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

- 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

- Receive player input
- Process player actions
- Handle updates
- Perform checkpointing

30 ticks/sec

Draw

≈ 33ms

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

<table>
<thead>
<tr>
<th></th>
<th>All Objects</th>
<th>Dirty Objects</th>
<th>Eager Copy</th>
<th>Copy On Update</th>
<th>Double Backup</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive-Snapshot</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dribble-And-Copy-On-Update</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Atomic-Copy-Dirty-Objects</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partial-Redo</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Copy-On-Update</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Copy-On-Update-Partial-Redo</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
## Checkpointing Algorithms

<table>
<thead>
<tr>
<th></th>
<th>All Objects</th>
<th>Dirty Objects</th>
<th>Eager Copy</th>
<th>Copy On Update</th>
<th>Double Backup</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive-Snapshot</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Dribble-And-Copy-On-Update</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Atomic-Copy-Dirty-Objects</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Partial-Redo</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Copy-On-Update</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Copy-On-Update-Partial-Redo</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
# Checkpointing Algorithms

<table>
<thead>
<tr>
<th>Naive-Snapshot</th>
<th>Partial-Redo</th>
<th>Copy-On-Update</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>All Objects</strong></td>
<td><strong>Dirty Objects</strong></td>
<td><strong>All Objects</strong></td>
</tr>
<tr>
<td>Log</td>
<td>Double Backup</td>
<td>Log</td>
</tr>
<tr>
<td>Eager Copy</td>
<td>COU</td>
<td>Eager Copy</td>
</tr>
<tr>
<td>Race</td>
<td>Strength</td>
<td>Agility</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>Human</td>
<td>108</td>
<td>73</td>
</tr>
<tr>
<td>Dwarf</td>
<td>110</td>
<td>69</td>
</tr>
<tr>
<td>Night Elf</td>
<td>105</td>
<td>78</td>
</tr>
<tr>
<td>Gnome</td>
<td>103</td>
<td>76</td>
</tr>
<tr>
<td>Draenei</td>
<td>109</td>
<td>70</td>
</tr>
<tr>
<td>Orc</td>
<td>111</td>
<td>70</td>
</tr>
<tr>
<td>Troll</td>
<td>109</td>
<td>75</td>
</tr>
<tr>
<td>Undead</td>
<td>107</td>
<td>71</td>
</tr>
<tr>
<td>Blood Elf</td>
<td>105</td>
<td>75</td>
</tr>
<tr>
<td>Tauren</td>
<td>113</td>
<td>68</td>
</tr>
</tbody>
</table>
Naive-Snapshot

Update

Draw

Update

Draw

<table>
<thead>
<tr>
<th>All Objects</th>
<th>Dirty Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log</td>
<td>Double Backup</td>
</tr>
<tr>
<td>Eager Copy</td>
<td>COU</td>
</tr>
</tbody>
</table>

Race | Strength | Agility | Stamina | Intellect | Spirit | Armor | Health |
-----|----------|---------|---------|-----------|--------|-------|--------|
Human|          |         |         |           |        |       |        |
Dwarf|          |         |         |           |        |       |        |
Night Elf|      |         |         |           |        |       |        |
Gnome|          |         |         |           |        |       |        |
Draenei|         |         |         |           |        |       |        |
Orc|          |         |         |           |        |       |        |
Troll|          |         |         |           |        |       |        |
Undead|          |         |         |           |        |       |        |
Blood Elf|     |         |         |           |        |       |        |
Tauren|          |         |         |           |        |       |        |
Partial-Redo

<table>
<thead>
<tr>
<th>All Objects</th>
<th>Dirty Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log</td>
<td>Double Backup</td>
</tr>
<tr>
<td>Eager Copy</td>
<td>COU</td>
</tr>
</tbody>
</table>

Flowchart:
1. Update
2. Draw
3. Update
4. Draw

Table:
<table>
<thead>
<tr>
<th>Race</th>
<th>Strength</th>
<th>Agility</th>
<th>Stamina</th>
<th>Intellect</th>
<th>Spirit</th>
<th>Armor</th>
<th>Health</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dwarf</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Night Elf</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gnome</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dreniel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Orc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Troll</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Undead</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blood Elf</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tauren</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Copy-On-Update

Copy object to main memory before updating it
Copy-On-Update

Copy object to main memory before updating it

<table>
<thead>
<tr>
<th>All Objects</th>
<th>Dirty Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log</td>
<td>Double Backup</td>
</tr>
<tr>
<td>Eager Copy</td>
<td>COU</td>
</tr>
</tbody>
</table>
Copy-On-Update

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

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**
  \[ \Delta T_{memcpy}(k) = \frac{k \cdot S_{obj}}{B_{mem}} \]

- **Disk write time**
  \[ \Delta T_{disk-write}(k) = \frac{k \cdot S_{obj}}{B_{disk}} \]

- **Update handler overhead**
  \[ \Delta T_{overhead} = O_{bit} + O_{lock} + \Delta T_{memcpy} \]

- **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

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

- $O_{\text{bit}}$: Overhead to test a dirty bit
- $O_{\text{lock}}$: The cost to lock an object
- $\Delta T_{\text{memcpy}}$: Cost of a memory copy of an atomic object
- $\Delta T_{\text{memcpy}} = \frac{S_{\text{obj}}}{B_{\text{mem}}}$
  - $S_{\text{obj}}$: Size of an object
  - $B_{\text{mem}}$: 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

(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?