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

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

July 3, 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.

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
MicroFuge Overview III

[Diagram showing clients interacting with MicroFuge and cloud storage, with steps labeled 1 to 4.]
Deadline Cache (DLC) - Components

Clients

DeadLineCache

Multiple LRU Queues

m1 m2 ... mn

Adaptive Divisors

Eviction

Multiple LRU queue for different deadline ranges
Deadline Cache (DLC) - Components

- Clients: C₁, C₂, C₃, ..., Cₘ
- DeadLineCache
  - Adaptive Divisors: m₁, m₂, ..., mₙ
  - Multiple LRU Queues: q₁, q₂, ..., qₙ
  - Eviction
  - 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

key_2
400

Queue 2
(33-66] ms
m2 = 1

key_3
300

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

key_2
400

Queue 2
(33-66] ms
m2 = 1

key_3
300

Modified Recency Value (MRV) =

Current_timestamp - Stored_timestamp
Queue-Specific Divisor
DLC - A Cache Eviction Example (4)

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

Modified Recency Value (MRV) =
\[
\text{Current\_timestamp} - \text{Stored\_timestamp} \div \text{Queue-Specific Divisor}
\]
DLC - A Cache Eviction Example (5)

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

Queue 1
(0-33] ms
m1 = 3

key_1
200
MRV = 100

Queue 2
(33-66] ms
m2 = 1

key_3
300
MRV = 200

key_2
400

Evict largest MRV

Modified Recency Value (MRV) =

\[
\text{Current\_timestamp} - \text{Stored\_timestamp} = \frac{\text{Queue-Specific Divisor}}{}
\]
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)

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

Queue 1
(0-33] ms
m1 = 3

key_1
200

Waterloo
500

Queue 2
(33-66] ms
m2 = 1

key_2
400

Legend

Queue #
Range Divisor

KEY
TIMESTAMP

Insert with current timestamp
DLC - A Cache Eviction Example (8)

Client: `cachePut`
{key: `Waterloo`,
deadline: 56 ms
missed: `true`);
DLC - A Cache Eviction Example (9)

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

Queue 1
(0-33] ms
m1 = 3

key_1
200

Waterloo
500

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

key_2
400

Legend

Queue #
Range
Divisor

KEY
TIMESTAMP

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

key_1
200

key_2
400

Waterloo
500

Queue 2
(33-66] ms
m2 = 1.6

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

Clients

C_1
C_2
C_3
...
C_n

Deadline Scheduler

Replica Selection

Performance Modelling

Distributed Data Store

d_1
d_2
...
d_m
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.

![Diagram showing a client issuing a cache lookup to DLC, with key 'Waterloo' and data servers 1 and 2 containing the same data labeled 'Waterloo'.]
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.
The client makes a call to the selected DLS and waits for its turn to access the data server.
DLS - An Example (7)

- Snapshot of scheduler’s pending queue.

![Diagram showing a snapshot of scheduler’s pending queue with keys and deadline timestamps.]

- Running Requests
  - Req-1
  - Req-2
The new item is inserted according to earliest deadline first ordering.

---

**Pending Requests Queue**
(Ordering: Earliest Deadline First)

<table>
<thead>
<tr>
<th>Request</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foo</td>
<td>120</td>
</tr>
<tr>
<td>Waterloo</td>
<td>100</td>
</tr>
<tr>
<td>Bar</td>
<td>50</td>
</tr>
</tbody>
</table>

---

**Running Requests**

- Req-1
- Req-2
DLS - An Example (9)

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

### Pending Requests Queue
(Ordering: Earliest Deadline First)

- Foo: 120
- Waterloo: 100
- Bar: 50

### Running Requests

- Req-1
- Empty
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.
The client issues the read request to the data server.
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.

<table>
<thead>
<tr>
<th>Range</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>10-30ms</td>
<td>20%</td>
</tr>
<tr>
<td>30-100ms</td>
<td>30%</td>
</tr>
<tr>
<td>100-1000ms</td>
<td>50%</td>
</tr>
</tbody>
</table>

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

Figure: Cache hit rate for 192 concurrent clients with DLC and Memcached.
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 $\beta$ 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.
Figure: Deadline miss vs. rejection rates with respect to various values of system parameter $\beta$ for 192 clients.
MicroFuge at a Glance

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

```java
// 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.