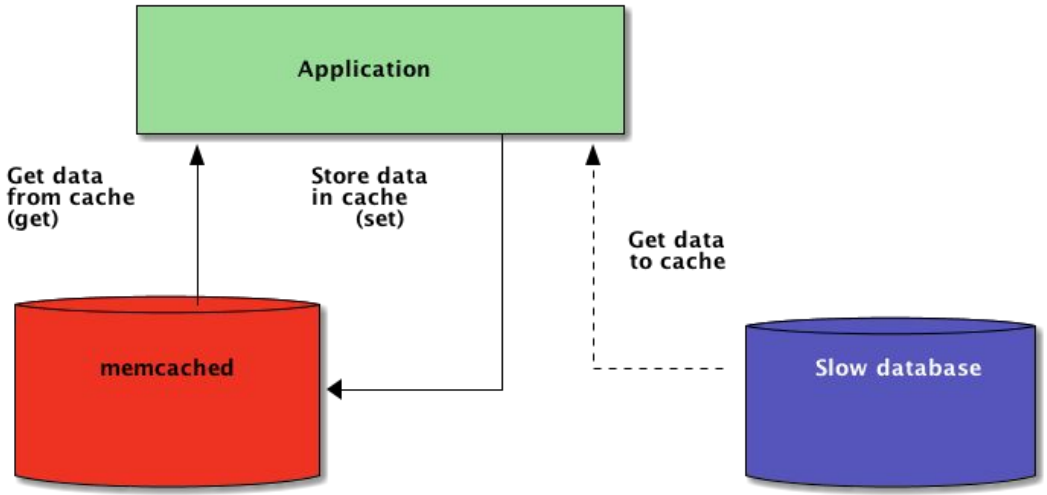
SVMemcached: Using an online classifier to dynamically amplify cache hits

Zach Cohen and Zoya Shoaib
COS 518 Project Interim presentation
17 April 2019

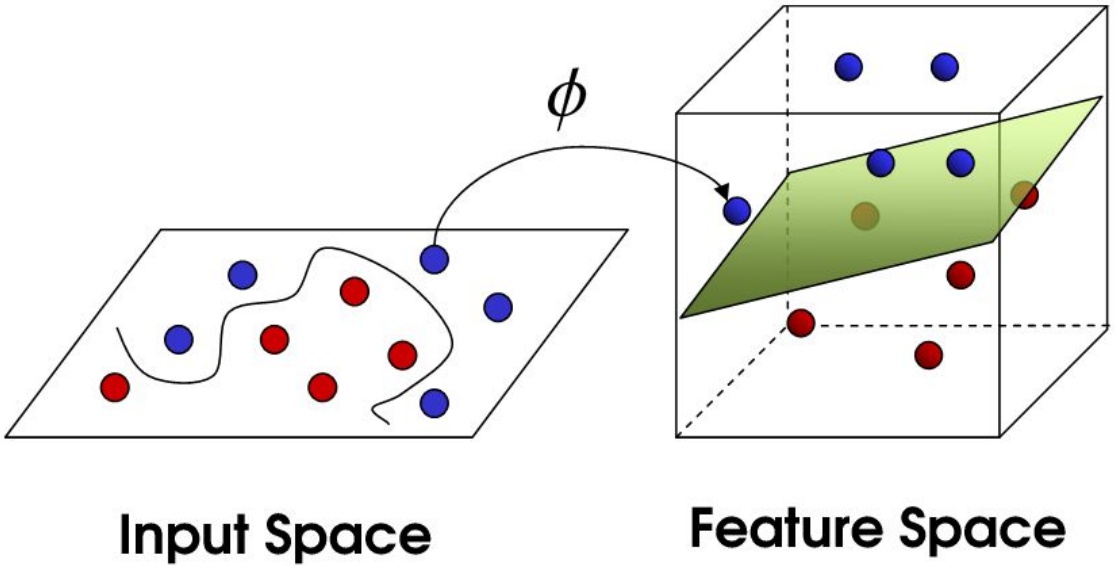
Background + problem statement

- Memcached keeps recently queried keys in a quickly retrievable cache
- Uses LRU caching algorithm
- When cache is full, eviction policy leaves recently used keys in cache
- LRU is a useful heuristic, but it may miss patterns in data that are unintuitive
- *Can we be smarter about what stays in cache?*



Key idea

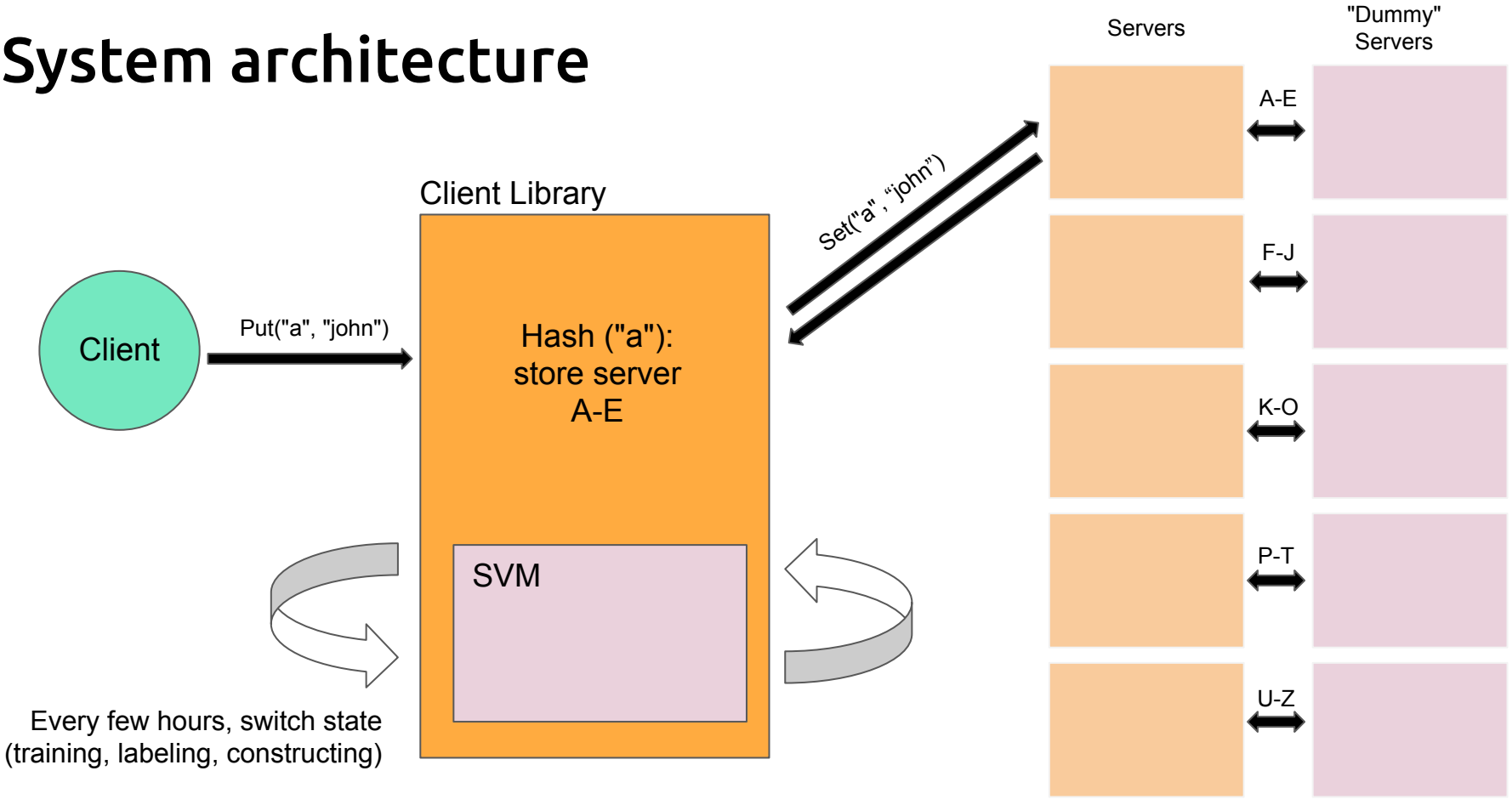
- Instead of assuming *a priori* structure over the data, try to learn a linear classifier (SVM) over the queries to label them as “should be in cache” and “shouldn’t be in cache” (think Flashfield)
- Classify using non-obvious and unintuitive patterns of data based on read patterns of the user



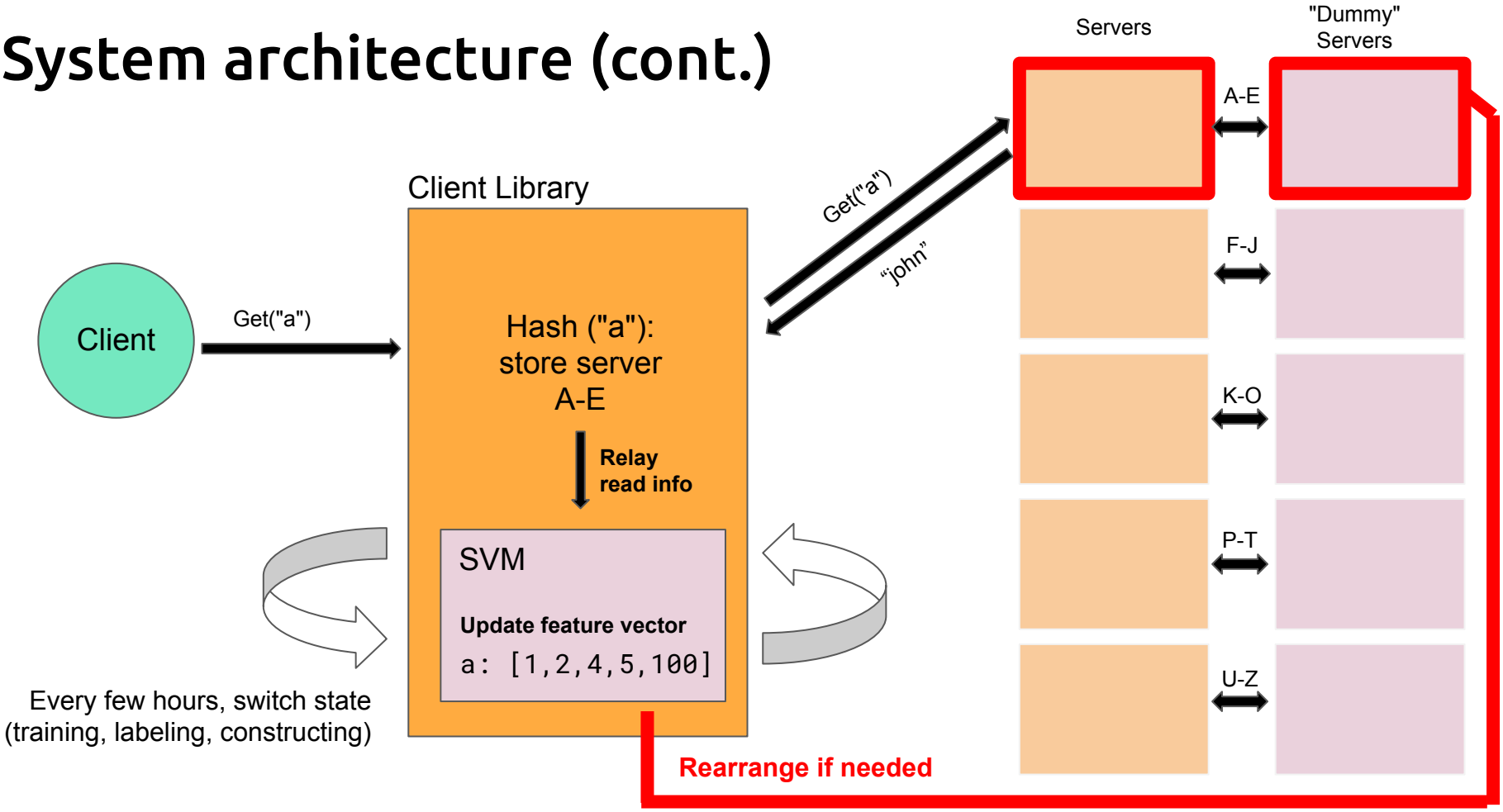
Key challenges

- How to make inference decisions online that don’t degrade throughput?
  - Rotate out a primary cache with a “dummy cache”— the client retrieves keys from the primary caches, and the SVM manages the dummy cache, switching them behind the scenes when appropriate (**detailed in next slide**)
- How to “stay alive” when classifier determines new cache configuration?
  - Buffer reads locally while cache swap takes place
- What data attributes help make effective partitions?
  - We opt to use the Flashfield ones:
    i. Number of past reads on the object
    ii. The average time between reads on an object
    iii. The time between the last two reads
    iv. The maximum time between two reads
    v. The time it took to read an object after its initial write
  - Training phase collects data, labeling phase tracks queries in the subsequent period

System architecture



System architecture (cont.)



Technical Details

- Use pymemcache client library paired with a memcache server binary
- Proof of concept model uses Unix domain sockets between memcached clients and servers running on parallel processes on the same computer
  - If we can get a local version working, we'll begin to explore using SVMemcached on different machines with TCP connections (likely running on several AWS instances)
- Client library spawns a new thread upon creation on which the SVM runs
  - `SVM_state = TRAIN || LABEL || CONFIG`
  - Transition between states on a cycle (configurable by user)
  - Attempting to build a predictive relationship between feature vectors built during training phase and labels produced in label phase (did the key get queried or not?)
  - In config phase, classifier is learned between data labeled "queried" and "not queried", dummy caches configured accordingly, and brought online by the SVM

Implementation status

- Successfully spawned and tested a simple memcache client with get and set requests
- Begun experimenting with various artificial read/write patterns (and building r/w patterns as well, like random reads, sequences of reads that prefer recent reads, etc.) to see how well the features we're using predict read patterns
- Configured multiple concurrent memcache servers that partition a simple key space
- Implemented the SVM in python and tested a few samples; we've begun the process of integrating it with the client library by starting it on its own execution thread and having it maintain read statistics for keys

Evaluation

- Will be running benchmarks comparing stock memcache to our SVMemcache
  - Specifically evaluating cache **hit rate** and **throughput** on the SVMemcache
  - In the short term, making an inference decision and swapping in and out caches may work to actually *increase* overall latency; since this is a proof of concept project, we're primarily looking at hit rate just to see that this works
- Tweak hyperparameters of the SVM to determine which ones generate highest hit rate
  - These involve adjusting how much we penalize a hyperplane that can't cleanly separate two label classes (if they aren't linearly separable)
- Track **number of forced evictions** done by the online memcache (which is still running its own LRU replacement policy)
  - The idea here is that we should see evictions go down significantly after several training/labeling/config phases

Plan for final month

- Finish system for swapping the "dummy" and online servers
  - Flush all the keys in the "dummy" cache
  - Send multiple set requests from "dummy" servers to the memcache servers
- Get a toy example working
- Evaluate SVMemcache's hit rate and throughput with the new design
  - Memcache server API allows for querying hit rate of keys, so we'll periodically check that
- Compare SVMemcache's performance with that of memcache
  - Vary the hyperparameters of the SVM to see how they affect the results