MicroFuge: A Middleware Approach to Providing Performance Isolation in Cloud Storage Systems

Akshay Singh, **Xu Cui**, Benjamin Cassell, Bernard Wong and Khuzaima Daudjee

UNIVERSITY OF WATERLOO

June 25, 2014

Storage Resources in Cloud Datacenters

- Cloud computing allows sharing of resource at the cost of reduced isolation.
- Storage systems are highly sensitive to performance interference.
- Lack of performance isolation → Unpredictable latencies.

A Cloud Scenario

- In worst case, a particular HTTP request may require 35 database lookups.[1]
  - Response time can add up quickly.
- Amazon reported 100ms of latency cost them 1% in sales.[2]
- Google found an extra .5 seconds delay caused 20% drop in search traffic.[2]

– [1] Nathan Farrington and Alexey Andreyev, Facebook's Data Center Network Architecture.

– [2] Greg Lindem, Make Data Useful, `http://www.scribd.com/doc/4970486/Make-Data-Useful-by-Greg-Linden-Amazon-com`.

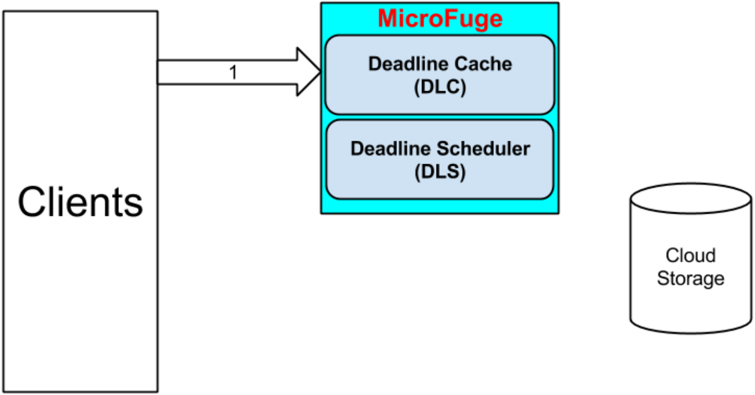
Performance Isolation

- Clients want to have performance guarantees in the shared environment.
- Possible solutions to performance isolation.
    - Dedicated resources.
    - Meet clients' response time requirements in the shared environment.
        - We represent response time requirements with **request deadlines**.
    - Meeting request deadlines → Performance isolation.

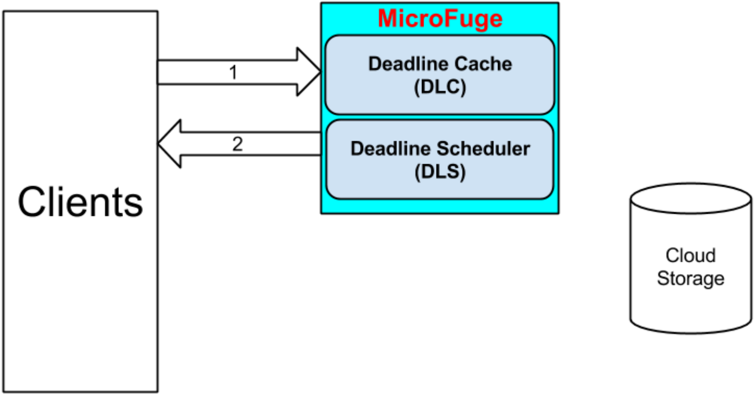
MicroFuge

- A distributed caching and scheduling middleware that provides performance isolation.
  - **Deadline Cache (DLC)**
    - Builds a performance model of the system.
    - Uses multiple LRU queues for deadline-aware eviction.
  - **Deadline Scheduler (DLS)**
    - Performs intelligent replica selection.
    - Implements feedback-driven deadline-aware scheduling.
    - Optionally performs admission control.
- Middleware: supports different cloud storage systems.

MicroFuge Overview I



MicroFuge Overview II



Clients

1

2

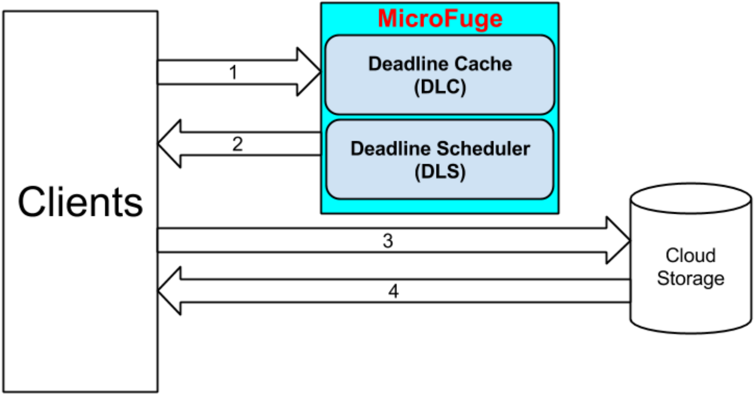
MicroFuge

Deadline Cache (DLC)

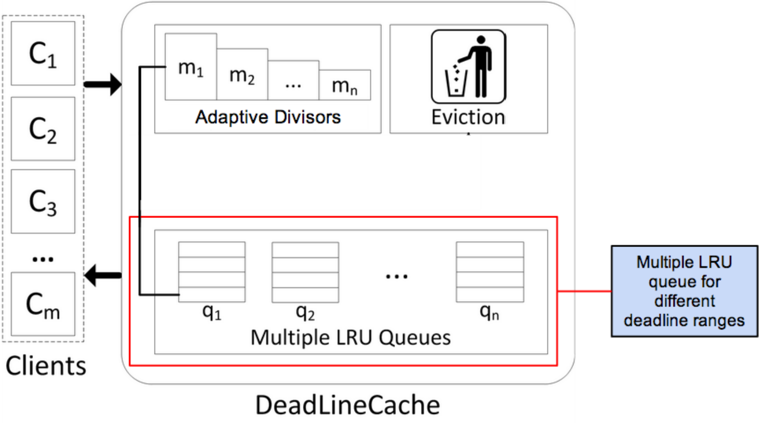
Deadline Scheduler (DLS)

Cloud Storage

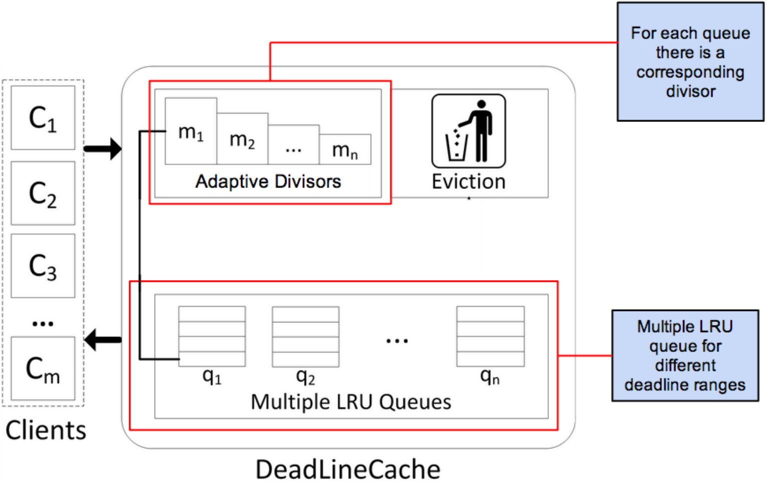
MicroFuge Overview III



Deadline Cache (DLC) - Components



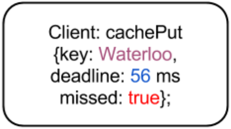
Deadline Cache (DLC) - Components



For each queue there is a corresponding divisor

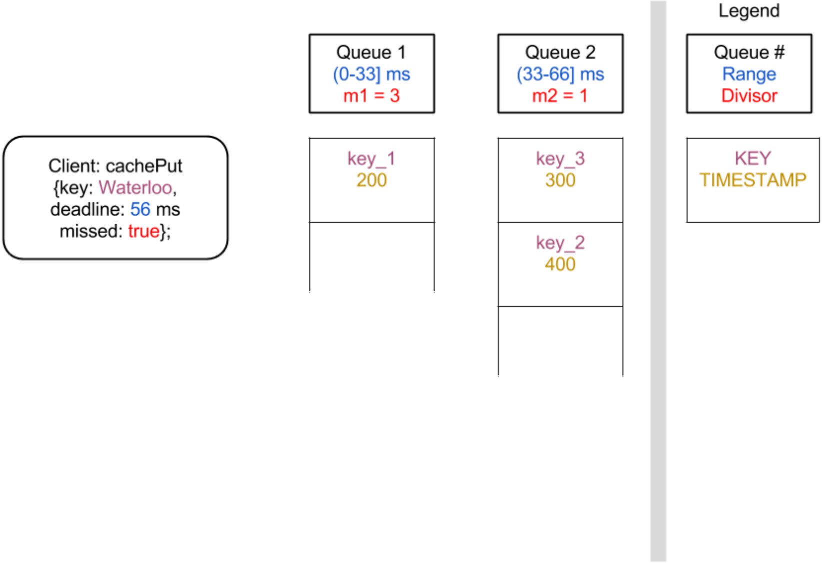
Multiple LRU queue for different deadline ranges

DLC - A Cache Eviction Example (1)



Client: cachePut
{key: Waterloo,
deadline: 56 ms
missed: true};

DLC - A Cache Eviction Example (2)



Client: cachePut
{key: Waterloo,
deadline: 56 ms
missed: true};

Queue 1
(0-33] ms
m1 = 3

key_1
200

Queue 2
(33-66] ms
m2 = 1

key_3
300

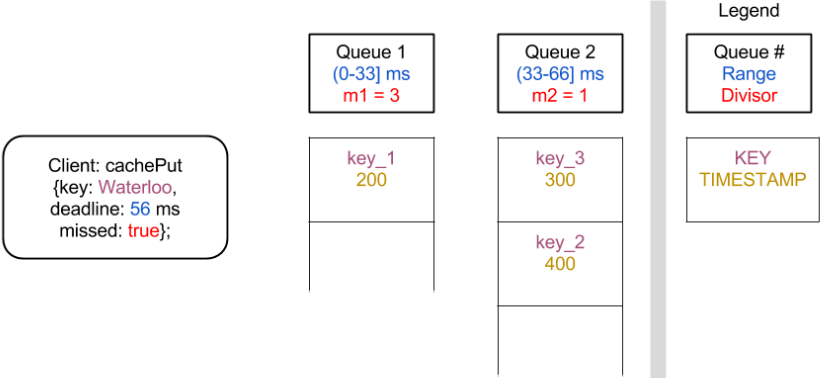
key_2
400

Legend

Queue #
Range
Divisor

KEY
TIMESTAMP

DLC - A Cache Eviction Example (3)



Client: cachePut {key: Waterloo, deadline: 56 ms missed: true};

Queue 1
(0-33] ms
m1 = 3

key_1
200

Queue 2
(33-66] ms
m2 = 1

key_3
300

key_2
400

Legend

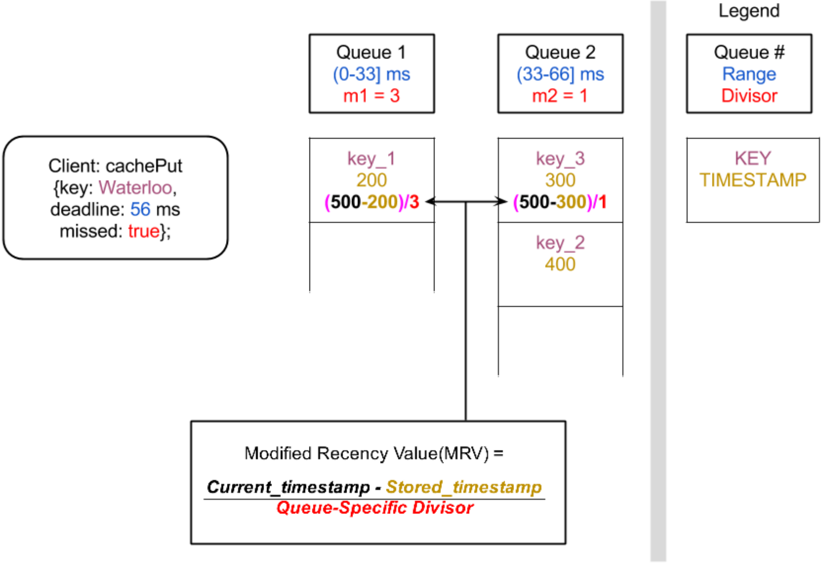
Queue #
Range
Divisor

KEY
TIMESTAMP

Modified Recency Value(MRV) =

$$\frac{\textit{Current\_timestamp} - \textit{Stored\_timestamp}}{\textit{Queue-Specific Divisor}}$$

DLC - A Cache Eviction Example (4)

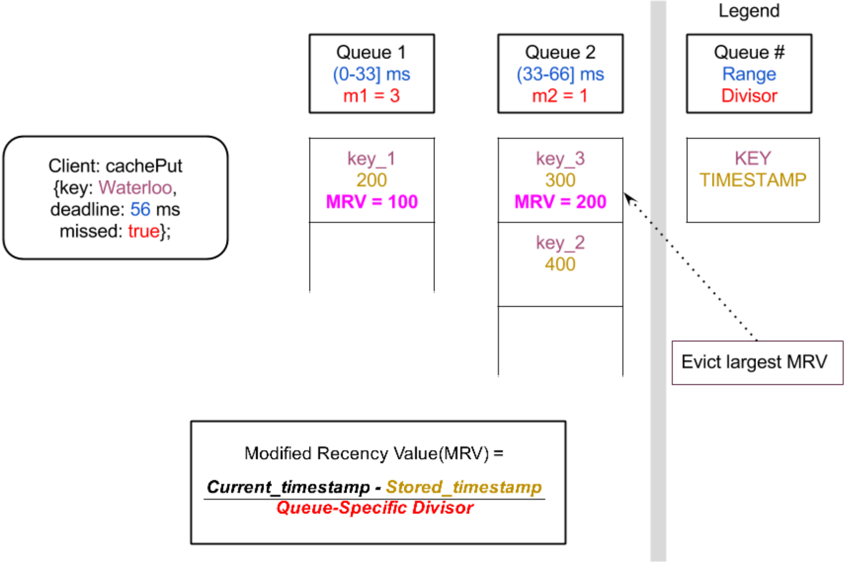


Client: cachePut
{key: Waterloo,
deadline: 56 ms
missed: true};

Modified Recency Value(MRV) =

$$\frac{\textit{Current\_timestamp} - \textit{Stored\_timestamp}}{\textit{Queue-Specific Divisor}}$$

DLC - A Cache Eviction Example (5)



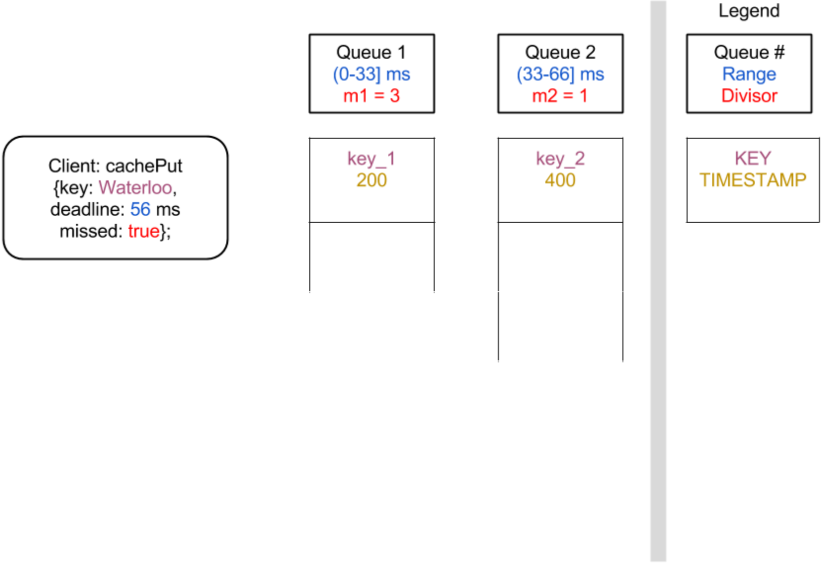
Client: cachePut {key: Waterloo, deadline: 56 ms missed: true};

Queue 1 (0-33] ms m1 = 3	Queue 2 (33-66] ms m2 = 1	Queue # Range Divisor
key_1 200 **MRV = 100**	key_3 300 **MRV = 200**	KEY TIMESTAMP
	key_2 400	

Modified Recency Value(MRV) =

$$\frac{\textit{Current\_timestamp} - \textit{Stored\_timestamp}}{\textit{Queue-Specific Divisor}}$$

DLC - A Cache Eviction Example (6)



Client: cachePut
{key: Waterloo,
deadline: 56 ms
missed: true};

Queue 1
(0-33] ms
m1 = 3

key_1
200

Queue 2
(33-66] ms
m2 = 1

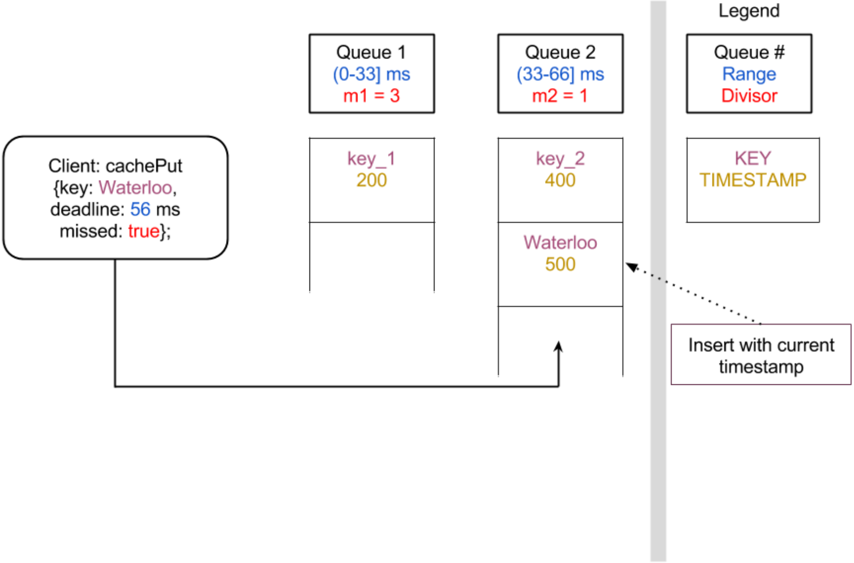
key_2
400

Legend

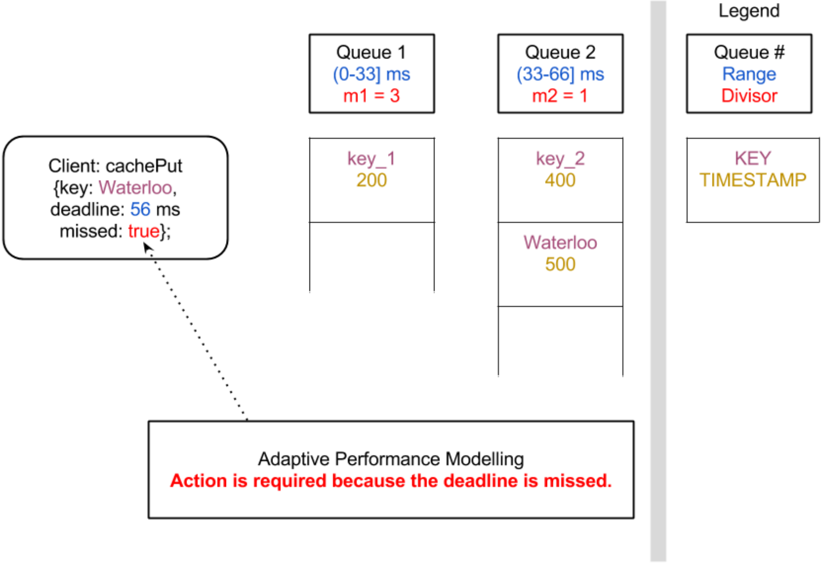
Queue #
Range
Divisor

KEY
TIMESTAMP

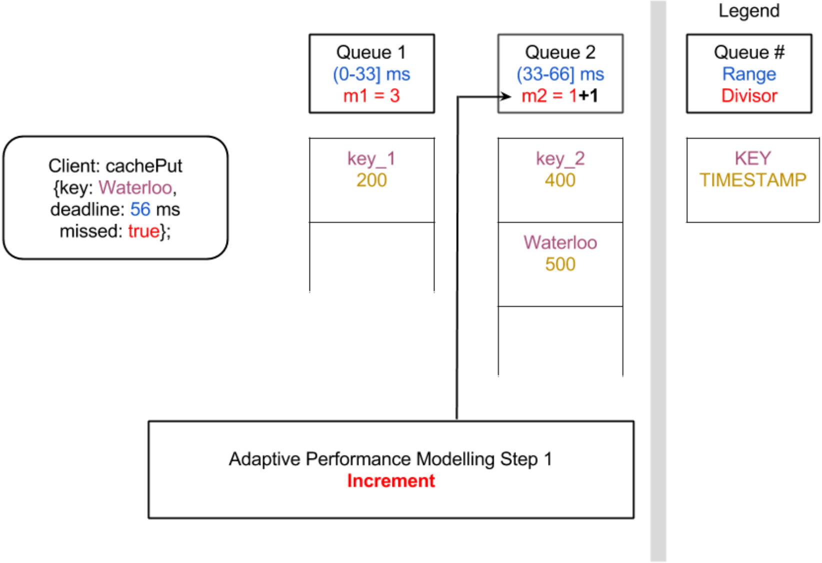
DLC - A Cache Eviction Example (7)



DLC - A Cache Eviction Example (8)



DLC - A Cache Eviction Example (9)



Legend

Queue #
Range
Divisor

KEY
TIMESTAMP

Client: cachePut
{key: Waterloo,
deadline: 56 ms
missed: true};

Queue 1
(0-33] ms
m1 = 3

key_1
200

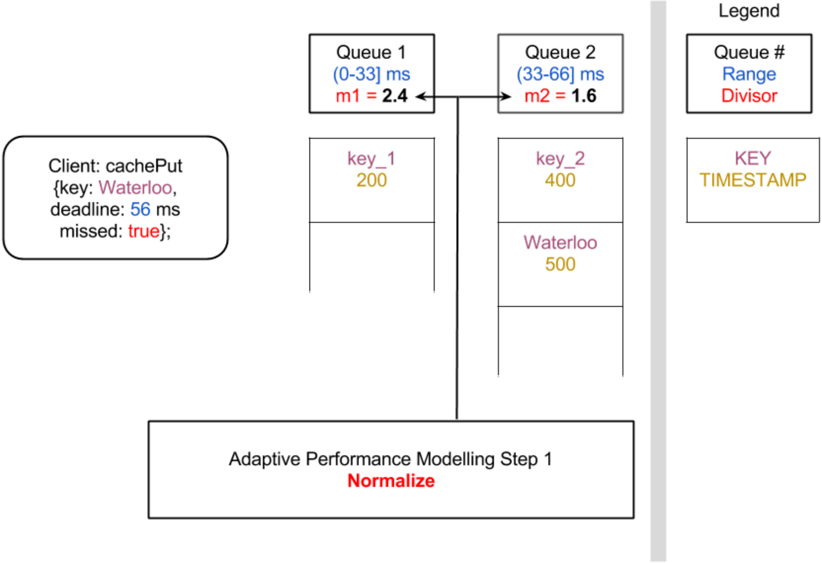
Queue 2
(33-66] ms
m2 = 1**+1**

key_2
400

Waterloo
500

Adaptive Performance Modelling Step 1
Increment

DLC - A Cache Eviction Example (10)



Client: cachePut
{key: Waterloo,
deadline: 56 ms
missed: true};

Queue 1
(0-33] ms
m1 = **2.4**

Queue 2
(33-66] ms
m2 = **1.6**

key_1
200

key_2
400

Waterloo
500

Legend

Queue #
Range
Divisor

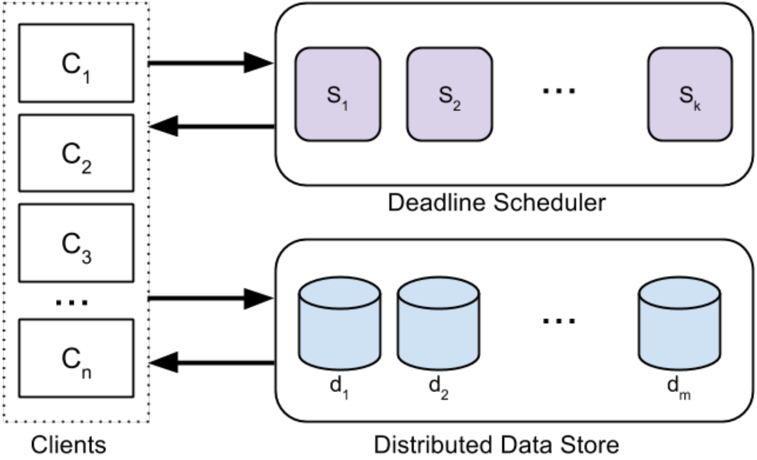
KEY
TIMESTAMP

Adaptive Performance Modelling Step 1
Normalize

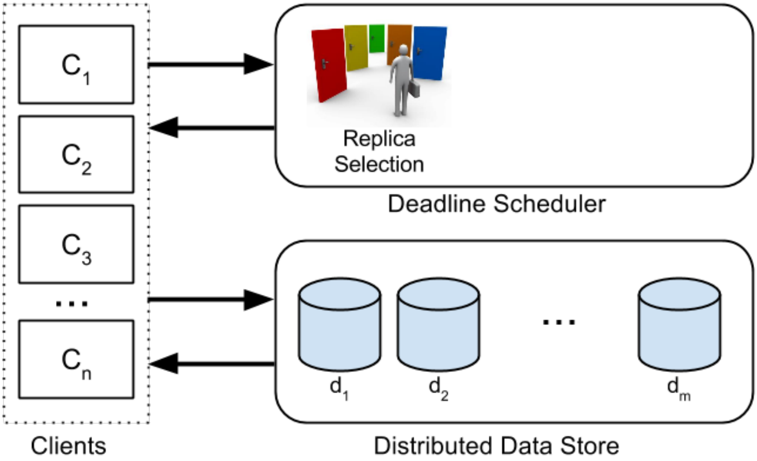
DLC - Benefits

- Multiple LRU queues enable DLC to perform deadline-aware evictions.
- Adaptive policy considers both the client request rate for each deadline range and the underlying system's performance.
- **DLC** offers adaptive deadline-aware caching.

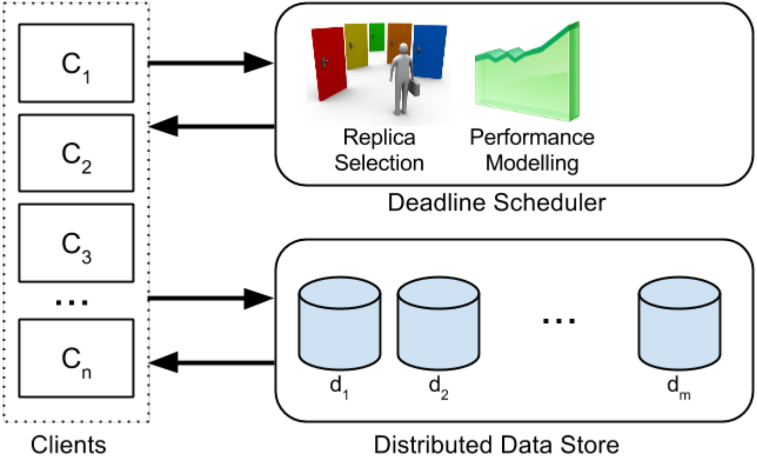
Deadline Scheduler (DLS) High-level Architecture I



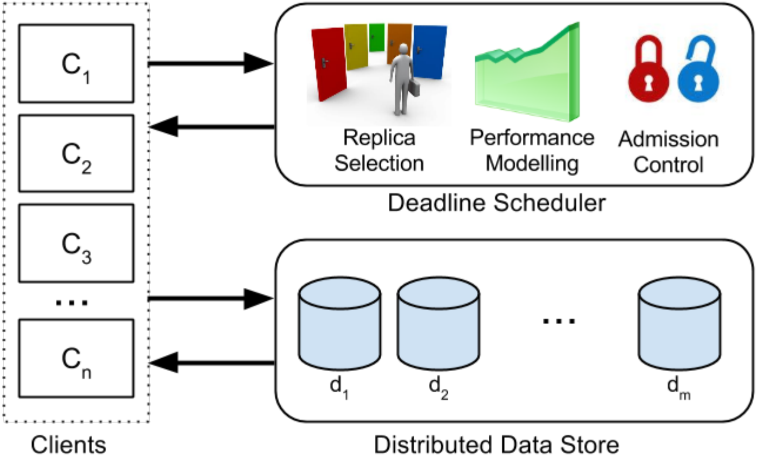
Deadline Scheduler (DLS) High-level Architecture II



Deadline Scheduler (DLS) High-level Architecture III

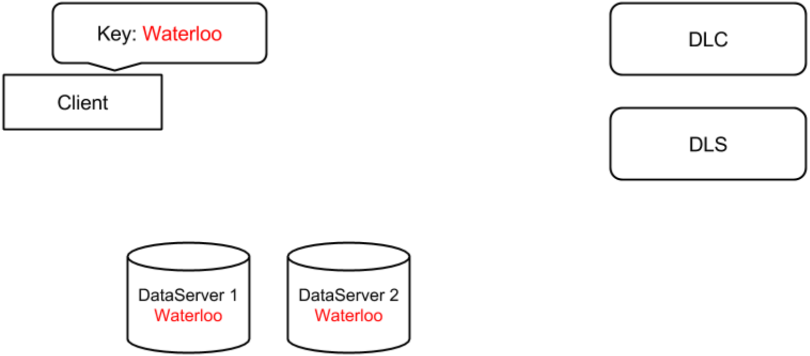


Deadline Scheduler (DLS) High-level Architecture IV



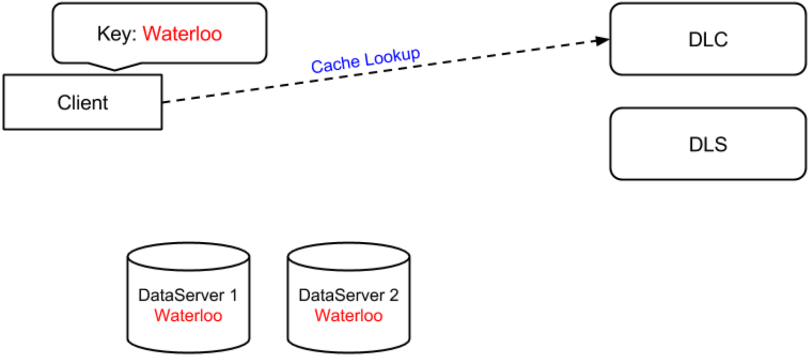
DLS - An Example (1)

- The client wants to perform a value lookup for the key Waterloo.



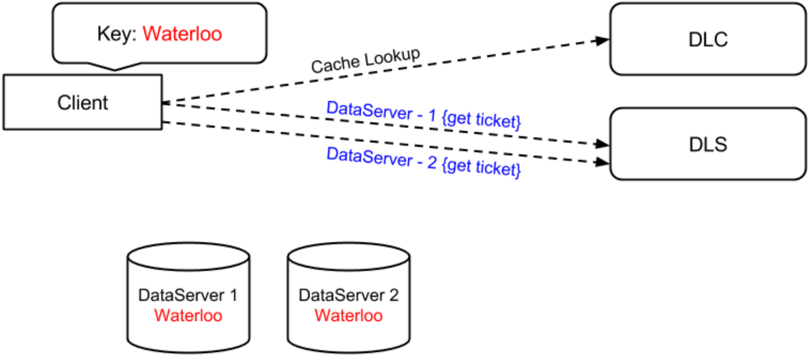
DLS - An Example (2)

- The client begins by issuing a cache lookup to DLC.



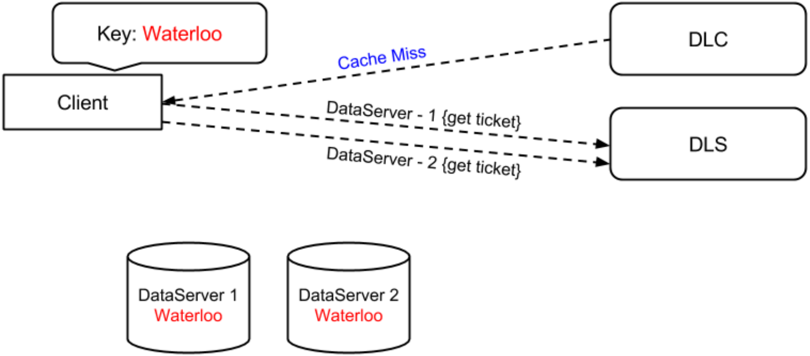
DLS - An Example (3)

- Issue two *get ticket* requests concurrently.



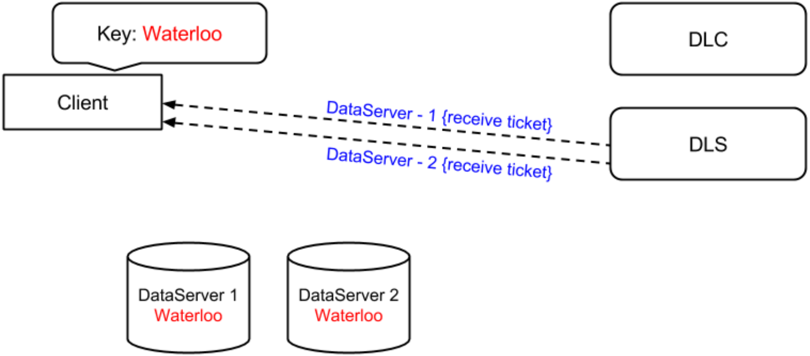
DLS - An Example (4)

- If the item is not in the cache, the client waits for DLS to return the tickets.



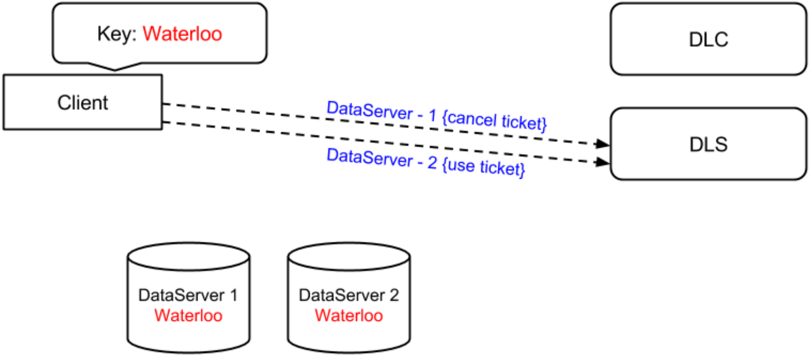
DLS - An Example (5)

- Returned tickets contain extra information to help the client to make an informed decision.



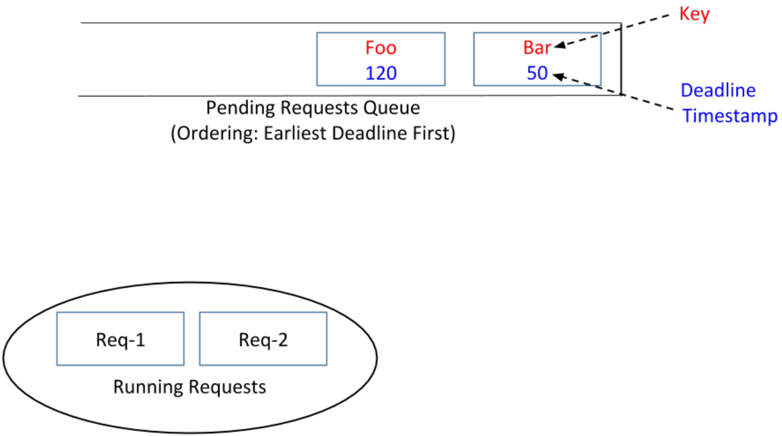
DLS - An Example (6)

- The client makes a blocking call to the selected DLS and waits for its turn to access the data server.



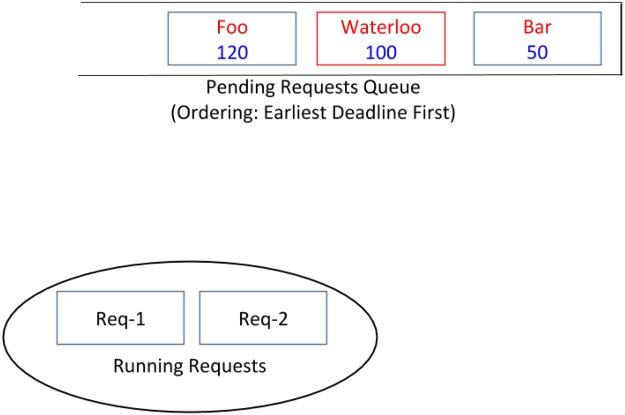
DLS - An Example (7)

- A snapshot of scheduler's pending queue.



DLS - An Example (8)

- The new item is inserted according to earliest deadline first ordering.



Foo
120

Waterloo
100

Bar
50

Pending Requests Queue
(Ordering: Earliest Deadline First)

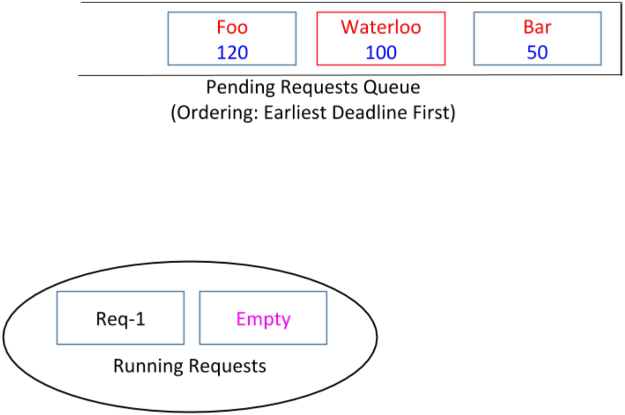
Req-1

Req-2

Running Requests

DLS - An Example (9)

- Let's assume one of the running requests just completed.



Foo 120

Waterloo 100

Bar 50

Pending Requests Queue
(Ordering: Earliest Deadline First)

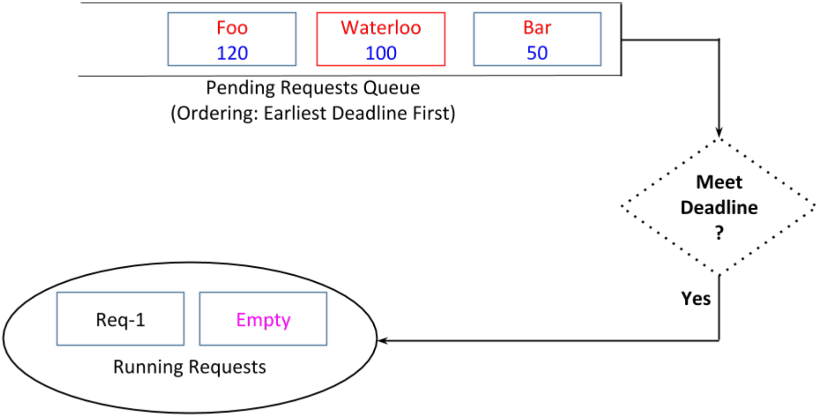
Req-1

Empty

Running Requests

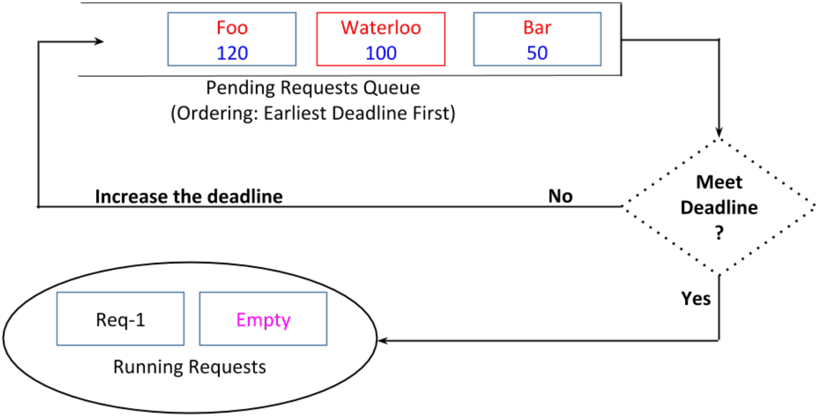
DLS - An Example (10)

- If the request deadline can be met, it will take one of the empty slots inside the running request pool.



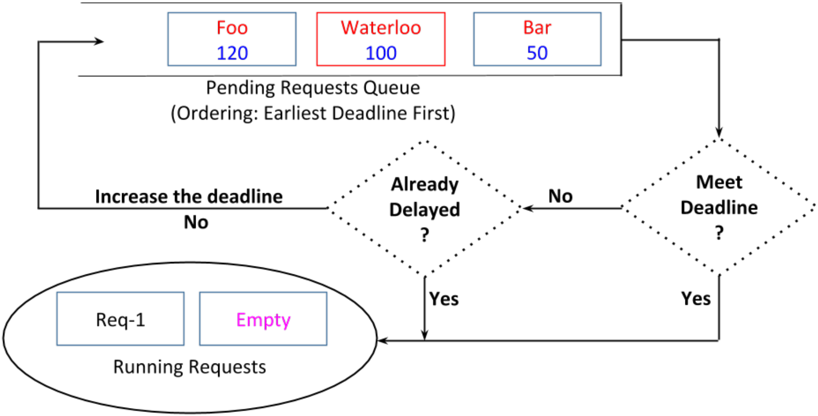
DLS - An Example (11)

- If request deadline cannot be met, DLS may increase the request's deadline and insert the request back into the queue.



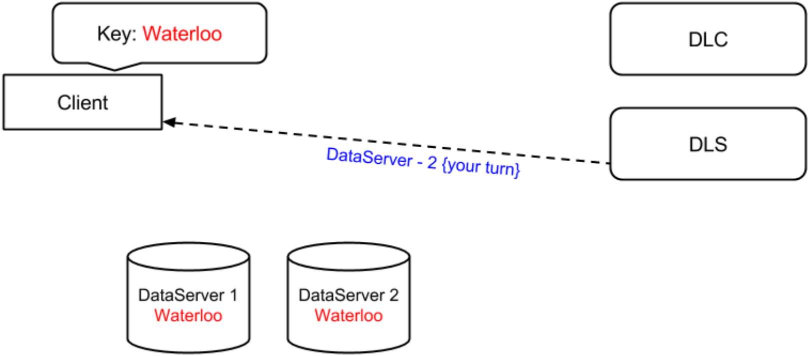
DLS - An Example (12)

- The push-back can happen at most once to prevent starvation.



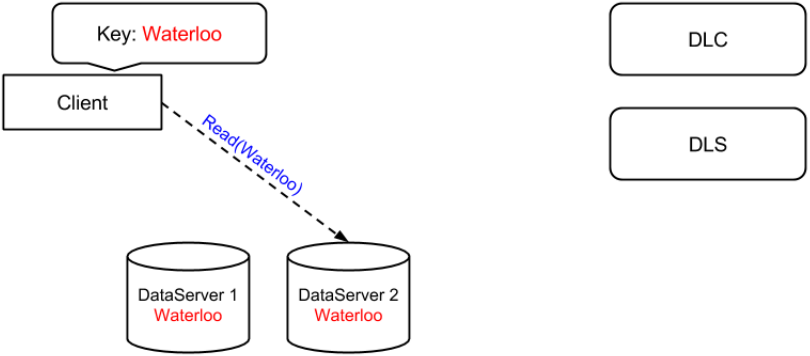
DLS - An Example (13)

- DLS informs the client that it can access the data server.



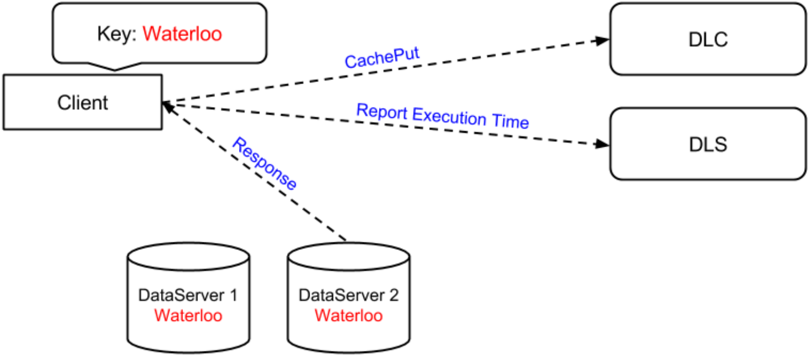
DLS - An Example (14)

- The client issues the read request to the data server.



DLS - An Example (15)

- After receiving the response, the client reports the execution time and concurrently inserts the data into the cache.

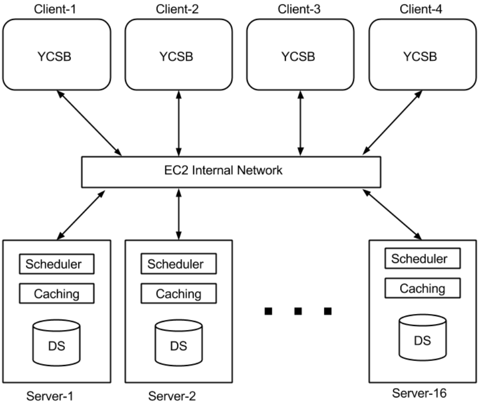


DLS - Benefits

- Deadline-aware load-balancing.
- A variant of earliest deadline first scheduling.
- Tunable admission control system.

Experimental Setup - The Cluster

- Twenty-node test cluster on AWS. Each cluster node is an m1.medium EC2 instance.



Experimental Setup - Details

- DataServer - Simple key-value store that uses leveldb.
- We use a replication factor of 3.
- Benchmarking System - Modified version of Yahoo! Cloud Serving Benchmark (YCSB).
  - Assign deadlines to each key.

Range	Percentage
10-30ms	20%
30-100ms	30%
100-1000ms	50%

- Data Set - 80 million records, 86.4 GB in size.
- Cache - Total capacity of 19.2GB.

Deadline-Aware Caching - **DLC**

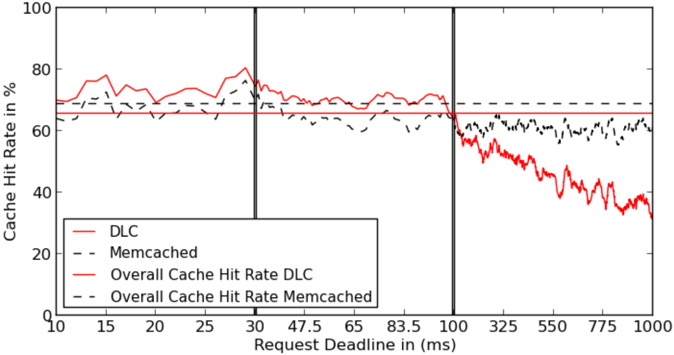


Figure : Cache hit rate for 192 concurrent clients with DLC and Memcached.

Deadline-Aware Caching - **Full MicroFuge**

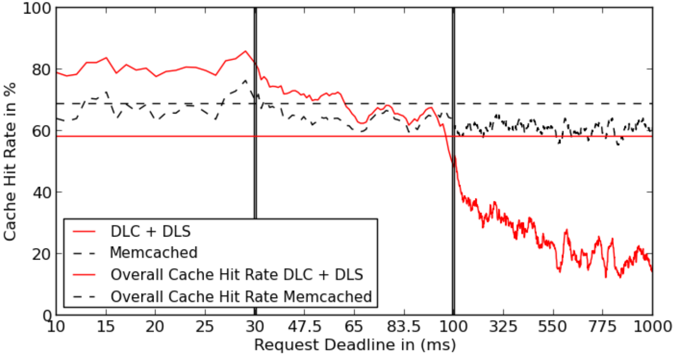


Figure : Cache hit rate for 192 concurrent clients with DLC + DLS and Memcached.

Deadline Miss Rate - **DLC**

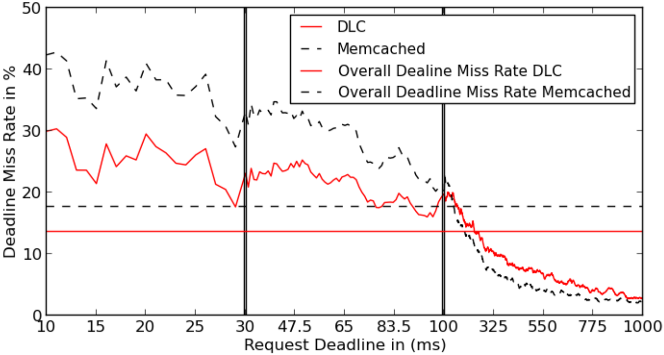


Figure : Deadline miss rate for 192 concurrent clients with DLC and Memcached.

Deadline Miss Rate - **Full MicroFuge**

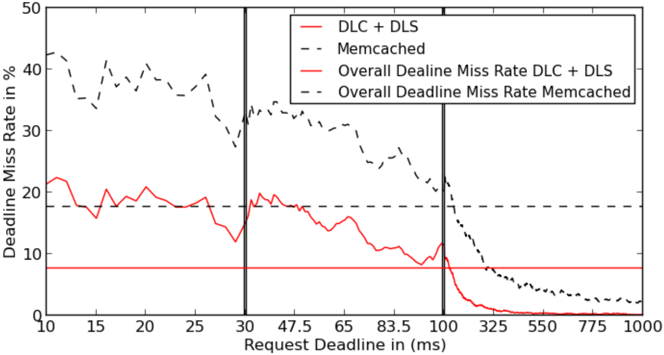


Figure : Deadline miss rate for 192 concurrent clients with DLC + DLS and Memcached.

Conclusion

- Predictable performance is necessary in multi-tenant environments.
- MicroFuge tackles the performance isolation problem with its deadline-aware caching and scheduling middleware.
- MicroFuge reduces deadline miss rate from 17.5% to 7.7% and it can be as low as 4.7% if we turn on the admission control.

Thank You.

DLS - Admission Control

- Bound the fraction of requests that miss their deadlines.
- Requests are rejected in two situations.
    - The request will be miss its own deadline.
    - The new request will cause already accepted requests to miss their deadlines.
- Provides a system parameter β as a knob to control the percentage of deadline misses.

Experimental Results - Deadline Miss with Admission Control

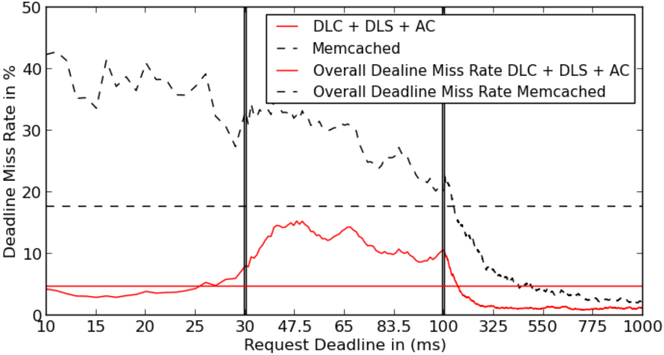


Figure : Deadline miss rate for 192 concurrent clients with DLC + DLS + AC and Memcached.

Experimental Results - Tunable Admission Control

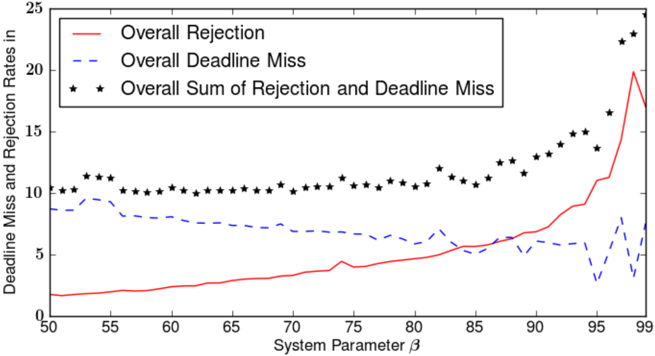
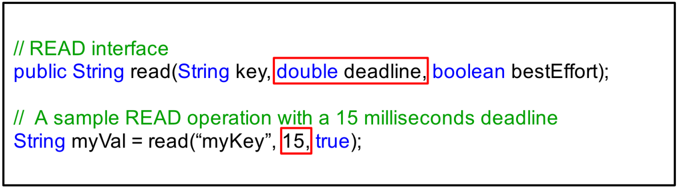


Figure : Deadline miss vs. rejection rates with respect to various values of system parameter β for 192 clients.

MicroFuge at a Glance

- Middleware for popular key-value storage.
- A modified version of the CRUD operation interface.



```
// READ interface
public String read(String key, double deadline, boolean bestEffort);

//  A sample READ operation with a 15 milliseconds deadline
String myVal = read(“myKey”, 15, true);
```

Figure : MicroFuge *read* operation interface.