ANALYZING AND TUNING CHINA MOBILE'S OPENSTACK PRODUCTION CLOUD

Tech Tip

October 2016

Photo from China Mobile Suzhou Data Center

Authors: Yingxin Cheng (Intel), Malini Bhandaru (Intel), Junwei Liu (China Mobile)

CONTENTS

FIGURES

TABLES

1 INTRODUCTION

Developers, architects and operators can face challenges when analyzing OpenStack* cloud software performance. Those who develop, deploy and operate cloud infrastructures must ensure their clouds meet service-level agreements (SLAs). Developers want to quickly locate and resolve design and implementation issues that limit cloud scaling. Deployers want to set up services, networks, and nodes to meet scale needs. Operators also want to establish the real capacity of their cloud under heavy workload conditions. Organizations considering deploying a private cloud need data to make educated decisions about whether OpenStack solutions can meet their needs.

OpenStack black-box testing often uses the rally [1] project, but for fine grained analysis, constant polling is required which would introduce an artificial load on the system and muddy the analysis. An alternative is to simulate the test system. While attractive in needing less hardware, it however requires accurate modelling of inter-component interactions and the time/resources spent in each component. For this case study, we were fortunate to gain access to China Mobile's production environment before it went live, which allowed us to test on real hardware. We then introduced limited extra logging to minimize any artificial system load.

This paper shares lessons learned from a careful, component level analysis of China Mobile's 1000-node OpenStack cloud. We uncovered three major issues in the course of this study, all addressed through OpenStack configuration changes. The result: a more stable and performant China Mobile OpenStack cloud and insights into scale bottlenecks and areas for future work.

2 PERFORMANCE STUDY – WHAT, WHY AND HOW?

2.1 CHINA MOBILE 1000-NODE PRODUCTION CLUSTER

China Mobile’s 1000 node production cloud runs **OpenStack Newton** RC2 (13.0.0.rc2). The cloud includes 530 compute hosts, five controller nodes, a three-node keystone cluster for identity and authentication services, a three-node active-active Galera Cluster for MySQL database services, a three-node RabbitMQ cluster for inter-component messaging support, and the remaining nodes set-up for storage and network services. Note: because the cloud runs on multi-core machines, multi-threaded processes can consume more than 100% of CPU.

Server configuration: 2 x Intel® Xeon® Processor E5-2680 v3 (30M Cache, 2.50 GHz), 128GB RAM (8 x 16GB HP® DDR4), 3TB Disk.

2.2 LOW OVERHEAD INSTRUMENTATION

Given the “slated-for-production” status of the cluster under test, we introduced additional logging, taking care to minimize overhead. That enabled us to further analyze the logs as a post processing step, without introducing latencies during execution.

Python-based OpenStack code lends itself to monkey-patching [3], enabling the easy introduction and elimination of monitoring instrumentation.

Even though Network Time Protocol was used to synchronize the nodes in the distributed cluster, its size made the synchronization precision inadequate to study fine-grained timing. To resolve the issue, we devised a method to adjust timestamps using causal constraint heuristics.

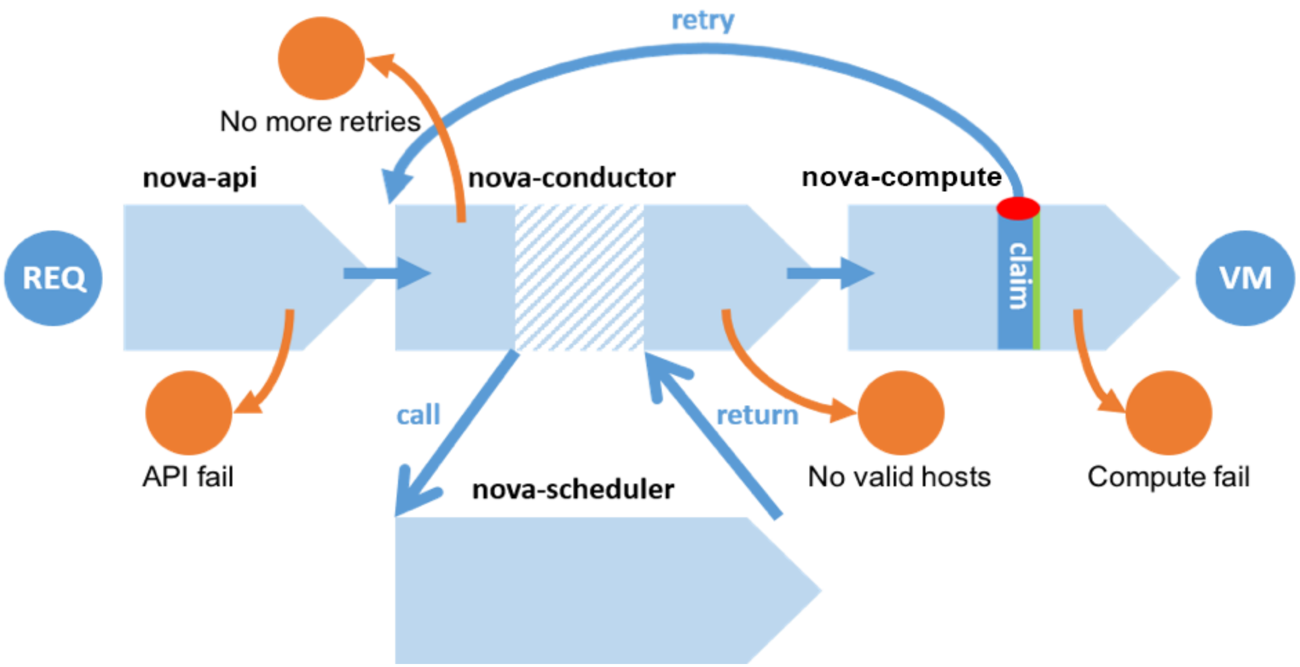
2.3 VIRTUAL MACHINE LAUNCH LATENCY

We focused on virtual machine (VM) launch latency as the most important feature of an Infrastructure-as-a-Service (IaaS) platform. Launch latency also involves most core OpenStack services and their components. Our work seeks to quantify performance as a function of the number of concurrent requests.

2.4 OPENSTACK NOVA ARCHITECTURE

To facilitate future discussions, we briefly delve into the OpenStack nova architecture which handles VM provisioning. Incoming requests first arrive at the nova-api service, which in turn hands off the task to the nova-conductor service, a layer of software that envelopes the database. The nova-conductor invokes the nova-scheduler, which attempts to identify a hospitable host for the workload. Once it identifies a host, the scheduler transfers control to the nova-compute service running on the selected host to launch the workload. The orange state transitions in **Figure 2-1** (below) represent error states that might stem from incorrectly formulated requests, input constraints that cannot be satisfied and/or a lack of resources:

FIGURE 2-1: ARCHITECTURE



2.5 EXPERIMENTS

We study performance under various load conditions, in particular handling concurrent VM launch requests ranging from five to 2000.

We further explore performance under the same load conditions but using different types and numbers of nova-scheduler services, to handle the volume of requests through horizontal scaling.

Using Zabbix [5], an open source monitoring tool, we study system resource consumption across the various cloud components with respect to time once requests enter the system.

Most valuable of all, we analyzed the logs over four separate runs, each a 20-minute time window with 2000 concurrent requests. We collected data on how many requests completed successfully, how many failed, and in what manner.

In the next section we share our findings.

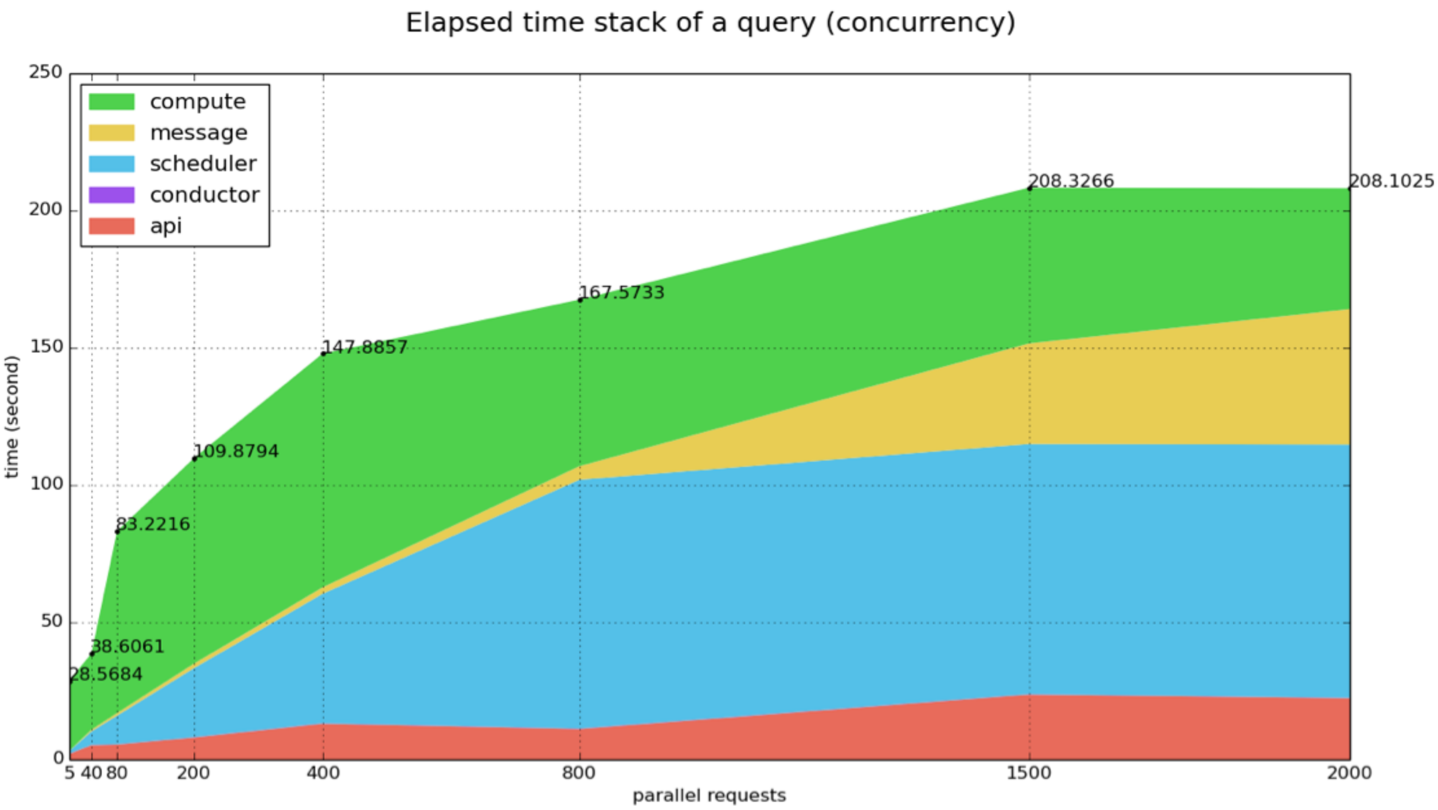
3 PERFORMANCE TEST INSIGHTS

3.1 EXECUTION TIMES UNDER VARYING REQUEST PRESSURE

Figure 3-1 shows nova component-level costs when the cloud handles five to 2000 concurrent launch requests. Comparing our results to those of an earlier study [2] with simulated compute hosts, we found major differences with respect to the cost in compute services and the scheduler saturation point. Real compute nodes—i.e., real hypervisors—take significantly longer to spawn VMs than we assumed, driving up the compute-host time component. Observing best practices for production environments, the message queue and data base services were deployed in clustered configurations to deliver the required stability and performance.

Note that the scheduler service time increases with load, but plateaus around 800 concurrent requests. As the number of concurrent requests rises beyond 800, saturating the scheduler, the cost of message-handling increases. While the nova-api service time rises with load, it never consumes more time than the scheduler component. The cloud succeeds in handling 1.78 requests per second.

FIGURE 3-1: SEND 5 ~ 2000 CONCURRENT REQUESTS TO 530 COMPUTE NODES



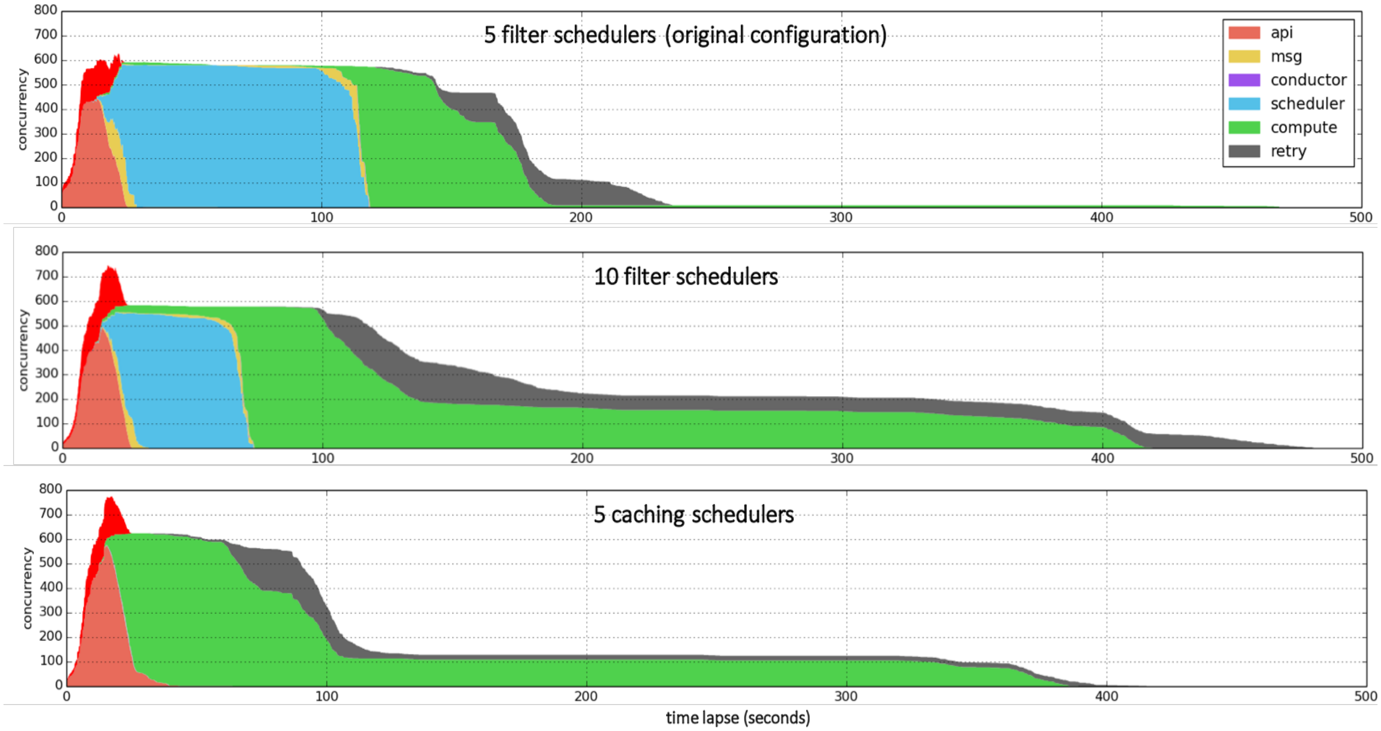
3.2 EXECUTION TIMES UNDER VARYING SCHEDULERS

Next, we explore the effect of five scheduler instances, versus 10 scheduler instances, versus five instances of a new caching scheduler. The caching scheduler eliminates the need to visit the database to retrieve resource usage updates prior to scheduling.

The graphs in **Figure 3-2** below support our hypothesis: using 10 scheduler instances instead of five roughly halves the time to tackle launch requests. As one might expect, using a caching scheduler radically shrinks the time for a host to schedule a workload. But introducing more scheduler instances impacts the nova-api component by causing more re-tries stemming from using stale resource views.

The graph’s red regions indicate failed schedule attempts in nova-api, and the dark grey regions indicate the time spent in re-tries—puzzling, given the availability of physical resources on the compute hosts. With five filter schedulers, the failure rate reached 25.7 percent and the retry rate reached 29.0 percent. Further, the time to spawn VM images fluctuates more with a larger number of schedulers—sometimes swinging from 6.45 seconds to 354 seconds (nearly six minutes), which begs further investigation.

FIGURE 3-2: VARYING SCHEDULER SETTINGS

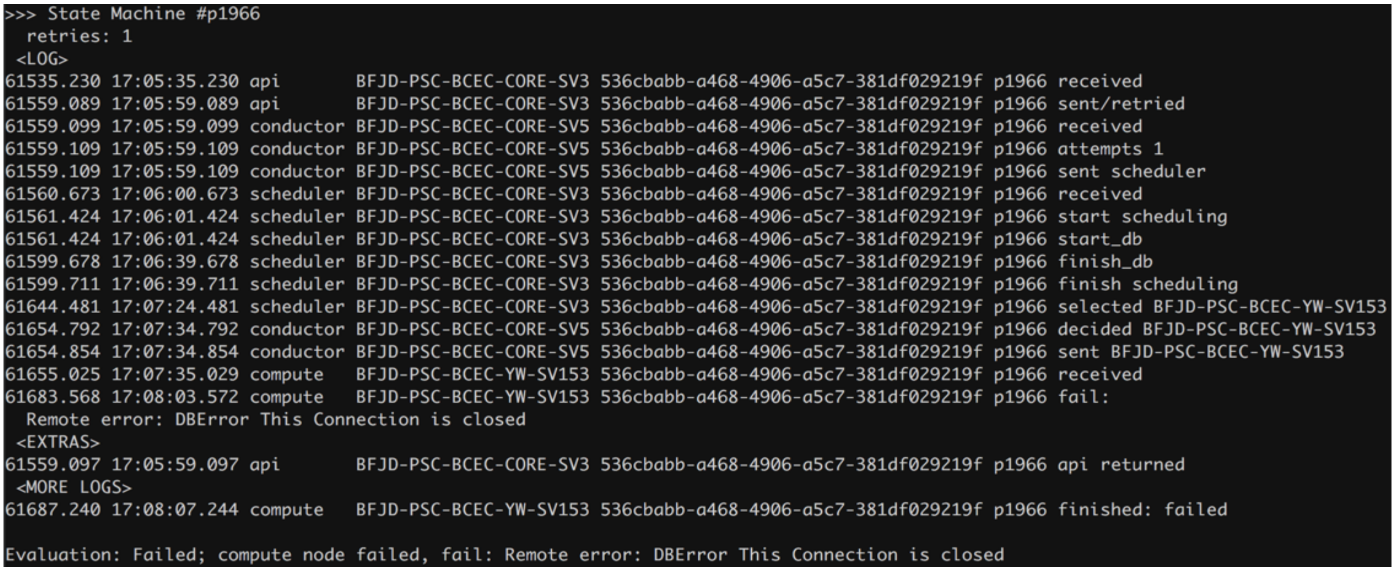


3.3 LAUNCH ERRORS – NUMBER AND NATURE

To better understand the launch process and the nature of errors, we analyzed detailed logs obtained from four runs of 2000 requests each. Of the 8000 total requests, 7953 were successfully tracked, with 4646 successfully completing, yielding a success rate of only 58.45 percent.

First, to obtain the necessary time synchronization precision, we adjusted log timestamps using causality heuristics. Then we developed a detailed state machine using log events such as enter-nova-api, exit-nova-api, and others. We next traced each event in this state space and classified the errors using their message strings.

FIGURE 3-3: TRACE FAILURE REQUESTS



```
>>> State Machine #p1966
  retries: 1
 <LOG>
61535.230 17:05:35.230 api       BFJD-PSC-BCEC-CORE-SV3 536cbabb-a468-4906-a5c7-381df029219f p1966 received
61559.089 17:05:59.089 api       BFJD-PSC-BCEC-CORE-SV3 536cbabb-a468-4906-a5c7-381df029219f p1966 sent/retried
61559.099 17:05:59.099 conductor BFJD-PSC-BCEC-CORE-SV5 536cbabb-a468-4906-a5c7-381df029219f p1966 received
61559.109 17:05:59.109 conductor BFJD-PSC-BCEC-CORE-SV5 536cbabb-a468-4906-a5c7-381df029219f p1966 attempts 1
61559.109 17:05:59.109 conductor BFJD-PSC-BCEC-CORE-SV5 536cbabb-a468-4906-a5c7-381df029219f p1966 sent scheduler
61560.673 17:06:00.673 scheduler BFJD-PSC-BCEC-CORE-SV3 536cbabb-a468-4906-a5c7-381df029219f p1966 received
61561.424 17:06:01.424 scheduler BFJD-PSC-BCEC-CORE-SV3 536cbabb-a468-4906-a5c7-381df029219f p1966 start scheduling
61561.424 17:06:01.424 scheduler BFJD-PSC-BCEC-CORE-SV3 536cbabb-a468-4906-a5c7-381df029219f p1966 start_db
61599.678 17:06:39.678 scheduler BFJD-PSC-BCEC-CORE-SV3 536cbabb-a468-4906-a5c7-381df029219f p1966 finish_db
61599.711 17:06:39.711 scheduler BFJD-PSC-BCEC-CORE-SV3 536cbabb-a468-4906-a5c7-381df029219f p1966 finish scheduling
61644.481 17:07:24.481 scheduler BFJD-PSC-BCEC-CORE-SV3 536cbabb-a468-4906-a5c7-381df029219f p1966 selected BFJD-PSC-BCEC-YW-SV153
61654.792 17:07:34.792 conductor BFJD-PSC-BCEC-CORE-SV5 536cbabb-a468-4906-a5c7-381df029219f p1966 decided BFJD-PSC-BCEC-YW-SV153
61654.854 17:07:34.854 conductor BFJD-PSC-BCEC-CORE-SV5 536cbabb-a468-4906-a5c7-381df029219f p1966 sent BFJD-PSC-BCEC-YW-SV153
61655.025 17:07:35.029 compute   BFJD-PSC-BCEC-YW-SV153 536cbabb-a468-4906-a5c7-381df029219f p1966 received
61683.568 17:08:03.572 compute   BFJD-PSC-BCEC-YW-SV153 536cbabb-a468-4906-a5c7-381df029219f p1966 fail:
  Remote error: DBError This Connection is closed
 <EXTRAS>
61559.097 17:05:59.097 api       BFJD-PSC-BCEC-CORE-SV3 536cbabb-a468-4906-a5c7-381df029219f p1966 api returned
 <MORE LOGS>
61687.240 17:08:07.244 compute   BFJD-PSC-BCEC-YW-SV153 536cbabb-a468-4906-a5c7-381df029219f p1966 finished: failed

Evaluation: Failed; compute node failed, fail: Remote error: DBError This Connection is closed
```

Figure 3-3 highlights a failed request in a 2000 request test run. The request-ID, “p1966” failed on compute node “YW-SV153” with the error logged as “loss of database connection.” The log files revealed that it occurred when the compute node tried to update its resource usage through nova-conductor. Further analysis revealed that nova-conductor ran into a database deadlock issue when it ran the “compute node update” operation. The logs show that after the request is received by the compute node, it takes approximately 30 seconds before the database operation fails, characteristic of load related failure.

The following table shows each error and its category and sub-categories based on the error message string. The state machine-based analysis helped eliminate duplicate and irrelevant error messages.

TABLE 1: ERRORS AND CATEGORIES

Component	Error Message	Count
API	This result object does not return rows.	2284
API	Not authorized for image	845
API	HTTP Internal Server Error (HTTP 500)	133
API	Unknown	8
Compute Node	neutron error creating port on network, retry	1222
Compute Node	Unknown, retry	12
Compute Node	Build of instance … Failure prepping block device	5
Compute Node	Build of instance … Failed to allocate the network(s), not rescheduling	1
Compute Node	Remote error: DB Error This connection is closed	1

3.3.1 TOP THREE ERRORS

- **Database deadlocks:** Deadlocks caused the majority of database failures reported by the nova-api service and occurred when running the “quota reserve” operation. After reaching the deadlock retry limit, the request failed with the error message “no rows returned.”
- **neutron port creation failure:** A substantial portion of the retries stemmed from the OpenStack neutron service failing to allocate a port for a VM. This issue caused at least 1222 compute-node retries and one compute-node failure.
- **keystone authentication failure:** Every stage in the OpenStack service pipeline attempts to authenticate the request with the OpenStack identity service, keystone. Under heavy load, authentication often times out. Of the 8000 total requests, this accounted for 47 failures to authenticate due to time-outs and ultimately caused 845 image access errors and 133 HTTP internal server errors.

3.4 CONFIGURATION FIXES

Further investigation revealed that the three major types of errors stemmed from misconfigurations and improper deployment decisions—not from software bugs:

- **Galera Cluster Configuration**—Misconfiguration of the Galera Cluster causes the database deadlocks. The original configuration used cluster-wide optimistic locking, resulting in failure and roll-back of the distributed quota reservations in the nova-api services, particularly as the launch request pressure increased. Using a single-node write strategy instead of allowing writes to all the Galera nodes resolved the issue.
- **Using Fernet tokens instead of PKI tokens in keystone**—Fernet tokens, compared to PKI tokens, perform better because of their smaller payload and non-persistent implementation. Moving to Fernet tokens resolved the authentication failures.
- **Setting "scheduler host subset size"** [4] to more than the number of schedulers decreases the chances of conflict and spreads the workload among compute nodes.
- **Setting "rpc response timeout"** to a longer interval allows the nova-conductor to wait longer for the RPC response from the nova-schedulers in a large cluster.
- **Increasing "rpc conn pool size"** allows the nova-scheduler service to concurrently accept more requests.
- **Increasing "max pool size" and "max overflow"** in the nova-api database and in the database configuration groups (given that the scheduler service directly connects to the database) allows a larger number of simultaneous connections to databases.

FIGURE 3-4: SERVICE QUALITY IMPROVEMENTS, BEFORE AND AFTER

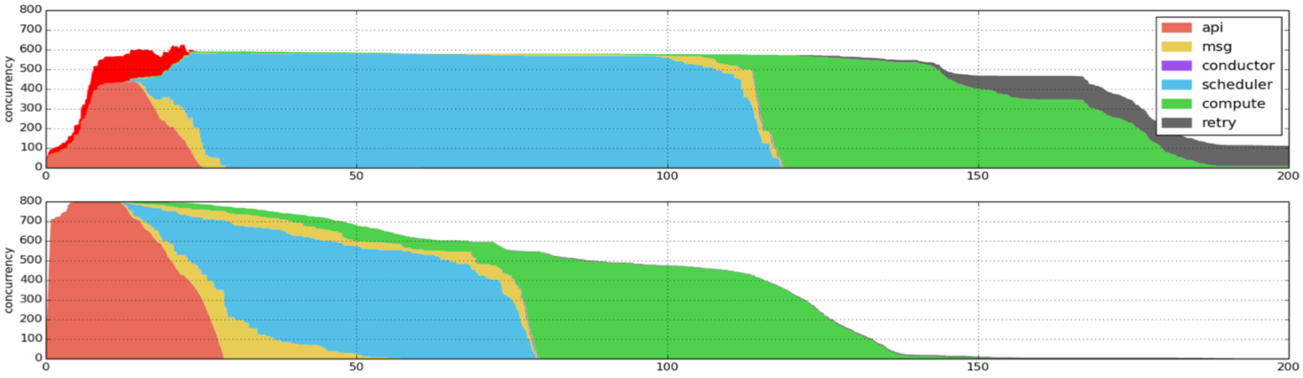


Figure 3-4 illustrates the overall improvements after troubleshooting and tuning. Resolving the three major root causes eliminated all API failures (in red) and unexpected retries (in grey). More randomly distributing workloads improved stability, including reducing the probability of port allocation time-out. Time spent in nova scheduling—thanks to switching to a single-node write strategy in the Galera Cluster—resulted in fewer time-outs, fewer rollbacks, and fewer retries.

- **The Failure rate** in nova-api decreased from 25.7 percent to 0 percent.
- **The Retry rate** decreased from 29.0 percent to 0.83 percent
- **The Overall Output** increased from 1.31 requests/second to 4.35 requests/second

3.5 SERVICE LEVEL PROFILING

For service level profiling we used Zabbix, an open source monitoring tool with a variety of plugins to monitor software applications such as MySQL database and RabbitMQ, in addition to being able to monitor system-level activities of individual processes. We set up monitoring agents on the servers hosting the OpenStack controllers, keystone (identity), the database, and the message queue. We examined usage under various loads, namely idle, 800, and 2000 concurrent requests in our 1000 node cluster. Our purpose in sharing this data is to illustrate what one might anticipate in terms of resource consumption in a 1000 node cluster, in order to guide hardware sizing efforts.

TABLE 2: OPENSTACK COMPONENTS' PEAK CONSUMPTION OF CPU, MEMORY, NETWORK RESOURCES

Item	Idle (min value)	800 requests (peak value)	2000 requests (peak value)
Nova CPU usage (nova-api)	58%	172%	219%
Nova CPU usage (nova-scheduler)	22%	95%	95%
Nova CPU usage (nova-conductor)	81%	270%	329%
Nova Memory usage (nova-api)	60GiB		
Nova Memory usage (nova-scheduler)	1.5GiB		
Nova Memory usage (nova-conductor)	16GiB		
Nova Incoming network (controller node)	527Kbps	35Mbps	41Mbps
Nova Outgoing network (controller node)	301Kbps	3.1Mbps	5.2Mbps
Keystone CPU usage	0%	169%	202%
Keystone Memory usage	68GiB		
Keystone Incoming network (controller node)	27Kbps	6.2Mbps	12Mbps
Keystone Outgoing network (controller node)	86Kbps	3.1Mbps	8.6Mbps
Database MySQL bytes received	250KBps	9.3MBps	6.9MBps
Database MySQL bytes sent	400KBps	33MBps	39MBps
Database MySQL queries rate	92qps	40Kqps	57Kqps
Database Incoming network	1.5Mbps	32Mbps	49Mbps
Database Outgoing network	2.5Mbps	180Mbps	222Mbps
Messaging RabbitMQ receive rate	19msgs/s	1.7Kmsgs/s	2.1Kmsgs/s
Messaging RabbitMQ deliver rate	32msgs/s	6.4Kmsgs/s	7.5Kmsgs/s
Messaging RabbitMQ file descriptors used	8.2K		
Messaging RabbitMQ sockets used	8.1K		
Messaging Incoming network	6.5Mbps	71Mbps	94Mbps
Messaging Outgoing network	11.9Mbps	125Mbps	179Mbps

FIGURE 3-5: CONTROLLER, KEYSTONE, RABBITMQ AND MYSQL UTILIZATION AS A FUNCTION OF TIME

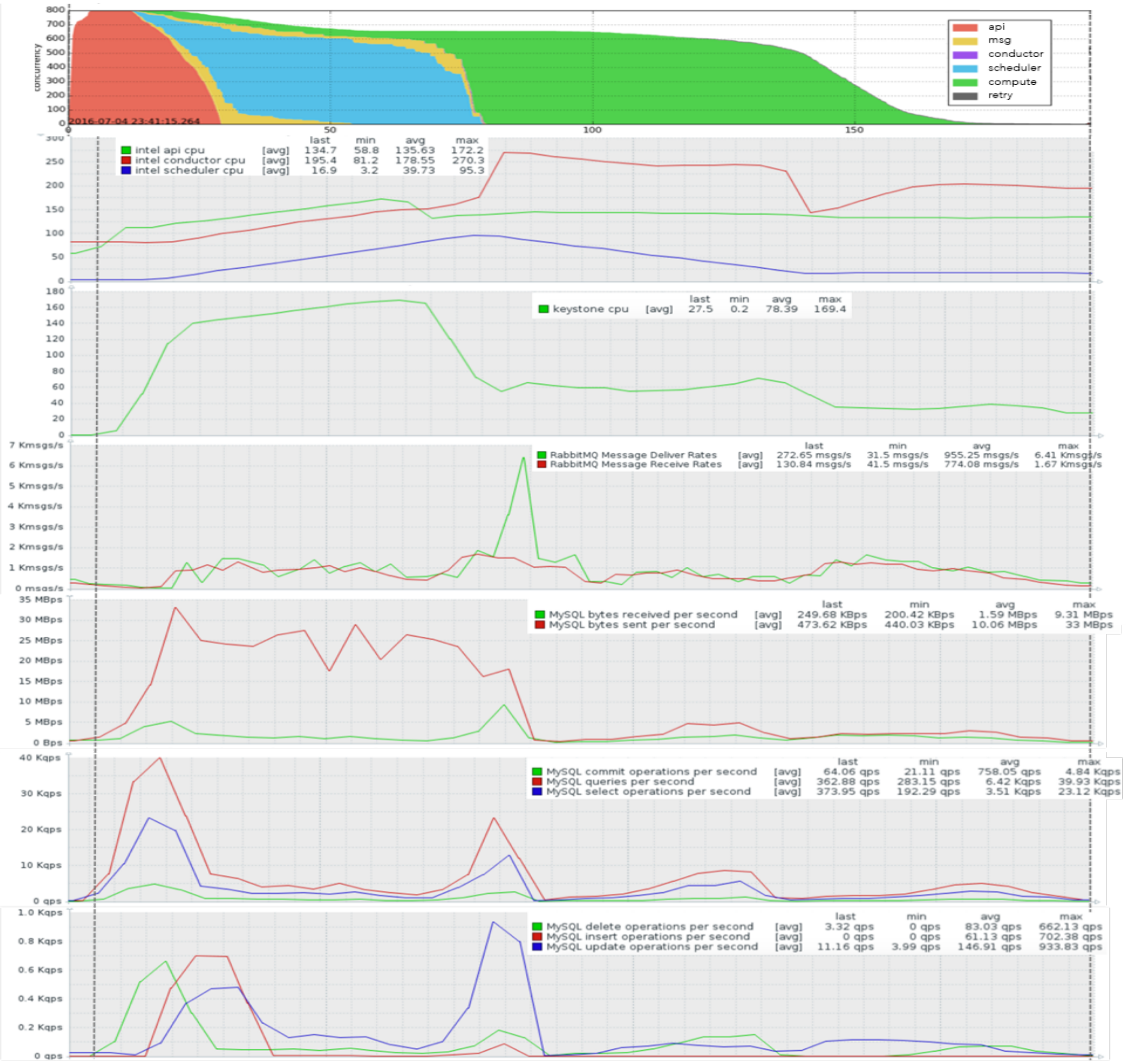


Figure 3-5 shows that the nova-api and keystone services consume processing resources as requests arrive into the system. As nova-scheduler resource requirements climb, they also shift further to the right in time. In addition, nova-scheduler directly communicates with the database to refresh its resource status cache, visible as a matched rise in database processing needs. The resource usage of nova-conductor rises only during the deployment phase (peaking at 270 percent), and thus it matches the rise of resource usage on the nova-compute nodes. Database resource usage remains high in the deployment phase because nova-conductor keeps it busy updating the compute node records. The deliver-rate and receive-rate of RabbitMQ start to rise after the first request leaves nova-api, and they peak when requests go to the compute nodes. The cloud ran on multi-core machines, allowing usage to exceed 100 percent. Note that because the nova-scheduler is inherently single-threaded, its CPU usage cannot exceed 100 percent.

3.6 ANALYSIS

The system profiling illustrates varied performance footprints of the nova services:

- **nova-api** imposes pressure upon keystone and MySQL services with numerous SQL operations, indicating a high degree of difficulty for performance optimizations due to complex interactions with other services.
- **nova-scheduler** configured with the filter scheduler driver requires outbound-intensive MySQL operations, and it cannot leverage multiple CPU cores. We began investigating a caching scheduler to reduce database burden and speed scheduling, and we have obtained promising preliminary results.
- **RabbitMQ pressure:** The decision delivery from nova-conductor to compute nodes imposes pressure upon RabbitMQ services, which seems inevitable. We propose investigating a lighter-weight messaging protocol.
- **VM provisioning by nova-compute** services relies on nova-conductor services to update the database. The number of nova-compute to nova-conductor services should match to avoid bottlenecks.

Zhen Yu Shen of China Mobile gives local media reporters a tour of one of several multi-thousand node, OpenStack clusters.

4 CONCLUSIONS AND FUTURE WORK

Our lightweight instrumentation along with technique for obtaining more precise time-synchronization enabled tracking of every VM launch request in nova. By analyzing each failure and classifying it, we identified three major issues and addressed them through configuration changes—obtaining a more stable and performant cloud.

We experimented with different cloud deployments and analyzed their merits. We confirmed that the filter scheduler creates the most significant bottleneck in a large OpenStack cloud, and that it stems from the cache-refresh design that imposes heavy pressure on the database. Switching to a caching scheduler relieves database pressure but also carries the risk of operating with stale resource views. More efficient database indexing mechanisms show promise as a way to reduce database pressure to prune the list of hosts before reaching out to the database. Alternatively, updating the schedulers by the compute nodes—on any resource change—boosts performance by eliminating the calls to update the scheduler database cached resource view on each schedule request. For those willing to relax the need to find the optimal host for scheduling, using cells, and/or partitioning the hosts into subsets can speed scheduling, especially with known short-running workloads. Other areas to explore include a lighter messaging component.

Our system-level profiling of OpenStack services showed patterns and exposed potential bottlenecks, and offers areas to explore for redesign and improvement.

References:

[1] https://wiki.openstack.org/wiki/Rally
[2] https://www.openstack.org/assets/presentation-media/7129-Dive-into-nova-scheduler-performance-summit.pdf
[3] http://en.wikipedia.org/wiki/Monkey_patch
[4] http://docs.openstack.org/mitaka/config-reference/compute/scheduler.html
[5] http://www.zabbix.com/

Acknowledgements:

Hao Li of China Mobile helped prepare the experiments.
Qing Wang and Jian-Feng Ding of Intel organized the collaboration with China Mobile.
Maggie Liang of Intel developed the case study with Dan Fineberg editing for publication.

5 LEGAL NOTICES

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at [intel.com].

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Testing methods and tools:

- Performance studied under varying load conditions, handling concurrent VM requests ranging from five to 2000.
- Performance also explored under the same load conditions but using different types and numbers of nova-scheduler services to handle the volume of requests through horizontal scaling.
- System resource consumption studied across the various cloud components using Zabbix [5], an open source monitoring tool.
- Log analysis over a 20-minute time window with 2000 concurrent requests. Data collected on how requests completed successfully, failed, and in what manner.

Configuration:

- The 1000-node experiments were carried out by the authors at China Mobile's datacenter.
  - The 1000 node cloud runs OpenStack Newton RC2 (13.0.0.rc2).
  - It includes 530 compute hosts, five controller nodes, a three-node keystone cluster for identity and authentication services, a three-node Galera Cluster for MySQL database services, a three-node RabbitMQ cluster for inter-component messaging support, and the remaining nodes set-up for storage and network services.
- Servers:
  - CPUs—2 x Intel® Xeon® Processor E5-2680 v3 (30M Cache, 2.50 GHz)
  - Memory—128GB RAM (8 x 16GB HP® DDR4)
  - Storage—3TB Local disk

For more information on performance of Intel technologies, go to www.intel.com/performance.
Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.
*Other names and brands may be claimed as the property of others.