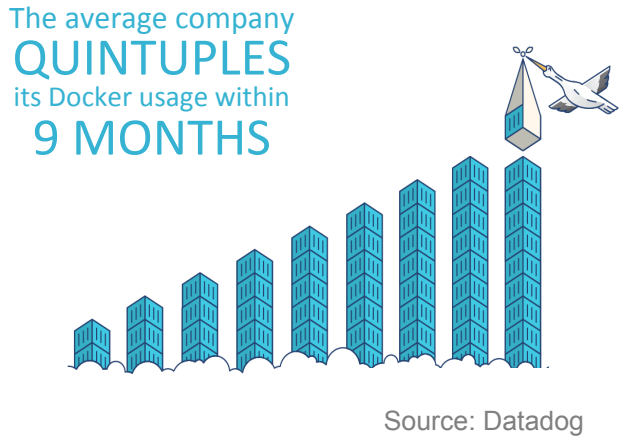
Improving Docker Registry Design based on Production Workload Analysis

Ali Anwar, Mohamed Mohamed, Vasily Tarasov, Michael Littley, Lukas Rupprecht, Yue Cheng, Nannan Zhao, Dimitrios Skourtis, Amit S. Warke, Heiko Ludwig, Dean Hildebrand, and Ali R. Butt

Containers will be a $2.7B market by 2020*

- Containers accelerate software development and distribution.
- In 2017 alone, Docker adoption went up by 40%.
- Containers use in enterprise and cloud infrastructure is expected to grow much faster.



Source: Datadog

*http://bit.ly/2uryjDI

Docker usage patterns remain a mystery

- How are Docker containers used and managed?
- How can we streamline Docker workflows?
- How do we facilitate Docker performance analysis?

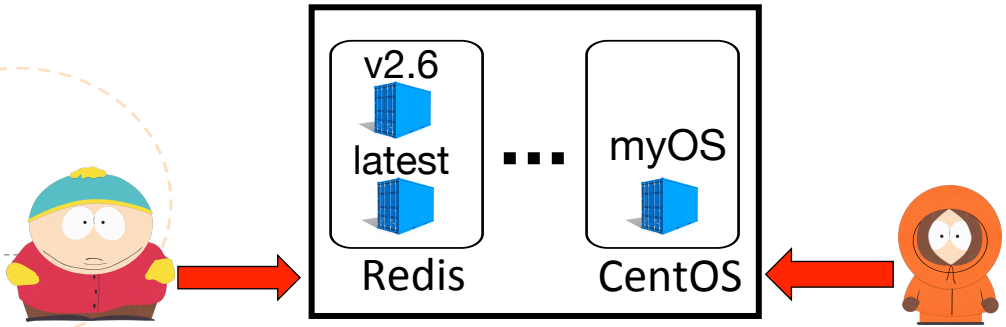
Our contribution: Characterization and optimization of Docker workflow

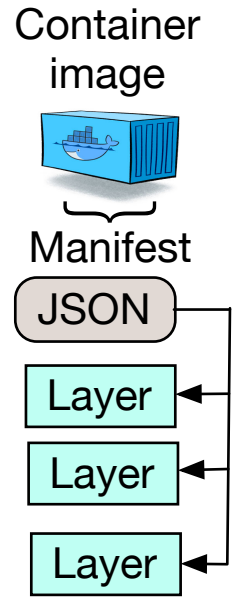
- Conduct a large-scale analysis of a real-world Docker workload from geo-distributed IBM container service
- Provide insights and develop heuristics to increase Docker performance
- Develop an open source Docker workflow analysis tool*

* https://dssl.cs.vt.edu/drtp/

Background: Docker container image

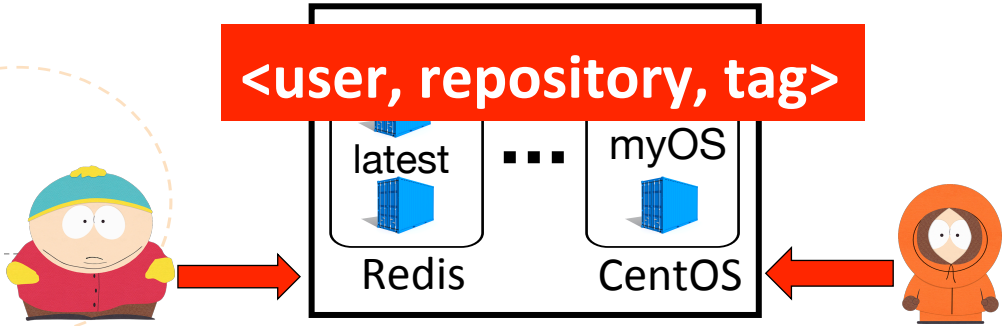
- Container images are divided into **layers.**
- The metadata file is called **manifest**.
- **Users** create **repositories** to store images.
- Images in a repository can have different **tags** (versions).

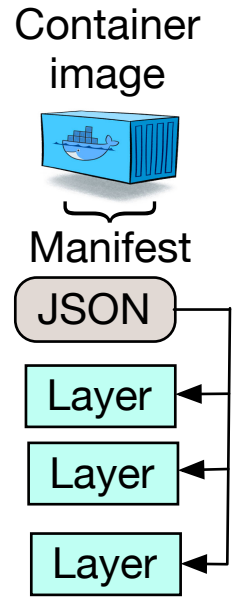




Background: Docker container image

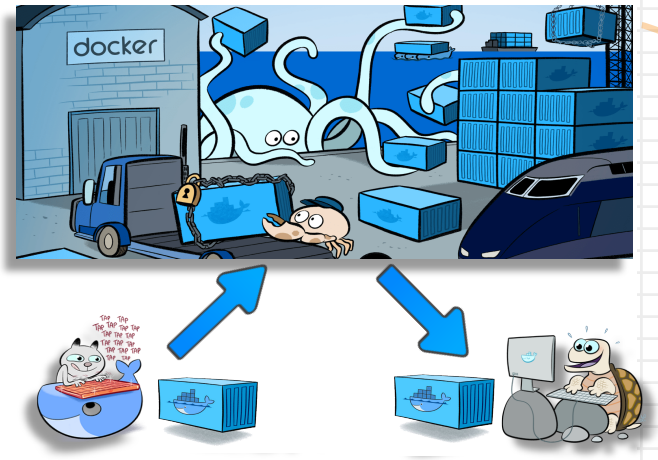
- Container images are divided into **layers.**
- The metadata file is called **manifest**.
- **Users** create **repositories** to store images.
- Images in a repository can have different **tags** (versions).





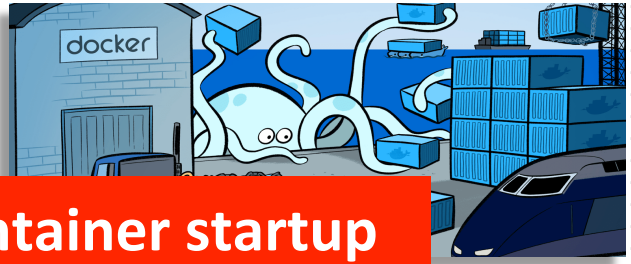
Background: Docker container registry

- Docker container images are stored online in **Docker registry**.
  - Push image:
    1. HEAD layers
    2. POST/PUT layer
    3. PUT manifest
  - Pull image:
    1. GET manifest
    2. GET layers

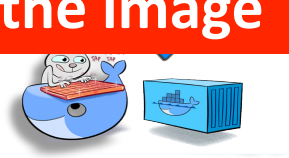


Background: Docker container registry

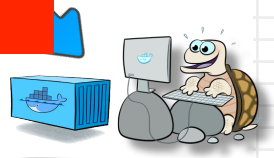
- Docker container images are stored online in **Docker registry**.
  - Push image:
    1. HEAD layers
  - Pull image:
    1. GET manifest
    2. GET layers



Significant amount of a container startup time is spent in pulling the image



docker push



docker pull

The IBM Cloud Docker registry traces

- Capture a diverse set of customers: individuals, small & medium businesses, government institutions
- Cover five geographical locations and seven availability zones
- Span 75 days and 38M requests that account for more than ~181TB of data transferred

IBM Docker registry service

Five geographical locations constitute seven Availability Zones (AZ):

Production

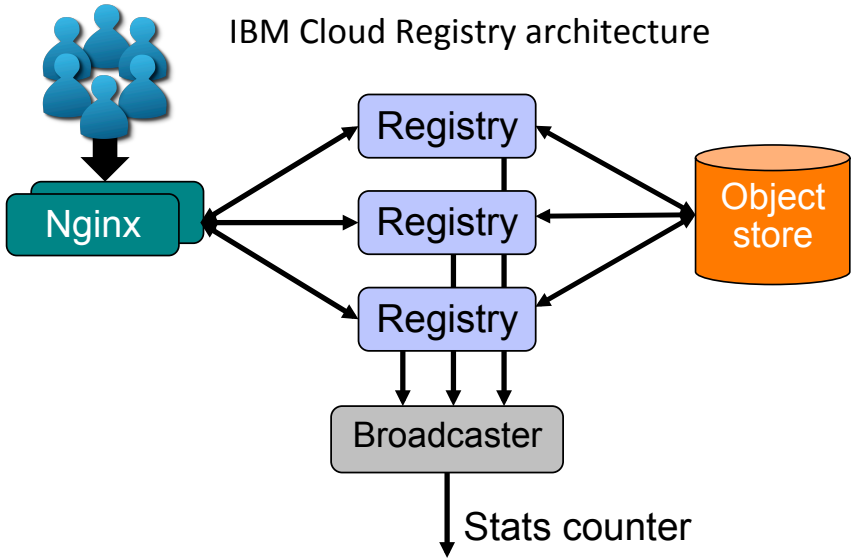
1. Dallas (**dal**)
2. London (**lon**)
3. Frankfurt (**fra**)
4. Sydney (**syd**)

IBM Internal

5. Staging (**stg**)

Testing*

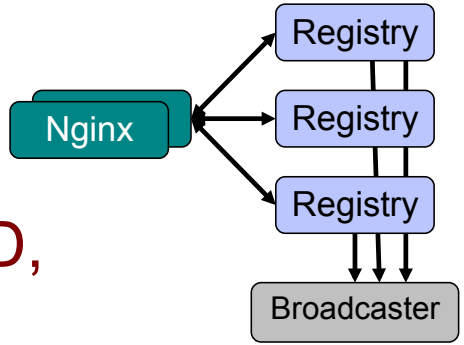
6. Prestaging (**prs**)
7. Development (**dev**)



*The registry setup is identical, except prs and dev are only half the size of the other Azs.

Tracing methodology

- Collected data from **Registry**, **Nginx**, and **Broadcaster**
- Studied requests: GET, PUT, HEAD, PATCH, POST
- Combined traces by matching the incoming HTTP request identifier across the components
- Removed redundant fields and anonymized the traces

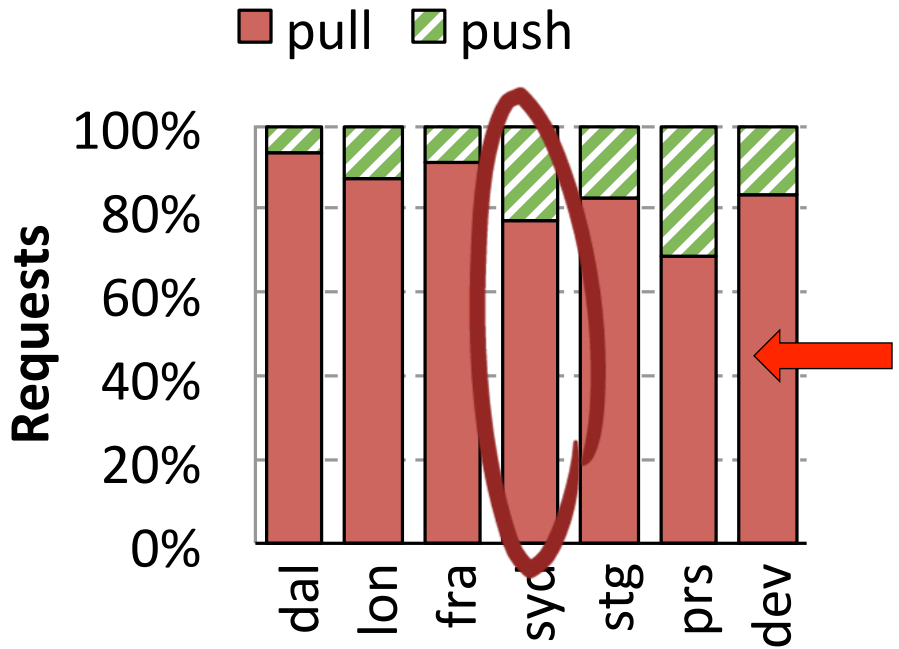


Anonymized log sample


```
{
    "host": "579633FD",
    "http.request.duration": 0.879271282,
    "http.request.method": "GET",
    "http.request.remoteaddr": "40535JF8",
    "http.request.uri": "v2/CA64KJ67/AS87D65G/blobs/B26S986D",
    "http.request.useragent": "docker/17.04.0-ce go/go1.7.5..)",
    "http.response.status": 200,
    "http.response.written": 1518,
    "id": "9F63984H",
    "timestamp": "2017-07-01T01:39:37.098Z"
}
```

Q1: What is the distribution of request types?

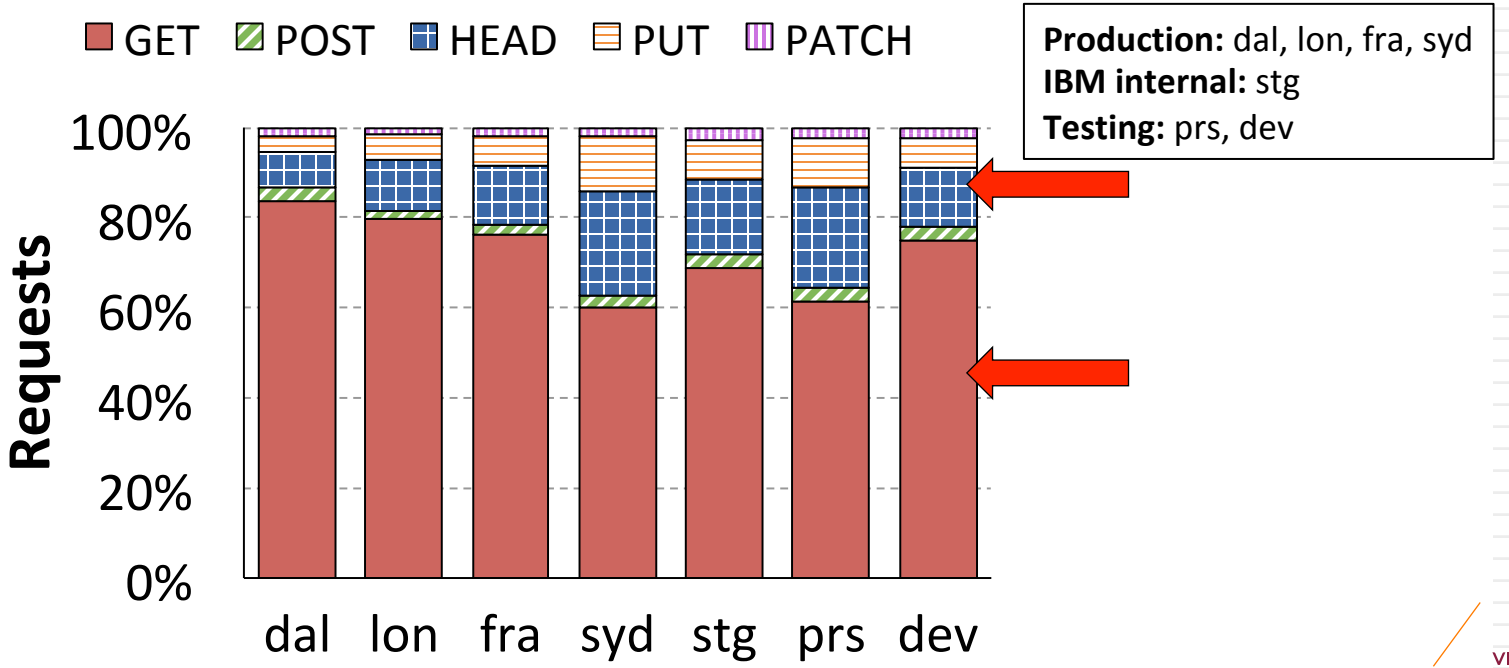
80%–95% of requests are reads (pulls)



Production: dal, lon, fra, syd
IBM internal: stg
Testing: prs, dev

Q1: What is the distribution of request types?

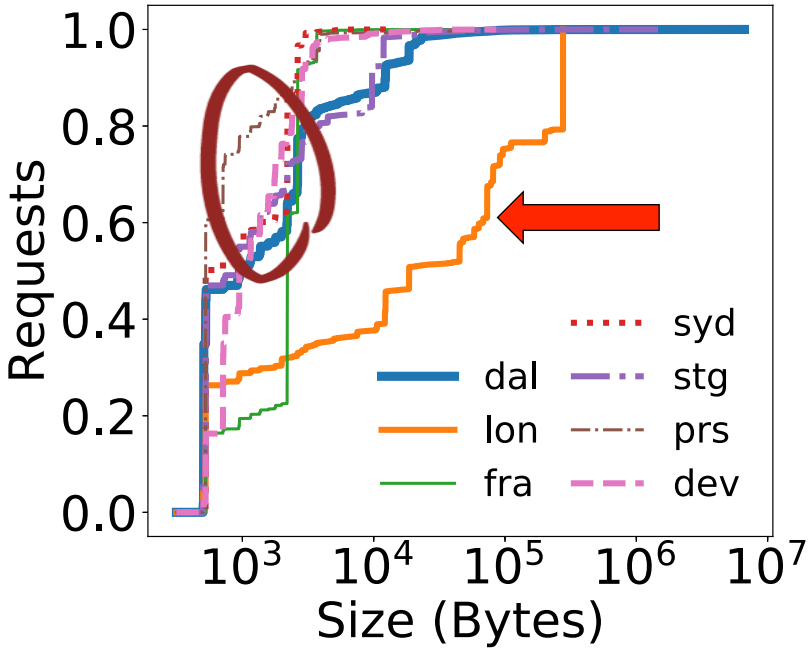
60% of the requests are GET and 10%–22% are HEAD requests



VIRGINIA TECH

Q2: What is the manifest size distribution?

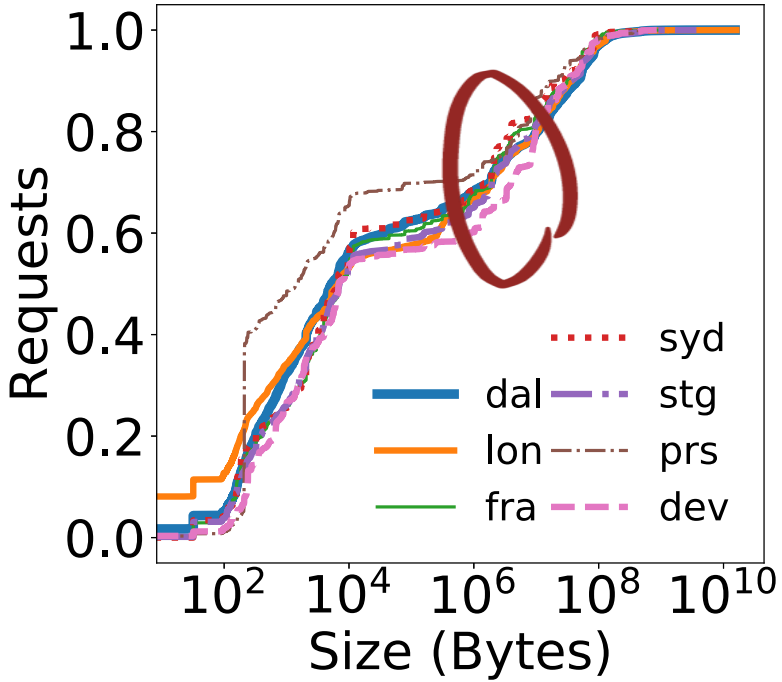
Typical manifest size is around 1 KB



Production: dal, lon, fra, syd
IBM internal: stg
Testing: prs, dev

Q3: What is the layer size distribution?

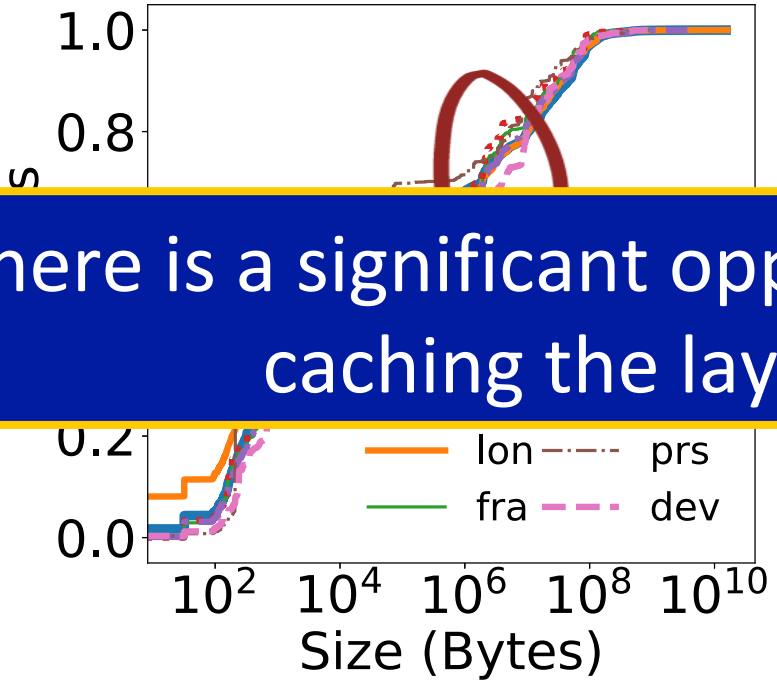
65% of the layers are smaller than 1 MB and around 80% are smaller than 10 MB



Production: dal, lon, fra, syd
IBM internal: stg
Testing: prs, dev

Q3: What is the layer size distribution?

65% of the layers are smaller than 1 MB and around 80% are smaller than 10 MB

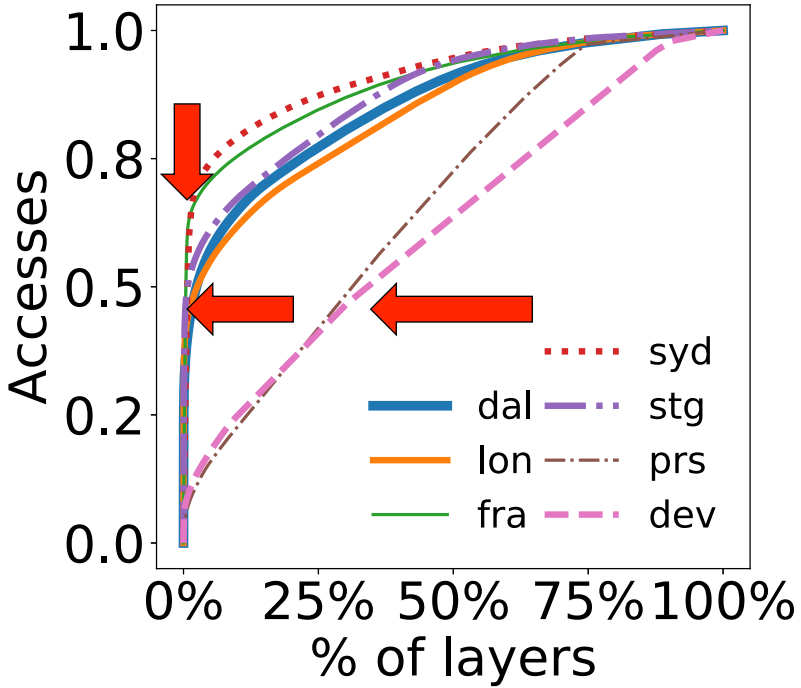


Production: dal, lon, fra, syd
IBM internal: stg
Testing: prs, dev

There is a significant opportunity for caching the layers

Q4: Is there spatial locality?

1% of most accessed layers account for 42% and 59% of all requests in dal and syd, respectively

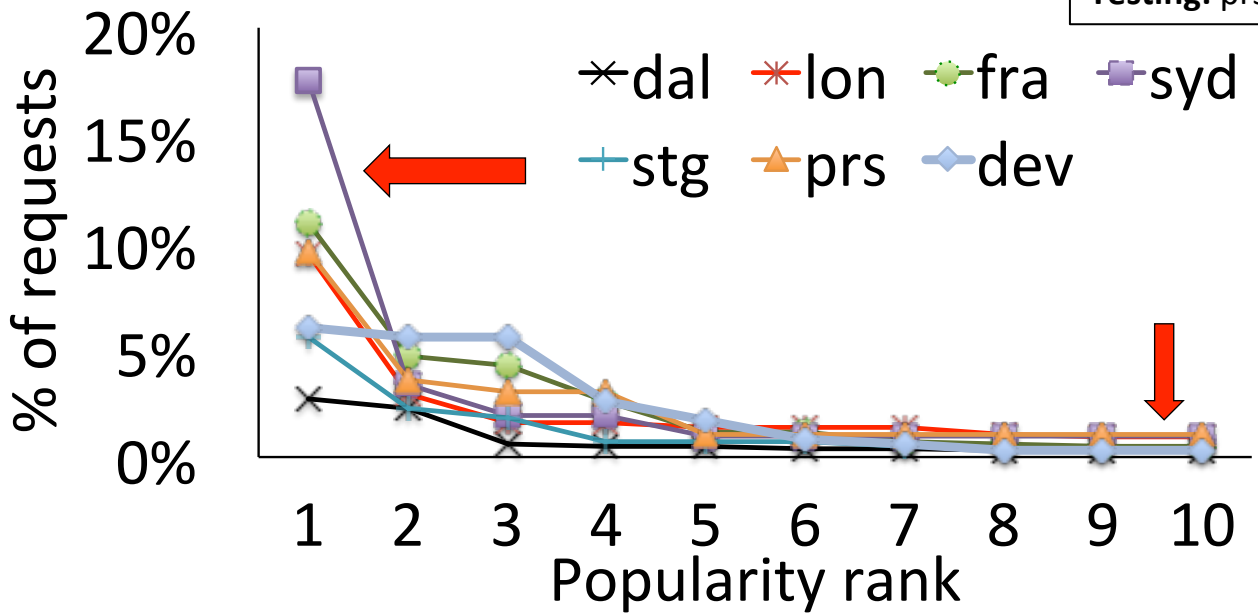


Production: dal, lon, fra, syd
IBM internal: stg
Testing: prs, dev

Q4: Is there spatial locality?

The popularity rate drops rapidly as we move from most popular to tenth most popular layer

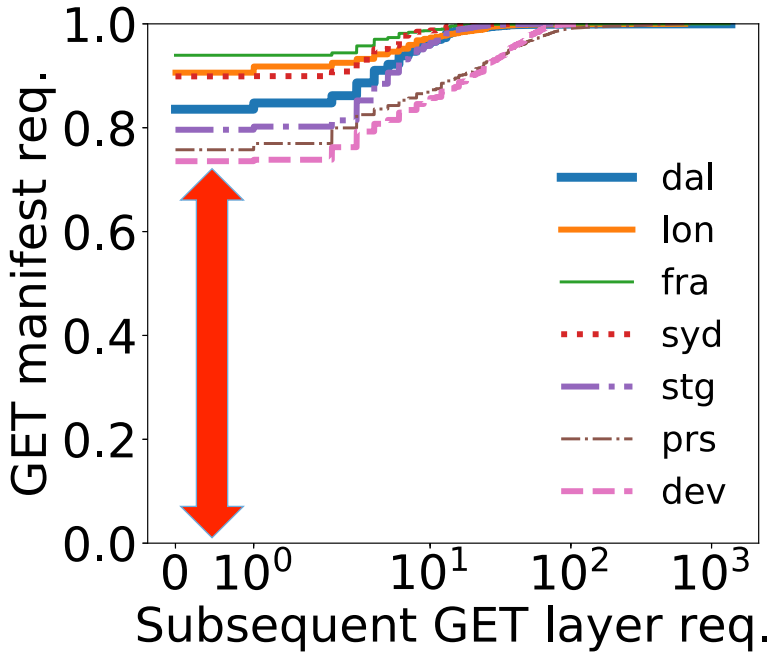
Production: dal, lon, fra, syd
IBM internal: stg
Testing: prs, dev



VIRGINIA TECH

Q5: Can future requests be predicted?

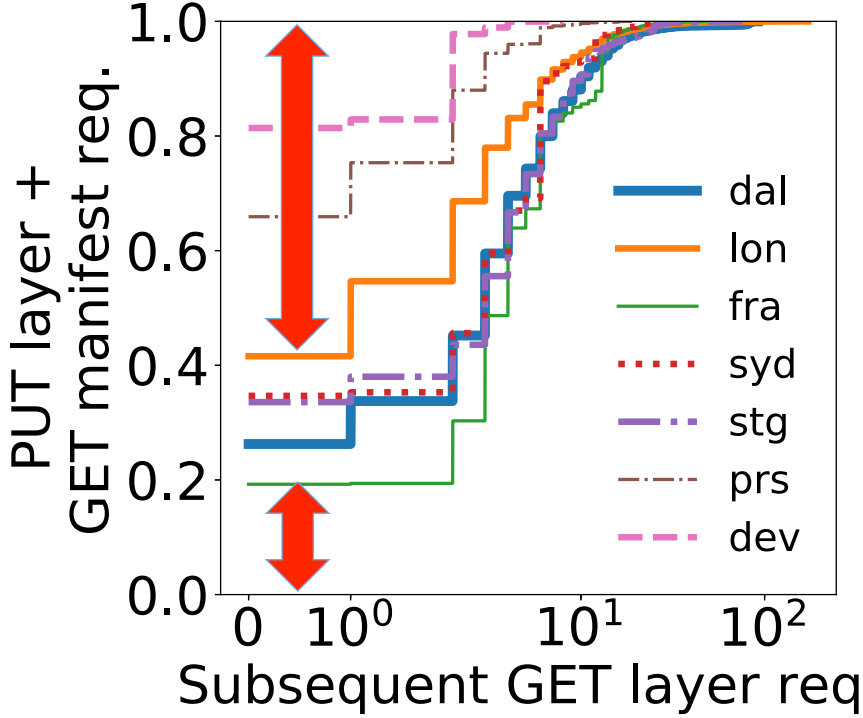
GET manifest requests are not followed by any subsequent GET layer request



Production: dal, lon, fra, syd
IBM internal: stg
Testing: prs, dev

Q5: Can future requests be predicted?

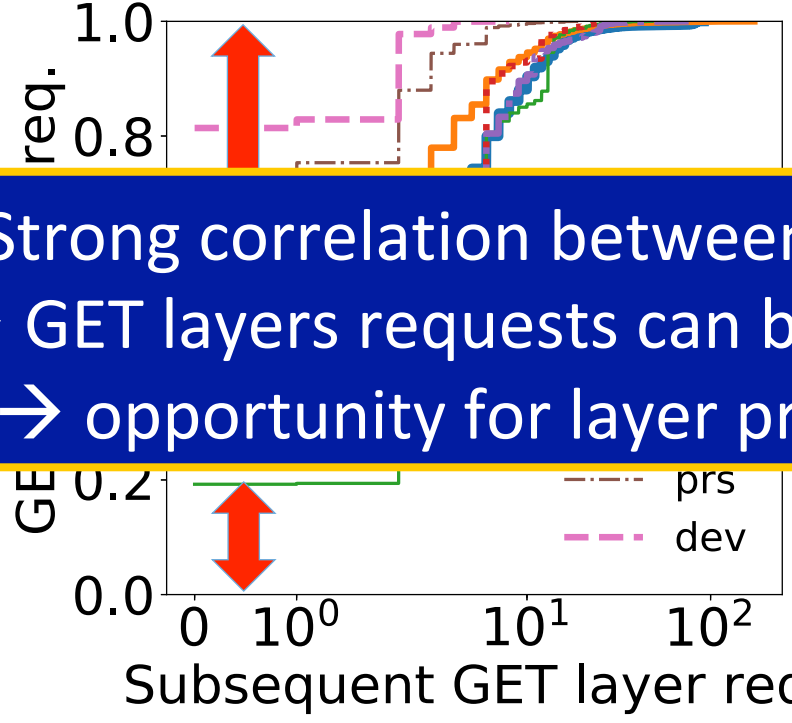
Significant increase in subsequent GET layer requests within a session



Production: dal, lon, fra, syd
IBM internal: stg
Testing: prs, dev

Q5: Can future requests be predicted?

Significant increase in subsequent GET layer requests within a session



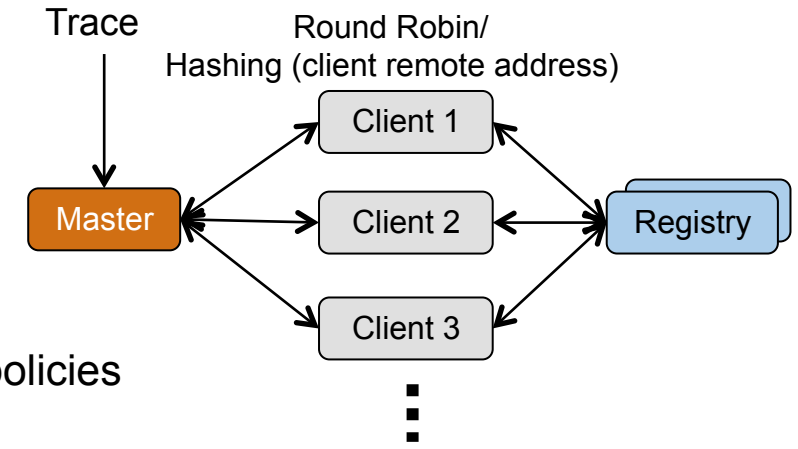
Production: dal, lon, fra, syd
IBM internal: stg
Testing: prs, dev

Strong correlation between requests
→ GET layers requests can be predicted
→ opportunity for layer prefetching

Enabling further analysis: Trace re-player

Performance analysis mode

- Study throughput and latency
- Understand effect of CPU, Memory, Storage, Network



Offline analysis mode

- Simulate prefetching and caching policies
- Explore cache efficacy

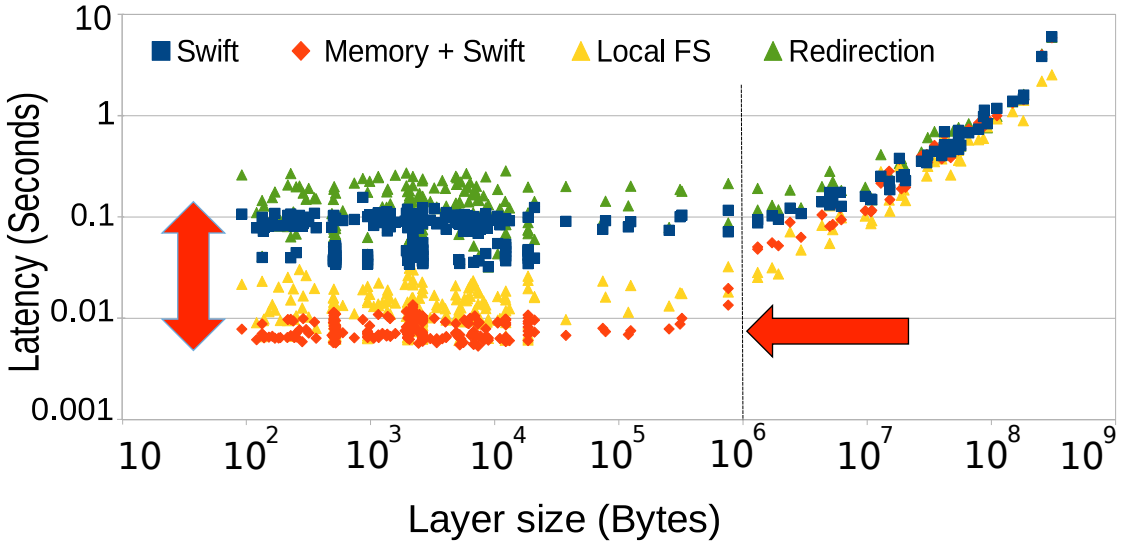
Additional analysis

- Analyze request arrival rate at user define granularity
- Study effect of deduplication on registry size

Effect of backend storage technologies

Experimental setup:

- Registry on 32 core machine with 64 GB RAM and 512 GB SSD
- Swift object store on **10** similar nodes
- Trace re-player on **6** additional nodes

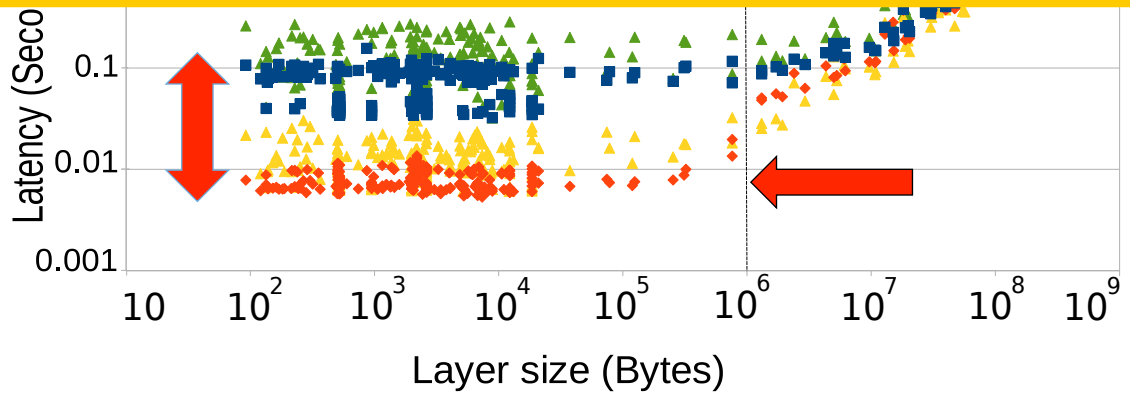


Effect of backend storage technologies

Experimental setup:

- Registry on 32 core machine with 64 GB RAM and 512 GB SSD

Fast backend storage/cache for the registry can significantly improve the overall performance



VIRGINIA TECH

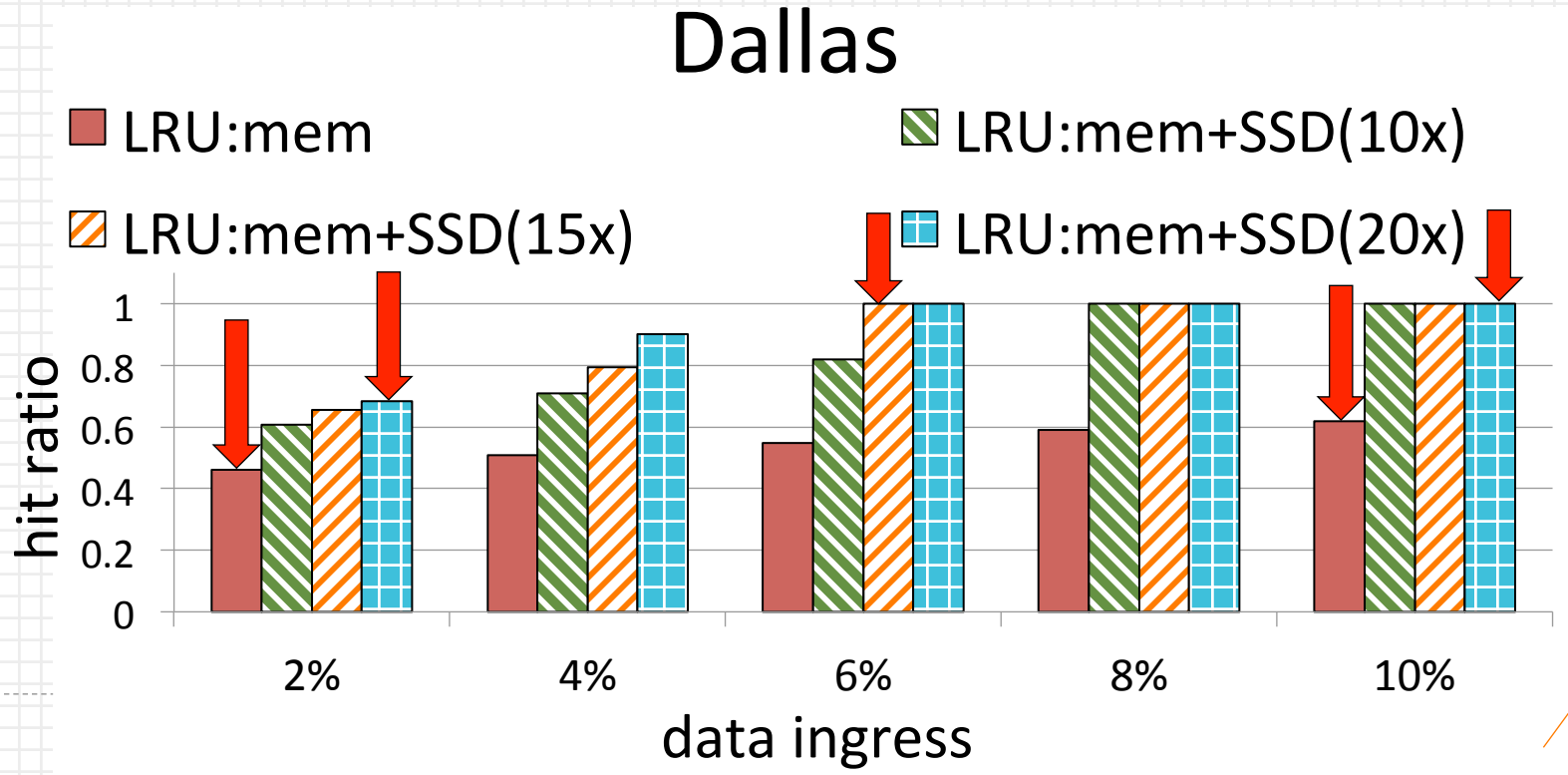
Effect of a two-level Main Memory+SSD cache

Experimental setup:

- Small layers (<100 MB) are stored in the main memory
- Replacement policy for both cache level is LRU
- Studied cache sizes:
  RAM: 2%, 4%, 6%, 8%, and 10% of the data ingress
  SSD: 10x, 15x, 20x the size of RAM cache

- Layers are content addressable
  → cache invalidation is not a problem

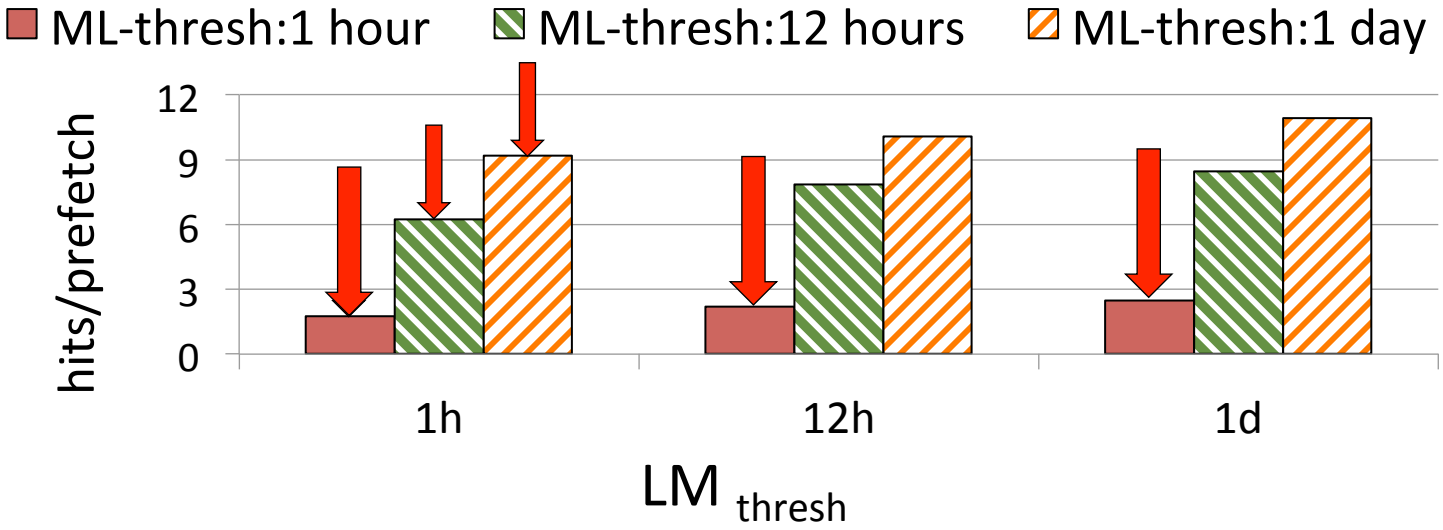
Two-level cache: Main memory+SSD



Benefit of layer prefetching

PUT layer → GET manifest → GET layer

LM_{thresh} ML_{thresh}



Summary

- We perform a quantitative characterization of a production Docker registry deployment
  - Registry workload is read intensive
  - Layers sizes are small
  - Strong correlation exists between layer requests
- We propose effective caching and prefetching strategies for container layers
- We enable further Docker investigation and optimization by making our traces and the trace re-player tool open source*

* https://dssl.cs.vt.edu/drtp/

Thank You!

Questions & contact: Ali Anwar, ali@vt.edu

https://dssl.cs.vt.edu/