Picocenter: Supporting long-lived, mostly-idle applications in cloud environments

Liang Zhang*  
Theophilus Benson§  
James Litton‡  
Dave Levin‡  
Frank Cangialosi‡  
Alan Mislove*  

*Northeastern University  
‡University of Maryland  
§Duke University
Open-source alternatives

- iRedMail
- modoboa
- Dropbox
- ownCloud
- Pydio
- Seafile
- GitLab
- GitBucket
- diaspora*
- Hubzilla
- flickr
- Lychee
- Mediadrop
- Google Play Music
- Plex
Service oriented applications: **long-lived**

For single user or small group: **mostly-idle**

**Long-lived, mostly-idle (LLMI) applications**

How do users run LLMI applications in cloud?
Running LLMI applications in cloud

➤ Platform as a Service (PaaS)
  ➤ Limited programming environment
  ➤ Limited network protocol support

➤ Infrastructure as a Service (IaaS)
  ➤ User manages OS and software stack
  ➤ Can be expensive to run

Can we run LLMI applications in cloud efficiently?
This talk

Goal: support LLMI applications in cloud environments

Requirements:

- Run wide variety of applications
- Run efficiently so that we can dramatically lower cost
- Be deployable in the cloud today

Picocenter

- Be able to run lots of LLMI applications in cloud
- Swap idle application to cloud storage
- Swap in application quickly when it is being requested
Related work

➤ Application running environment
  ➤ Operating system containers
  ➤ Dedicated runtime

➤ Swapping
  ➤ Pre-paging and migration
  ➤ Checkpoint and restore

➤ Picocenter
  ➤ First attempt to leverage them for LLMI apps running in the cloud

BSD Jail

Embassies

App Engine

Virtual Machine migration

Jettison

BLCR

DMTCP
Outline

➤ Introduction

➤ Design

➤ Evaluation

➤ Conclusion
Running LLMI applications in Picocenter

```json
{ init: , port: [80] }
```

PICOCENTER

{DNS: “liang-nginx.picocenter.com”}
Running LLMI applications in Picocenter

DNS: liang-nginx.picocenter.com

IP: 52.91.124.236

GET /index.html
Picocenter internals

DNS: liang-nginx.picocenter.com
IP: worker-2
GET /index.html
GET /index.html
Swapping strategies

**Full checkpoint**
- Slow start due to download of all pages
- Fast processing time because all pages are fetched
- Not all pages are necessary

**Reactive page faulting**
- Only need page meta info to start
- Slow processing due to page fetching on page faults
- Minimum pages for application processing

Can we combine the best of both strategies?
ActiveSet

- Predict pages that are needed for the request
- Reduce total download size
- Minimize round trips of page faults
- Page prediction: most recently used
- Future: prediction based on ports, ML on page faults
Implementation

➤ LLMI application runs in a process-like environment
  ➤ Use Linux container (LXC)

➤ ActiveSet
  ➤ Modified CRIU to map page to files
  ➤ Catch page faults with FUSE
Outline

➤ Introduction

➤ Design

➤ Evaluation

➤ Conclusion
Evaluation

➤ How quickly can Picocenter revive real-world processes from cloud storage?

➤ How does the ActiveSet technique help to reduce application reviving time?

➤ How does Picocenter perform with a challenging real-world application?

➤ What is the estimate cost of running applications in Picocenter

Please refer to the paper
How quickly can Picocenter revive real-world processes from cloud storage?

- Host Picocenter with ActiveSet in Amazon Virginia (VA) datacenter
  - Hot: application is alive; warm: swap in from disk; cold: swap in from cloud

- Results
  - Restore overhead: ~120 ms for warm and ~220 ms for cold
How quickly can Picocenter revive real-world processes from cloud storage?

➤ Host Picocenter with ActiveSet in Amazon Virginia (VA) datacenter

➤ **hot:** application is alive; **warm:** swap in from disk; **cold:** swap in from cloud

➤ Client requests from Virginia (VA), Oregon (OR), Frankfurt (DE) or Tokyo (JP)

➤ Results

➤ Restore overhead: ~120 ms for warm and ~220 ms for cold

➤ Overhead can be dwarfed by the end-to-end performance of the protocol itself
How does the ActiveSet technique help to reduce application reviving time?

- Control experiment on ActiveSets
  - Total memory is configured to 64 MB
  - Vary the working set size between 4 KB and 8 MB
  - Download pages in blocks; each block has 32 pages
- ActiveSet technique significantly outperforms the baseline approaches
Conclusion

➤ Picocenter: a new approach for cloud computation
  ➤ Support long-lived, mostly-idle (LLMI) applications
  ➤ Swap idle application to cloud storage
  ➤ Provide process-like environment
  ➤ Swap in real world applications in under 250 ms
  ➤ Open source: https://github.com/leoliangzhang/Picocenter

Thank you!

Questions?

liang@ccs.neu.edu