ascar.io

Increasing Performance Through Automated Contention Management

(Luster Developers Day '16)

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

Baskin Engineering UC SANTA CRUZ

SSRC STORAGE SYSTEMS RESEARCH CENTER

CENTER FOR RESEARCH IN STORAGE SYSTEMS

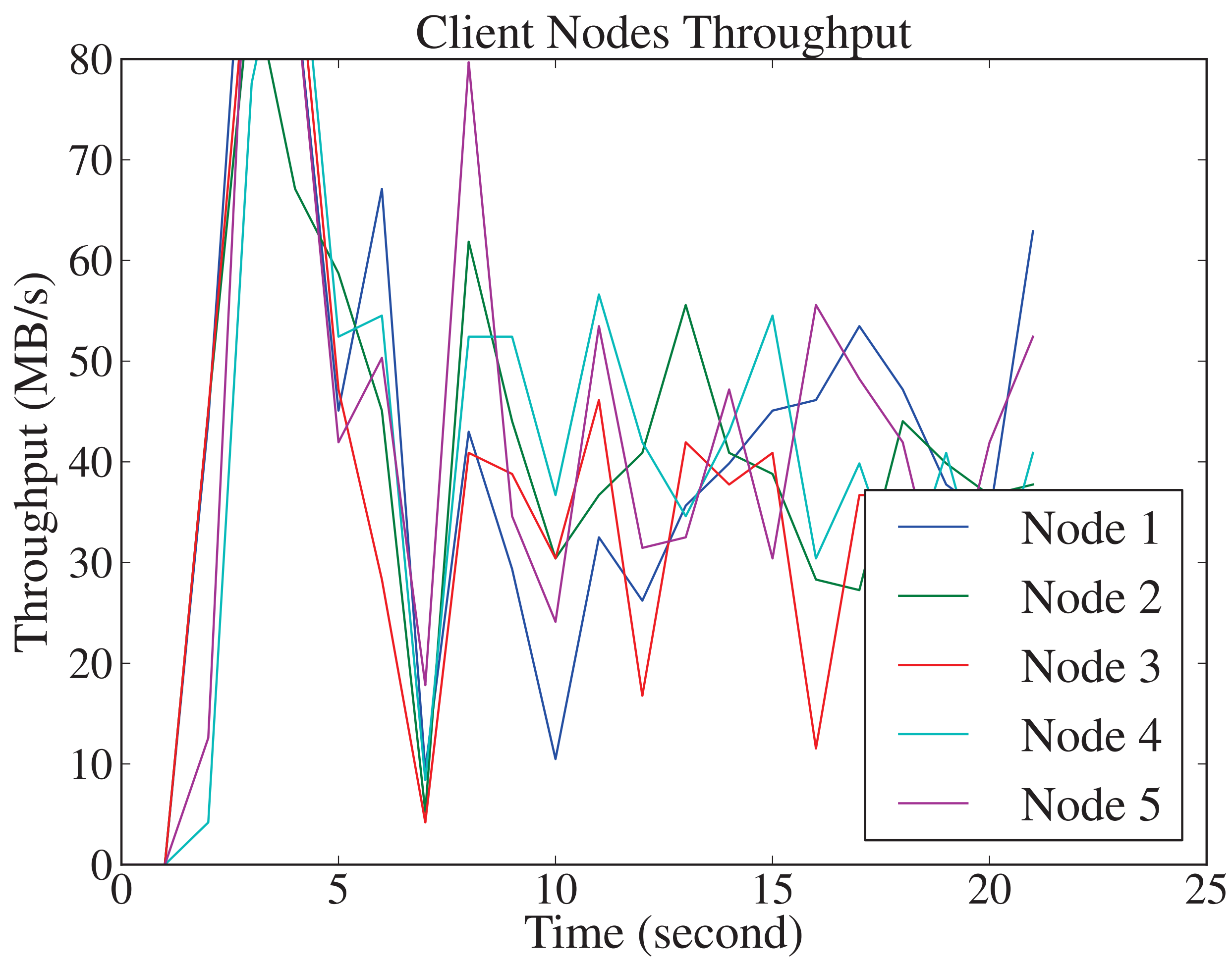
Challenge: consistent performance at peak times

congestion harms *efficiency* and *throughput*

ascar.io
ssrc

Challenge: consistent performance at peak times

congestion causes *fluctuation*



client throughput of a random write workload

5 nodes accessing 5 servers

The problem we are trying to solve

Improve throughput or fairness during congestion

or both at the same time!

End-to-end coverage

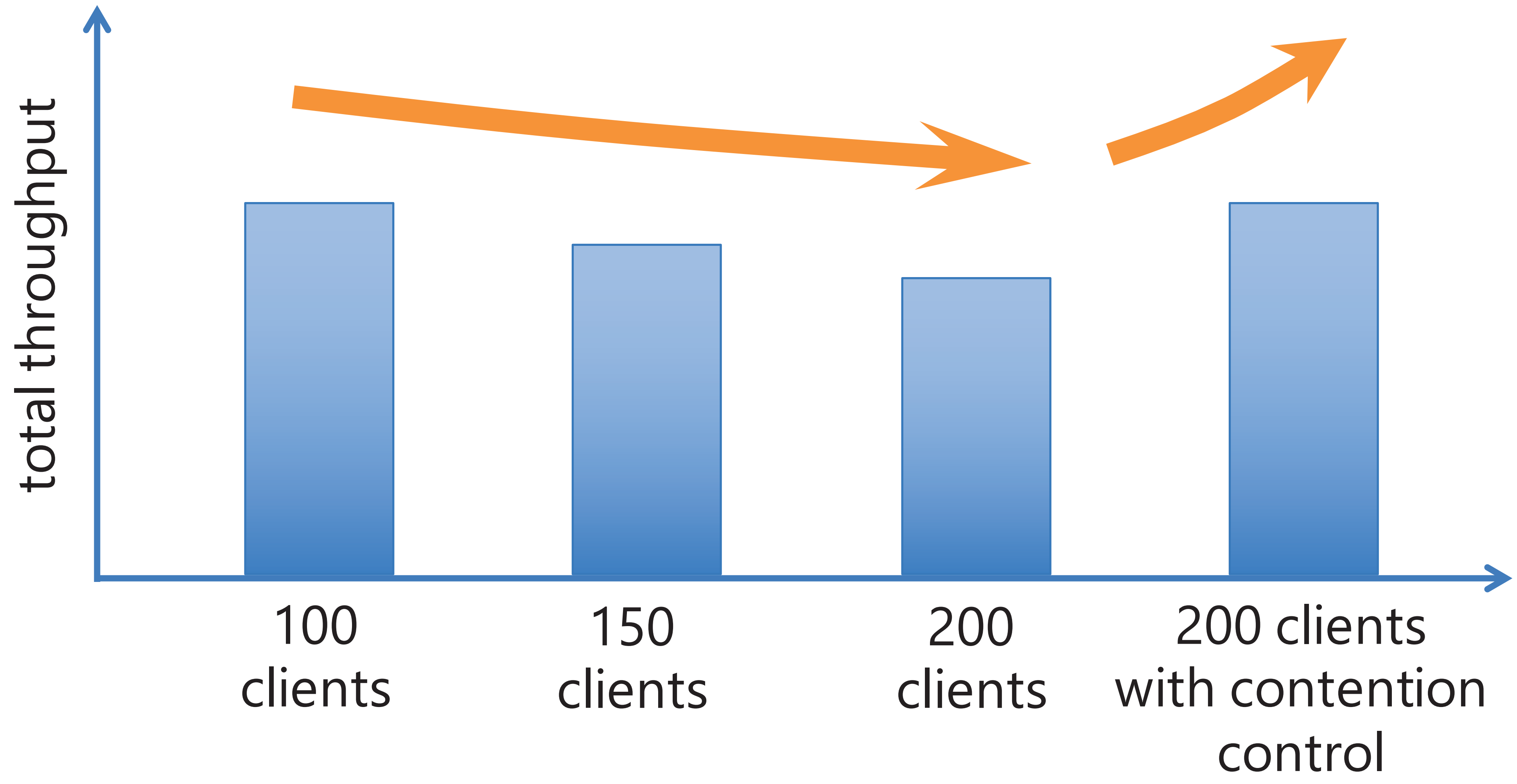
handling congestion at OSC, network, OSS, and OST

Fully automatic and requires little human effort

modern systems are very dynamic, and we won't have time to create models

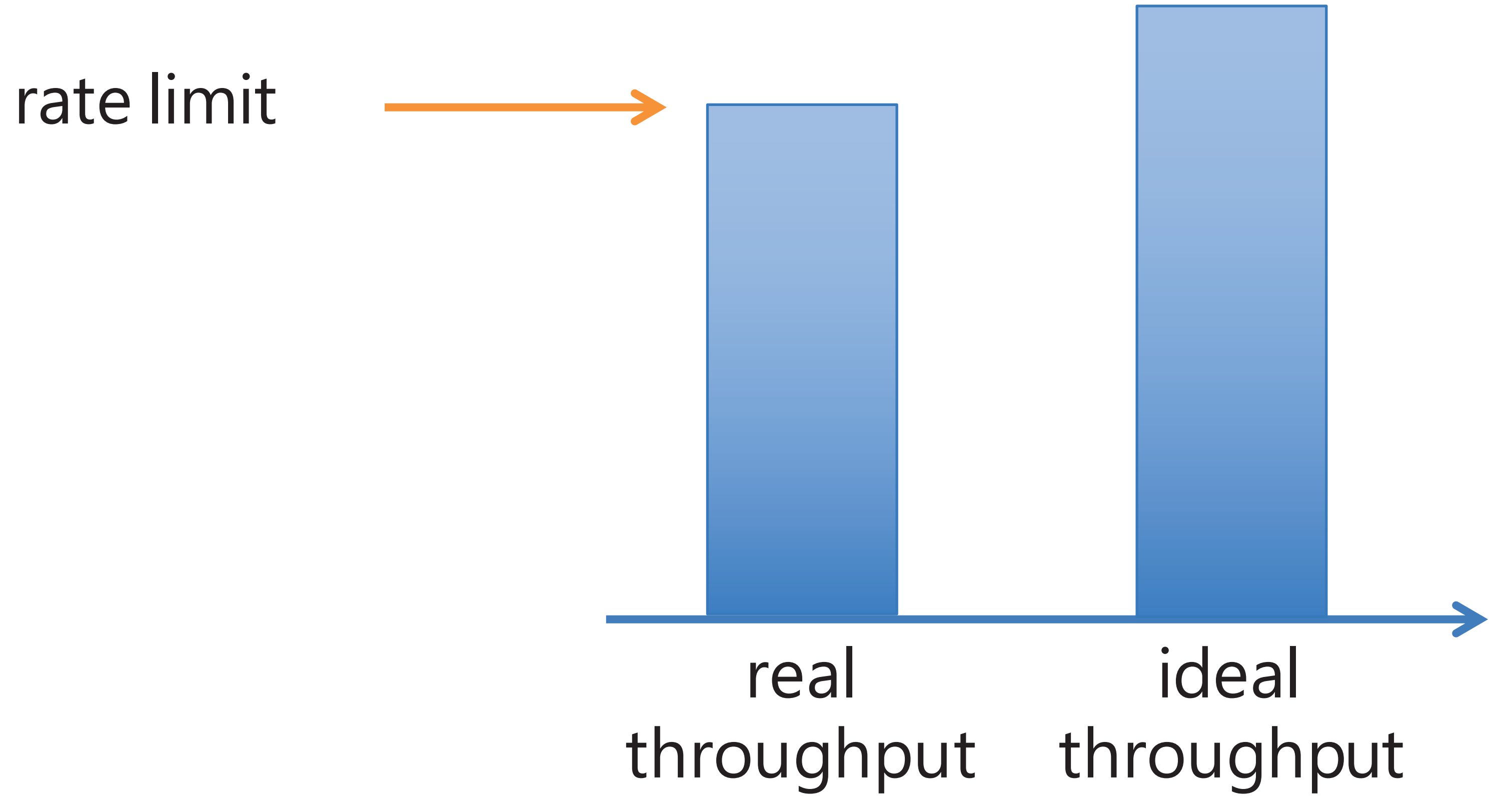
Rate limiting can improve performance

… if done properly

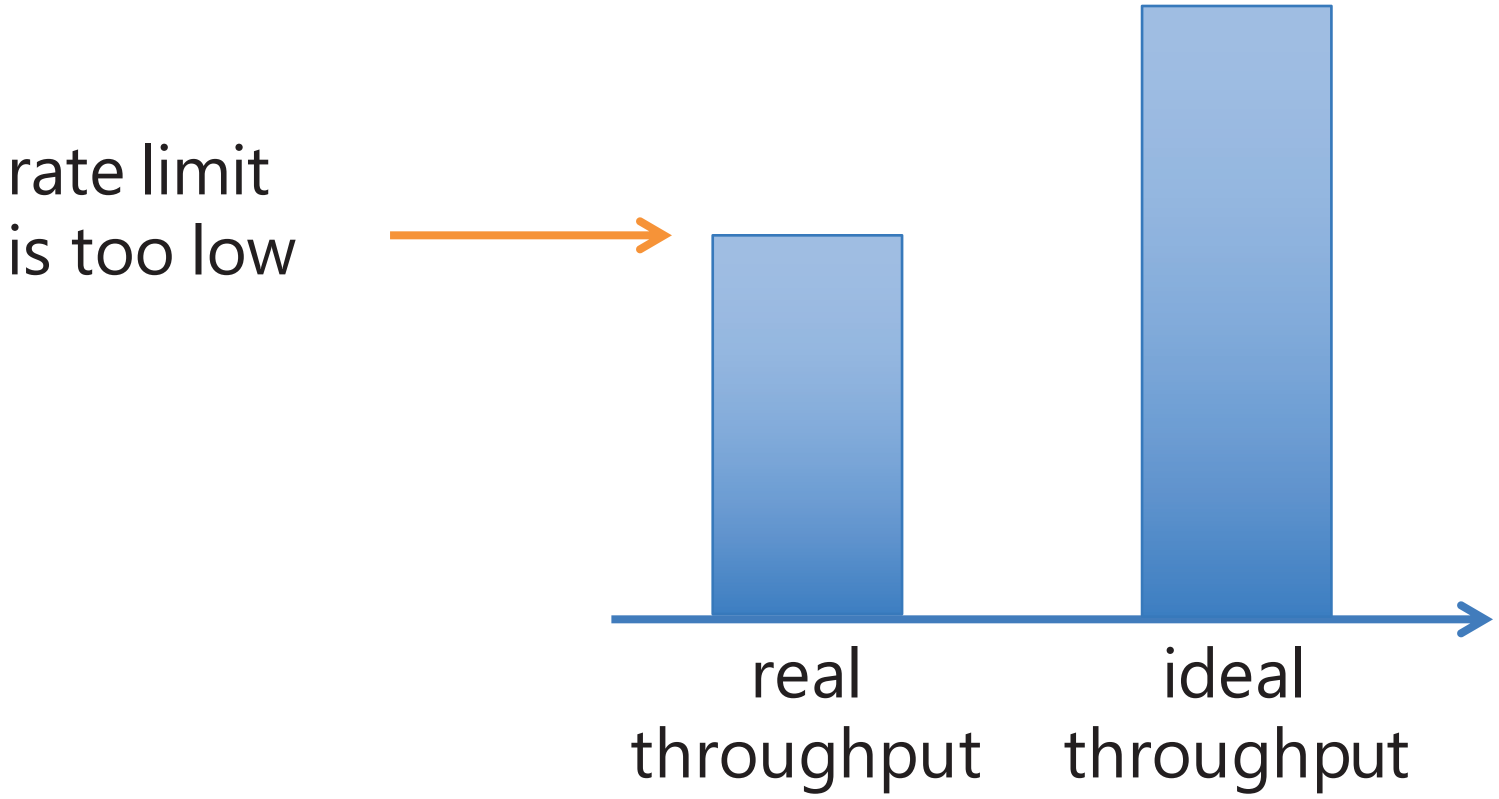


ascar.io
ssrc

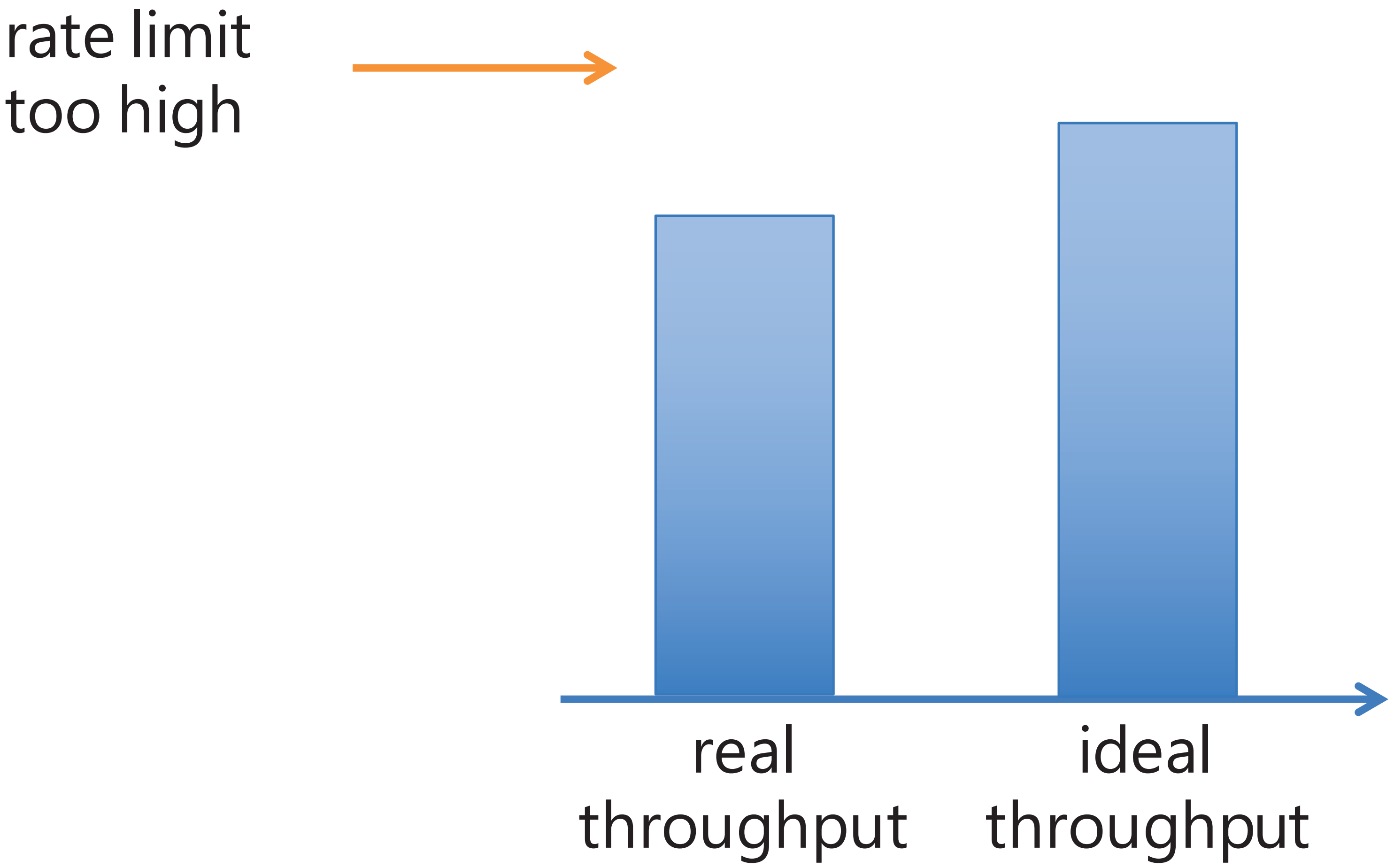
Challenges of distributed I/O rate control:
1. Where is the *sweet spot*?



Challenges of distributed I/O rate control:
1. Where is the *sweet spot*?



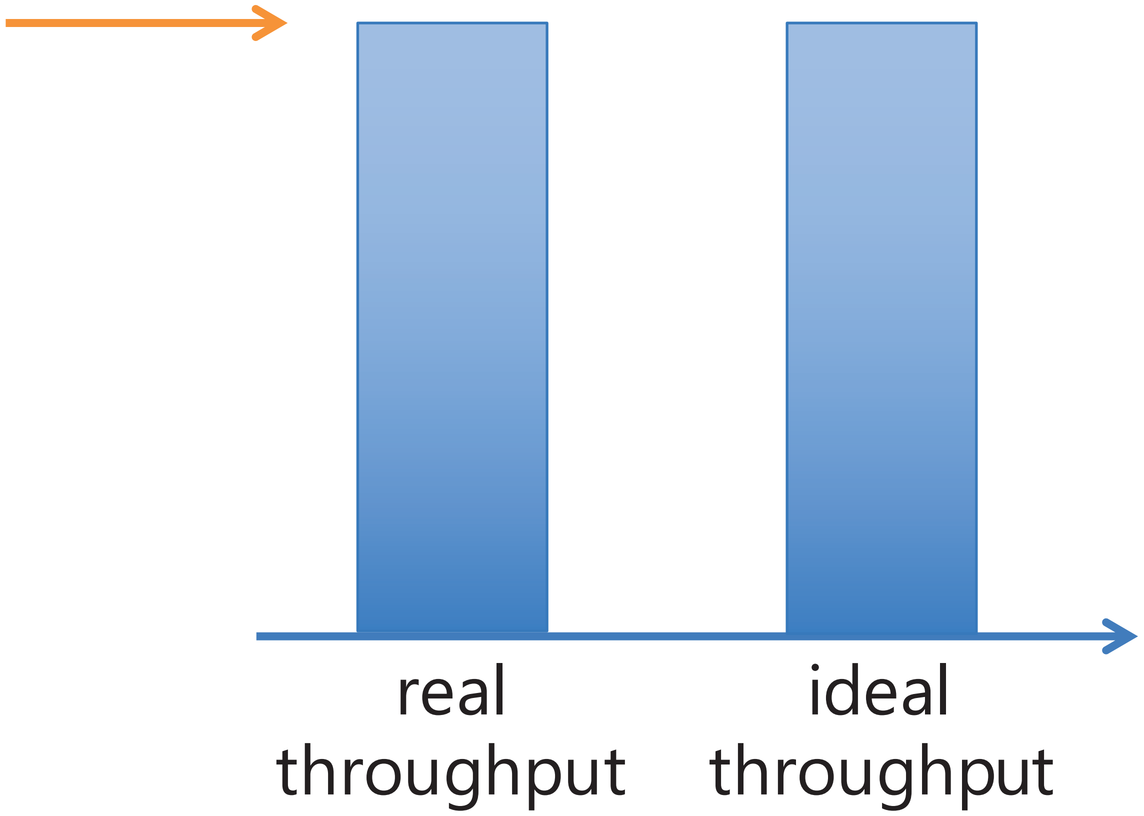
Challenges of distributed I/O rate control: 1. Where is the *sweet spot*?



Created by Yan Li <yanli@ascar.io> http://ascar.io

Challenges of distributed I/O rate control: 1. Where is the *sweet spot*?

sweet rate limit spot



Capability discovery usually involves communication:

- between clients
- with a central controller

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

Intra-node communication can grow at $O(n^2)$

Adds overhead to already congested network

Low responsiveness for highly dynamic workload

ascar.io
ssrc

Created by Yan Li <yanli@ascar.io> http://ascar.io

ASCAR: Automatic Storage Contention Alleviation and Reduction

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

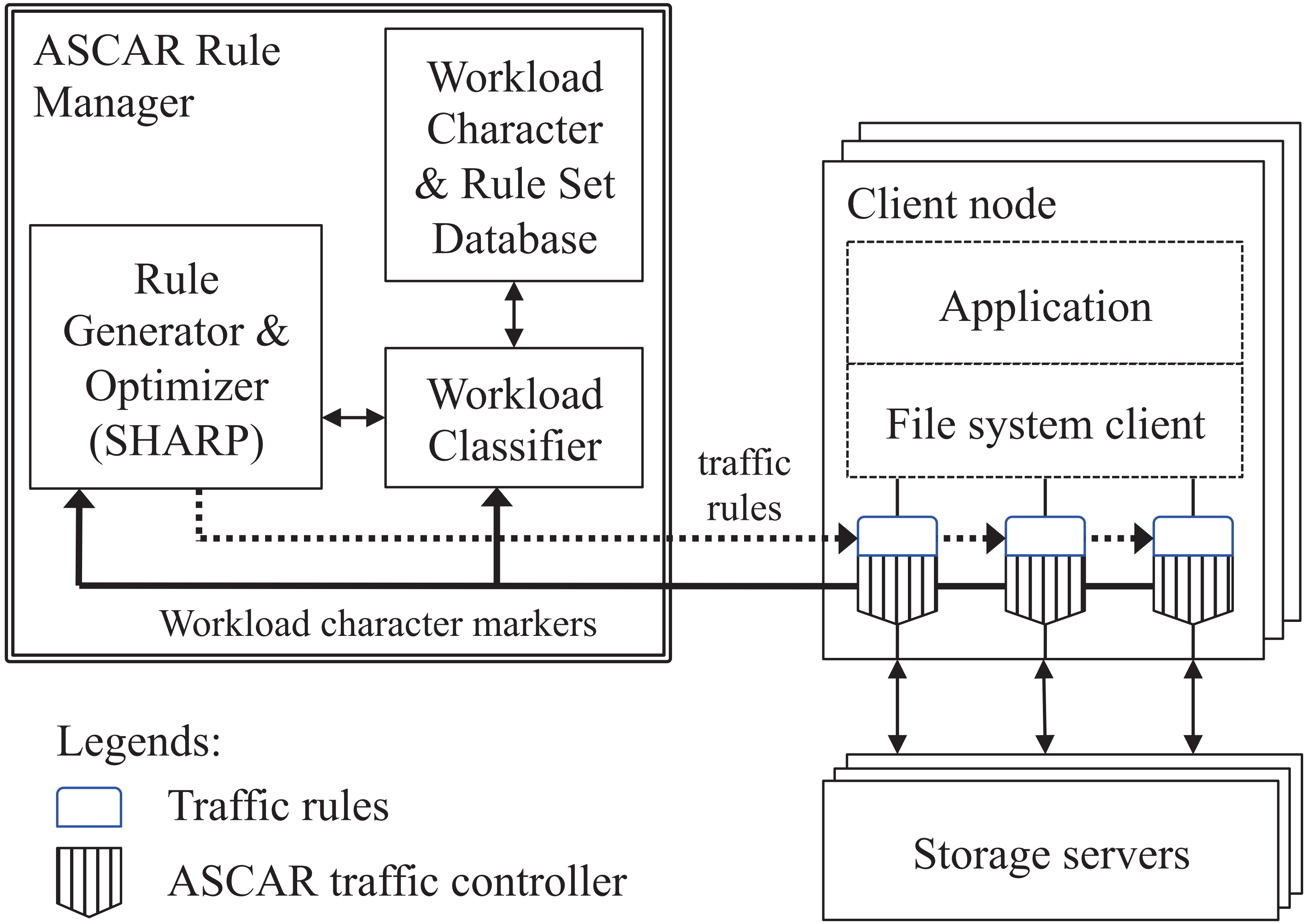
Use machine learning and heuristics for rule generation and optimization

no prior knowledge of the system or workload is required

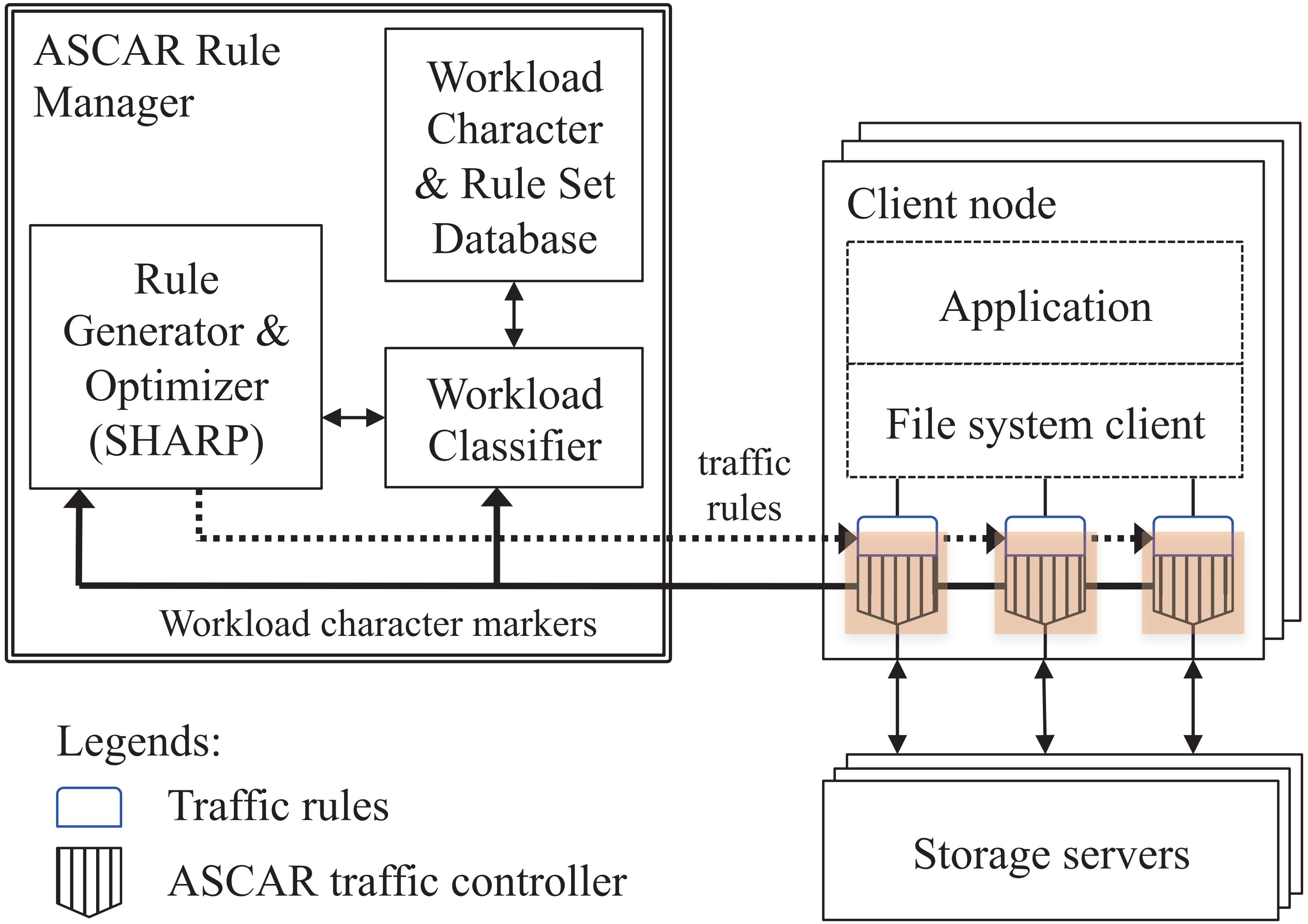
ascar.io

ssrc

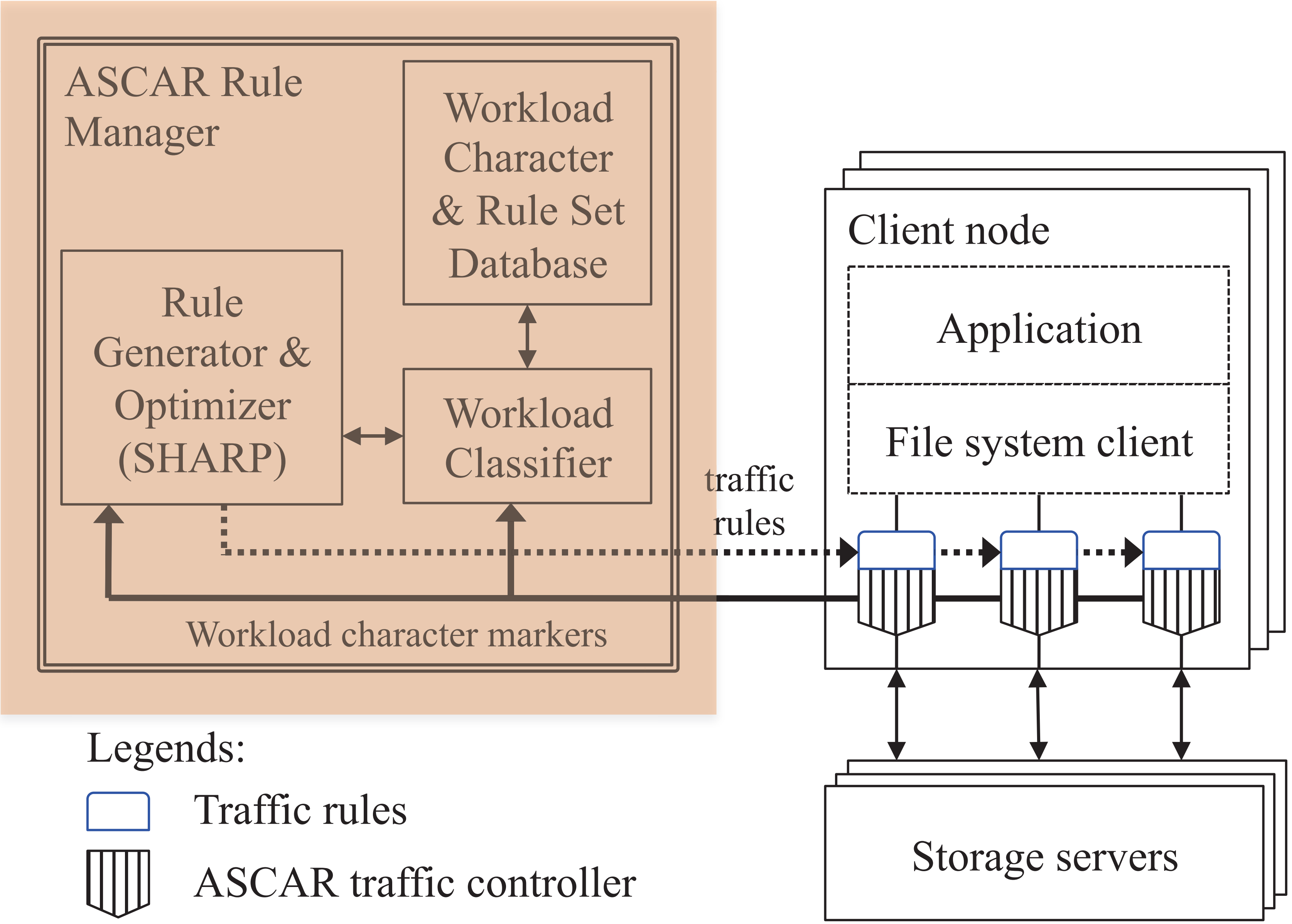
Components of the ASCAR prototype



Components of the ASCAR prototype

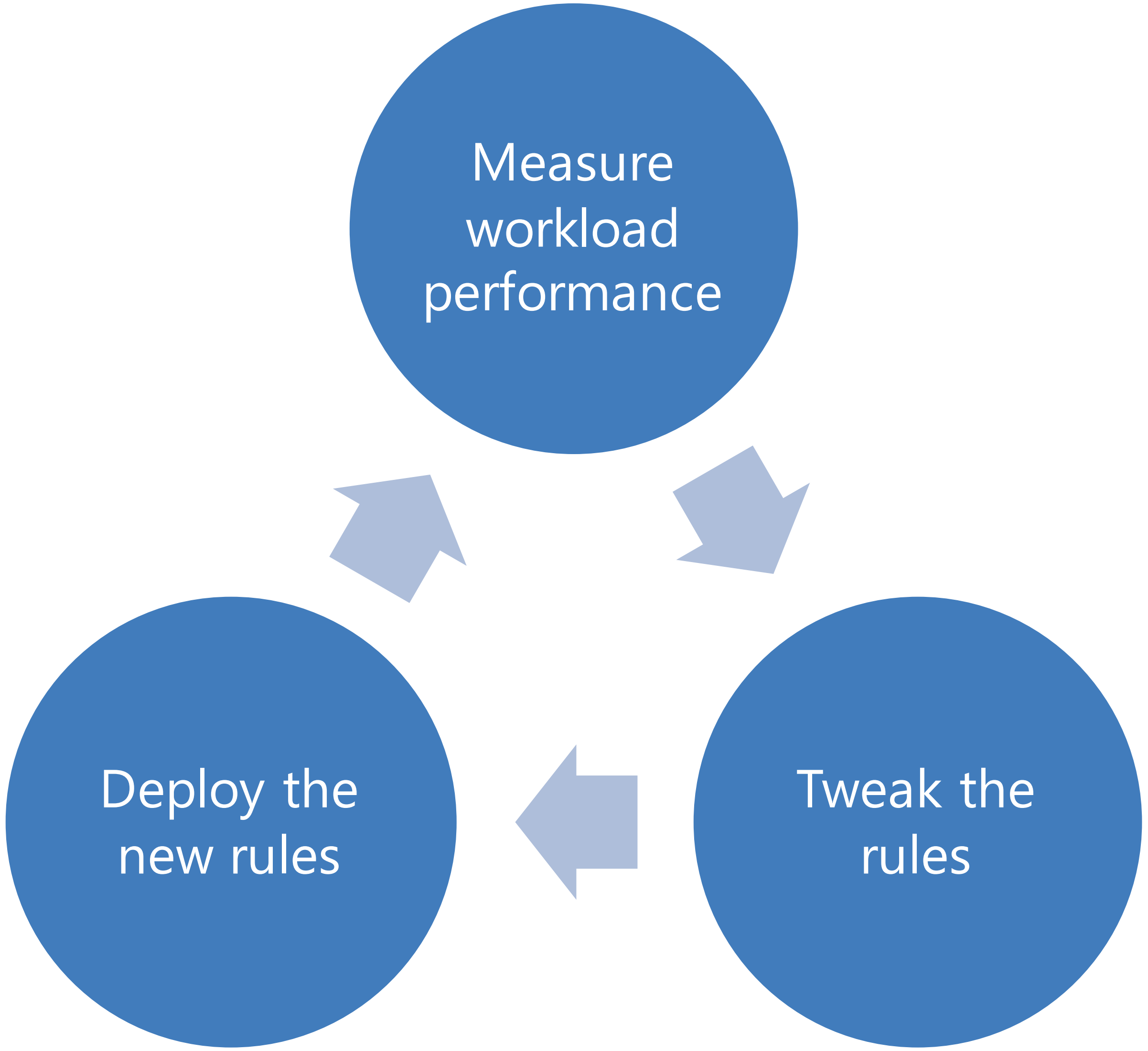


Components of the ASCAR prototype



Created by Yan Li <yanli@ascar.io> http://ascar.io

Generating rules for a certain workload



ascar.io

Rule-based Contention Control

Rules tell the controller how to react to congestions
tweak the congestion window according to request processing latency

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

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

An action describes how to change the I/O queue depth and rate limit: $\langle m, b, \tau \rangle$
new_depth = $m \times$ old_depth + b
τ is the rate limit

What does a rule set look like?

ack ewma	PT ratio	m	b	τ	Times	Avg. ack ewma	Avg. PT ratio
[41, 48)	[2.4, 4.5)	1	-1.7	33	3011	45	3.2
[48, ∞)	[0, 4.5)	1	0.9	40	7426	60	2.6

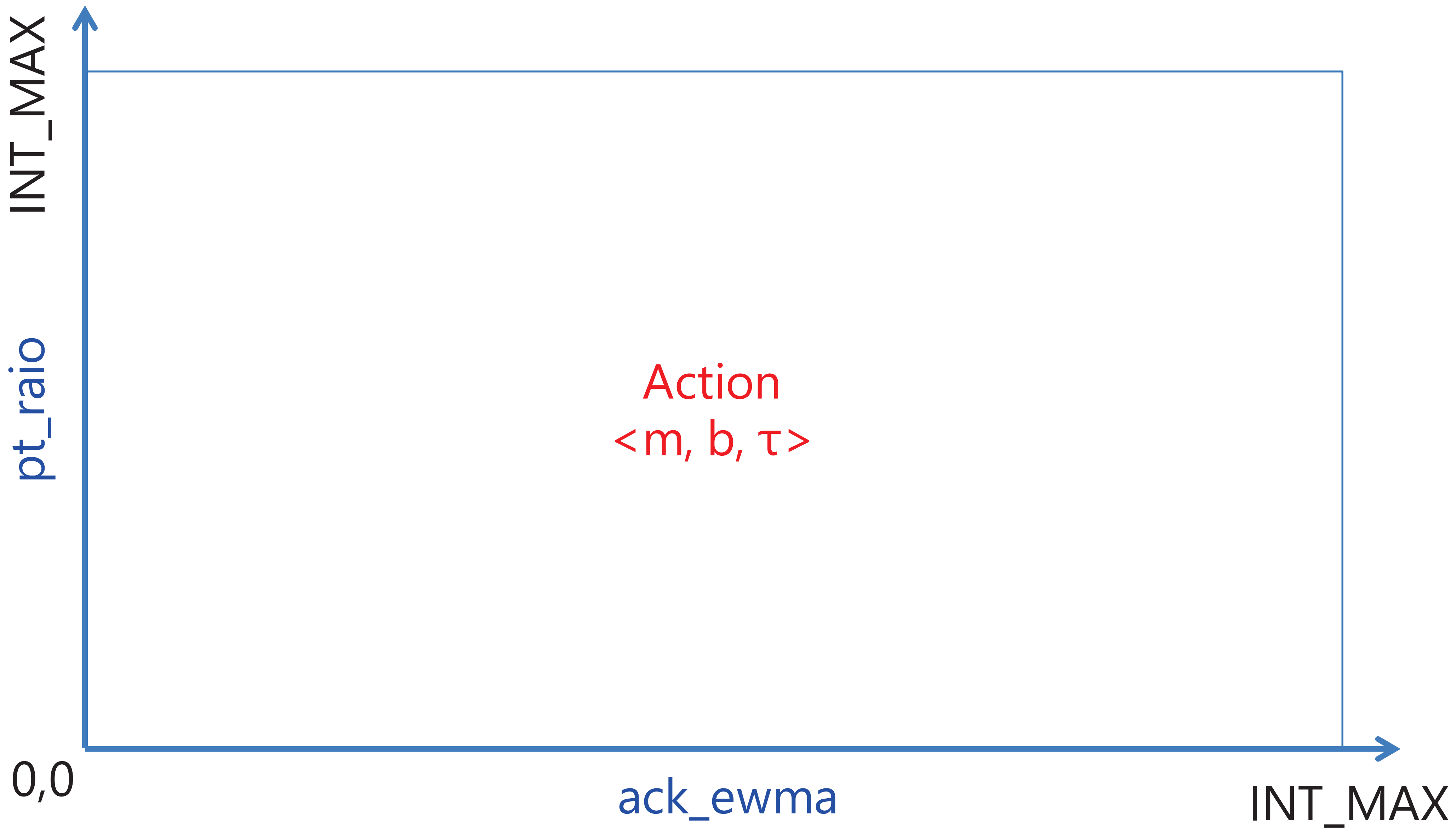
Simplified:
showing only ack_ewma (without send_ewma)
showing only the two most triggered rules

ascar.io
ssrc

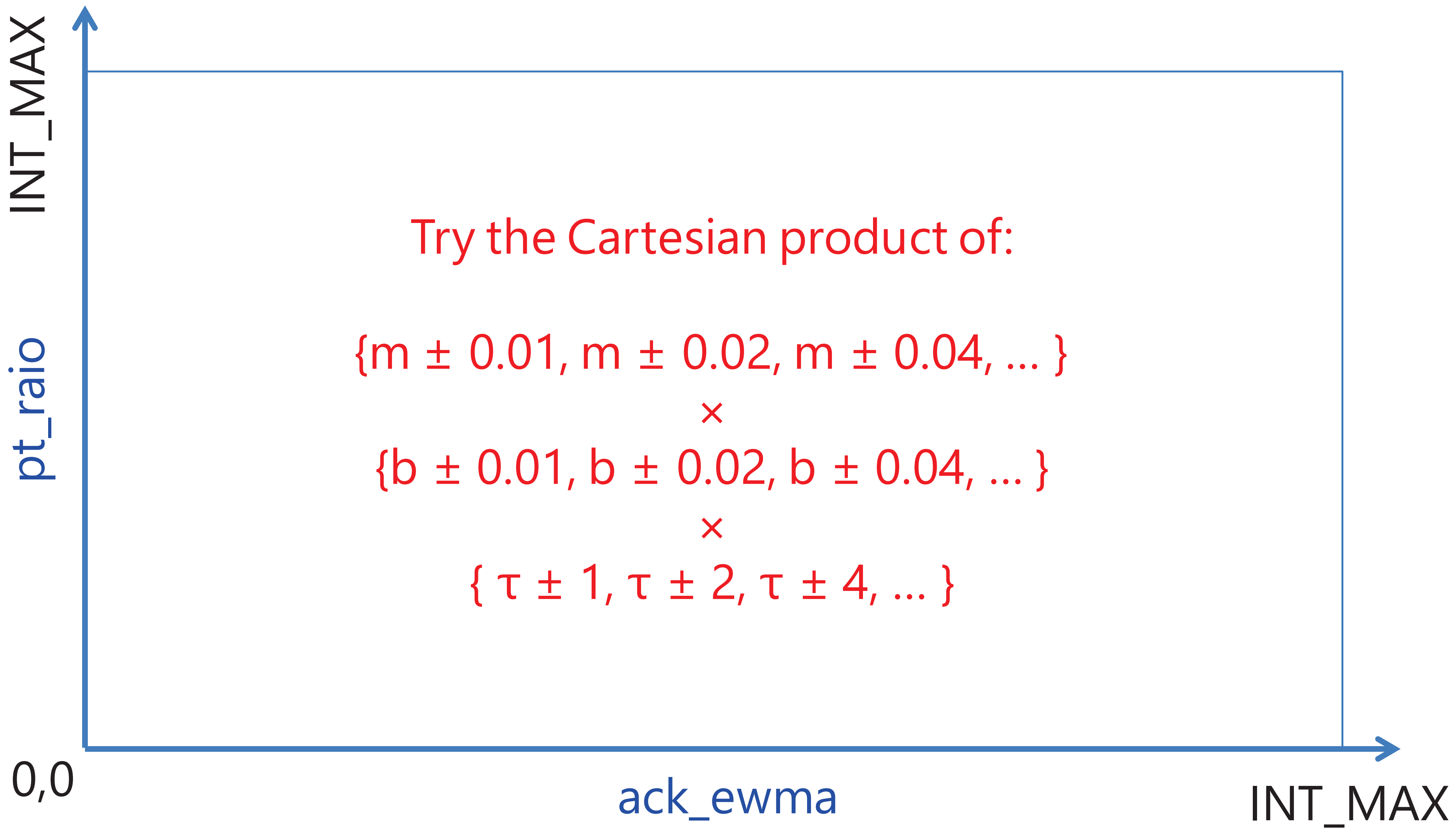
What does a rule set look like?

15,2
0,41001,0,41104,0,237,-1,-176,32840,717,33501,34012,179
0,41001,0,41104,237,445,-1,-176,32840,1197,35489,35772,315
0,41001,41104,48551,0,237,-1,-176,32840,181,39637,42759,180
0,41001,41104,48551,237,445,-1,-176,32840,324,39616,42875,327
41001,48427,0,41104,0,237,-1,-176,32840,107,42038,40307,188
41001,48427,0,41104,237,445,-1,-176,32840,231,42093,40112,308
41001,48427,41104,48551,0,237,-1,-176,32840,1515,44599,44955,173
41001,48427,41104,48551,237,445,-1,-176,32840,3011,44864,44967,318
0,48427,0,48551,445,2147483647,-1,-58,33980,1903,40353,40714,730
0,48427,48551,2147483647,0,445,-1,-58,33980,697,46398,50612,266
0,48427,48551,2147483647,445,2147483647,-1,-58,33980,221,45984,52309,703
48427,2147483647,0,48551,0,445,-1,-58,33980,581,49626,47402,281
48427,2147483647,0,48551,445,2147483647,-1,-58,33980,143,49957,47146,774
48427,2147483647,48551,2147483647,0,445,-1,90,40396,7426,60457,60714,256
48427,2147483647,48551,2147483647,445,2147483647,-1,-58,33980,2226,60599,61187,737

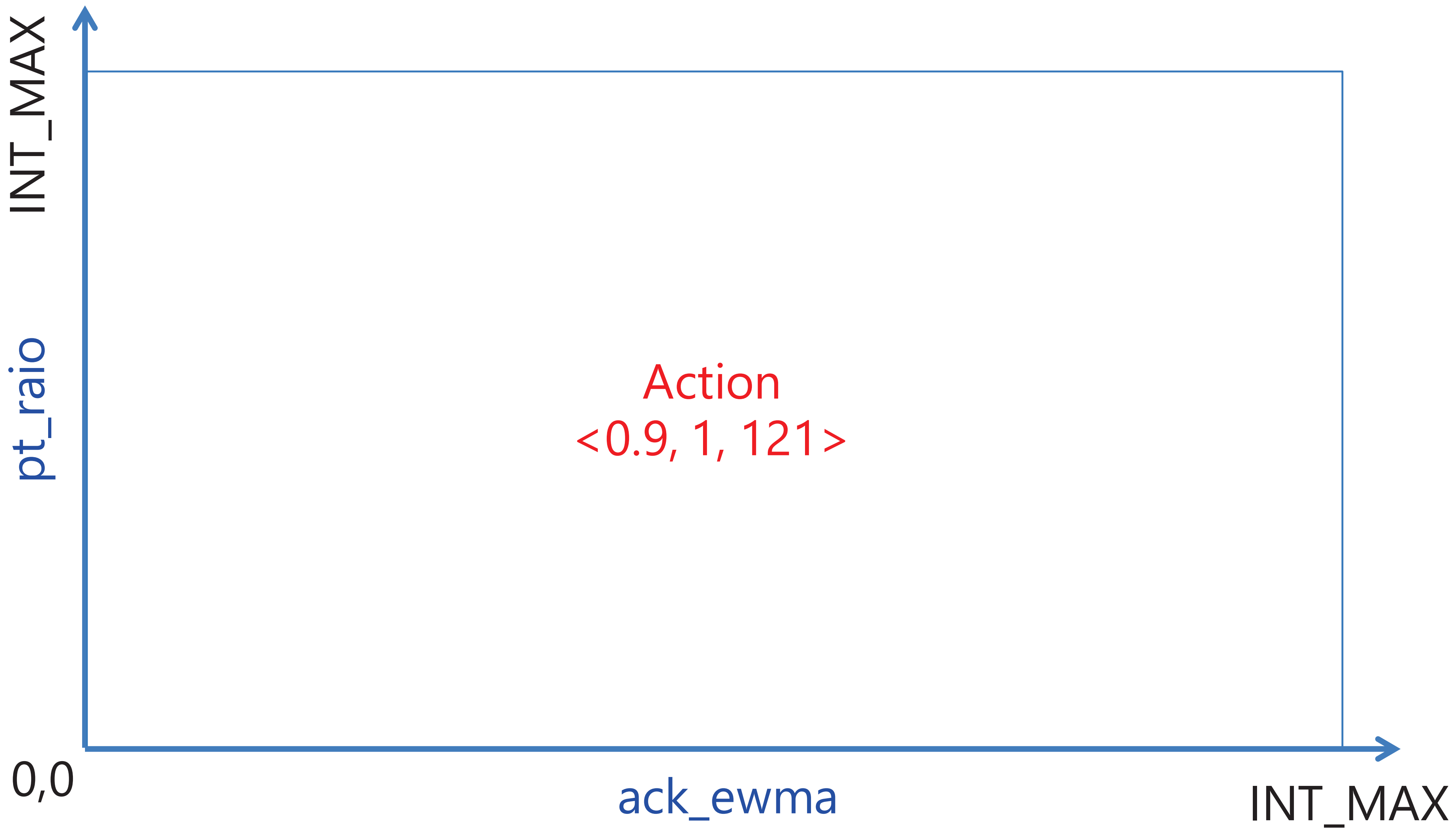
Begin with one rule: the whole state space maps to one action



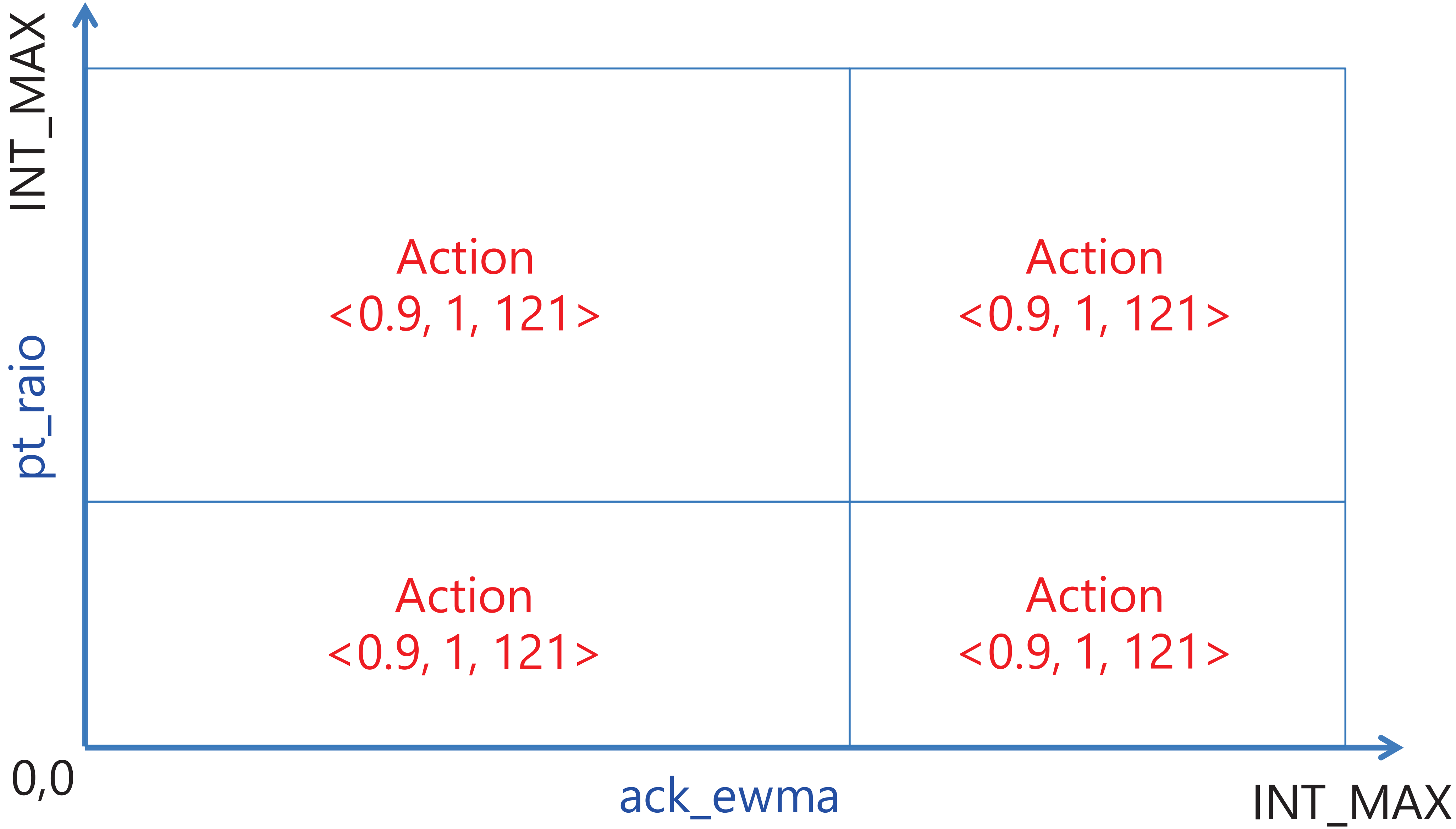
Try different values of <m, b, τ> with the workload



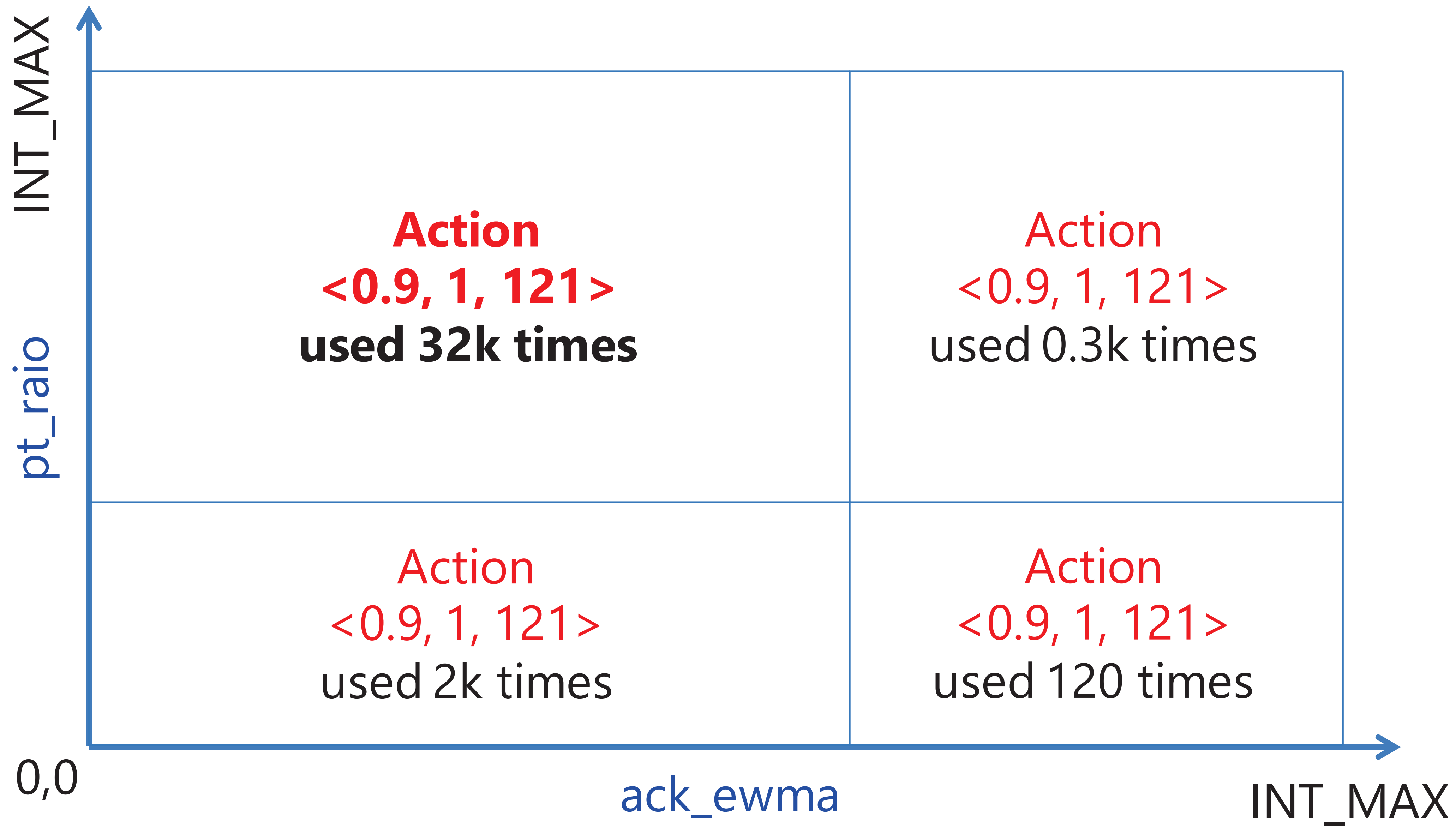
Find the rule that yields highest performance



Split the state space at the most observed state values

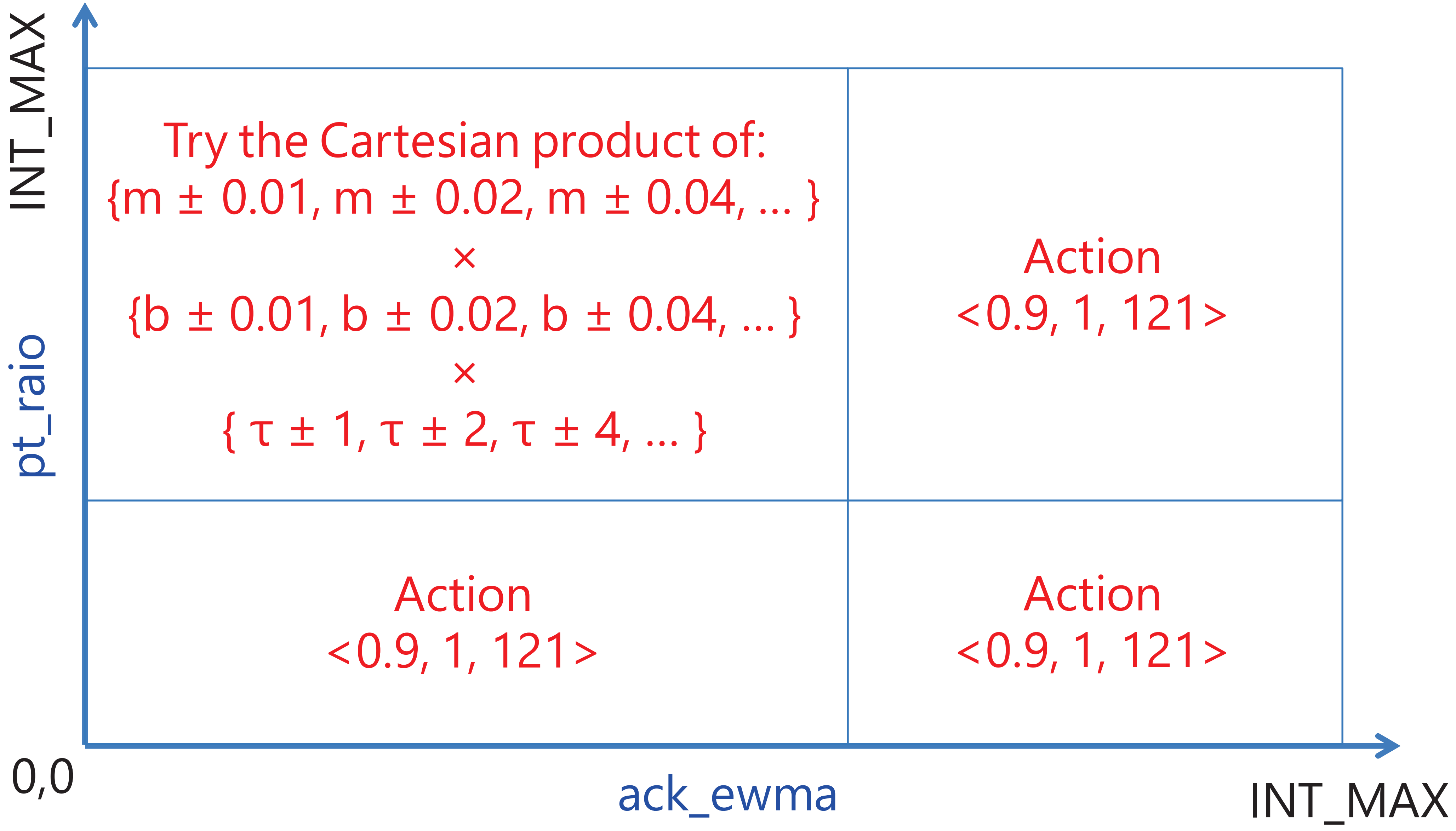


Run the workload, find out the rules that was triggered most often

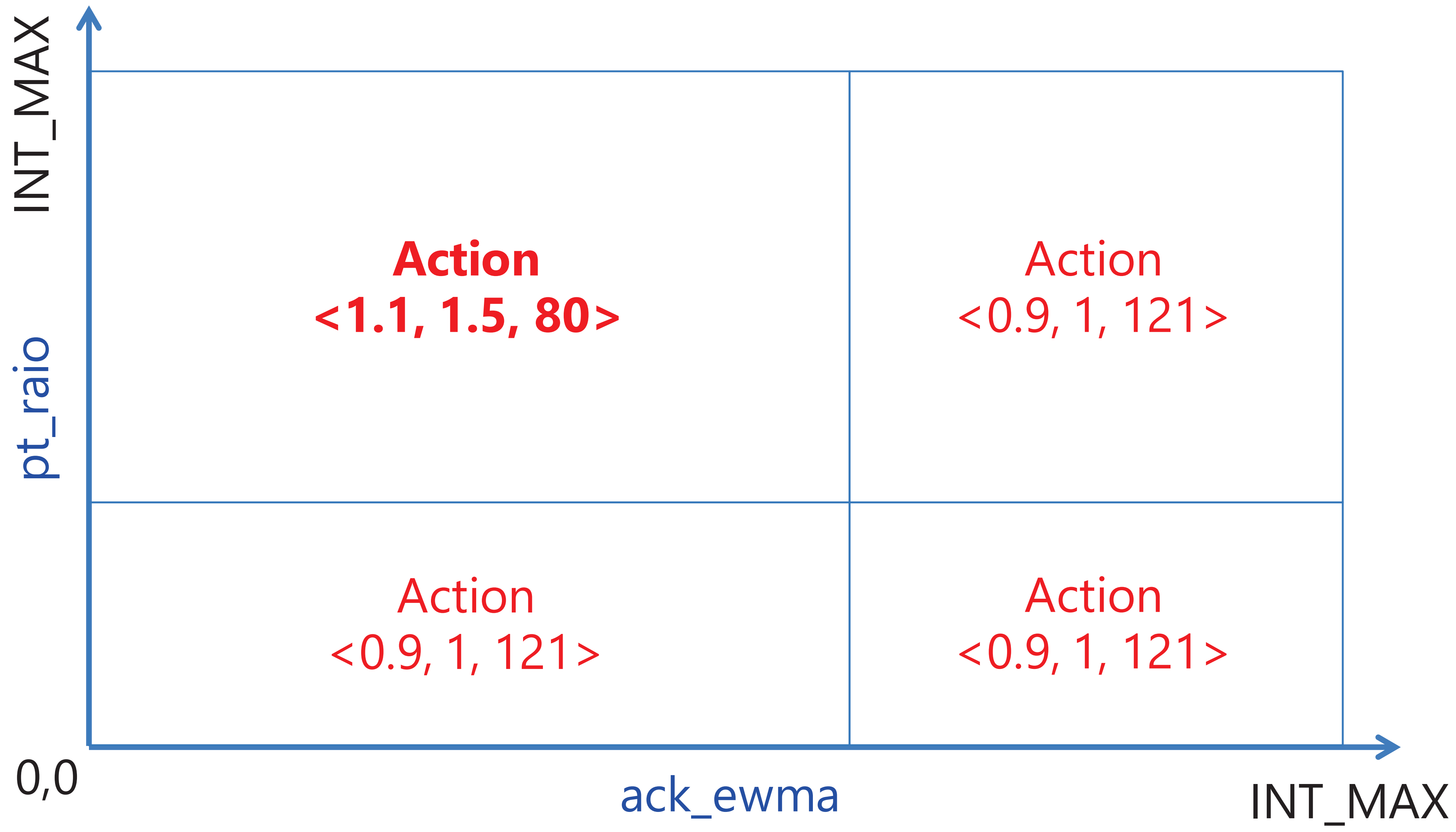


ascar.io
ssrc

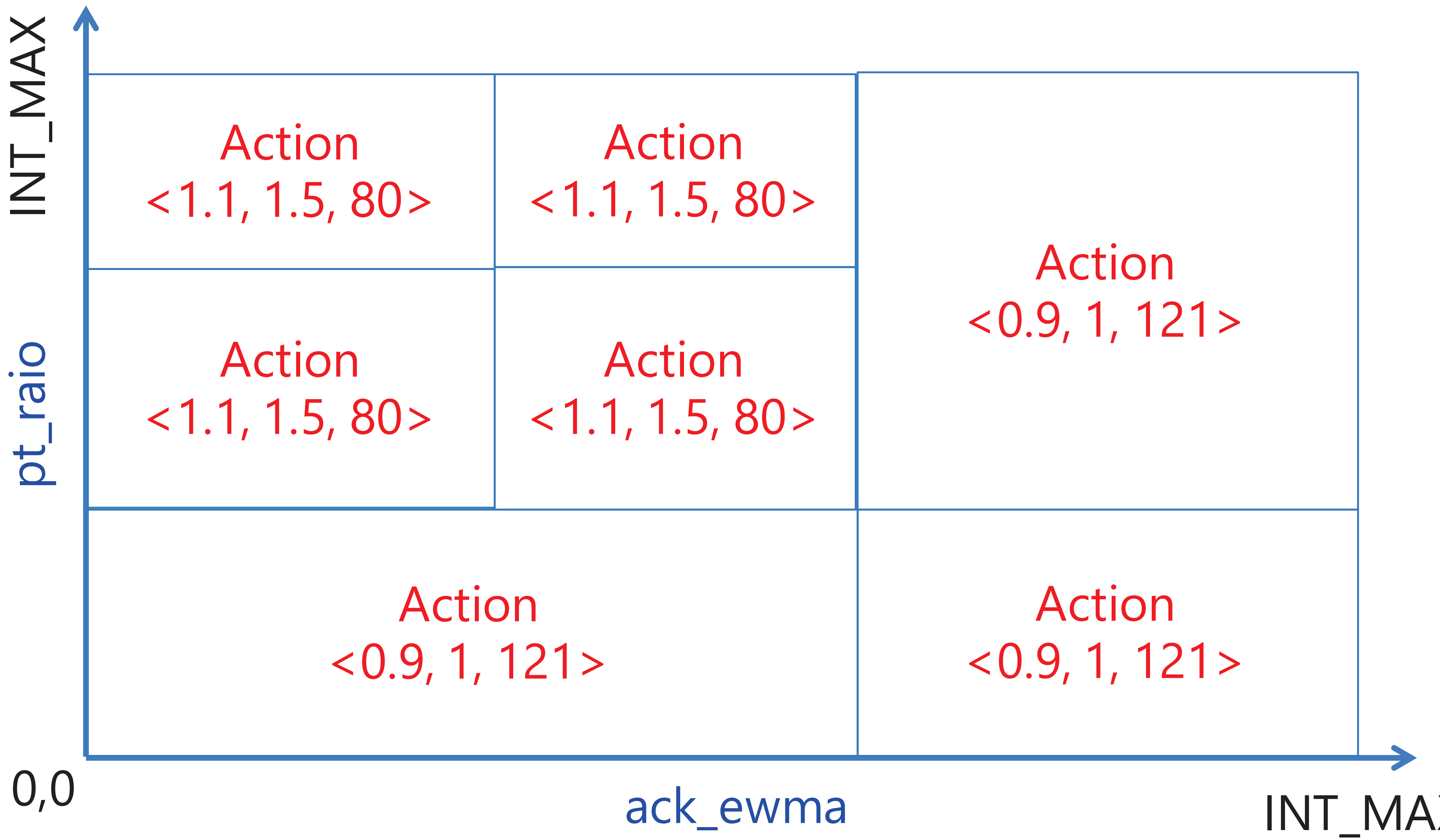
Improve the most used rules by sweeping all possible values of <m, b, τ>



Find the action that works best

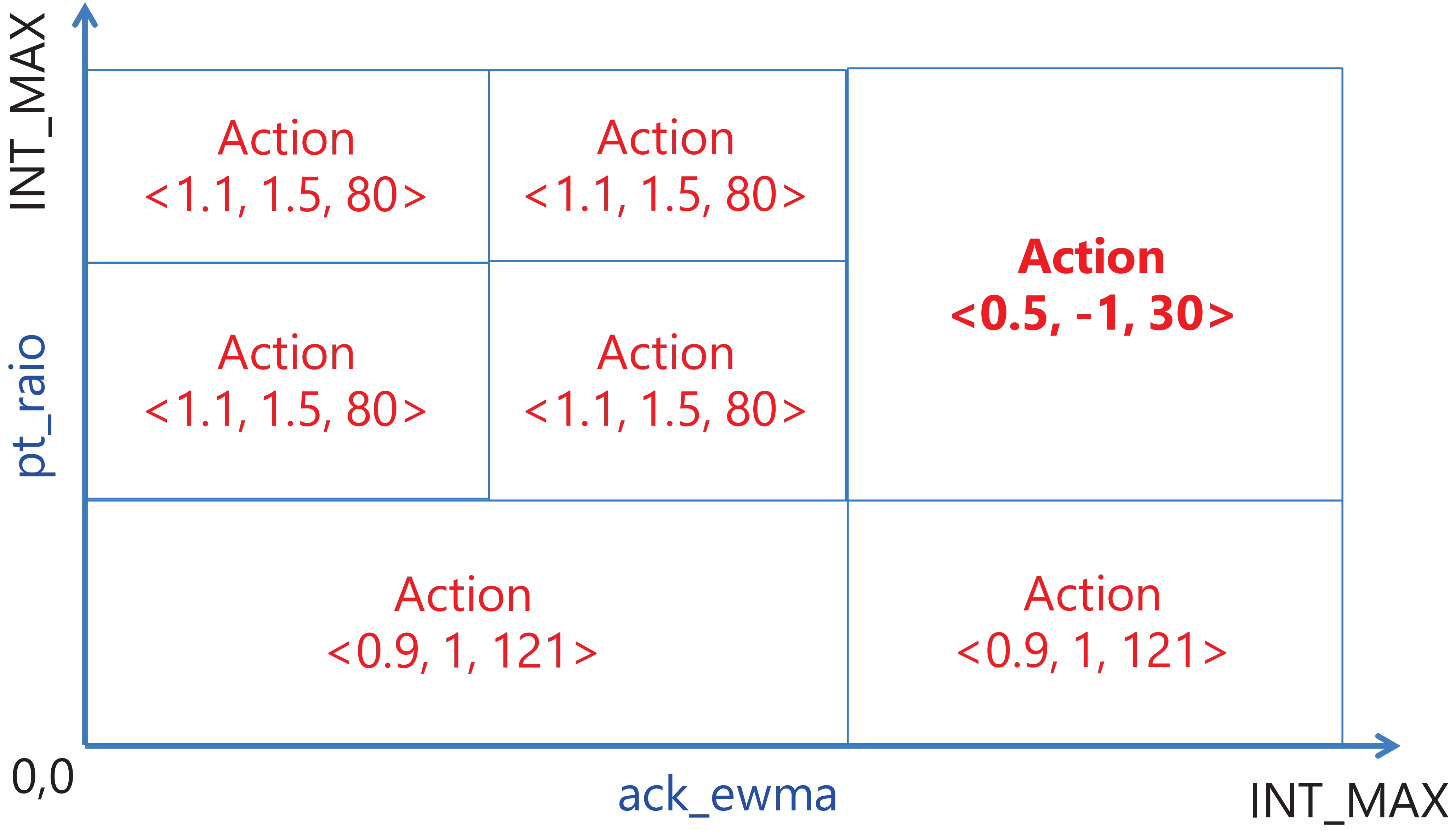


Split the most used rule's state space at the most observed state values

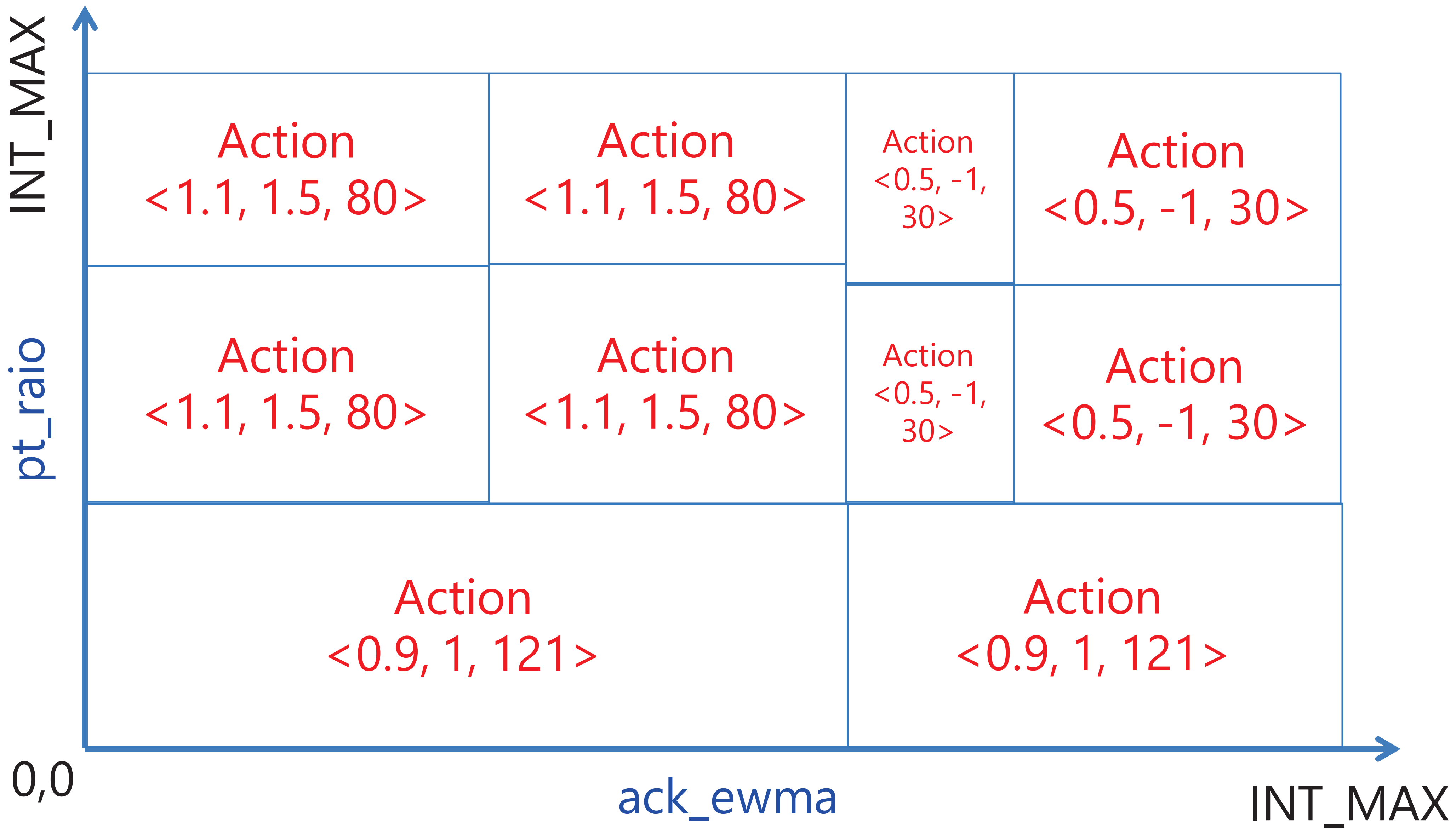


ascar.io ssrc

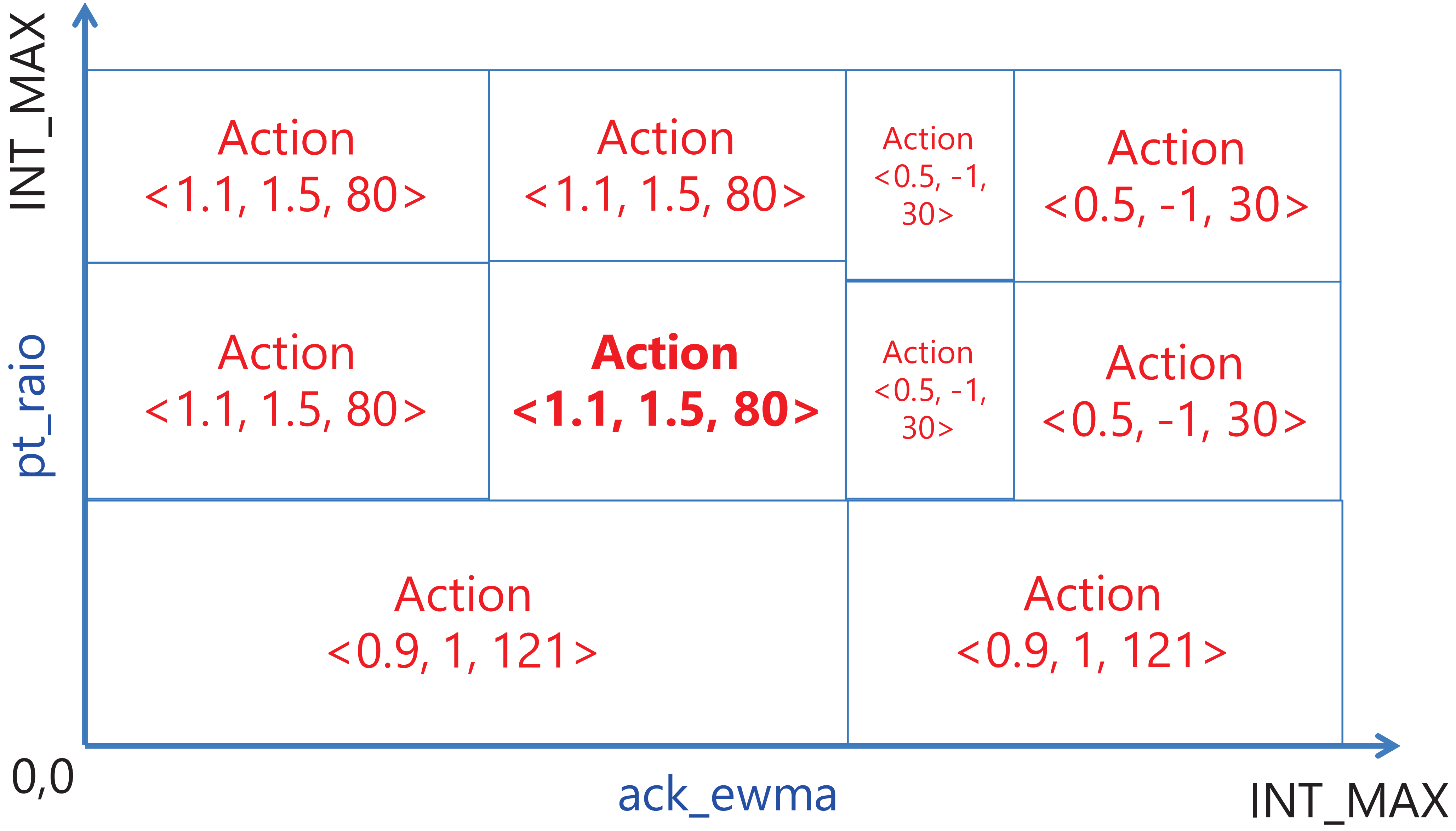
Run workload, find the most used rule, and improve it



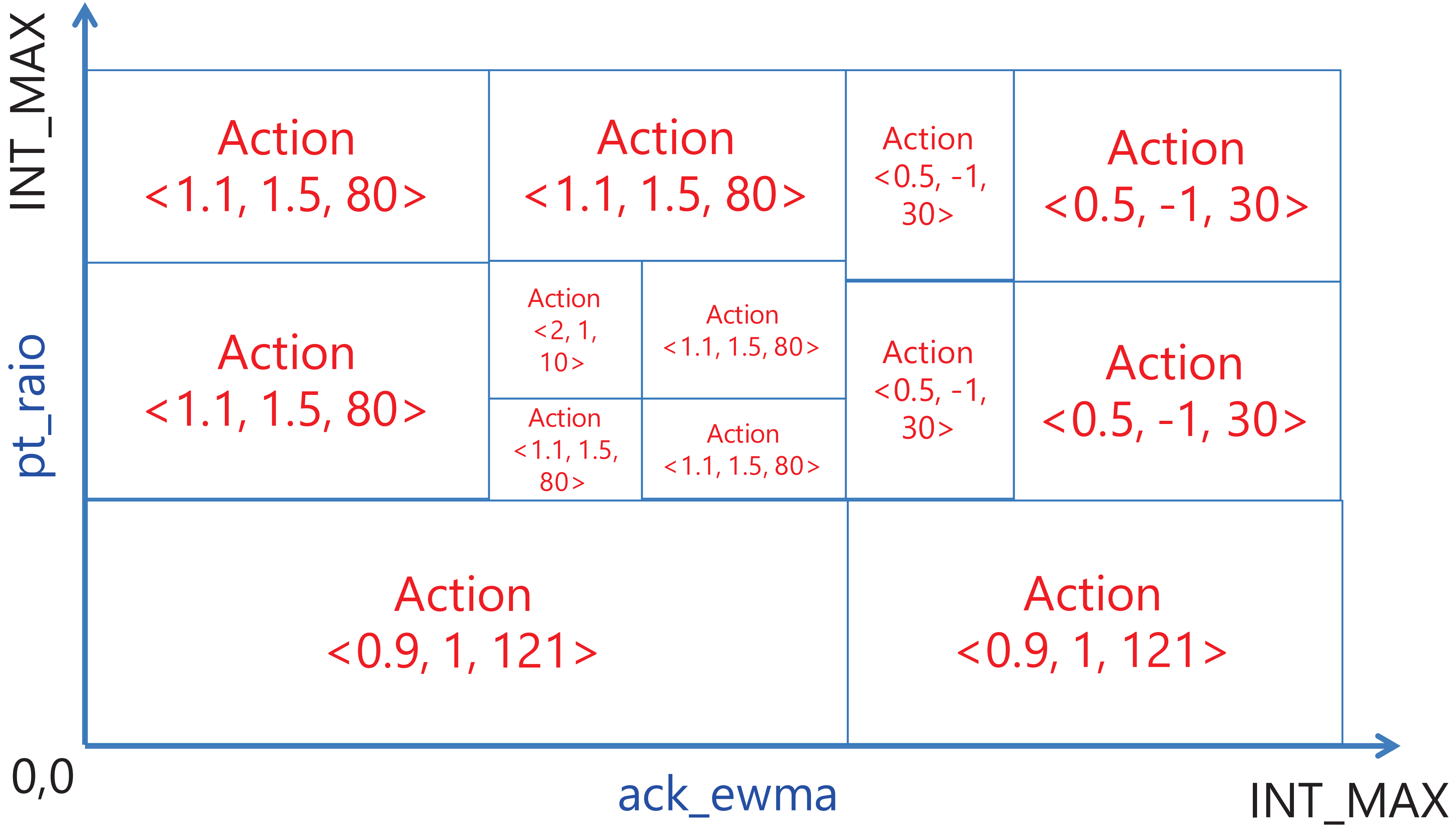
After find the best action, split the most used rule



After find the best action, split the most used rule



Repeat this process



Prototype and Evaluation

An ASCAR prototype for Lustre

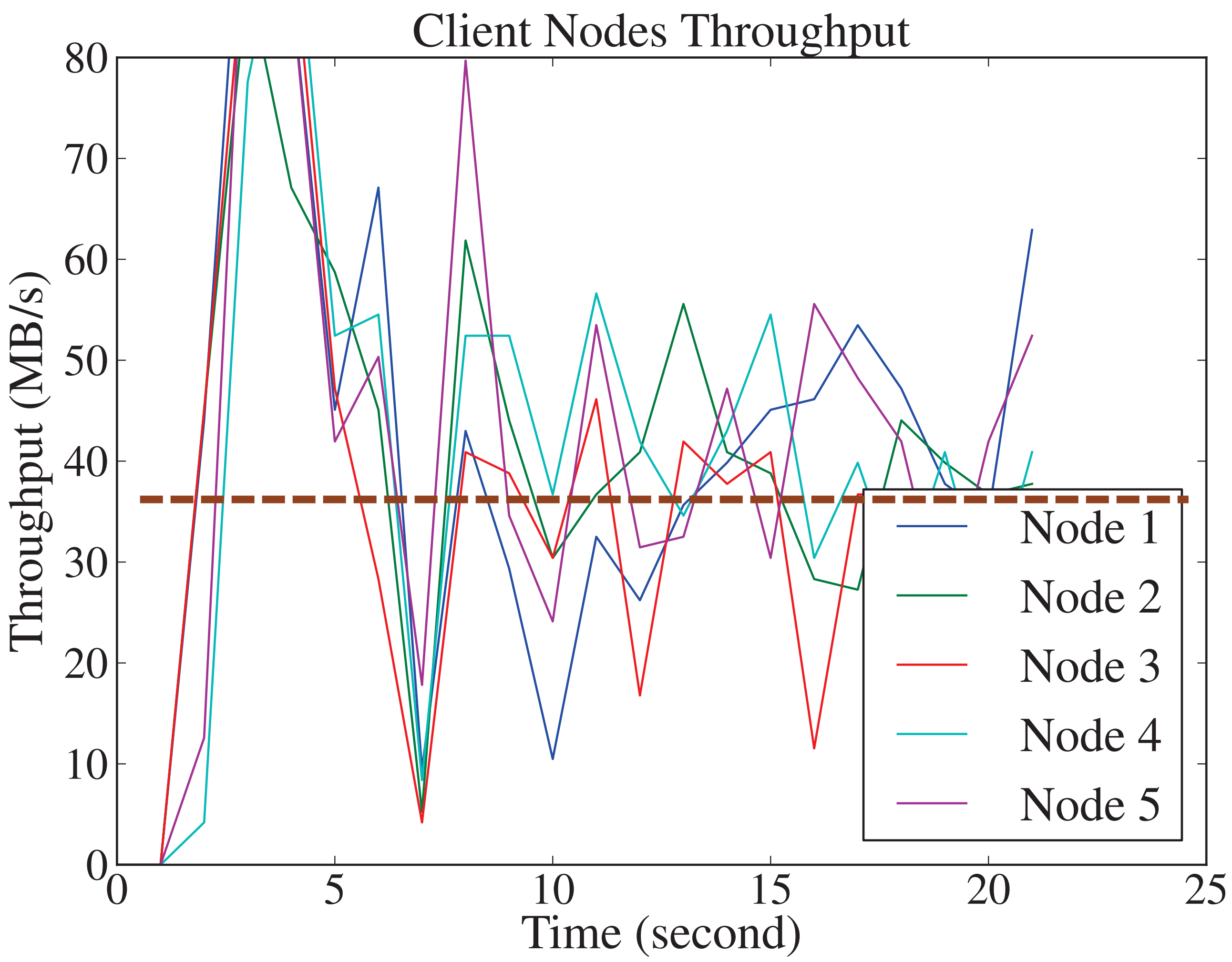
Patched Lustre client to add congestion control

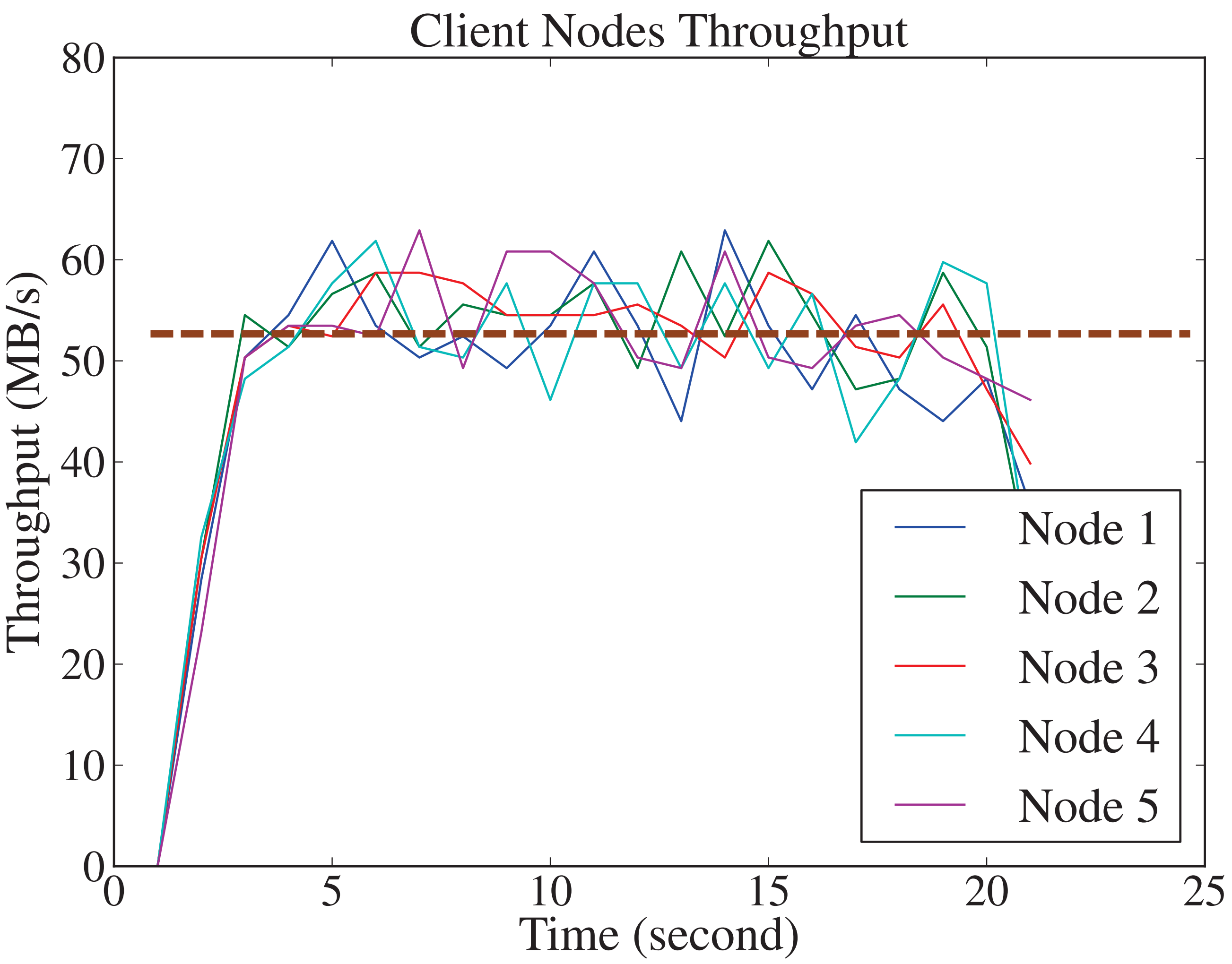
no change to server or other parts

Hardware: 5 servers, 5 clients

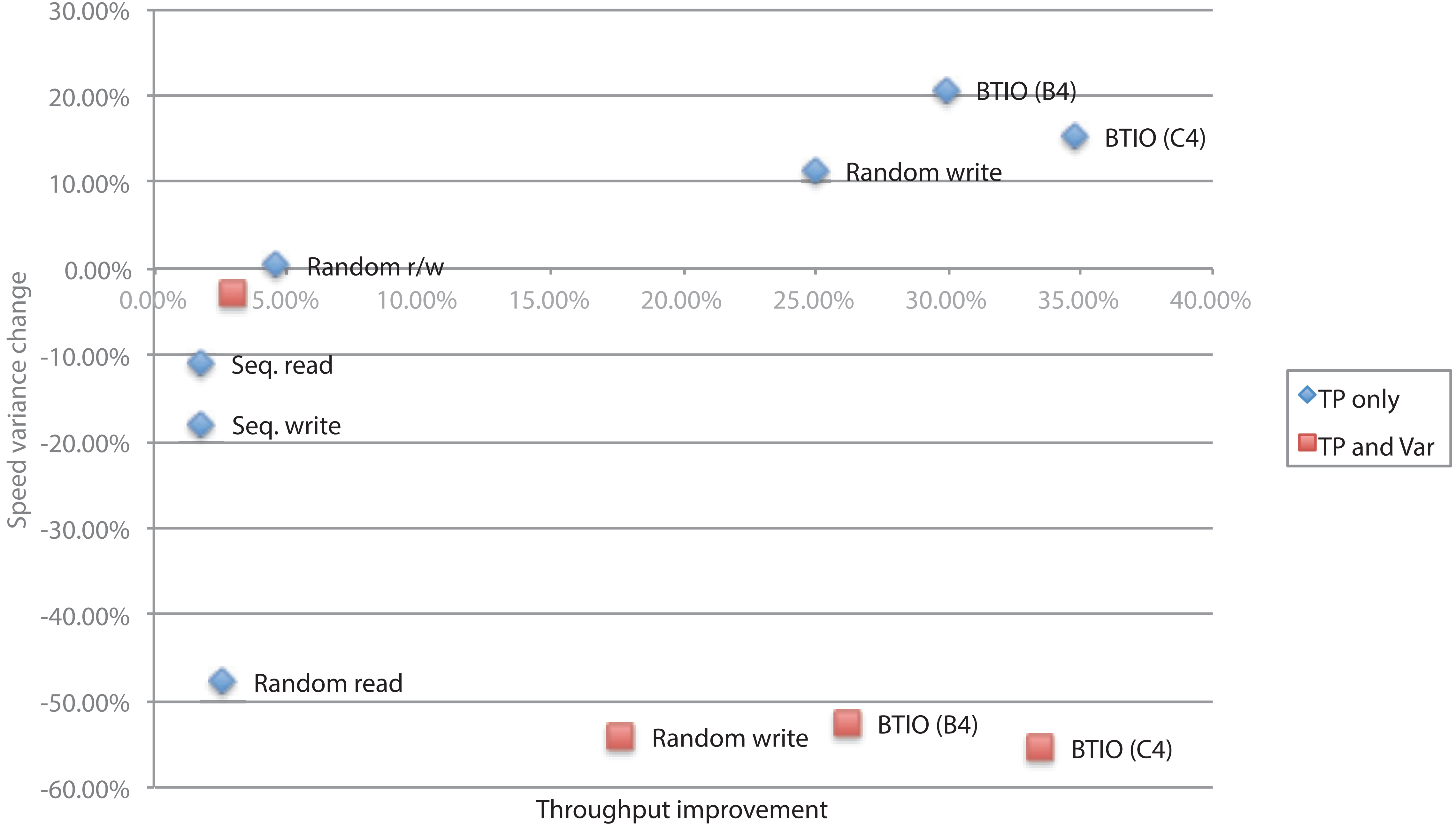
Intel Xeon CPU E3-1230 V2 @ 3.30GHz, 16 GB RAM,
Intel 330 SSD for the OS,
dedicated 7200 RPM HGST Travelstar Z7K500 hard drive for Lustre,
Gigabit Ethernet

ASCAR is good at increasing throughout and decreasing speed variance





Workload Throughput Improvements



Changes to Lustre

Deploying traffic rules to the kernel and reading back statistics

Through procfs: /proc/fs/lustre/osc/*/qos_rules

We also need to read how many times each rule is triggered

Patched lustre/obdclass/lprocfs_status.c to support loading rule files larger than 4 KB

using LIBCFS_ALLOC_ATOMIC() instead of __get_free_page()

Added two fields to /proc/fs/lustre/osc/*/import for real time read/write throughput of osc

Calculating congestion statistics

Updated every time a reply is received

Using a modified equation for calculating ewma because no float support

At the beginning of brw_interpret()

We don't know about the overhead yet.

Process time of each RPC request

We changed the protocol and embedded sent_time in each outgoing request and use that to calculate the process time. (Alternative ways?)

Controlling I/O queue depth

Changing max_rpcs_in_flight

Also patched in brw_interpret()

Frequency is limited

We used twice per second in all our experiments

What about overhead?

Is there a better alternative?

Changes to the protocol

Embed the sent_time in outgoing RPC packets

ptlrpc/pack_generic.c: replaced o_padding_{4,5}

No need to change the server

The server just sends back the sent_time

Is there a better alternative way?

Rate limiting

Imposing a minimum gap between RPCs

By introducing delays in osc_build_rpc()

Using udelay(), usleep_range(), and msleep() according to sleep duration

Is it better to do this in osc_check_rpcs() instead of just sleeping?

Size of the patch

File	LOC	Changes
include/ascar.h	179	Traffic controller
osc/osc_request.c	169	Traffic controller
osc/qos_rules.c	116	Traffic rule set parser
ascar_sharp.sh	374	SHARP main program
osc/lproc_osc.c	110	The procfs interface
gen_candidate_rules.py	166	Implementation of GenerateCandidateRulesets()
split_rule.py	145	Implementation of SplitRule()
ascar-tests/ (dir)	396	Test cases

ascar.io ssrc

Project Status

Paper and source code of our prototype are published http://ascar.io, https://github.com/mlogic/ascar-lustre-2.4-client

Prototype done on Lustre 2.4. Porting to 2.8.

We will start to work with the community to push our patch upstream

Created by Yan Li <yanli@ascar.io> http://ascar.io

Future Work and Research Questions

Collaboration

Evaluation on a larger scale

Are there features or work-in-progress that can collaborate with ASCAR?

Are there hints from OST we can use?

Online rule optimization

Current ASCAR prototype requires a lengthy offline learning process

Online tweaking of rules using random-restart hill climbing

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

What is the best way to dump details of each op of the past 2 minutes to user space?

Requirement: start time, end time, file handle (or name), op type, offset, length, OST ID

How to monitor the change of workload?

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. average positional gap between seq. ops
3. average temporal gap between seq. ops

Sample:
Different 60% read + 40% write workloads

Created by Yan Li <yanli@ascar.io> http://ascar.io

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

ascar.io
ssrc

Acknowledgments

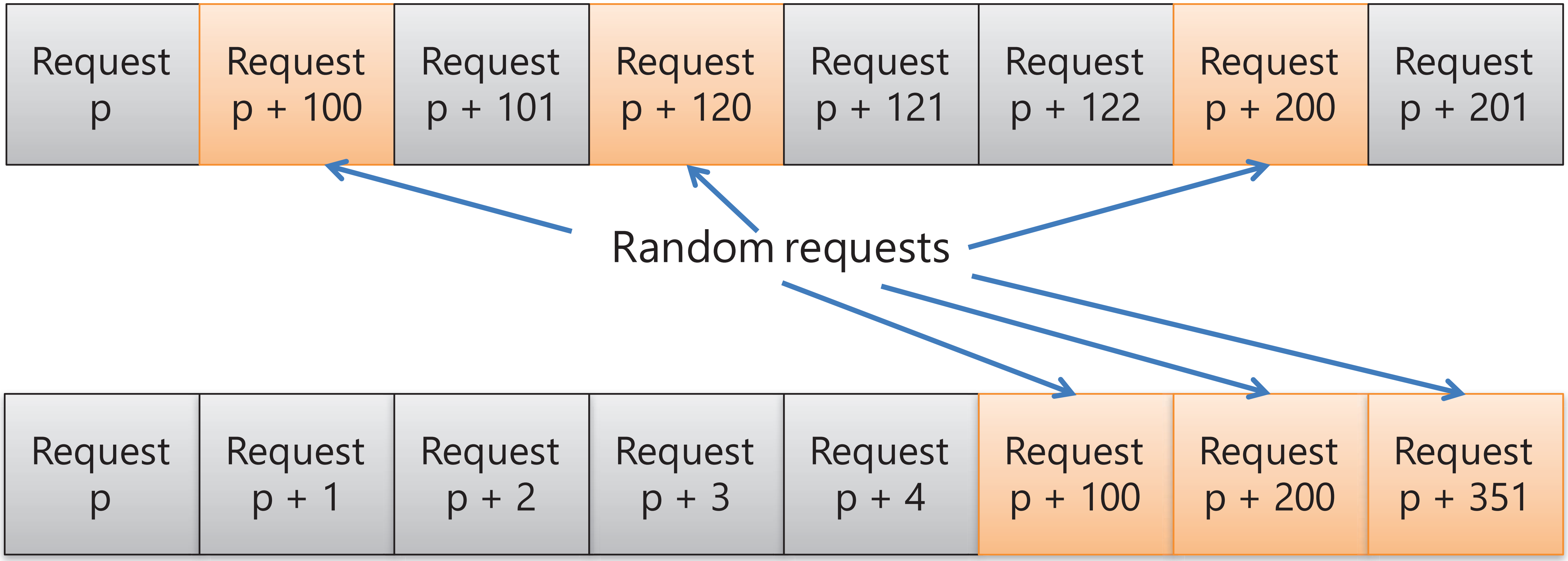
This research was supported in part by the National Science Foundation under awards IIP-1266400, CCF-1219163, CNS- 1018928, the Department of Energy under award DE-FC02- 10ER26017/DESC0005417, Symantec Graduate Fellowship, and industrial members of the Center for Research in Storage Systems. We would 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.

ASCAR project: http://ascar.io

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

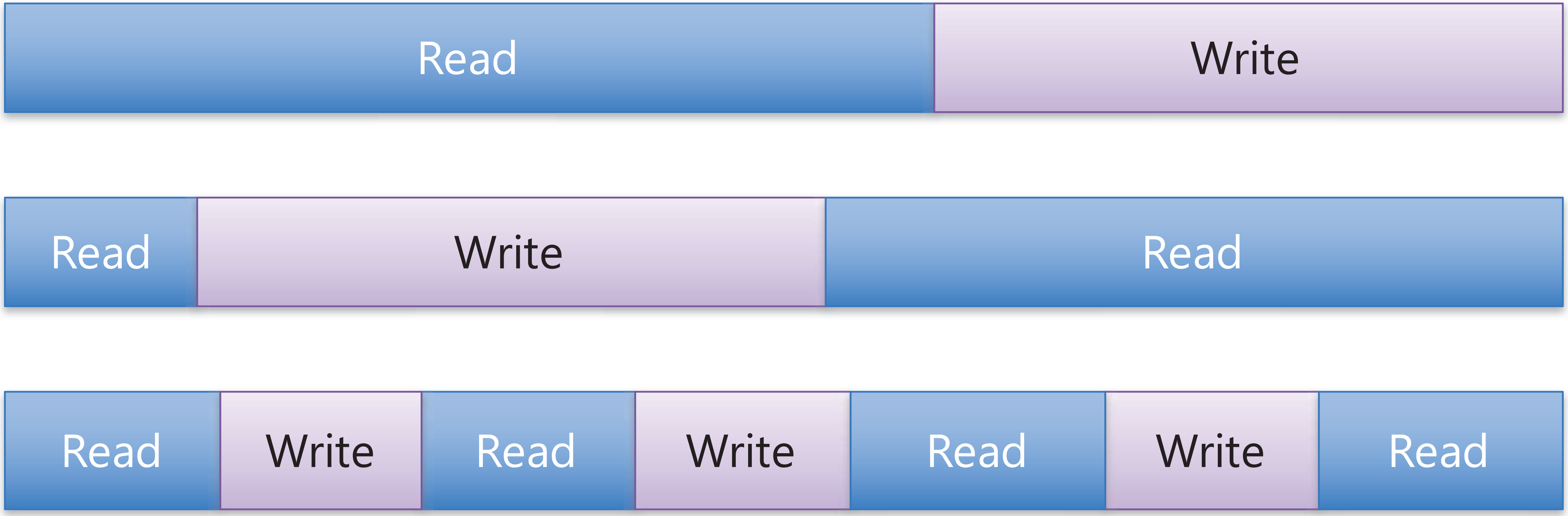
Backup

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



ascar.io
ssrc

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



Sample Congestion State Statistics