October 2021

Key-Value Store Using a Network Packet Filter Situated in the OS Kernel

Andrew C. Jackson

Key-Value Store Using a Network Packet Filter Situated in the OS Kernel

ABSTRACT

When thousands of machines coordinate a small piece of information, a network key-value store can struggle to keep up with the query load. An underlying cause of the capacity bottleneck is that the key-value store is usually held in user space, not kernel space. This disclosure leverages the run-in-kernel capabilities of a network packet filter to deliver very fast responses to key-value (hash map) requests. The network packet filter is capable of communicating with hash-map type memory accesses and can rewrite and redirect packets. When assembled this way, a packet with the query key can be returned to the requester with the value, with the entire request-response operation taking place in kernel space.

KEYWORDS

- Key-value store
- Hash map
- Berkeley packet filter (BPF)
- Enhanced BPF (eBPF)
- Packet filter
- Associative array
- Network key-value
- OS Kernel
- Kernel space processing
- Distributed computing

BACKGROUND

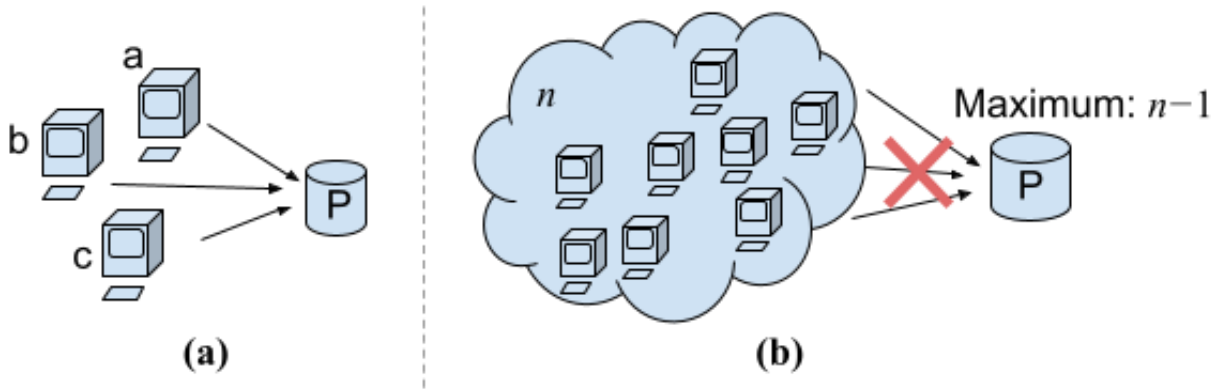

Fig. 1: (a) A distributed computation served by a single node; (b) The single node becomes a bottleneck

Fig. 1(a) illustrates a distributed computation in which compute nodes {a, b, c} access a dynamic state stored in a key-value node P. For example, the dynamic state can be a version of a dataset that any of the compute nodes can update. In the simplest design, the key-value node serves, upon request, the latest value of the (potentially fast-changing) state to a compute node. The key-value node has a maximum capacity $n-1$. If the number of compute nodes accessing the key-value node reaches *n* or greater, as illustrated Fig. 1(b), then the key-value node may no longer be available (in the sense of the consistency-availability-partition-tolerance (CAP) theorem).

While other topologies explore CAP trade-offs, the physical limit of per-machine loading capacity affects all consistent options. When thousands of machines coordinate a small piece of information, a network key-value store can struggle to keep up with the query load. As explained below, an underlying cause of the capacity bottleneck is that the key-value store is usually held in user space, not kernel space.

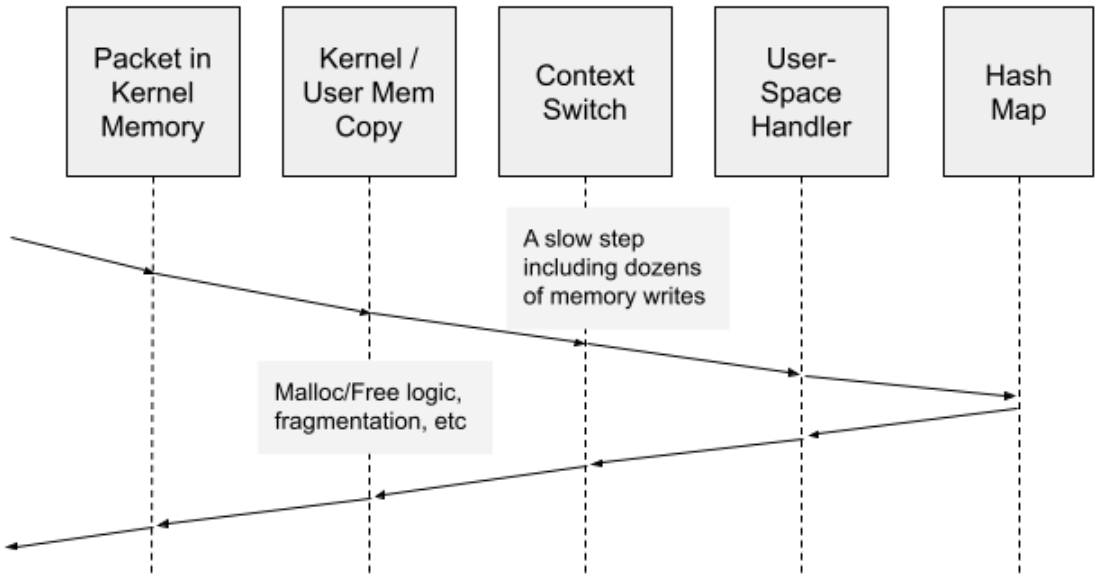

Fig. 2: A user-space key-value store takes substantial time to respond

Fig. 2 illustrates a network key-value database (also known as a request-response server) that operates in user space (as opposed to kernel space). Handling an incoming packet in kernel memory requires many time-consuming steps, e.g., copying the data into user memory (which in turn involves memory allocation and deallocation); switching context; passing to a user-space handler; etc., before it reaches the key-value store (also known as hash map). On the return journey from the key-value store, the response requires an equivalent number of steps. It is the presence of the many processing steps that results in slow responses, which in turn limits the availability of the key-value node.

Distributed computation tasks such as MapReduce cycles run across tens of thousands of machines, sometimes over multiple days of runtime. If some knowledge is found that must integrate into all ongoing processing tasks, communication across all these nodes can be difficult.

A network packet filter, such as the enhanced BPF (eBPF), has a programming facility that provides checked code that can run in-kernel on raw packet data that comes in through an

open port. Such a network packet filter is capable of communicating with hash-map type memory accesses and can rewrite and redirect packets.

DESCRIPTION

This disclosure leverages the run-in-kernel capabilities of a network packet filter to deliver very fast responses to hash map requests. The network packet filter is capable of communicating with hash-map type memory accesses and can rewrite and redirect packets. When assembled this way, a packet with the query “key” can be returned to the requester with the value, with the entire request-response operation taking place in kernel space.

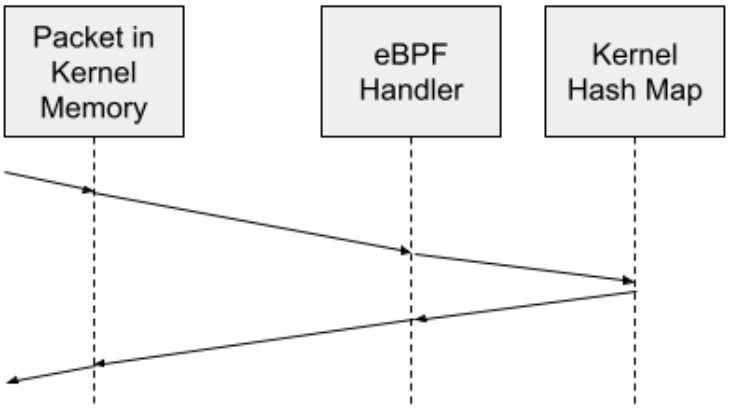

Fig. 3: A kernel-space key-value store that leverages a network packet filter

Fig. 3 illustrates a kernel space network key-value store that leverages a network packet filter. A packet in kernel memory is processed by a filter handler, e.g., an eBPF handler, and the key carried by the packet is returned by the kernel hash map. Effectively, the key-value database is based on an eBPF packet handler, which acts as a request-response engine.

The entire request-response operation occurs within the kernel, such that the overhead of kernel space to user space transformation (and back) is avoided. There are no context switches, memory allocations or deallocations that consume time. The eBPF prover ensures code safety,

making it safer than a kernel C module doing the same work. In the case of identical key and value sizes, the per-request memory usage is limited to the processing stack and the memory allocated to the one incoming packet.

For trusted environments like logs clusters, encryption and authentication can be added. In such environments, exceptional situations can arise, e.g., a problematic piece of analysis. While a problematic analysis may remain unrecovered within a pipeline run, thousands of other analyses are in jeopardy of being subject to code failure. The high cost of reruns can be mitigated, using the described techniques, by using an urgent coordination of a denylist to enable work to proceed. Examples of other applications include authentication cookie lookup, user-state coordination for the purposes of moving a user between loaded backends, and in general, any application where a single piece of rapidly changing information needs to be made available across many servers.

In this manner, the described techniques achieve a very fast request-response turnaround for key-value processing in a BPF-capable kernel, e.g., Linux, with memory safe code. The described BPF-based key-value database increases the availability of the same hardware (presuming an unsaturated network) to handle more nodes while using less processor and memory resources.

CONCLUSION

This disclosure leverages the run-in-kernel capabilities of a network packet filter to deliver very fast responses to key-value (hash map) requests. The network packet filter is capable of communicating with hash-map type memory accesses and can rewrite and redirect packets. When assembled this way, a packet with the query key can be returned to the requester with the value, with the entire request-response operation taking place in kernel space.