Running Netflix on Cassandra in the Cloud
(I’m skipping all the cloud intro etc. did that yesterday… Netflix runs in the cloud, if you hadn’t figured that out already you aren’t paying attention and should go read slideshare.net/netflix)
Netflix Deployed on AWS

2009
- Content
  - Content Management
  - EC2 Encoding
  - S3 Petabytes
- Logs
  - S3 Terabytes
  - EMR
  - Hive & Pig
- Play
  - Business Intelligence
  - CDN routing
  - Bookmarks
  - Logging
- WWW
  - DRM
  - Search
  - Movie Choosing
  - Ratings
- API
  - Sign-Up
  - Movie Choosing
  - Social Facebook
- CS
  - Metadata
  - Device Config
  - TV Movie Choosing
  - Customer Call Log
  - CS Analytics

2009
- CDNs
- ISPs
- Terabits
- Customers

2010
- 2009
- 2010
- 2010
- 2011
Cassandra on AWS

A highly available and durable deployment pattern
Cassandra Service Pattern

Service REST Clients

Data Access REST Service
Astyanax Cassandra Client

Cassandra Cluster
Managed by Priam
Between 6 and 72 nodes

Datacenter
Update Flow

Appdynamics Service Flow Visualization
Production Deployment

- Over 50 Cassandra Clusters
- Over 500 nodes
- Over 30TB of daily backups
- Biggest cluster 72 nodes
- 1 cluster over 250Kwrites/s
High Availability

• Cassandra stores 3 local copies, 1 per zone
  – Synchronous access, durable, highly available
  – Read/Write One fastest, use for fire and forget
  – Read/Write Quorum 2 of 3, use for read-after-write

• AWS Availability Zones
  – Separate buildings
  – Separate power etc.
  – Fairly close together
Triple Replicated Persistence
Cassandra maintenance drops individual replicas
Cassandra Instance Architecture

Linux Base AMI (CentOS)

- Priam
- Cassandra Manager
- Token Management, Backups, Autoscaling
- Tomcat/Java7

Java7

- AppDynamics appagent monitoring
- GC and thread dump logging

Cassandra 1.09

Monitoring
- Log rotation
- AppDynamics machineagent
- Etc.
Priam – Cassandra Automation
Available at http://github.com/netflix

- Netflix Platform Tomcat Code
- Zero touch auto-configuration
- State management for Cassandra JVM
- Token allocation and assignment
- Broken node auto-replacement
- Full and incremental backup to S3
- Restore sequencing from S3
- Grow/Shrink Cassandra “ring”
Astyanax
Available at http://github.com/netflix

• Features
  – Complete abstraction of connection pool from RPC protocol
  – Fluent Style API
  – Operation retry with backoff
  – Token aware

• Recipes
  – Distributed row lock (without zookeeper)
  – Multi-DC row lock
  – Uniqueness constraint
  – Multi-row uniqueness constraint
  – Chunked and multi-threaded large file storage
Paginate through all columns in a row

```java
ColumnList<String> columns;
int pageize = 10;
try {
    RowQuery<String, String> query = keyspace
        .prepareQuery(CF_STANDARD1)
        .getKey("A")
        .setIsPaginating()
        .withColumnRange(new RangeBuilder().setMaxSize(pageize).build());

    while (!(columns = query.execute().getResult()).isEmpty()) {
        for (Column<String> c : columns) {
        }
    }
} catch (ConnectionException e) {
}
```
“Traditional” Cassandra Write Data Flows
Single Region, Multiple Availability Zone, Not Token Aware

1. Client Writes to any Cassandra Node
2. Coordinator Node replicates to nodes and Zones
3. Nodes return ack to coordinator
4. Coordinator returns ack to client
5. Data written to internal commit log disk (no more than 10 seconds later)

If a node goes offline, hinted handoff completes the write when the node comes back up.

Requests can choose to wait for one node, a quorum, or all nodes to ack the write

SSTable disk writes and compactions occur asynchronously
Astyanax - Cassandra Write Data Flows
Single Region, Multiple Availability Zone, **Token Aware**

1. Client Writes to local coordinator
2. Coordinator writes to other zones
3. Nodes return ack
4. Data written to internal commit log disks (no more than 10 seconds later)

If a node goes offline, hinted handoff completes the write when the node comes back up.

Requests can choose to wait for one node, a quorum, or all nodes to ack the write

SSTable disk writes and compactions occur asynchronously
Data Flows for Multi-Region Writes

Token Aware, Consistency Level = Local Quorum

1. Client writes to local replicas
2. Local write acks returned to Client which continues when 2 of 3 local nodes are committed
3. Local coordinator writes to remote coordinator.
4. When data arrives, remote coordinator node acks and copies to other remote zones
5. Remote nodes ack to local coordinator
6. Data flushed to internal commit log disks (no more than 10 seconds later)

If a node or region goes offline, hinted handoff completes the write when the node comes back up. Nightly global compare and repair jobs ensure everything stays consistent.
Extending to Multi-Region

Added production UK/Ireland support with no downtime
Minimize impact on original cluster using bulk backup move

1. Create cluster in EU
2. Backup US cluster to S3
3. Restore backup in EU
4. Local repair EU cluster
5. Global repair/join

Take a Boeing 737 on a domestic flight, upgrade it to a 747 by adding more engines, fuel and bigger wings and fly it to Europe without landing it on the way...
Cassandra Backup

• Full Backup
  – Time based snapshot
  – SSTable compress -> S3

• Incremental
  – SSTable write triggers compressed copy to S3

• Archive
  – Copy cross region
Cassandra Explorer
Open source on github soon
ETL for Cassandra

- Data is de-normalized over many clusters!
- Too many to restore from backups for ETL
- Solution – read backup files using Hadoop
- Aegisthus
  - High throughput raw SSTable processing
  - Re-normalizes many clusters to a consistent view
  - Extract, Transform, then Load into Teradata
Asgard


• Replacement for AWS Console at Scale
  – Groovy/Grails/JVM based
  – Supports all AWS regions on a global basis
  – Specific to AWS feature set

• Hides the AWS credentials
  – Use AWS IAM to issue restricted keys for Asgard
  – Each Asgard instance manages one account
  – One install each for test, prod, audit accounts
Open Source Projects

Legend

- Github / Techblog
- Apache Contributions
- Techblog Post
- Coming Soon

- Priam
  Cassandra as a Service
- Astyanax
  Cassandra client for Java
- CassJMeter
  Cassandra test suite
- Cassandra Multi-region EC2 datastore support
- Aegisthus
  Hadoop ETL for Cassandra
- Explorers
- Governorator - Library lifecycle and dependency injection
- Odin
  Workflow orchestration
- Async logging
- Exhibitor
  Zookeeper as a Service
- Curator
  Zookeeper Patterns
- EVCache
  Memcached as a Service
- Eureka / Discovery
  Service Directory
- Archaius
  Dynamics Properties Service
- EntryPoints
- Server-side latency/error injection
- REST Client + mid-tier LB
- Configuration REST endpoints
- Servo and Autoscaling Scripts
- Honu
  Log4j streaming to Hadoop
- Circuit Breaker
  Robust service pattern
- Asgard - AutoScaleGroup based AWS console
- Chaos Monkey
  Robustness verification
- Latency Monkey
- Janitor Monkey
- Bakeries and AMI
- Build dynaslaves

©Rettet kan ikke visa.
Computeren har muligvis ikke
hukommelse nok til at åbne
billedet, eller billedet er
muligvis blevet beskadiget.
Genstart computeren, og åbn
derefter filen igen. Hvis det
Benchmarks and Scalability
Scalability from 48 to 288 nodes on AWS


**Client Writes/s by node count – Replication Factor = 3**

- Used 288 of m1.xlarge
  - 4 CPU, 15 GB RAM, 8 ECU
- Cassandra 0.86
- Benchmark config only existed for about 1hr

---

**DataStax**

ONE MILLION WRITES/SEC in the CLOUD? That’s Fast.

APACHE CASSANDRA™ Fast.
“Some people skate to the puck, I skate to where the puck is going to be”

Wayne Gretzky
Cassandra on AWS

<table>
<thead>
<tr>
<th>The Past</th>
<th>The Future</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance: m2.4xlarge</td>
<td>Instance: hi1.4xlarge</td>
</tr>
<tr>
<td>Storage: 2 drives, 1.7TB</td>
<td>Storage: 2 SSD volumes, 2TB</td>
</tr>
<tr>
<td>CPU: 8 Cores, 26 ECU</td>
<td>CPU: 8 HT cores, 35 ECU</td>
</tr>
<tr>
<td>RAM: 68GB</td>
<td>RAM: 64GB</td>
</tr>
<tr>
<td>Network: 1Gbit</td>
<td>Network: 10Gbit</td>
</tr>
<tr>
<td>IOPS: ~500</td>
<td>IOPS: ~100,000</td>
</tr>
<tr>
<td>Throughput: ~100Mbyte/s</td>
<td>Throughput: ~1Gbyte/s</td>
</tr>
<tr>
<td>Cost: $1.80/hr</td>
<td>Cost: $3.10/hr</td>
</tr>
</tbody>
</table>
Cassandra Disk vs. SSD Benchmark

Same Throughput, Lower Latency, Half Cost
Get stuck with wrong config
Wait
Wait
Wait
File tickets
Ask permission
Wait
Wait
Wait
Things we don’t do
Run out of space/power
Plan capacity in advance
Wait
Have meetings with IT
Wait
Things we do do.

Run benchmarks.

Live on stage.

(but this time it’s a recording).
Live Demo Workload

• Jenkins automation
  – Jmeter load driver
  – Asgard provisioning
  – Priam instance management

• Traffic
  – Reading/writing whole 100 column rows
  – Randomly selected from 25M row keys
  – Run for 10 minutes, then double ring size
Screenshots from Live Demo

Cassandra Summit 2012 last July
Live on stage demo
Asgard

cass_perf apps, with no instances running
# Jmeter Setup

## Build parameters

### Build #4

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORKLOAD_COMMENT</td>
<td>Add a comment to describe what this run is doing</td>
</tr>
<tr>
<td>PROFILE</td>
<td>parallel</td>
</tr>
<tr>
<td>DURATION</td>
<td>1800</td>
</tr>
<tr>
<td>INSTANCES</td>
<td>20</td>
</tr>
<tr>
<td>JMETER_EXPERIMENT</td>
<td>cass_write_read_rows.jmx</td>
</tr>
<tr>
<td>JMETER_PROPERTY</td>
<td></td>
</tr>
<tr>
<td>REMOVE_JMETER_INSTANCES</td>
<td>1</td>
</tr>
<tr>
<td>CSV_LIST</td>
<td>active1a.csv, active1b.csv, active2a.csv, active2ab.csv</td>
</tr>
<tr>
<td>SPLIT_CSV</td>
<td>1</td>
</tr>
</tbody>
</table>
## Jmeter Setup

### Build parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPLIT_CSV</td>
<td>1</td>
</tr>
<tr>
<td>THREAD_LIST</td>
<td>20</td>
</tr>
<tr>
<td>LOOP_COUNT</td>
<td>timed</td>
</tr>
<tr>
<td>REGION</td>
<td><code>us-east-1</code></td>
</tr>
<tr>
<td>TARGET_ASG</td>
<td><code>cass_perf_test,0.11</code></td>
</tr>
<tr>
<td>JMETIER_SAMPLE_NAMES</td>
<td><code>Cassandra_Batch_Put,Cassandra_Get_Range_Slice</code></td>
</tr>
<tr>
<td>MERGE_DATA</td>
<td>0</td>
</tr>
<tr>
<td>EPIC_CONFIG</td>
<td><code>cass_scale_list,cluster:cass_perf_test@us-east1.awastest+cass_scale_list,cluster:cass_perf_test@us-east1.awastest+cass_scale_list</code></td>
</tr>
<tr>
<td>NODESTATS</td>
<td><code>cass_perf_test,0.3</code></td>
</tr>
<tr>
<td>COLLECT_PROFILE</td>
<td><code>none</code></td>
</tr>
</tbody>
</table>

CSV files to copy over from data directory. Can be "none". By default copied from depot/cloud/performance/jmeter-test-harness/performance/data

Split the CSV files evenly across the clients

List of thread counts for each experiment eg 16 20 30. Experiments will be run in order

Type of experiment - "timed", "forever" or number_of_iterations

Region to run the test in

The service ASG that we are targeting performance. Will create a servers.csv file with a list of instances in this service. Parameter is 

List of Samplers in the JMeter experiment, this list will be post processed and plotted

Should we merge the JMeter results from all the client machines

List of epic to cull at the end of the run parameter format list1,cluster:list2:cluster:... list format: output_file_title:counter1...:counterN

Collect stats on Target ASG. Parameter is , or none

Collect Studio profile on node. Parameter is , or none
# Jmeter Setup

## Build parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLLECT_HEAP_ALLOCATIONS</td>
<td>none</td>
</tr>
<tr>
<td>CLEANUP_SCRIPT</td>
<td>none</td>
</tr>
<tr>
<td>TARGET_DEPLOY</td>
<td>ami-g9f85000 hi1.4xlarge 1 1 create_keyspace</td>
</tr>
<tr>
<td>CASSANDRA</td>
<td>1</td>
</tr>
<tr>
<td>JMETTER_INSTANCE_TYPE</td>
<td>m2.2xlarge</td>
</tr>
<tr>
<td>TARGET_TEARDOWN</td>
<td>1</td>
</tr>
<tr>
<td>PRESERVE_RUN</td>
<td>0</td>
</tr>
<tr>
<td>PARALLEL_SCRIPT</td>
<td>800: cass_perf_test.create_double_ring</td>
</tr>
<tr>
<td>DELTA</td>
<td>240</td>
</tr>
<tr>
<td>REPLAY_BUILD</td>
<td>jmeter_client-22_cassandra_test-54</td>
</tr>
</tbody>
</table>
Asgard

Initial set of cass instances up and running

<table>
<thead>
<tr>
<th>Name</th>
<th>ASGs</th>
<th>Instances</th>
<th>Type</th>
<th>Description</th>
<th>Email</th>
<th>Owner</th>
<th>Create Time</th>
<th>Update Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>cass_perf_m2_test</td>
<td>3</td>
<td>12</td>
<td>Web Service</td>
<td>Test Cassandra perf</td>
<td></td>
<td>Denis</td>
<td>2012-06-07 15:37:37 PDT</td>
<td>2012-06-07 15:37:37 PDT</td>
</tr>
<tr>
<td>cass_perf_sub</td>
<td>0</td>
<td>0</td>
<td>Web Service</td>
<td>Test Subscriber with live data</td>
<td></td>
<td>Denis Sheahan</td>
<td>2011-06-15 17:02:26 PDT</td>
<td>2011-06-15 17:02:26 PDT</td>
</tr>
<tr>
<td>cass_perf_test</td>
<td>3</td>
<td>12</td>
<td>Standalone Application</td>
<td>Cassandra performance test</td>
<td></td>
<td>Denis</td>
<td>2012-06-03 06:50:13 PDT</td>
<td>2012-06-03 06:50:13 PDT</td>
</tr>
</tbody>
</table>
Kiklos (open source soon)

Clusters growing from 12 to 24

in-service, bootstrapping, garbage-collecting, cass-down
Kiklos

Clusters growing from 12 to 24

in-service, bootstrapping, garbage-collecting, cass-down
Disclaimers

• We didn’t have time to tune the demo
• These are the plots from the live demo run
• Run’s need to be longer to get to steady state
• Data size only reached around 5GB per node
• Plenty of “I wonder why it did that” remains
• It’s a fair comparison, but not the best absolute performance possible for this workload and configuration
• When you remove the IO bottleneck, the next few bottlenecks appear...
Activity during the talk 10:30-11:30

Custom AppDynamics dashboard showing CPU and IOPS per node
Jmeter Plots

• Plots are the output of the Jenkins build

• Each instance has its own set of plots

• Each availability zone has its own summary plots

• One of the three zone summary plots is compared for each metric

• Plot collection is currently duplicated as we are transitioning from “Epic” to “Atlas”
Jenkins

Collected results and graphs after job has completed
Billedet kan ikke vises. Computeren har muligvis ikke hukommelse nok til at åbne billedet, eller billedet er muligvis blevet beskadiget. Genstart computeren, og åbn derefter filen igen. Hvis det

The past
m2.4xlarge

Instances per zone

The future
hi1.4xlarge

Cass Instances

Frame: PT31M, End: 2012-08-08T11:18 PDT, Step: PT1M
The past m2.4xlarge

Transactions per zone, same as total client transactions

The future hi1.4xlarge
Billedet kan ikke vises. Computeren har muligvis ikke hukommelse nok til at åbne billedet, eller billedet er muligvis blevet beskadiget. Genstart computeren, og åbn derefter filen igen. Hvis det

The past

m2.4xlarge

Cassandra Biggest Compaction

- Customers Biggest Compaction
  - Max : 3.983 G
  - Min : 1.069 G
  - Avg : 2.432 G
  - Last : 2.021 G

Frame: PT31M, End: 2012-08-08T11:18 PDT, Step: PT1M

The future

hi1.4xlarge

Cassandra Biggest Compaction

- Customers Biggest Compaction
  - Max : 5.611 G
  - Min : 0.652 G
  - Avg : 3.251 G
  - Last : 5.611 G

Frame: PT31M, End: 2012-08-08T11:19 PDT, Step: PT1M
Billedet kan ikke vises. Computeren har muligvis ikke hukommelse nok til at åbne billedet, eller billedet er muligvis blevet beskadiget. Genstart computeren, og åbn derefter filen igen. Hvis det

The past
m2.4xlarge

Thousands of
Microseconds

The future
hi1.4xlarge

Frame: PT31M, End: 2012-08-08T11:18 PDT, Step: PT1M

Cassandra Read Latency

- Customers read latency
  - Max: 56.320 k
  - Min: 0.274 k
  - Avg: 16.141 k
  - Last: 1.252 k

Frame: PT31M, End: 2012-08-08T11:18 PDT, Step: PT1M

Cassandra Read Latency

- Customers read latency
  - Max: 774.138 k
  - Min: 53.340 k
  - Avg: 343.365 k
  - Last: 342.040 k

Frame: PT31M, End: 2012-08-08T11:19 PDT, Step: PT1M
The past
m2.4xlarge

Cass CPU Usage

- **user and system cpu**
- **Max:** 44.500  **Min:** 1.000
- **Avg:** 30.094  **Last:** 24.500

Frame: PT31M, End: 2012-08-08T11:18 PDT, Step: PT1M

The future
hi1.4xlarge

Cass CPU Usage

- **user and system cpu**
- **Max:** 49.750  **Min:** 1.000
- **Avg:** 32.097  **Last:** 22.500

Frame: PT31M, End: 2012-08-08T11:18 PDT, Step: PT1M
Billedet kan ikke vises. Computeren har muligvis ikke hukommelse nok til at åbne billedet, eller billedet er muligvis blevet beskadiget. Genstart computeren, og åbn derefter filen igen. Hvis det

The past
m2.4xlarge

The future
hi1.4xlarge
Next Steps

• Migrate Production Cassandra to SSD
  – Several clusters done
  – ~100 SSD nodes running

• Autoscale Cassandra using Priam
  – Cassandra 1.2 Vnodes make this easier
  – Shrink Cassandra cluster every night
Skynet

A Netflix Hackday project that might just terminate the world...

(hack currently only implemented in Powerpoint – luckily)
The Plot (kinda)

• Skynet is a sentient computer that defends itself when people try to turn it off

• Connor is the guy who eventually turns it off

• Terminator is the robot sent to kill Connor
The Hacktors

• Cass_skynet is a Cassandra cluster that detects that it is being attacked and responds
• Connor_monkey kills cass_skynet nodes
• Terminator_monkey kills connor_monkey nodes
The Hacktion

• Cass_skynet stores a history of its world and action scripts that trigger from what it sees

• Action response to losing a node
  – Auto-replace node and grow cluster size by one

• Action response to losing more nodes
  – Replicate cluster into a new zone or region

• Action response to seeing a Connor_monkey
  – Startup a Terminator_monkey in that zone
Implementation (plan)

- Priam
  - Autoreplace missing nodes
  - Grow cass_skynet cluster
  - Increase replication to new zone (new code)
  - Increase replication to new region (new code)
- Cassandra Keyspaces
  - Actions – scripts to be run
  - Memory – record event log of everything seen
- Cron job once a minute
  - Extract actions from Cassandra and execute
- Chaos Monkey configuration
  - Terminator_monkey: pick a zone, kill any connor_monkey
  - Connor_monkey: kill any cass_skynet or terminator_monkey
Computeren har muligvis ikke hukommelse nok til at åbne billedet, eller billedet er muligvis blevet beskadiget. Genstart computeren, og åbn derefter filen igen.
Takeaway

*Netflix has built and deployed a scalable global platform based on Cassandra and AWS.*

*Key components of the Netflix PaaS are being released as Open Source projects so you can build your own custom PaaS.*

*SSD’s in the cloud are awesome….*

http://github.com/Netflix
http://techblog.netflix.com
http://slideshare.net/Netflix

http://www.linkedin.com/in/adriancockcroft
@adrianco #netflixcloud #cassandra12