Performance Evaluation in Multi-Tenant In-memory Data Grids

By:

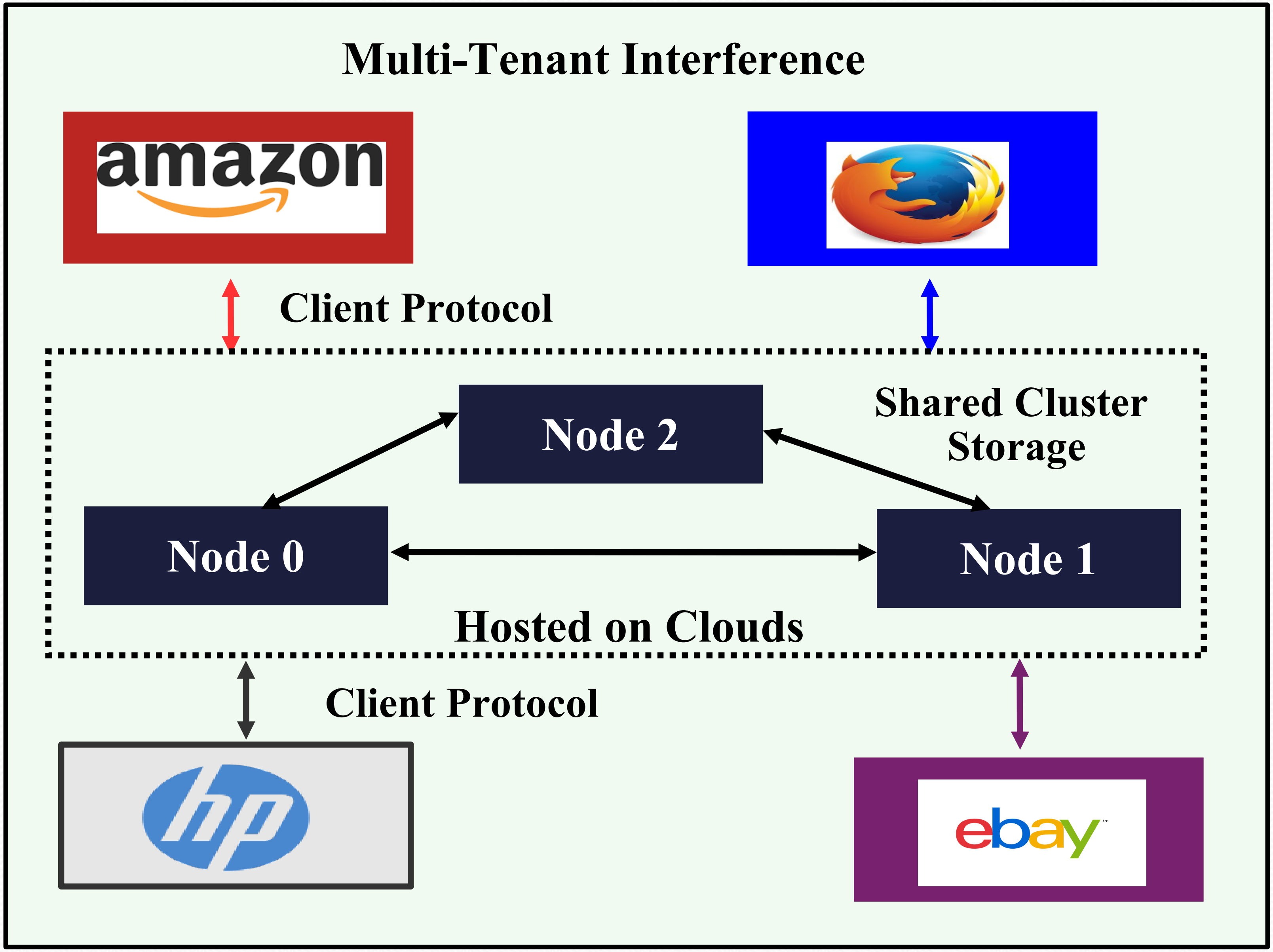
Anwesha Das

North Carolina State University

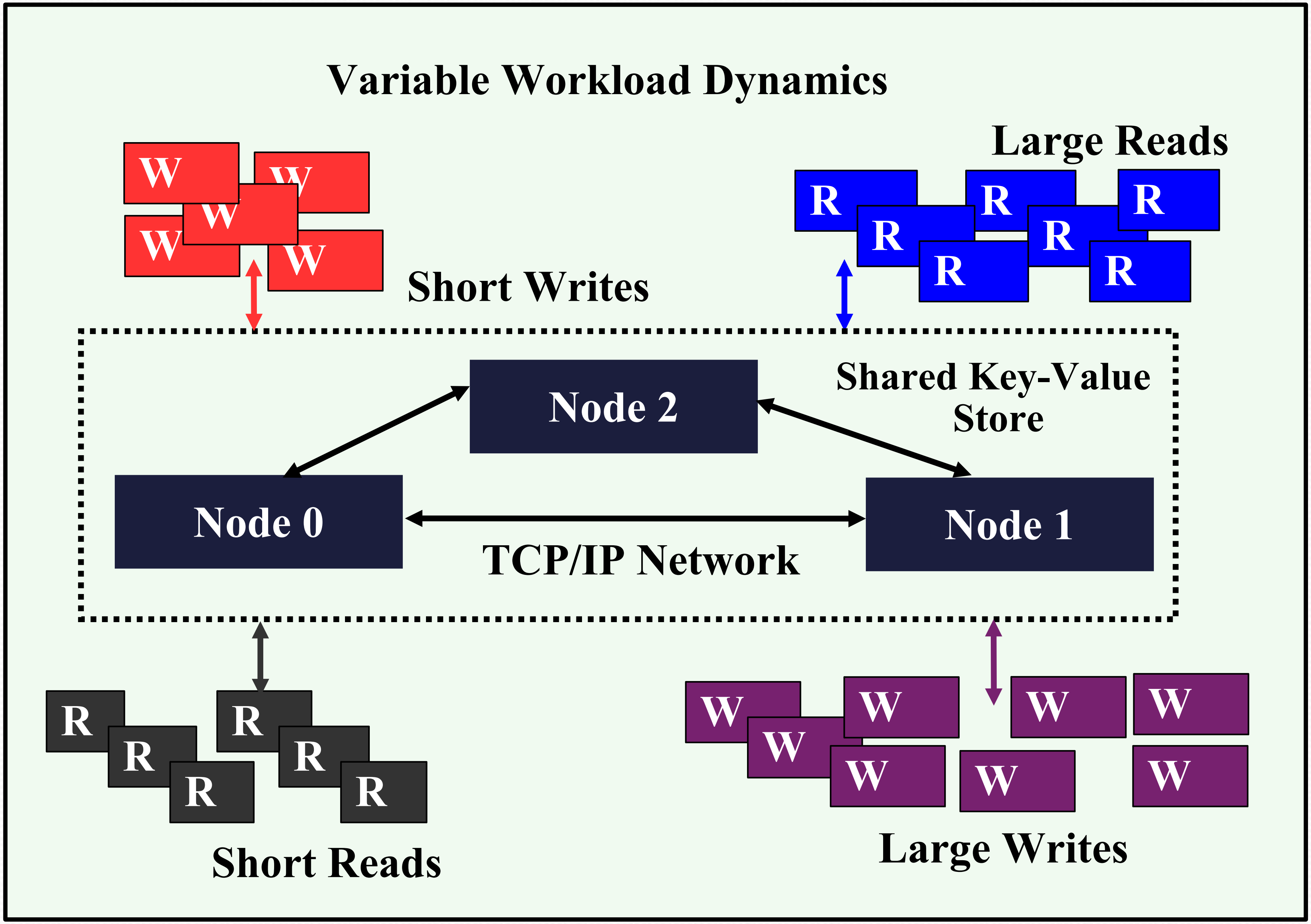
Outline

- Problem
- Research Goals
- Solution Approach
- Experimental Evaluation
- Related Work
- Conclusion/Future Work

Motivation

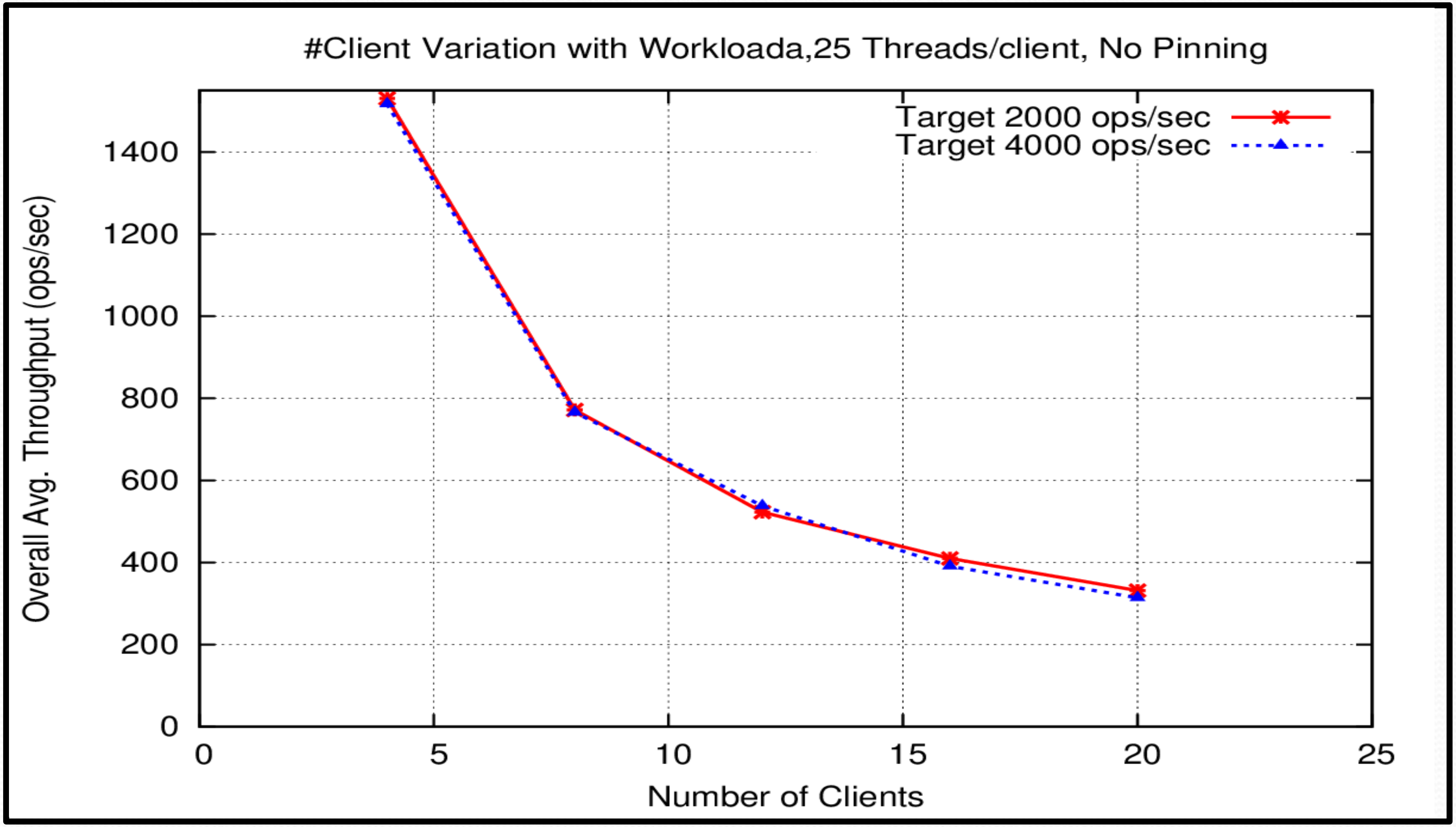

Multi-Tenant Interference
amazon
Client Protocol
Shared Cluster Storage
Node 2
Node 0
Node 1
Hosted on Clouds
Client Protocol
hp
ebay

Motivation



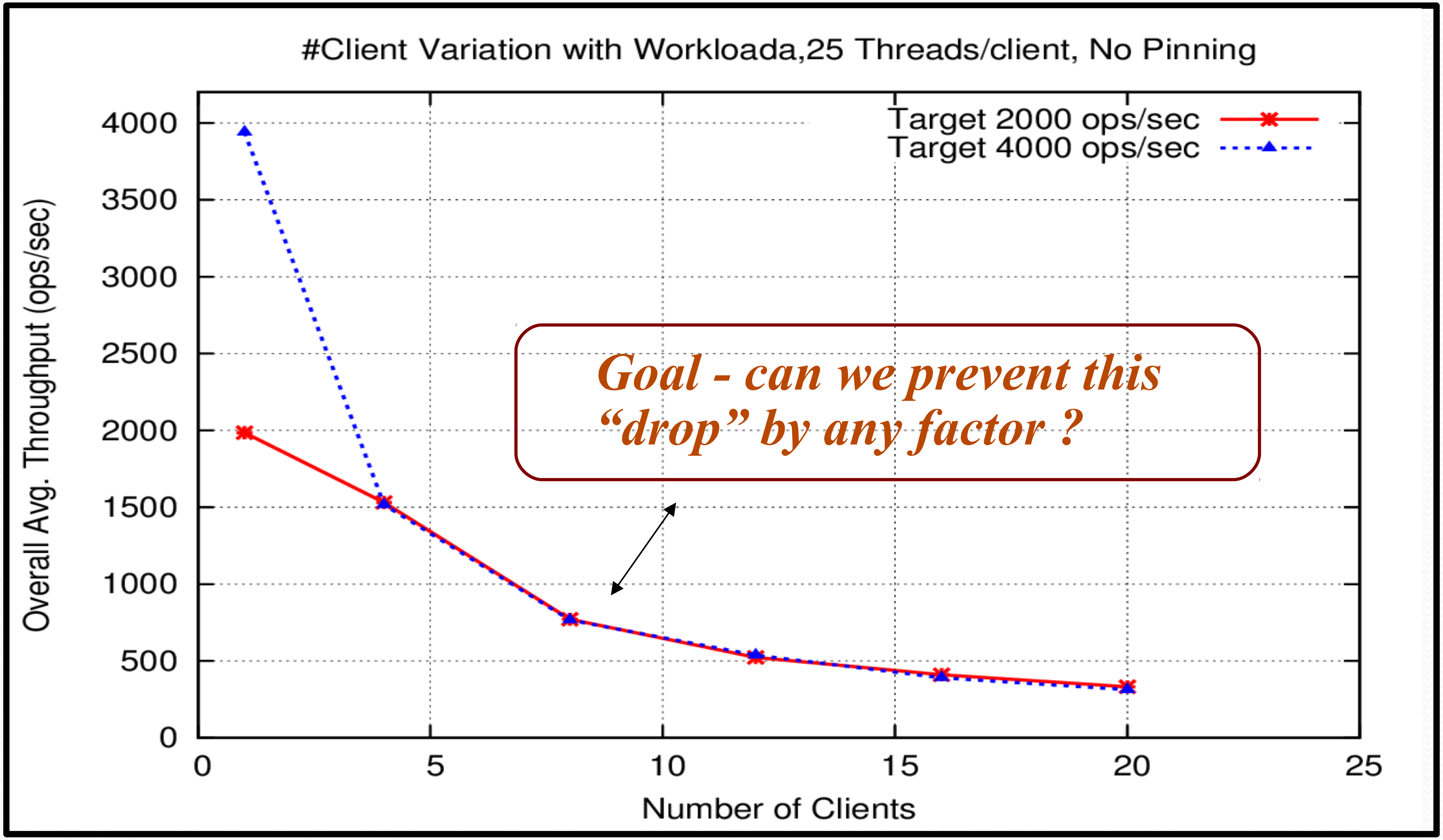
Variable Workload Dynamics
W
W
W
W
W
Short Writes
Large Reads
R
R
R
R
R
R
R
Shared Key-Value Store
Node 2
Node 0
Node 1
TCP/IP Network
R
R
R
R
R
R
Short Reads
W
W
W
W
W
W
W
W
W
Large Writes

Problem



- ✓ ***Decrease in throughput with increase in number of clients***
- ✓ ***Presence of contention leading to performance degradation***

Problem



Problems ? *Resource Contention, Performance Degradation*

Aim *– Understand the source of contention, Find out performance bottlenecks in Hazelcast, How to alleviate contention ??*

Key-Value Store Users

Key-Value Stores	Open Source
BigTable	*No*
Pnut	*No*
DynamoDB	*No*
MongoDB	*Yes*
Voldemort	*Yes*
HBase	*Yes*
HyperTable	*Yes*
ZBase	*Yes*
Cassandra	*Yes*
MemcacheD	*Yes*
Redis	*Yes*
Hazelcast	*Yes*

Wide Commercial and Academic Usage

Outline

- Problem
- Research Goals
- Solution Approach
- Experimental Evaluation
- Related Work
- Conclusion/Future Work

Objective

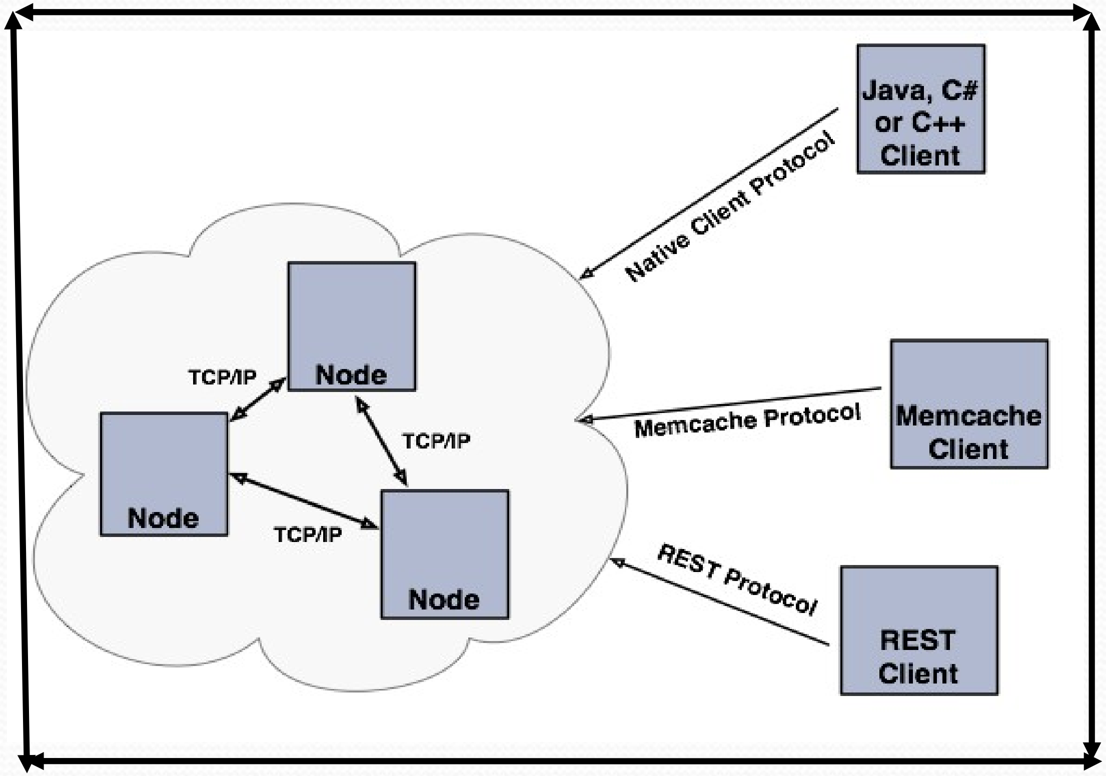
- ***Understand Hazelcast in the context of multi-tenancy.***
- ***How high is the performance degradation ?***
- ***Observe performance characteristics with varying number of clients, workload type, target throughput, thread count.***
- ***What causes degradation ? Investigate performance bottlenecks in Hazelcast to eliminate contention.***

Outline

- Problem
- Research Goals
- Solution Approach
- Experimental Evaluation
- Related Work
- Conclusion/Future Work

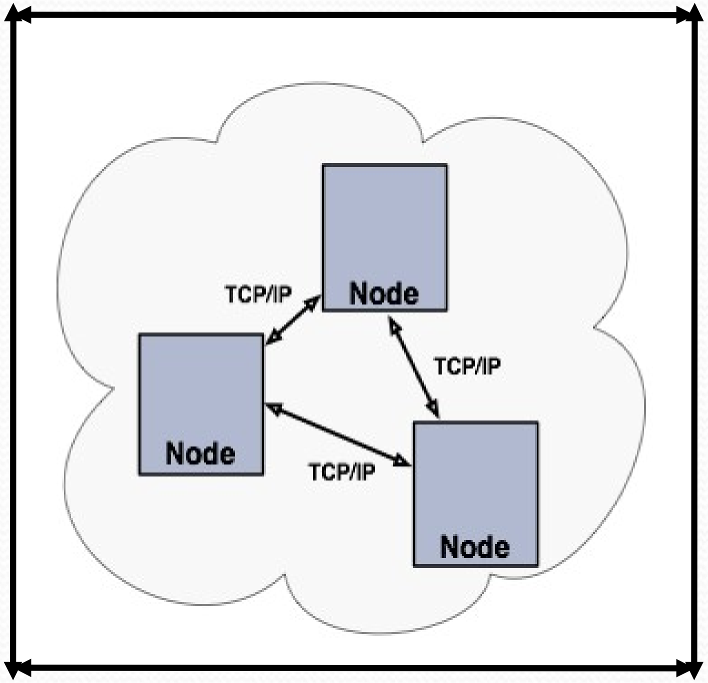
Hazelcast Architecture

Client to Cluster



Hazelcast

Peer to Peer



Client – No data
Default – 271 Partitions

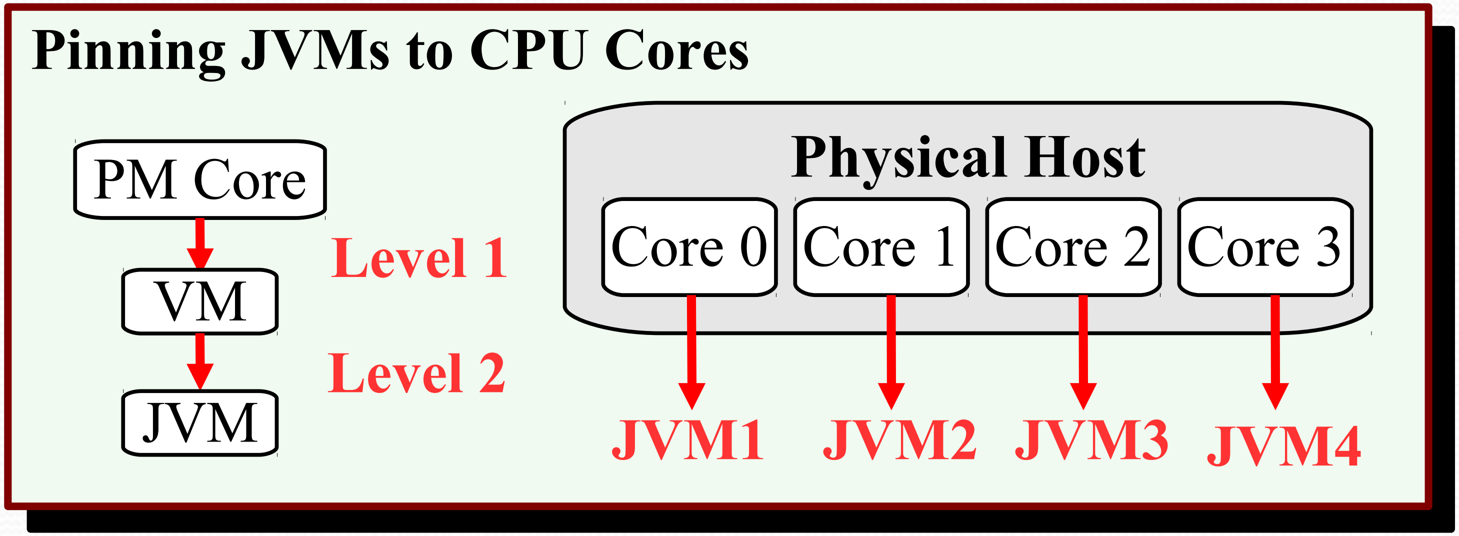
Node

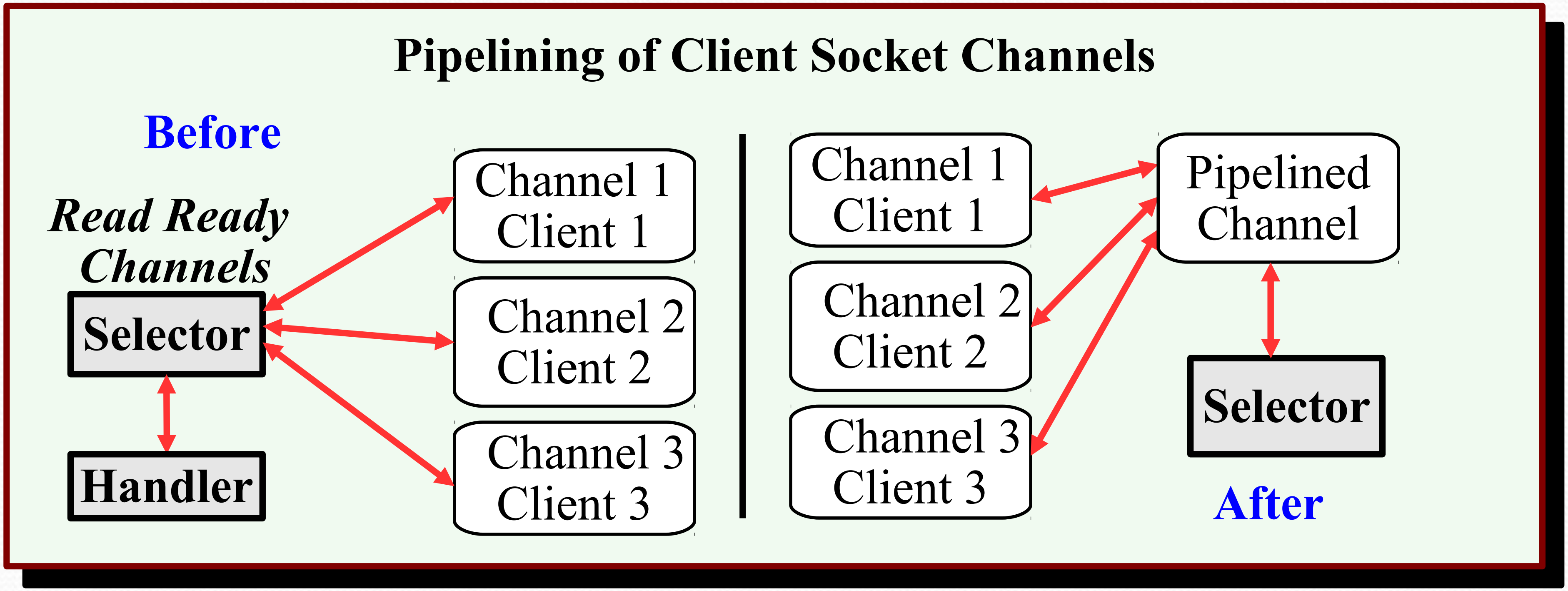
Data

Primary

Replica

Solution Approach





Outline

- Problem
- Research Goals
- Solution Approach
- Experimental Evaluation
- Related Work
- Conclusion/Future Work

Implementation

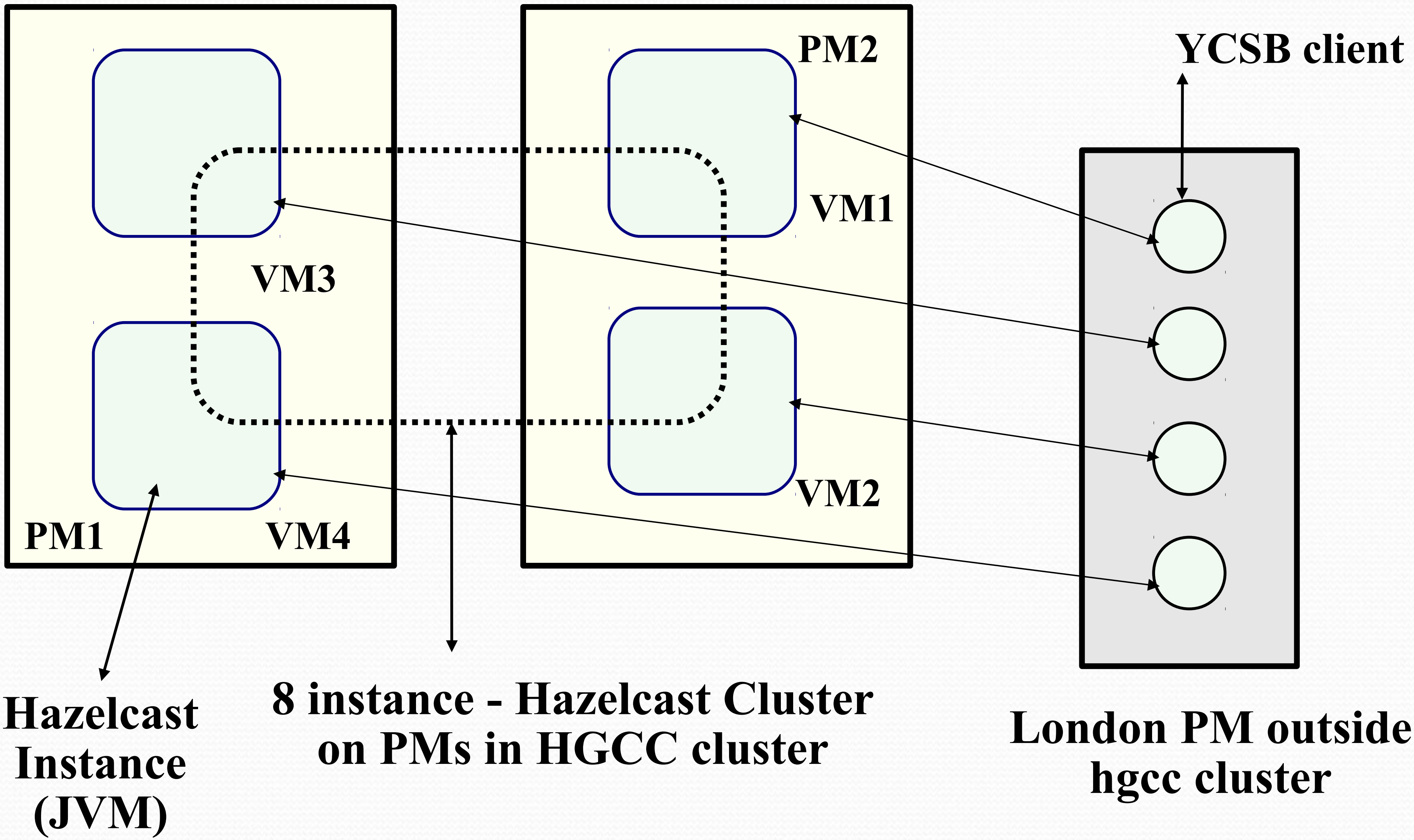
Software

- Java ⟷ ***Hazelcast Code***
- Netty Library Used
- YCSB – Yahoo Cloud Serving Benchmark ⟷ ***Evaluation***

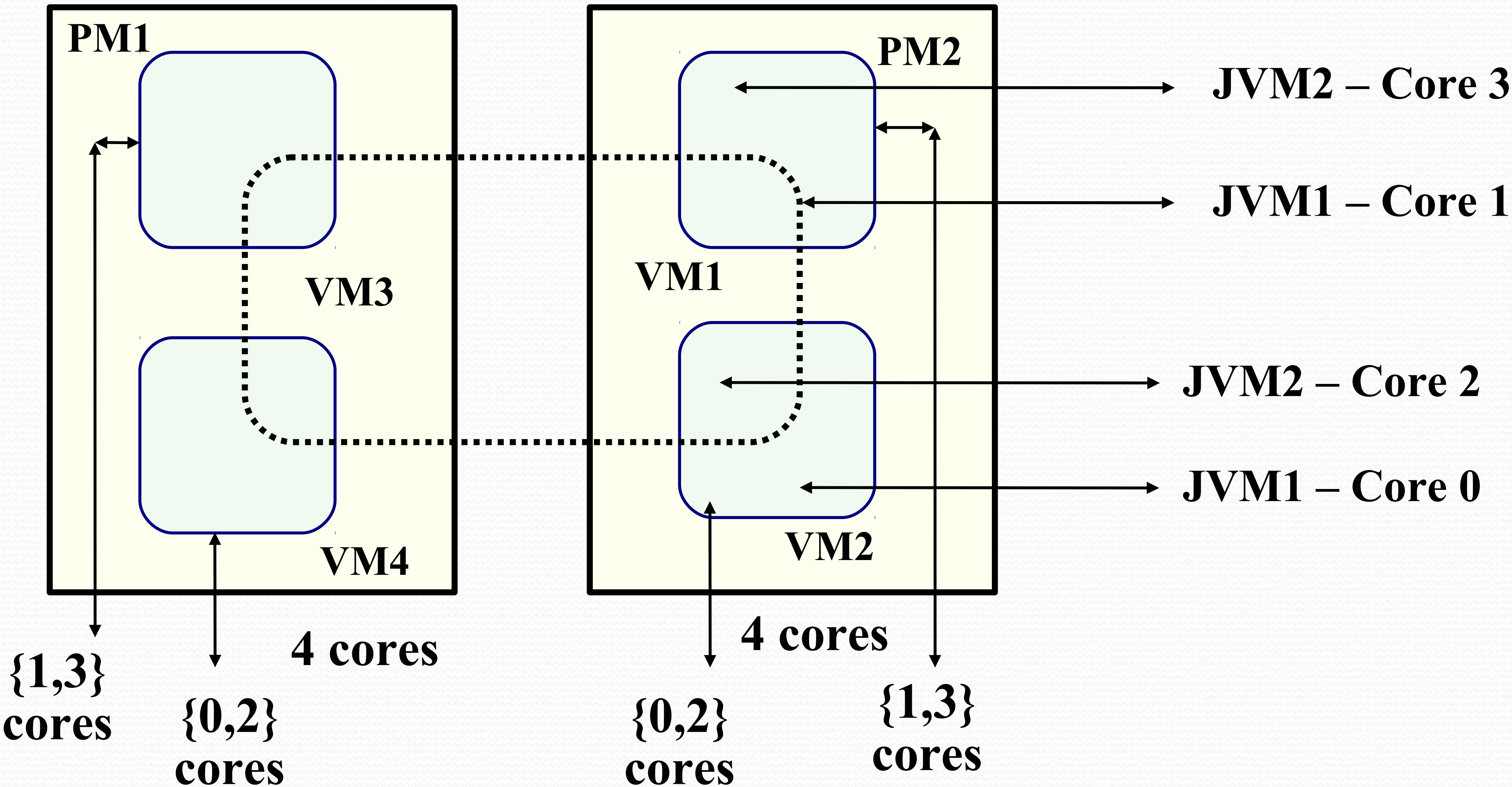
Hardware/OS Platform

- Nodes on HGCC Cluster
  - Quad-core Xeon 2.53GHz CPU, 8GB memory
- VMs running with Ubuntu 12.04 32bit with 4 GB memory and 2 vpcus on 2 HGCC nodes

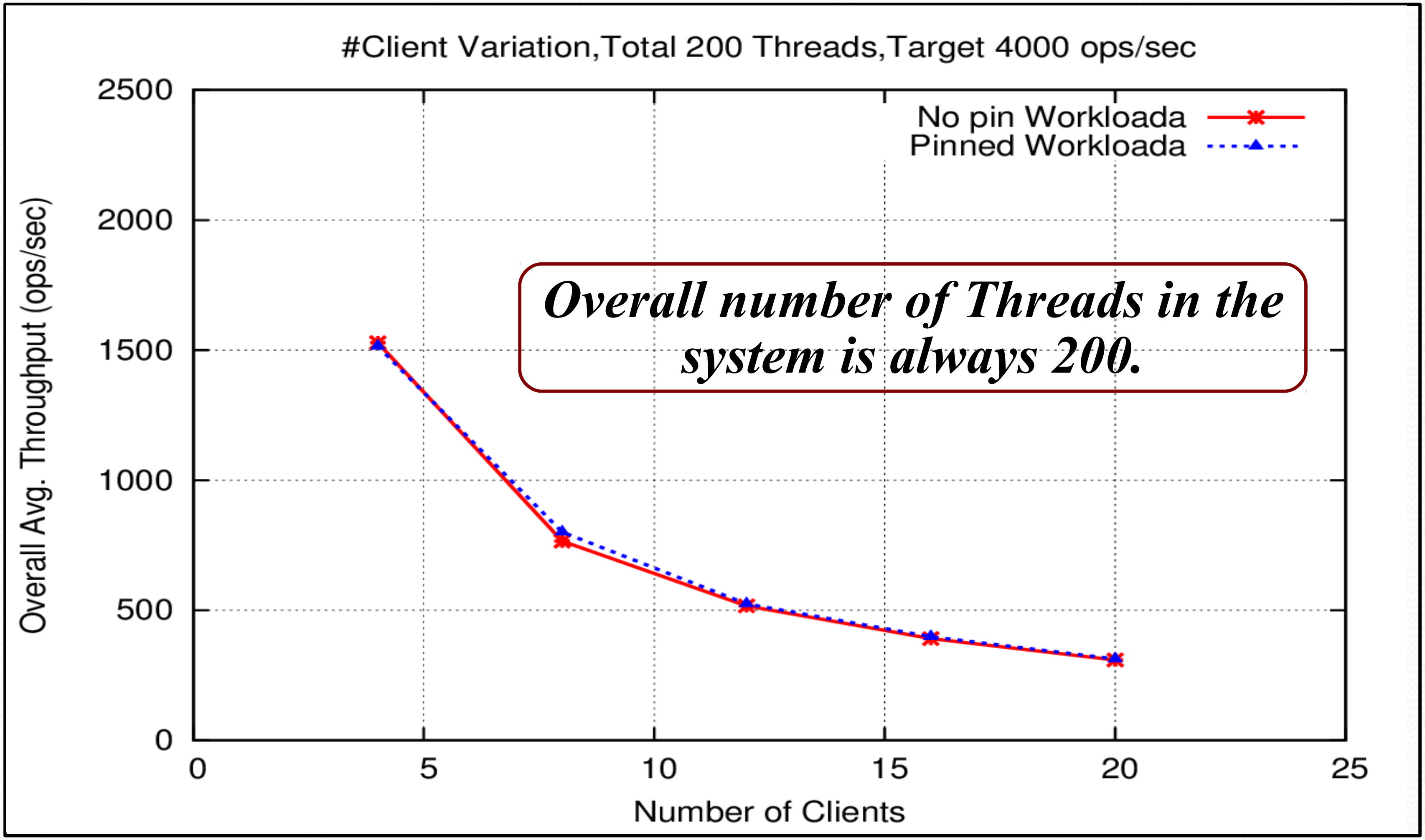
Experimental Set-Up



Set-Up with Pinning

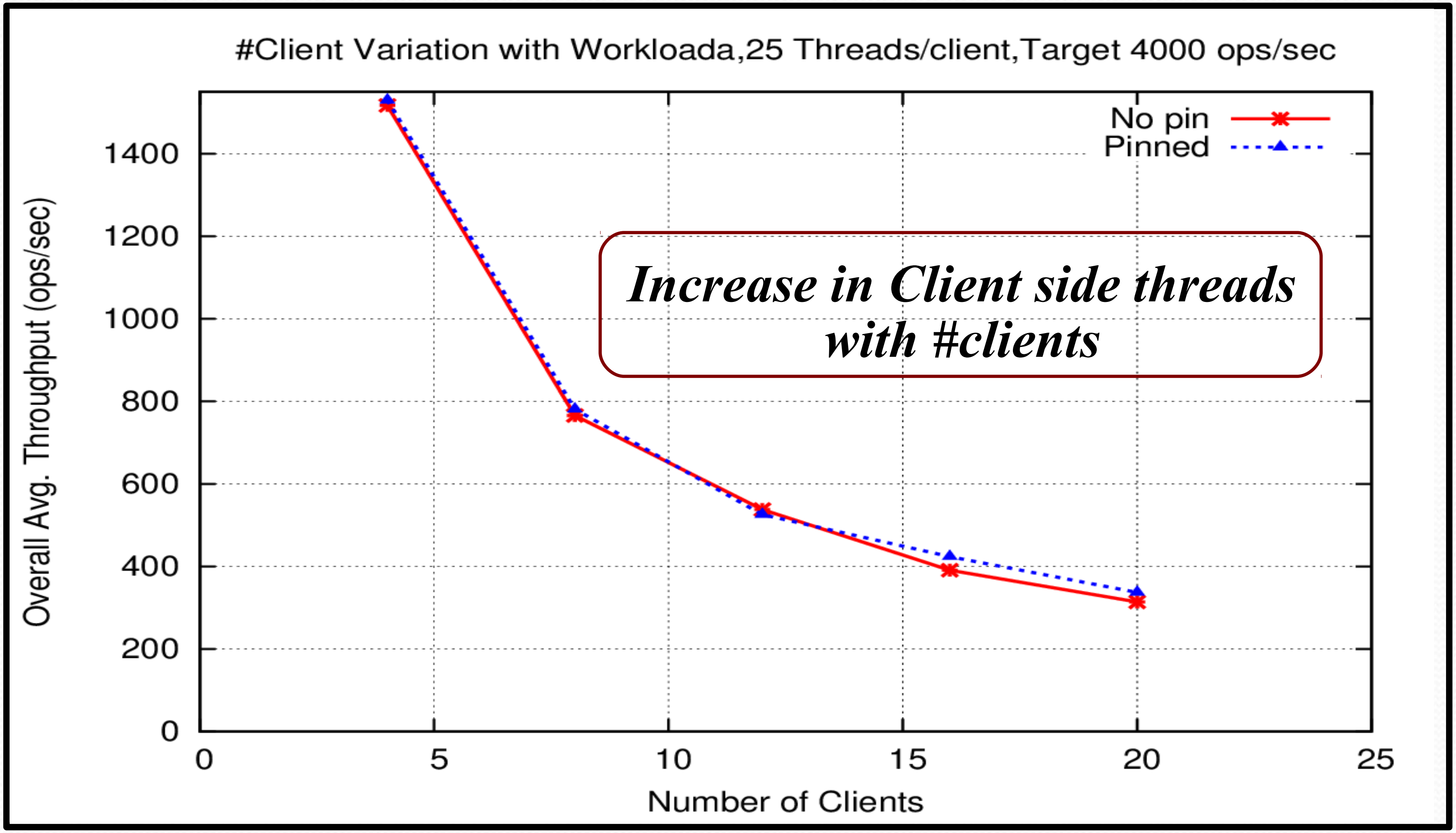


Client Count Variation



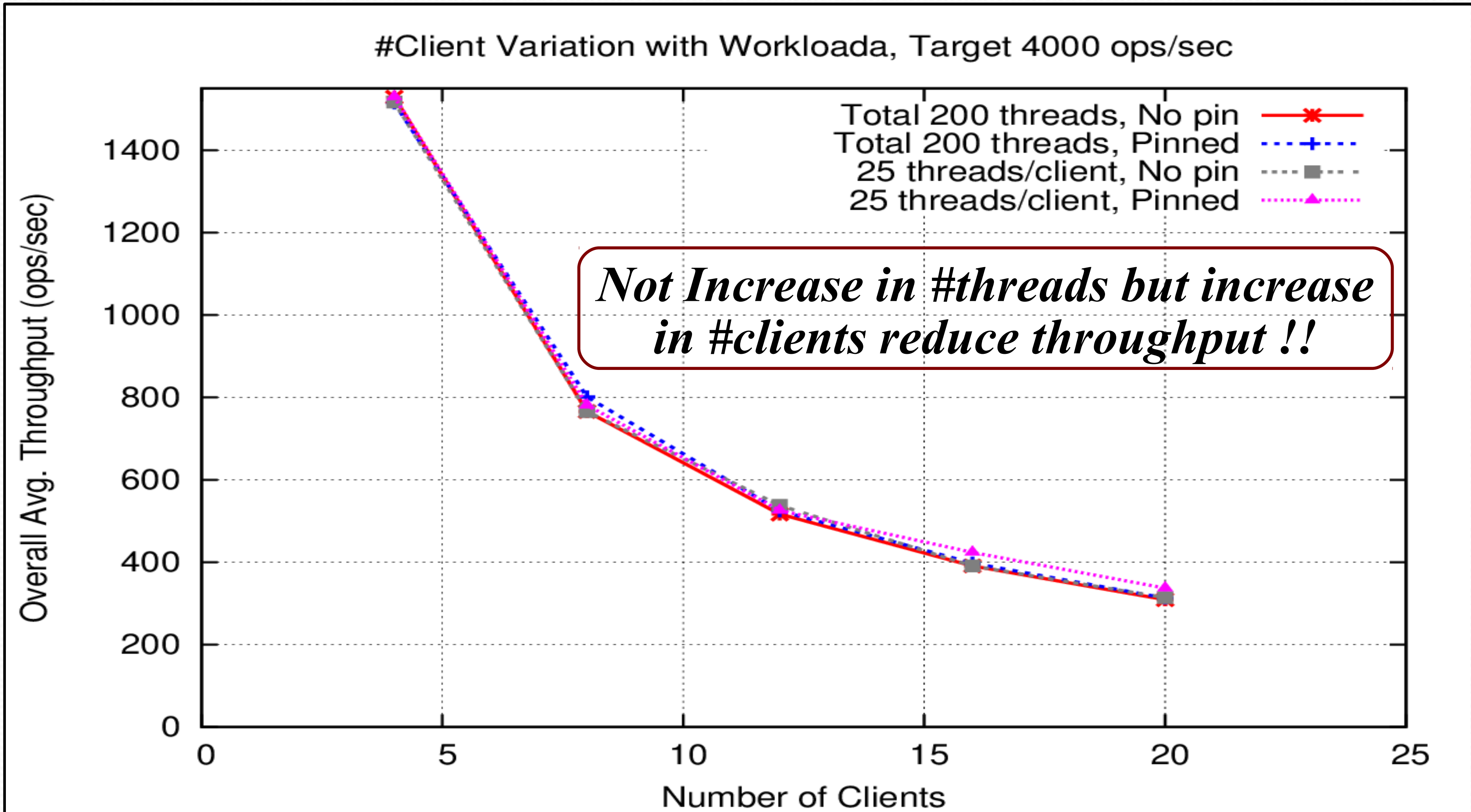
- ***Total number of client side threads used to generate workload fixed to 200, threads per client varies accordingly***
- ***No difference in performance between with and without pinning***

Client Count Variation



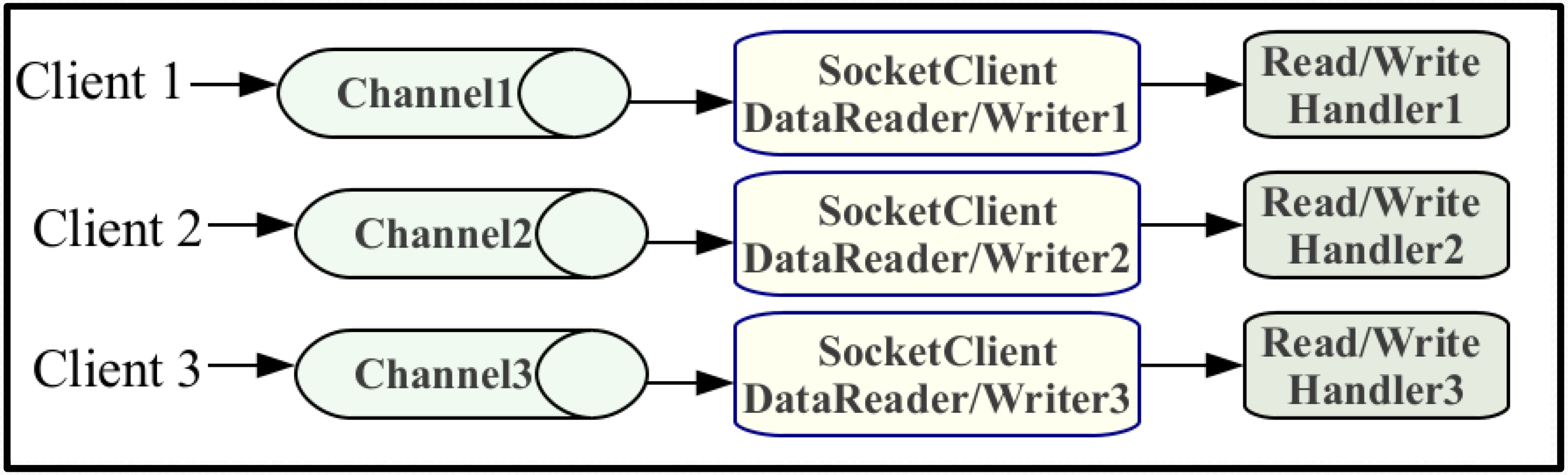
- ***Total number of client side threads used to generate workload increases with every client***
- ***No difference in performance between with and without pinning***

Client Count Variation

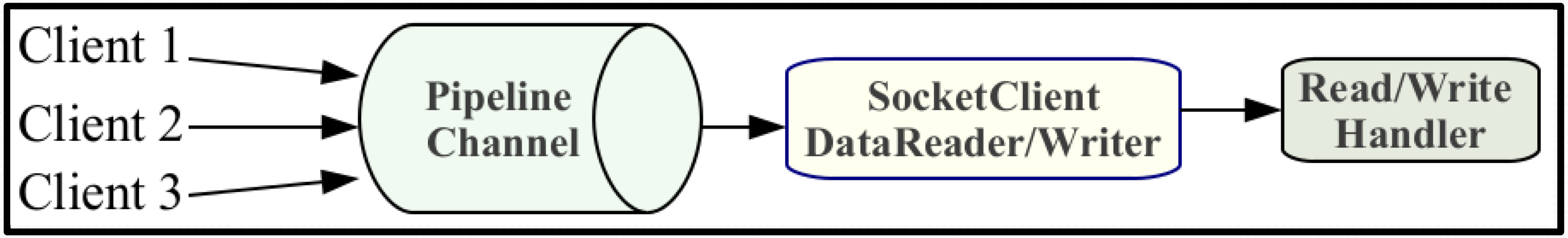


- ✓ *Pinning does not help in performance improvement*
- ✓ *Thread migration/Context Switches across cores is not high enough to affect performance*

Multiplexing Socket Channels

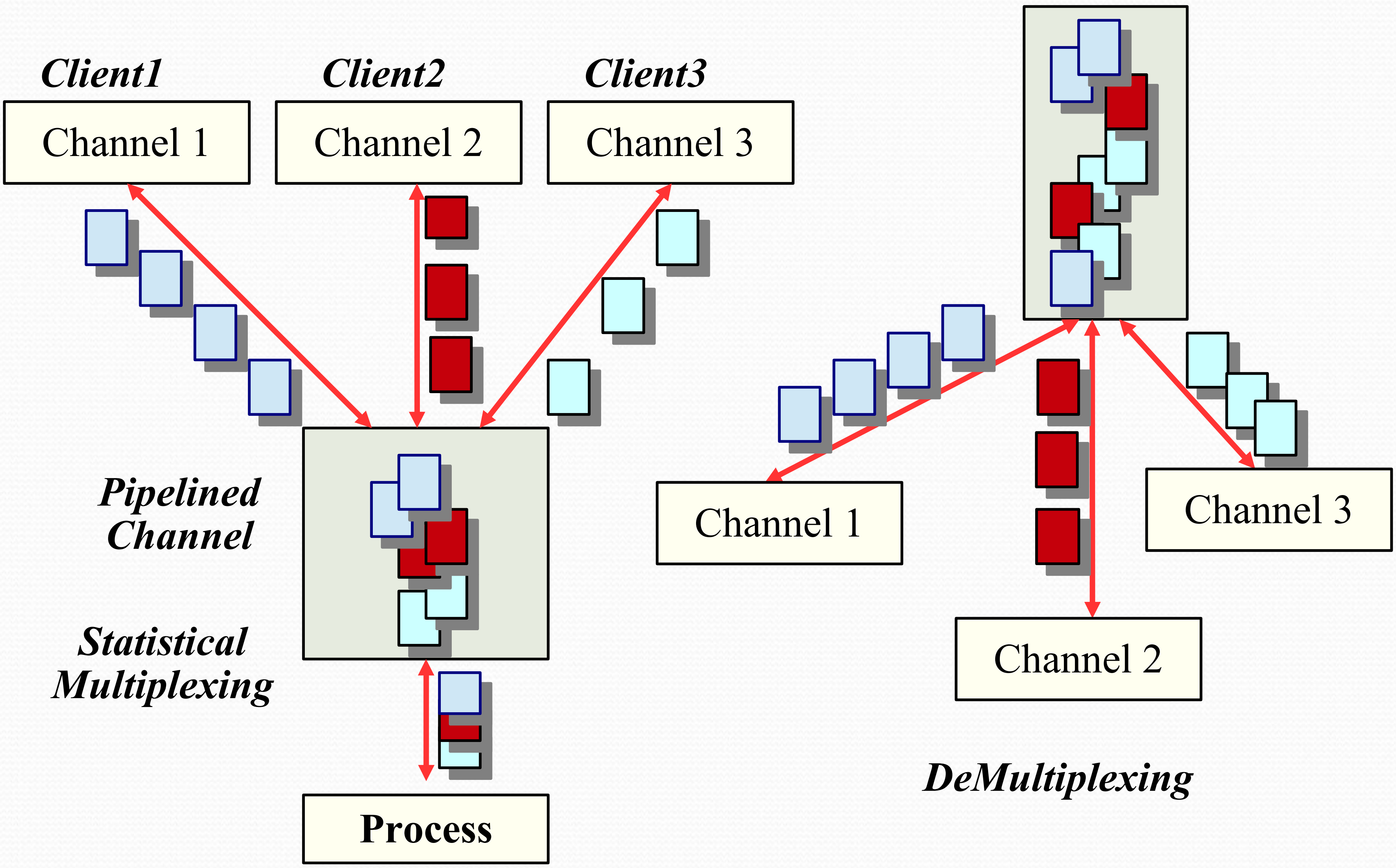


Before

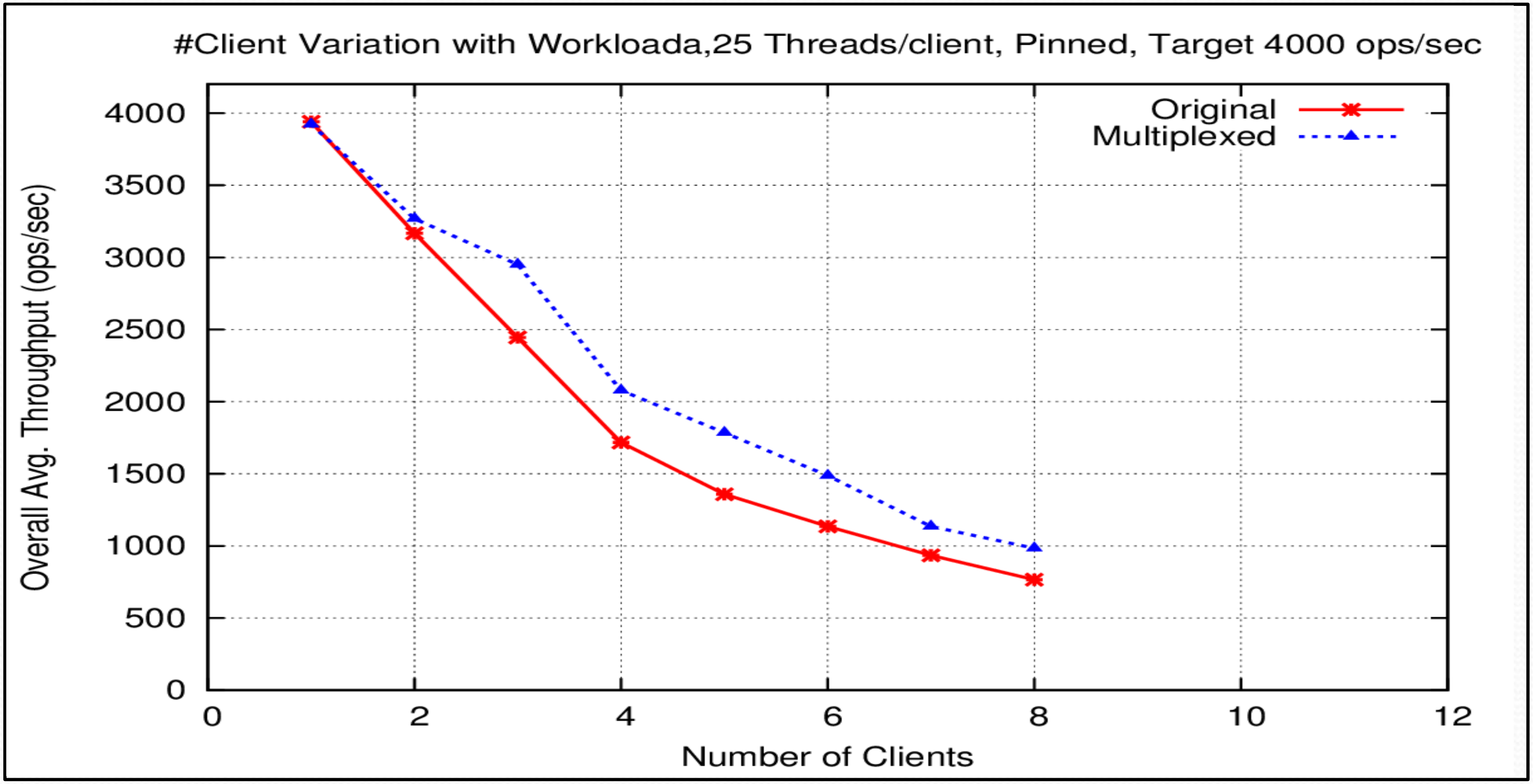


After

Multiplexing Socket Channels

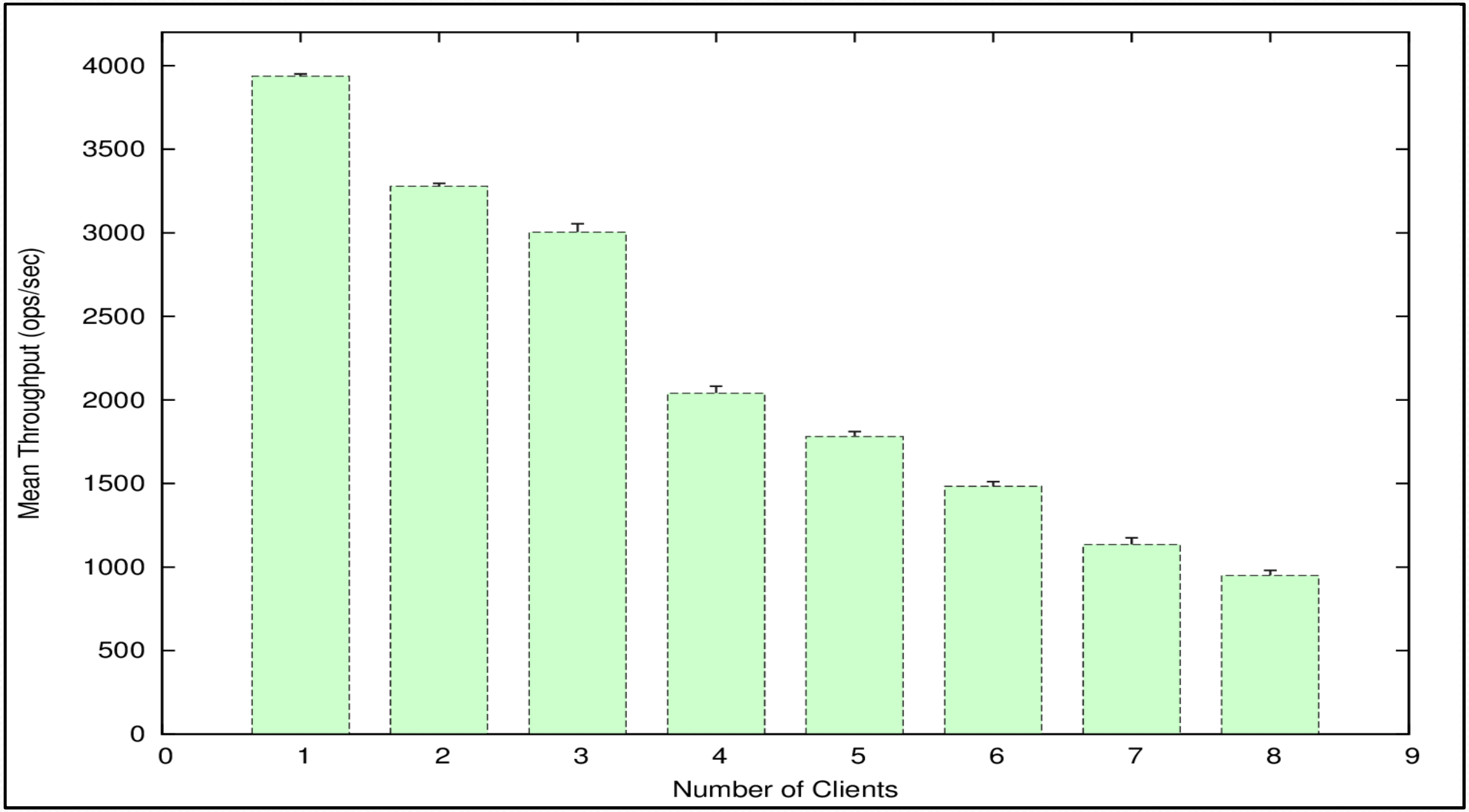


Multiplexing Socket Channels



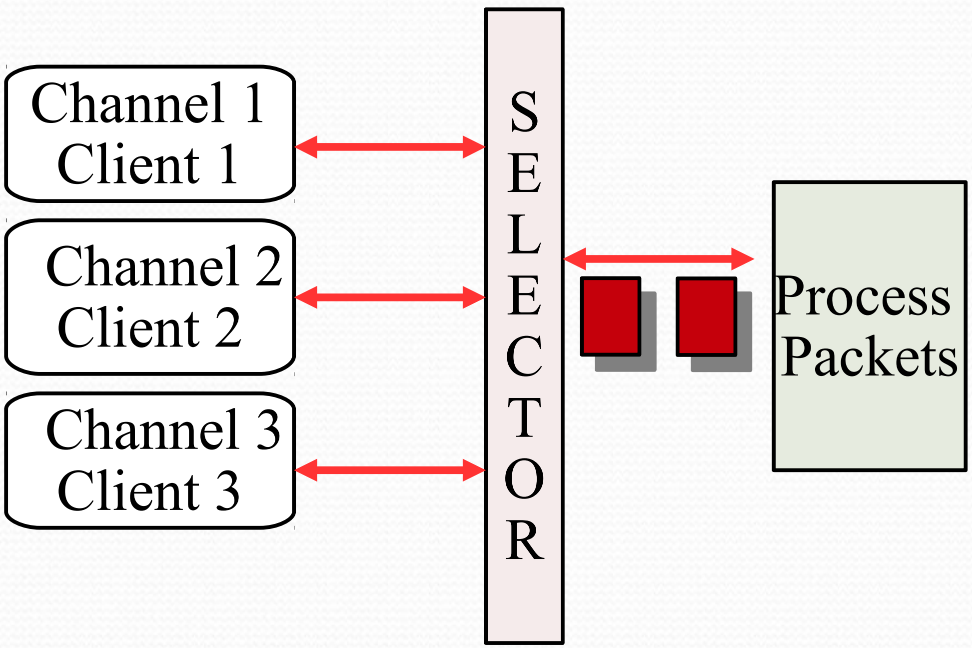
- *Performance Improvement*
- *% increase in throughput as high as 20% for 5 clients*

Multiplexing Socket Channels

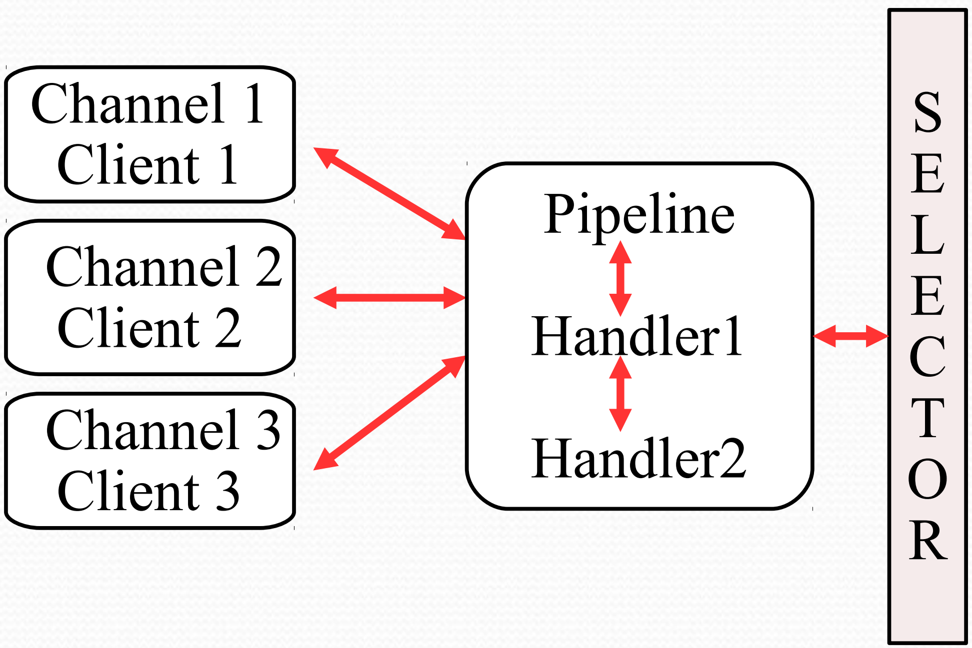


- ✓ ***Standard Deviation did not exceed 50***
- ✓ ***Less endpoint management, better performance***

Multiplexing Socket Channels

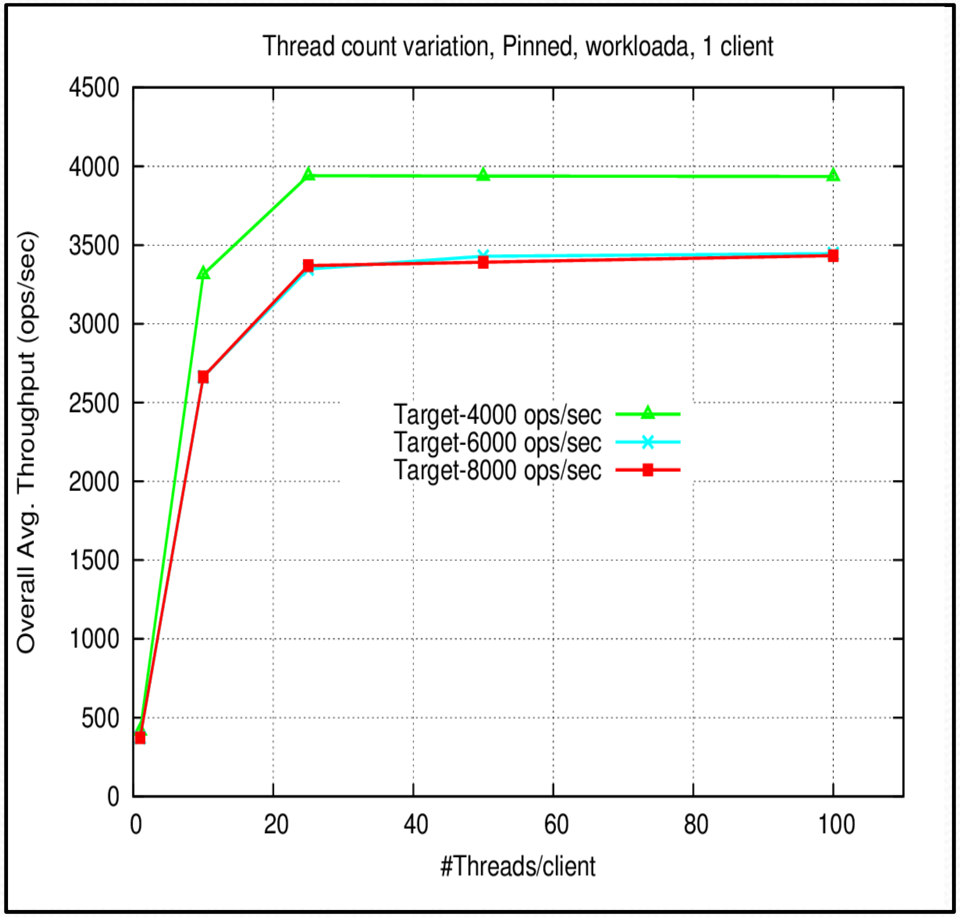


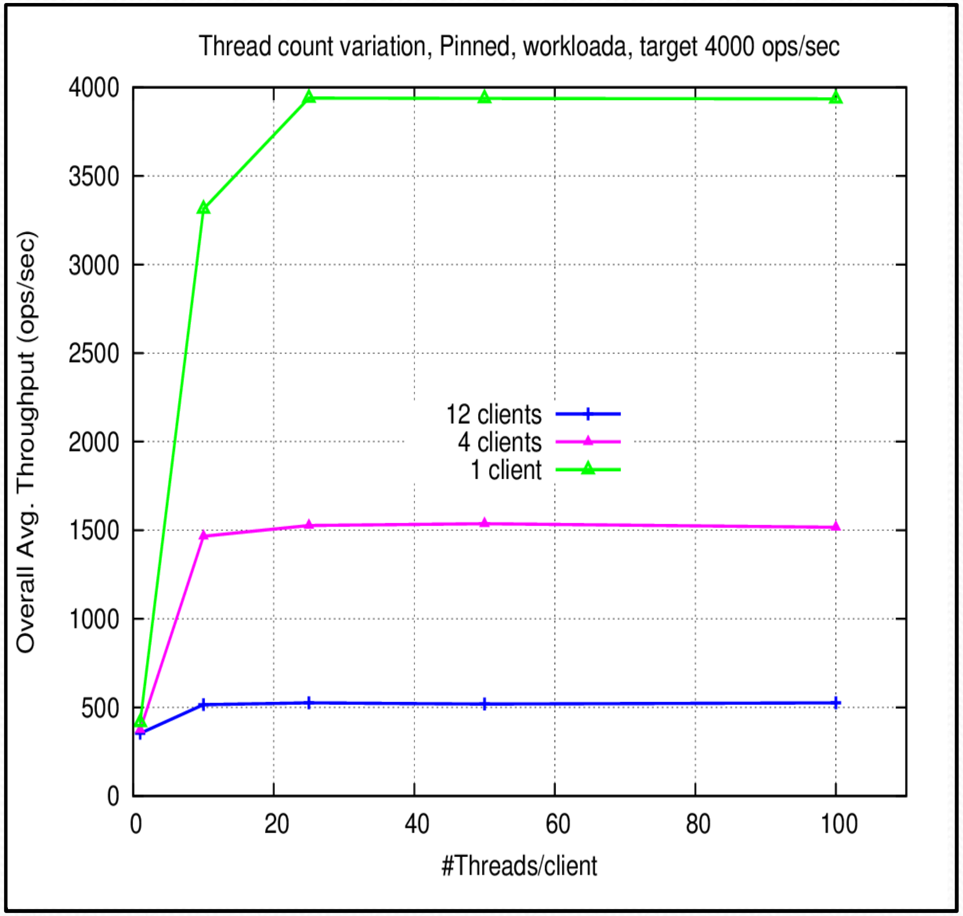
- *Select/Poll*
- *More Context Switches*
- *More Runnable Instances - higher threads per request*



- *Less Context Switches*
- *Less threads per request*
- *Better performance with concurrency*

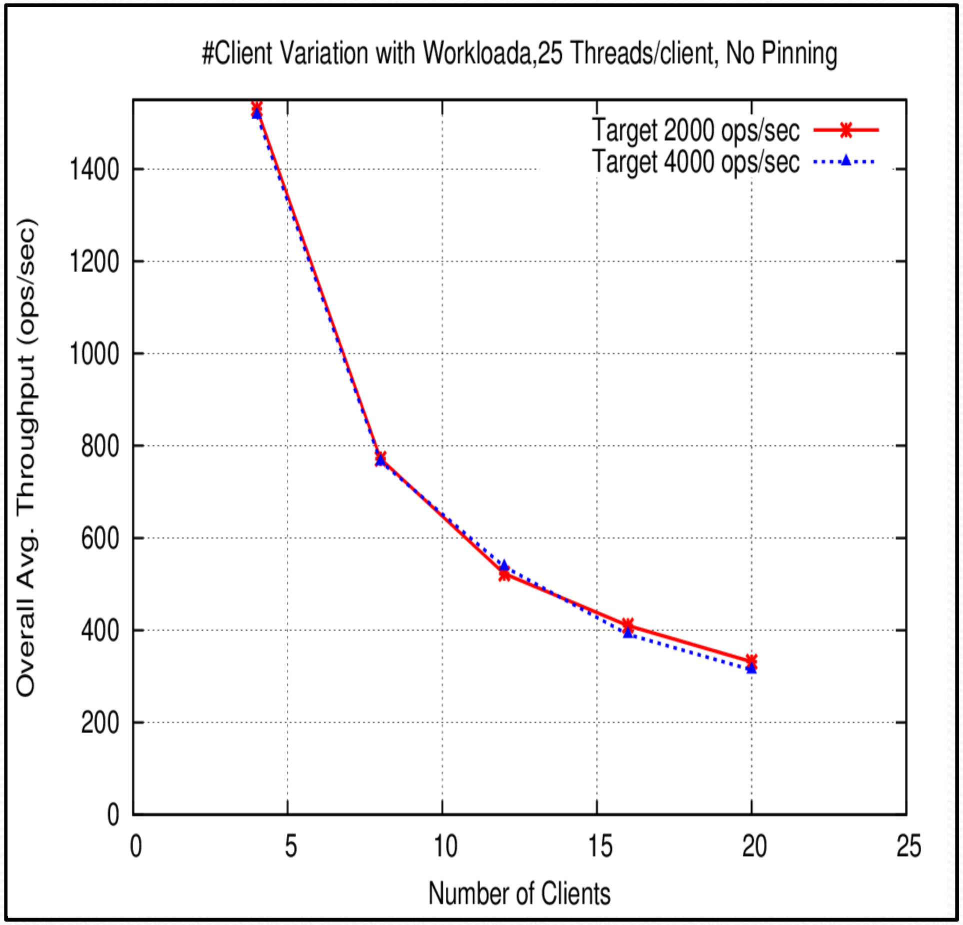
Client Thread Count Variation

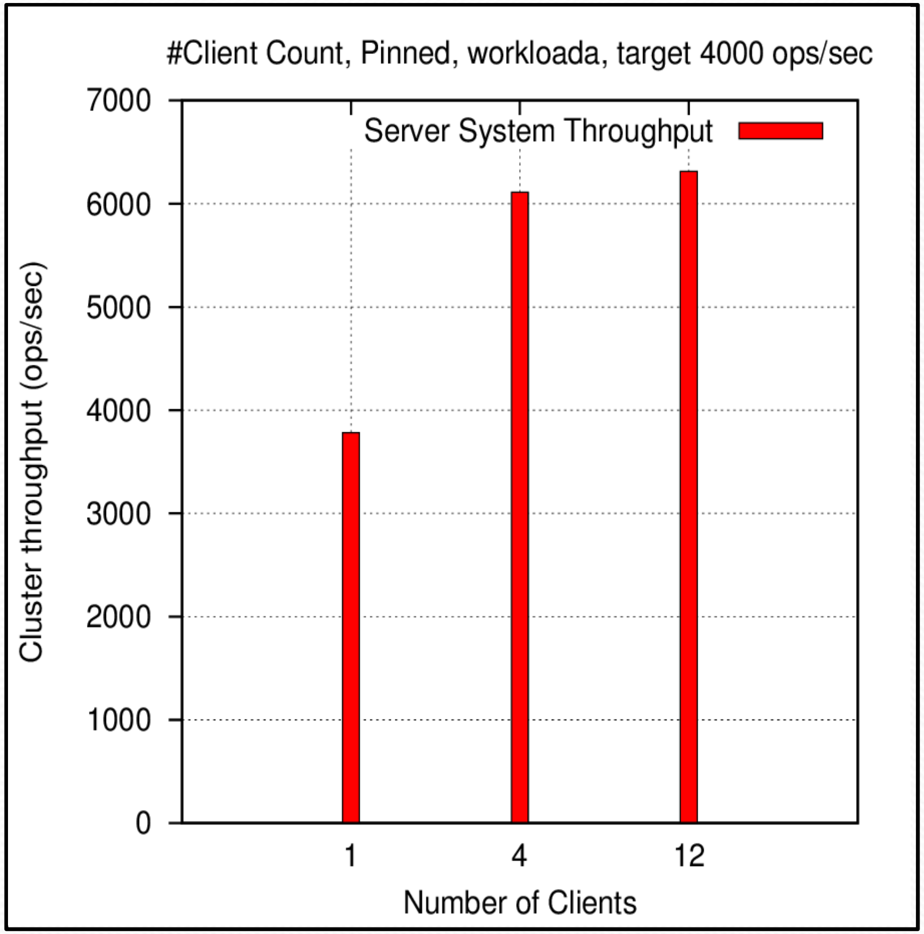




- ***Single client achieves target, beyond 4000 ops/sec performance drops.***
- ***With both 4 & 12 clients, system reaches maximum performance level at 25 threads/client beyond which there is no fluctuation.***

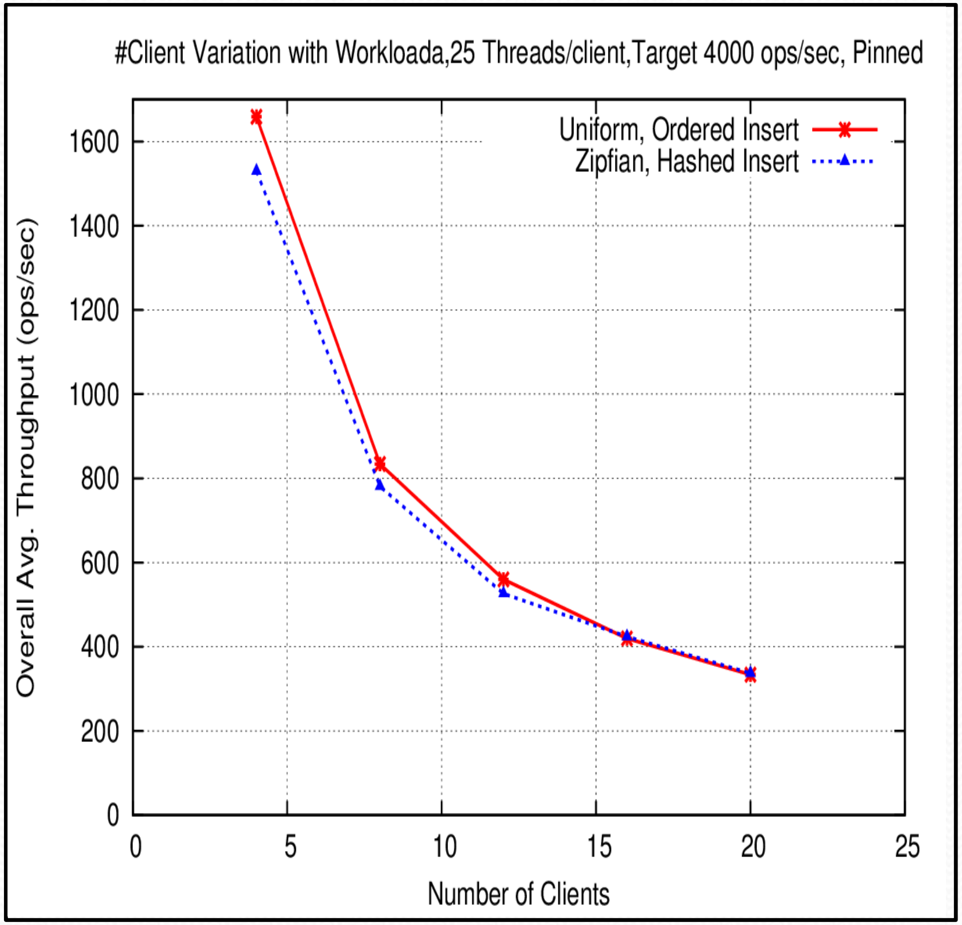
Target Throughput

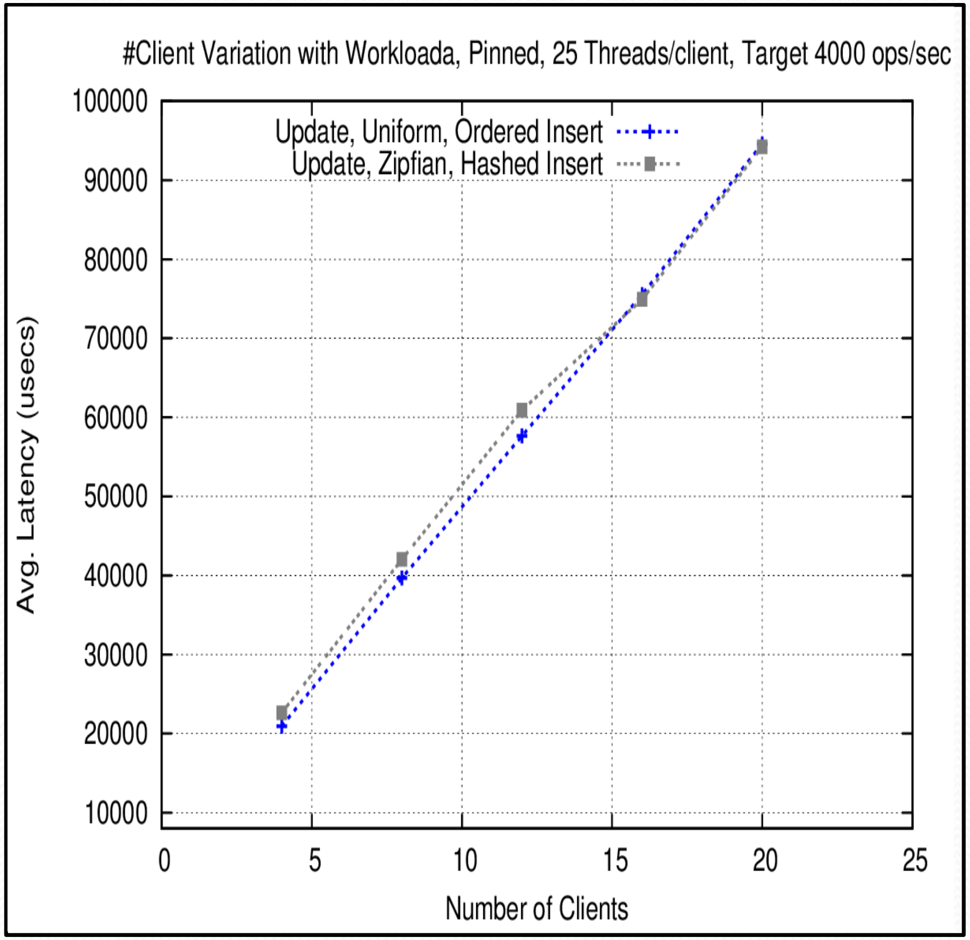




- ✓ ***Target doubled, no difference in throughput with the same #clients.***
- ✓ ***Cluster throughput from the server's perspective does not exceed 6300 ops/sec.***

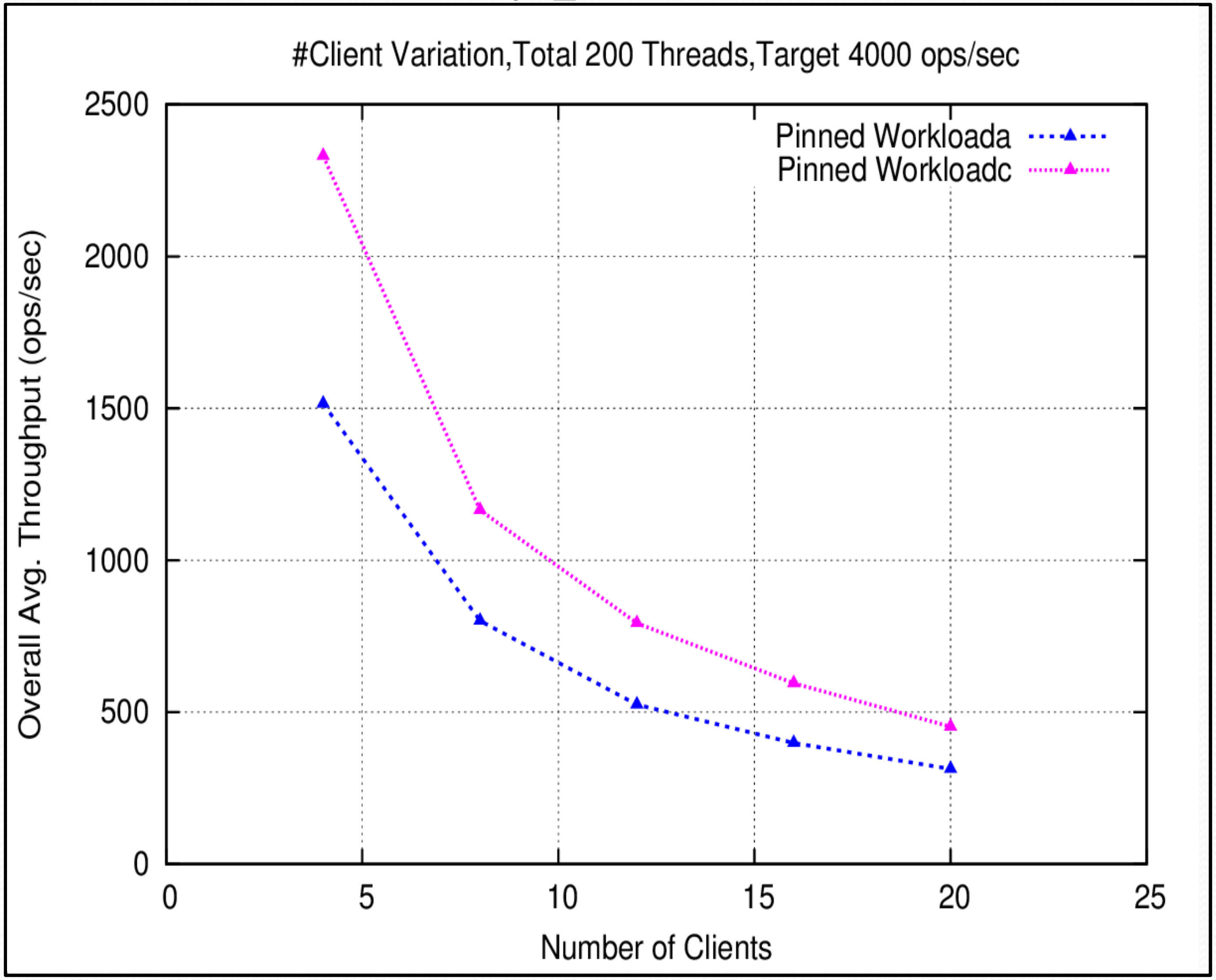
Insert Order & Distribution Type





Whether ordered or hashed inserts, irrespective of uniform or zipfian distribution, the performance degradation is similar.

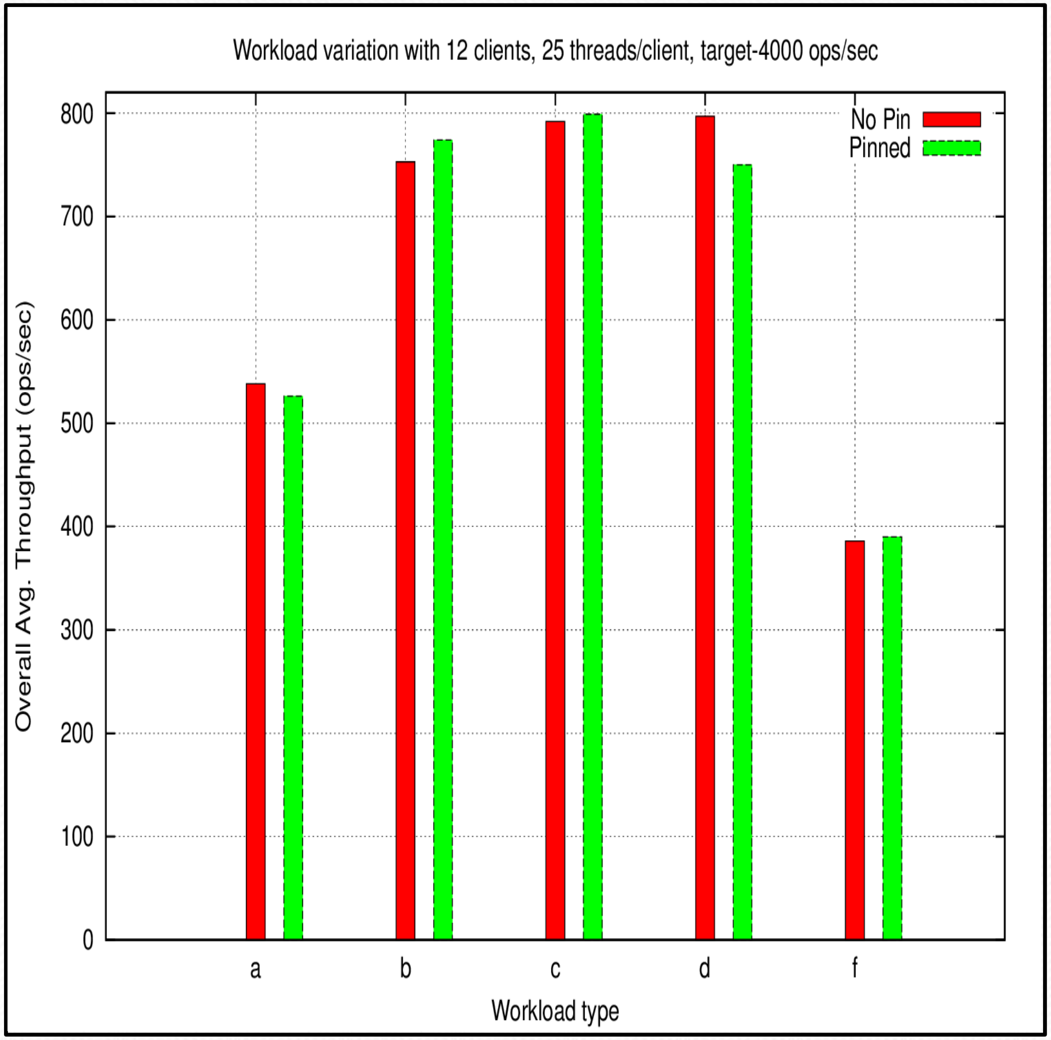
Workload Type Variation

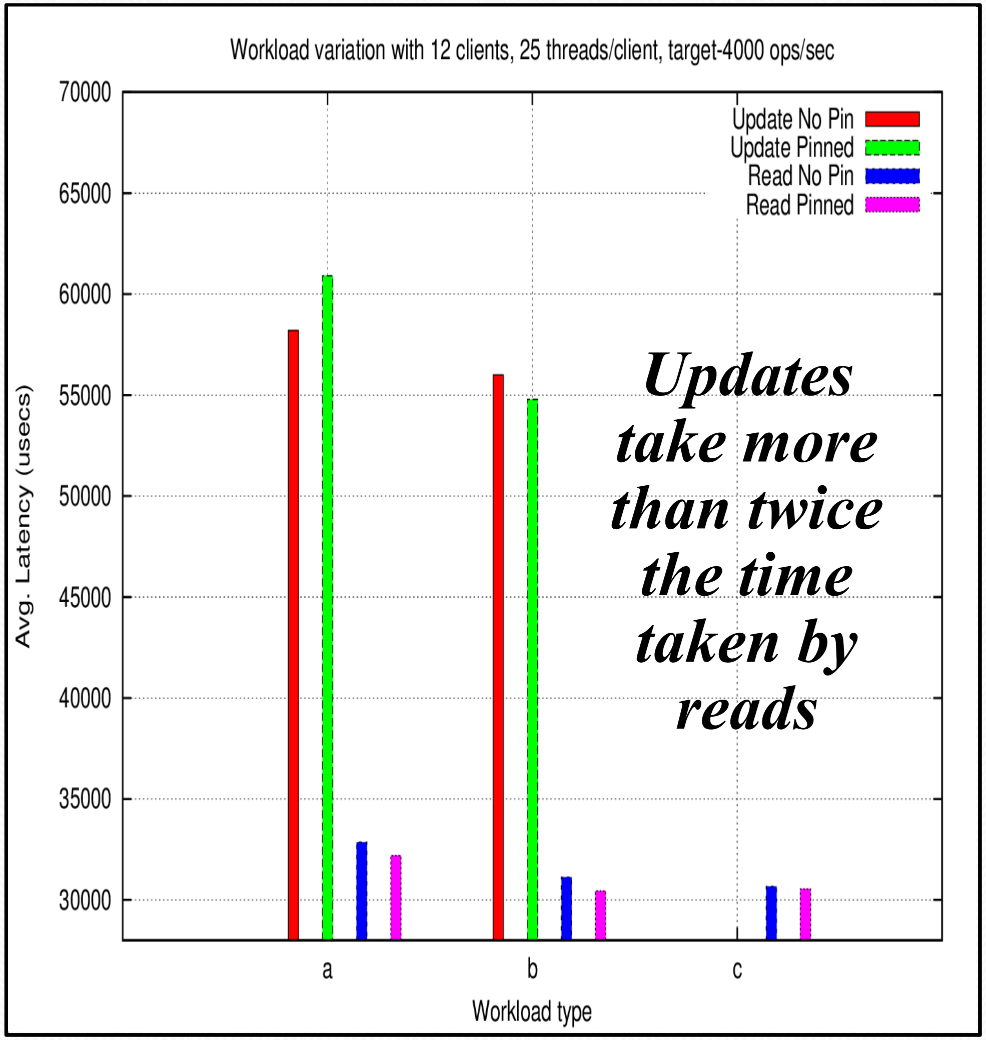


- Workloada – 50%read, 50%update
- Workloadb -
  - 95% read
  - 5% update
- Workloadc -
  - 100% read
- Workloadd -
  - 95% read
  - 5% insert
- Workloadf -
  - 50% read
  - 50% read-modify-write

Even read-only workload suffers throttled throughput.

Workload Type Variation





More updates, lower the throughput, read-modify-write costlier than update.

Outline

- Problem
- Research Goals
- Solution Approach
- Experimental Evaluation
- Related Work
- Conclusion/Future Work

Related Work

- Centralized Solutions
  - Pisces, Borg, MROrchestrator, HDFS based HFS,
  - Changed scheduling logic

- Decentralized Solutions
  - Apollo, Sparrow, Omega, Mercury
  - Comprehensive scheduler considering diverse heterogenous workloads

- Two-level schedulers
  - Cake, Mesos, Yarn
  - Focus on Hadoop Style Applications

- Cache Replacement Schemes, Cluster Load balancing
  - Gdwheel, Camp, MBal, Scads

- How to process data through nultiple entry points ?
  - Contention elimination in Multi-tenant data grids
  - Another perspective of solving the problem

Outline

- Problem
- Research Goals
- Solution Approach
- Experimental Evaluation
- Related Work
- Conclusion/Future Work

Concluding Remarks

- Performance degradation is clear with increasing clients.
- JVM-CPU Pinning does not help in improving performance.
- Beyond a threshold and a specific target, increasing the number of client threads do not affect performance.
- Insert Order and Distribution Type have no tangible affect at-least if all clients have similar transaction type.
- Statistical multiplexing of client socket channels improve overall throughput.

Future Work

- Investigate efficient ways to process requests through multiple entry points.

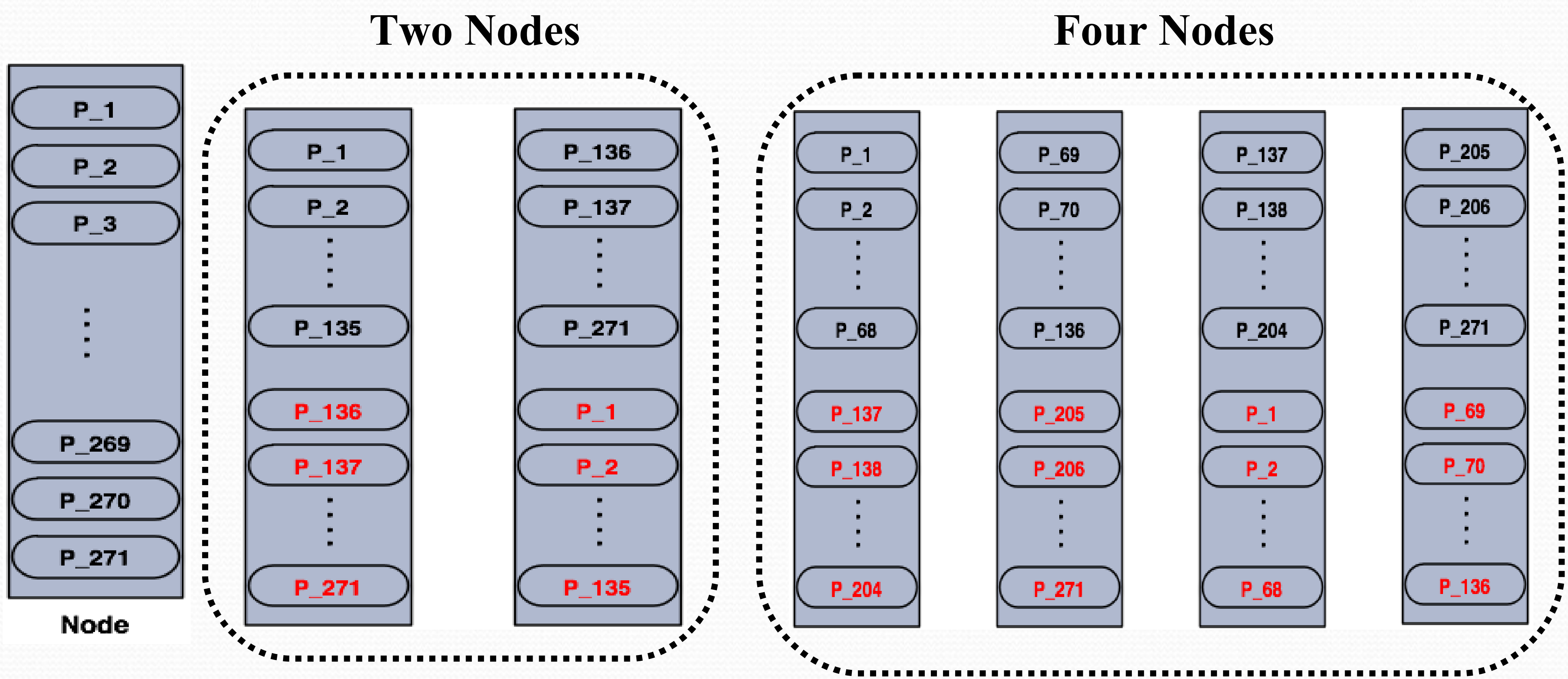
- Considering the scale of cluster, can performance be improved through multiplexing, avoiding excessive parallelization ? What are the major limitations ?

- Diagnose *failures* in such systems which affect performance.

Thank You

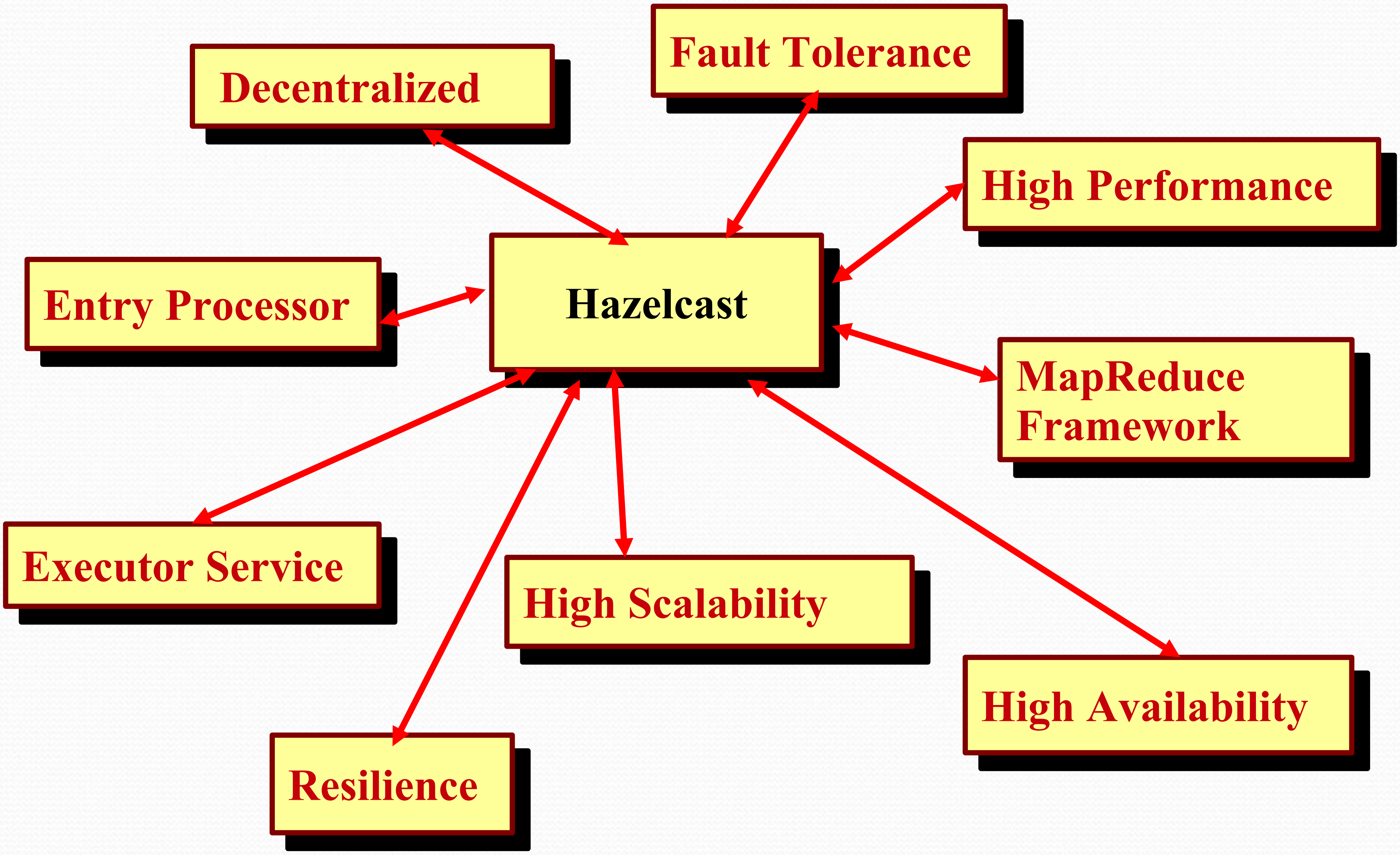
Backup Slides

Architecture

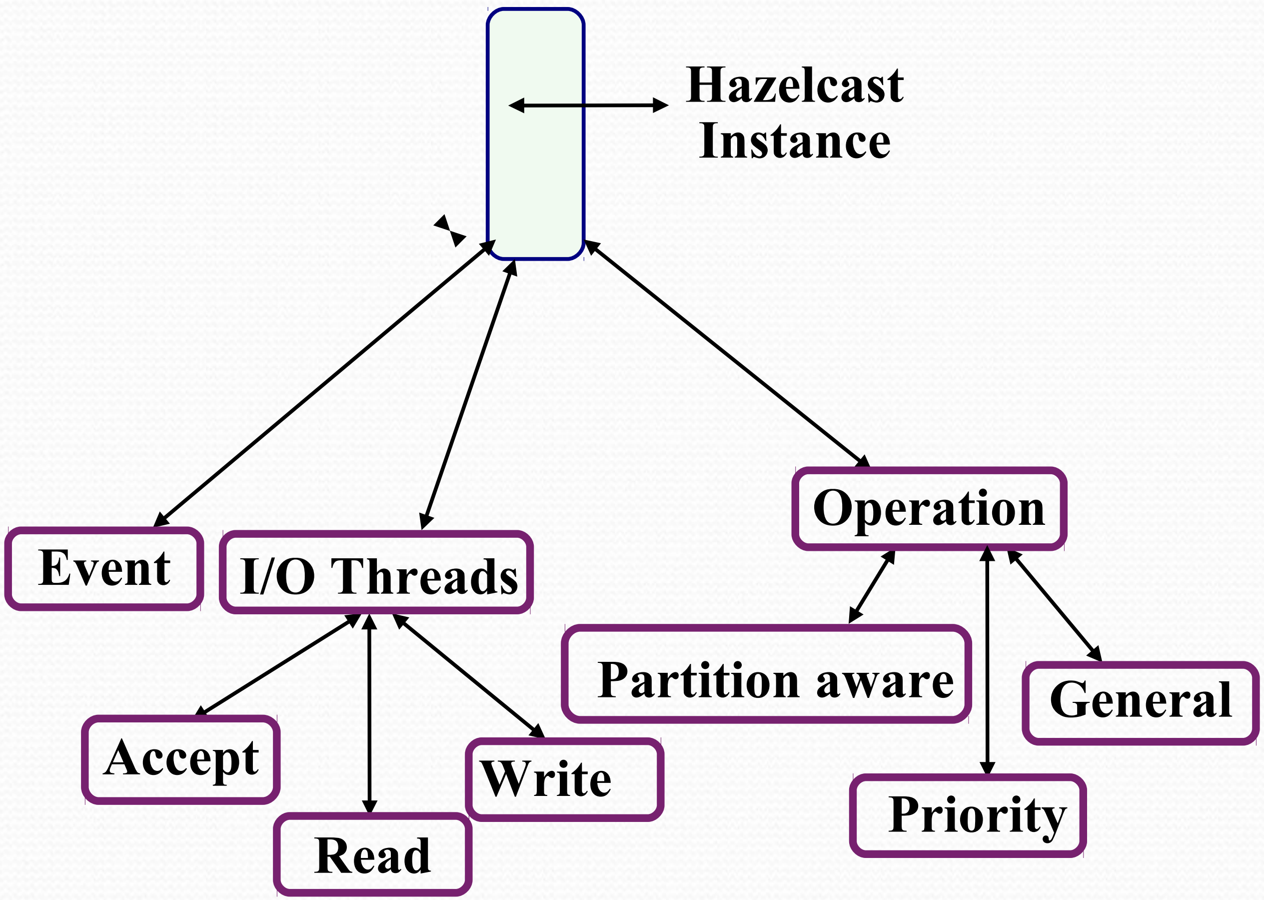


Multiple Instances of Hazelcast cluster in 1 physical Node or across multiple physical nodes possible.

Hazelcast Features



Internal Threading Model

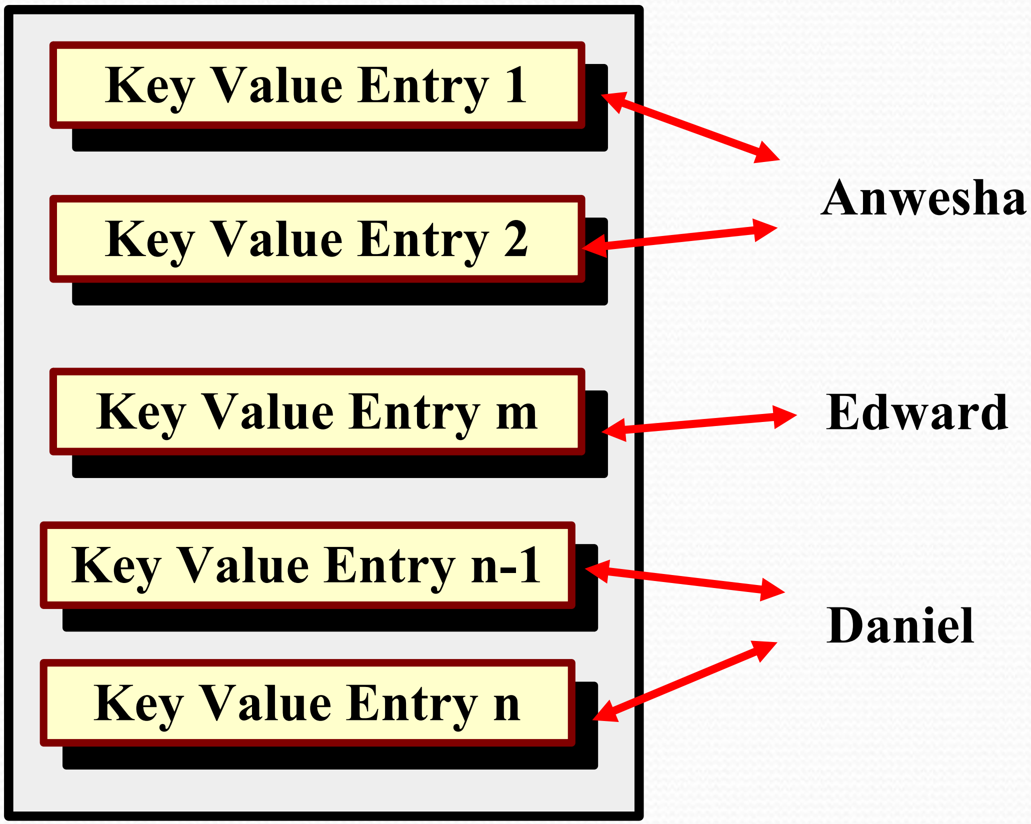


Each thread has its own queue.

Default counts:

- Event Threads – 5
- Work Queue capacity - 1000000
- I/O Threads – 7
  - 1 accept
  - 3 read
  - 3 write
- Operation Threads - CPU core count * 2
- Priority Queue
- Iexecutor Threads -
- Executor Service – 16
- Query operator -
- ExecutorPoolSize – 5 * hzClient CPU core count

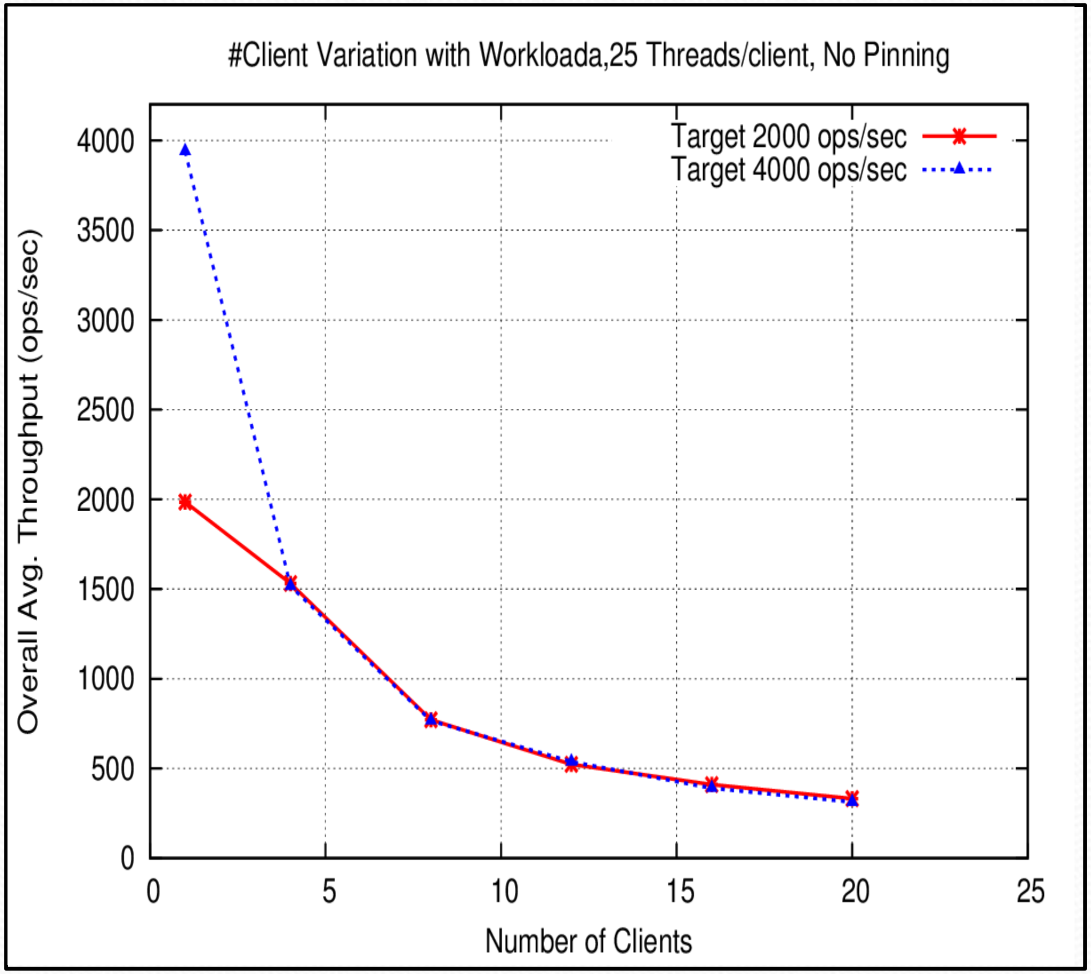
Java Client Protocol



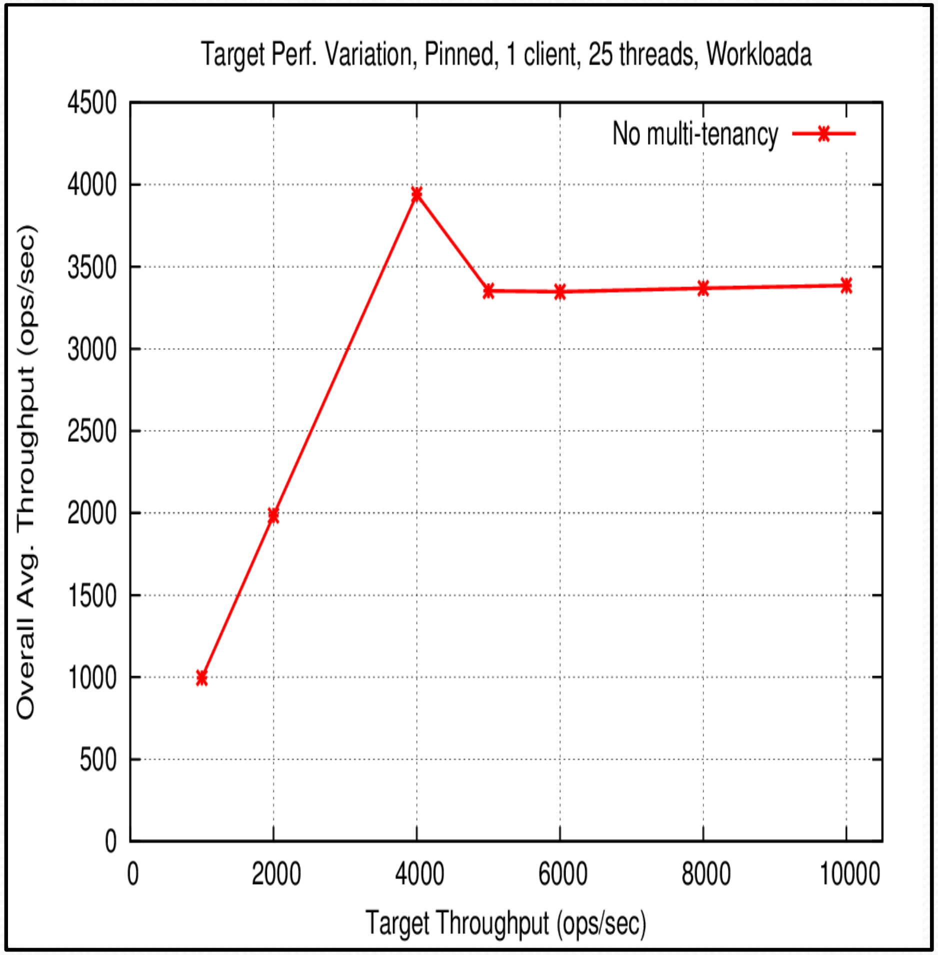
Shared Distributed Map A across multiple clients

- ✓ Open Connection
- ✓ Tenant Authentication
- ✓ Retrieve Partition List
- ✓ Send Operation Messages
- ✓ Receive Responses
- ✓ Cluster Member Updates
- ✓ Close Connection

With Single Client (No Contention)

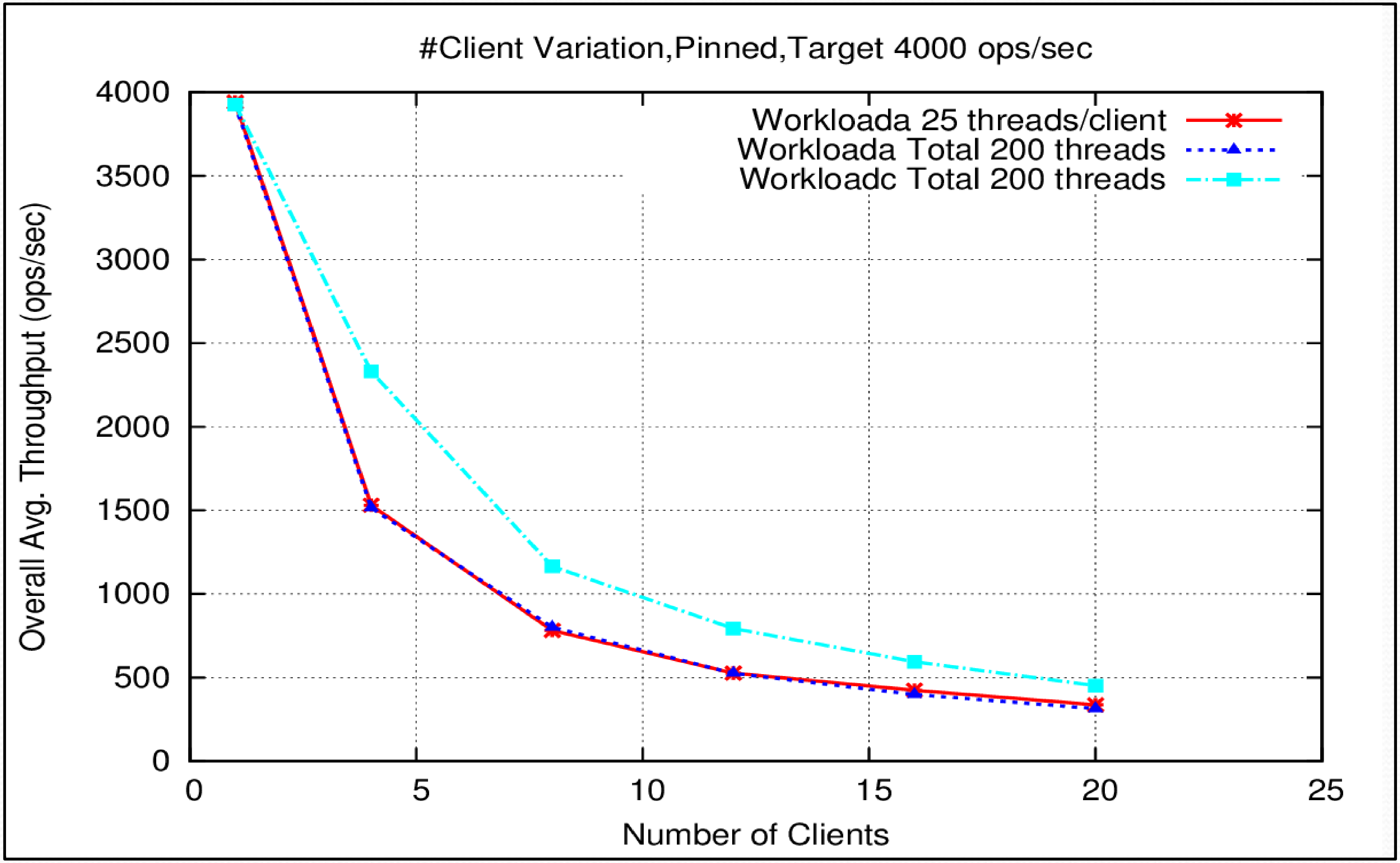


With even higher targets, multi-tenancy will give no better performance



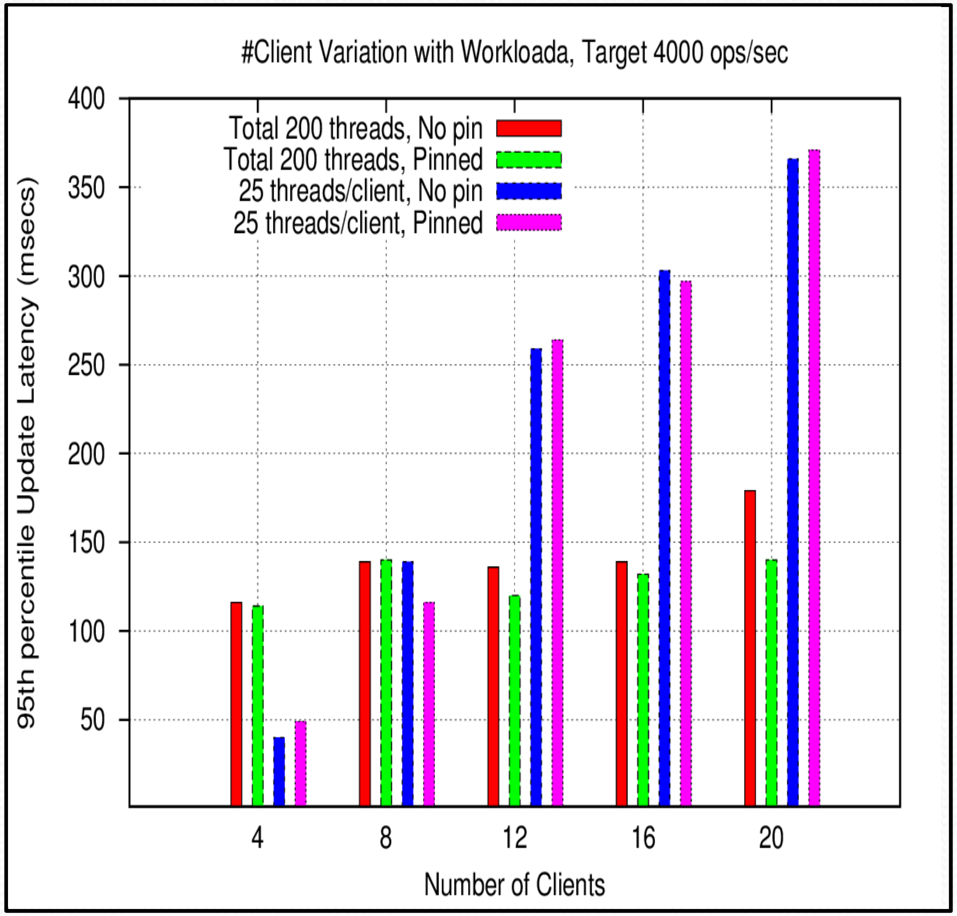
Threshold – 4000 ops/sec

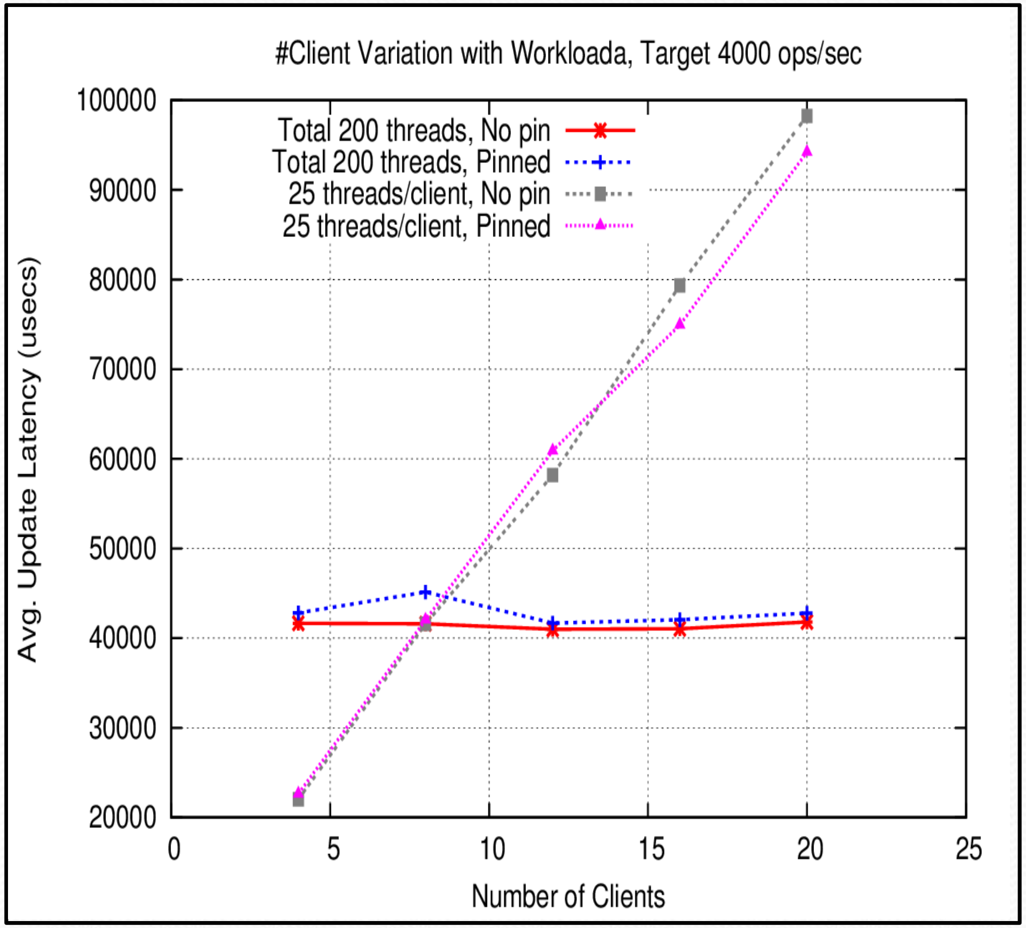
With Single Client (No Contention)



High Drop from 1 client to 4 client, Read-only better throughput !!

Client Count Variation





- ***Well distributed workload, better overall response time***
- ***Excessive parallelization w.r.t target increases latency***