Regulating Traffic in a Crowded Cache: Overcoming the Container Explosion Problem

Kevin Gao, Tim Shaffer (Advisor), Kyle Chard (Advisor)
University of California, Berkeley, University of Notre Dame, University of Chicago

Abstract

- Containers are used by multi-user online services to deploy customized computing environments for users.
- There is often significant overlap between users’ environments and thus sharing and caching containers can improve response time and performance.
- We explore caching approaches leveraging various metrics (e.g., package size, installation time, popularity) for services that create Python environments.
- We design a simulator that evaluates custom heuristics and compares their performance to an LRU Cache implementation.
- Using production workloads from Binder, we show that our methods can reduce total storage consumption by 1-3% and container creation time by 6-11% when compared with a least recently used strategy.

Methodology

Simulator

- Loops through each binder launch temporarily.
- Considers:  
  - Version mismatch
  - Constraints
- Assigns a score to each container based on combination of metric values.
- Implements container sharing and a lowest score caching policy.

Weights

- Frequency of launches for each unique container image. We see a long tail distribution where few containers are launched many times, while most containers are launched infrequently.
- Dynamic Count: average(dynamically free) / (# of launches). An online approach that counts the number of package invocations within a sliding window.

Heuristic Evaluation

- Version: average(version count) / min(current date - date of first version). How often a package version changes.
- Popularity: average(stars/median(stars) + forks/median(forks)). A combination of package popularity (Github statistics).
- Size: sum(size). Total package size on disk.
- Time: sum(time). Total package install time.
- Dynamic Count: average(dynamically free) / (# of launches).

Weighted Combination

- Weights
  - Naïve: Total package size on disk.
  - Total package install time.
  - Weighted Combination: Let M = iteration over m metrics, m_coeff = weighting metric, which is randomly generated from a Gaussian distribution.
    - 1. Naïve: sum(m_coeff * M) / sum(m_coeff)
    - 2. CacheRank: G = max(m_coeff * M) / sum(m_coeff)
    - 3. CR/s standardization: G = m_coeff * M in (0,1) / sum(m_coeff)

MRU Cache Protection

- First x containers in LRU are "protected" from eviction.
- Mixed strategy to prioritize most recently used items at the front of the cache and apply MRU protection to the back of the cache.

Container Sharing

- Base LRU: No containers are shared. Requests must use a unique container for each repository.
- Identical: Repositories with the same packages may share containers.
- Combined: Repositories with a subset of packages from another repository may share that container.

References

1. Binder: https://mybinder.org
3. T. Shaffer et al., "Solving the Container Explosion Problem for Distributed High Throughput Computing IRPS, 2020

Acknowledgements

This work was supported in part by the National Science Foundation grant NSF-TF42530 (BayGATE Ex) and NSF-2004984 (Juswic)