Editor: Robert Blumen
SalesForce
robert@robertblumen.com

ScyllaDB Optimizes Database Architecture to Maximize Hardware Performance

Nishant Suneja

From the Editor

In Episode 354 of "Software Engineering Radio," guest Avi Kivity, chief technology officer of ScyllaDB, talks with host Ninchant Suneja about ScyllaDB and what makes it a high-performance version of Cassandra, a distributed key-value data store. To hear the full interview, visit http://www.se-radio.net or access our archives via RSS at http://feeds.feedburner.com/se-radio.—*Robert Blumen*

Ninchant Suneja: Briefly describe ScyllaDB.

Avi Kivity: ScyllaDB is a distributed no-SQL database. It's compatible with Cassandra. ScyllaDB's high performance gets the most out of hardware and brings low latency and high throughput to applications. It is autonomous, self-tuning, and adaptable.

What was the motivation for building ScyllaDB?

Some years ago, we were building OSv, a cloud operating system, and we set out to optimize Cassandra for OSv to achieve performance gains. Because of bad performance characteristics in Cassandra, there was nothing we could do at the operating-system level. But we liked Cassandra's shared-nothing and fully symmetric architecture. The combination of good architecture and poor implementation represented an opportunity, so we pivoted from OSv and began working on ScyllaDB.

Compare ScyllaDB and Cassandra in terms of throughput and latency.

Benchmarks show that you can reach a million operations per node with ScyllaDB, and throughput grows with [the] number of nodes. This depends on the workload: Complex operations, or operations with large amounts or larger rows, will have lower throughput

Digital Object Identifier 10.1109/MS.2019.2909854
Date of publication: 18 June 2019

and fewer operations per node. Simpler operations achieve higher throughput. Throughput is affected by how the database does internal maintenance, things like repair and bootstrapping. These must be considered, and we have benchmarks and blog posts about them.

The ScyllaDB website shows ScyllaDB achieving throughput on a three-node cluster for Yahoo! Cloud that Cassandra was achieving on a 30-node cluster with lower P99 latencies, representing 10-times cost savings.

The less hardware you need, the more savings you get; conversely, the same amount of hardware can do more or achieve lower throughput.

ScyllaDB is written in C++ and Cassandra in Java. Why?

The top motivation was control. Java is a managed language. It runs in the JVM, which mediates what you do with hardware and the operating system. C++ allows you to do anything you like. You can do bad things like accessing the freed memory, but you can lay out data and memory however you like, access any OS function, or bypass the OS.

What about garbage collection's impact on latency?

Garbage collection can pause a node for hundreds of milliseconds or more and do unpredictable things. Problems propagate across the cluster in a distributed environment because a node is not responding.

Without a garbage collector, how does ScyllaDB manage heap fragmentation?

Because many C or C++ databases or in-memory databases don't do compaction, problems arise when changing the average size of stored data. When you try to store a larger object, you have to evict the entire cache to find contiguous space to store the new data.

With ScyllaDB, we have an additional log-structured allocator to store data. When we allocate, we allocate to a new position, and, when we free, we do not immediately reuse the freed data. Because the log-structured allocator is aware of the location and the data types of all the data that is stored in it, the allocator can move the data whenever it wants and, thereby, compact the heap. It uses 128-K segments to store this data, so it fragments larger data.

What is a shared-nothing architecture?

Typically, architectures share an expensive storage-area network accessible from multiple computers, a disk with multiple interfaces connected to multiple SCSI buses to implement clusters, providing a shared pool of storage connected to multiple machines. While this could always fit on your shared storage pool, storage was expensive.

Like Cassandra, ScyllaDB uses shared-nothing architecture among nodes and cores. Each core accesses only its own memory, files, and connections, reducing the need for locking, [which is] expensive in a shared architecture. ScyllaDB allows use of a large number of available cores, which is almost impossible in a traditional architecture.

What is a disadvantage of having multiple application threads per core?

The kernel scheduler must decide which thread runs each time, so you lose control. You need control in something like a database. To mediate access to data, you need locks, which are becoming more expensive. When you transition from one thread to another, you must go through the kernel, and context switches are expensive and are becoming slower over time. Having one thread per core, the kernel needn't decide which thread to run; all thread-switching costs are gone.

In ScyllaDB, how do two cores communicate, and how do application threads with one application running on each core communicate?

Every pair of cores is connected by a pair of single-producer, single-consumer queues, implemented with just memory barriers. One core places a message into the queue, the other retrieves the message, does what was requested, and responds in the return queue. When you apply batching, each operation is amortized: Many messages are pulled from the queue at once, reducing cost per message and providing high throughput.

Why would intercore communication be necessary?

We are trying to reduce it. Intercore communication is most common with client connections. A client connecting to a node doesn't know which shard it's connected to and sends requests to the node but not to a particular shard. A request could arrive at one shard that must be processed by a different shard or core.

We enhanced the protocol and drivers to allow each client driver to generate one connection per core so that the driver can send the request directly at the core that will service the request. Intercore communications reduce performance and generate imbalance, the bane of thread-per-core implementations.

What is the Seastar framework, and what is its relationship with ScyllaDB?

Seastar is a generic framework for thread-per-core applications. It's oriented at server applications that mix multicore, IO, and networking. Many applications do asynchronous networking, but few frameworks do asynchronous IO and fully utilize the **thread-per-core** architecture. Distributed databases, distributed file systems, and other storage-intensive applications benefit from Seastar.

Why does ScyllaDB use Seastar's user-space disk IO scheduler instead of the Linux-kernel scheduler?

Again, control. The best place to queue IO is in the device itself or in the kernel. Queueing in user space, however, gives you control. The device can schedule IO for best throughput, but, when the device or kernel reorders requests, the order may not be what you want. By queueing in user space, we control what gets serviced next and give the device and kernel just enough to achieve high throughput.

How many IO queues does ScyllaDB maintain in user space?

There are two levels of IO queues. One is per shard, containing all of the IOs that are destined to go into the disk for that shard. If the disk does not have high concurrency, we keep fewer IO queues than there are shards, to restrict the concurrency of IO going into the disk to a sustainable level without incurring latency. The second level is one IO-queue service class and pair shard. Service classes are things like query, compaction, commit log, table flushes, and repair. We isolate those operations from each other so none can dominate.

What are the advantages of using a user-space TCP/IP stack over something like NFSD, which runs in the kernel?

Seastar supports a user-space TCP/IP stack, but ScyllaDB does not use the stack by default. The stack can be turned on, but we don't recommend it for production. We decided that the incremental benefit and performance was not worth incurring more work for the user.

Does ScyllaDB still maintain the notion of one TCP/IP stack per CPU core, or is the TCP/IP stack shared across all the cores?

Only partially. Yes for the user-space TCP/IP stack, but for the POSIX TCP/IP stack, only partially. We have a connection per core so you can send messages directly to other nodes, and we configure the kernel TCP/IP stack to reduce core hops.

What factors should be considered by a user of the Seastar framework to integrate the TCP/IP stack in an application?

Use a user-space stack when you have high packet rates. The overhead of processing a packet in the kernel and distributing the packet to the correct core is higher than in user space, when you can make sure that the packet arrives at the correct core in the first place. A high-packet-rate application is a good use for the user-space TCP/IP stack. The application must also be amenable to thread per core.

Why did you diverge from Cassandra in abandoning the Linux-kernel page cache for user-space role-based cache managed in ScyllaDB?

Control again. Linux page cache gives the kernel control over how the cache is managed and how things are evicted when the cache is filled. The kernel is not tuned to what we want. Having an object cache allows us to get better memory utilization and reduce our CPU utilization. For an object cache, you need something specialized; you can't use the page cache.

Is there a trend for more kernel functionality to come into user space, giving rise to the microkernel notion?

An infrastructural application, like a database or file system, will be run on thousands of nodes, not all in one user, and in a large number of places. It is worth the effort to specialize all of those algorithms and use them for the application because the effort is amortized over many installations. As cores proliferate in machines, the payoff increases.

Since it's now maintaining all the cache in the user space by itself, what are the cache-eviction and cache-invalidation strategies in ScyllaDB?

When we flush a memtable to disk, we have to do something with the object in the memtable. If the partition we are flushing also exists in the cache, we merge the data into that cache. If we know that the partition is new and doesn't exist on disk, we also merge the partition with the cache. If the partition exists on disk but not in the cache, we throw it away.

Can you describe how read and write requests will go through the whole ScyllaDB stack?

The client issues a request to a coordinator. Every ScyllaDB node has a role both of a replica and a coordinator.

The coordinator is client-facing. If the client is using a token or a driver, it will select a coordinator that is also acting as a replica. A coordinator is then on the same node as a replica for that data, saving network cost.

The driver sends a request to the coordinator, which selects the replicas that participate in that request. In a write, all replicas participate; in a read, perhaps fewer. The driver sends messages across the network to those replicas. If it's a driver supplied by ScyllaDB, the request will also arrive at the correct core, so you get better throughput than latency.

Same for a write request?

Reads and writes are similar except for the number of replicas contacted. For writes, all nodes that are up are contacted to make sure that all replicas are current. This delivers the best throughput and latency. You can request a core room, which increases consistency but reduces throughput. Or you contact all replicas for that role for best consistency, but availability may be affected if a node is down.

Cassandra and ScyllaDB both have the gossip protocol for membership changes in the cluster. Can I have a cluster with both ScyllaDB nodes and Cassandra nodes?

No, they are not wire compatible. Wire compatibility would restrict our freedom of implementation. Also, integrating ScyllaDB into a Cassandra cluster might destabilize the Cassandra cluster while migrating. The typical migration path is to have parallel clusters running. The client sends each write into both clusters to keep data consistent. When everything is working, you switch over to ScyllaDB and continue with one cluster.

How will ScyllaDB implement lightweight transactions in future release[s]?

We will use the Raft protocol instead of the older Paxos protocol to establish a leader for every group of partitions. The leader coordinates all [writing] to a group of partitions, preventing concurrent writes.

Is there an application-level API compatibility between ScyllaDB and Cassandra and complete support for the Cassandra query language?

If we add extensions, we try to make sure they don't conflict with the Cassandra language. There may be gaps when we haven't completed a feature, such as lightweight transactions. In addition to the language, the protocol is the same. You can use the same driver you've always used. Migrating from Cassandra to ScyllaDB usually requires no changes.

What features does ScyllaDB provide for live backup and restore in case of a region failure?

SOFTWARE ENGINEERING RADIO

Visit www.se-radio.net to listen to these and other insightful hour-long podcasts.

RECENT EPISODE

- 361—Daniel Berg, who is a distinguished Engineer at IBM cloud unit, talks about Istio service mesh and how it lets developers deploy services into the cloud in a secure, efficient fashion by removing the burden of devops from the core service logic. Host Nishant Suneja begins the show by talking about the about the need for a service-mesh-based microservice architecture, which is followed by a discussion on the Istio project's design goals.
- 363—Host Adam Gordon Bell speaks with Jonathan Boccara, author of *The Legacy Code Programmer's Toolbox*, about reading legacy code, developing the right attitude for approaching legacy code, and several techniques for improving your legacy code skill set.
- 364—Peter Zaitsev of Percona discusses with host Nate Black how to choose the right open source database and why you may need multiple databases. Topics include: vendor lock-in and the role of cloud database services, the value of experience, judging what use cases databases are bad at, why there is no free lunch when it comes to scalability, and why it doesn't usually make sense to copy the largest scale players.

UPCOMING EPISODES

- Thorstein Ball discusses implementing an interpreter
- Diomidis Spinellis talks about effective debugging
- Arnon Axelrod chats about test automation.

ABOUT THE AUTHOR

NISHANT SUNEJA is a software developer at Amazon Web Services, solving distributed systems problems in the Dynamodb team in Palo Alto, California. Previously, he has worked at Yahoo! and Nvidia, among other companies. Suneja received his master's degree in computer science from Stonybrook University, New York, and his bachelor's degree from the International Institute of Information Technology, Hyderabad, India. Contact him at nishantsuneja@gmail.com.

ScyllaDB can make sure that there is one replica for data on each availability zone. If a zone dies, you can recover data from the surviving zones. If you have a region or a data center drop, you can rebuild the data from other surviving data centers. For a complete disaster, you have backup and restore. You can snapshot the database and copy files to an offline storage location, then reconstruct at the point of backup.

This would require some downtime for the application.

Yes. For live backup, you create another data center to replicate to. That center contains all the data in your database, so you can also switch your applications to that. Most deployments use a multimaster database, with reads and writes to all of [the] data centers in parallel.

Are data centers in continuous sync with each other? Are they synchronizing continuously [or] asynchronously?

Both modes are supported. You can have synchronous replication so that a write waits until other data centers are fully synchronized, giving a high guarantee of consistency but trading off latency. Or you can have asynchronous replication and get the response only from the local replicas as ScyllaDB continues replication in the background.

The delay is just a few hundred milliseconds for the data to propagate. ScyllaDB writes the data into [the] local disk and tries to replicate the data later in case of failure. There are also repair facilities to resynchronize in case of lost connections between data centers.

What upcoming features are you working on, and when will they be released?

[Version] 3.0 will have improved support for large partitions. Queries will be faster, and the amount of storage used will be lower. Others are material skews, lightweight transactions, and tiered storage.

Will you track the Cassandra features in the future or branch off?

We will continue providing Cassandra features.

IEEE Software (ISSN 0740-7459) is published bimonthly by the IEEE Computer Society. IEEE headquarters: Three Park Ave., 17th Floor, New York, NY 10016-5997. IEEE Computer Society Publications Office: 10662 Los Vaqueros Cir., Los Alamitos, CA 90720; +1 714 821 8380; fax +1 714 821 4010. IEEE Computer Society headquarters: 2001 L St., Ste. 700, Washington, DC 20036. Subscribe to *IEEE Software* by visiting www.computer.org/software.

Postmaster: Send undelivered copies and address changes to *IEEE Software*, Membership Processing Dept., IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854-4141. Periodicals Postage Paid at New York, NY, and at additional mailing offices. Canadian GST #125634188. Canada Post Publications Mail Agreement Number 40013885. Return undeliverable Canadian addresses to PO Box 122, Niagara Falls, ON L2E 6S8, Canada. Printed in the USA.