IMPLEMENTING ULTRA-LOW-LATENCY DATACENTER SERVICES WITH PROGRAMMABLE LOGIC

A key-value store implemented in field-programmable gate array logic runs 14 times faster while using 13 times less power than the best software implementation using kernel bypass. A record-breaking round-trip network latency below a half-microsecond (0.5 μs) and throughput of 150 million searches per second were achieved for the KVS using a single Stratix V FPGA.

Association of a key to a value is an essential network service in modern datacenters. Keys refer to alphanumeric strings or binary data; values are associated with keys and may change over time. Key-value store (KVS) systems are typically deployed in datacenters to exchange data between servers as users shop, trade, and send data over the Web. Widely used KVS deployments include Dynamo at Amazon (http://aws.amazon.com/dynamodb) and Memcached at Zynga, Twitter, and Facebook.¹ Telecommunication carriers use KVS servers to track the status of mobile users, devices, and network activities.

Today, millions of database servers are used every second to associate billions of keys with values due to human-to-human and human-to-machine interactions (see the “Database Services” sidebar). In the future, far more powerful servers will need to handle trillions of machine-to-machine interactions per second. Reducing latency improves network service responsiveness. Efficient KVS implementations reduce the capital expenses of building out a datacenter service and the operating expenses to power it.

Critical KVS metrics are latency, throughput, and power. Stock traders, brokers, and dealers, as well as option exchanges trading in co-location facilities, need to look up order IDs with low latency. Consumer search and social networking websites need high throughput to search for keywords and match user IDs. Telecommunication and data communication providers want both high throughput and low latency to look up phone numbers, electronic serial numbers (ESNs), and Internet Protocol (IP) addresses as increasing volumes of data stream over wired and wireless networks. By increasing server throughput, lowering latency, and reducing the energy required per operation, datacenter operators can provide services with faster response at lower cost.

In this article, we report record-breaking KVS performance through the use of a new, open-standard message format. Compact, binary-encoded key-value pairs are sent over Ethernet. All message processing and lookup functions are offloaded to field-programmable gate array

John W. Lockwood
Madhu Monga
Algo-Logic Systems
Database Services

Traditionally, the Structured Query Language (SQL) was used as a standard interface between client machines and relational databases. SQL queries were transferred over TCP/IP sockets between client machines and the database servers that stored structured data.

Massive growth in the number of mobile users, quantity of Internet-attached devices, extent of captured information, and amount of collected metadata led to massive growth of datasets. Large-scale cloud services drove the shift from the use of relational database to NoSQL. NoSQL databases provide a more efficient means to store, manage, and analyze big data than SQL databases. Further, NoSQL APIs simplify the way that software developers store and search unstructured data.

Networking services have traditionally been implemented as user-space applications running on top of a host operating system. More recently, there has been a trend to more optimally implement networking functions in software by avoiding the overhead of the Linux kernel, interrupts, and context switches. As such, we benchmark two software KVS implementations—one that uses traditional network sockets to interface with the Linux kernel, and one that uses Intel’s optimized Data Plane Development Kit to bypass the kernel.

Most existing databases are deployed using standard microprocessor-based servers. Datacenter operators typically provision multiple servers to meet the demands of increased network traffic. With standard servers, as analyzed by Michael Ferdman and colleagues, performance improvements were sublinear with the increased workloads.

References


(FPGA) logic using an off-the-shelf FPGA expansion card installed in a standard server. As with scale-out software, additional FPGA-accelerated services can be deployed to handle larger workloads but with far less hardware.

We introduce and use the Open Compute Storage Message (OCSM) to encapsulate the key-value pairs in a packet payload and send them to KVS on FPGA over Ethernet. The messages convey similar data to that used by the memcached format, but the message format is structured to reduce the overhead of text data parsing and variable-length field extraction in hardware.

We implemented two versions of FPGA KVS that differ in how key-value pairs are stored: one uses the FPGA’s on-chip memory, whereas the other uses off-chip memory near the FPGA. The implementation with on-chip memory achieves a record-breaking latency of only 467 ns on 10 Gigabit Ethernet (10 GE) and an estimated latency of only 388 ns on 40 GE to return a value from a key carried in a UDP/IP packet with a 32-byte payload. A single instance of the search core in FPGA logic can support a throughput of 40 million searches per second (MSPS) over a 10-GE interface or 150 MSPS over a 40-GE interface.

The second implementation of KVS uses off-chip DRAM to search a 64-byte message. For a low-cost hardware device with 8 GBytes of DDR3, a single instance of this implementation can support a throughput of 20 MSPS on a 10-GE link and support a table depth of up to 48 million entries. KVS scales to support higher throughputs, limited only by the number of available Ethernet ports and memory banks. The depth of the table scales with the size of the off-chip memories attached to the FPGA.

Open Compute Storage Message

Open standards are critical for the success of datacenter application deployments. OCSM is a new, compact, open-standard packet format suitable for use in both Intranet and Internet-wide KVS. The minimum length of OCSM for KVS using the FPGA’s on-chip memory is just 32 bytes, and the minimum length of OCSM for KVS using the off-chip memory is 64 bytes.

We transport OCSM over the User Datagram Protocol (UDP) to minimize message transmission overhead. Transporting OCSM over TCP would be redundant because KVS acknowledges all messages and the open source software API handles retransmission of lost requests. Multiple OCSMs can be put into a single UDP packet. For a maximum transmission unit (MTU) of 1,514 bytes, the
UDP payload can contain up to 45 32-byte messages or 22 64-byte messages.

OCvSM places no limitation on the size of the key and value that can be supported. For an application that requires a key wider than 96 bits, API function calls can hash over the wider keys to generate a 96-bit key. Similarly, for applications that need to store values wider than 96 or 352 bits, multiple locations can be used to store the longer value in KVS. To look up longer values, the total latency is simply the time required for the first lookup to identify the length of the value, plus the time required for the second dependent lookup, followed by a series of pipelined lookup operations. The throughput proportionally scales with the number of operations required per message.

Implementations of Scale-Up KVS

KVS can be implemented in software in multiple ways. Traditional software implementations are built as a user-space application program that communicates with the network operating system via standard BSD-style network sockets. Optimized software implementations avoid the overhead of processing the packets in the kernel by dedicating CPU cores to read packets directly from the network interface card (NIC).

Search using Software

Traditional software sockets allow user-space applications to bind and accept network connections as a service provided by the operating system’s kernel. The user-space program implements the key-value network service. This program receives packets with keys from a UDP/IP or TCP/IP network socket and then sends packets back with the value. The kernel, in turn, controls the NIC to send and receive packets on the Ethernet interface.

Using an Intel core i7 4770k 3.4-GHz CPU and an Intel 10-GE NIC running on the CentOS (RedHat-based) operating system, we implemented OCSM-based KVS in C. We used the software socket implementation as a baseline for our software and hardware-accelerated implementations.

Search using DPDK

Intel’s Data Plane Development Kit (DPDK) is a set of data plane libraries and network interface controller drivers for fast data processing on an Intel Architecture (IA) platform. To achieve high performance in the implementation, a low overhead run-to-completion model is implemented and devices are accessed via polling to eliminate the overhead of context switches in the interrupt processing. The Linux kernel is bypassed, and dedicated cores are used on Intel i7 processors to directly process packets as they arrive from the NIC.

Search using FPGA

Rather than using a standard NIC to forward packets to software, the fastest version of the packet processing datapath and KVS are implemented in FPGA logic. For the 10-GE version, we ran the KVS on a Nallatech P385 board (www.nallatech.com/solutions/fpga-accelerated-computing/pcie-accelerator-cards), and for the 40-GE version, we ran the KVS application on a BitWare S5HQ board (www.alteraboard.com/product/s5-pcie-hq), each with an Altera Stratix V A7 FPGA device.

The FPGA implementation of KVS uses the Second Generation Exact Match Search Engine (EMSE2; http://algo-logic.com/emse2) core that emulates the function of a binary content addressable memory. The EMSE2 core has two variations: an on-chip memory for all data accesses and a fully off-chip version. On-chip memory is best suited for small tables that require the fastest lookup rates and lowest latencies. A fully off-chip version of the EMSE2 core uses DDR3 memory to store up to 48 million key-value pairs. The large table’s depth is limited only by the size of the off-chip memory.

As Figure 1 shows, the two versions of the EMSE2 were connected to both on-chip and off-chip memories and to packet handlers that read data from the receive side of the Ethernet media access control (MAC), parse the packet, extract the key, and then perform a lookup using the EMSE2. The resulting values are encapsulated into a response packet and sent to the transmit side of the Ethernet MAC. The FPGA datapath, as shown in Figure 1, instantiates Algo-Logic’s Ethernet 10-GE MAC or Altera’s 40-GE MAC/PHY. The rest of the datapath is responsible for extracting the 32-byte or 64-byte OCSMs for either on-chip or off-chip memory.
**Live Demonstration of the KVS**

FPGA KVS was demonstrated and available over the Internet through SCinet, the high-capacity network that supported applications and experiments at the 2015 Supercomputing conference. Two IPv4 addresses were reserved for KVS. Fast KVS and Large KVS were accessible over a 10-GE interface at IP 140.221.165.230 and 140.221.165.231, respectively, in a network setup described in Figure 2. A Web GUI for FPGA KVS over a 40-GE interface demonstrated the latency for live lookups.

The network setup for FPGA KVS over the 10-GE interface, shown in Figure 2, included a Layer 2 10-GE switch with SFP+ ports, a host machine to run a Web GUI, and a server to host the FPGA card for the KVS. We connected the KVS to SCinet over a 10-GE interface. From the Layer 2 switch, one of the SFP+ ports was used to connect to SCinet, while two SFP+ ports were connected to KVS on-chip and off-chip implementations. Another port from the Layer 2 switch was connected to the host machine running a Web GUI that displayed measured latency. Using their public IP address, both Fast KVS and Large KVS were made reachable from the public Internet.

The network setup for FPGA KVS over the 40-GE interface included another switch with QSFP+ ports to connect the 40-GE client machine to FPGA KVS. Fast KVS was subsequently demonstrated over that network and the results shown on the Web GUI over SCinet.

KVS scales out for use in the datacenter. Whereas a single instance of Fast KVS over a 40-GE interface in an FPGA supports a throughput of 150 MSPS, a scaled-out
HOT INTERCONNECTS

Benchmarks
We evaluated each of the software, DPDK, and FPGA implementations of KVS using the on-chip table over the 10-GE interface using a series of tests to determine latency, throughput, and power usage. As Berk Atikoglu and colleagues discussed, most (99.8 percent) of the KVS workload is composed of lookup requests.

Update and delete operations are used infrequently (0.2 percent). Thus, we benchmark KVS latency and throughput with lookup operations.

Latency Measurement
The total latency of KVS is a combination of latency through the lookup IP, datapath, PHY + MAC, SFP+/QSFP+ module delay, and fiber delay. Figure 4 breaks down the latency measurements at each of these points involved in the KVS search transaction over 10-GE and 40-GE interfaces.

The EMSE2 latency is the time required by the EMSE2 IP to complete the search transaction. The EMSE2 IP interfaces with the datapath responsible for parsing the UDP packet and extracting OCSM. The search latency is the time required to parse OCSM and do the lookup using the EMSE2 IP. The datapath then interfaces with the PHY + MAC module, and this latency through the 10-GE PHY + MAC module is added to the search latency to obtain the card latency. Finally, the host latency is calculated as the sum of the delay through the SFP+ or QSFP+ module, the fiber, and the card latency.

As Figure 5 shows, the FPGA implementation of KVS using an Altera Stratix V FPGA achieved the lowest latency of 0.467 μs. Because the number of clock cycles required to process the search is engineered into the logic as a constant, the search achieves jitter-free operation.

The DPDK implementation had the next lowest latency of 6.43 μs. By dedicating a CPU core to spin-wait on incoming packets, the KVS application avoided the Linux kernel’s overhead but still used the CPU to perform the lookup operation.

Finally, the traditional software socket implementation had the worst average latency of 41.54 μs. The overhead of forwarding...
packets through the kernel and using software to sequentially process the packets was approximately 88 times slower than the results achieved with the FPGA implementation. The latency distribution of the implementation using software with sockets had a wide variation of approximately 10 μs in the time required to process packets with a standard deviation of approximately 2 μs.

**Throughput Measurements**

Throughput tests were performed to measure the number of OCSMs sent and received. The throughput was determined as the maximum request rate that results in all sent messages processed properly. The maximum throughput of 40 MSPS from the table stored in on-chip memory is limited by the number of 32-byte messages received on a 10-GE link. The maximum throughput of 20 MSPS from the table stored in off-chip memory is limited by the number of 64-byte messages received on a 10-GE link.

Figure 4. Latency measurement for FPGA KVS. The figure shows the measured latency at different checkpoints in the datapath.

Figure 6’s vertical axis shows the percent of messages processed as a function of the traffic sent to the system. All three implementations (sockets, DPDK, and FPGA) initially processed 100 percent of the messages received. However, as the traffic load increased beyond several million messages per second, the fraction of packets processed decreased.

First, the socket implementation and then the DPDK implementation became unable to sustain full throughput. However, the FPGA implementation was able to maintain full throughput up through the maximum speed of the 10-GE interface.

**Power Measurement Mechanism**

Power usage was determined by taking the system’s power consumption rate and dividing it by the system’s maximum throughput with the result measured in units of Joules/OCSM. The FPGA implementation achieved by far the lowest energy consumption both in terms of marginal power and total power.
Figure 5. Latency distribution for FPGA, Data Plane Development Kit (DPDK), and socket implementations of KVS over the 10-GE interface.

Figure 6. Throughput comparison of software, DPDK, and FPGA implementation of KVS. Note that the FPGA version of the KVS never drops any packets, whereas the software version of the KVS cannot keep up with the offered load of requests.
Tables 1 and 2 show a summary of the performance for socket, DPDK, and FPGA. Compared with DPDK, the latency and power consumption of FPGA are 14 and 13 times less than DPDK, respectively, whereas the maximum throughput of FPGA is 2.5 times larger than DPDK. In comparison to the socket implementation, the latency and power consumption of FPGA is 88 and 21 times less than the socket, respectively, whereas the maximum throughput of FPGA is 10 times larger than the socket implementation. As we can see, the FPGA implementation achieved the highest performance in all metrics over software by one or more orders of magnitude for processing the same workload. Furthermore, FPGA power consumption of 0.52 µJoules per OCSM for processing the compact binary OCSM is 63 times lower than the 33 µJoules/OCSM observed from another FPGA implementation that processes memcached messages. Combined, the implementation in programmable logic and use of the smaller OCSM enables deployment of KVS in datacenters with a performance improvement that is several orders of magnitude better than the existing KVS systems deployed in software.

The impact of a fast KVS within datacenters will be significant. By reducing the latency and increasing the throughput of information shared between machines, sites can scale up the number of users that they service. Going forward, Algo-Logic is working to accelerate a wide base of applications that benefit from KVS in FPGA, including low-latency finance (trading, compliance), communications (network status, short messages), database speedup, sensor tracking for location and movement, social status updates, and multimedia such as multiplayer games, video, and audio.

Acknowledgments
We thank members of the Algo-Logic team for help in developing and testing KVS hardware and software. We also greatly appreciate the help of Emily Iwankowitsch, who edited multiple drafts of this document.

References
HOT INTERCONNECTS


John W. Lockwood is the CEO of Algo-Logic Systems. His research interests include design of low-latency networking systems, reconfigurable logic, and high-performance computing. Lockwood received a PhD in electrical and computer engineering from the University of Illinois, Urbana–Champaign.

He is a member of IEEE, ACM, and Tau Beta Pi. Contact him at jwlockwd@algo-logic.com.

Madhu Monga is a technical staff member at Algo-Logic Systems. Her research interests include designing high-performance FPGA-based solutions for datacenters. Monga received an MS in computer engineering from Iowa State University. Contact her at madhu@algo-logic.com.

Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.