HYBRID ADAPTIVE CHECKPOINTING FOR VIRTUAL MACHINE FAULT TOLERANCE

Abel Souza (Umeå University)
Alessandro Papadopoulos (Mälardalen University)
Luis Tomás (Red Hat)
David Gilbert (Red Hat)
Johan Tordsson (Umeå University)

abel.souza@umu.se
INTRODUCTION
MOTIVATION

MISSION CRITICAL
CURRENT SOLUTIONS & TRADEOFFS

<table>
<thead>
<tr>
<th></th>
<th>Type</th>
<th>Feasibility*</th>
<th>Cost*</th>
<th>Overhead* **</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i)</td>
<td>Application</td>
<td>+-</td>
<td>$$$</td>
<td>&gt;= 0</td>
</tr>
<tr>
<td>(ii)</td>
<td>Hardware</td>
<td>+</td>
<td>$$$</td>
<td>~ 0</td>
</tr>
<tr>
<td>(iii)</td>
<td>Agnostic (VMs)</td>
<td>+</td>
<td>$</td>
<td>&gt; 0</td>
</tr>
</tbody>
</table>

All solutions hide system errors from end-users, who can access main instance without interruptions even in the presence of faults.

* Educated Guess
** In Application Performance
WEB/INTERACTIVE APPLICATIONS – AGNOSTIC FAULT TOLERANT APPROACHES

Active/Passive (A/P) vs Active/Active (A/A)
TRADITIONAL CHECKPOINT MODE (A/P)
**COLO - COARSE GRAIN LOCK STEPPING*** (A/A)

RESOURCE TRADEOFFS: COLO & CHECKPOINT MODE

![Graph showing CPU load and synchronization bandwidth over time.](image)
# PROBLEM

<table>
<thead>
<tr>
<th>COLO</th>
<th>CHECKPOINT</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Lower latency</td>
<td>+ Less CPU usage</td>
</tr>
<tr>
<td>- Workload can increase</td>
<td>- Greater latency</td>
</tr>
<tr>
<td>synchronization frequency:</td>
<td>- More network usage</td>
</tr>
<tr>
<td>more <strong>network usage</strong></td>
<td></td>
</tr>
<tr>
<td>- More CPU usage</td>
<td></td>
</tr>
</tbody>
</table>

![Diagram: One Size Fits All](image_url)
IDEA: HYBRID APPROACH

• Main idea is to control how long we stay in COLO or checkpoint modes based on the frequency of checkpoints;

• This frequency indicates workload aspects to decrease VM Downtime Ratio, i.e., the overall amount of time the VM is paused due to a checkpoint;

\[
Downtime \ Ratio = \frac{\text{Avg(Downtime)}}{\text{Avg(Uptime)}}
\]
HYBRID APPROACH

[Diagram showing a hybrid approach with COLO, Run, Checkpoint, and PVM CPU and SVM CPU phases with responses and comparisons]
HYBRID THRESHOLD MODE DESIGN

If COLO is good (on average), we decrease how long we will spend in Checkpoint mode by a BETA factor (for the next switching).

If COLO is bad (on average), we increase how long we will spend in checkpoint mode for the next switching by an ALPHA factor (for the next switching).
The PI-controller accepts a user-defined set-point ($sp$, system’s aim) and $cps$ (checkpoints per second) as inputs, both used to calculate $\rho$, the controller’s mode switcher.
IMPLEMENTATION W/ OPENSTACK

PVM (Host 1)

OpenStack (nova-compute)
LibVirt
QEMU
Hybrid Mechanism

SVM (Host 2)

OpenStack (nova-compute)
LibVirt
QEMU

Network Buffering & Comparison

Network Buffering

Deploy PVM

Deploy SVM

Checkpoint Stream (RDMA)
EVALUATION

- **Three real applications** were used in order to evaluate the proposed hybrid approaches:
  - RUBiS online auction benchmark;
    - Database (I/O);
    - **Very deterministic**;
  - BugZilla Tracking System;
    - Multi-threaded Bug filing application;
    - **Non-deterministic**;
  - Video Streaming.
    - Heavily-threaded application;
    - **Highly non-deterministic**;
- All applications run 10 times, 30 minutes in each mode:
  - Checkpoint, COLO, Threshold and Controlled (Hybrid).
RESULTS - SYNTHETIC WORKLOAD

![Graph showing downtime ratio and latency over time for different strategies: Checkpoint, COLO, Threshold, and Hybrid.](image)
RESULTS – BUGZILLA WORKLOAD

- Checkpoint
- COLO
- Threshold
- Hybrid

![Graphs showing Downtime Ratio and Response Time over time and request number.](image-url)
RESULTS – VIDEO STREAMING WORKLOAD

- Checkpoint
- COLO
- Threshold
- Hybrid

Graph 1: Downtime Ratio vs. Time (s)

Graph 2: Latency (ms) vs. Frame (s)
DISCUSSION

• Average latency and throughput are where tradeoffs can be seen

• Our solution has not under-performed in any scenario, only by overheads for CPU and Network usages

• In particular, unpredictable behavior may be caused by many different factors:
  
  o Timestamp, unique-IDs per request, multi-threading...
CONCLUSIONS

• Controller follows workload behavior and decreases Downtime Ratios for each VM, without impact in application performance

• Machine/Reinforcement Learning could be used to learn workload
  
  o Potential to speed up the mode learning/convergence process;

  o Additional metrics for decision making policy (such as Performance)

• The adaptation ability is relevant in many use-cases

  o Operators offer infrastructure without asking what users will run.
HYBRID ADAPTIVE CHECKPOINTING FOR VIRTUAL MACHINE FAULT TOLERANCE

Abel Souza (Umeå University, Sweden)
Alessandro Papadopoulos (Mälardalen University, Sweden)
Luis Tomás (Red Hat, Spain)
David Gilbert (Red Hat, U.K.)
Johan Tordsson (Umeå University, Sweden)

abel.souza@umu.se

UMEÅ UNIVERSITY

The research leading to these results has received funding from the EC Seventh Framework Programme FP7/2007-2011 under grant agreement n° 609828