DADI: Block-Level Image Service for Agile and Elastic Application Deployment

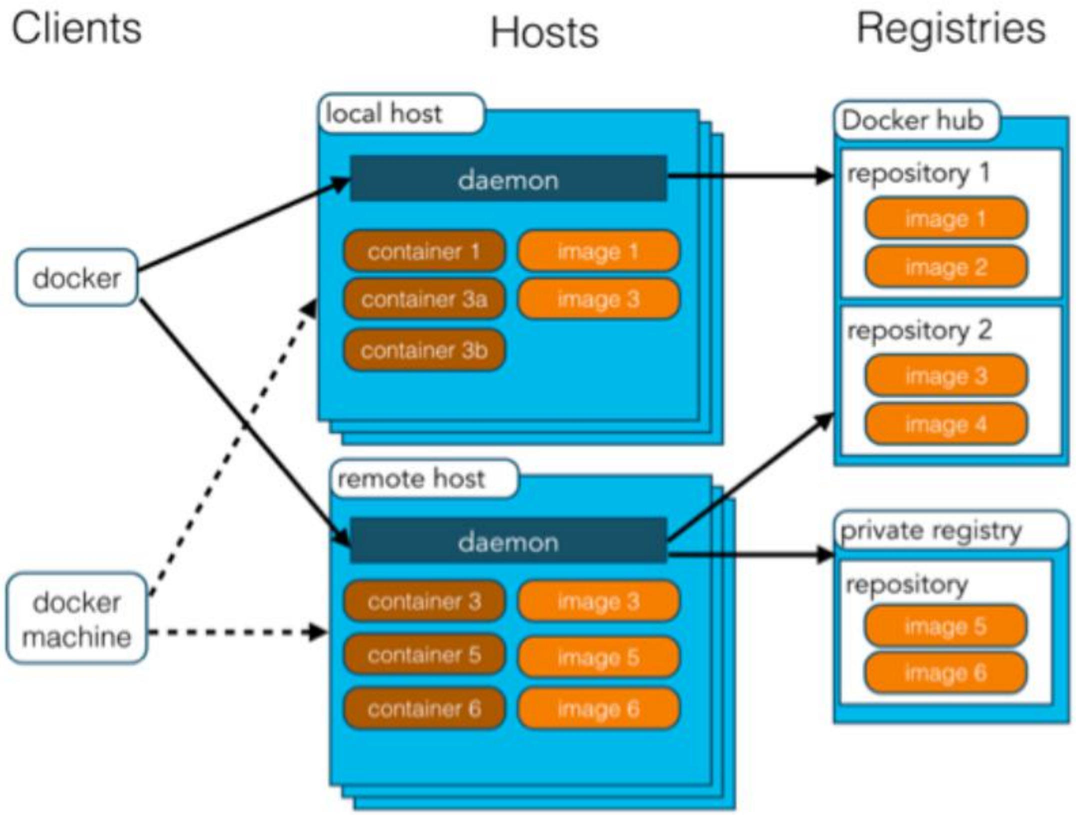
Huiba Li, Yifan Yuan, Rui Du, Kai Ma, Lanzheng Liu, and Windsor Hsu,

Alibaba Group

ATC 2020

Background

- **elastic container deployment**



- Push/Pull
- Image
  - layer
- Container
  - an running instance

Traditonal designs

- Image
  - Common layer are downloaded only once,shared by mutiple container as needed.
  - Each layer is a tarball of differences (addition, deletion or update of files) from a previous layer.

Figure 1: Layered Container Image. The image layers (L1, L2) are read-only shared by multiple containers (C1, C2) while the container layers (LC) are privately writable.

- Writing to a file in the image must copy the entire file to the writable layer.
- loading an instance costs much time(downlaod and decompress)

which is also the problem we attention

Remote Image

- Remote Image
  - Image data is fetched over the network on-demand in a fine-grained manner rather than downloaded as a whole.
  - Reason: Part of the image is actually needed during the typical life-cycle of a container.
  - Problem:
    - random read access(Remote Image)
    - the standard layer tarball was designed for sequential reading and unpacking

Types of remote Image(existed)

- File-System-Based
  - natural extension of container image
  - difficult to statisfy the image service such as Windows workloads running on Linux hosts.
  - hard to support advanced features, hard link, cross-layer reference, etc
- Block-Snapshot-Based
  - Using the similarity "copy-on write" between snapshot in block stores and layer in container.
  - not identical things, one is a point-in-time view and one refers to the incremental change relative to a state.
  - need to improve

File or Block?

- The benefits of a layered image are not contingent on representing the layers as sets of file changes in practice. (observation)
- Block-based layers can achieve the same effect and it also has its own advantage, such as fine-grained on-demand data transfer of remote images etc.
- Using Block-based.

High level Idea

- At the basis of using block-device-based image, improve technologies such as downloading, compression/ decompression, image manage to improve the performance.

DADI Image

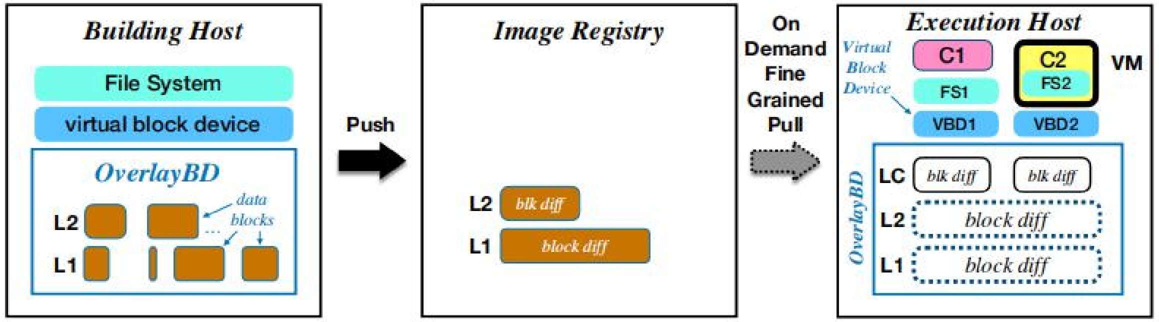


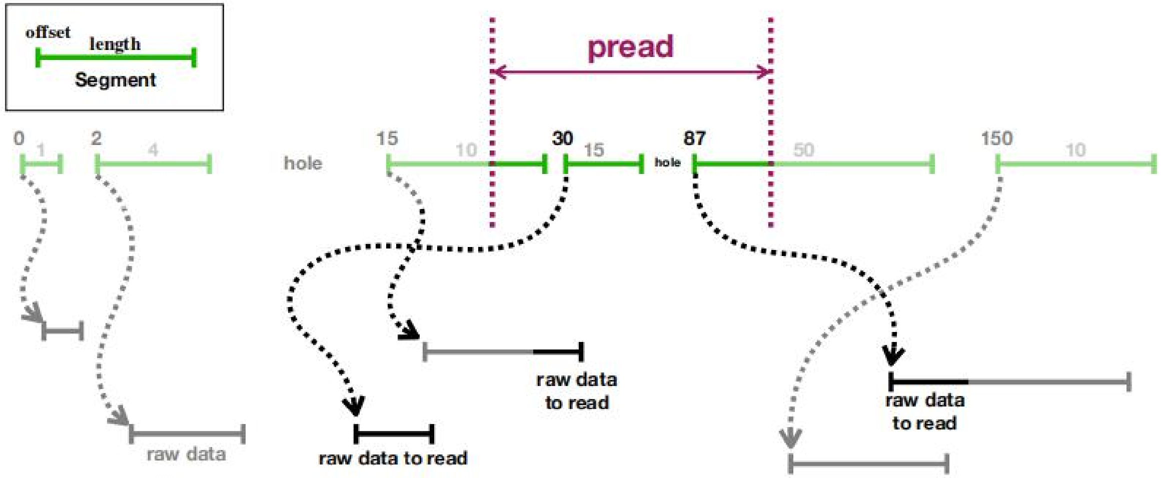
Figure 2: DADI Image. DADI image layer (L1, L2) consists of modified data blocks. DADI uses an overlay block device to provide each container (C1, C2) with a merged view of its layers.

- Models an image as a virtual block device while retaining the layered feature.
- Each layer is viewed a collection of modified data blocks under the file system (selective), the layer is constituted og raw data and its index.
- Use layer and changeset: <u>for any block, the latest change takes effect.</u>

Index

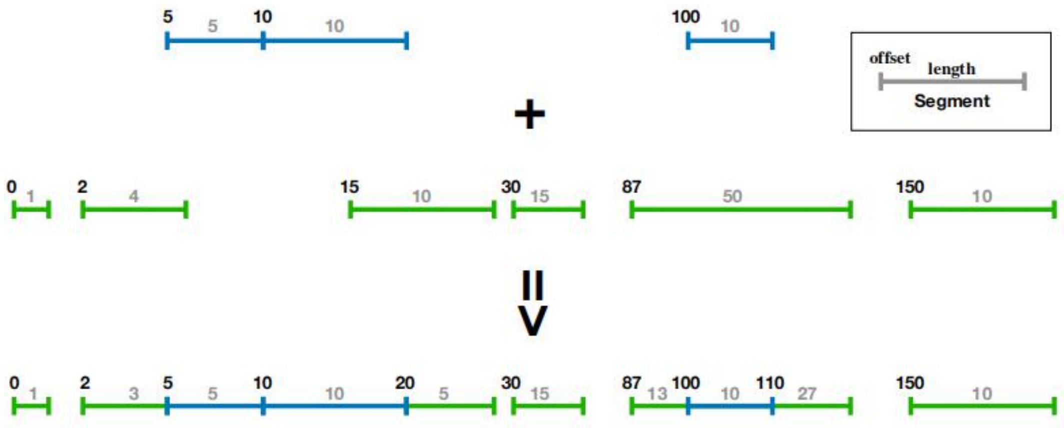
- Raw data Index :
  - based on variable-length segment
  - segments in one layer is non-overlapping.
  - Index look-up in one layer is just a binary search.

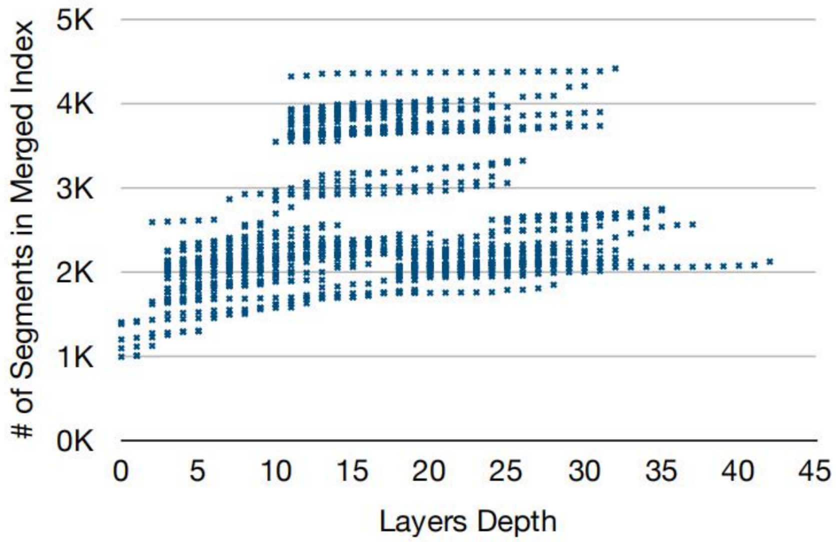
```
struct Segment {
  uint64_t offset:48;  // offset in image's LBA
  uint16_t length;     // length of the change
  uint64_t moffset:48; // mapped offset in layer blob
  uint16_t pos:12;     // position in the layer stack
  uint8_t flags:4;     // zeroed? etc.
  uint64_t end() {return offset + length;}
};
```



Merged View of Layers

- Index look-up time complexity
  - n layers, M segements. ➡ n*logM
  - The whole index loke-up time increases as n increases.
  - Take layer change rule into consideration, the index can be merged.
  - Just need a pos point record the segment comes from which layer.





logM

ZFile

- Support random read operation and online decompression.
- fixed length file is compressed into variable length block.
- Decompressed only needed chunks

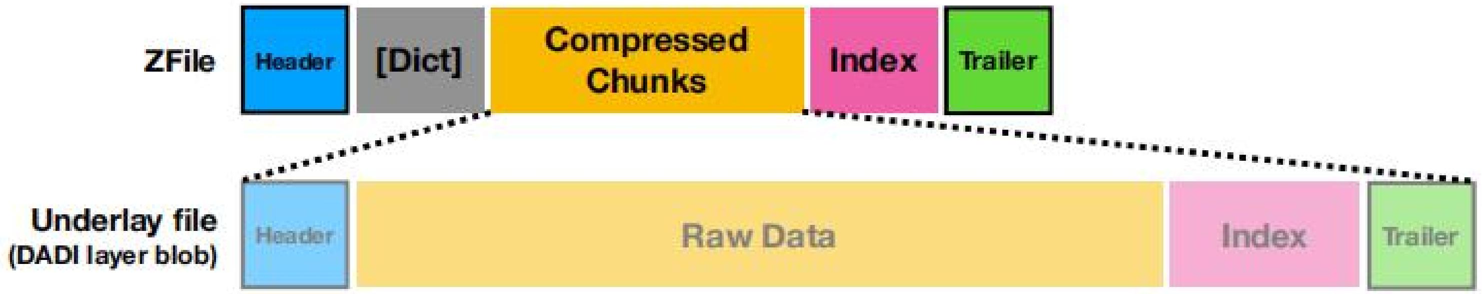


Figure 8: ZFile Format.

- Dict: record compression algorithm (lz4, zstd, gzip)
- Index: record compressed Chunks offset and length.

Writable Layer

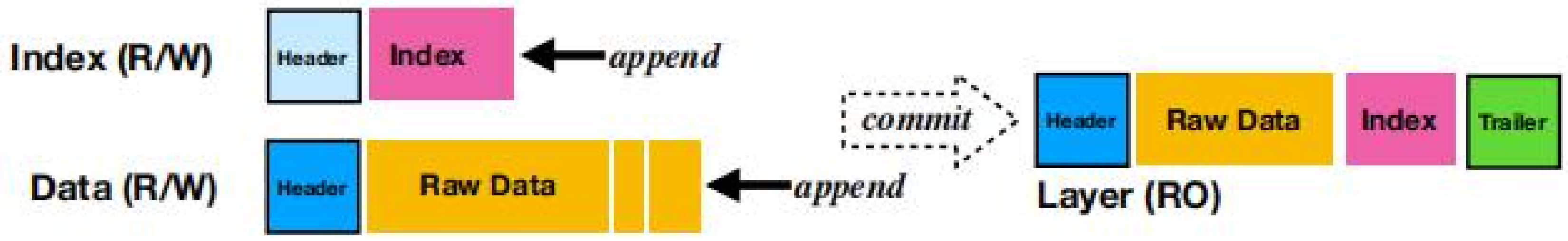
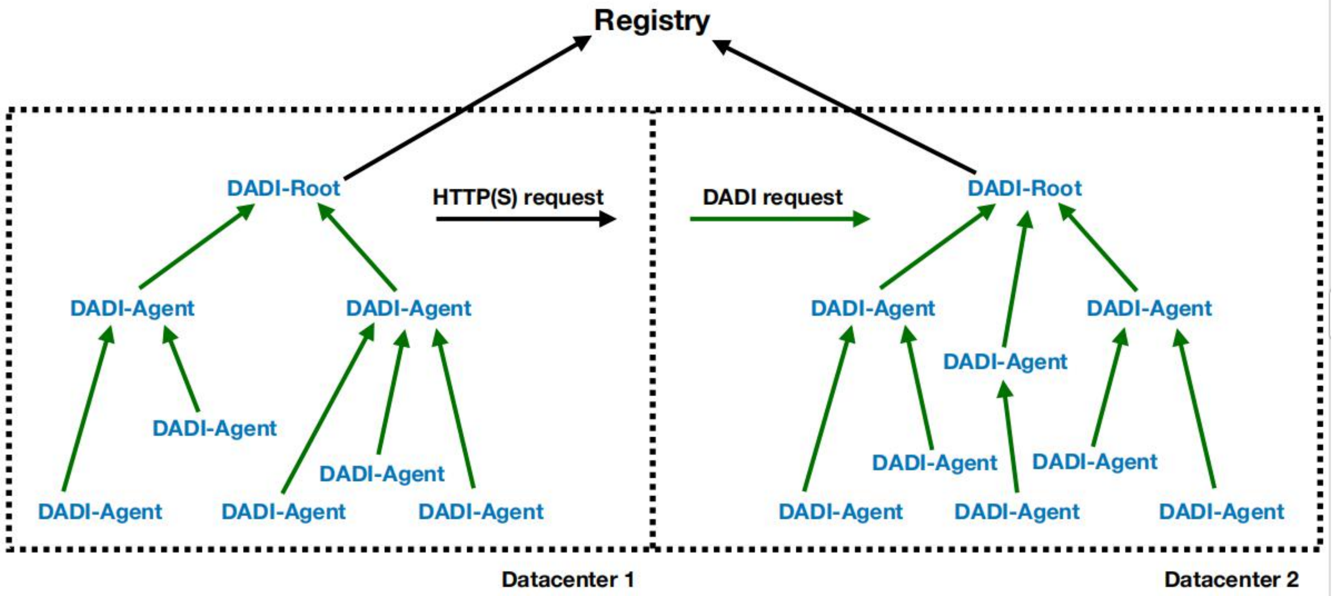


Figure 10: DADI's Writable Layer.

- log-structured
- The index for the writable layer is maintained in memory as a red-black tree.
- When committed, copy the live data blocks and index records to a new file in layer format, sorting and combining them according to their LBAs.

P2P Data Transfer



- a tree-structured overlay topology instead of the rarest-first policy
- DADI-Root manages the topology trees within its own datacenter, fetches data blocks from the Registry to a local persistent cache
- data requests upward, recursively.

Implementation

- Implement I/O Path for cgroups Runtime and Virtualized Runtime
- Container Engine Integration
  - Integrated with Docker through a Docker graph driver plugin
  - implemented the drivers to recognize existing and DADI image formats.
- Image Building
  - stack-and-commit
- Provide Deployment Options
  - image share?
  - where to fetch data?

Evaluation

- Comparison
  - the container startup latency (DADI to that with the standard tarball image, Slacker, CRFS, LVM, and P2P image download)
  - analyze the I/O performance inside the container.
- NVMe SSDs as local storage, virtual disks on public clouds refer to such a disk as “cloud disk”
- Physical servers: dual-way multi-core Xeon CPUs and 10GbE or higher-speed NICs.Each VM is equipped with 4 CPU cores and 8 GBs of memory

Startup latency (a single instance)

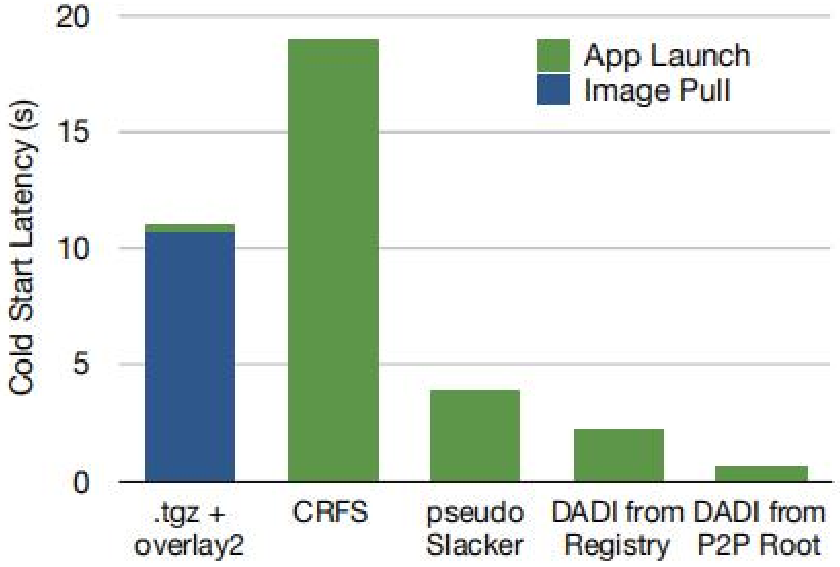


Figure 14: Cold Startup Latency.

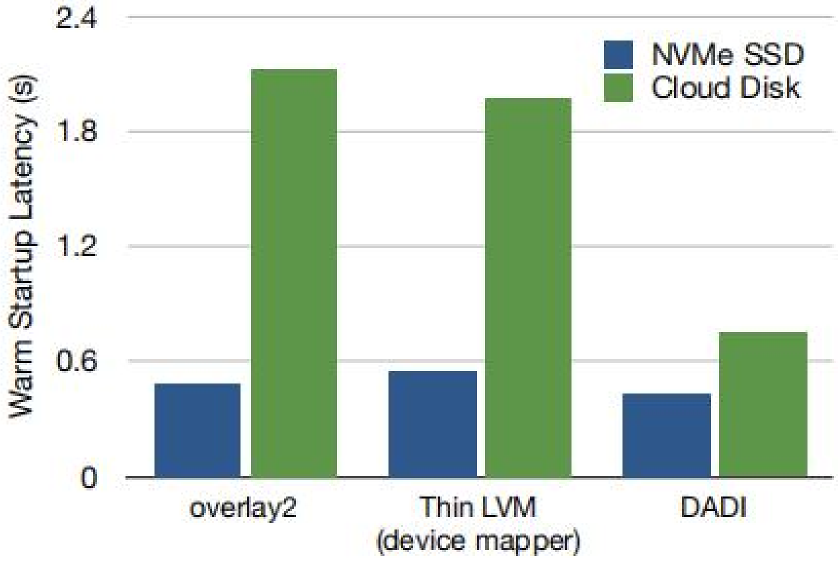


Figure 15: Warm Startup Latency.

- Container cold startup time is reduced with DADI obviously.
- DADI performs 15%~25% better than overlayfs and LVM on NVMe SSD, and more than 2 times better on cloud disk

Startup latency

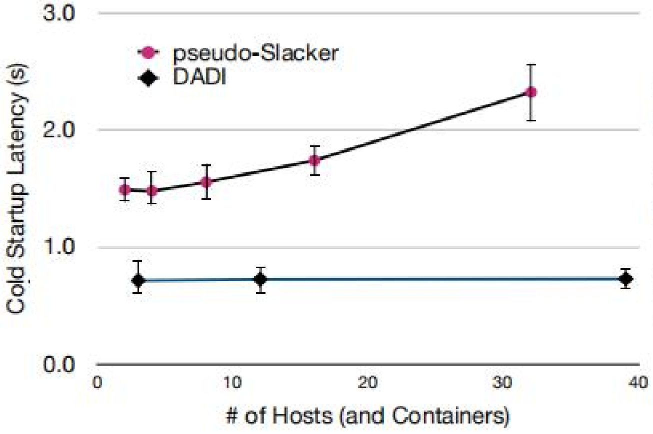


Figure 17: Batch Cold Startup Latency. Bars indicate 10 and 90 percentiles.

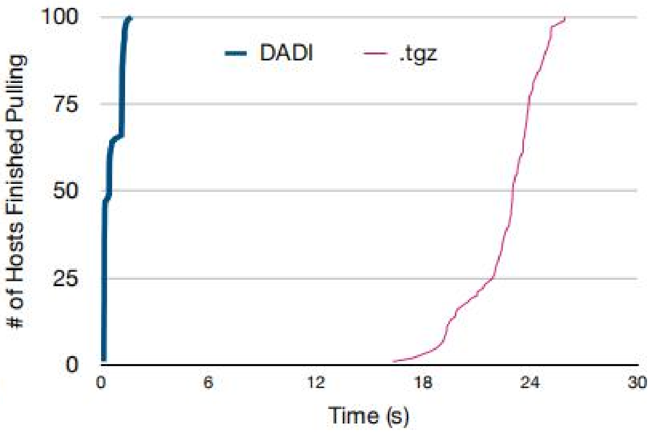


Figure 18: Time to Pull Image in Production Environment.

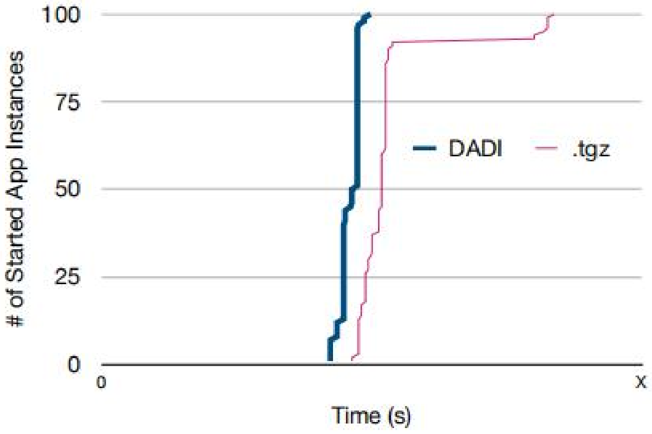


Figure 19: Time to Launch Application in Production Environment.

- DADI realises second-level application deployment capability

Scalability

Its image consists of 16 layers with a total size of 575MB in ZFile format and 894MB uncompressed.

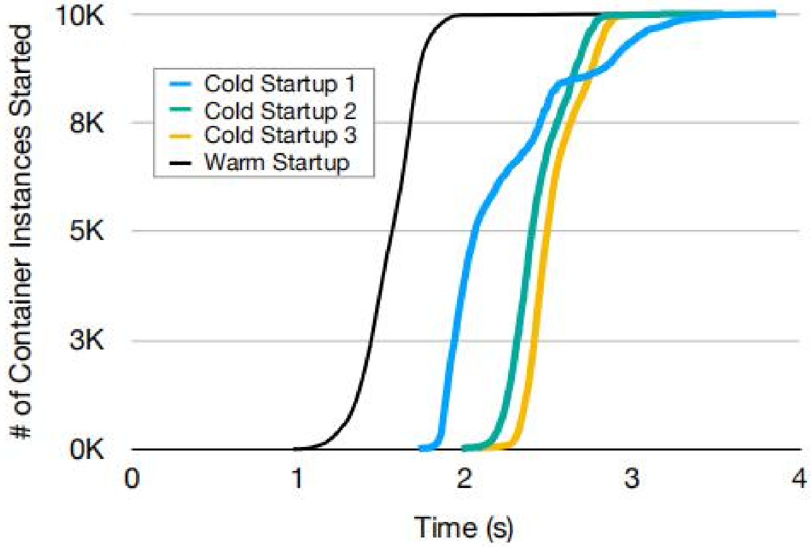


Figure 20: Startup Latency using DADI (Large-Scale Startup).

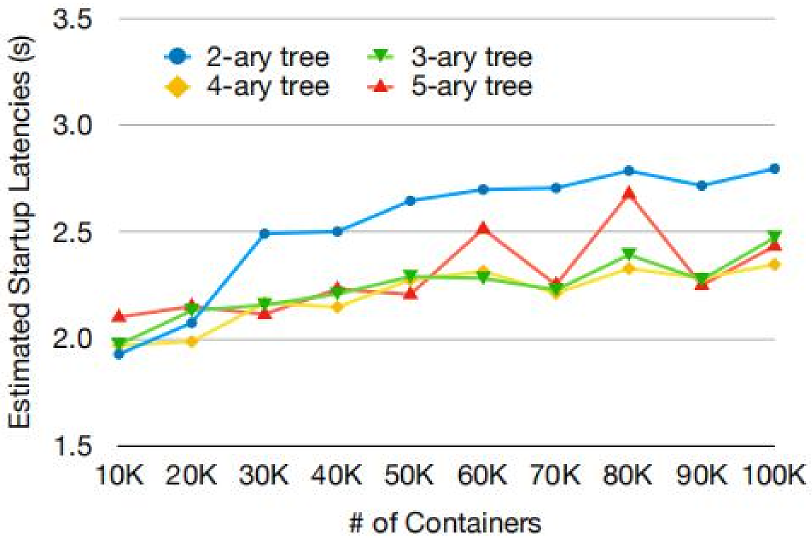


Figure 21: Projected Startup Latency using DADI (Hyper-Scale Startup).

- Even each host has a total of 10,000 containers, startup time is very fast.

- The startup time is largely flat as the number of containers increases. A binary tree for P2P is best when there are fewer than 20,000 participating hosts. A 3-ary or 4-ary tree works better.

I/O Performance

DADI offers comparable performance to LVM

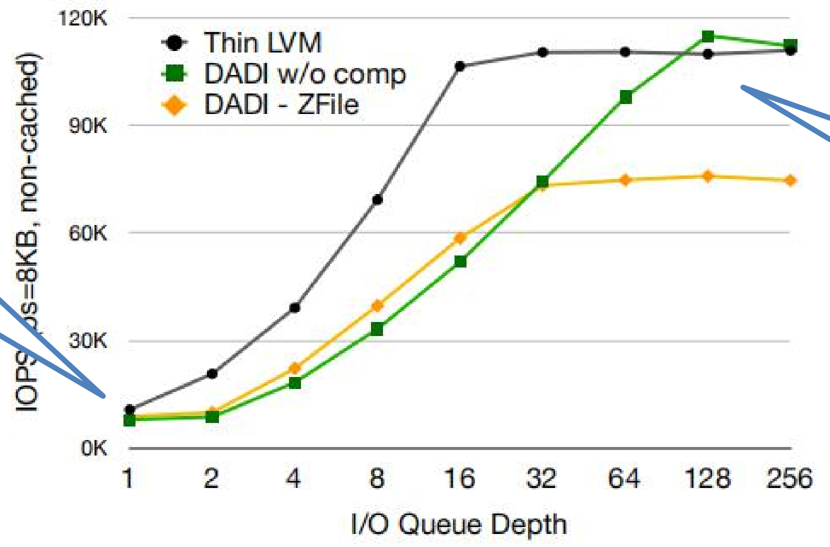


Figure 22: Uncached Random Read Performance.

DADI's index is more efficient than that of LVM.

- DADI with compression performs 10%~20% better than without compression when the I/O queue depth is less than 32

I/O performance

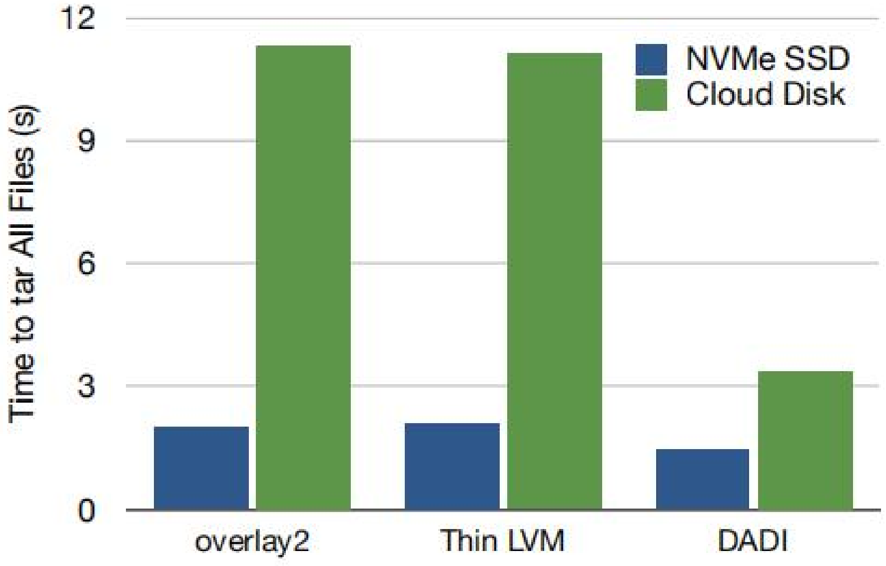


Figure 24: Time to `tar` All Files.

- ➢DADI's compression performance is effiectly.
- ➢Reason: the entire image comes from inside the container and the effect of compression in reducing the amount of data transferred

Image Building Speed

- The dockerfile creates 15 new layers comprising 7,944 files with a total size of 545MB, and includes a few chmod operations that trigger copy-ups in overlayfs-backed images
- the time to commit or compress the image is not included in this measurement

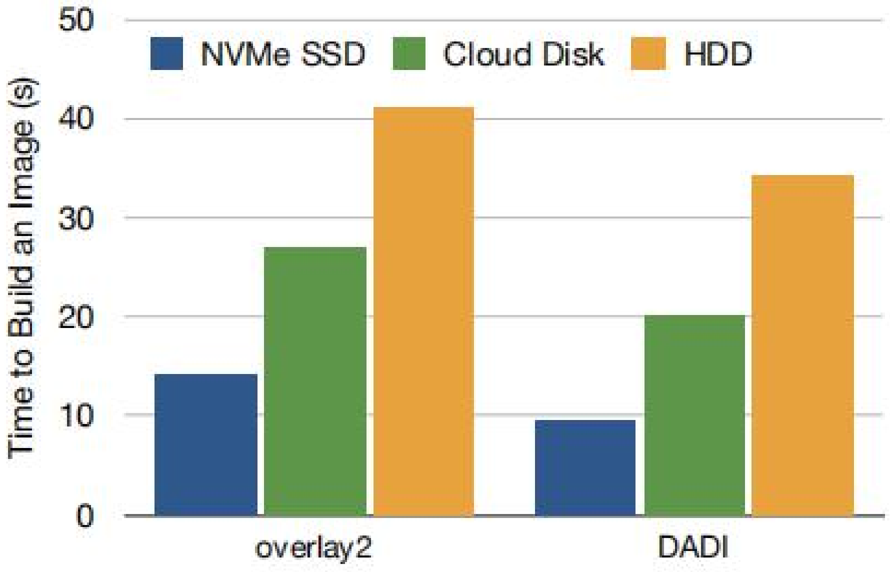


Figure 25: Time to Build Image.

- 20%~40% faster on DADI than on overlayfs

Conclusion

- Designed and implemented a block-level remote image service for containers.
- File system and platform agnostic, enabling applications to be elastically deployed in different environments.
- Further facilitates optimizations to increase agility.
- Superior evaluation environment

Comments