Scalable but Wasteful:
Current State of Replication in the Cloud

Venkata Swaroop Matte
The Pennsylvania State University

Aleksey Charapko
University of New Hampshire

Abutalib Aghayev
The Pennsylvania State University

Strongly Consistent Replication

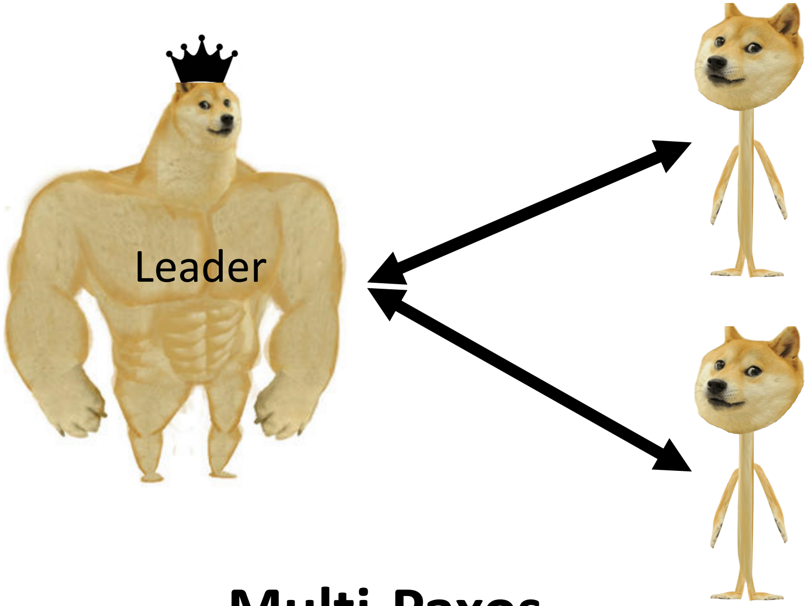
- Used in Cloud datastores and Configuration management
- Rely on Consensus protocols (replication protocols)
- Achieve High-throughput

Cockroach DB

Google Cloud Spanner

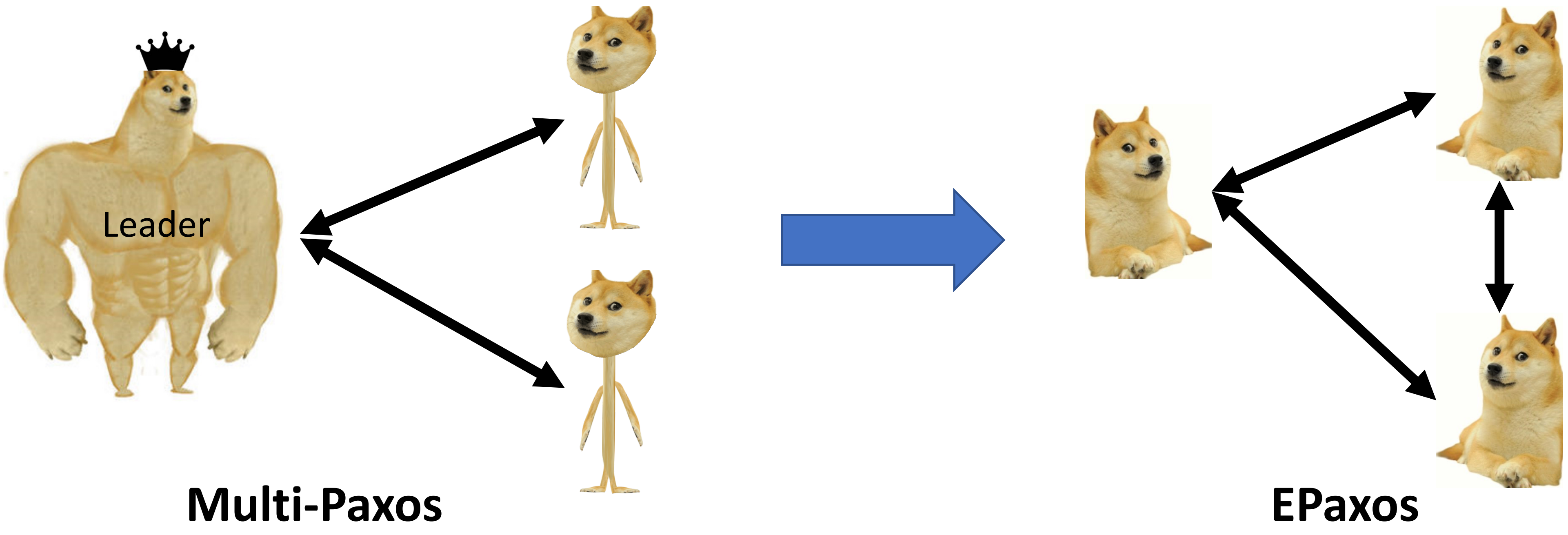
APACHE ZooKeeper™

How to optimize for Throughput?



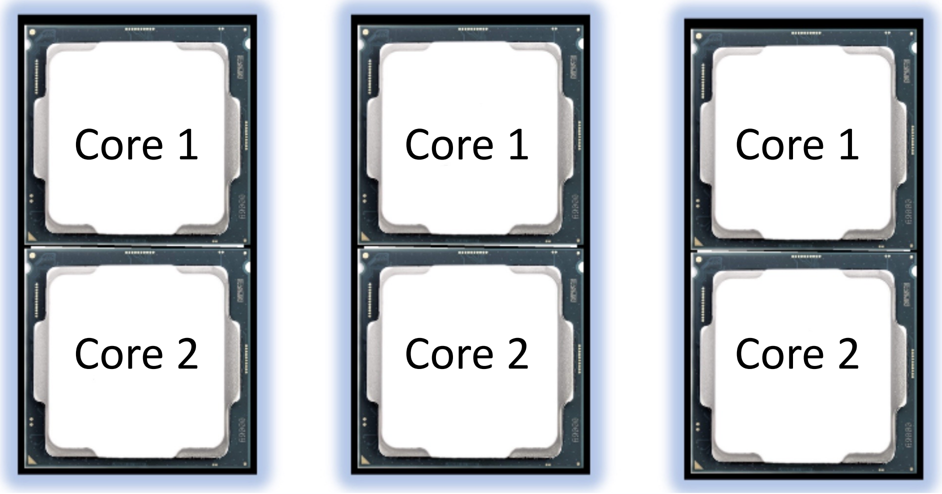
Multi-Paxos

One way to optimize: Shift the load



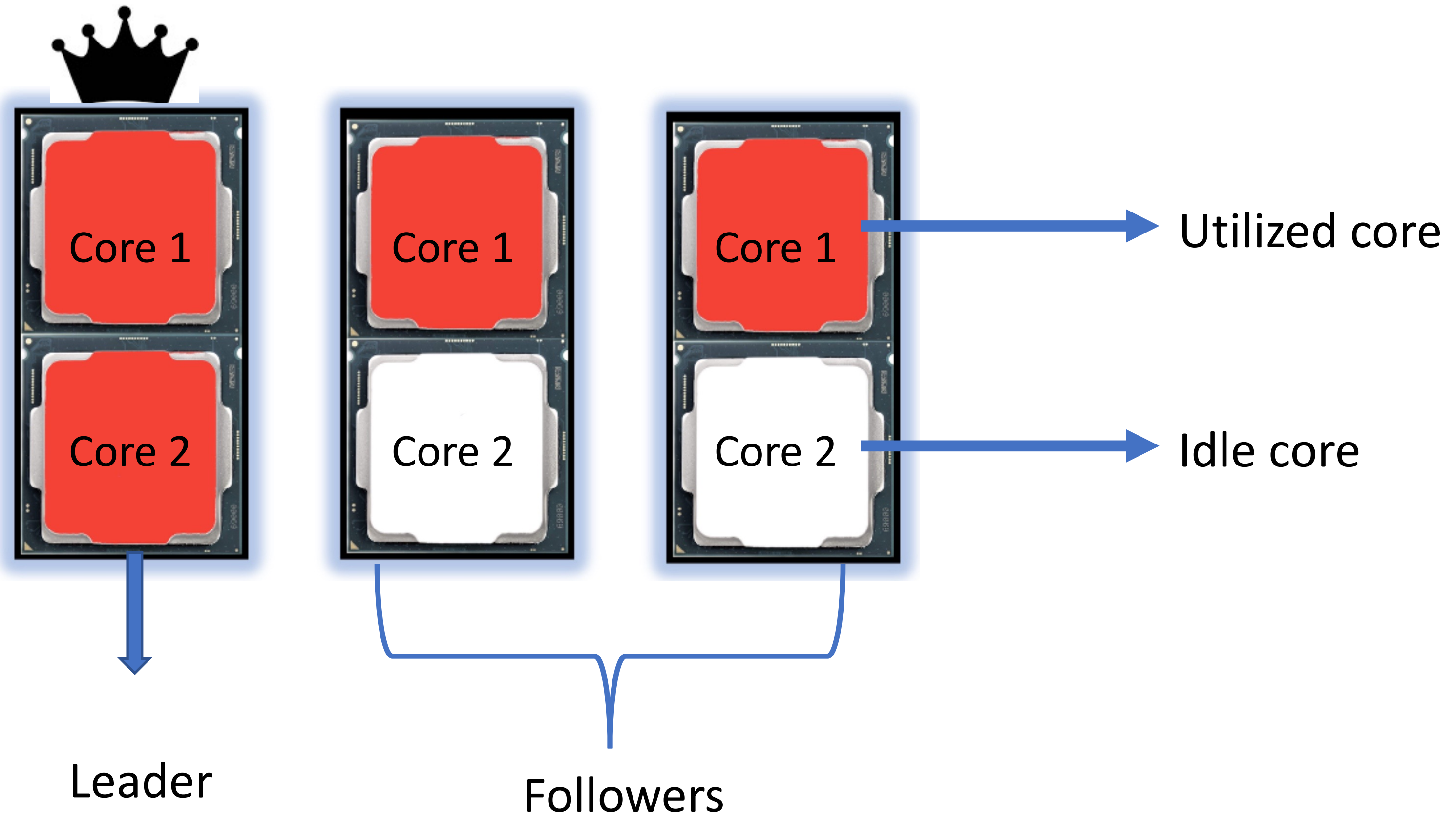
- ***Many protocols shift work from the bottleneck to the under-utilized node***
- Examples: EPaxos, SDPaxos and PigPaxos

Resource utilization of replication protocols



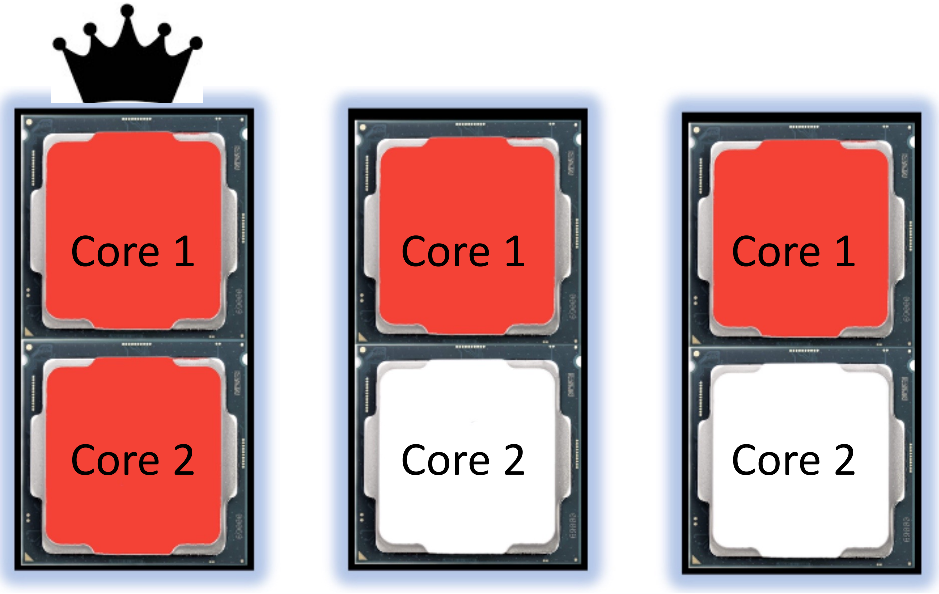
3 nodes with 2 cores each

Resource utilization of replication protocols

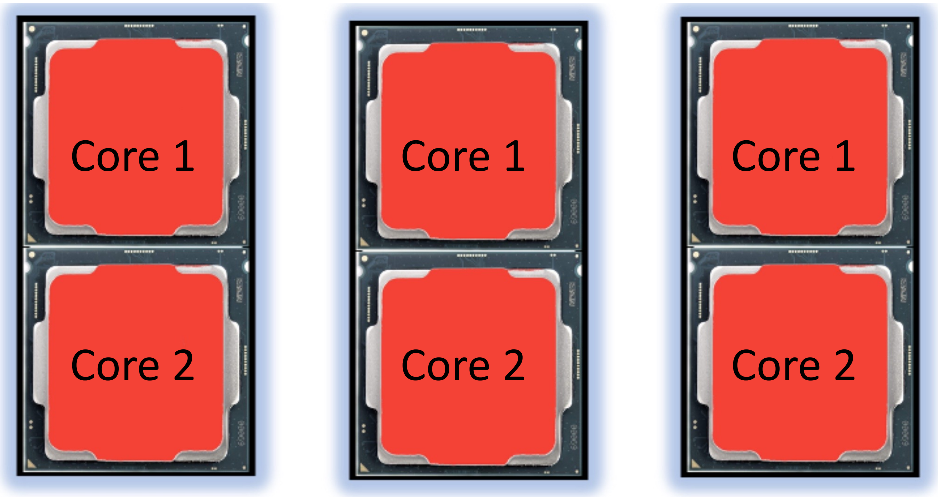


Multi-Paxos

Resource utilization of replication protocols



Multi-Paxos

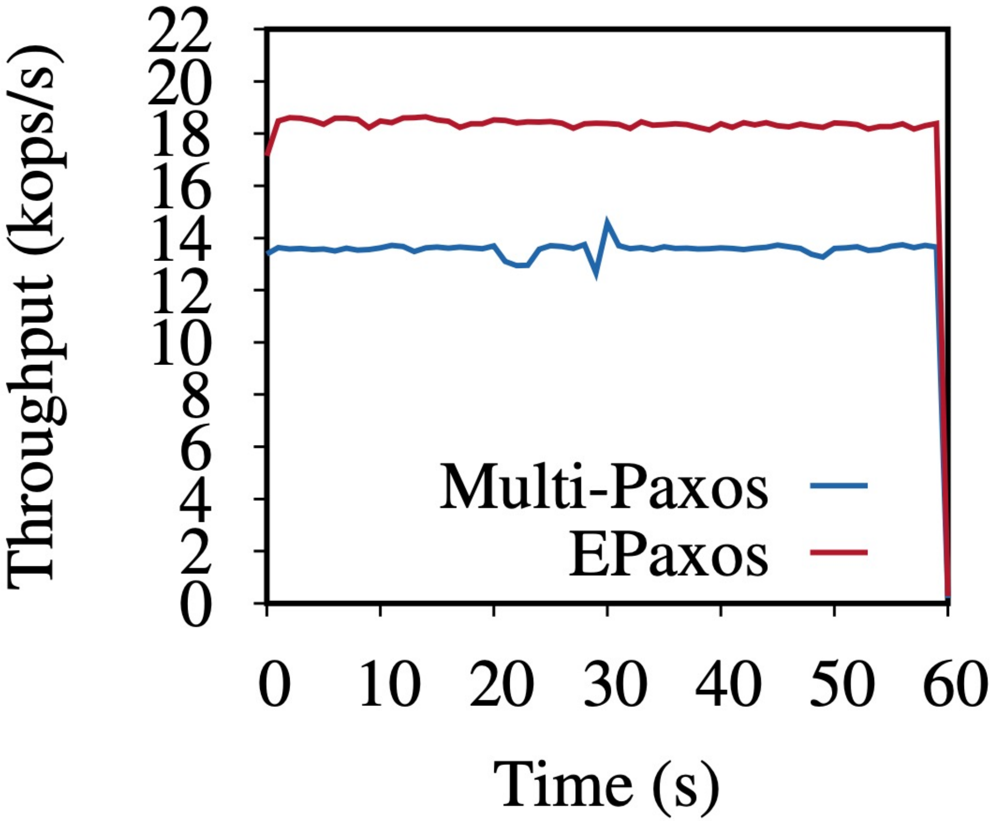


EPaxos

EPaxos also utilizes the idle cores to achieve high throughput

Confirming performance gains

- Single Instance
- 5 AWS EC2 m5a.large nodes
- Each 2 vCPU, 8GB RAM
- 50% write workload

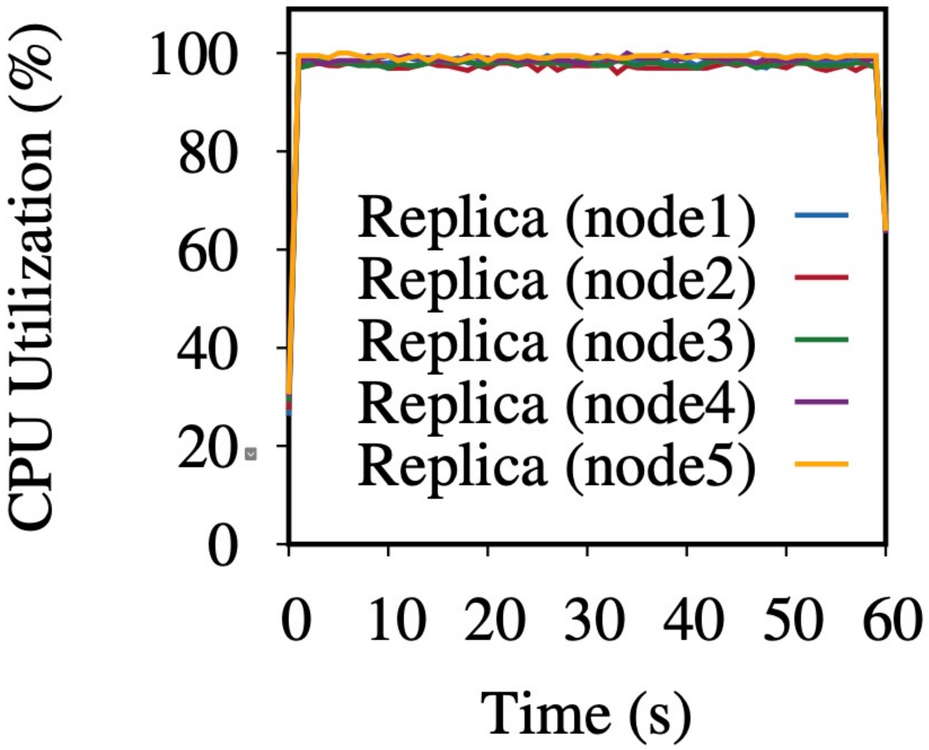


Throughput of Multi-Paxos and EPaxos

EPaxos achieves 20% higher throughput compared to Multi-Paxos

Missing piece: Resource efficiency

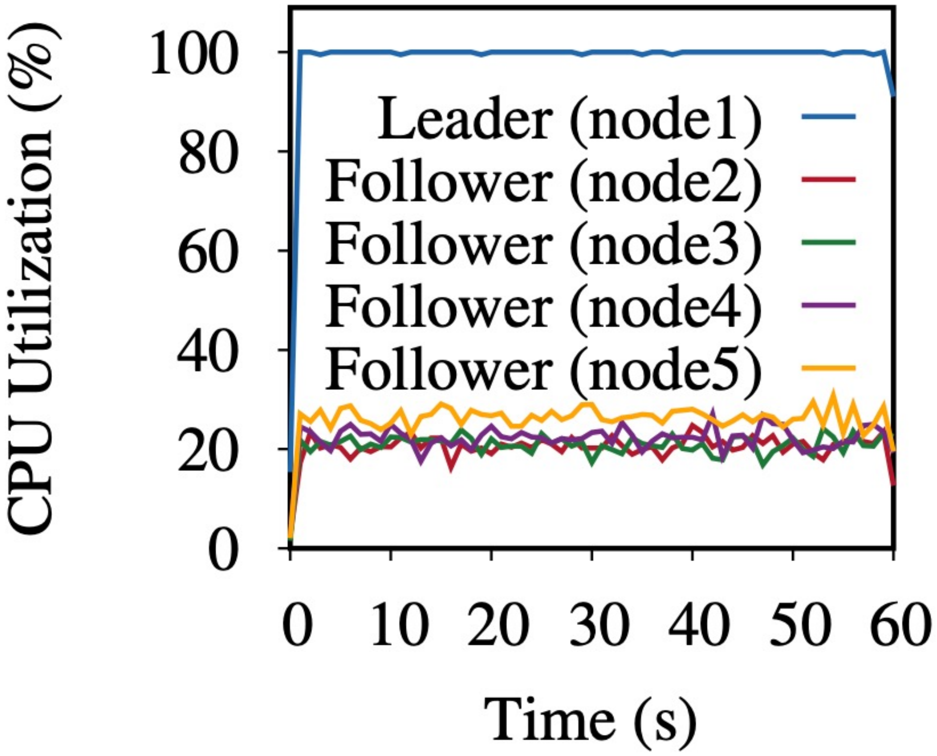
EPaxos



500% Utilization

18 kops/s

Multi-Paxos



200% Utilization

14 kops/s

Multi-Paxos shows better resource efficiency compared to EPaxos

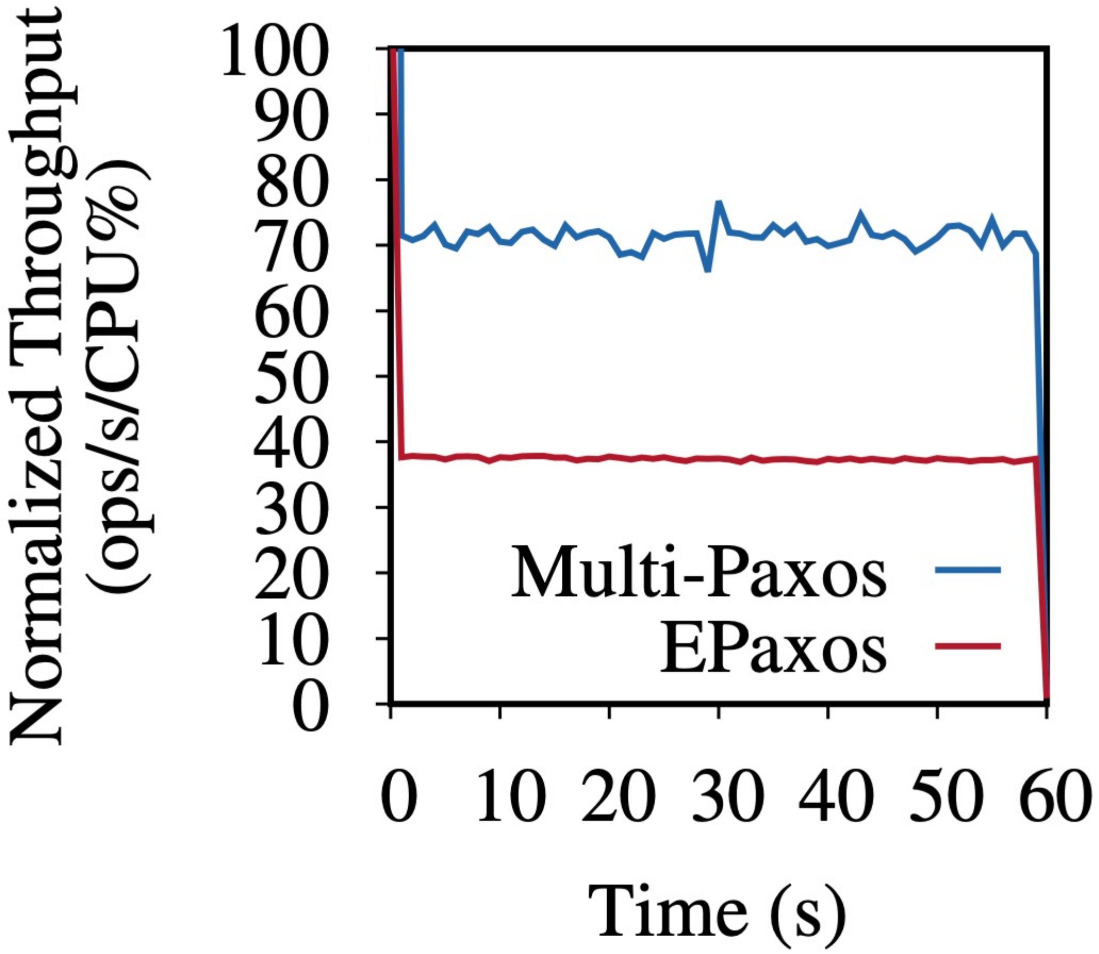
Metric to analyze Resource efficiency

Throughput-per-unit-of-constraining-resource-utilization

- Used CPU utilization to identify resource efficiency

- This metric determines the added cost of removing bottleneck

Throughput-per-unit-of-aggregate-CPU-Utilization



Metric shows the resource efficiency of replication protocols

Relevance of resource efficiency in Cloud

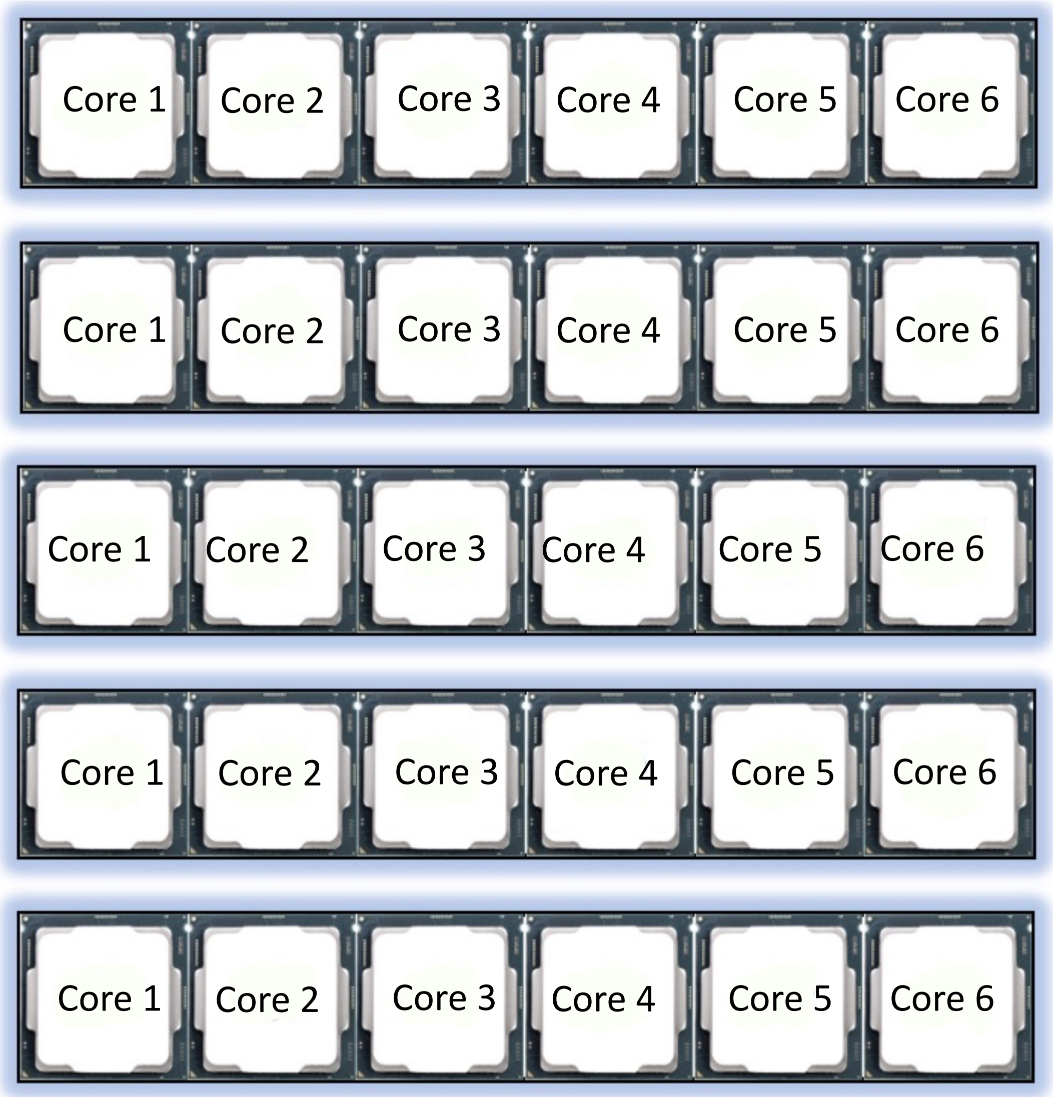
- Important in a pay-as-you-go utility model like Cloud

- Replication protocols are optimized for dedicated VMs

- Whereas Cloud is sharded and resource packed

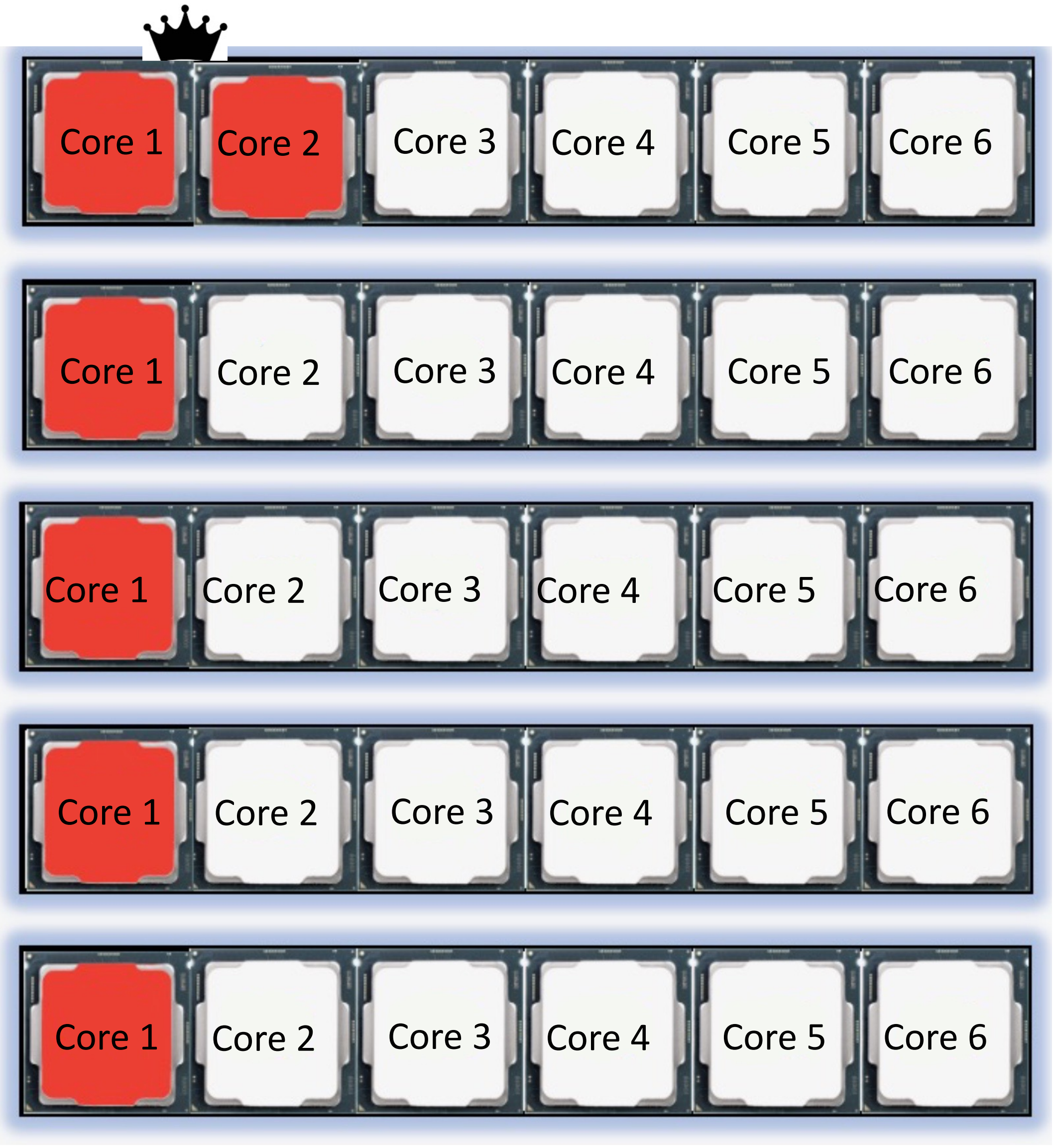
- Spanner, CockroachDB, and YugabyteDB support many instances from different shards on the same physical machine

Example: Packing in a resource constrained setting



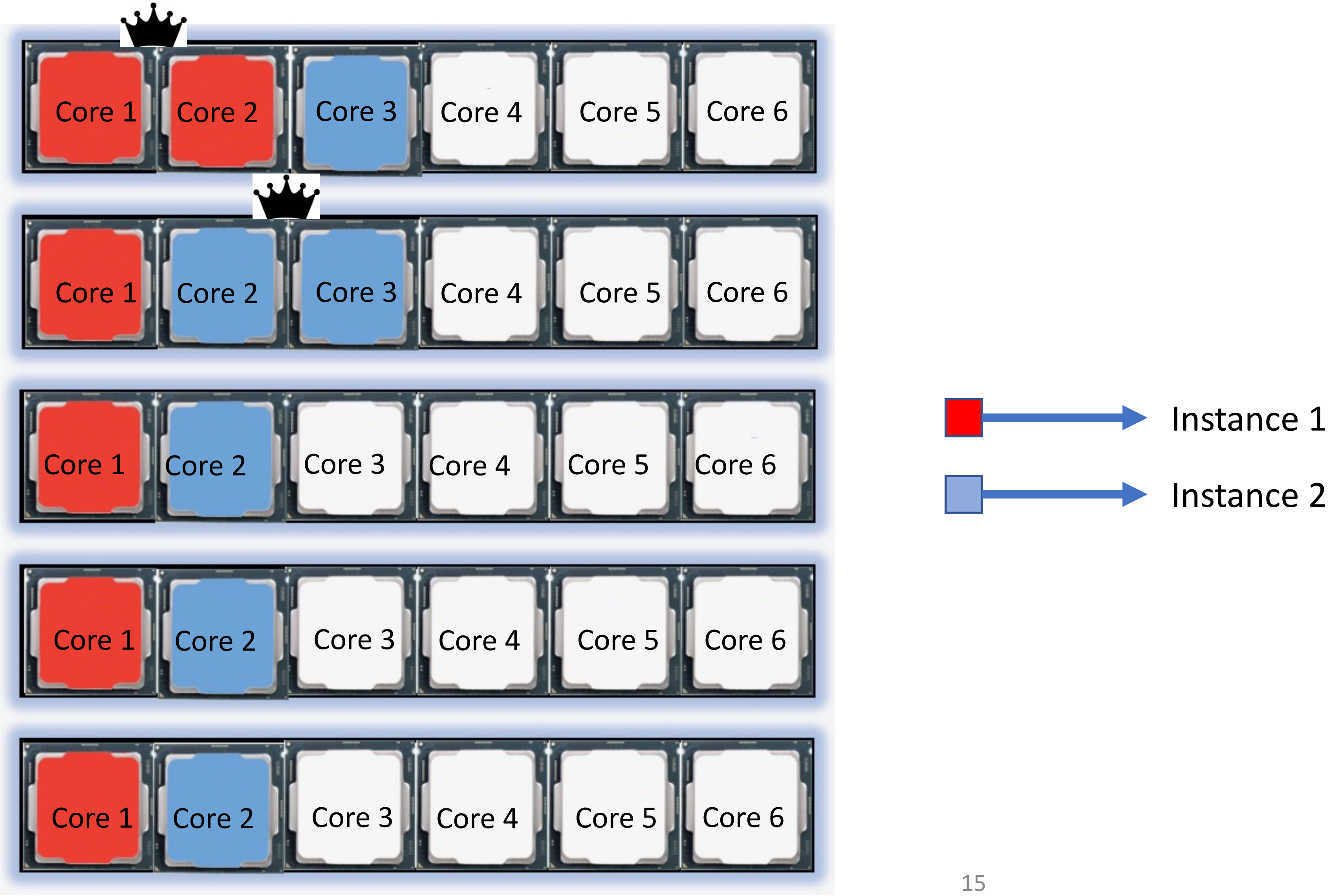
5 nodes with 6 cores each

Example: Packing in a resource constrained setting

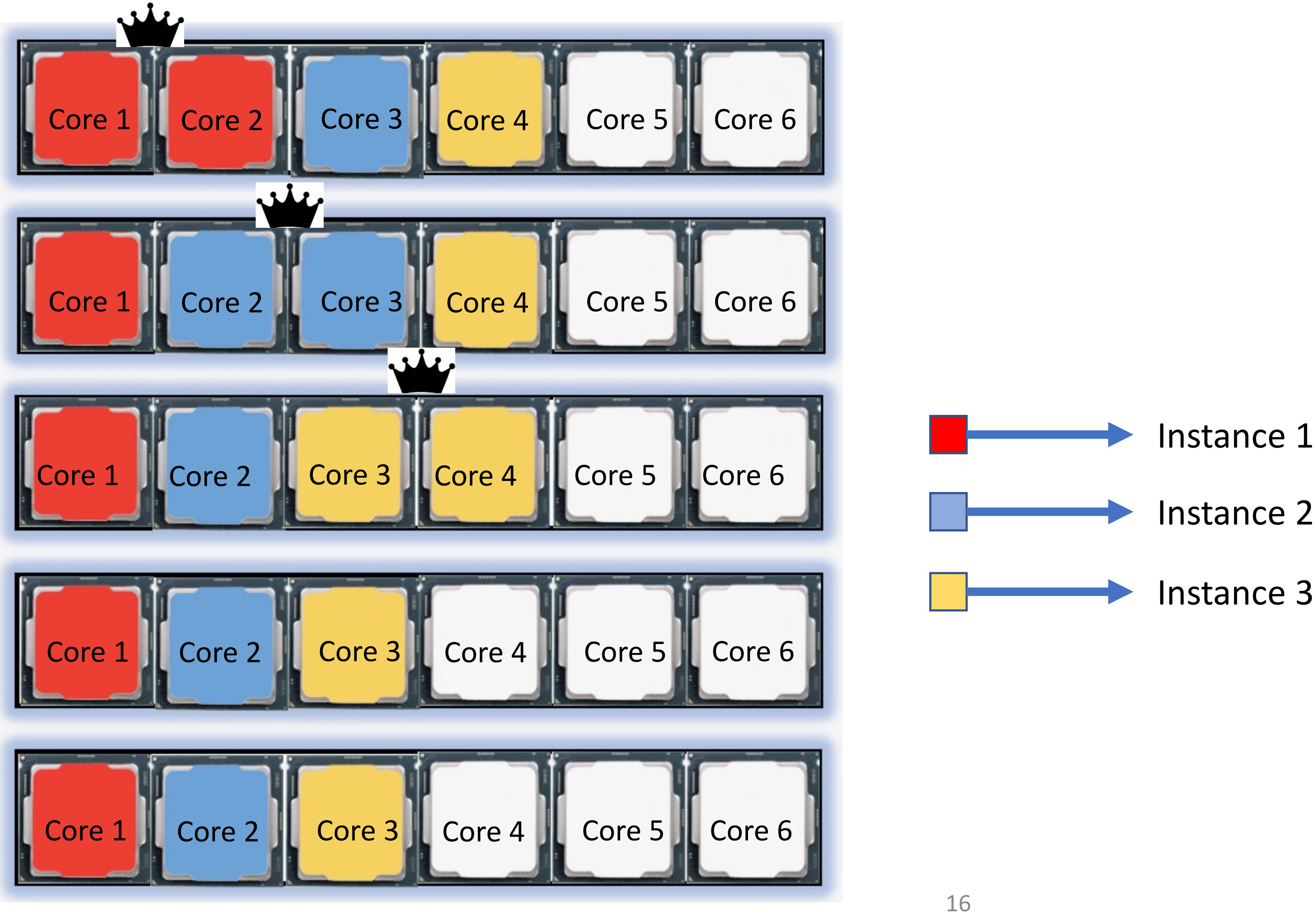


🟥 → Instance 1

Example: Packing in a resource constrained setting



Example: Packing in a resource constrained setting



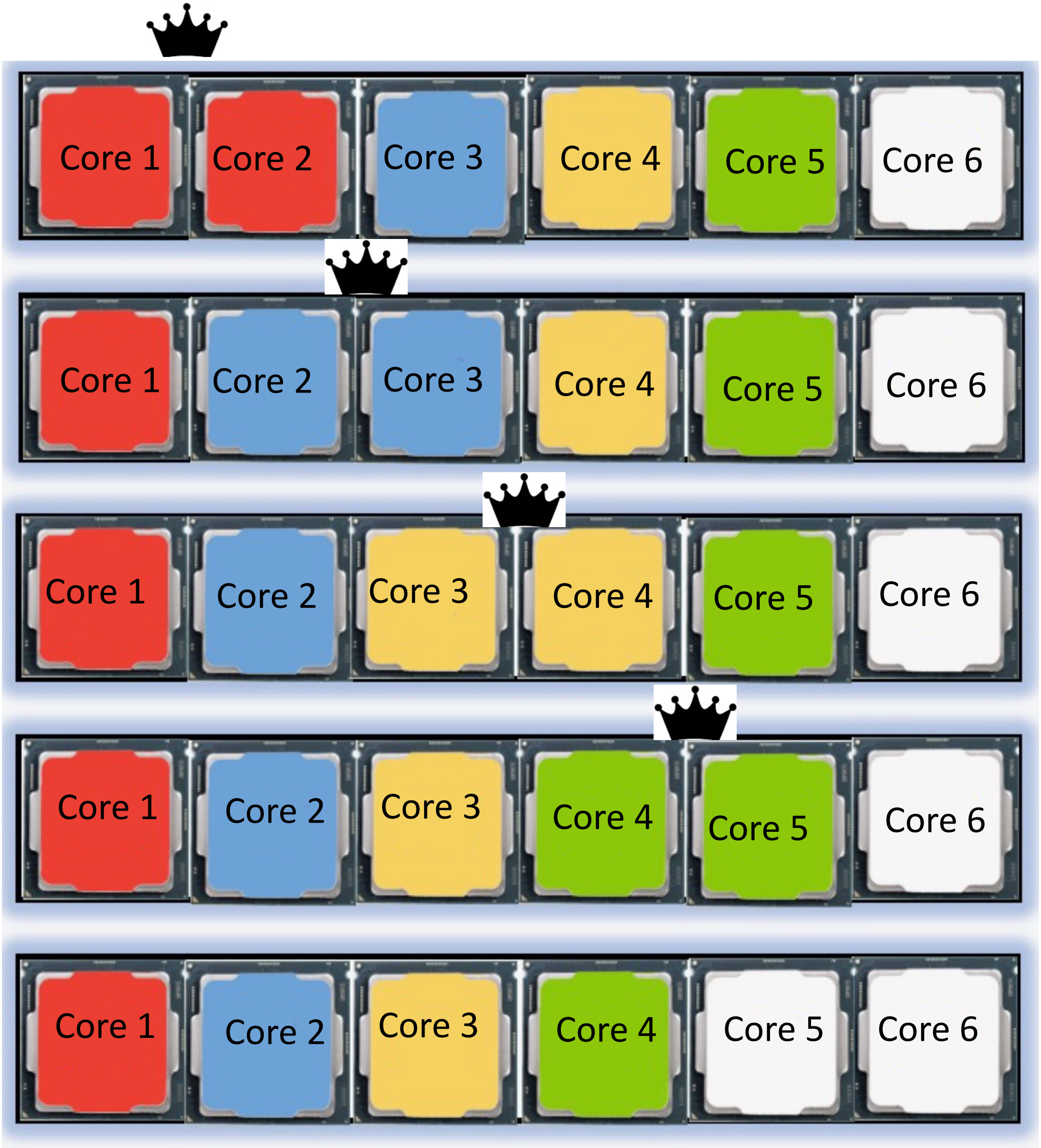
🟥 Core 1	🟥 Core 2	🟦 Core 3	🟨 Core 4	⬜ Core 5	⬜ Core 6
🟥 Core 1	🟦 Core 2	🟦 Core 3	🟨 Core 4	⬜ Core 5	⬜ Core 6
🟥 Core 1	🟦 Core 2	🟨 Core 3	🟨 Core 4	⬜ Core 5	⬜ Core 6
🟥 Core 1	🟦 Core 2	🟨 Core 3	⬜ Core 4	⬜ Core 5	⬜ Core 6
🟥 Core 1	🟦 Core 2	🟨 Core 3	⬜ Core 4	⬜ Core 5	⬜ Core 6

🟥 → Instance 1

🟦 → Instance 2

🟨 → Instance 3

Example: Packing in a resource constrained setting



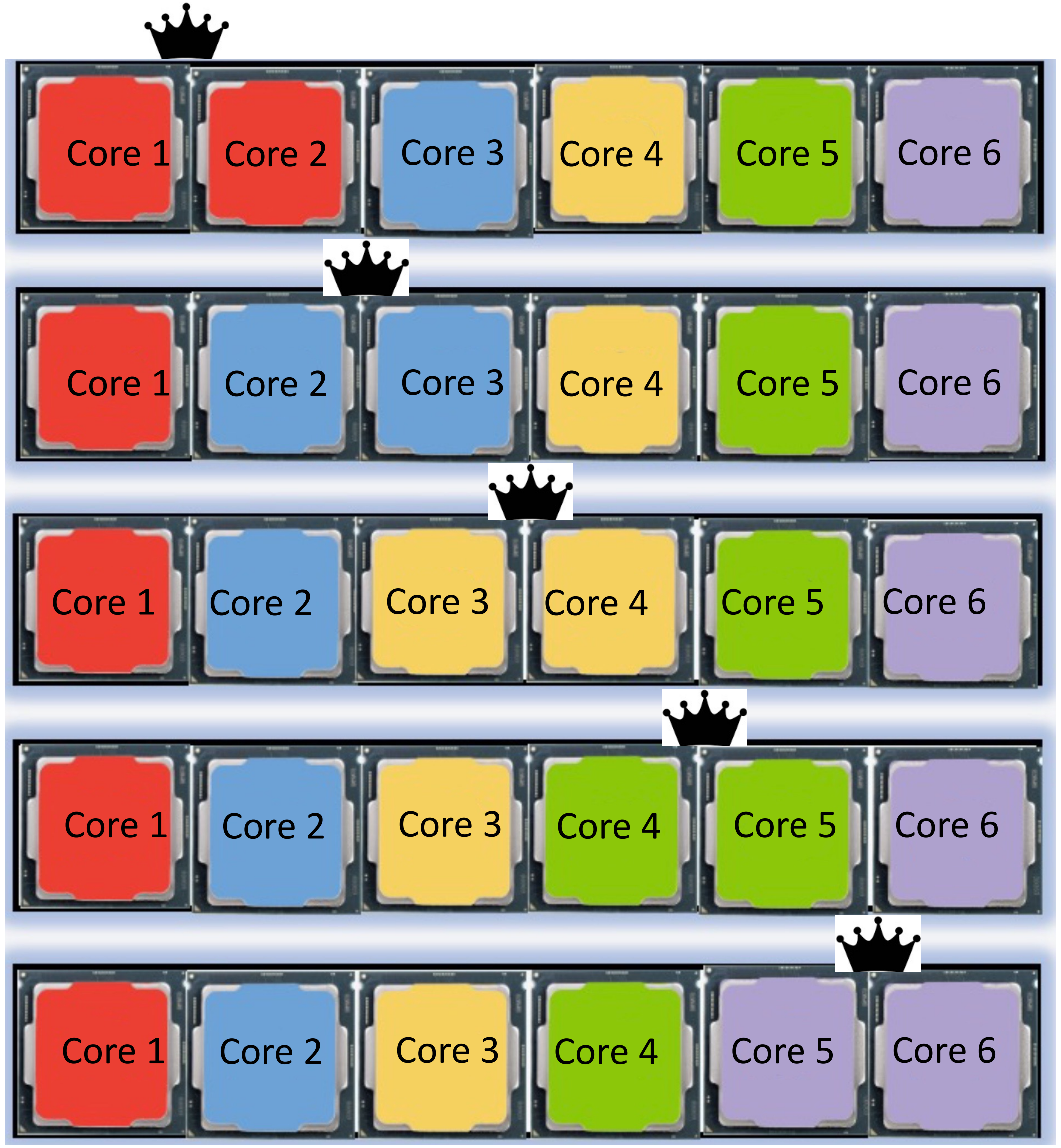
🟥 → Instance 1

🟦 → Instance 2

🟨 → Instance 3

🟩 → Instance 4

Example: Packing in a resource constrained setting



🟥 → Instance 1

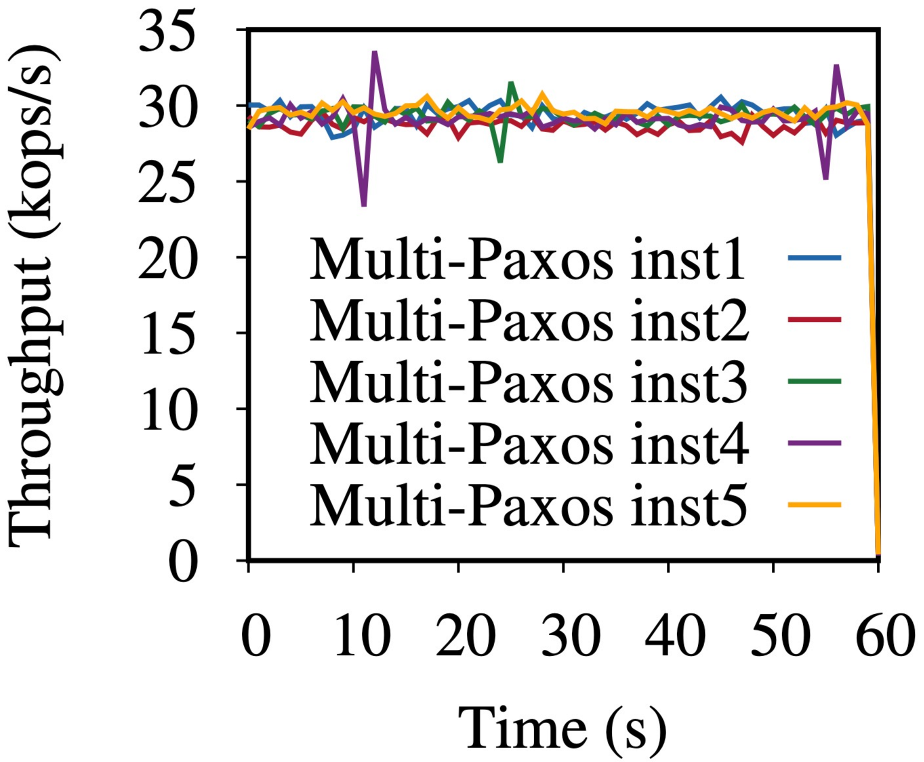
🟦 → Instance 2

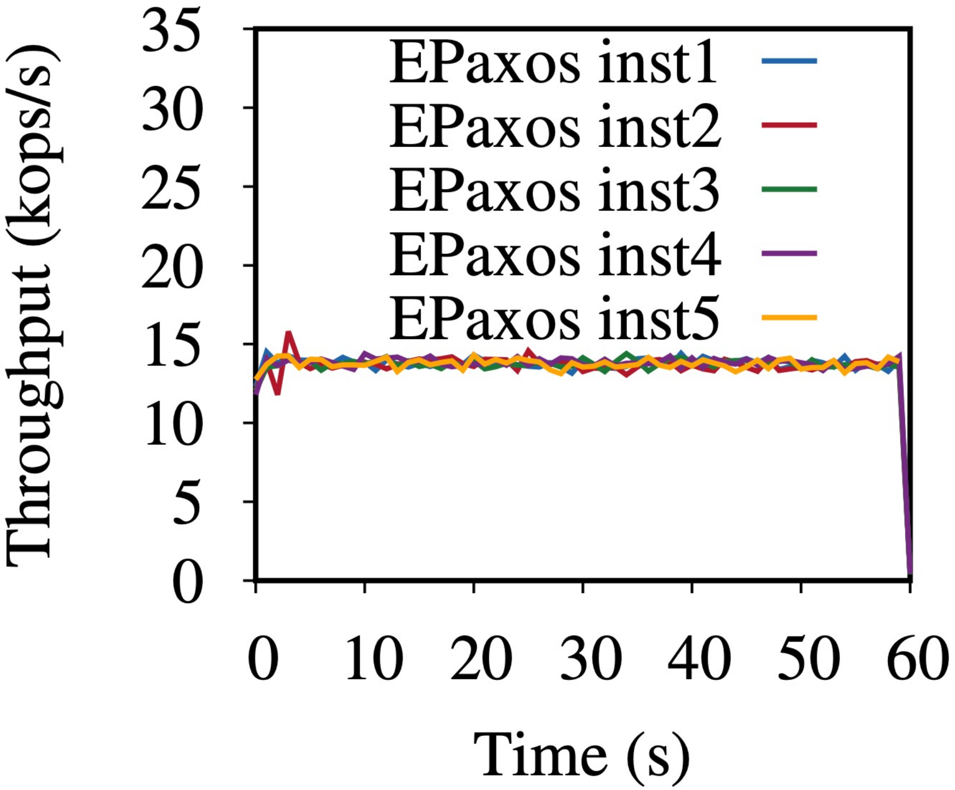
🟨 → Instance 3

🟩 → Instance 4

🟪 → Instance 5

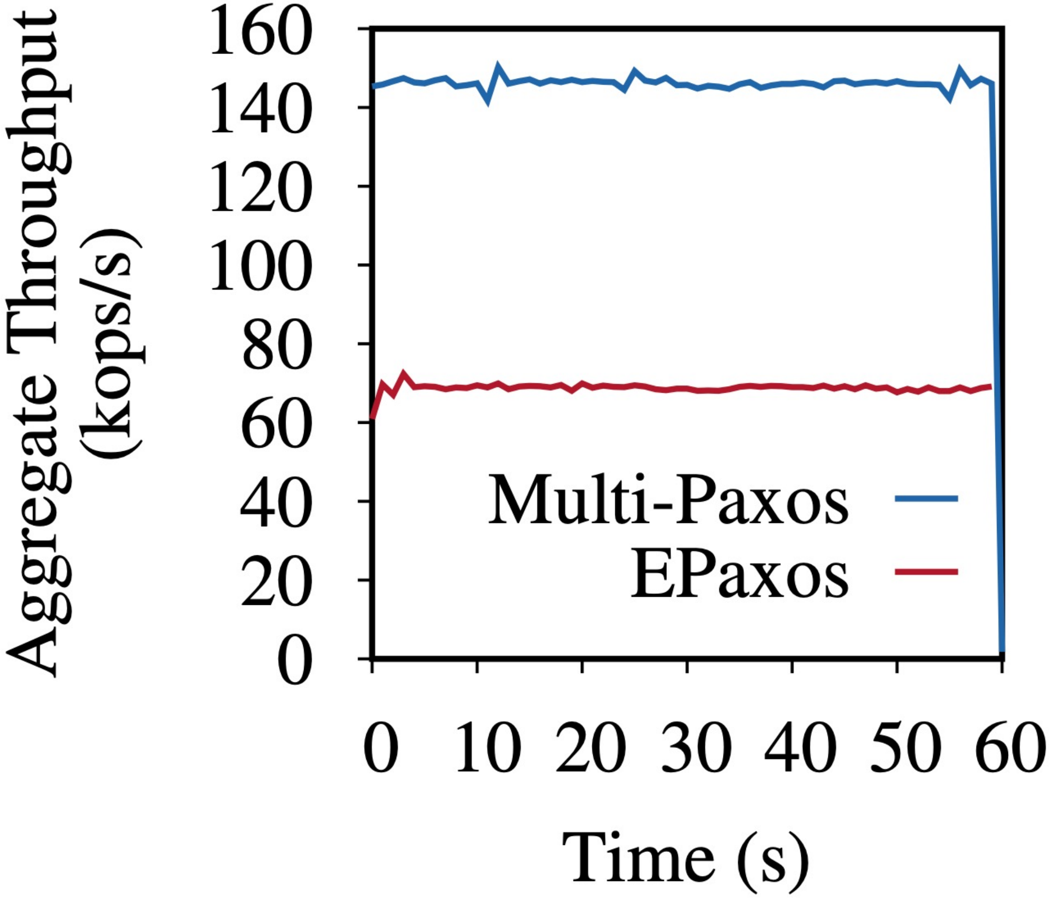
Experiment: Packing 5 instances in Cloud





- 5 Instance of Multi-Paxos/EPaxos
- 5 AWS EC2 m5a.2xlarge nodes
- Each 8 vCPU, 32GB RAM
- 50% write workload

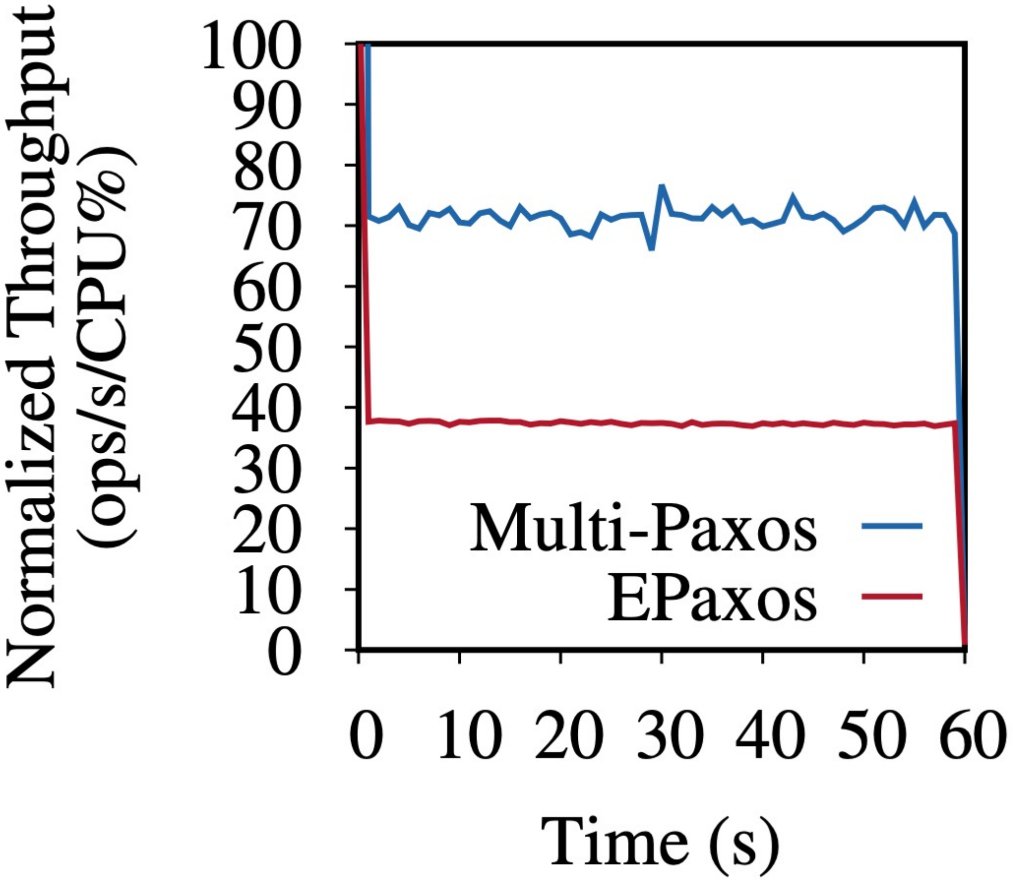
Aggregate Throughput



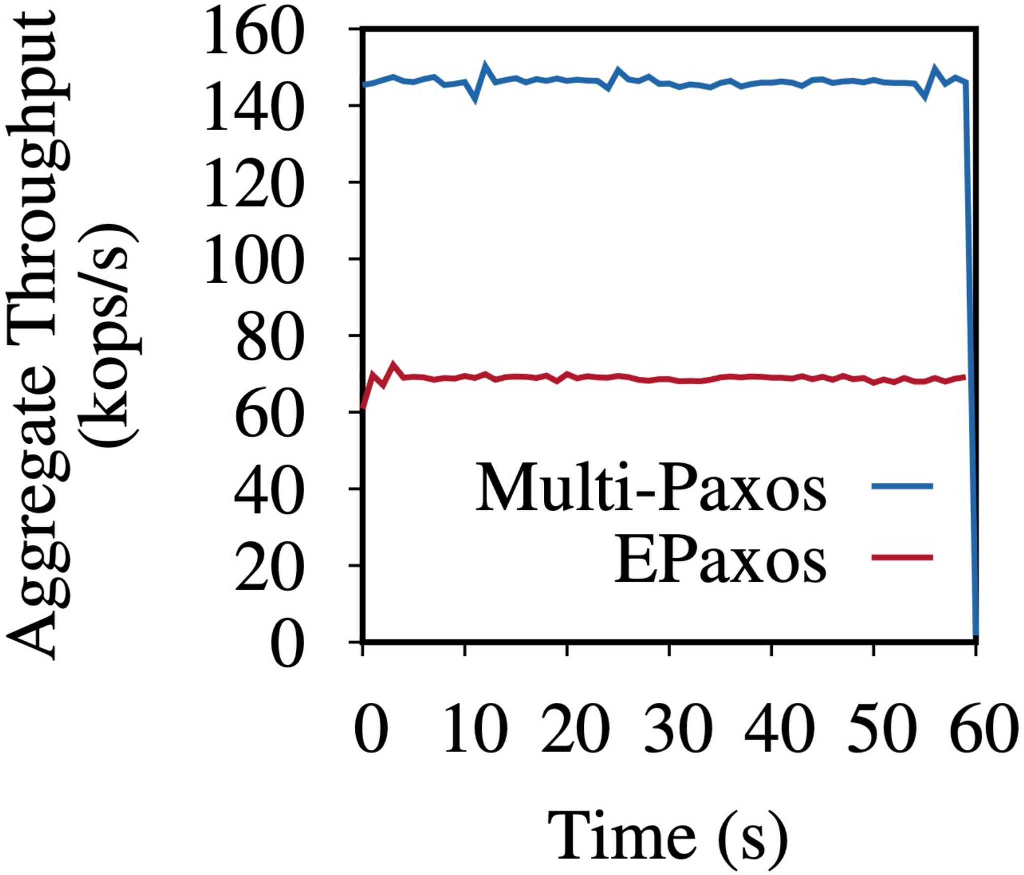
Aggregate throughput of Multi-Paxos and EPaxos with 5 instances packed together

Why throughput-per-unit-of-constraining-resource-utilization?

It is a good proxy for the performance of replication protocols in Cloud setting

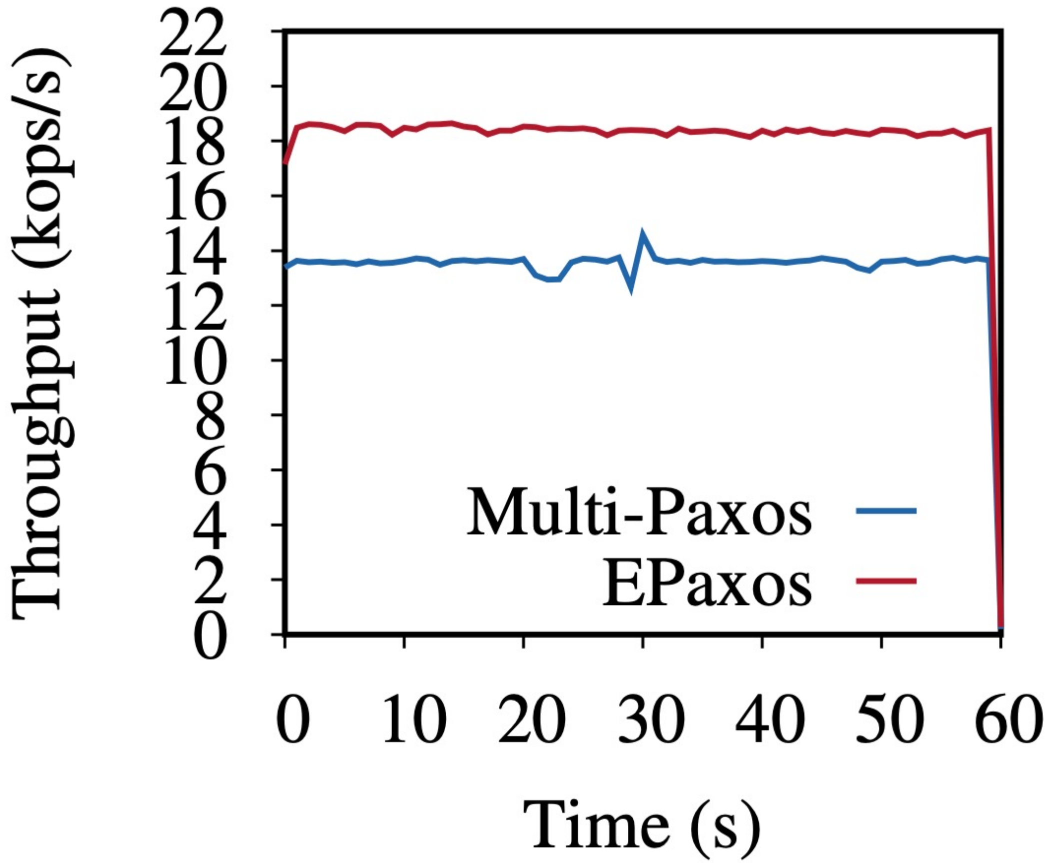


dedicated resource setting

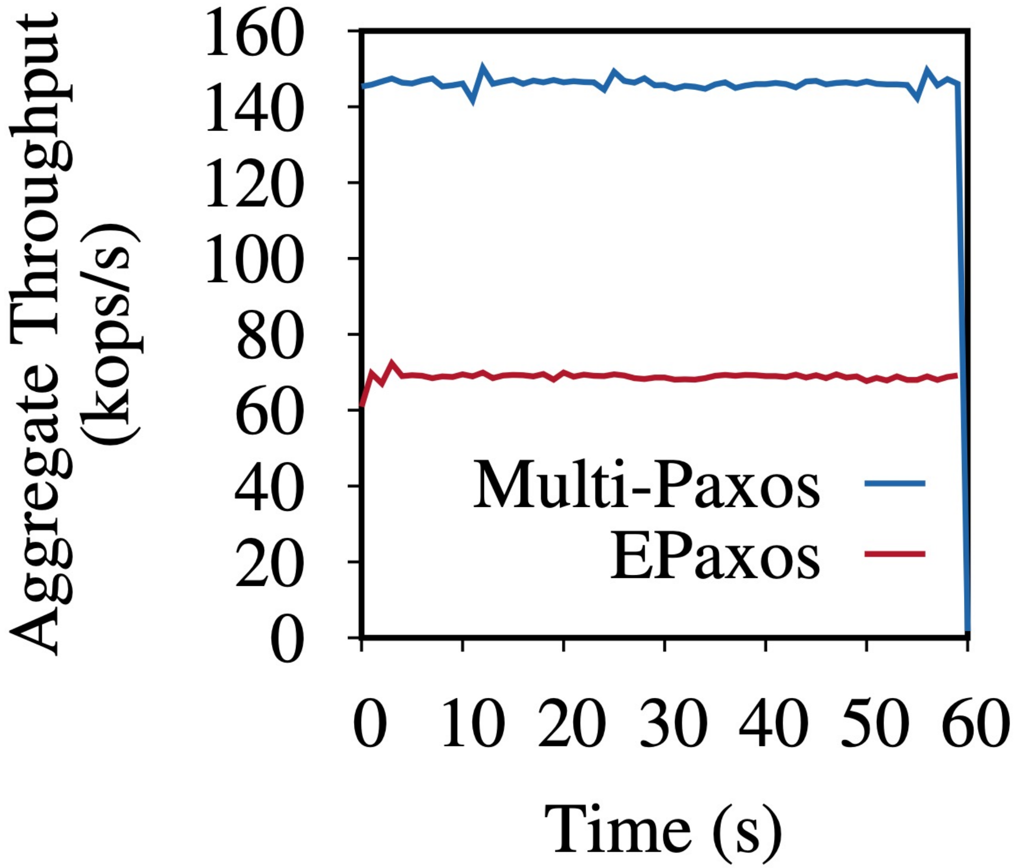


shared resource setting

Conclusion: Scalable but Wasteful



dedicated resource setting



Fixed-budget shared resource setting

Resource efficiency plays a key role for replication protocols when moving from a dedicated to shared resource setting