BMC: Accelerating Memcached using Safe In-kernel Caching and Pre-stack Processing

Yoann Ghigoff\textsuperscript{1,2,3} Julien Sopena\textsuperscript{2} Kahina Lazri\textsuperscript{1} Antoine Blin\textsuperscript{4} Gilles Muller\textsuperscript{3}

\textsuperscript{1}Orange Labs \hspace{1em} \textsuperscript{2}LIP6 – Sorbonne University \hspace{1em} \textsuperscript{3}Inria \hspace{1em} \textsuperscript{4}Gandi

Networked Systems Design and Implementation, 12–14 April, 2021
• In-memory key-value store, simple GET/SET interface
• Used by web services to off load work from databases
• Given its crucial role, Memcached must be able to sustain high network load
Memcached suffers from known kernel overheads:

- Concurrent data structures (e.g. sockets) used by multiple threads
- System calls

![Graph showing CPU usage (%) vs. number of threads for different system calls: sys_epoll_wait, sys_recvfrom, sys_sendmsg.](image)

# of threads: 1, 2, 3, 4, 5, 6, 7, 8
CPU usage (%): 0, 10, 20, 30, 40, 50
Memcached suffers from known kernel overheads:
- Concurrent data structures (e.g. sockets) used by multiple threads
- System calls

![Graph showing throughput vs. number of threads]

- Throughput (KReq/s)
- # of threads
- vanilla Memcached
Memcached suffers from known kernel overheads:

- Concurrent data structures (e.g. sockets) used by multiple threads
- System calls
Memcached suffers from known kernel overheads:

- Concurrent data structures (e.g. sockets) used by multiple threads
- System calls
- Per-packet TCP/IP processing

![Diagram showing network interface card, RX cores, network driver, network stack, socket API, and Memcached with throughput metrics]
Memcached suffers from known kernel overheads:

- Concurrent data structures (e.g. sockets) used by multiple threads
- System calls
- Per-packet TCP/IP processing
Enable a Memcached server to respond to *get* requests without executing the whole network stack.
BMC: in-kernel proxy

Enable a Memcached server to respond to *get* requests without executing the whole network stack.

Filters memcached packets in order to:

1. Serve *get* requests on behalf of the application.
2. Ensure cache coherence as simply as possible.
3. Perform cache updates transparently to the application.
Filters memcached packets in order to:

1. Serve get requests on behalf of the application
Filters memcached packets in order to:

1. Serve get requests on behalf of the application
BMC: in-kernel proxy

Filters memcached packets in order to:

1. Serve get requests on behalf of the application
Filters memcached packets in order to:

1. Serve get requests on behalf of the application
2. Ensure cache coherence as simply as possible
Filters memcached packets in order to:

1. Serve get requests on behalf of the application
2. Ensure cache coherence as simply as possible
BMC: in-kernel proxy

Filters memcached packets in order to:

1. Serve get requests on behalf of the application
2. Ensure cache coherence as simply as possible
BMC: in-kernel proxy

Filters memcached packets in order to:
1. Serve get requests on behalf of the application
2. Ensure cache coherence as simply as possible
Filters memcached packets in order to:

1. Serve get requests on behalf of the application
2. Ensure cache coherence as simply as possible
Filters memcached packets in order to:

1. Serve get requests on behalf of the application
2. Ensure cache coherence as simply as possible
Filters memcached packets in order to:

1. Serve get requests on behalf of the application
2. Ensure cache coherence as simply as possible
3. Perform cache updates transparently to the application
Filters memcached packets in order to:

1. Serve get requests on behalf of the application
2. Ensure cache coherence as simply as possible
3. Perform cache updates transparently to the application
Filters memcached packets in order to:

1. Serve get requests on behalf of the application
2. Ensure cache coherence as simply as possible
3. Perform cache updates transparently to the application
Filters memcached packets in order to:

1. Serve get requests on behalf of the application
2. Ensure cache coherence as simply as possible
3. Perform cache updates transparently to the application
Implementation

eBPF
Linux’s extended version of BPF. The eBPF infrastructure offers the ability to run user-supplied programs inside the kernel.

Benefits of eBPF

- Safety through static analysis
- JIT compilation
- Network driver attach point (XDP)
Implementation

eBPF
Linux’s extended version of BPF. The eBPF infrastructure offers the ability to run user-supplied programs inside the kernel.

Limitations of eBPF
Static analysis doesn’t scale to complex application logic
- Only a limited number of BPF instructions can be analyzed
- Loops must have static bounds
- No dynamic memory allocation
Dealing with eBPF’s limitations

• Bounding data (memcached keys/data/packets)
Dealing with eBPF’s limitations

- Bounding data (memcached keys/data/packets)
- Using a rolling hash function (FNV-1a)
Dealing with eBPF’s limitations

- Bounding data (memcached keys/data/packets)
- Using a rolling hash function (*FNV-1a*)
- Partitioning complex functions
Experimental setup

Two clients generate memcached workload:
- 100 million distinct memcached keys
- Zipf key distribution
- 16-byte keys and 32-byte values
- 30:1 GET/SET ratio

Server under test:
- Linux 5.3.0
- Memcached 1.5.19 with 10 GB of memory
- BMC with 2.5 GB of memory
Throughput

- Up to 18x compared to vanilla Memcached
- Up to 6x compared to MemcachedSR

![Throughput Chart]

**Throughput (MReq/s)**

- **vanilla Memcached alone**
- **MemcachedSR alone**
- **MemcachedSR**
- **BMC**
Throughput

- Up to 18x compared to vanilla Memcached
- Up to 6x compared to MemcachedSR
- No observable deterioration with a worst-case workload
Receive-Process-Reply latency

- Median of memcached hits and misses with BMC is respectively 21.8 and 21.6 µs
- 2.11 µs for a BMC cache hit
- Memcached operations are about 1 µs faster when not running BMC

![Figure 1: with BMC](image1.png)

![Figure 2: without BMC](image2.png)
Comparison to kernel-bypass: Seastar

Memcached

Socket API

Network stack
- BMC
- Network driver

RX core

Network interface card

Seastar / Memcached

DPDK

RX core

Network interface card

Network stack
- BMC
- Network driver
Comparison to kernel-bypass: throughput

- Up to 5x higher throughput on favorable workload
- Better performance scaling on mixed workload
Comparison to kernel-bypass: CPU usage

- Up to 5x higher throughput on favorable workload
- Better performance scaling on mixed workload
- 3x times less CPU resources to achieve similar throughput

![CPU usage chart](chart.png)
Conclusion

BMC

– uses in-kernel caching to serve Memcached requests after they have been received by the network driver
– works with unmodified software on commodity hardware
– offers significant throughput improvement
– introduces negligible overhead

On-going work: Optimized eviction algorithm
Thank you

For more questions:

yoann.ghigoff@inria.fr