An Evaluation of Checkpoint Recovery for MMOGs

Authors: Marcos Vaz Salles, Tuan Cao, Benjamin Sowell, Alan Demers, Johannes Gehrke, Christoph Koch, Walker White

Presented by: Kurt Teichman

Outline

- Introduction
- Architecture of a MMO
- Main Memory DBMS Recovery
  - *Algorithms (Check pointing techniques)*
- Experimental Setup
  - Check pointing Algorithmic Framework
- Simulation Model
- Experimental Stuff

Introduction

- MMOs have high update rates
  - the entire game state stored in memory
- Goal: Execute at frame rate (30-60hz/fps)
- Uniformity > Performance
- Traditional ARIES-style recovery will not be optimal for various types of MMO updates
  - Limited scalability (have to buy the expensive stuff)
  - Over-partition virtual worlds
  - Ad-hoc solutions

Architecture of a MMO

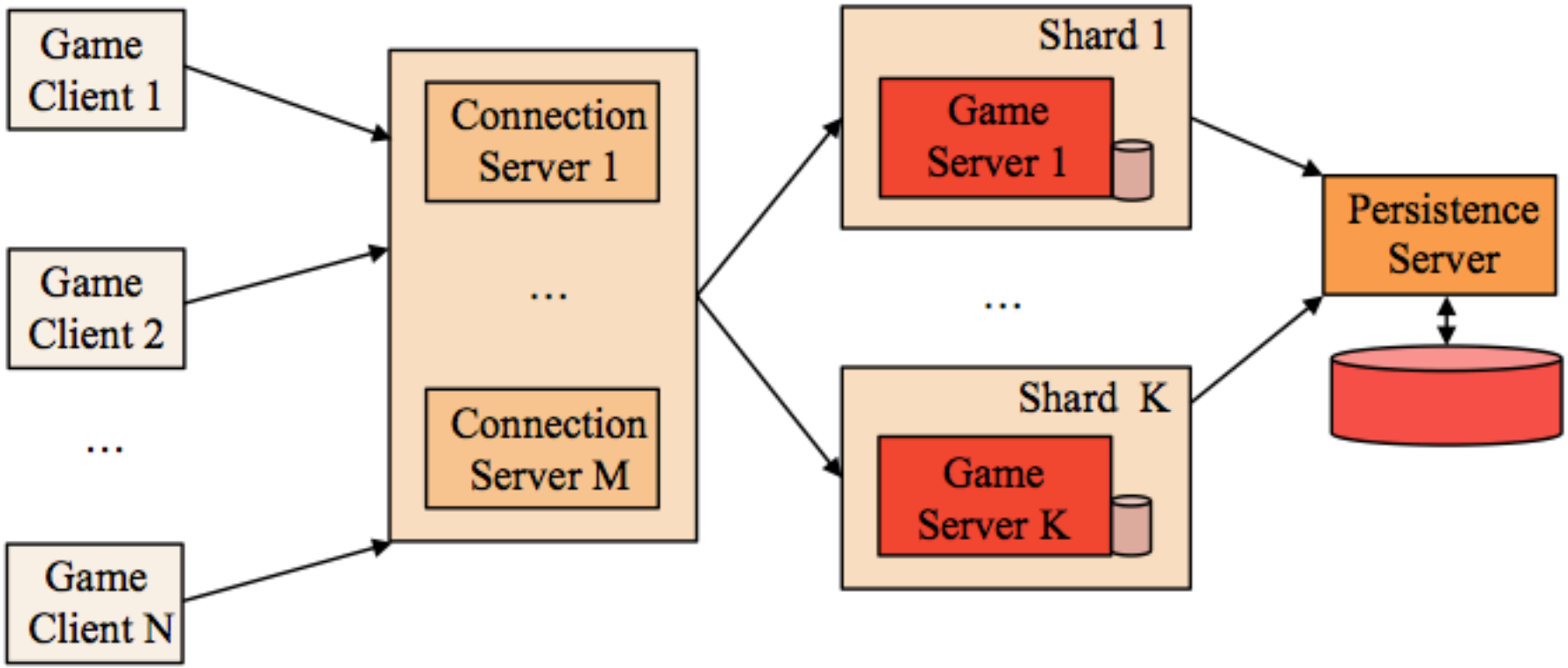


Figure 1: Architecture of a typical MMO.

Requirements

- Small overhead
  - ✦ Entire checkpointing process must fit into the game simulation
- Uniform overhead
  - ✦ Low latency, no hiccup in the game
- No data loss
  - ✦ Recover to the point of the crash

Requirements

➡ Performance Criteria

- Small overhead
  - ➡ Average Overhead Time
- Uniform overhead
  - ➡ Overhead Distribution
- No data loss
  - ➡ Checkpointing Time
  - ➡ Recovery Time

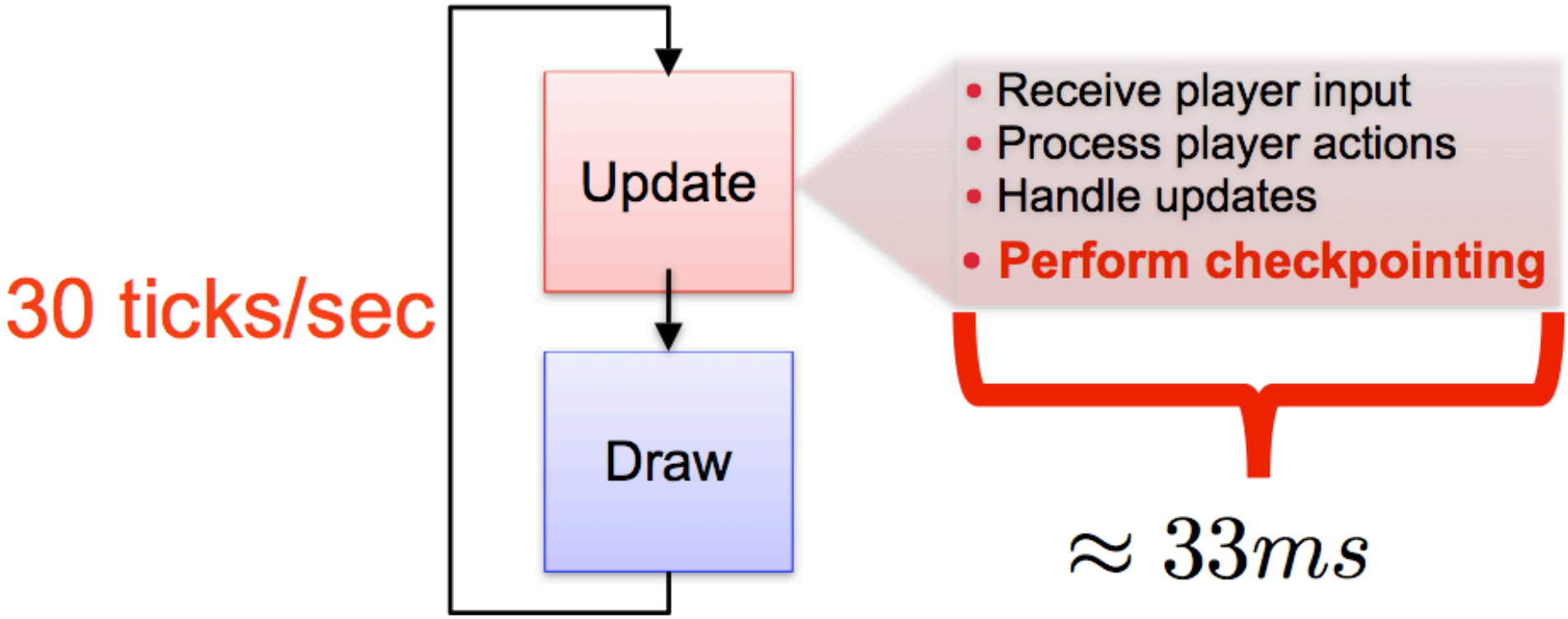
Main Memory DBMS Recovery

- In-memory copy timing
  - *Eager copy*
  - *Copy-on-update*
- Objects copied
  - *All objects*
  - *Dirty objects*
- Data Organization on disk
  - *Double-backup*
  - *Log files*

Main Memory DBMS Recovery Cont.

- Checkpoint Algorithms
  - Naïve-Snapshot
  - Dribble-and-Copy-on-Update
  - Atomic-Copy-Dirty-Objects
  - Partial-Redo
  - Copy-on-Update
  - Copy-on-Update-Partial-Redo

The Game Loop



The game state is consistent at the end of every tick.

Checkpointing Algorithms

	All Objects	Dirty Objects	Eager Copy	Copy On Update	Double Backup	Log
Naive-Snapshot	X		X			X
Dribble-And-Copy-On-Update	X			X		X
Atomic-Copy-Dirty-Objects		X	X		X	
Partial-Redo		X	X			X
Copy-On-Update		X		X	X	
Copy-On-Update-Partial-Redo		X		X		X

Checkpointing Algorithms

	All Objects	Dirty Objects	Eager Copy	Copy On Update	Double Backup	Log
Naive-Snapshot	X		X			X
Dribble-And-Copy-On-Update	X			X		X
Atomic-Copy-Dirty-Objects		X	X		X	
Partial-Redo		X	X			X
Copy-On-Update		X		X	X	
Copy-On-Update-Partial-Redo		X		X		X

Checkpointing Algorithms

Naive-Snapshot	
All Objects	Dirty Objects
Log	Double Backup
Eager Copy	COU

Partial-Redo	
All Objects	Dirty Objects
Log	Double Backup
Eager Copy	COU

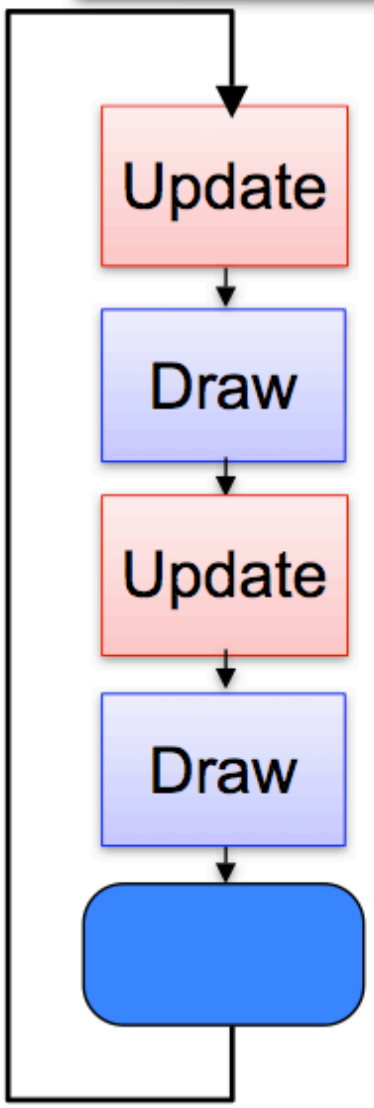
Copy-On-Update	
All Objects	Dirty Objects
Log	Double Backup
Eager Copy	COU

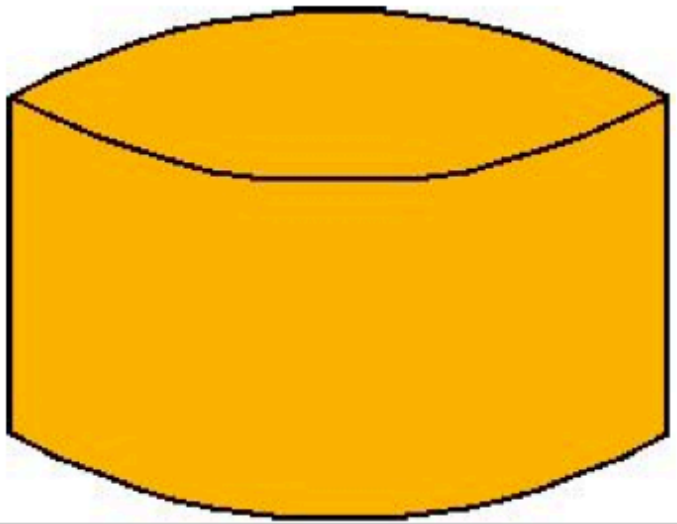
The Game State

Race	Strength	Agility	Stamina	Intellect	Spirit	Armor	Health
Human	108	73	99	29	46	146	2169
Dwarf	110	69	102	28	41	138	2199
Night Elf	105	78	98	29	42	156	2159
Gnome	103	76	98	33	42	152	2159
Draenei	109	70	98	30	44	140	2159
Orc	111	70	101	26	45	140	2189
Troll	109	75	100	25	43	150	2179
Undead	107	71	100	27	47	142	2179
Blood Elf	105	75	97	33	41	150	2149
Tauren	113	68	101	24	44	136	2298

Naive-Snapshot

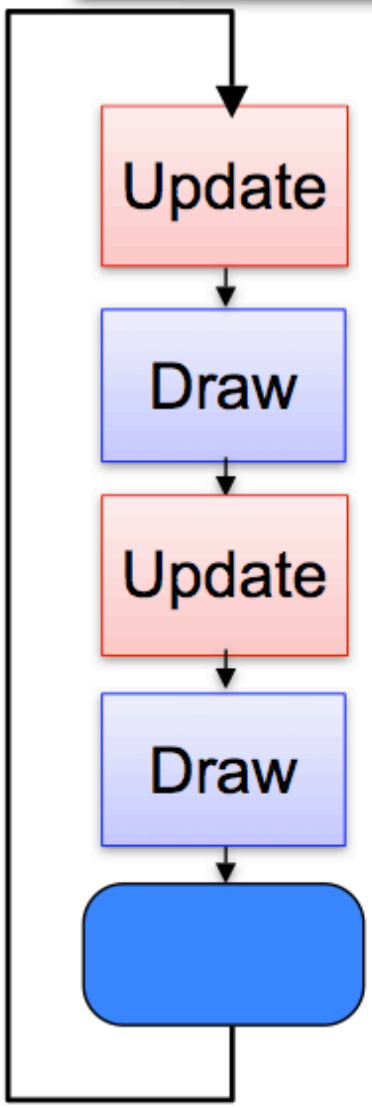
All Objects	Dirty Objects
Log	Double Backup
Eager Copy	COU

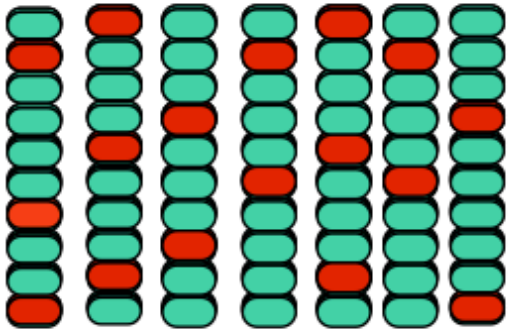


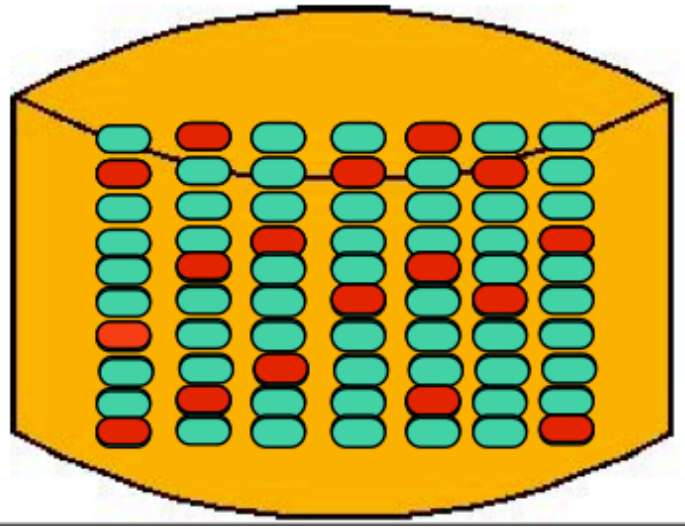


Naive-Snapshot

All Objects	Dirty Objects
Log	Double Backup
Eager Copy	COU

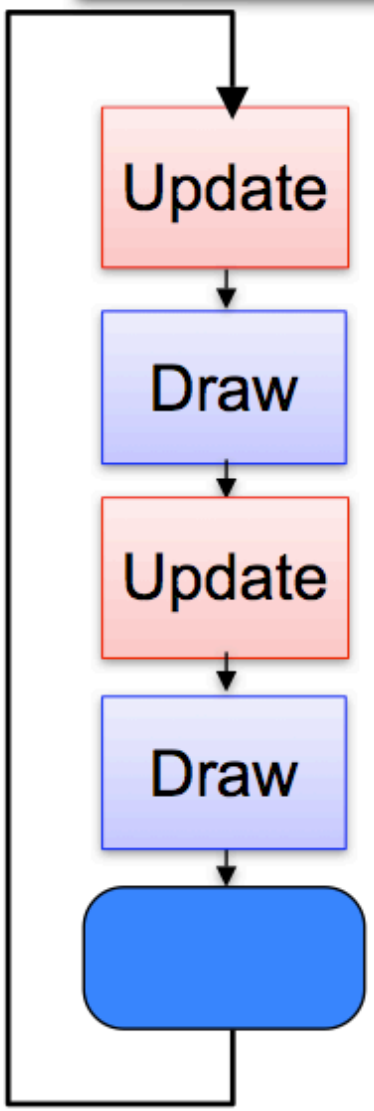


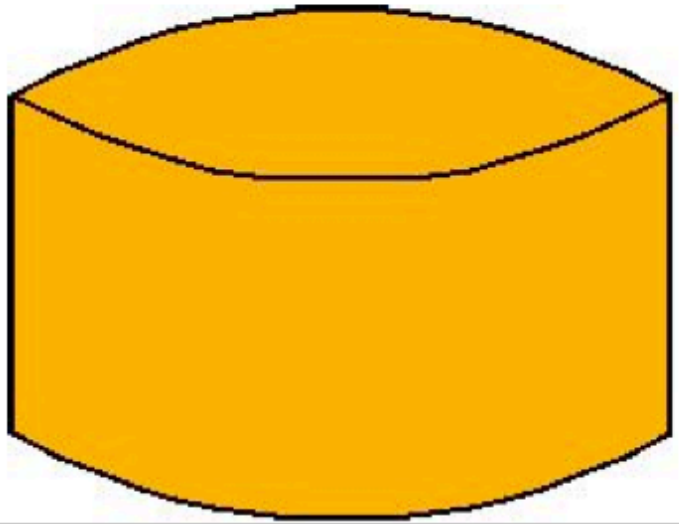




Partial-Redo

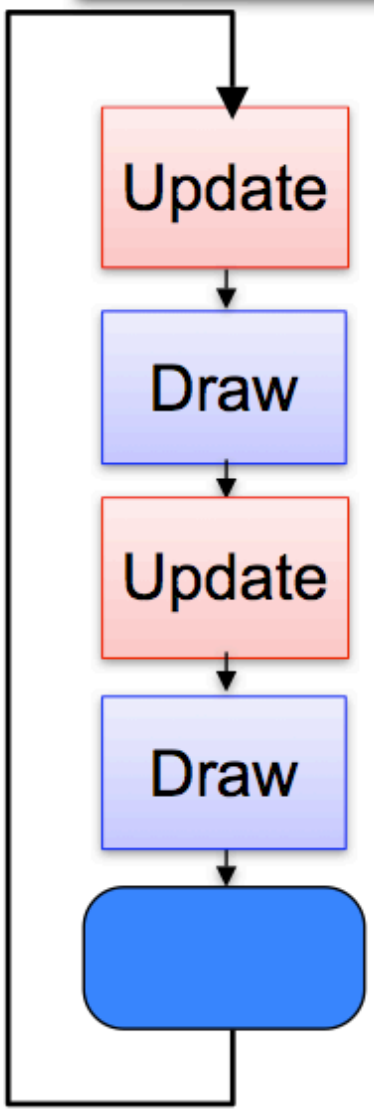
All Objects	Dirty Objects
Log	Double Backup
Eager Copy	COU

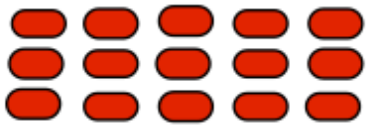


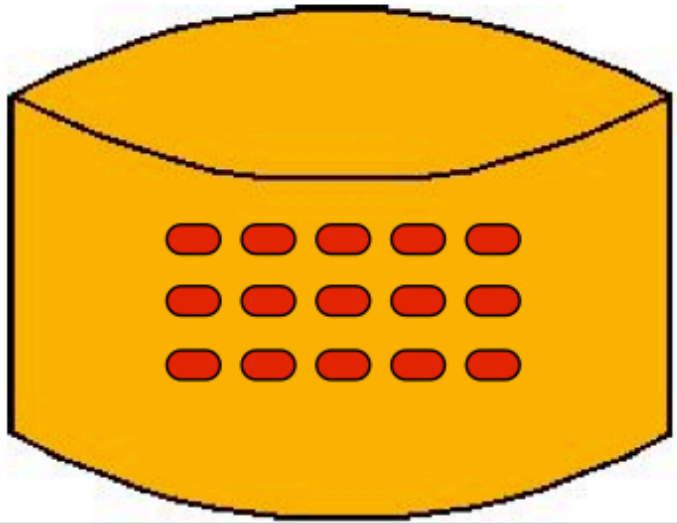


Partial-Redo

All Objects	Dirty Objects
Log	Double Backup
Eager Copy	COU

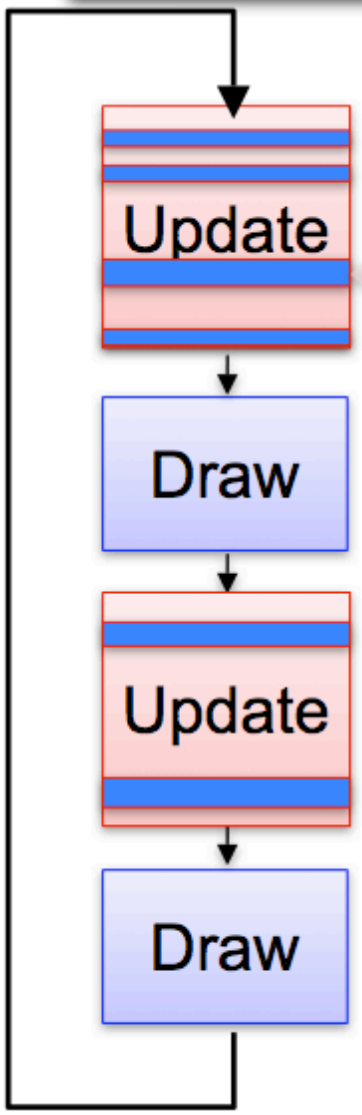




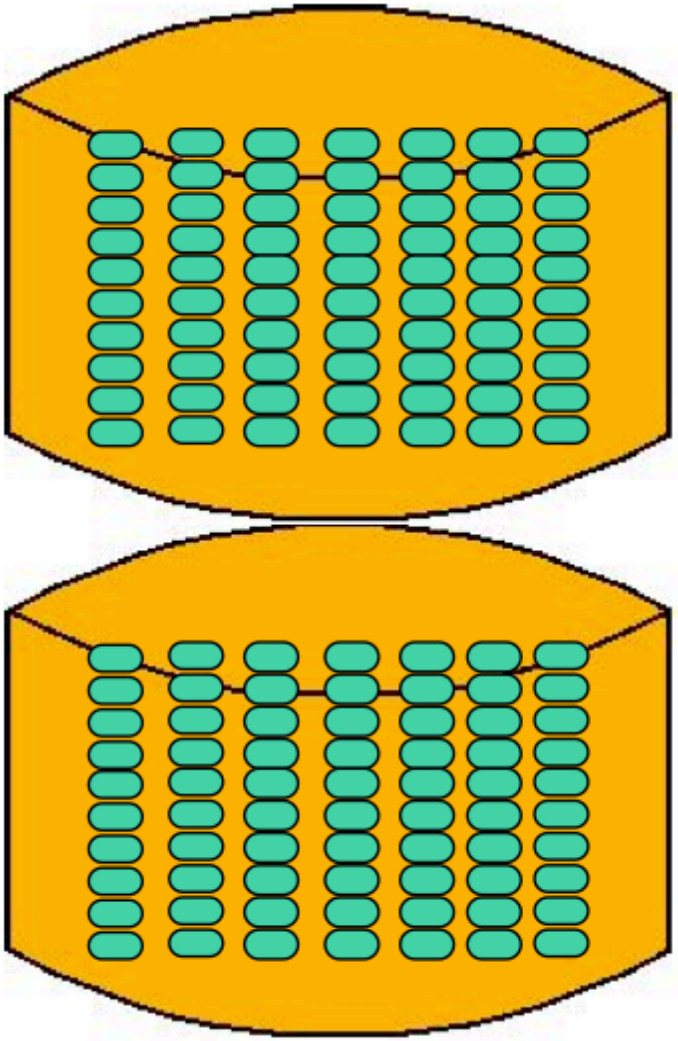


Copy-On-Update

All Objects	Dirty Objects
Log	Double Backup
Eager Copy	COU

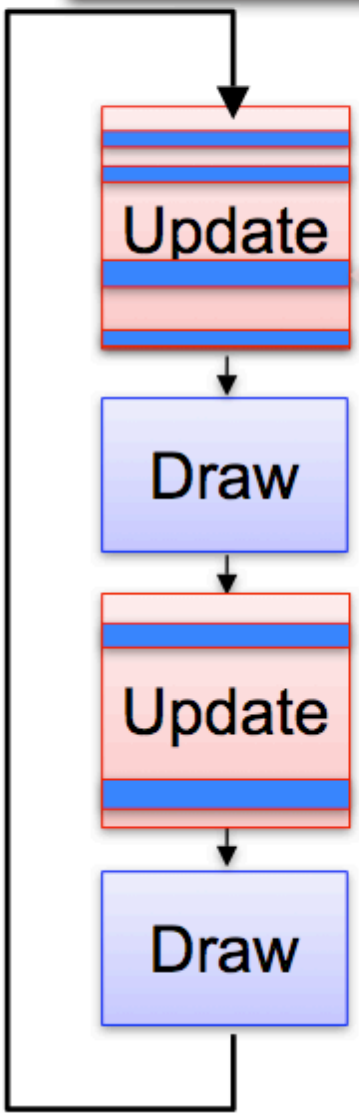


Copy object to main memory before updating it

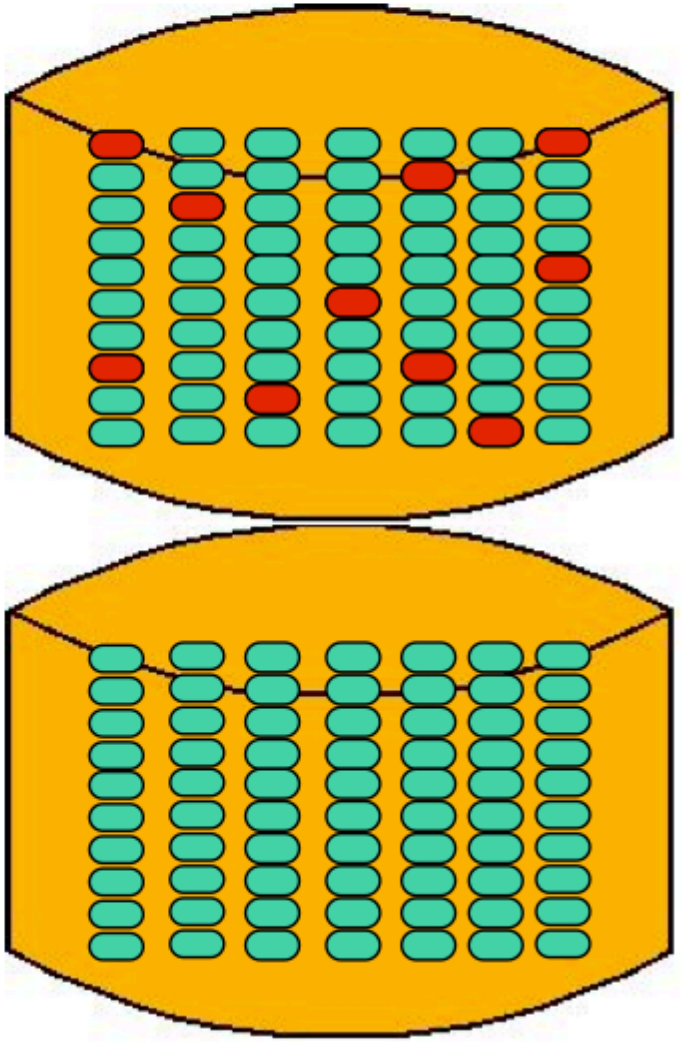


Copy-On-Update

All Objects	Dirty Objects
Log	Double Backup
Eager Copy	COU

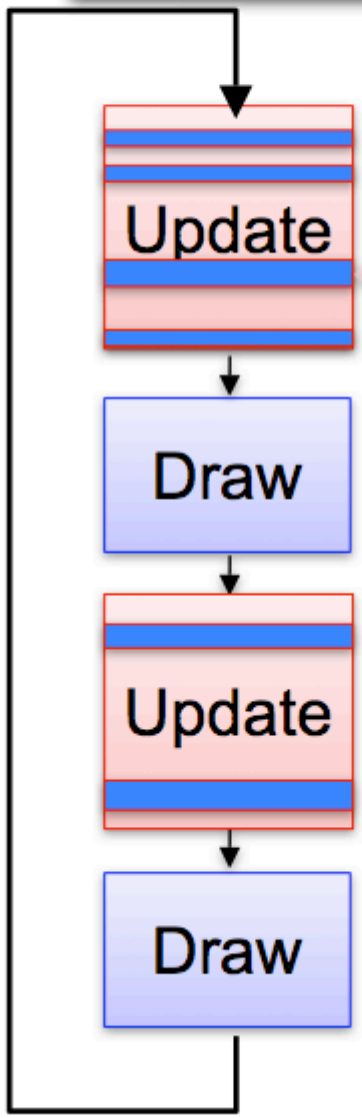


Copy object to main memory before updating it

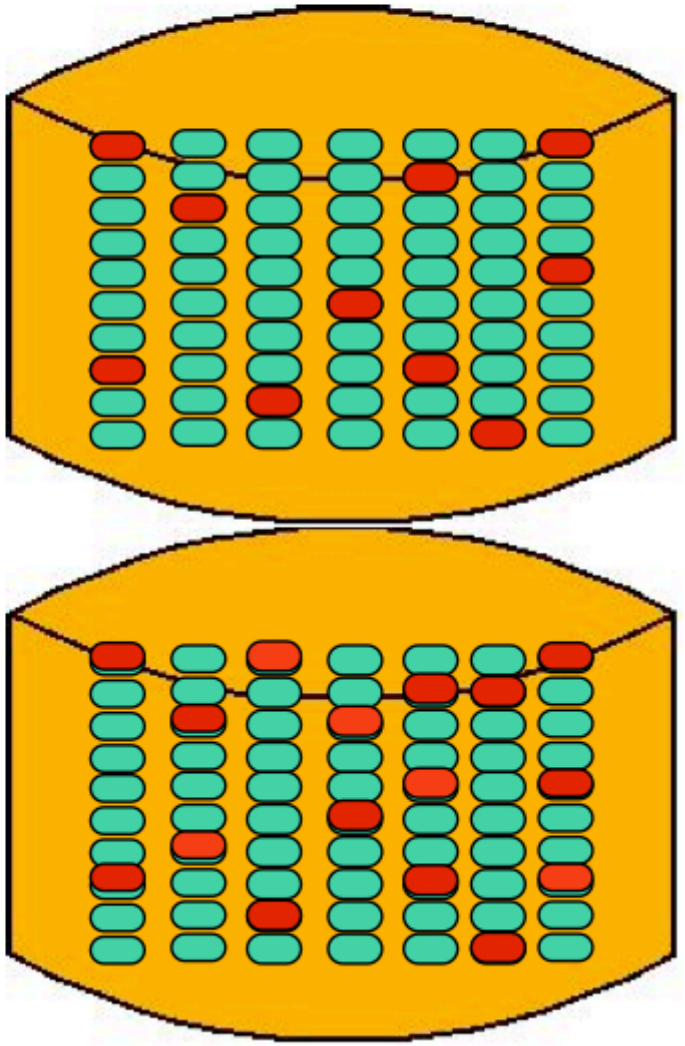


Copy-On-Update

All Objects	Dirty Objects
Log	Double Backup
Eager Copy	COU



Copy object to main memory before updating it



Experimental Stuff

- Simulation model
  - Ability to evaluate different types of hardware
  - Reduces effort to implement all algorithms described
  - Others can repeat results (with java file)
- Datasets
  - Zipfian distribution tracefile
  - Prototype game; updates logged to tracefile

parameter	notation	setting
Tick Frequency	F_{tick}	30 Hz
Atomic Object Size	S_{obj}	512 bytes
Memory Bandwidth	B_{mem}	2.2 GB/s
Memory Latency	O_{mem}	100 ns
Lock overhead	O_{lock}	145 ns
Bit test/set overhead	O_{bit}	2 ns
Disk Bandwidth	B_{disk}	60 MB/s

Table 3: Parameters for cost estimation

Performance Model Components

Assume $k \leq n$ where n is the number of atomic game state objects

- In main-memory copy time

$$\Delta T_{memcopy}(k) = \frac{k \cdot S_{obj}}{B_{mem}}$$

- Disk write time

$$\Delta T_{disk\text{-}write}(k) = \frac{k \cdot S_{obj}}{B_{disk}}$$

- Update handler overhead

$$\Delta T_{overhead} = O_{bit} + O_{lock} + \Delta T_{memcopy}$$

- Recovery Time

$$\Delta T_{recovery} = \Delta T_{restore} + \Delta T_{replay}$$

$$\Delta T_{restore} = \frac{(k \cdot C + n) \cdot S_{obj}}{B_{disk}}$$

(Partial-Redos)

$$\Delta T_{restore} = \frac{n \cdot S_{obj}}{B_{disk}}$$

(The other algorithms)

Update Handler Overhead

$$\Delta T_{overhead} = O_{bit} + O_{lock} + \Delta T_{memcopy}$$

O_{bit} Overhead to test a dirty bit

O_{lock} The cost to lock an object

$\Delta T_{memcopy}$ Cost of a memory copy of an atomic object

$$\Delta T_{memcopy} = \frac{S_{obj}}{B_{mem}}$$

S_{obj} Size of an object

B_{mem} Memory bandwidth

Requirements

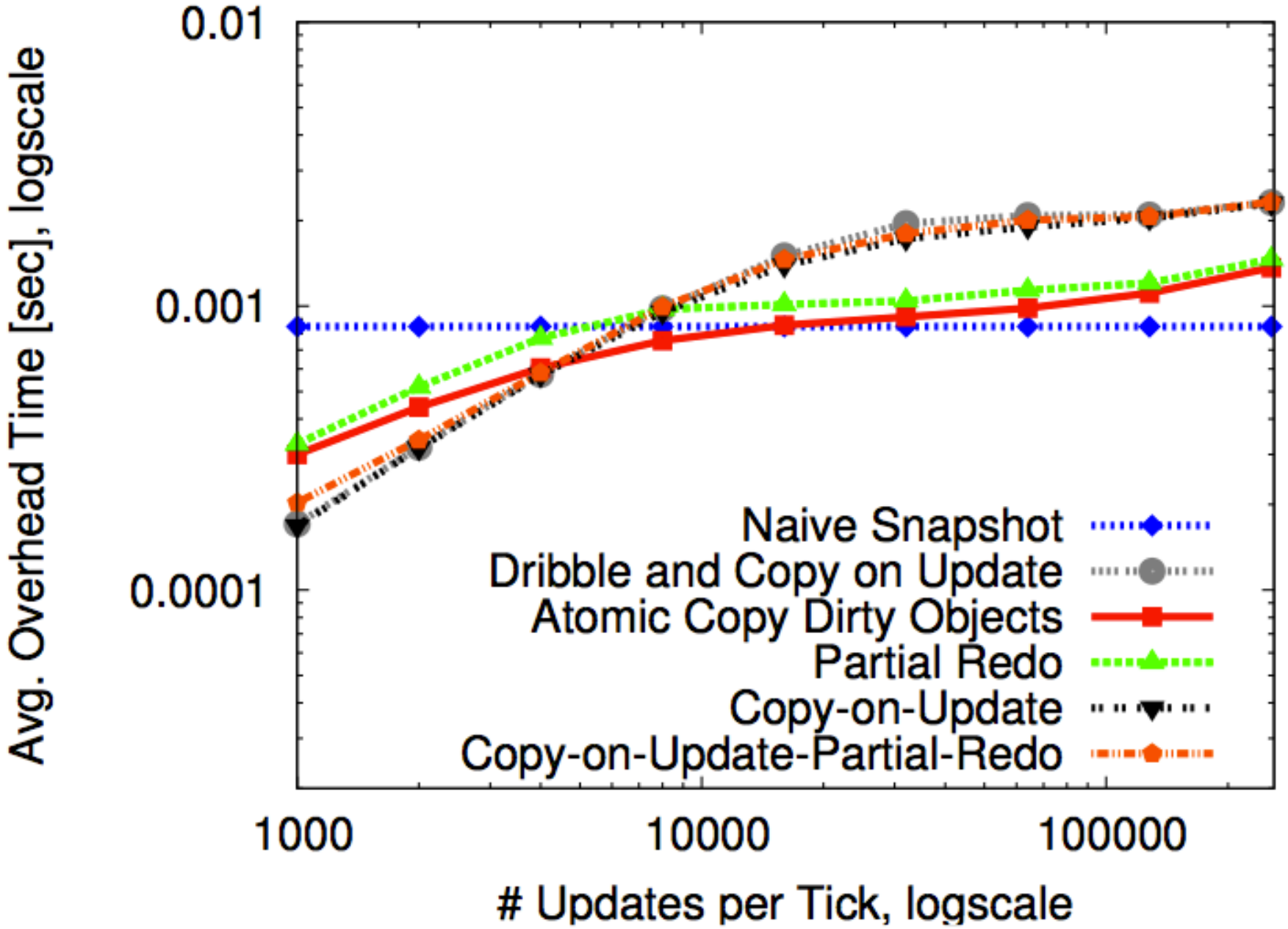
- Small overhead
  - ✦ Entire checkpointing process must fit into the game simulation
- Uniform overhead
  - ✦ Low latency, no hiccup in the game
- No data loss
  - ✦ Recover to the point of the crash

Data Set 1: Synthetic Trace

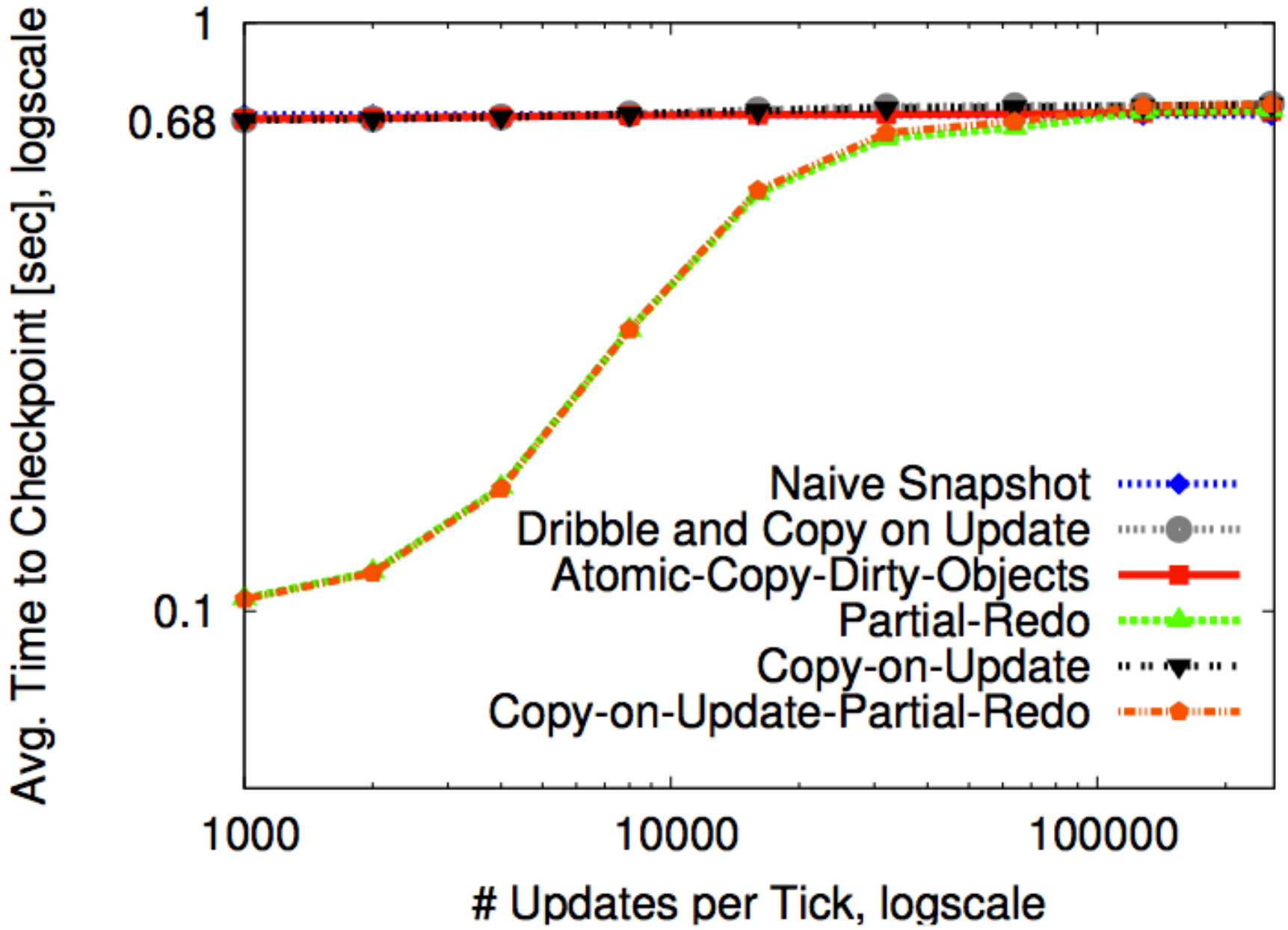
- Generate updates according to a Zipf distribution
- Vary the number of updates per tick from 1k to 256k updates per tick

Zipfian Distribution

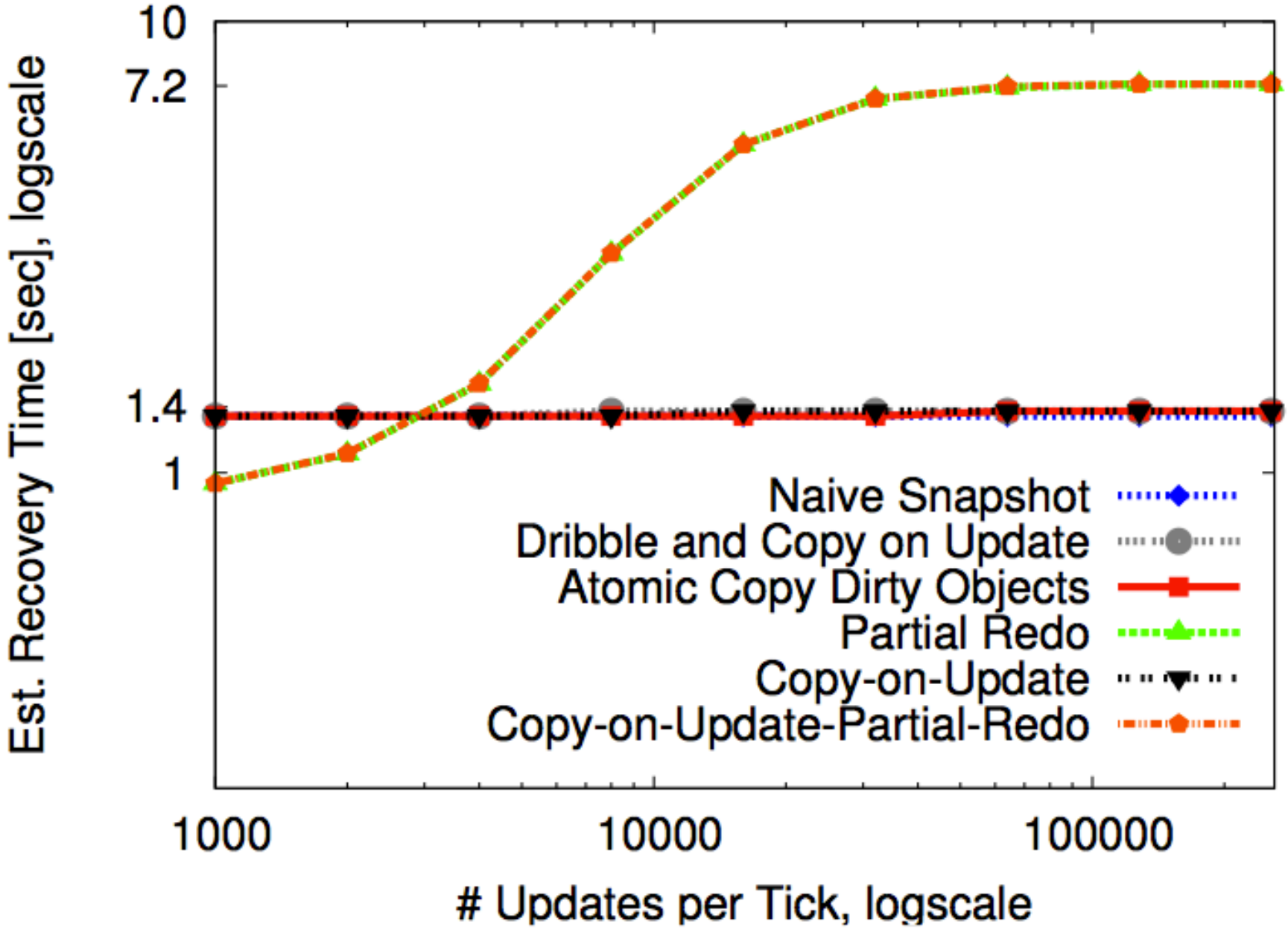
- Linguist *Kingsley Zipf*
- *Given some corpus of natural language utterances, the frequency of any word is inversely proportional to its rank in the frequency table. (wikipedia.org)*



(a) Updates per tick vs. overhead time



(b) Updates per tick vs. time to checkpoint



(c) Updates per tick vs. recovery time

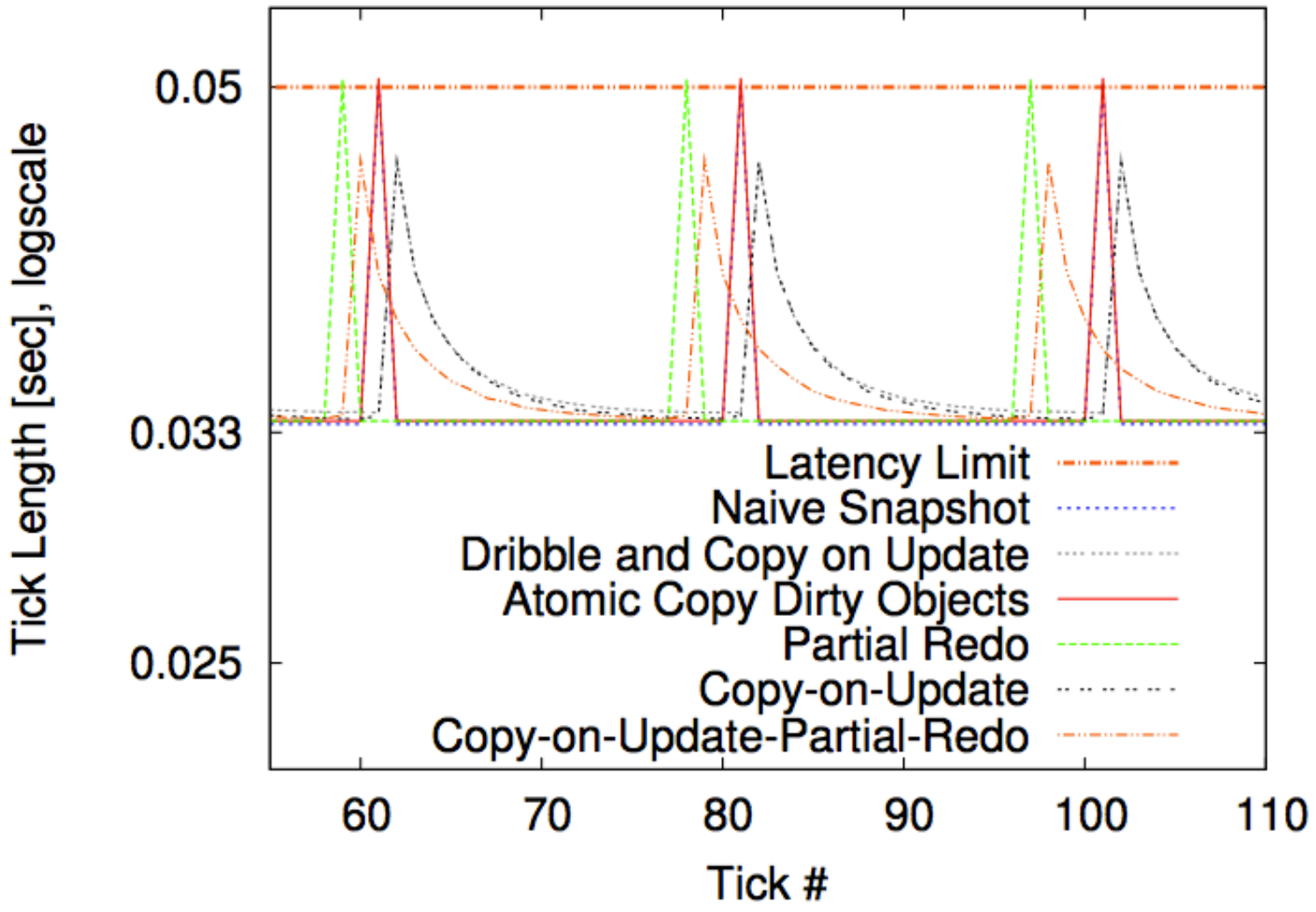
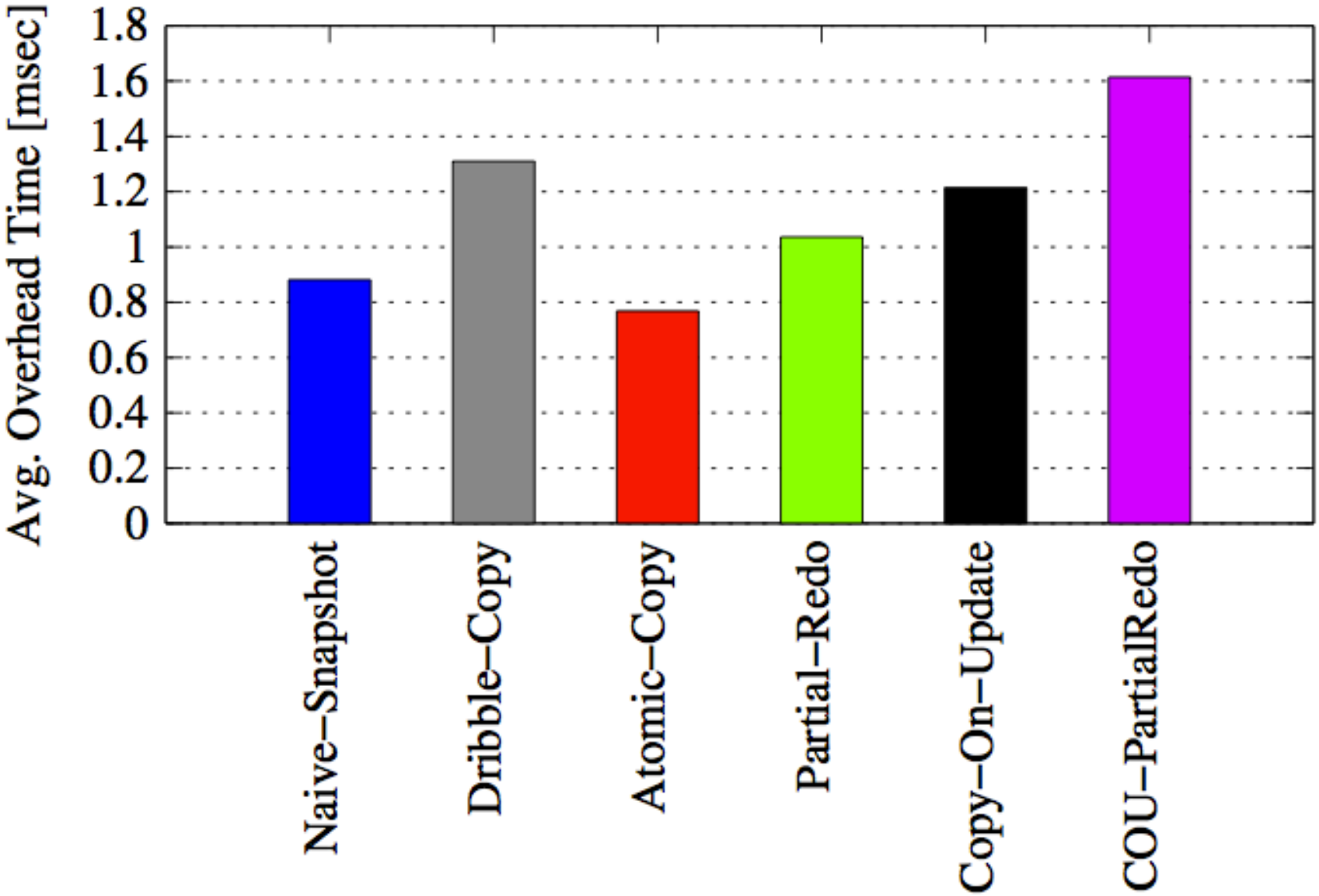


Figure 3: Latency analysis: 10M objects, 64K updates per tick.

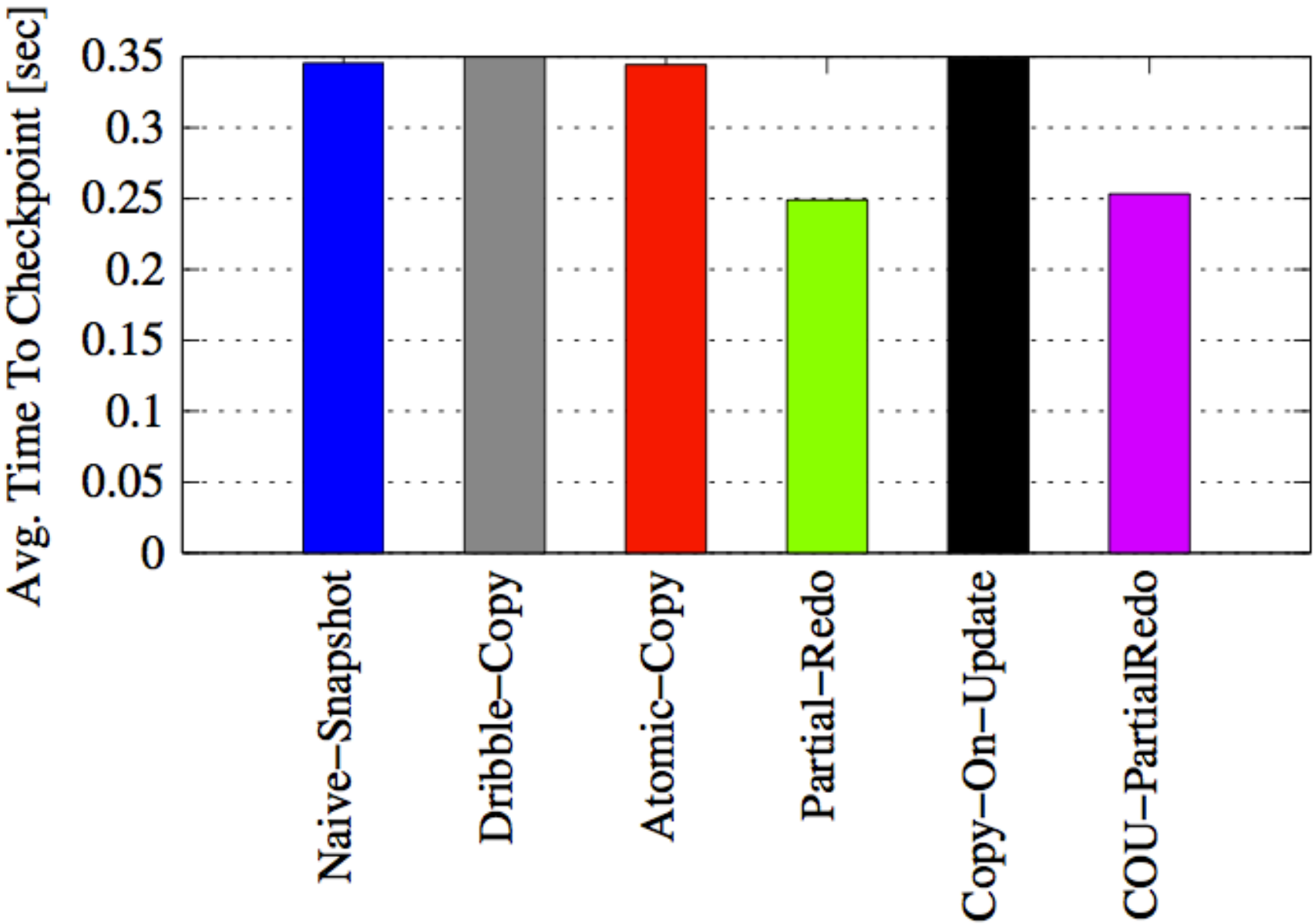
Data Set 2: Prototype Game Trace

- We simulated a medieval battle of the type common in many MMOs
- Knights, archers and healers, divided into two teams
- The objective is to defeat as many enemies as possible

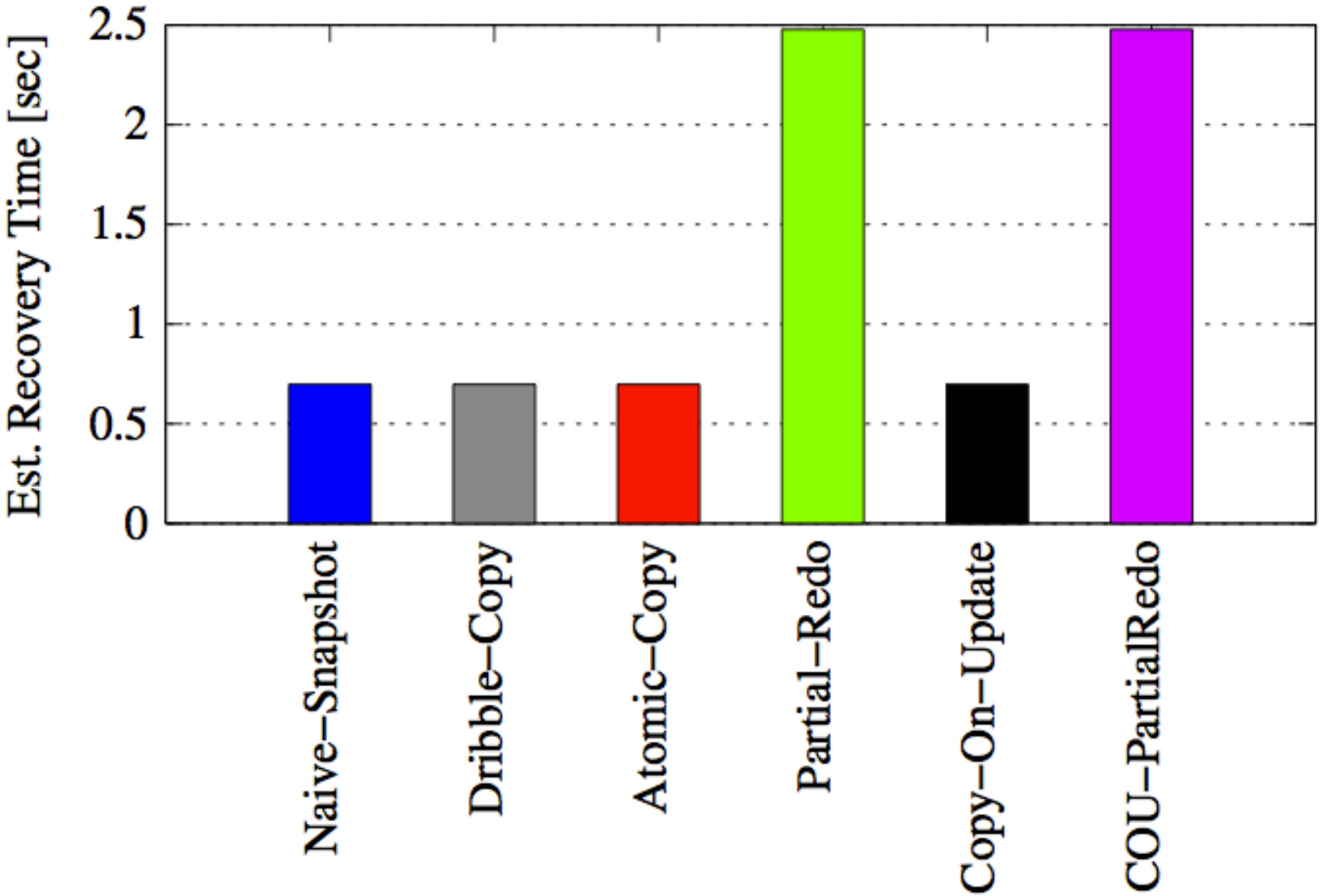
Refer to http://www.cs.cornell.edu/~wmwhite/papers/2007-SIGMOD-Games.pdf



(a) Overhead time

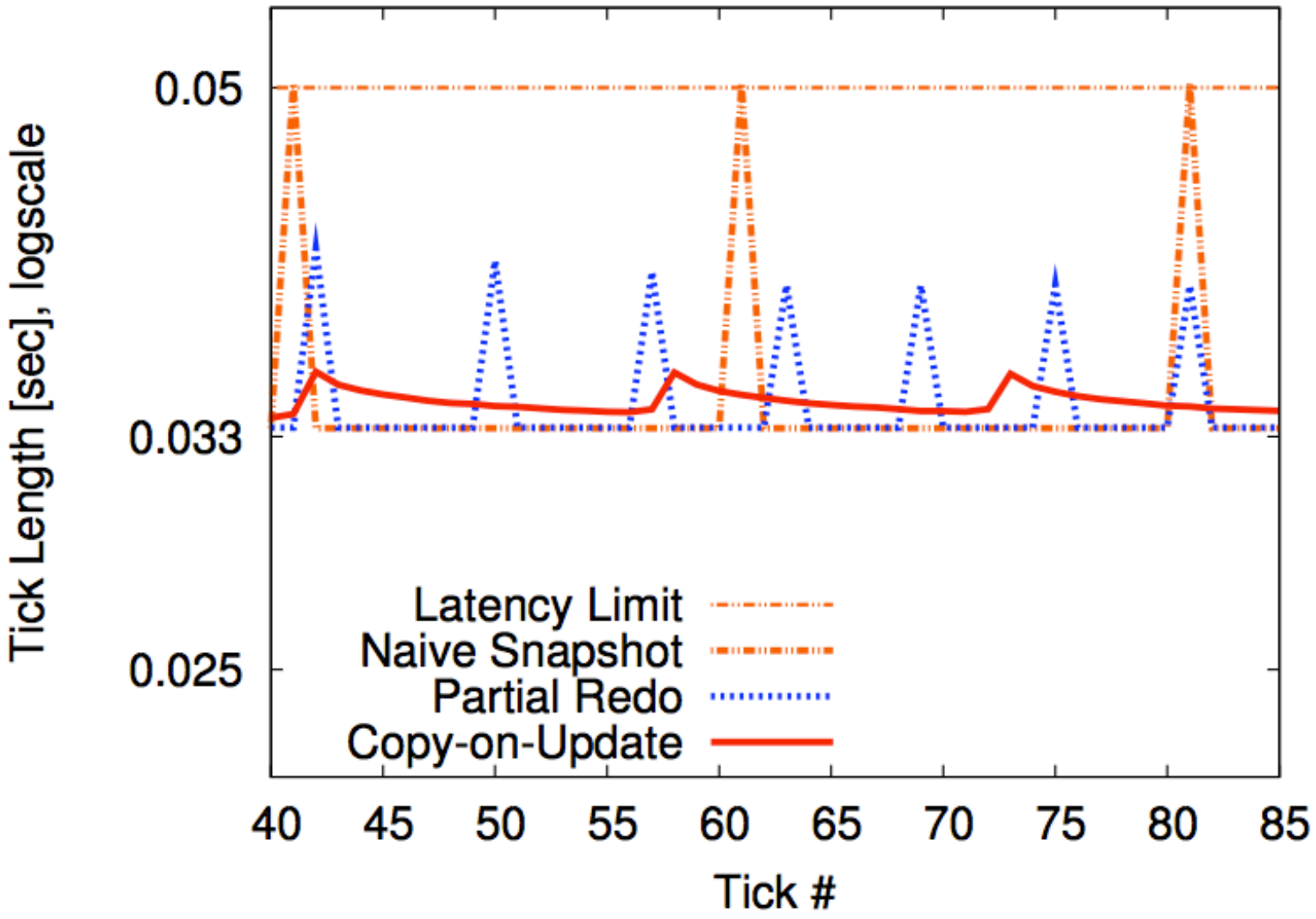


(b) Time to checkpoint

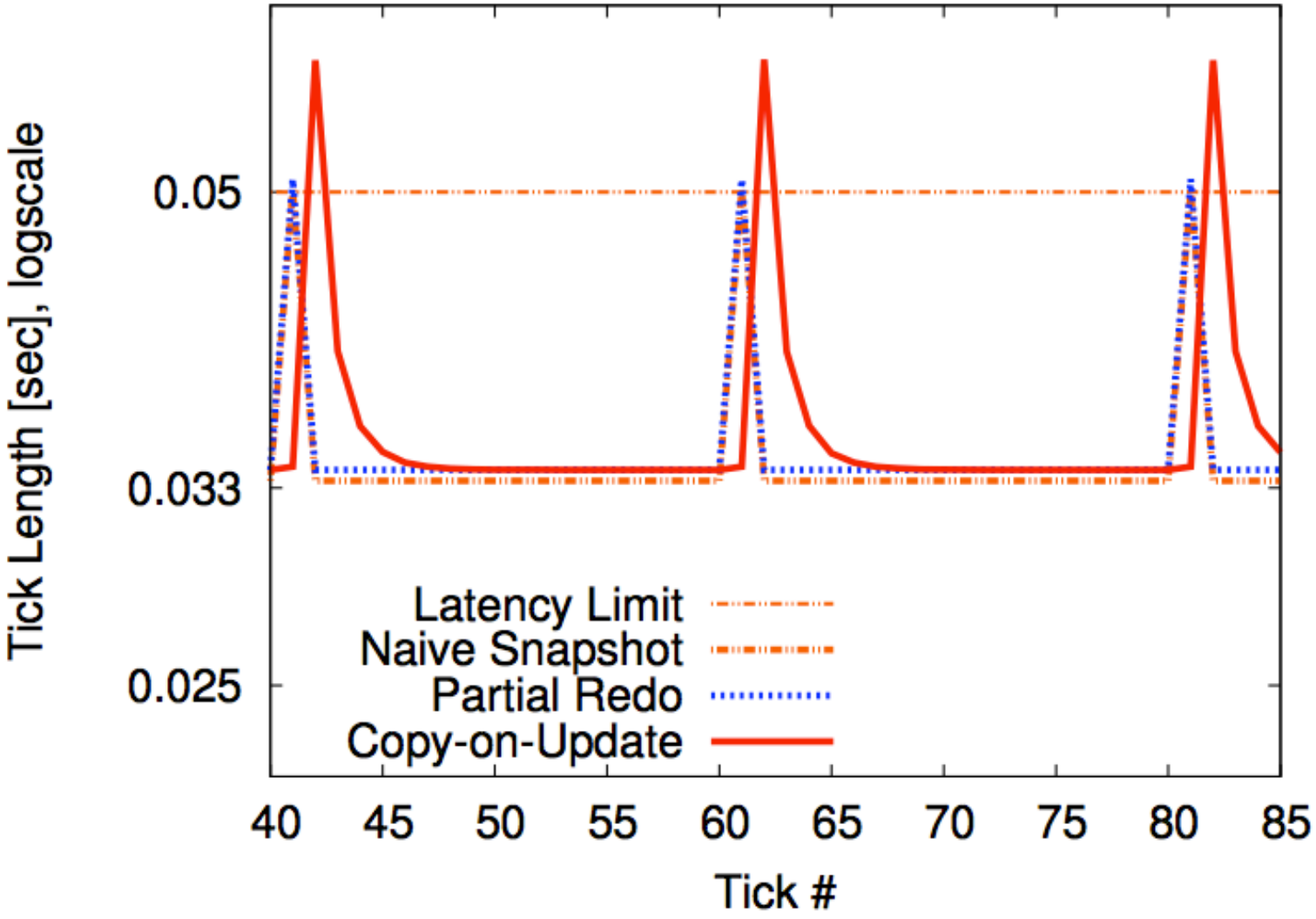


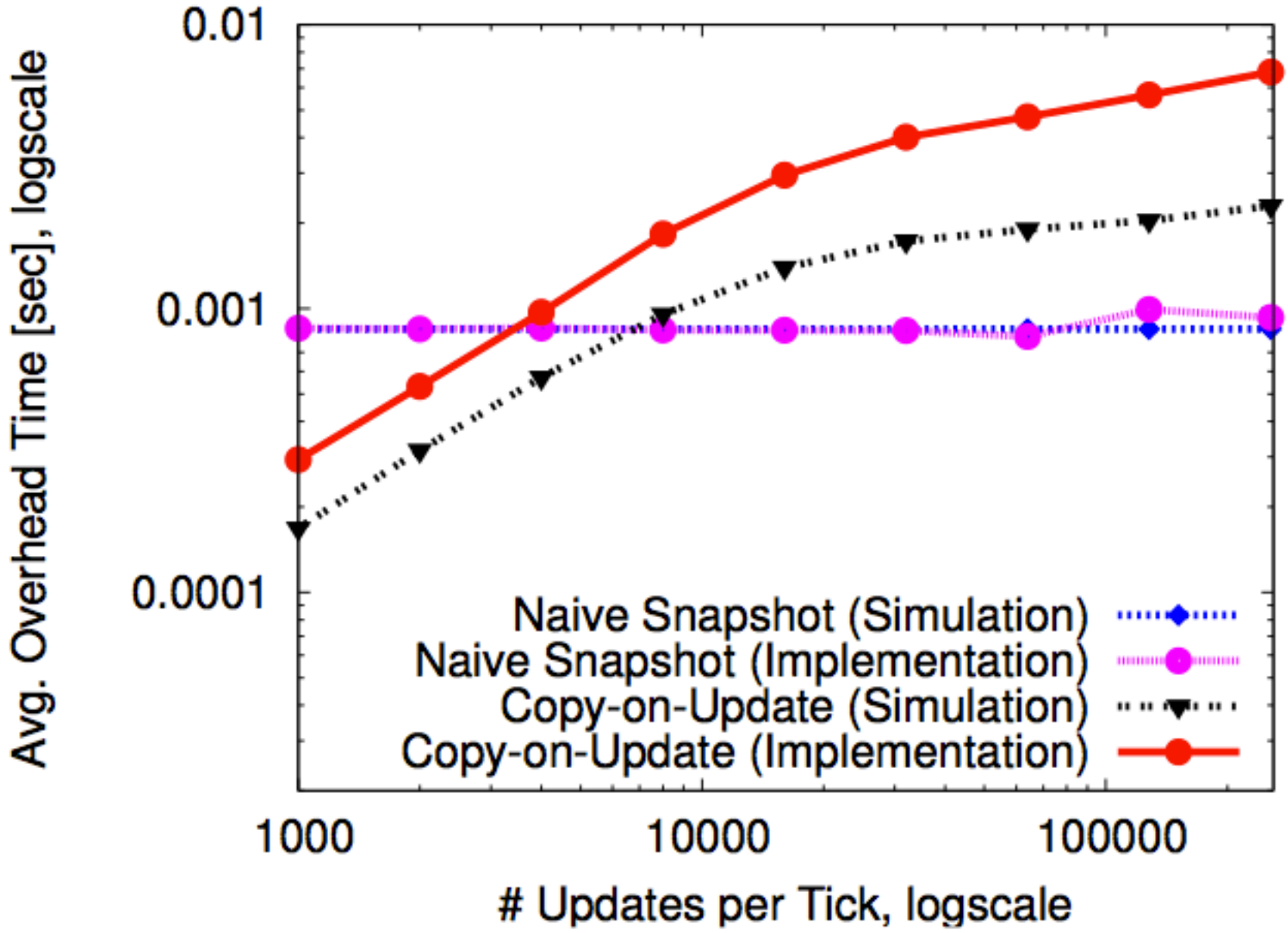
(c) Recovery time

Latency Analysis
(8k updates per tick)

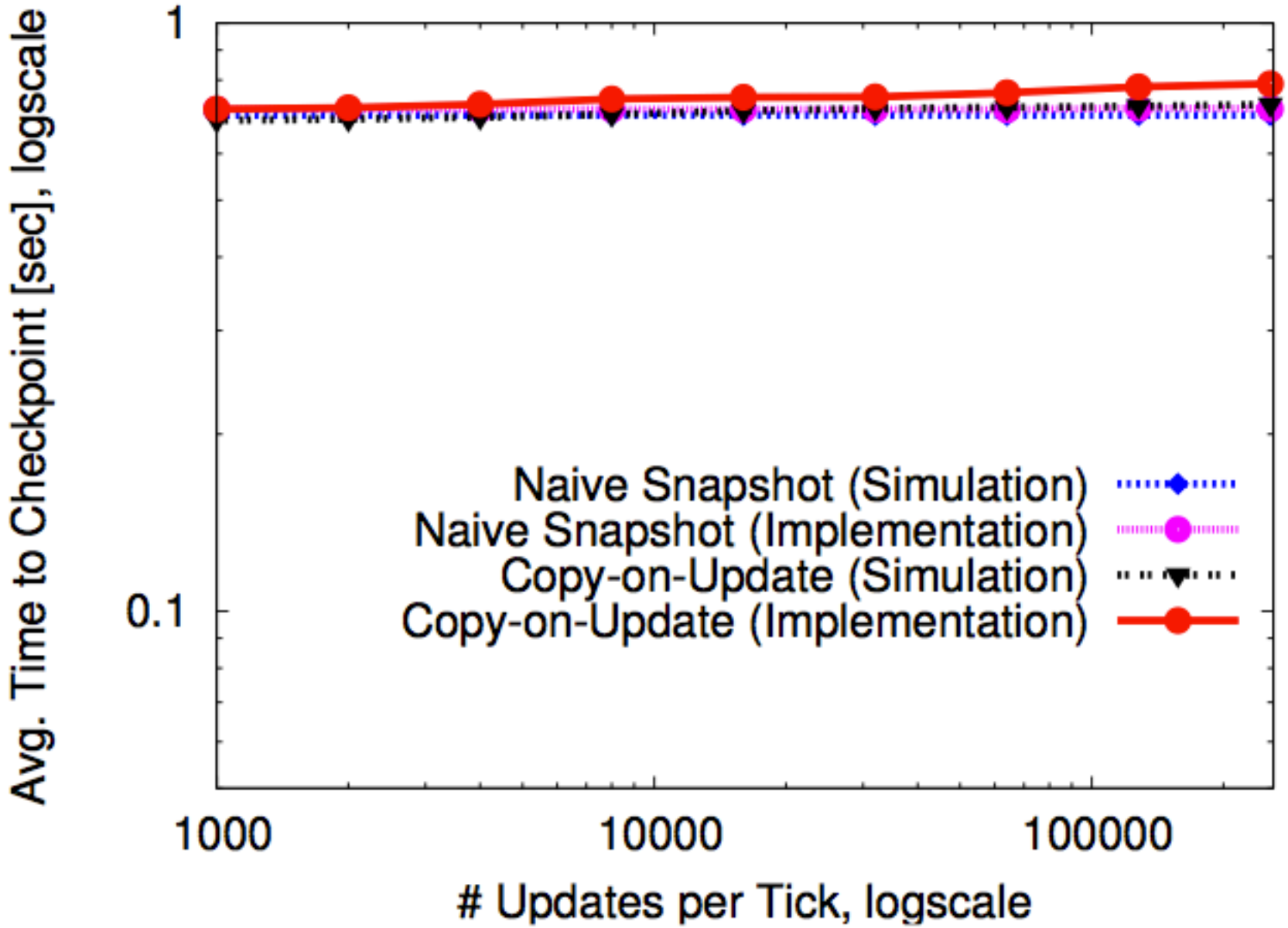


Latency Analysis
(256k updates per tick)

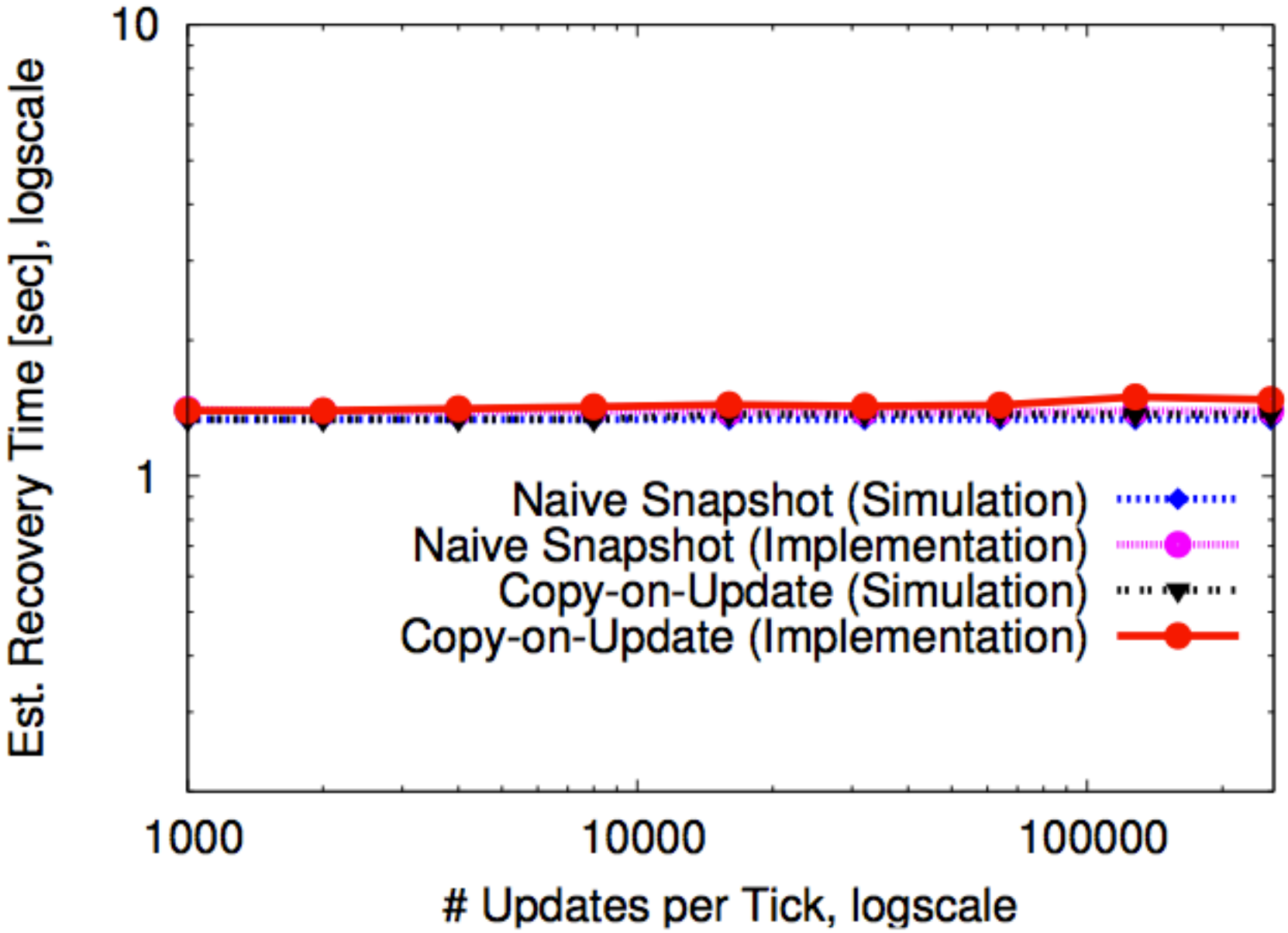




(a) Updates per tick vs. overhead time



(b) Updates per tick vs. time to checkpoint



(c) Updates per tick vs. recovery time

Questions?

Why did the recovery time for COU-partial redo & partial-redo suck so bad in comparison to Naïve Snapshot & Dribble on update? All of these methods used log storage.

Why did Copy-on-Update have lower overhead than COU-Partial Redo?