Agenda

Background
Goals
Methods
Conclusion
Non-Agenda

- Docker
- Microservices
- Node.js
- Docker

- Orchestration
- JVM GC Tuning
- JSON over HTTP
- Docker
More Non-Agenda

- Cache lines, coherency protocols
- NUMA
- Algorithms are the only thing that matters, everything else is implementation detail
- Docker
Background - ScyllaDB

- Clustered NoSQL database compatible with Apache Cassandra
- ~10X performance on same hardware
- Low latency, esp. higher percentiles
- Self tuning
- C++14, fully asynchronous; Seastar!
YCSB Benchmark: 3 node Scylla cluster vs 3, 9, 15, 30 Cassandra machines

![Graph showing benchmark results](image-url)
Log-Structured Merge Tree

- SStable 1
- SStable 2
- SStable 3
- SStable 4
- SStable 5

Time

Foreground Job

Background Job
High-level Goals

● **Efficiency:**
  ○ Make the most out of every cycle

● **Utilization:**
  ○ Squeeze every cycle from the machine

● **Control**
  ○ Spend the cycles on what we want, when we want
Characterizing the problem

- Large numbers of small operations
  - Make coordination cheap
- Lots of communications
  - Within the machine
  - With disk
  - With other machines
Asynchrony, Everywhere
General Architecture

- Thread-per-core design
  - Never block
- Asynchronous networking
- Asynchronous file I/O
- Asynchronous multicore
Scylla has its own task scheduler

Traditional stack

Thread is a function pointer

Stack is a byte array from 64k to megabytes

Context switch cost is high. Large stacks pollutes the caches

Scylla’s stack

Promise is a pointer to eventually computed value

Task is a pointer to a lambda function

No sharing, millions of parallel events
The Concurrency Dilemma
Concurrency = Throughput * Latency
Fundamental performance equation

Throughput = \frac{\text{Concurrency}}{\text{Latency}}
Fundamental performance equation

\[ \text{Latency} = \frac{\text{Concurrency}}{\text{Throughput}} \]
Lower bounds for concurrency

- Disks want minimum iodepth for full throughput (heads/chips)
- Remote nodes need concurrency to hide network latency and their own min. concurrency
- Compute wants work for each core
Results of Mathematical Analysis

● Want high concurrency (for throughput)
● Want low concurrency (for latency)
● Resources require concurrency for full utilization
Sources of concurrency

● Users
  ○ Reduce concurrency / add nodes

● Internal processes
  ○ Generate as much concurrency as possible
  ○ Schedule
Resource Scheduling

- User read: 30
- User write: 12
- Compaction (internal): 50
- Streaming (internal): 50

Scheduler → Storage

8
Why not the Linux I/O scheduler?

- Can only communicate priority by originating thread
- Will reorder/merge like crazy
- Disable
Figuring out optimal disk concurrency
Cache design

Cache files or objects?
Using the kernel page cache

- 4k granularity
- Thread-safe
- Synchronous APIs
- General-purpose
- Lack of control (1)
- Lack of control (2)

- Exists
- Hundreds of hacker-years
- Handling lots of edge cases
Unified cache

Cassandra

Row cache

On-heap / Off-heap

Key cache

Linux page cache

SSTables

Tuning

Parasitic rows

Page faults

App thread

Kernel

SSD

Page fault

Suspend thread

Initiate I/O

Context switch

I/O completes

Interrupt

Context switch

Map page

Resume thread

Your data (300b)

SSTable page (4k)
Workload Conditioning
Workload Conditioning

- Internal feedback loops to balance competing loads
Replacing the system memory allocator
System memory allocator problems

- Thread safe
- Allocation back pressure
Seastar memory allocator

- **Non-Thread safe!**
  - Each core gets a private memory pool
- **Allocation back pressure**
  - Allocator calls a callback when low on memory
  - Scylla evicts cache in response
One allocator is not enough
Remaining problems with malloc/free

- Memory gets fragmented over time
  - If workload changes sizes of allocated objects
- Allocating a large contiguous block requires evicting most of cache
OOM :(
Log-structured memory allocation

- The cache
  - Large majority of memory allocated
  - Small subset of allocation sites
- Teach allocator how to move allocated objects around
  - Updating references
Log-structured memory allocation

Fancy Animation
Future Improvements
Userspace TCP/IP stack

- Thread-per-core design
- Use DPDK to drive hardware
- Present as experimental mode
  - Needs more testing and productization
Query Compilation to Native Code

- Use LLVM to JIT-compile CQL queries
- Embed database schema and internal object layouts into the query
Conclusions

- Full control of the software stack can generate big payoffs
- Careful system design can maximize throughput
- Without sacrificing latency
- Without requiring endless end-user tuning
- While having a lot of fun
How to interact

- **Download:** http://www.scylladb.com
- **Twitter:** @ScyllaDB
- **Source:** http://github.com/scylladb/scylla
- **Mailing lists:** scylladb-user @ groups.google.com
- **Company site & blog:** http://www.scylladb.com
THE SCYLLA IS THE LIMIT
Thank you.