

Distributed Systems

30. Distributed Caching

Paul Krzyzanowski
Rutgers University
Fall 2015

December 6, 2015

© 2013-2015 Paul Krzyzanowski

1

Caching

- **Purpose of a cache**
 - Temporary storage to increase data access speeds
 - Increase effective bandwidth by caching most frequently used data
- **Raw data from slow devices**
 - Memory cache on CPUs
 - Buffer cache in operating system
 - Chubby file data and metadata
 - GFS master caches all metadata in memory
- **Computed data**
 - Results of database queries or file searches
 - Avoid the need to look the same thing up again

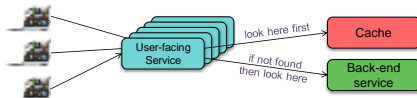
December 6, 2015

© 2013-2015 Paul Krzyzanowski

2

Distributed In-Memory Caching

- A network memory-based caching service
 - Shared by many – typically used by front-end services
- Stores frequently-used (key, value) data
 - Old data gets evicted
- General purpose
 - Not tied to a specific back-end service
- Not transparent (usually)
 - Because it's a general-purpose service, the programmer gets involved



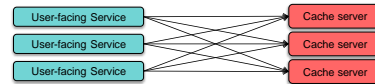
December 6, 2015

© 2013-2015 Paul Krzyzanowski

3

Deployment Models

- **Separate caching server**
 - One or more computers whose sole purpose is to provide a caching service
- **Or share cache memory among servers**
 - Take advantage of free memory from lightly-loaded nodes



December 6, 2015

© 2013-2015 Paul Krzyzanowski

4

What would you use it for?

- **Cache user session state** on web application servers
 - No need to keep user coming back to the same computer
- **Cache user preferences, shopping carts, etc.**
 - Avoid repeated database lookup
- **Cache rendered HTML pages**
 - Avoid server-side includes, JSP/ASP/PHP code

December 6, 2015

© 2013-2015 Paul Krzyzanowski

5

Example: memcached

- Free & open source distributed memory caching
- Used by
 - Facebook, Wikipedia, Flickr, Twitter, YouTube, Digg, Bebo, WordPress, Craigslist, ...
- Protocol
 - Binary & ASCII versions
- Client APIs for
 - command line, C/C++, Go, PHP, Java, Python, Ruby, Perl, Windows/.NET, MySQL, PostgreSQL, Erlang, Lua, LISP, ColdFusion, io



memcached.org

December 6, 2015

© 2013-2015 Paul Krzyzanowski

6

Example: memcached

- Key-Value store
 - Cache is made up of { *key, value, expiration time, flags* }
 - All access is $O(1)$
- Client software
 - Provided with a list of *memcached* servers
 - Hashing algorithm: chooses a server based on the *key*
- Server software
 - Stores keys and values in an in-memory hash table
 - Throw out old data when necessary
 - LRU cache and time-based expiration
 - Objects expire after a minute to ensure stale data is not returned
 - Servers are unaware of each other

December 6, 2015

© 2013-2015 Paul Krzyzanowski

7

Memcached API

- Commands sent over TCP (UDP also available)
 - Connection may be kept open indefinitely.
- Commands
 - **Storage**
 - Storage commands take an expiration time in seconds from current time or 0 = forever (but may be deleted)
 - **set** – store data
 - **add** – store data only if the server *does not* have data for the key
 - **replace** – store data if the server *does* have data for the key
 - **append** – add data after existing data
 - **prepend** – add data before existing data
 - **cas** – check & set: *store data only if no one else updated it since I fetched it* (cas = unique, 64-bit value associated with the item)
 - **Retrieval**
 - **get** – retrieve one or more keys: returns *key, flags, bytes, and cas unique*

December 6, 2015

© 2013-2015 Paul Krzyzanowski

8

Memcached API

- Commands
 - **Deletion**
 - **delete** *key*
 - **Increment/decrement**
 - Treat data as a 64-bit unsigned integer and add/subtract value
 - **incr** *key value* – increment *key* by *value*
 - **decr** *key value* – decrement *key* by *value*
 - **Update expiration**
 - **touch** *key exptime* – Update the expiration time
 - **Get Statistics**
 - **stats** – various options for reporting statistics
 - **Flush**
 - **flush_all** – clear the cache

December 6, 2015

© 2013-2015 Paul Krzyzanowski

9

Another example: Redis

Memory cache + in-memory database + message broker

- Open source: see redis.io
- Text-based command interface
- Features
 - Key-value store
 - Transactions
 - Publish/subscribe messaging
 - Expiration of data
 - Built-in replication
 - Optional disk persistence
 - Lua scripting (via EVAL command)
 - Automatic partitioning with Redis Cluster
- Used by
 - Twitter, GitHub, Weibo, Pinterest, Snapchat, Craigslist, Digg, StackOverflow, Flickr



December 6, 2015

© 2013-2015 Paul Krzyzanowski

10

Redis Data Types

- **Strings**
 - Simplest type; only type supported in memcached)
- **Lists**
 - Collections of strings sorted by order of insertion
- **Sets**
 - Collections of unique, unsorted strings
- **Sorted sets**
 - Every element is associated with a **score** (floating point number)
 - Elements sorted by score
 - Operations to retrieve ranges (e.g., top 10, bottom 10)
- **Hashes**
 - Maps of fields associated with values (fields & values are strings)
- **Bitmaps**
 - Commands to treat strings as bits (set/clear bits)
- **HyperLogLogs**
 - Probabilistic data structure to estimate the cardinality of a set
 - Count # of unique items without storing the entire set of items
 - Use a fixed amount of memory

December 6, 2015

© 2013-2015 Paul Krzyzanowski

11

Redis as a memory cache

- Set expiration for specific keys
 - Associate a timeout with a key
 - Key deleted after the timeout


```
SET mykey "hello"
EXPIRE mykey 10
```

expire key in 10 seconds!
- Tell it to automatically evict (delete) old data
- Methods of eviction
 - LRU (least recently used)
 - LRU only for keys that have an expiration time
 - Random
 - Random only for keys that have an expiration time

December 6, 2015

© 2013-2015 Paul Krzyzanowski

12

Redis as an in-memory database

- **EXEC**
 - Execute queued commands in a transaction
- **MULTI**
 - Mark the start of a transaction (operations queued until EXEC)
- **DISCARD**
 - Abort transaction & revert to previous values
- **WATCH**
 - Check-and-set behavior to ensure mutual exclusion
 - Monitor keys to detect changes
 - Abort if change takes place

December 6, 2015

© 2013-2015 Paul Krzyzanowski

13

Redis as a message broker

- Publish/subscribe model
 - Senders (publishers) do not send messages to specific receivers
 - Messages go to channels
 - Subscribers listen to one or more channels, receiving messages of interest
- Allows for scalability and dynamic topology
 - Publishers do not know subscribers
 - Subscribers do not know publishers
- Support for pattern-based channels
 - Subscribe to all channel names matching a pattern

December 6, 2015

© 2013-2015 Paul Krzyzanowski

14

Redis partitioning

- Data can be partitioned across multiple computers
- Types
 - Range partitioning
 - Use table that maps ranges to instances
 - Hash partitioning
 - Based on hash(key): works with any key
- Who does the partitioning?
 - Client-side partitioning
 - Proxy-assisted partitioning
 - Query forwarding

December 6, 2015

© 2013-2015 Paul Krzyzanowski

15

The End

December 6, 2015

© 2013-2015 Paul Krzyzanowski

16