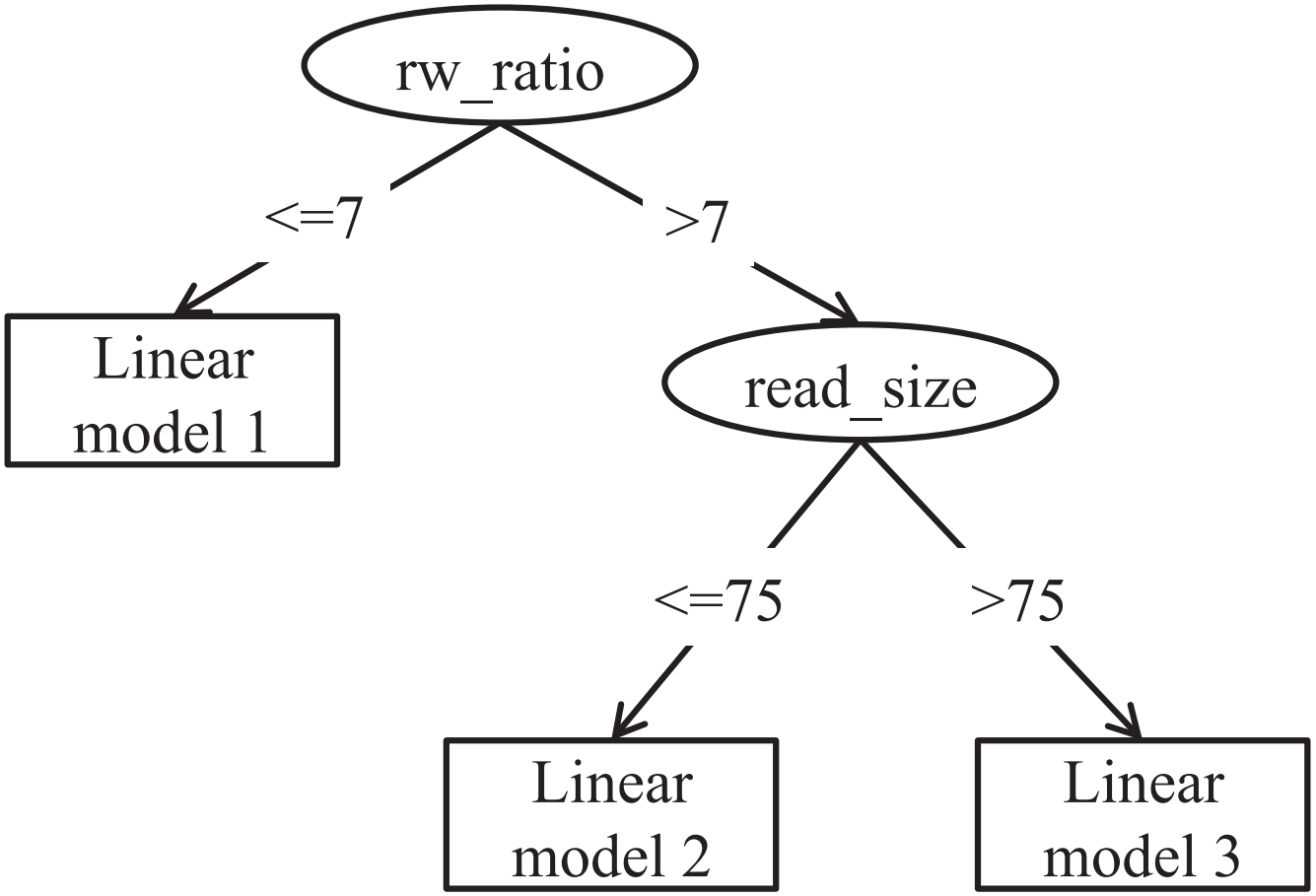
Calculating similarity of workloads

Goal: to determine if they can use the same contention control rules

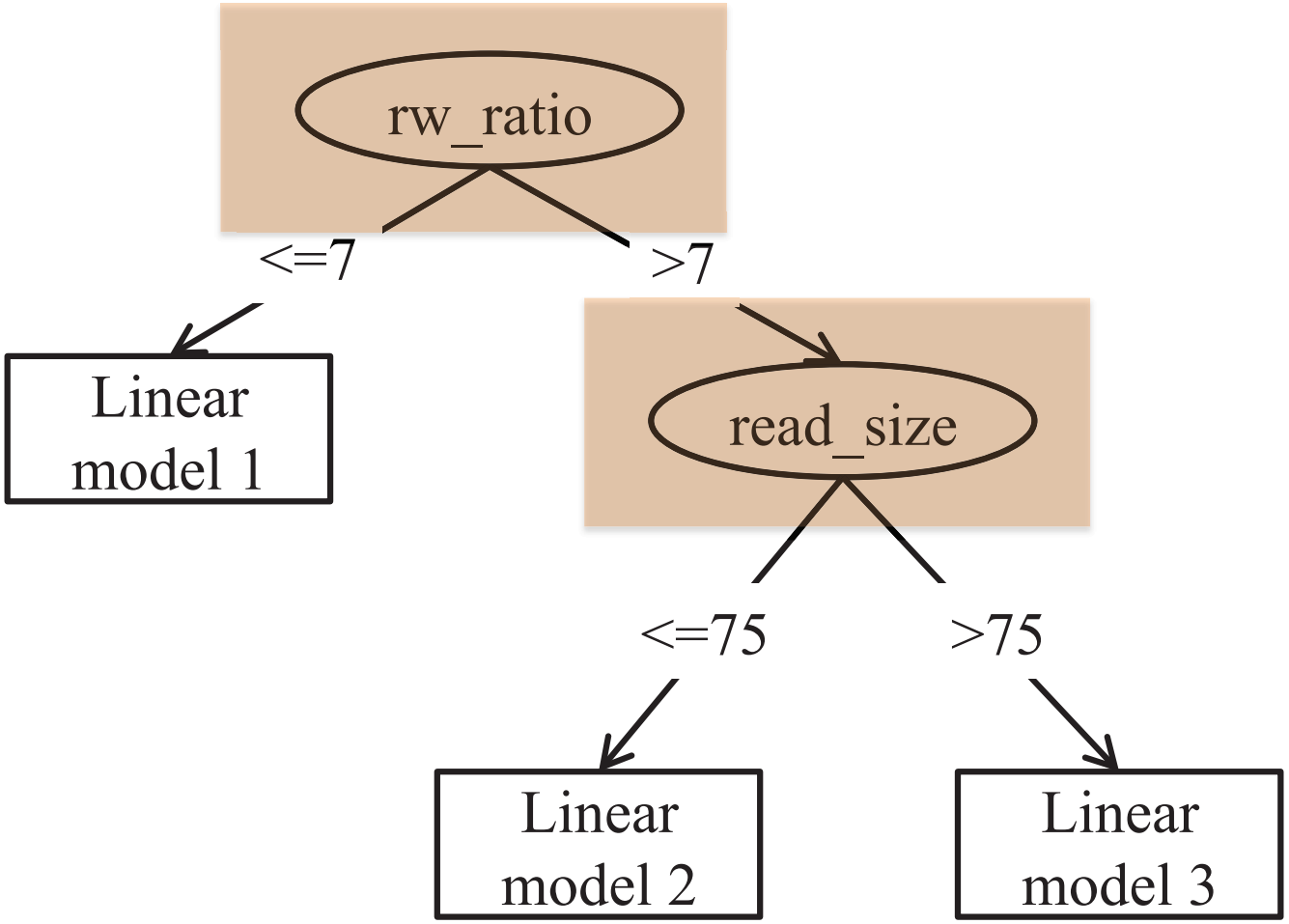
How: start from a known workload and tweak its features, measuring the efficiency of existing rules on the changed workload

Result: we used the results of hundreds of benchmarks to generate a decision tree

Calculating similarity of workloads



Calculating similarity of workloads



This decision tree can precisely predict the performance of using a specific rule set with a new workload

Correlation coefficient: 0.8676

Mean absolute error: 0.038

Root mean squared error: 0.0462

Conclusion: ASCAR …

Does not require knowledge of the system or workloads

fully unsupervised

Only needs client-side controllers

no need to change hardware/application/network/server software

Can improve highly changeable workloads

like burst I/O

Work-conserving

no wasted bandwidth

Conclusion: ASCAR …

we distilled feature set and method for calculating the similarity between workload in terms of applying contention control rules

source code of our prototype is published as an open source project

http://www.ssrc.ucsc.edu/ascar.html

Future work: online rule optimization

Current ASCAR prototype requires a lengthy offline learning process

Use cloud computing services for doing evaluation on a larger scale

Online tweaking of rules using random-restart hill climbing

Also need to evaluate the ASCAR algorithm on other workloads: database, web services

Acknowledgments

This research was supported in part by the National Science Foundation under awards IIP-1266400, CCF-1219163, CNS- 1018928, the Department of Energy under award DE-FC02- 10ER26017/DESC0005417, Symantec Graduate Fellowship, and industrial members of the Center for Research in Storage Systems. We would like to thank the sponsors of the Storage Systems Research Center (SSRC), including Avago Technologies, Center for Information Technology Research in the Interest of Society (CITRIS of UC Santa Cruz), Department of Energy/Office of Science, EMC, Hewlett Packard Laboratories, Intel Corporation, National Science Foundation, NetApp, Sandisk, Seagate Technology, Symantec, and Toshiba for their generous support.

Thanks!

ASCAR project:
http://www.ssrc.ucsc.edu/ascar.html

Contact:
Yan Li <yanli@cs.ucsc.edu>