A Multi-level Caching Architecture for Stateful Stream Computation

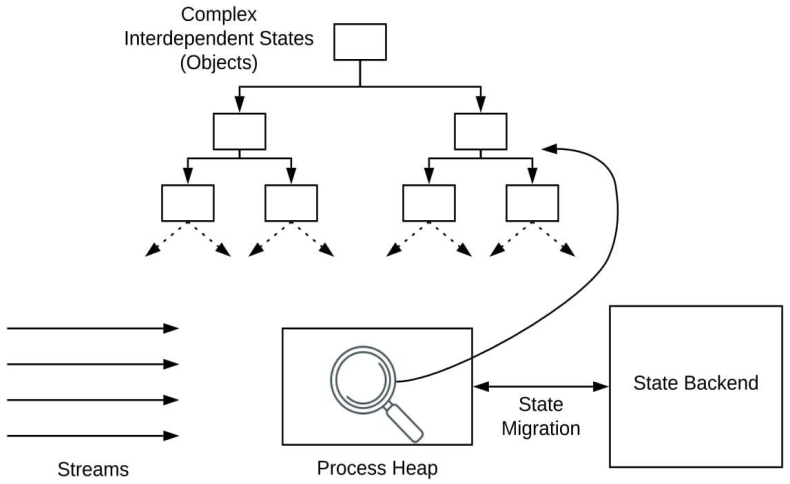
Muhammed Tawfiqul Islam, Renata Borovica-Gajic, Shanika Karunasekera

School of Computing and Information Systems (CIS)
The University of Melbourne, Australia

16th ACM International Conference on Distributed and Event-based Systems (DEBS22), Copenhagen, Denmark

Introduction

- **Stream processing** is used for real-time applications that deal with large volumes, velocities, and varieties of data.
- **Intermediate states** of computations might need to be **retained in memory** until a streaming application is complete.
- Memory **capacity** is **limited** in a worker/server node.
- Multiple parallel processes might **share primary memory**.
- As a result, a streaming application might **fail** to run or complete due to a **memory shortage**.
- Also, need support for **complex state representation**.



Existing Approaches

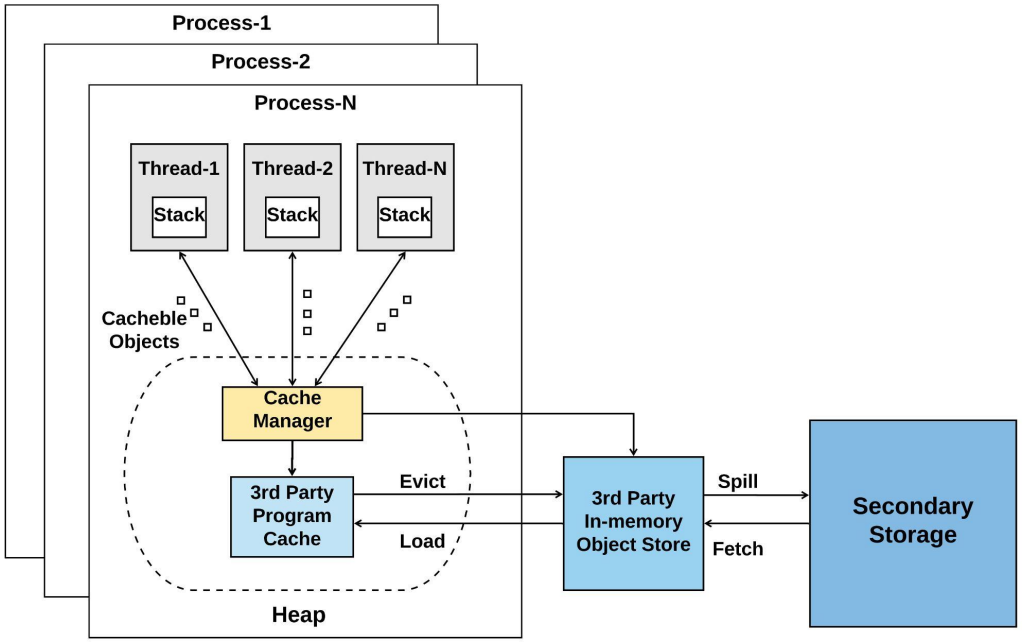
- **Spilling state** information to disk alleviates the problem by allowing the query to finish, but will cause significant performance overhead.
- State management through periodic **checkpointing** is **insufficient** for dealing with **dynamic state updates** in a real-time application.
- **In-memory** based **object store** showcases poor performance due to the **added communications** with the external object store.
- No support for **complex state representations** such as graphs, hashes, and trees that cannot be stored using a key-value mapping.

Contributions

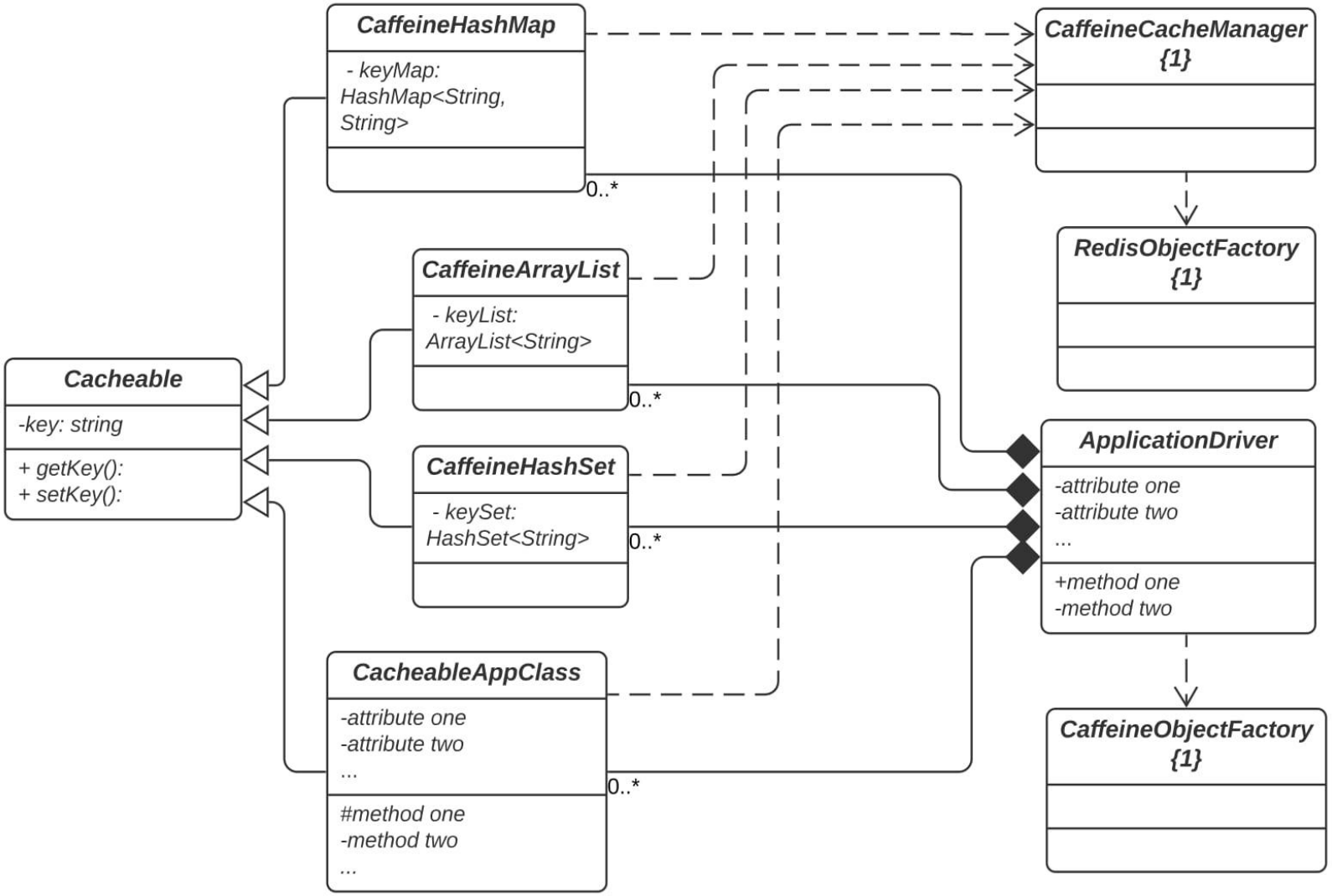
- **Scalable multi-level caching architecture** to support the state management of complex streaming applications.
- Prototype API in Java for the proposed multi-level caching architecture as a **caching library** for developing stateful stream applications that benefit from in-memory caching.
- Integrated various data structures into the caching API to represent **complex states**.
- The caching API manages the application state and cache levels **transparently**.
- Implemented **real-time** streaming and **synthetic** applications by utilizing the prototype API to showcase the performance benefits.

System Architecture

- Each process has its own **singleton** program cache, which is shared by multiple threads within that process.
- Different processes on the same host **share** the same in-memory object store and secondary storage.
- The cache manager provides a **transparent interface** to the application to enable caching support.
- 3rd party program cache (e.g., **Caffeine**) is used to **retain frequently accessed objects**.
- Objects are **evicted** (based on **cache policy**) to the object store when cache **memory limit** is reached.



Multi-level Caching Library (class diagram)

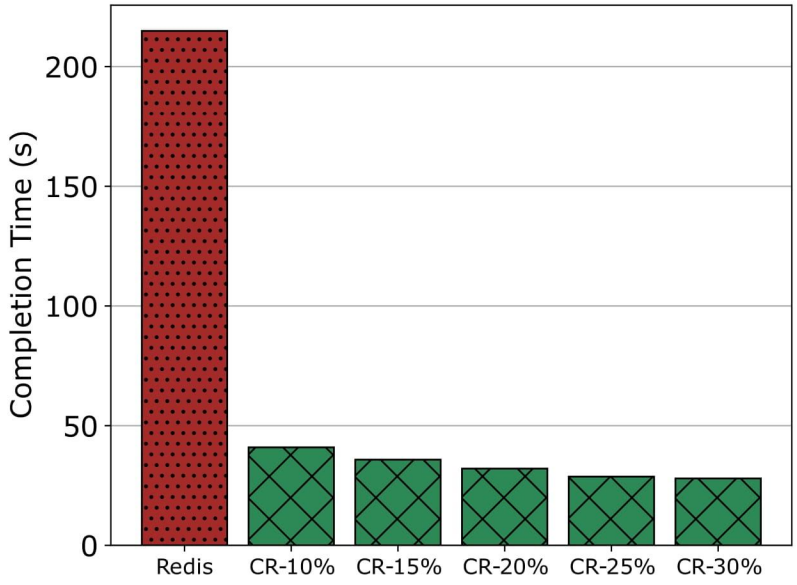


Open-source caching library: https://github.com/tawfiqul-islam/MemoryManagerJVMRedis

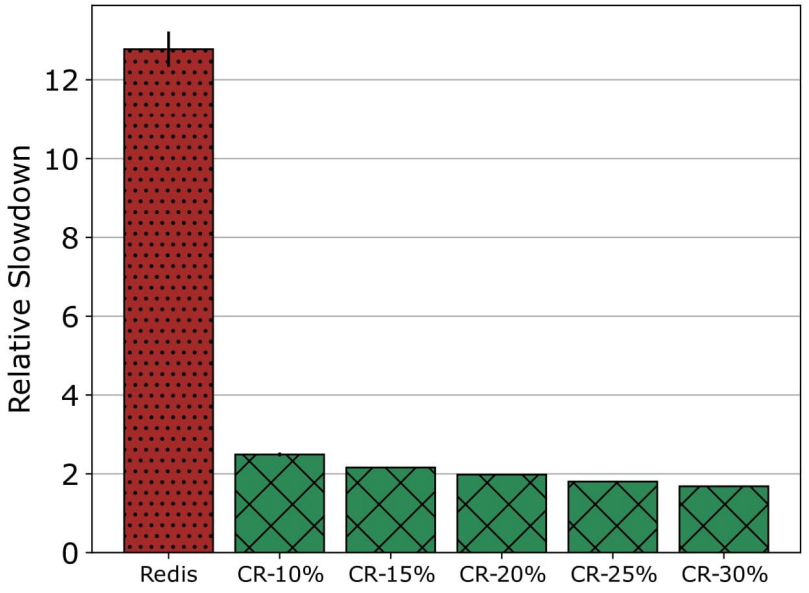
Performance Evaluation

- **Experiment Setup:** Integrated to **RAPID** (Real time Analytics Platform for Interactive Data Mining), deployed on **Nectar Research Cloud** Virtual Machines.
- **Benchmark Applications:**
  - **Synthetic** application
    - real-time objects (files) access and storage into JVM heap
    - Zipf distribution to generate object access patterns
  - **Real** application
    - spatio-temporal event detection algorithm
    - collects real-time data streams from Twitter
    - quadtree-based method to split the geographical space into multi-scale regions based on the density of social media data
    - unsupervised statistical approach is performed to identify regions with an unexpected density of social posts
    - update states (quadtree join, merge, and prune) in real-time
    - requires representing and updating **complex** computation state
- **State Management Approaches:**
  - JVM-based (native application)
  - Redis-only cache
  - Caffeine-Redis cache (proposed multi-level caching approach)

Application Performance - Synthetic

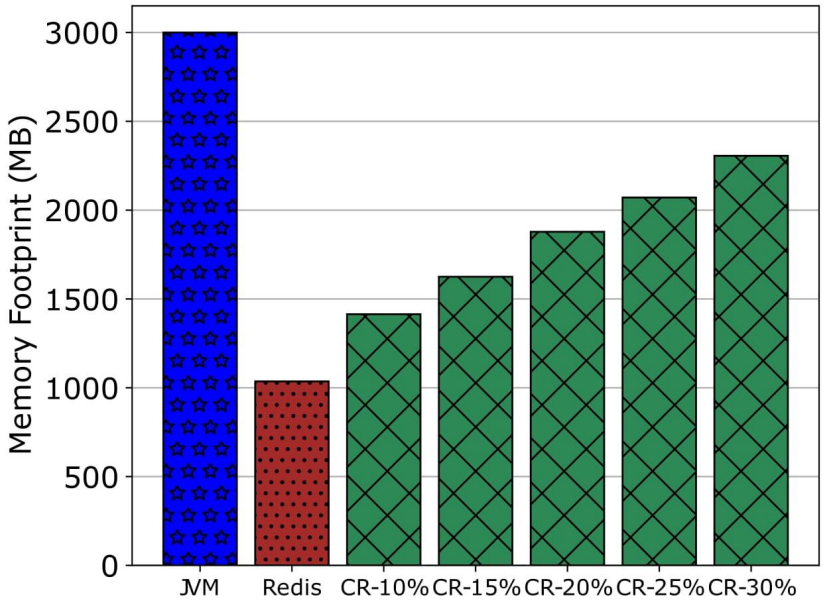


(a) Application Completion Time



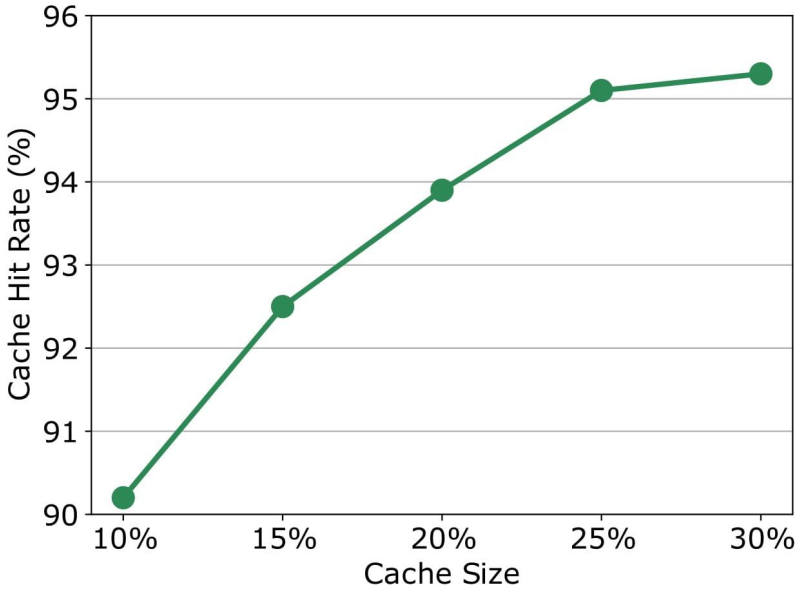
(b) Relative Performance (slowdowns as compared to the native JVM application)

Memory Footprint Analysis - Synthetic



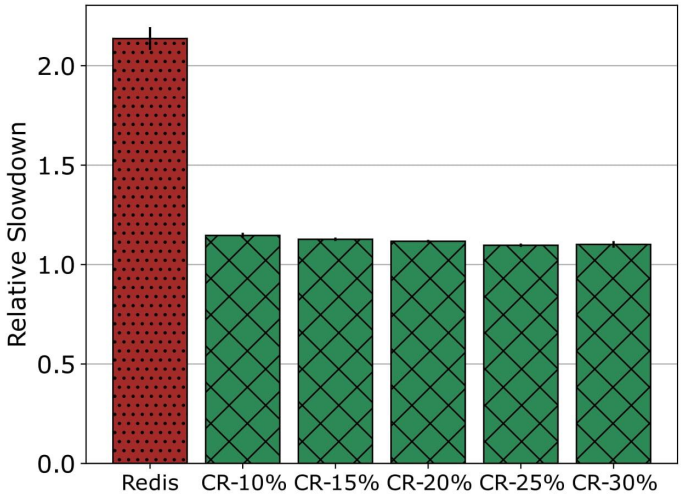
(a) Memory usages from each approach (JVM heap space is capped at 3GB for all the algorithms)

Effects of Cache Size - Synthetic

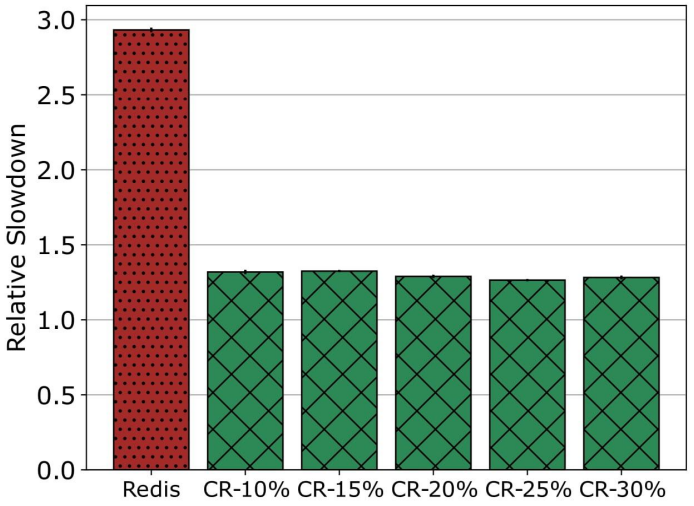


(a) Effects of cache size on cache hit rates

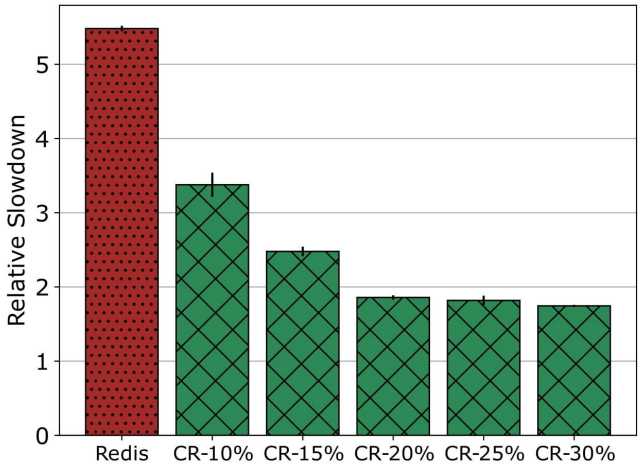
Application Performance - Real



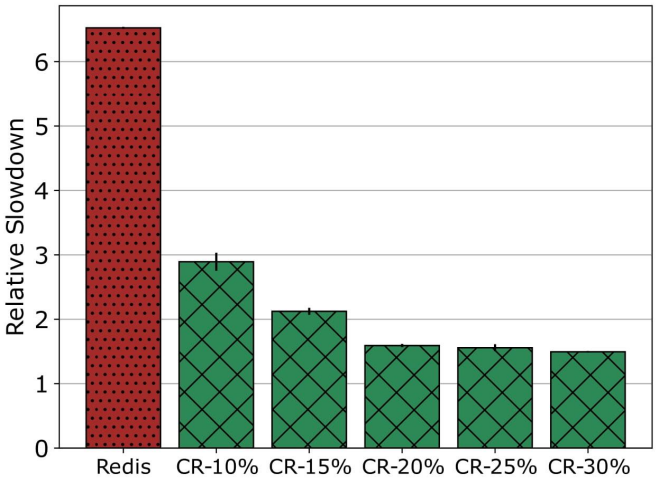
(a) Interval-1 (#Tweets = 120k)



(b) Interval-2 (#Tweets = 131k)

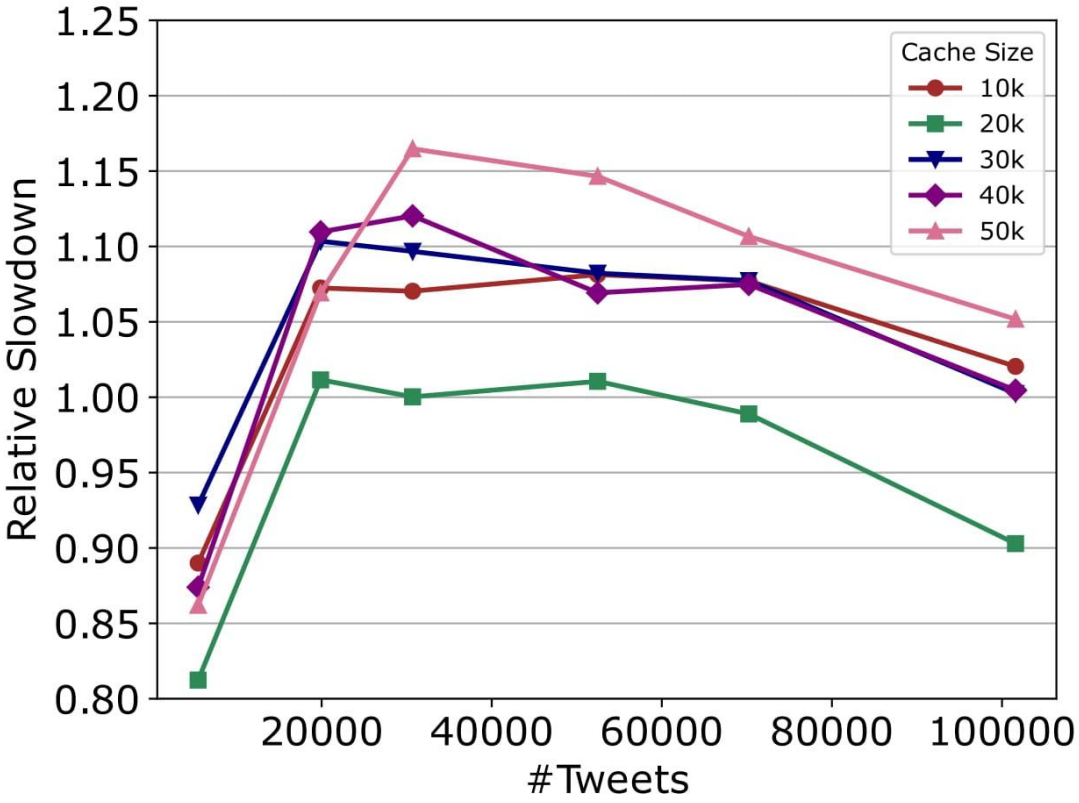


(c) Interval-3 (#Tweets = 157k)



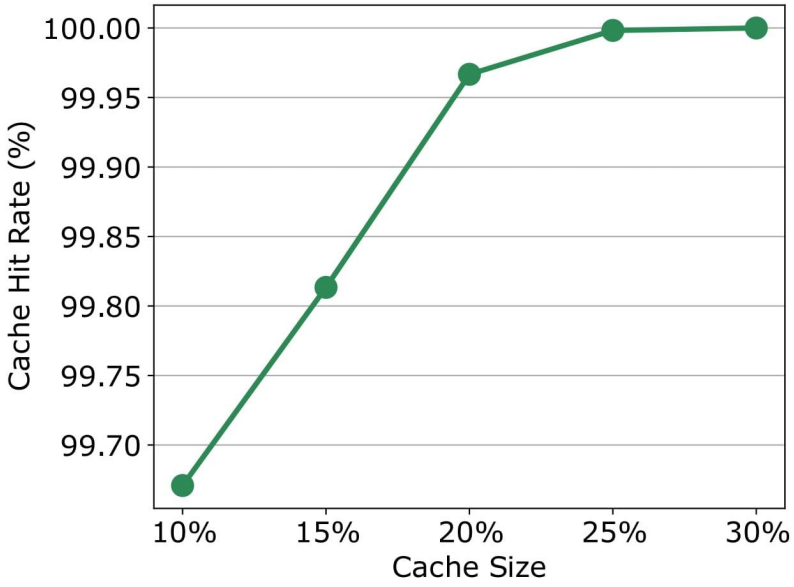
(d) Interval-4 (#Tweets = 171k)

Effects of Cache Overhead - Real

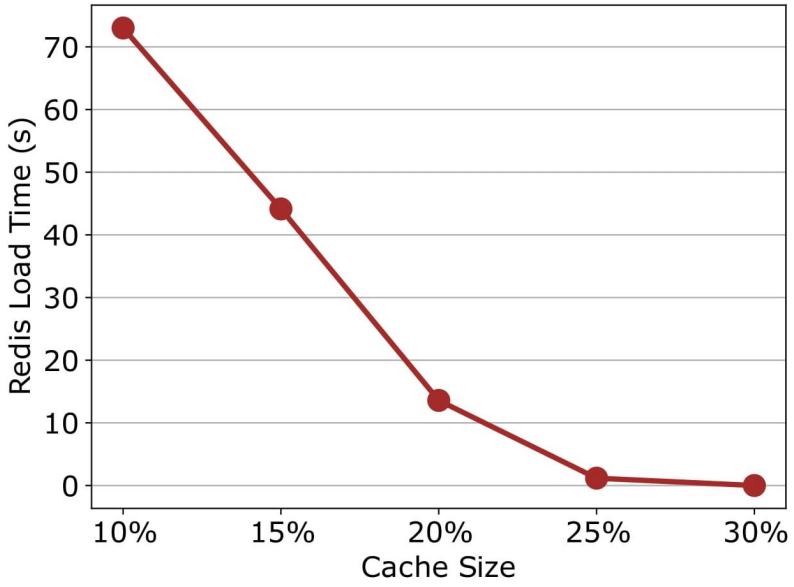


(a) Effects of 3rd party program cache overhead on application performance

Effects of Cache Size - Real



(a) Effects of cache size on cache hit rates



(a) Effects of cache size on Redis load times

Conclusions and Future Work

Conclusions:

- Memory limitations in servers for streaming applications is a critical problem
- Proposed a multi-level caching architecture to enable caching support for streaming applications
- Frequent objects are always kept at fastest (closest) level of the cache to boost application performance
- Slower memory levels are used to evict infrequent objects when cache limit is reached
- Support for representation of complex object states

Future Work:

- Devise new caching policies for the program cache
- Incorporate the caching architecture in emerging streaming applications
- Support for other program caches and in-memory object stores (e.g., memcached, Hazelcast)