Achieving Enhanced Performance Combining Checkpointing and Dynamic State Partitioning

Henrique Goulart

João Trombeta
Álvaro Franco
Odorico Mendizabal

Universidade Federal de Santa Catarina - UFSC

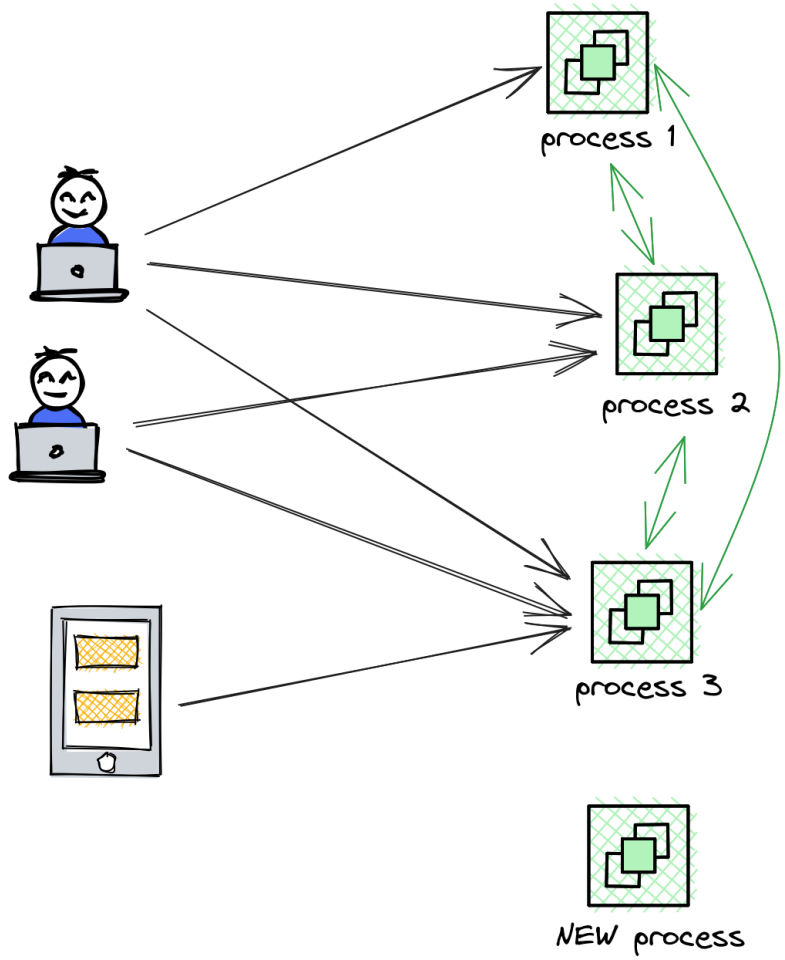
SBAC-PAD 2023

Summary

- Introduction
- Problem
- Evaluation
- Conclusion

Introduction
Distributed systems

- Distributed systems
  - Multiple process/systems running
  - Handling multiple requests simultaneously
  - Handling high volume of data
  - E.g.: replicated state machine running with 3 replicas
- **Challenges**
  - A failure/fault happens in one process/worker
  - Scalability requirement – peak usage

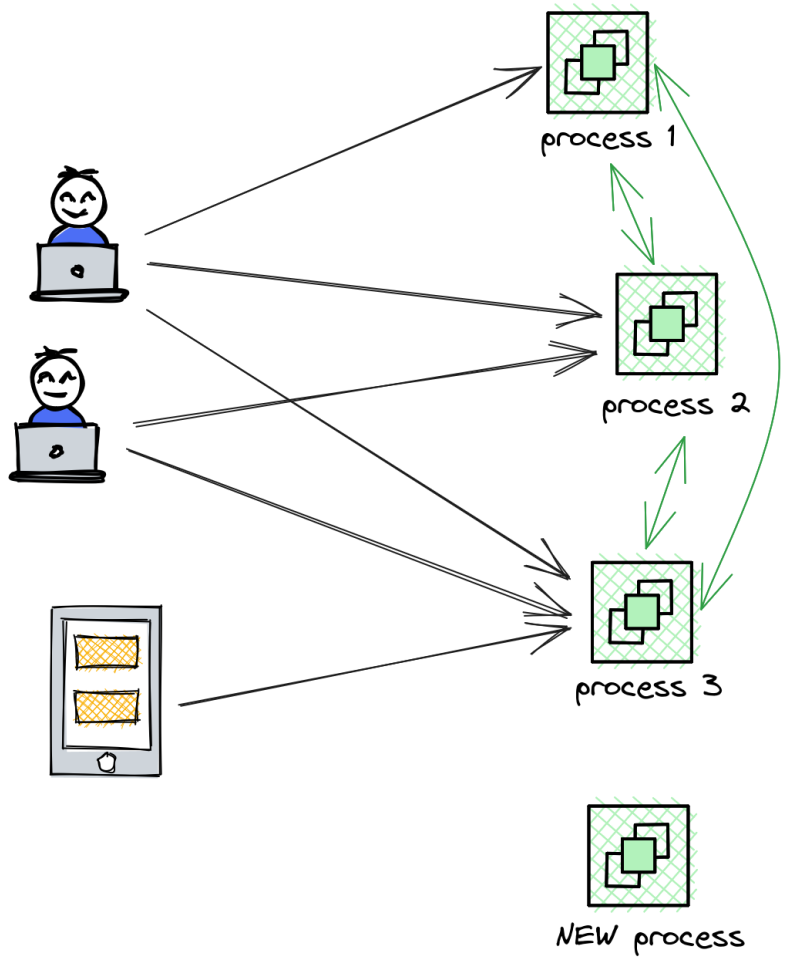


Introduction

Distributed systems

- Distributed systems
  - Multiple process/systems running
  - Handling multiple requests simultaneously
  - Handling high volume of data
  - E.g.: replicated state machine running with 3 replicas
- **Challenges**
  - A failure/fault happens in one process/worker
  - Scalability requirement – peak usage

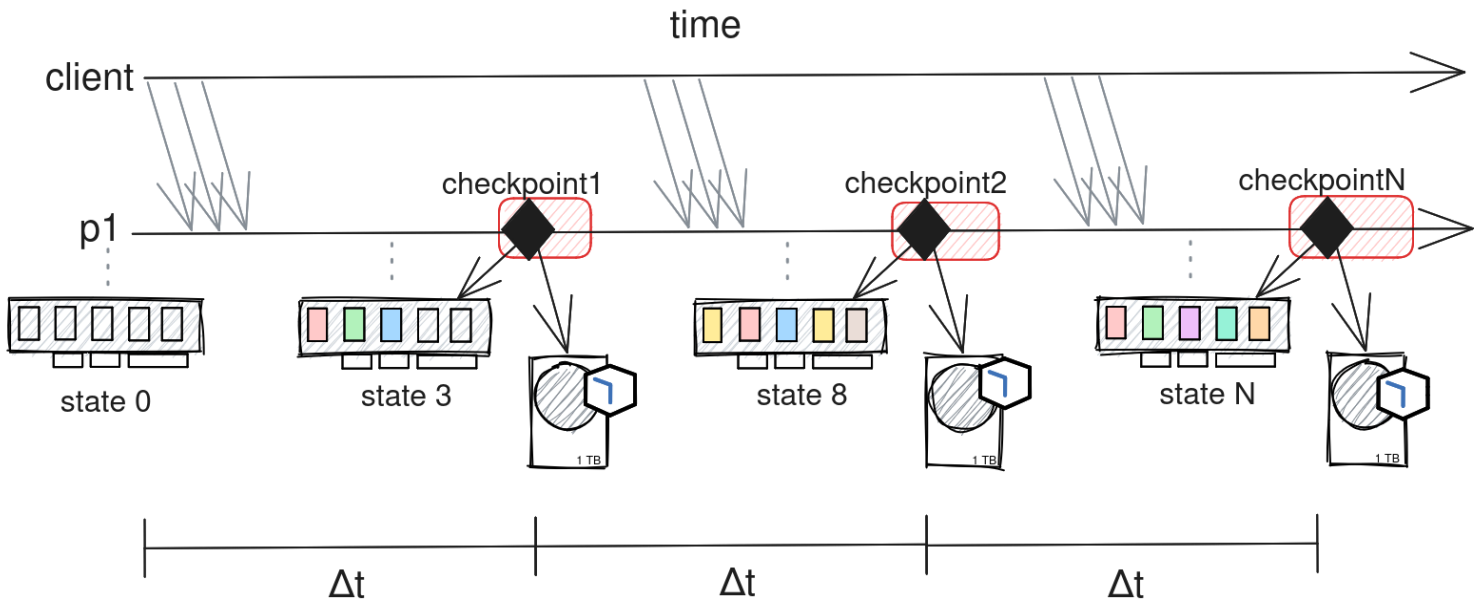
Add **checkpoint/recovery** durability strategy



Introduction

Durability strategies - Checkpointing

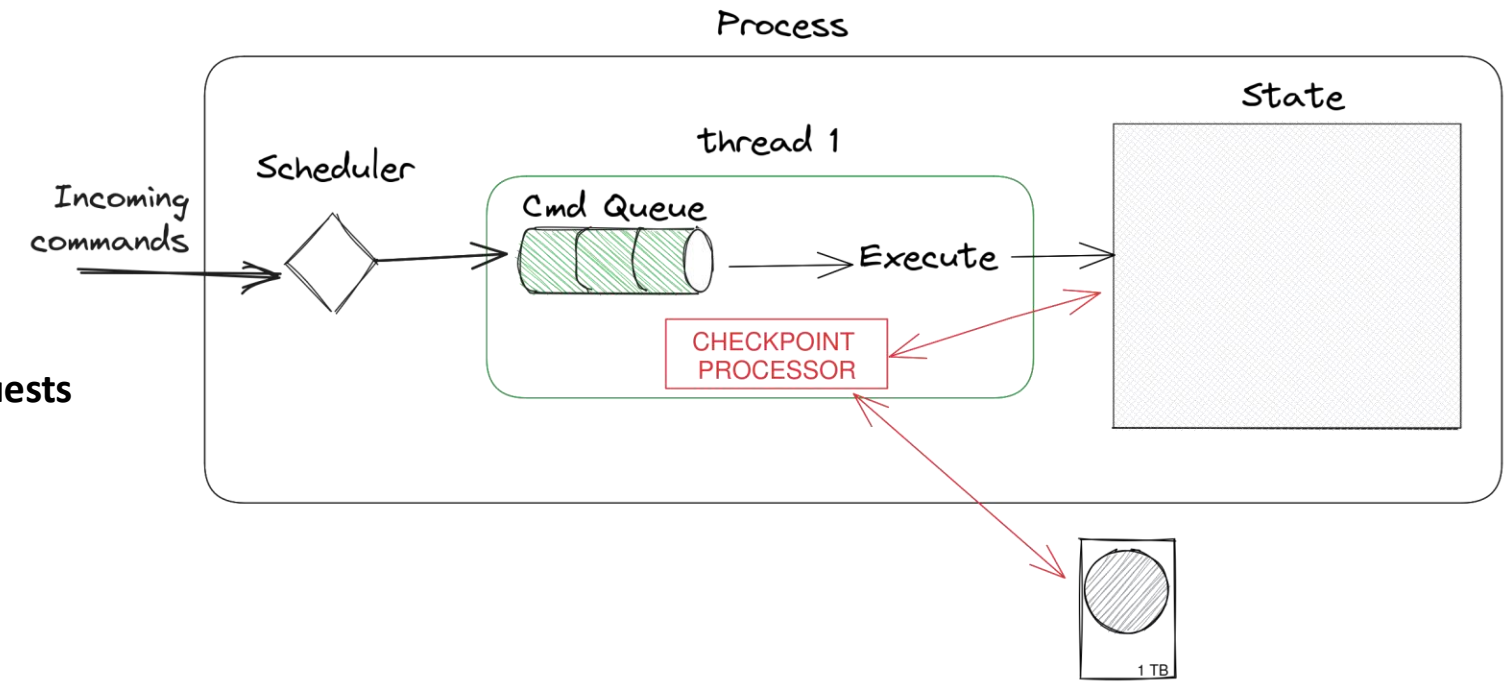
- Checkpoint/recovery
  - Periodically capture snapshots
- **Challenges**
  - **I/O operation that is expensive**
  - **Checkpoint synchronization with requests**



Introduction

Durability strategies - Checkpointing

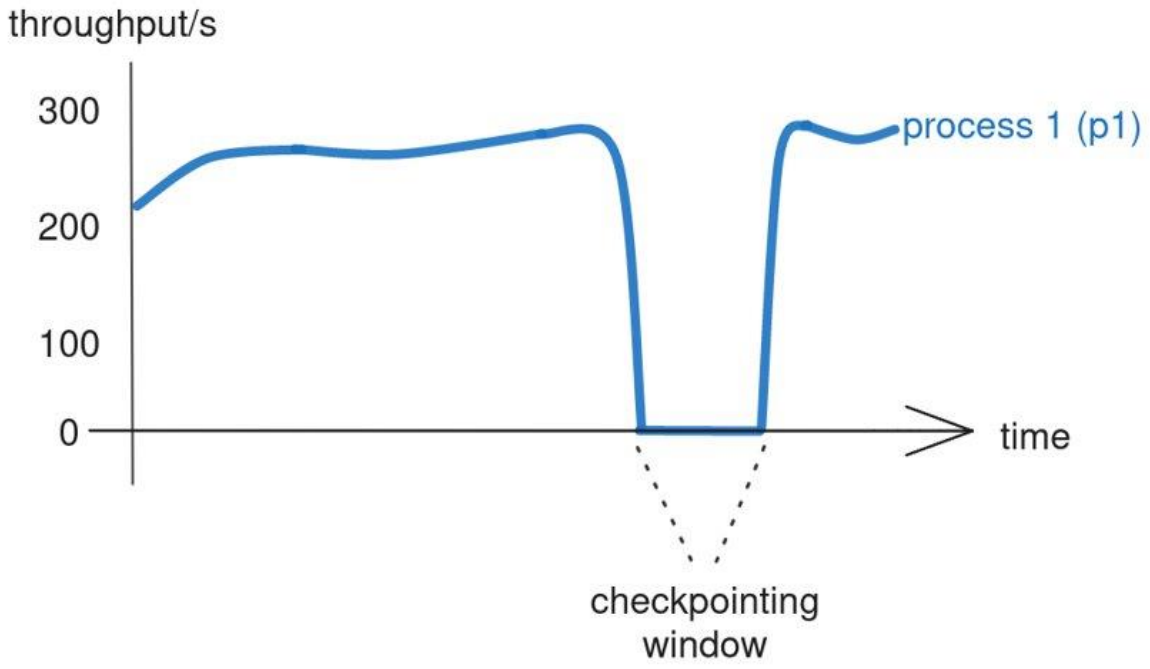
- Checkpoint/recovery
  - Periodically capture snapshots
- **Challenges**
  - **I/O operation that is expensive**
  - **Checkpoint synchronization with incoming requests**



Introduction

Durability strategies - Checkpointing

- Checkpoint/recovery
  - Periodically capture snapshots
- **Challenges**
  - **I/O operation that is expensive**
  - **Checkpoint synchronization with incoming requests**

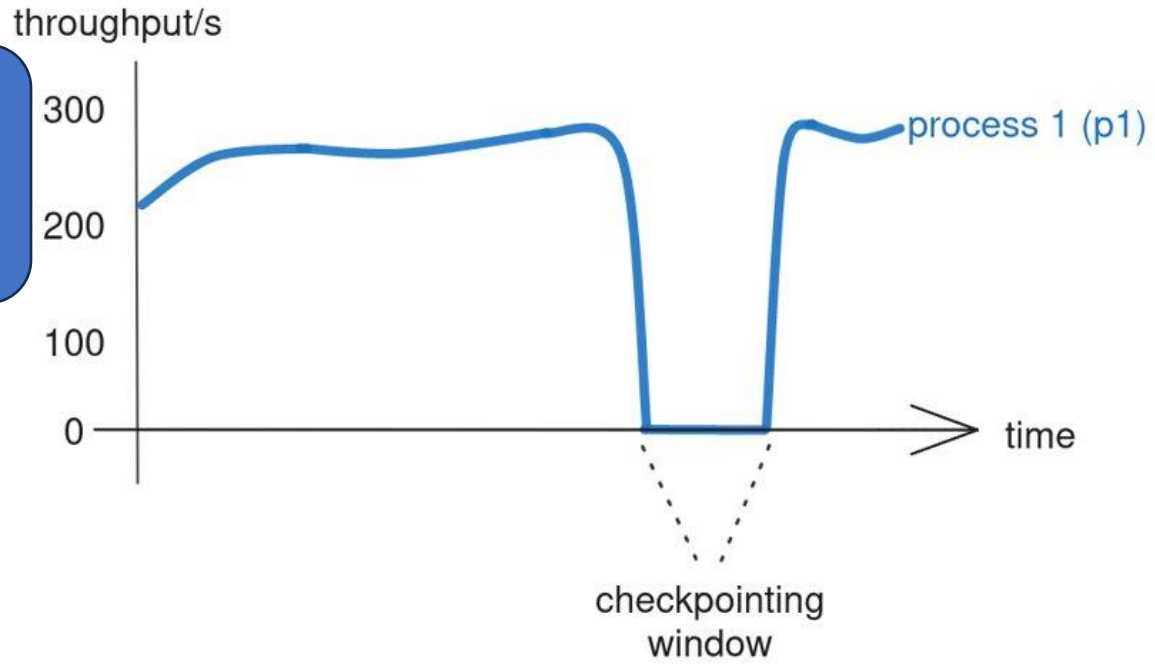


Introduction

Durability strategies - Checkpointing

- Checkpoint/recovery
  - Periodically capture snapshots
- **Challenges**
  - **I/O operation that is expensive**
  - **Checkpoint synchronization with incoming requests**

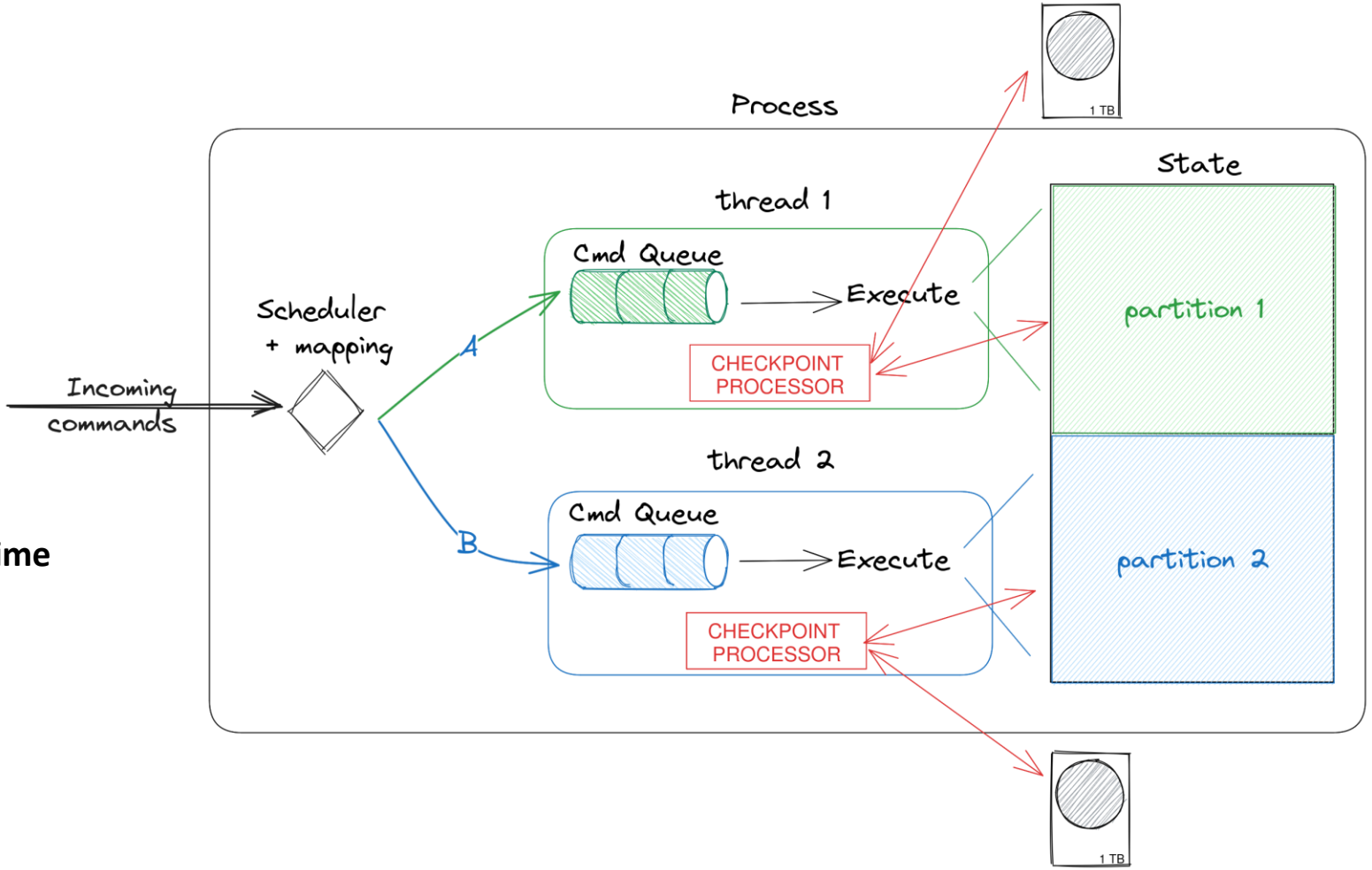
Add **State Partition**



Introduction

Enhancing checkpointing with state partitioning

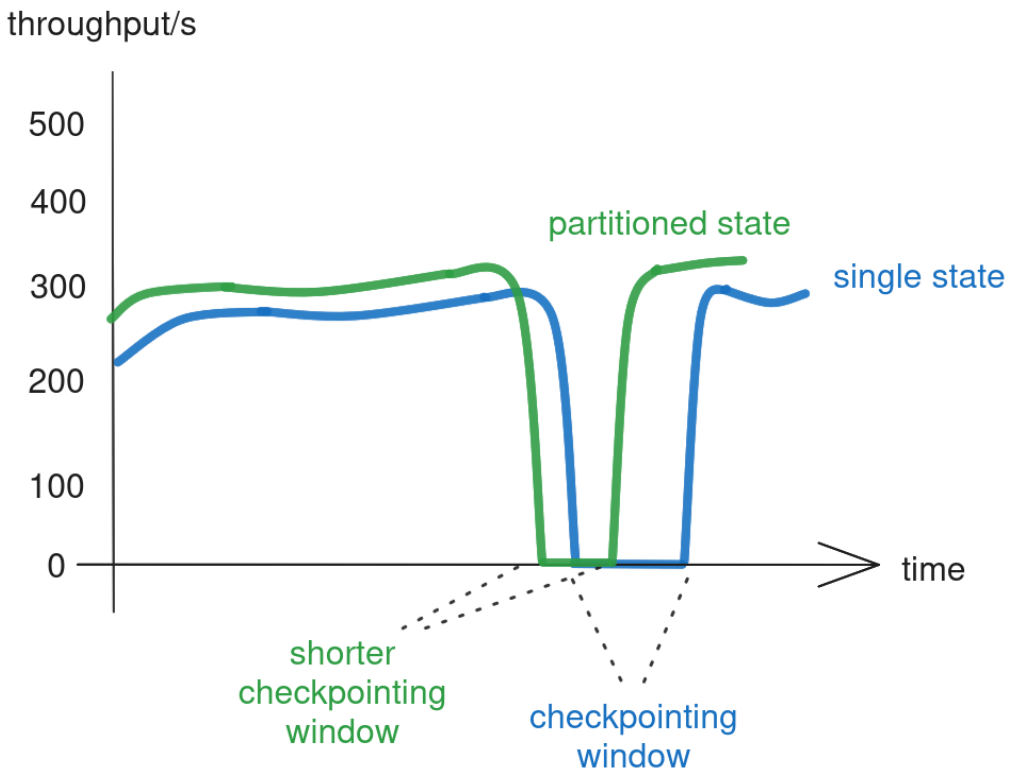
- State partitioning
  - Each thread takes care of a portion of the state
  - Increase the overall throughput
  - Overhead of managing multiple threads
  - **Checkpointing in parallel**
  - **Checkpoint executes faster**
  - **May have shorter checkpoint synchronization time**
  - **Need to synchronize multiple-partition access**



Introduction

Enhancing checkpointing with state partitioning

- State partitioning
  - Each thread takes care of a portion of the state
  - Increase the overall throughput
  - Overhead of managing multiple threads
  - **Checkpointing in parallel**
  - **Checkpoint executes faster**
  - **May have shorter checkpoint synchronization time**
  - ** **Need to synchronize multiple-partition access**

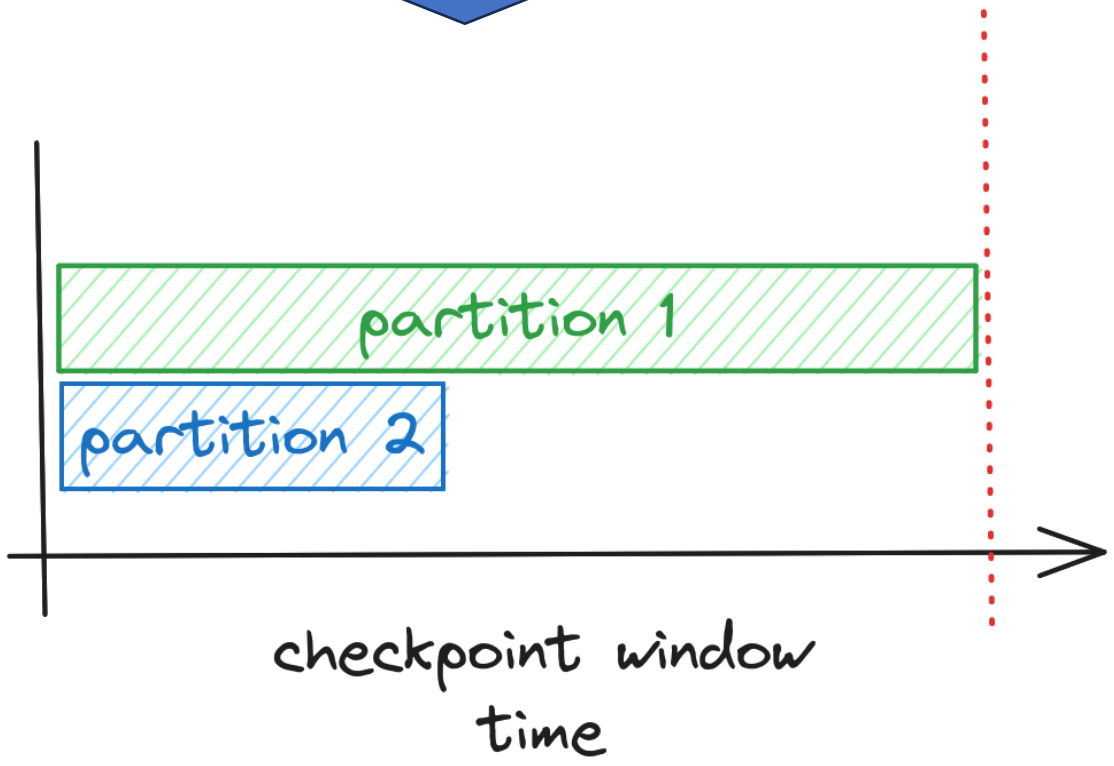


Introduction

Enhancing checkpointing with state partitioning

Checkpoint will take as long as the largest partition

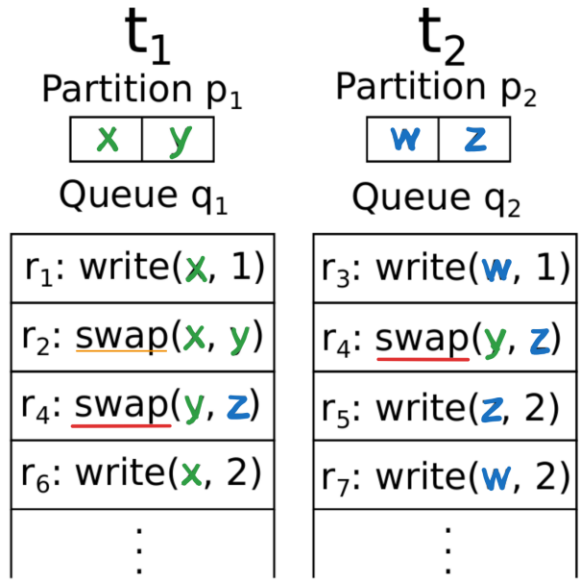
- State partitioning
  - Each thread takes care of a portion of the state
  - Increase the overall throughput
  - Overhead of managing multiple threads
  - **Checkpointing in parallel**
  - **Checkpoint executes faster**
  - **May have shorter checkpoint synchronization time**
  - ** **Need to synchronize multiple-partition access**

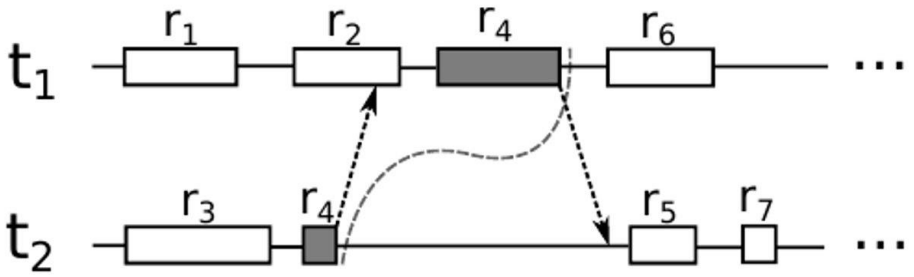


Introduction

Enhancing checkpointing with state partitioning

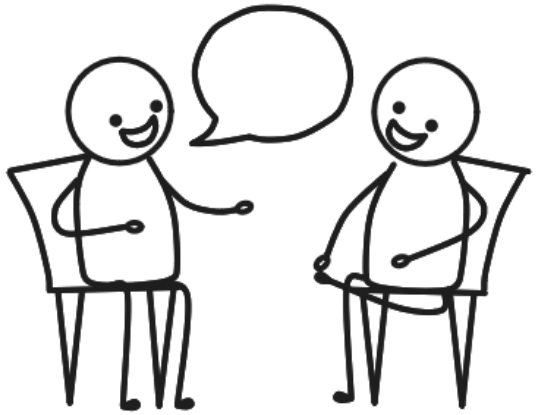
- State partitioning
  - Each thread takes care of a portion of the state
  - Increase the overall throughput
  - Overhead of managing multiple threads
  - **Checkpointing in parallel**
  - **Checkpoint might execute faster**
  - **May have shorter checkpoint synchronization time**
  - ** **Need to synchronize multiple-partition access**





Problem statement

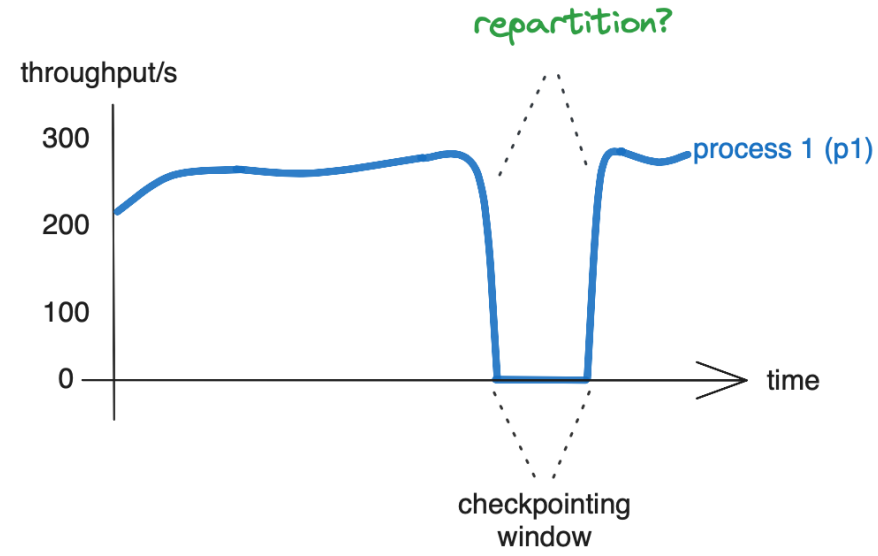
- The challenges/problems found:
  - Synchronizing multi-partition operations is **expensive**
  - **Unbalanced states** may impact **negatively** the total **checkpoint time**

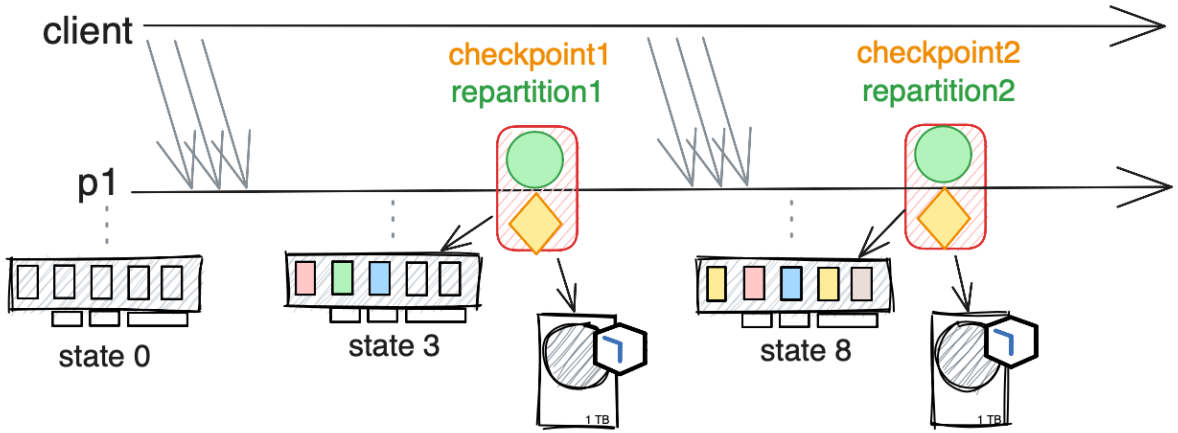


- What if:
  - We run a **repartition algorithm while checkpoint is running?**
  - Balanced state partitions?
  - Fewer synchronizations between partitions?
  - Would be a good idea?

Problem statement

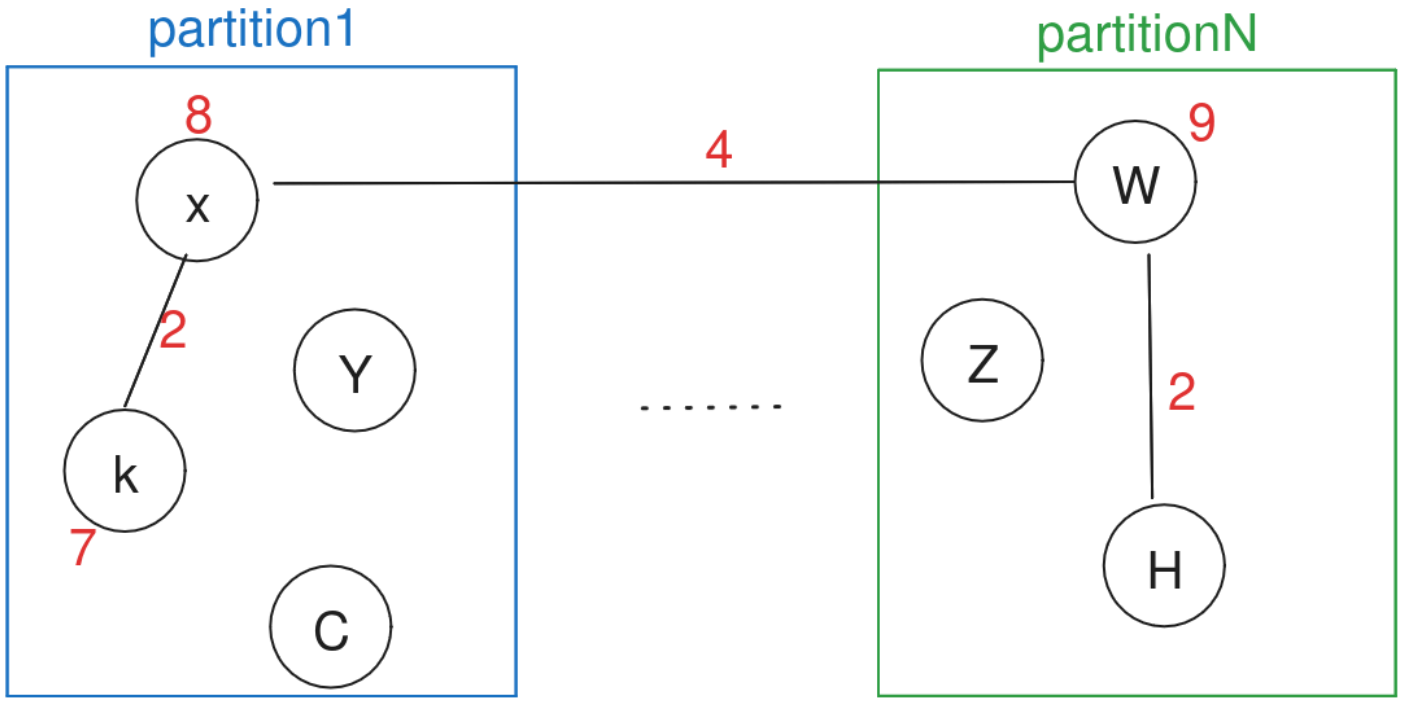
- The challenges/problems found:
  - Synchronizing multi-partition operations is **expensive**
  - **Unbalanced states** may impact **negatively** the total **checkpoint time**
- What if:
  - We run a **repartition algorithm while checkpoint is running?**
  - Balanced state partitions?
  - Fewer synchronizations?
  - Would be a good idea?





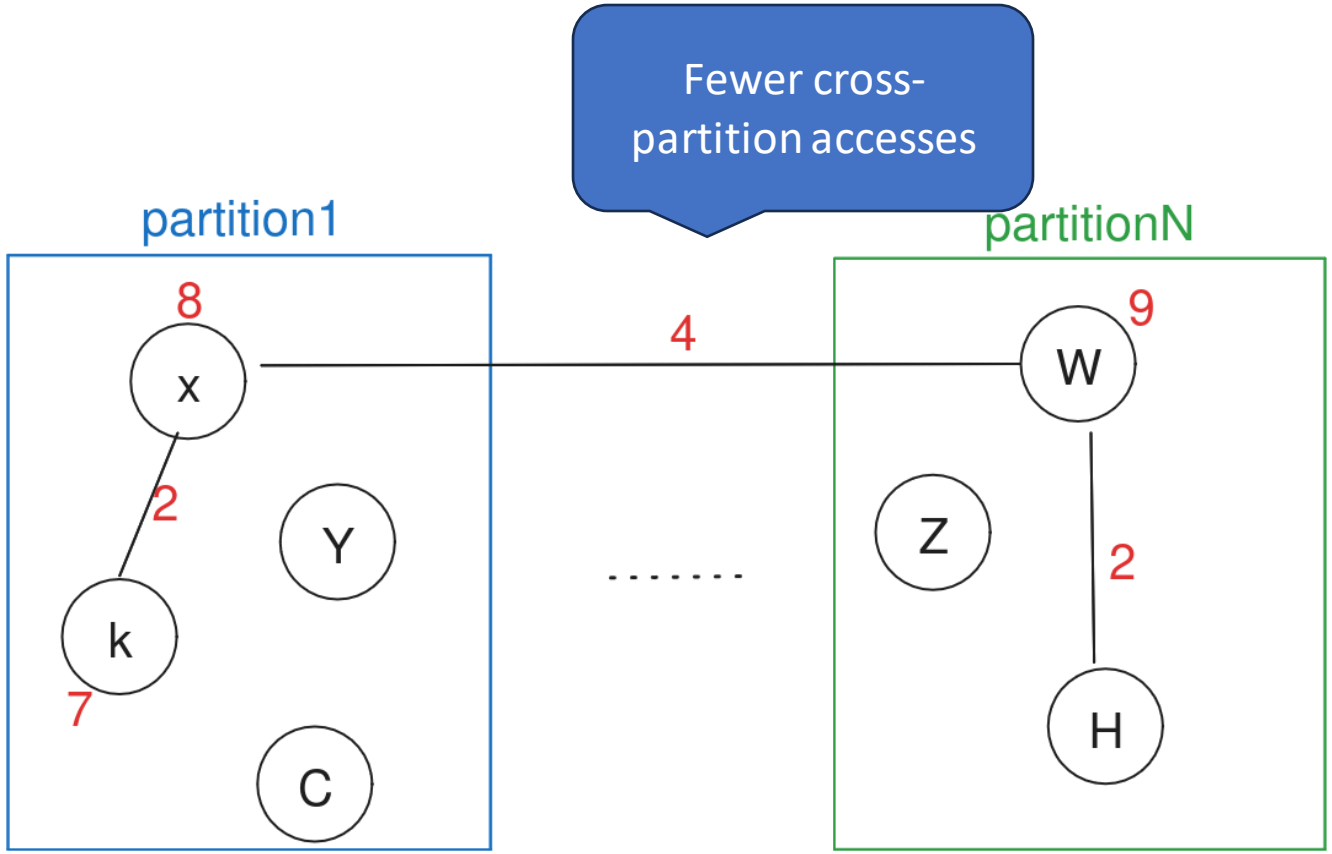
Problem statement

- How?
- A graph representation
- Each Key is a vertex
- Each Edge is an access between 2 vertices
- Key weight = # accesses
- Edge weight = # access 2 keys



Problem statement

- Objective:
- Balanced state partitions
- Fewer multi-partition operation/synchronization



Problem optimization

Phase 1: finds a balanced partition

- **Phase 1:**
  - An integer variable "Y", which means the slack to pack all vertices into partitions

$$x_{ip} = \begin{cases} 1, & \text{if vertex } i \text{ belongs to partition } p, \\ 0, & \text{otherwise.} \end{cases}$$

  - "k" partitions
  - Set of k partitions P = {p1, …, pk}
  - ci for vertex i = weight for vertex
  - wij for edge ij = weight for edge

Constraint (2): each partition is not too "heavy" since we want to minimize "Y"

$$\begin{aligned} \min \quad & y \\ \text{subject to:} \quad & \textstyle\sum_{p \in P} x_{ip} = 1, \forall i \in V && (1) \\ & \textstyle\sum_{i \in V} c_i x_{ip} \leq \frac{C}{k} + y, \forall p \in P && (2) \\ & x_{ip} \in \{0, 1\}, \forall i \in V \text{ and } \forall p \in P && (3) \\ & y \in \mathbb{Z}^+. && (4) \end{aligned}$$

Problem optimization

Phase 2: Finds a minimum cut considering the balanced partitions given

- **Phase 2:**
  - Finds a minimum cut considering the balanced partitions given
  - a set of k partitions $P = \{p_1, \ldots, p_k\}$ and $C = \sum_{i \in V} c_i$.

Constraints 7, 8, and 9: find the edges inside partitions

$$z_{ijp} = \begin{cases} 1, & \text{if edge } i-j \text{ has the both vertices } i \text{ and } j \text{ in partition } p, \\ 0, & \text{otherwise.} \end{cases}$$

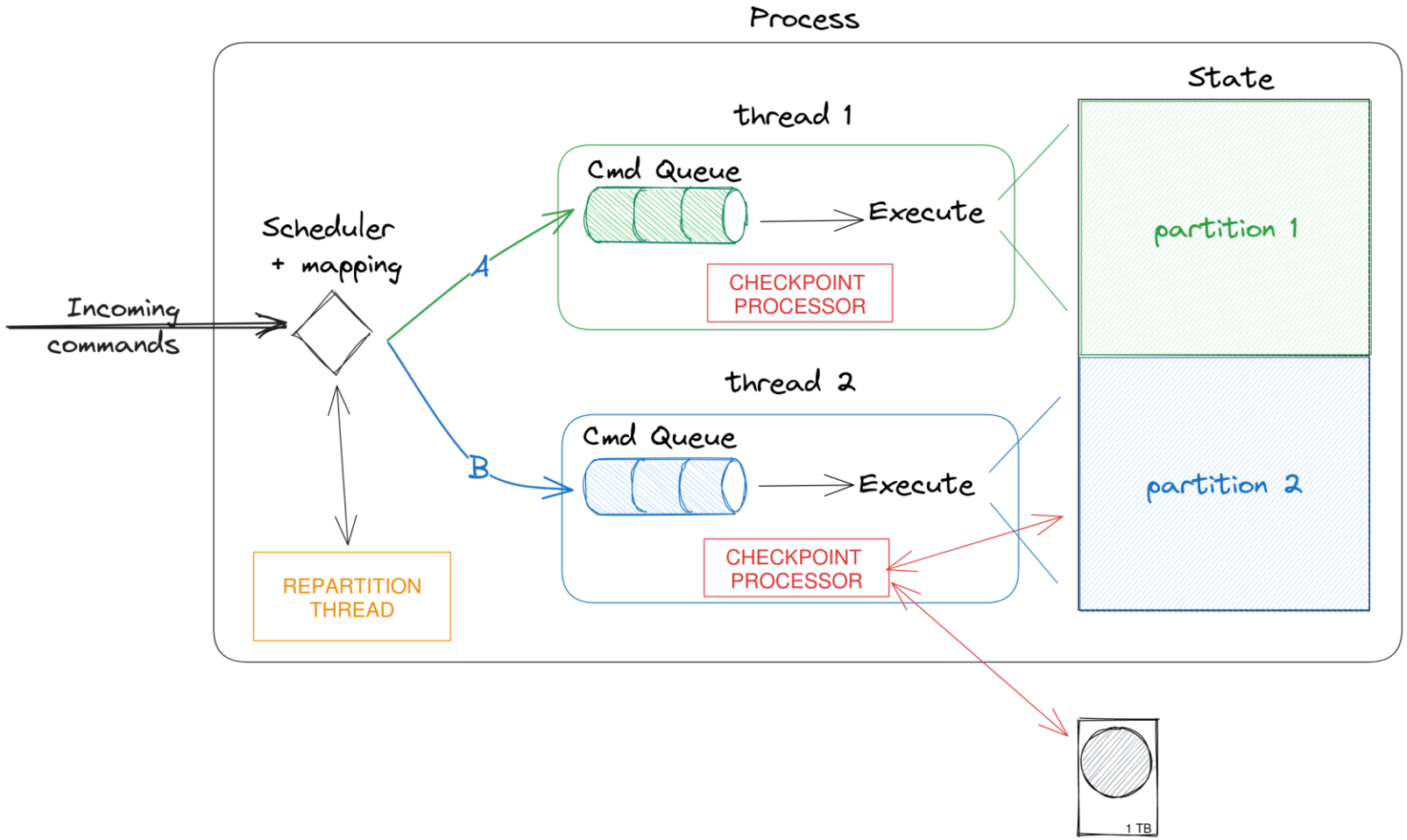
$$M = \frac{C}{k} + y.$$

$$\begin{aligned} \min \quad & \textstyle\sum_{i-j \in E} w_{ij}(1 - \sum_{p \in P} z_{ijp}) \\ \text{subject to:} \quad & \textstyle\sum_{p \in P} x_{ip} = 1, \forall i \in V && (5) \\ & \textstyle\sum_{i \in V} c_i x_{ip} \leq M, \forall p \in P && (6) \\ & z_{ijp} \leq x_{ip}, \forall i-j \in E \text{ and } \forall p \in P && (7) \\ & z_{ijp} \leq x_{jp}, \forall i-j \in E \text{ and } \forall p \in P && (8) \\ & z_{ijp} \geq \tfrac{1}{2}(x_{ip} + x_{jp}) - \tfrac{1}{2}, \forall i-j \in E, \\ & \qquad \forall p \in P && (9) \\ & x_{ip} \in \{0,1\}, \forall i \in V \text{ and } \forall p \in P && (10) \\ & z_{ijp} \in \{0,1\}, \forall i-j \in E. && (11) \end{aligned}$$

Evaluation

Prototype architecture

- State partitioned
- Checkpoint partitioned
- **Scheduler with a graph partition mapping**
- Repartitioning thread
- **Checkpoint & repartition are synced**



Evaluation

- Base values using Round-robin
- METIS repartitioning algorithm

Evaluation
Workload

Workload	Read %	Write %	Range Scan %	Characteristic	# requests	Has edges
YCSB-A	50%	50%	-%	**Single-variable**	94mi requests in 8mi unique keys	N
YCSB-D	95%	5%	-%	**Single-variable (MOST RECENT)**	110mi requests in 10mi unique keys	N
YCSB-E	0%	5%	95%	**Multi-variable**	6mi requests in 10mi unique keys	Y

*Ignored YCSB-B and YCSB-C workloads as we anticipated that they wouldn't produce very interesting results, mostly because they have similar distribution as workload YCSB-A but 95% and 100% of read respectively.

Evaluation

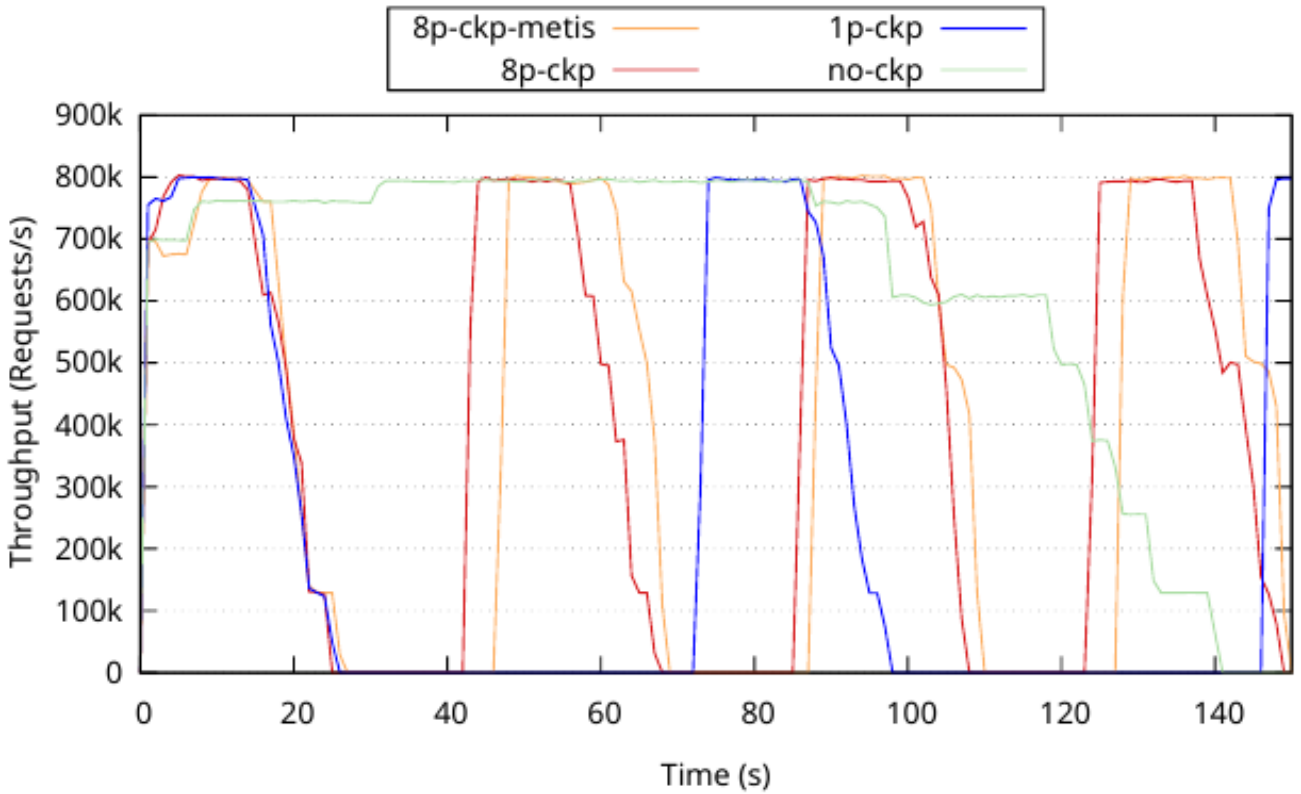
Metrics

- Throughput over time
- Makespan:
  - Total elapsed time to execute all operations
- Request distribution among partitions
- Cross-border requests
- Checkpoint times and sizes
- Checkpoint vs repartition times

Evaluation

Throughput – YCSB-A (50% Write, 50% Read, Single Key)

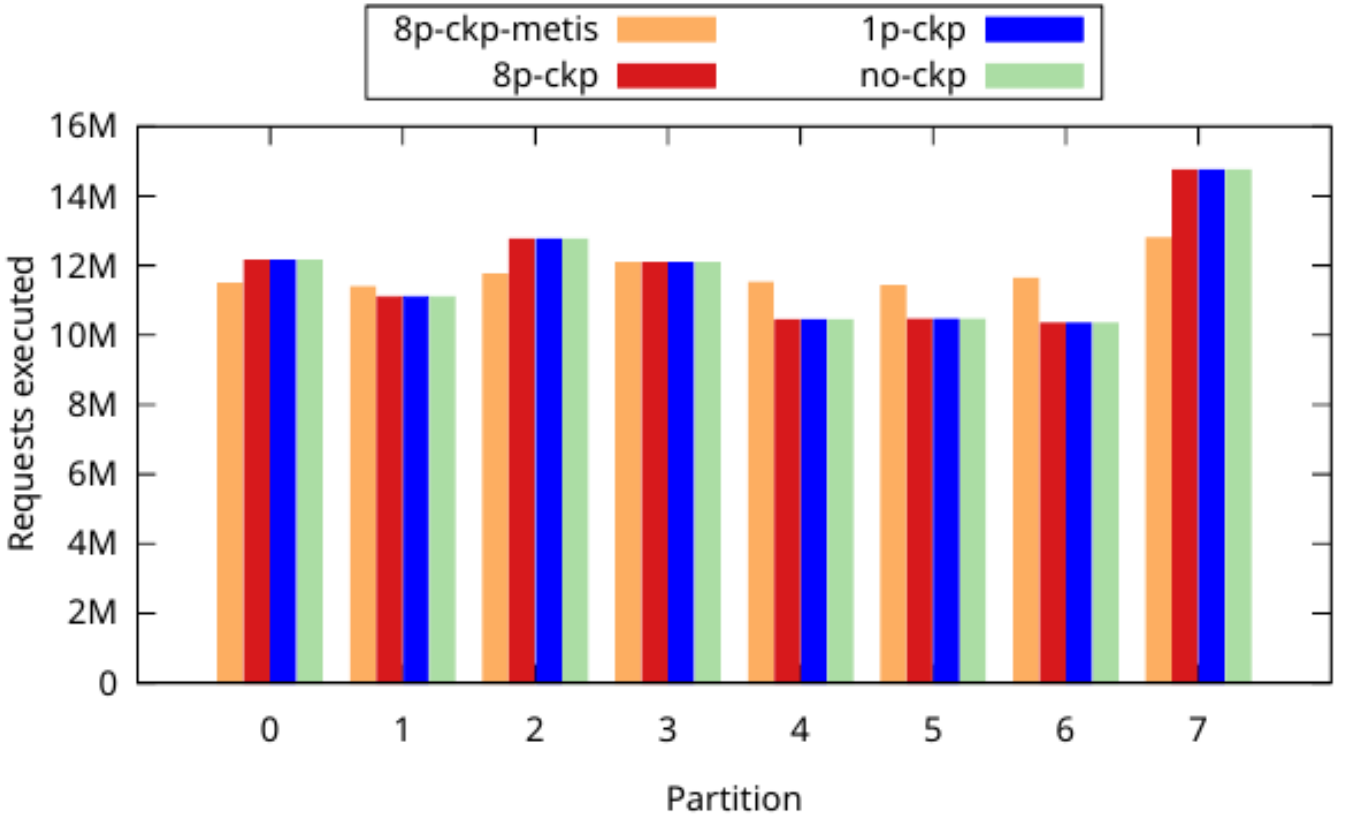
- Throughput:
  - Green: no checkpoint at all
  - Blue: single/1 partition checkpoint
  - Red: 8 partitions checkpoint (round robin)
  - Orange: 8 partitions checkpoint (METIS)
- Results:
  - Parallel version performing better despite the threads-overhead (short periods of no-throughput)



Evaluation

Access Distribution– YCSB-A (50% Write, 50% Read, Single Key)

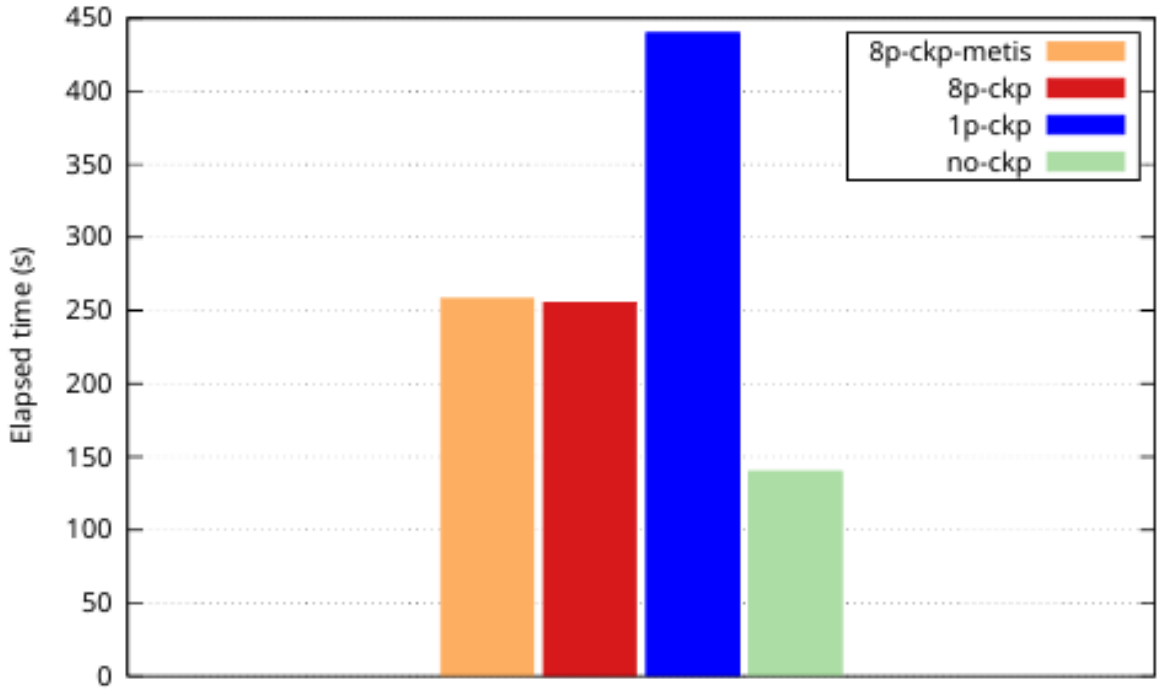
- Access Distribution:
  - Green: no checkpoint at all
  - Blue: single/1 partition checkpoint
  - Red: 8 partitions checkpoint (round robin)
  - Orange: 8 partitions checkpoint (METIS)
- Results:
  - METIS distributing well the requests among the partitions, better than Round-robin
  - Partition 7 with less data



Evaluation

Makespan– YCSB-A (50% Write, 50% Read, Single Key)

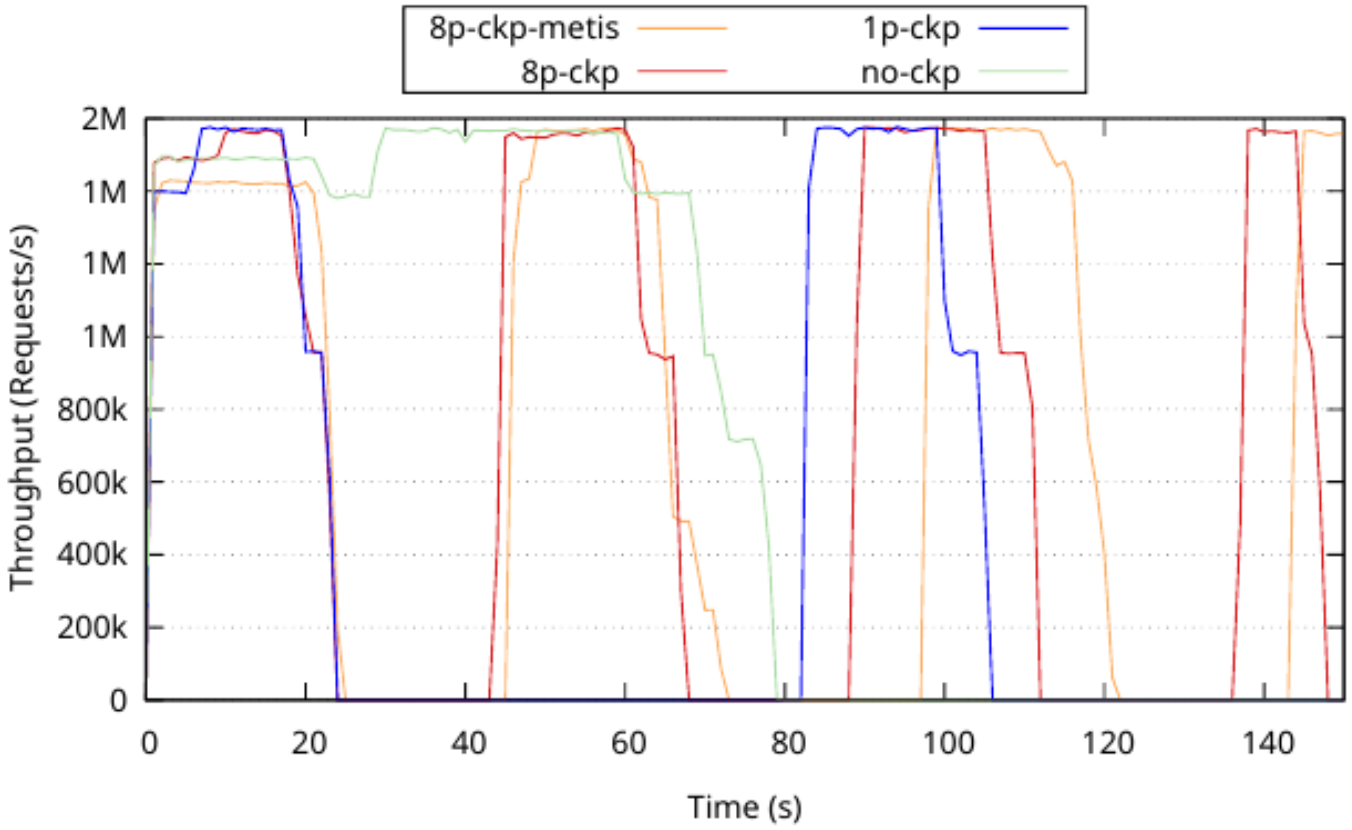
- Makespan (how long to complete):
  - Green: no checkpoint at all
  - Blue: single/1 partition checkpoint
  - Red: 8 partitions checkpoint (round robin)
  - Orange: 8 partitions checkpoint (METIS)
- Results:
  - METIS version has similar elapsed-time as Round-robin version despite the costs of running the algo
  - Better distribution compensates the cost for execution and checkpointing



Evaluation

Throughput – YCSB-D (5% Write, 95% Read, Single Key, Reads on Most Recent Inserted Keys)

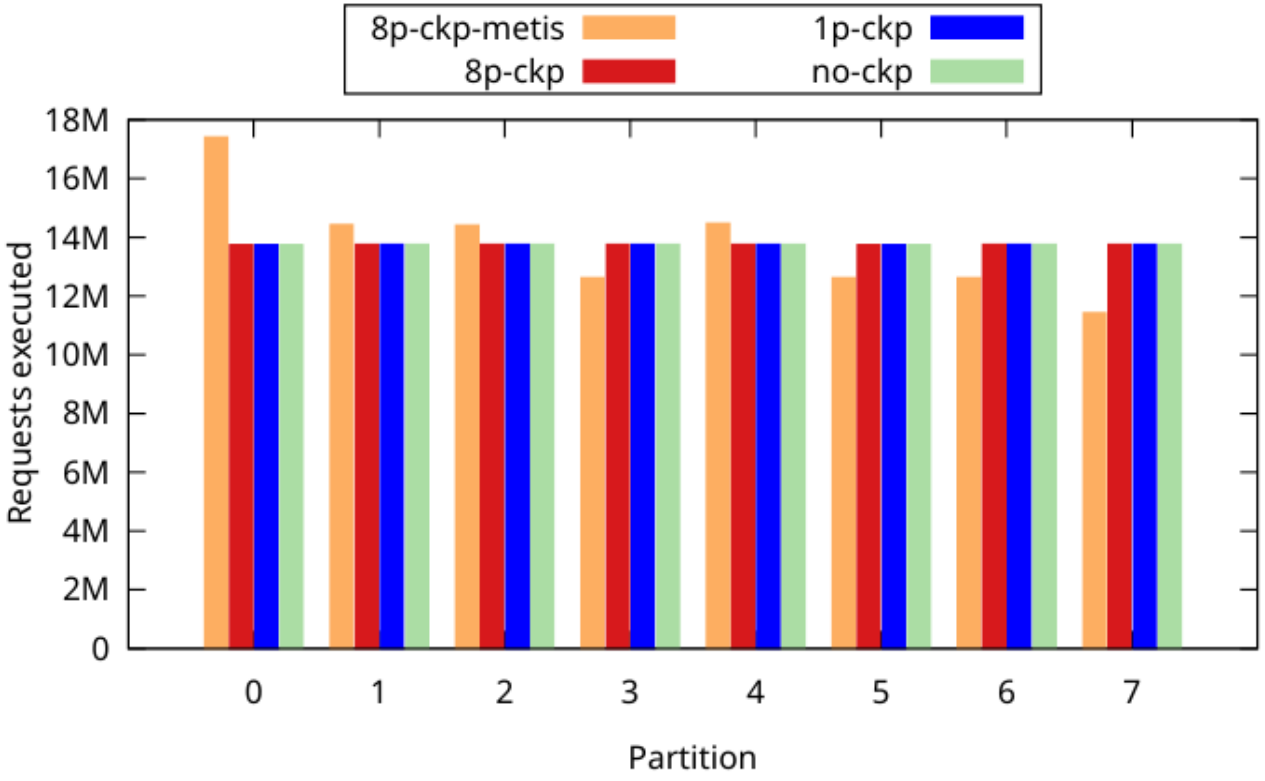
- Throughput:
  - Green: no checkpoint at all
  - Blue: single/1 partition checkpoint
  - Red: 8 partitions checkpoint (round robin)
  - Orange: 8 partitions checkpoint (METIS)
- Results:
  - Like the YCSB-A (previously presented)



Evaluation

Access distribution– YCSB-D (5% Write, 95% Read, Single Key, Reads on Most Recent Inserted Keys)

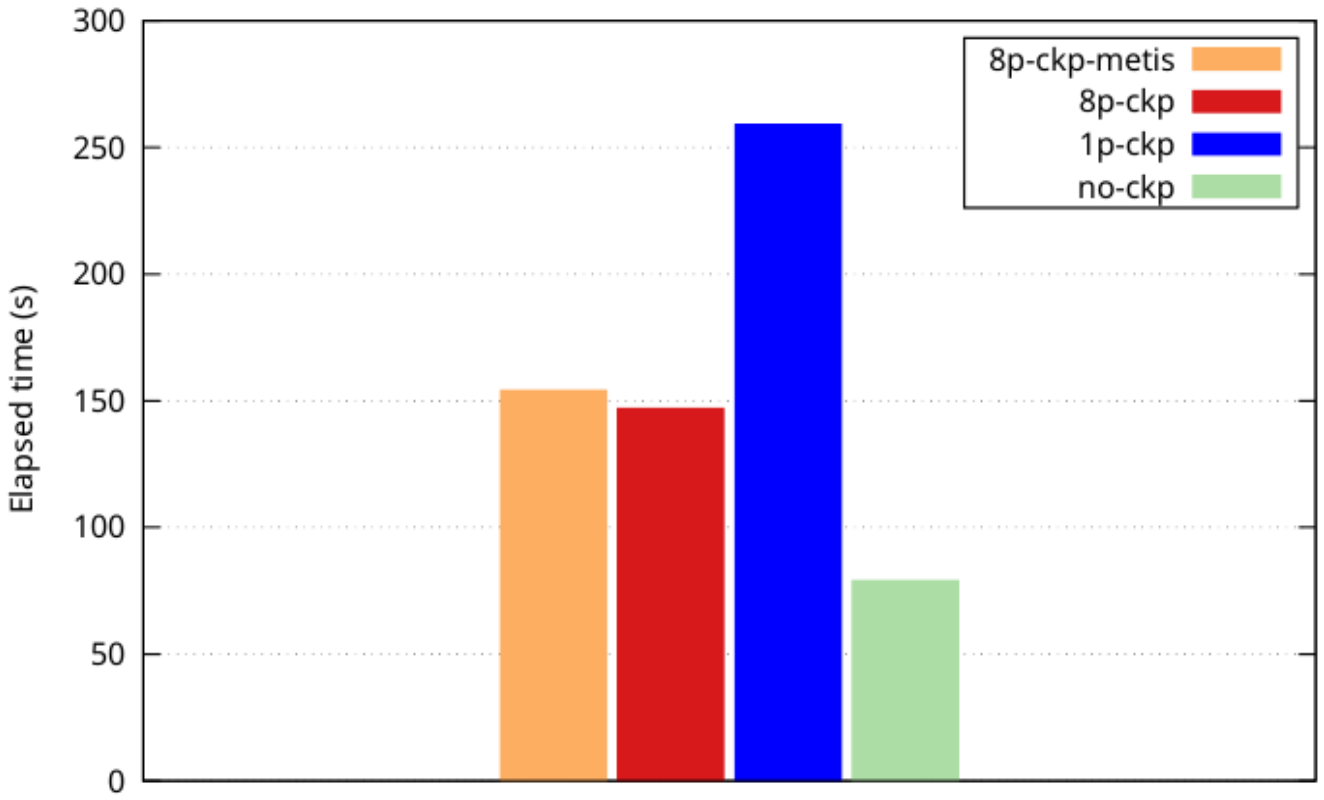
- Access Distribution:
  - Green: no checkpoint at all
  - Blue: single/1 partition checkpoint
  - Red: 8 partitions checkpoint (round robin)
  - Orange: 8 partitions checkpoint (METIS)
- Results:
  - METIS failed to balance the partitions.
  - Cause: novelty of most of the request's key
  - Round-robin has a tiny variation, distributing very well the data among the partitions



Evaluation

Makespan– YCSB-D (5% Write, 95% Read, Single Key, Reads on Most Recent Inserted Keys)

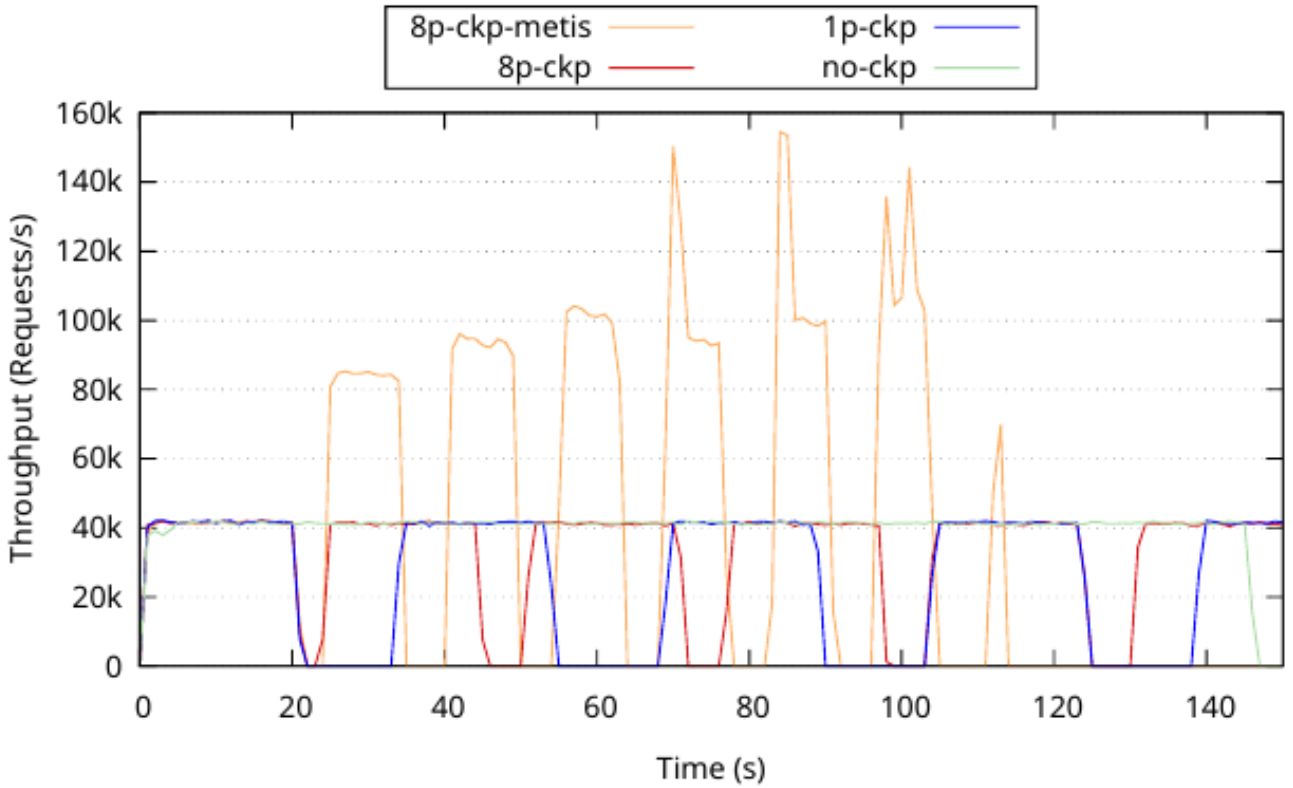
- Makespan (how long to complete):
  - Green: no checkpoint at all
  - Blue: single/1 partition checkpoint
  - Red: 8 partitions checkpoint (round robin)
  - Orange: 8 partitions checkpoint (METIS)
- Results:
  - METIS version has a noticeable cost – higher than the round-robin version



Evaluation

Throughput – YCSB-E (5% Write, 95% Read Scan up to 8 keys)

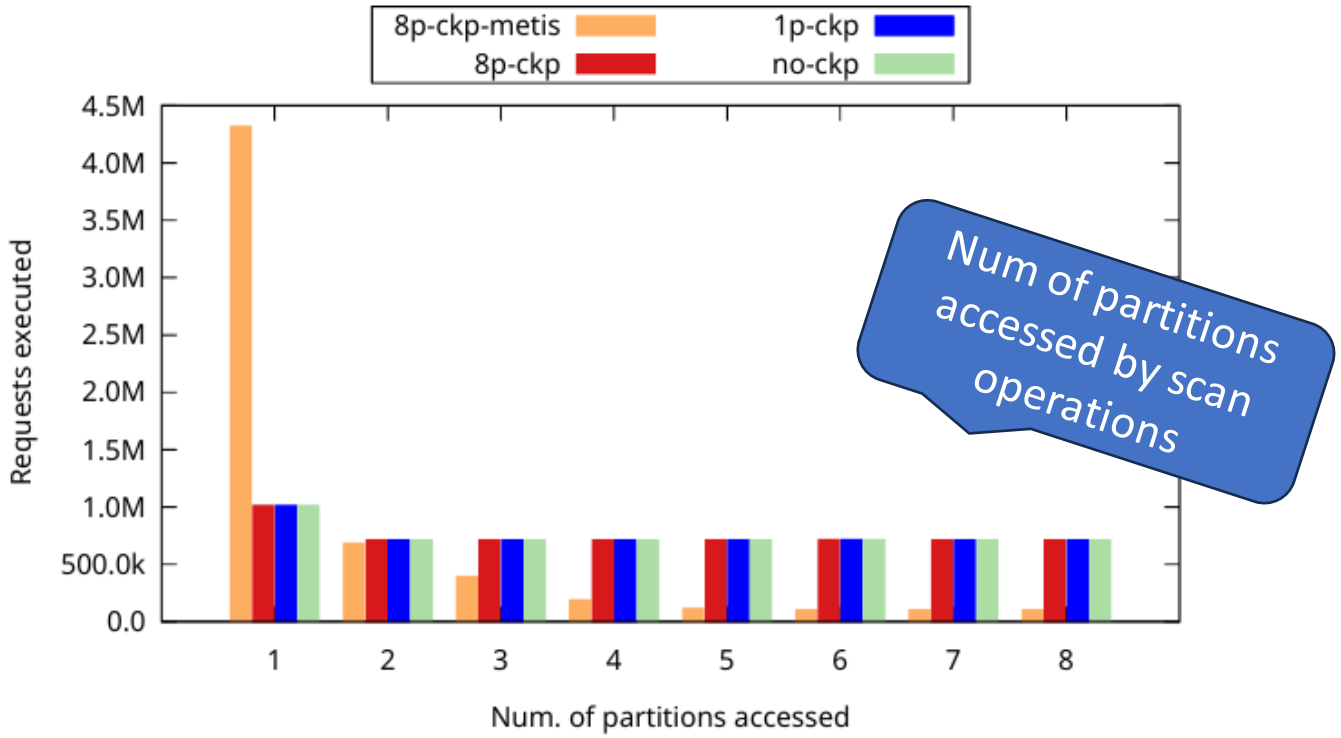
- Throughput:
  - Green: no checkpoint at all
  - Blue: single/1 partition checkpoint
  - Red: 8 partitions checkpoint (round robin)
  - Orange: 8 partitions checkpoint (METIS)
- Results:
  - METIS performing way better
  - Cause: fewer cross-partition access



Evaluation

Cross-border access– YCSB-E (5% Write, 95% Read Scan up to 8 keys)

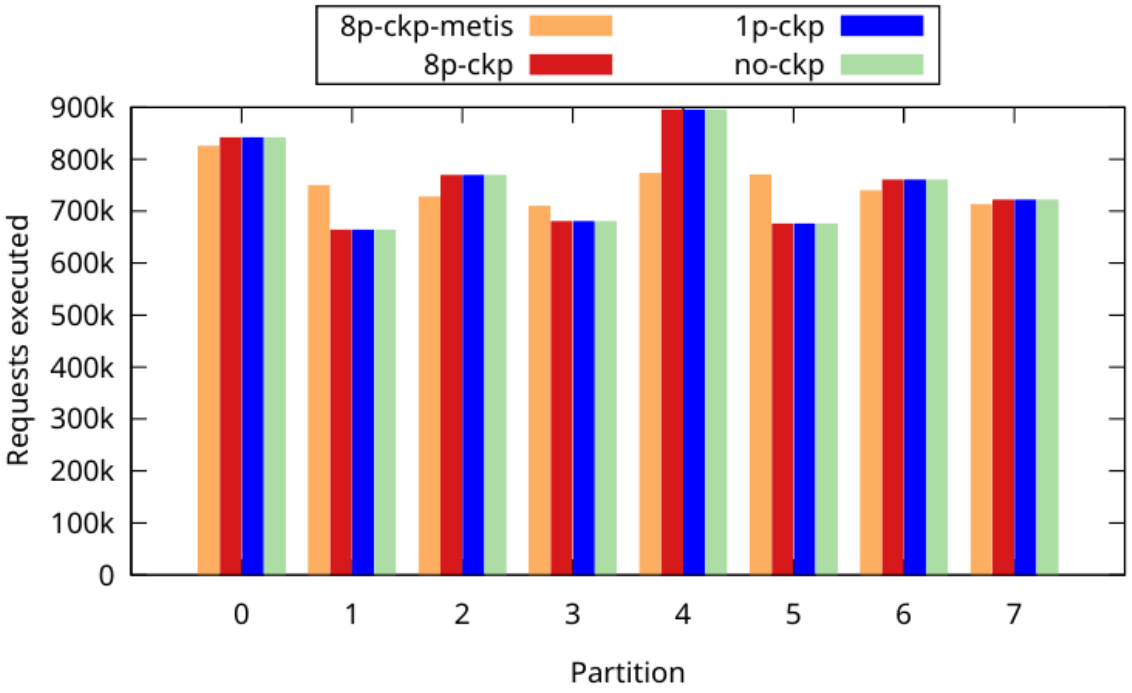
- Cross-border:
  - Green: no checkpoint at all
  - Blue: single/1 partition checkpoint
  - Red: 8 partitions checkpoint (round robin)
  - Orange: 8 partitions checkpoint (METIS)
- Results:
  - METIS able to move scan operations to the same partition (independently of the partition id)



Evaluation

Access distribution– YCSB-E (5% Write, 95% Read Scan up to 8 keys)

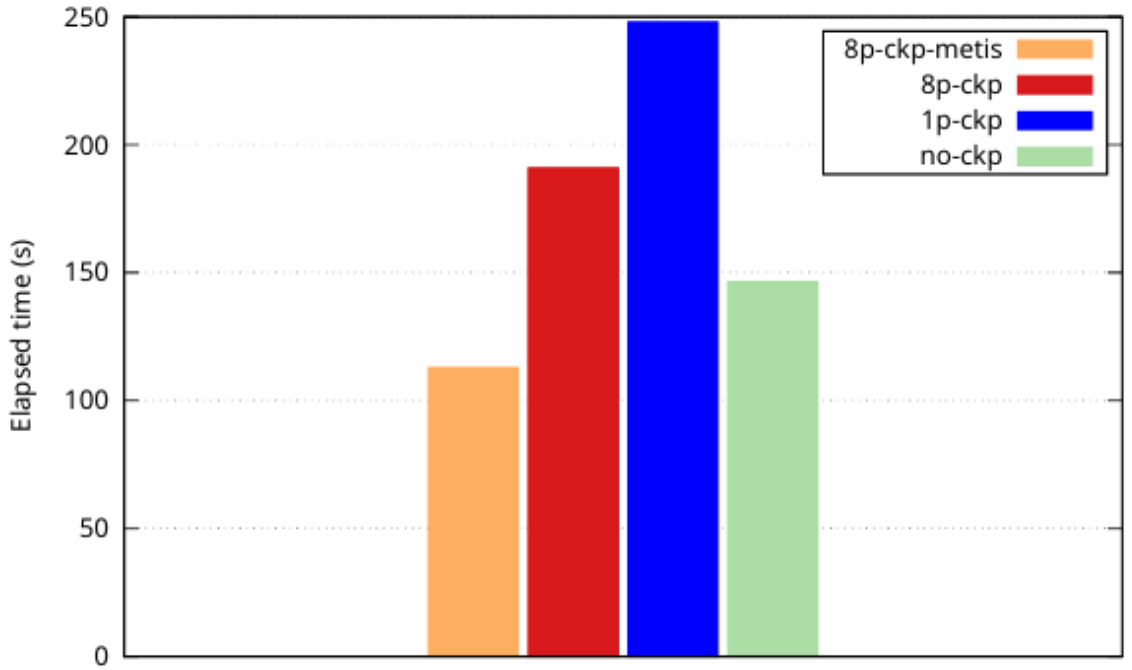
- Access Distribution:
  - Green: no checkpoint at all
  - Blue: single/1 partition checkpoint
  - Red: 8 partitions checkpoint (round robin)
  - Orange: 8 partitions checkpoint (METIS)
- Results:
  - METIS distributing well the requests among the partitions, better than Round-robin
  - Partition 4 with less data



Evaluation

Makespan– YCSB-E (5% Write, 95% Read Scan up to 8 keys)

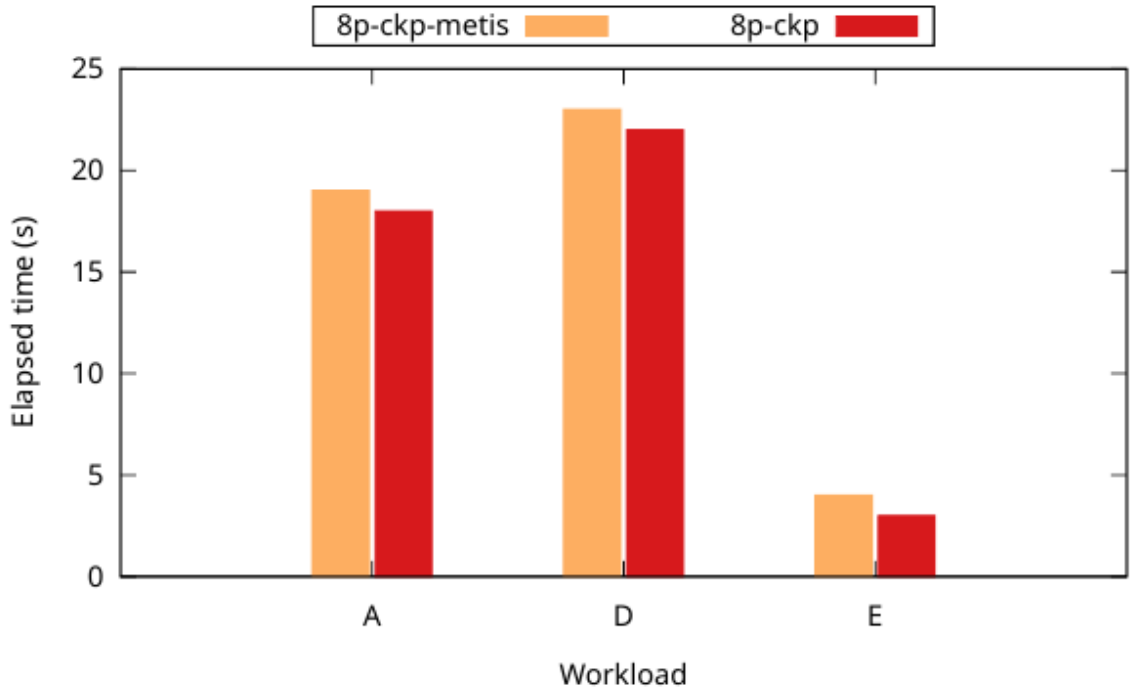
- Makespan:
  - Green: no checkpoint at all
  - Blue: single/1 partition checkpoint
  - Red: 8 partitions checkpoint (round robin)
  - Orange: 8 partitions checkpoint (METIS)
- Results:
  - METIS performing way better



Evaluation

Checkpoint elapsed time

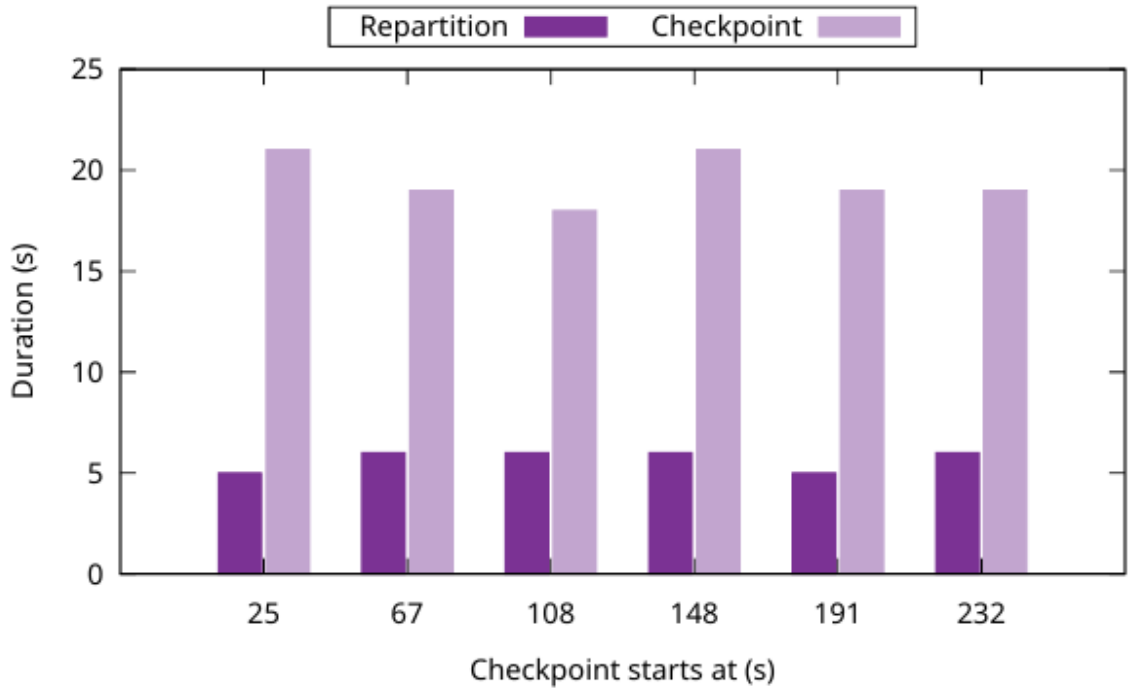
- Checkpoint elapsed time:
  - Red: 8 partitions checkpoint (round robin)
  - Orange: 8 partitions checkpoint (METIS)
- Results
  - Average elapsed time for each workload checkpoint
  - Both versions have similar time
  - Running METIS/Repartition has an extra cost but it's compensated by requests and checkpointing processing well the partitions



Evaluation

Checkpoint elapsed time

- Checkpoint elapsed time for YCSB-A:
  - Dark Purple: repartition elapsed time
  - Light Purple: checkpointing elapsed time
- Result
  - Repartition takes shorter time than checkpointing
  - Repartition does not affect dramatically or make the checkpoint waits for it to finish



Conclusion

- It's possible to have ***balanced state partitions*** when ***history*** can be used
- **Balanced state partitions** contributes to the total checkpointing elapsed time
- Having keys of multi-variable operations in the same partition **reduces the synchronization**
- Use **underutilized CPU** for **repartition** while **checkpointing** is doing **I/O** is **OK**
- Benefits of repartition compensates the costs, mainly for multi-variable operations:
  - **Costs** for other workloads are **marginally higher without adding much benefits**

Thanks

- Thanks to the partial research funding provided by FAPESC and PROPESQ

fapesc

Fundação de Amparo à Pesquisa e Inovação do Estado de Santa Catarina

UNIVERSIDADE FEDERAL DE SANTA CATARINA

Pró-Reitoria de Pesquisa e Inovação

PROPESQ

Achieving Enhanced Performance Combining Checkpointing and Dynamic State Partitioning

Henrique Goulart

João Trombeta
Álvaro Franco
Odorico Mendizabal

Universidade Federal de Santa Catarina - UFSC

SBAC-PAD 2023

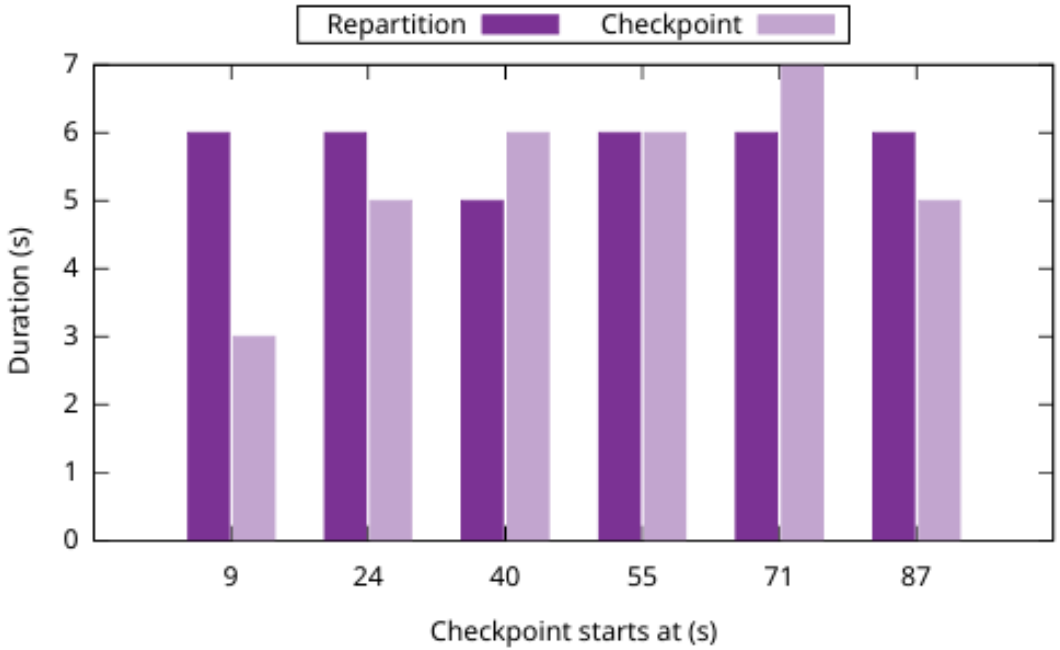
LaPeSD

UFSC

Evaluation

Checkpoint elapsed time

- Checkpoint elapsed time for YCSB-A:
  - Dark Purple: repartition elapsed time
  - Light Purple: checkpointing elapsed time
- Using 1KB value



Problem optimization

Phase 1: finds a balanced partition

- **Phase 1:**
  - An integer variable "Y", which means the slack to pack all vertices into partitions

$$x_{ip} = \begin{cases} 1, & \text{if vertex } i \text{ belongs to partition } p, \\ 0, & \text{otherwise.} \end{cases}$$

  - "k" partitions
  - Set of k partitions P = {p1, …, pk}
  - ci for vertex i = weight for vertex
  - wij for edge ij = weight for edge

Constraint (1): each vertex is in one partition

$$\begin{aligned} \min \quad & y \\ \text{subject to:} \quad & \textstyle\sum_{p \in P} x_{ip} = 1, \forall i \in V & (1) \\ & \textstyle\sum_{i \in V} c_i x_{ip} \leq \frac{C}{k} + y, \forall p \in P & (2) \\ & x_{ip} \in \{0, 1\}, \forall i \in V \text{ and } \forall p \in P & (3) \\ & y \in \mathbb{Z}^+. & (4) \end{aligned}$$

Problem optimization

Phase 1: finds a balanced partition

- **Phase 1:**
  - An integer variable "Y", which means the slack to pack all vertices into partitions

$$x_{ip} = \begin{cases} 1, & \text{if vertex } i \text{ belongs to partition } p, \\ 0, & \text{otherwise.} \end{cases}$$

  - "k" partitions
  - Set of k partitions P = {p1, …, pk}
  - ci for vertex i = weight for vertex
  - wij for edge ij = weight for edge

Constraint (2): each partition is not too "heavy" since we want to minimize "Y"

$$\begin{aligned} \min \quad & y \\ \text{subject to:} \quad & \textstyle\sum_{p \in P} x_{ip} = 1, \forall i \in V && (1) \\ & \textstyle\sum_{i \in V} c_i x_{ip} \leq \frac{C}{k} + y, \forall p \in P && (2) \\ & x_{ip} \in \{0, 1\}, \forall i \in V \text{ and } \forall p \in P && (3) \\ & y \in \mathbb{Z}^+. && (4) \end{aligned}$$

Problem optimization

Phase 1: finds a balanced partition

- **Phase 1:**
  - An integer variable "Y", which means the slack to pack all vertices into partitions

$$x_{ip} = \begin{cases} 1, & \text{if vertex } i \text{ belongs to partition } p, \\ 0, & \text{otherwise.} \end{cases}$$

  - "k" partitions
  - Set of k partitions P = {p1, …, pk}
  - ci for vertex i = weight for vertex
  - wij for edge ij = weight for edge

$$\min \ y$$

$$\text{subject to:}$$

$$\sum_{p} \qquad (1)$$

$$\sum_{i \in V} \qquad (2)$$

$$x_{ip} \in \{0, 1\}, \forall i \in V \text{ and } \forall p \in P \qquad (3)$$

$$y \in \mathbb{Z}^{+}. \qquad (4)$$

Constraint (3 and 4):
xip is binary
y is an integer

Problem optimization

Phase 2: Finds a minimum cut considering the balanced partitions given

- **Phase 2:**
  - Finds a minimum cut considering the balanced partitions given
  - a set of k partitions $P = \{p_1, \ldots, p_k\}$ and $C = \sum_{i \in V} c_i$.

Constraints 5 and 6: ensure balanced partitions

$$z_{ijp} = \begin{cases} 1, & \text{if edge } i-j \text{ has the both vertices } i \text{ and } j \text{ in partition } p, \\ 0, & \text{otherwise.} \end{cases}$$

$$M = \frac{C}{k} + y.$$

$$\begin{aligned} \min \quad & \textstyle\sum_{i-j \in E} w_{ij}(1 - \sum_{p \in P} z_{ijp}) \\ \text{subject to:} \quad & \textstyle\sum_{p \in P} x_{ip} = 1, \forall i \in V && (5) \\ & \textstyle\sum_{i \in V} c_i x_{ip} \leq M, \forall p \in P && (6) \\ & z_{ijp} \leq x_{ip}, \forall i-j \in E \text{ and } \forall p \in P && (7) \\ & z_{ijp} \leq x_{jp}, \forall i-j \in E \text{ and } \forall p \in P && (8) \\ & z_{ijp} \geq \tfrac{1}{2}(x_{ip} + x_{jp}) - \tfrac{1}{2}, \forall i-j \in E, \\ & \qquad \forall p \in P && (9) \\ & x_{ip} \in \{0,1\}, \forall i \in V \text{ and } \forall p \in P && (10) \\ & z_{ijp} \in \{0,1\}, \forall i-j \in E. && (11) \end{aligned}$$

Problem optimization

Phase 2: Finds a minimum cut considering the balanced partitions given

- **Phase 2:**
  - Finds a minimum cut considering the balanced partitions given
  - a set of k partitions $P = \{p_1, \ldots, p_k\}$ and $C = \sum_{i \in V} c_i$.

Constraints 7, 8, and 9: find the edges inside partitions

$$z_{ijp} = \begin{cases} 1, & \text{if edge } i-j \text{ has the both vertices } i \text{ and } j \text{ in partition } p, \\ 0, & \text{otherwise.} \end{cases}$$

$$M = \frac{C}{k} + y.$$

$$\begin{aligned} \min \quad & \textstyle\sum_{i-j \in E} w_{ij}(1 - \sum_{p \in P} z_{ijp}) \\ \text{subject to:} \quad & \textstyle\sum_{p \in P} x_{ip} = 1, \forall i \in V && (5) \\ & \textstyle\sum_{i \in V} c_i x_{ip} \leq M, \forall p \in P && (6) \\ & z_{ijp} \leq x_{ip}, \forall i-j \in E \text{ and } \forall p \in P && (7) \\ & z_{ijp} \leq x_{jp}, \forall i-j \in E \text{ and } \forall p \in P && (8) \\ & z_{ijp} \geq \tfrac{1}{2}(x_{ip} + x_{jp}) - \tfrac{1}{2}, \forall i-j \in E, \\ & \qquad \forall p \in P && (9) \\ & x_{ip} \in \{0,1\}, \forall i \in V \text{ and } \forall p \in P && (10) \\ & z_{ijp} \in \{0,1\}, \forall i-j \in E. && (11) \end{aligned}$$

Problem optimization

Phase 2: Finds a minimum cut considering the balanced partitions given

- **Phase 2:**
  - Finds a minimum cut considering the balanced partitions given
  - a set of k partitions $P = \{p_1, \ldots, p_k\}$ and $C = \sum_{i \in V} c_i$.

Constraints 10 and 11: mean that the variables xip and zijp are binaries

$$z_{ijp} = \begin{cases} 1, & \text{if edge } i-j \text{ has the both vertices } i \text{ and } j \text{ in partition } p, \\ 0, & \text{otherwise.} \end{cases}$$

$$M = \frac{C}{k} + y.$$

$$\begin{aligned}
&\min && \textstyle\sum_{i-j \in E} w_{ij}(1 - \sum_{p \in P} z_{ijp}) \\
&\text{subject to:} && \textstyle\sum_{p \in P} x_{ip} = 1, \forall i \in V && (5) \\
& && \textstyle\sum_{i \in V} c_i x_{ip} \leq M, \forall p \in P && (6) \\
& && z_{ijp} \leq x_{ip}, \forall i-j \in E \text{ and } \forall p \in P && (7) \\
& && z_{ijp} \leq x_{jp}, \forall i-j \in E \text{ and } \forall p \in P && (8) \\
& && z_{ijp} \geq \tfrac{1}{2}(x_{ip} + x_{jp}) - \tfrac{1}{2}, \forall i-j \in E, \\
& && \qquad \forall p \in P && (9) \\
& && x_{ip} \in \{0,1\}, \forall i \in V \text{ and } \forall p \in P && (10) \\
& && z_{ijp} \in \{0,1\}, \forall i-j \in E. && (11)
\end{aligned}$$