Nicolaas Kaashoek - Princeton University
Robert Morris - MIT

CheckSync

Making Go Applications Highly Available with Runtime Integrated Checkpoints

Motivation and Overview

- Achieving high availability is difficult

- Essential for critical components such as coordinators and lock servers

- Application agnostic solutions like VMs hurt performance

- CheckSync provides low-cost availability

- Lives in Go runtime transparent to application; suspend/checkpoint periodically

- Sends checkpoints to backup

Evaluation

Two key metrics:
1) Ease of use
2) Overhead on application performance

Comparison points:
- Remus: virtual machine live migration
- CRIU: checkpoint/restore for processes
- Application-based snapshotting

Applications used:
- MapReduce
- gonum: Go scientific computing library
- go-cache: key/value store written in Go

Ease of Use

Application	CheckSync	VM
MapReduce	5	0
gonum	0	0
go-cache	0	0

Lines of code changed per application

Key Ideas

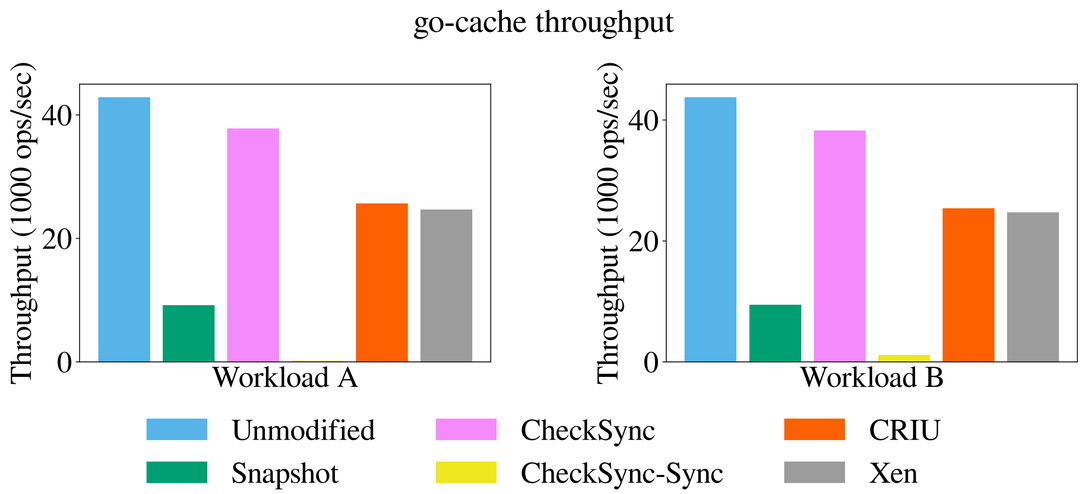
- Asynchronously checkpointing only application state more efficient than whole OS/VM

- Runtime integration makes checkpointing transparent and safe

- Simplifies system design and produces small checkpoints

- Supports multithreading using runtime thread management

- Incremental checkpointing optimization using /proc and runtime memory management reduces overhead



Results

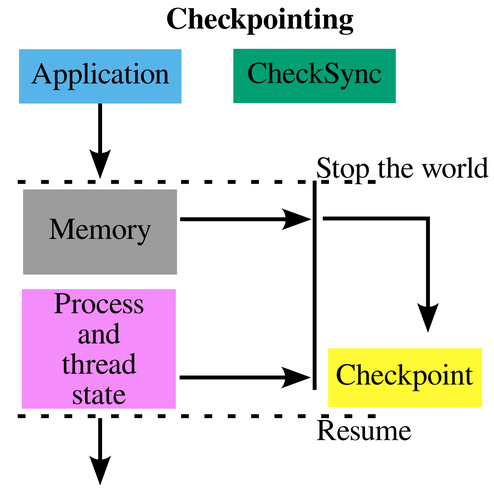
- Checkpoints smaller than existing solutions - approaches size of application-controlled snapshots

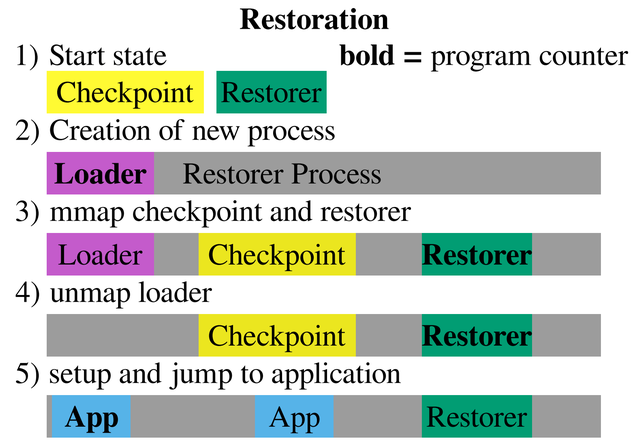
- Cost of using CheckSync lower than both CRIU and virtual machines

- Harder to use than virtual machines/CRIU, but applications can still easily adapt to it

Scan this QR code for a link to the paper and this poster

System Design





Restoration process gradually morphs restorer process into copy of application