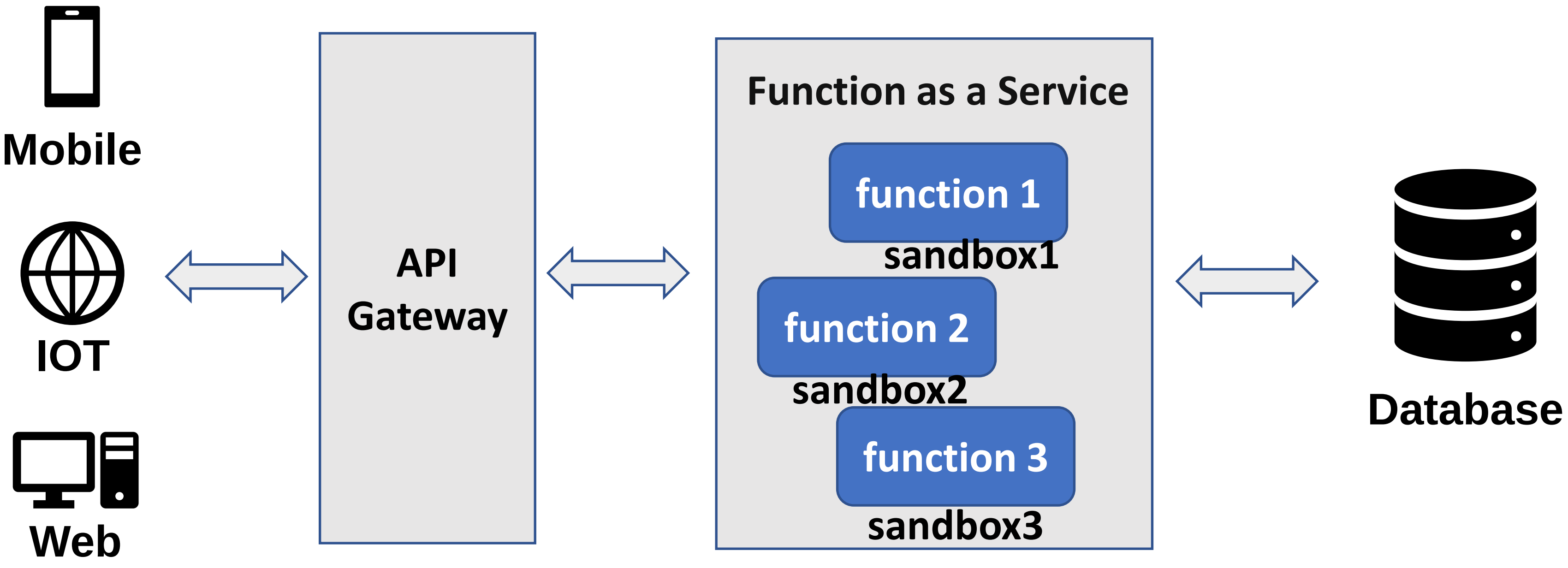
Memory Deduplication for Serverless Computing with Medes

EuroSys '22

Background : Serverless

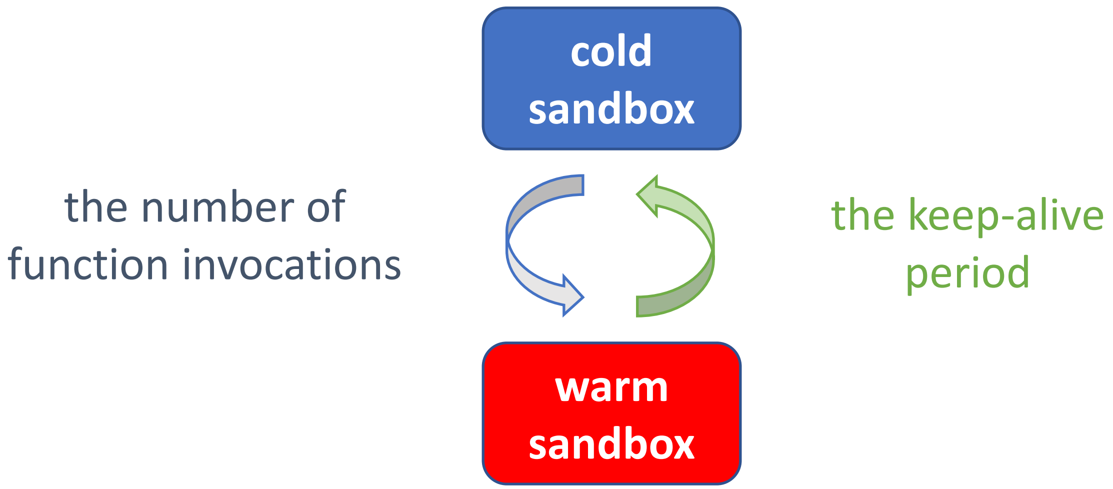
- it can be interpreted as a software system architecture method, commonly referred to as the **Serverless architecture**
- it can also represent a product form called a **Serverless product**



Serverless architecture

Background : Sandbox

- Serverless platforms manage performance and efficiency by toggling sandboxes between two states: **cold** and **warm**
- Today's platforms can only achieve **good performance** when spending **a lot of resources**



Motivation

Prior works

- eschewing fixed keep-alives in favor of adaptive keep-alive policies
- provisioning sandbox resources in anticipation of future invocations

Problem

- Sandboxes can only transition between hot and cold states, making it difficult to balance performance and efficiency

cold sandbox

- does not use any memory resources
- but induces long cold startup delays

warm sandbox

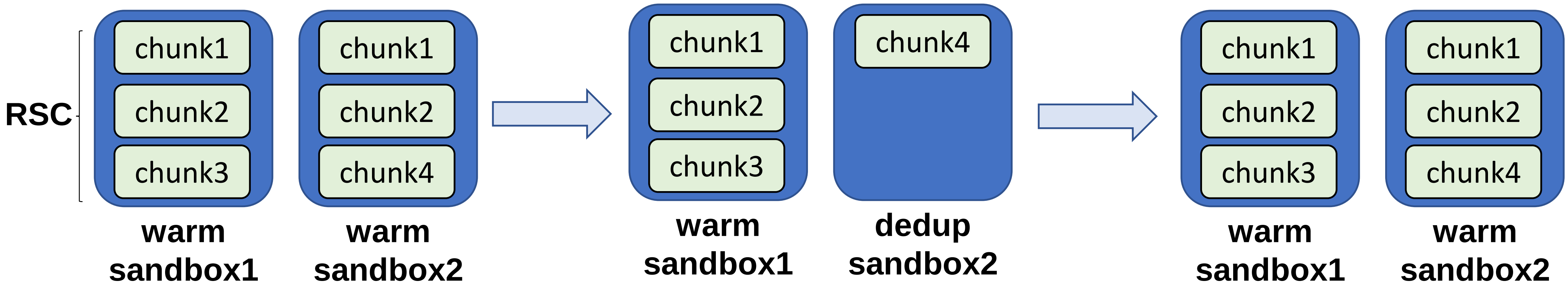
- consumes a large amount of memory
- but supports sandbox reuse

Motivation

- Improving the trade-off space by introducing **a new sandbox state - deduplicated state**

Fact :

- sandboxes of the same function can have upto 85% duplication in their memory state
- even across sandboxes of different functions, we can identify upto 80-90% duplication.



Main Idea

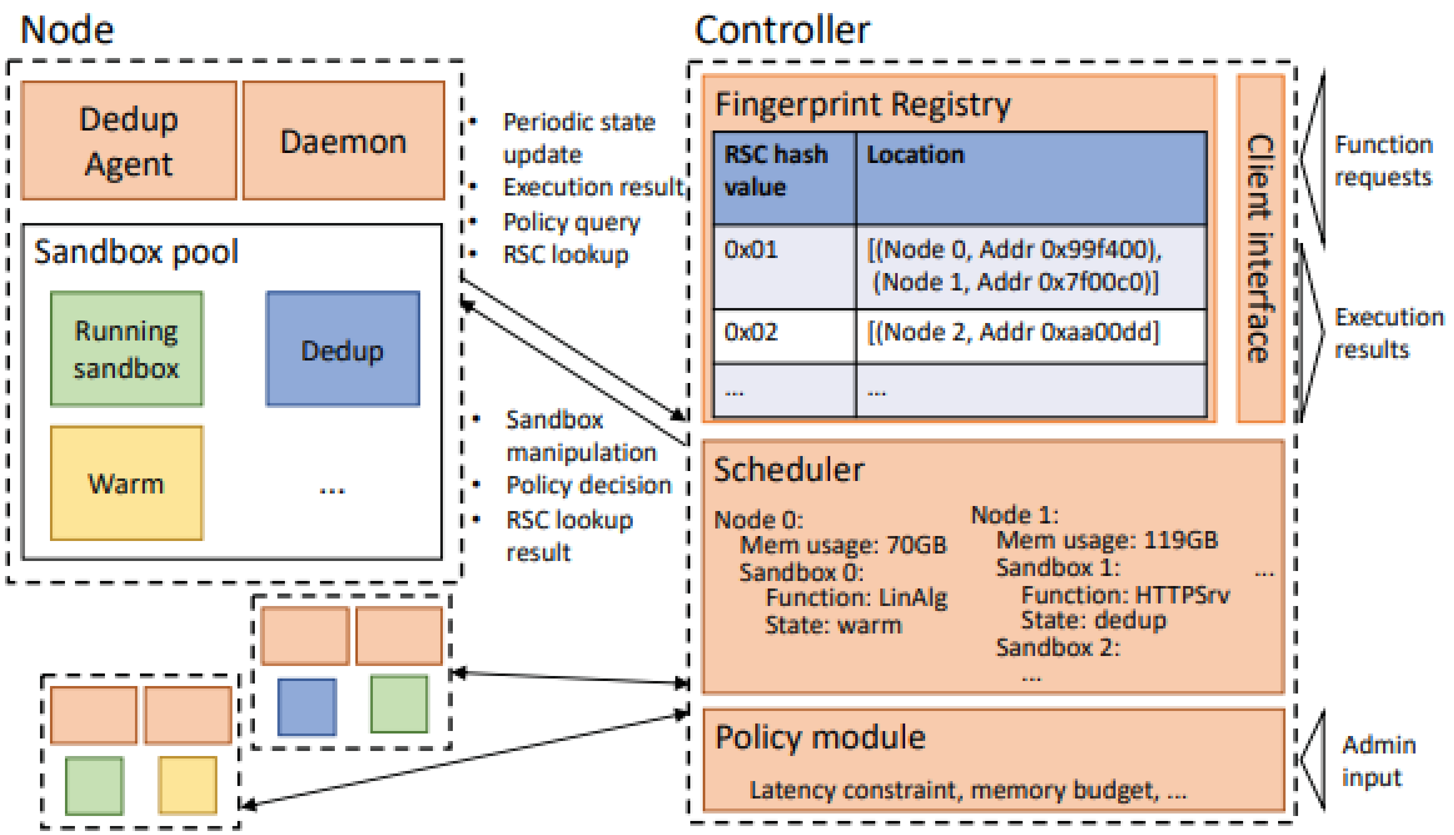
- **A novel serverless framework - Medes (Memory Deduplication for Serverless)**

  Proposed the dedup sandbox state and a novel deduplication mechanism

- **Specifically, it includes three aspects of design:**

  1. Design of Solve the Problem of of excessive memory usage
  2. Design of Fast Recovery from Dedup State
  3. Design of sandbox management strategies

Medes Architecture



- **Controller**
  1. interface to clients
  2. scheduler
  3. fingerprint registry
  4. policy module
- **Node**
  1. daemon
  2. dedup agent

Lifecycle of a sandbox

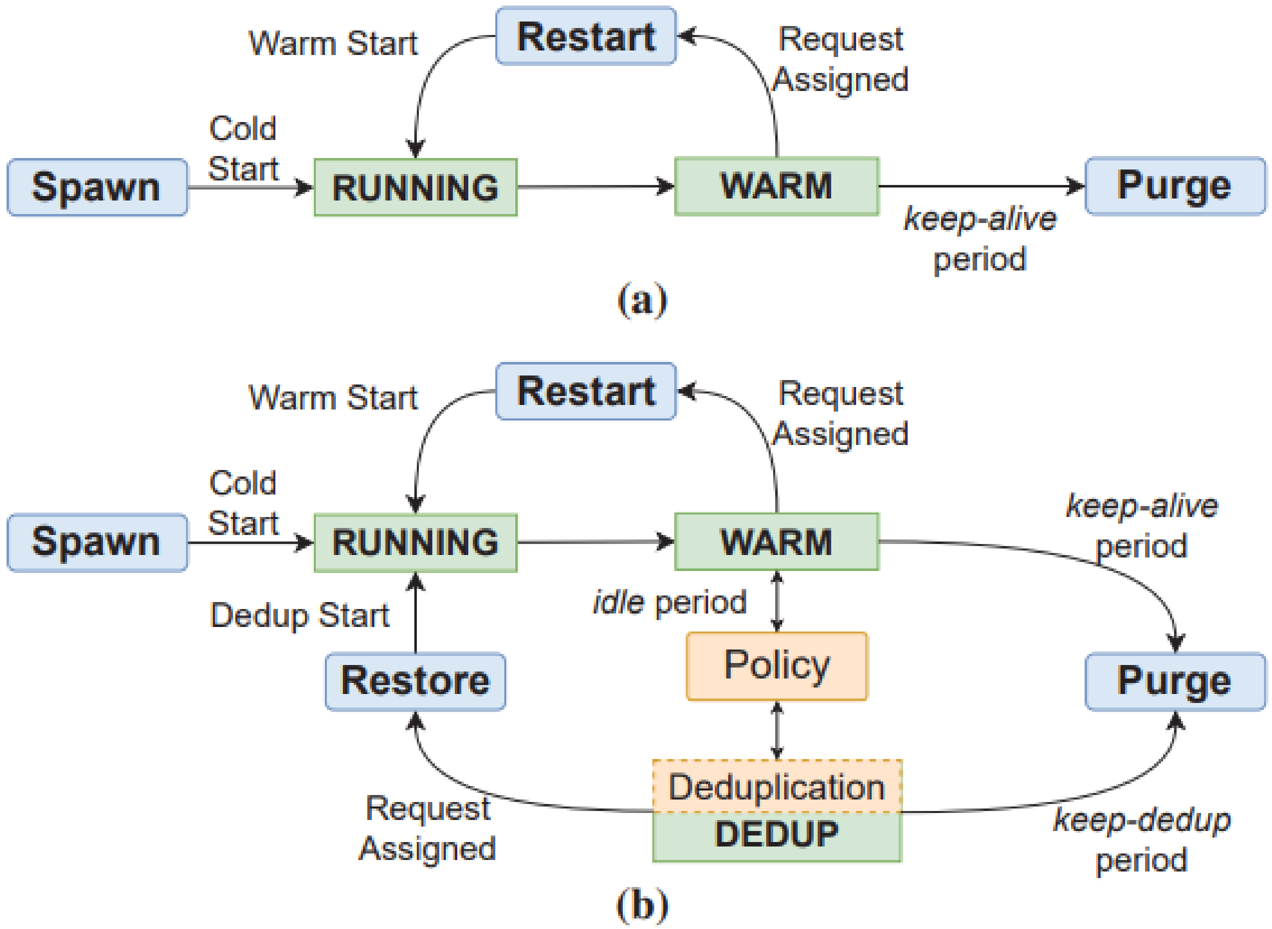
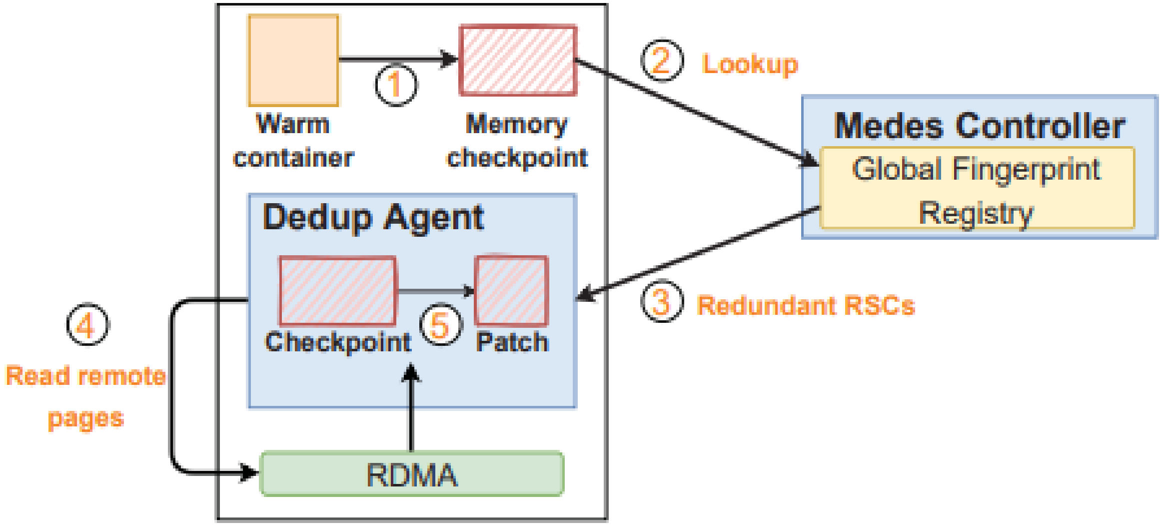


Figure 4. Lifecycle of a sandbox running on (a) Existing Platforms (b) Medes

1. **policy**
2. **idle period**
3. **keep-dedup period**

Medes Dedup Operations



- To extract the complete benefits offered via the dedup sandbox state, the dedup and restoration operations need to be scalable and fast

Design1

- **Design of Solve the Problem of of excessive memory usage**
  1. performing deduplication at the page granularity
  2. demarcating certain sandboxes as base sandboxes
  3. using value-sampled fingerprints

Design1

- **performing deduplication at the page granularity and using value-sampled fingerprints**

1.Store metadata for **each small chunk(64B)**.

① sandbox memory states : 100MB
② corresponds to nearly ~25K pages,
③ imply metadata for nearly 1.6M chunks for just 1 sandbox

2.**Page Fingerprints** and **Base Page**

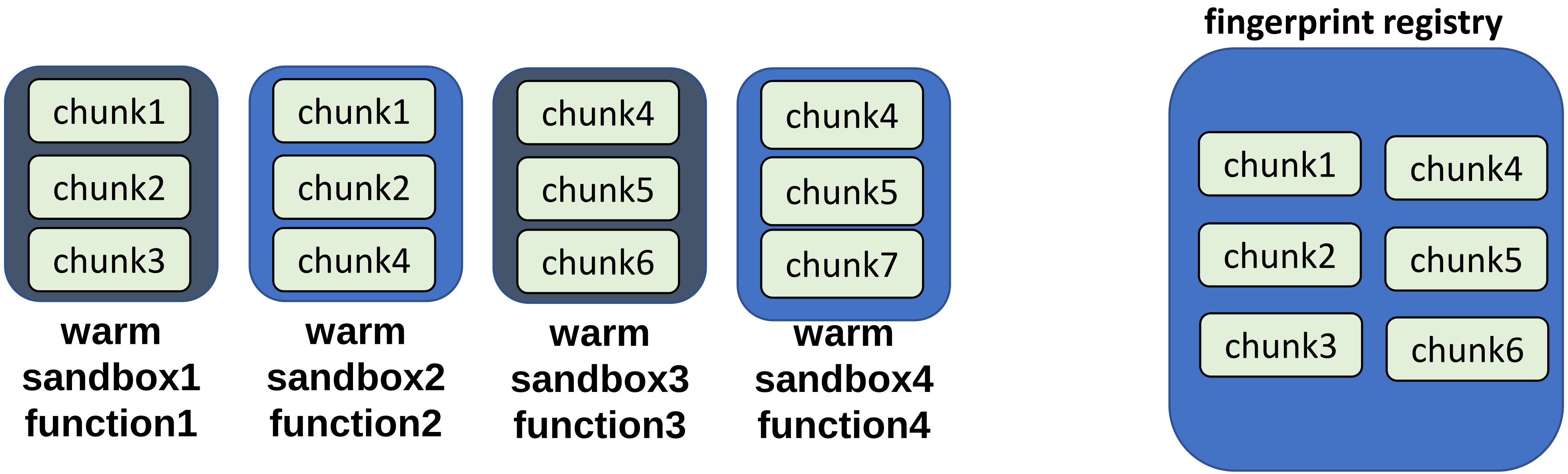
① a small subset of memory chunks, **value sampled** based on the last two bytes of the chunk
② This unordered set of **five chunk** hashes then acts as a **fingerprint of the page**
③ The candidate with the **maximum number of duplicate chunks** amongst the sampled chunks is chosen to be the base page

Design1

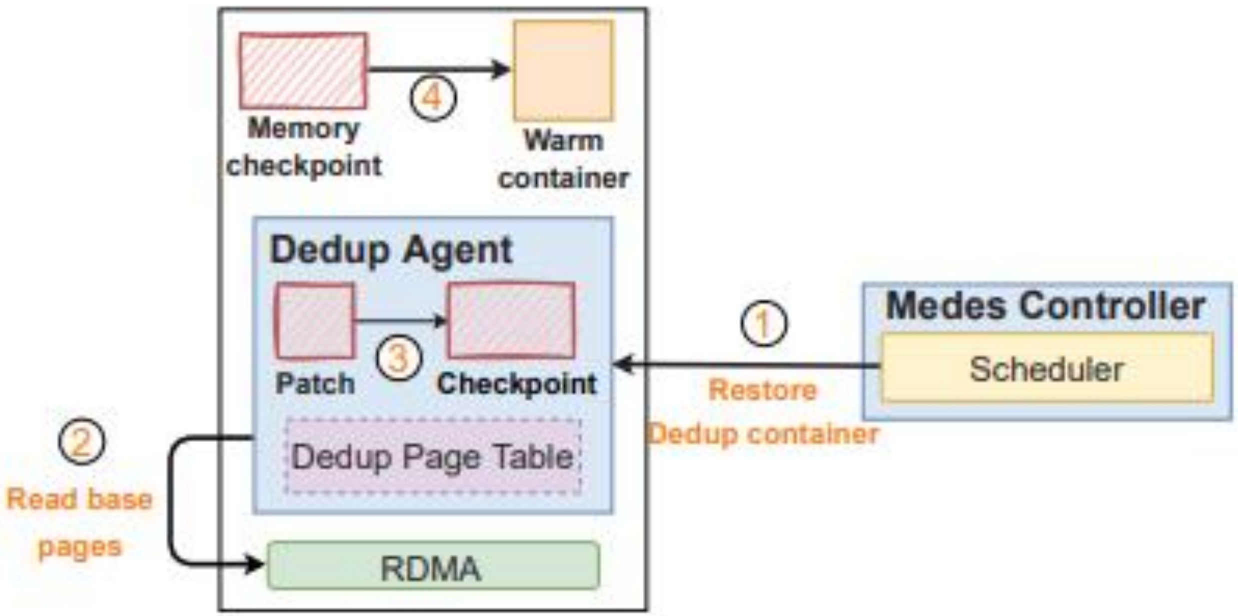
- **demarcating certain sandboxes as base sandboxes**

After sampling ,we still have nearly ~**100K** chunks to be stored for each sandbox.

① demarcate specific warm sandboxes as 'base sandboxes'.
② Only the unique memory chunks of these base sandboxes get inserted into the registry
③ D/B > T , D is the number of dedup sandboxes for a function, and B is the number of base sandboxes



Medes Restore Operations



Design2

- **Design of Fast Recovery from Dedup State**
    1. Accelerate dedup start using methods such as sandbox namespace creation and process tree reconstruction
    2. Save the container memory checkpoints in-memory to ensure fast restores rather than restoring from disk
    3. Using RDMA read operations to directly extract basic pages from the memory of remote machines, avoiding the use of remote CPUs for communication and generating low latency

Design3

- **Design of sandbox management strategies**

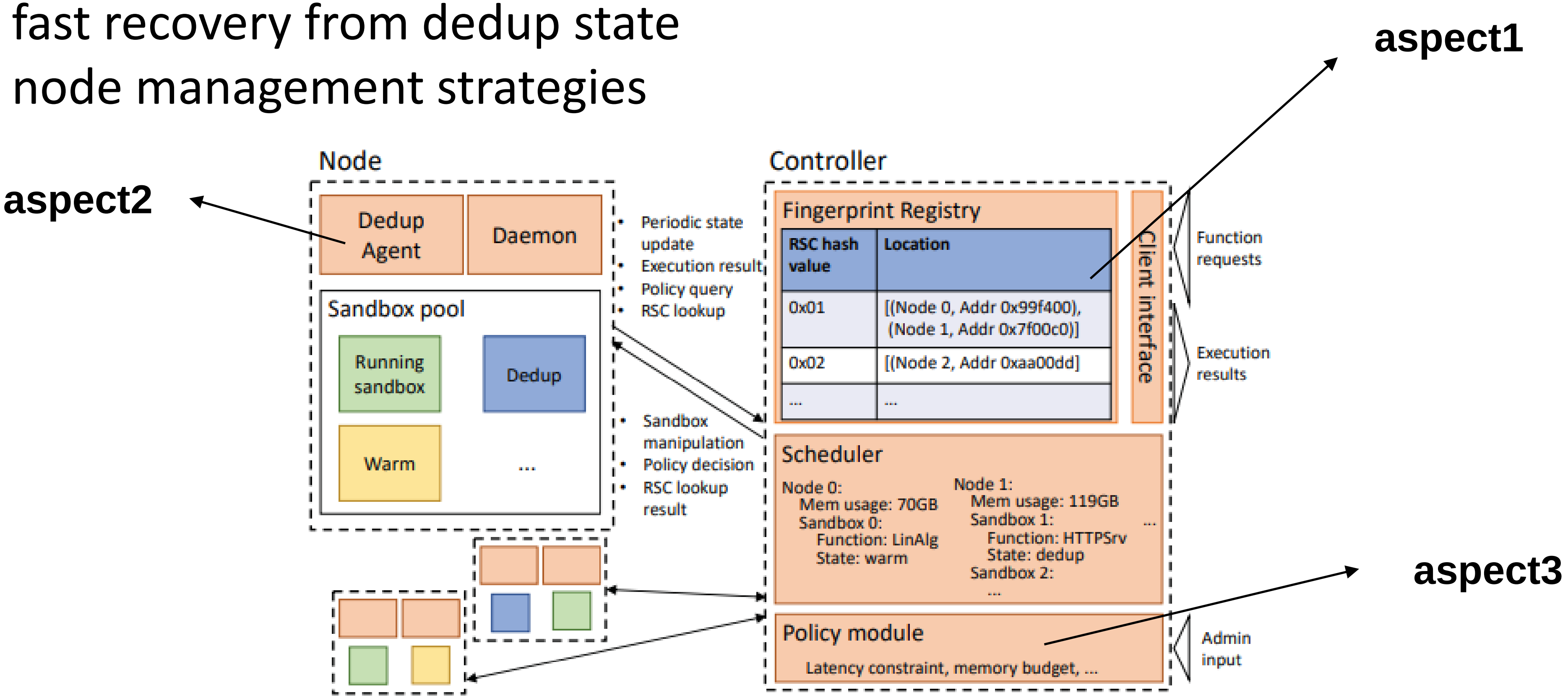
  the policy must make decisions based on:

  1. request arrival rates for the function,
  2. cluster memory pressure
  3. memory savings because of deduplication
  4. overheads of restoring deduplicated sandboxes.

Brief Summary

- The article focuses on the **dedup state** and designs solutions to address issues in the process of **deduplication and recovery** from three aspects:

① reducing metadata storage consumption
② fast recovery from dedup state
③ node management strategies



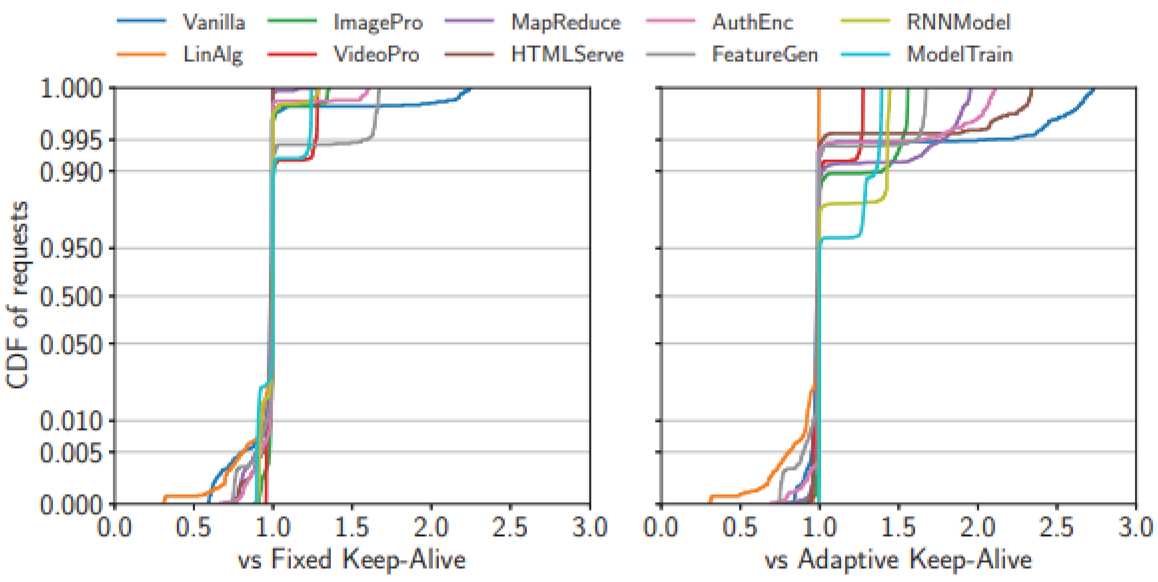
Evaluation

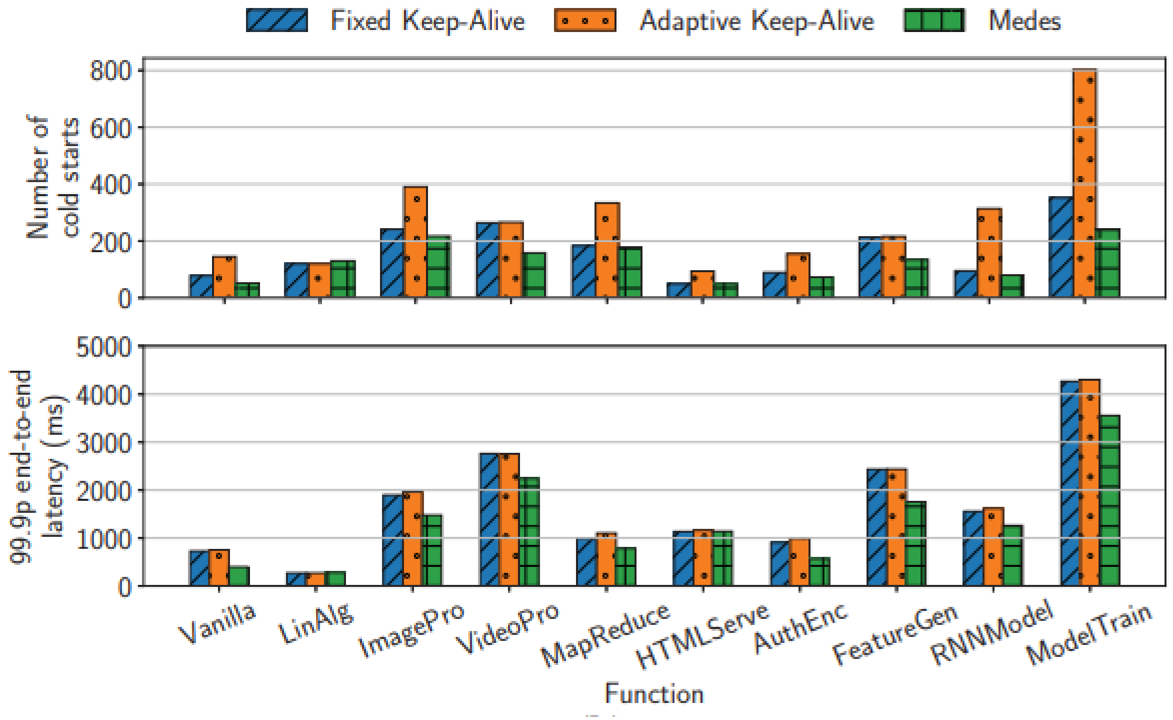
- evaluate Medes on a 20 node cluster
- All nodes have 64GB memory and a 10Gbps NIC
- One node out of these acts as the controller
- The remaining nodes are all accessible via an RDMA network

Baselines:

- fixed keep-alive policy
- adaptive keep-alive policy

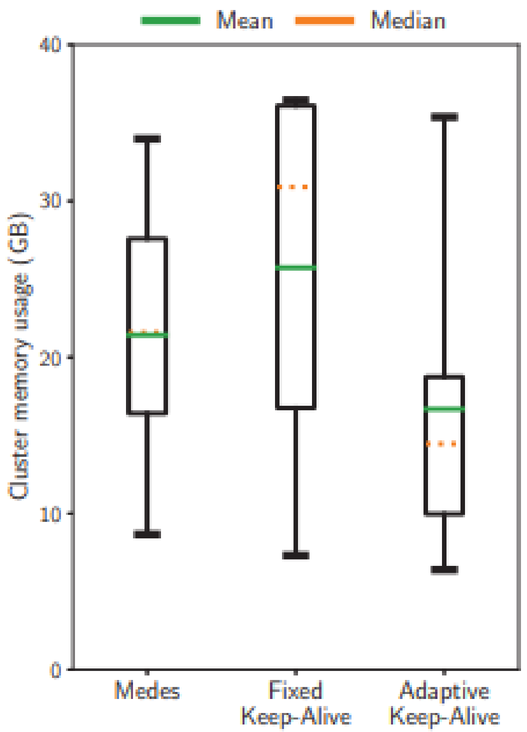
Evaluation

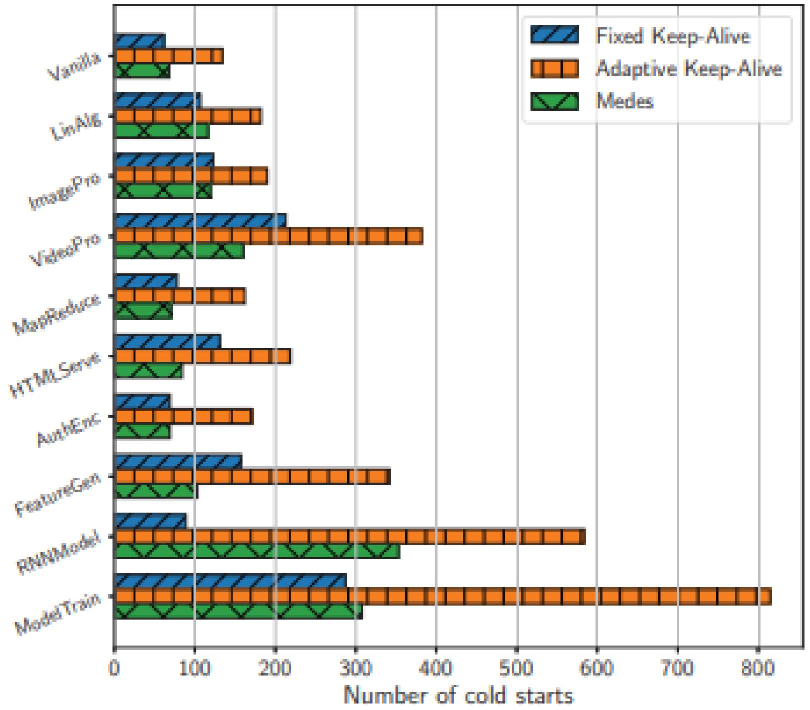




- Medes can provide up to 2.25× and 2.75× improvements in the end-to-end latencies
- Medes can provide up to 1.85× and 6.2× reductions in the number of cold starts across applications

Evaluation





- Medes can meet the latency targets in a smaller memory footprint as compared to fixed keep-alive policies.

About

Why choose

- Deepen the understanding of cloud computing
- Learned the possible dedup methods that may be applied between node systems