# Executive Summary

# OpenStack users need large scale deployments, and certainty their environments will perform successfully at scale. High Availability protects users data and applications, providing certainty when deploying OpenStack services with Ansible. The limits of High Availability when users apply more nodes remains a less certain prospect, however. This document presents preliminary results for service availability testing at scale. The results indicate that OpenStack services can perform well at scale. The results provide certainty that OpenStack Ansible at scale can support users OpenStack environments.

# Introduction

This document presents preliminary results from service availability testing. Research and testing recruited a 22 node OpenStack cluster configured for high availability, and installed with OpenStack Ansible. Rally and a modified variant of OS Faults were used to perform testing (<https://github.com/osic/os-faults>). Rally simulated various operations that a customer environment would run on a daily basis.

OpenStack Users need to know their environments will perform under pressure. They need certainty that their large scale deployments will work under pressure, and will not experience downtime or data loss. High Availability protects against system downtime and data loss. Protecting against these two problems is valuable for a variety of OpenStack users. High availability means that failure of a single service should not cause an overall system failure. Successfully demonstrating High Availability works following node failure demonstrates to users certainty in running their OpenStack environments.

Some of the largest OpenStack cloud environments recruit large scale clusters for ecommerce, utilities, research, production, and development. Testing service availability and faults at scale, improves the reliability of an OpenStack environment deployed with Ansible. Improved reliability. Testing benefits OpenStack users who need large scale deployments. The research presented in this paper provides certainty through evidence that large scale OpenStack deployments work when experiencing failure without losing performance or data.

# Measurements of success in this test design was the amount of API calls, and overall service operation time. Fault injections simulated failure under pressure.

# Services tested in the project included Nova, Neutron, Cinder, Glance, and Swift.

# System failure was simulated with a plugin. The following sections on Environment and Test Tools contain more information. Nova commands experienced rapid failure, or experienced no change. Neutron commands resisted failures, and operations continued in some instances until eventually failing, but with operations coexisting with failures. Keystone services showed degradations, but began to recover somewhat following each degradation. Most of the Glance service commands and operations persisted following failures. Cinder volumes were also resistant to failures. Swift services showed a broad array of dispersion following the failure simulation, however each failure degradation was tempered with recovery in the majority of instances tested.

The research presented in this paper demonstrated that OpenStack deployed with ansible at large scale will persist under pressure. Results overall indicated mostly positive trends for performance. The following sections provide a breakdown of performance by service. Results are presented in graph format.

# Configuration

## Environment

The test environment is a 22 node OpenStack (Newton) cluster deployed using OpenStack Ansible. The deployment includes the following services: Nova, Neutron, Cinder, Glance, and Swift. Three Ubuntu images served as the basis for testing performance. These included; ubuntu 14.04.5, ubuntu 16.04.1, and centos 7, which were configured as the image for instances to use when testing the OpenStack Ansible services. Three flavors were also available. These flavors ran the core OpenStack service (the osic-baremetal-comp flavor), and the optional block storage (the osic-baremetal-object flavor), object storage, and Ceph services (the osic-baremetal-object and osic-baremetal-block flavor).

The environment physical and virtual components included a Deploy Box connected to OnMental cloud servers. Playbooks and templates generated configuration files when creating instances from the Deploy Box. OpenStack Ansible Playbooks and templates installed the OpenStack services, and set up the networking configurations. The OpenStack environment recruited stable branches of the Mitaka and the Newton release.

The technical specifications regarding deployment steps, tools used and software versions can be found in the following link: <https://github.com/osic/ref-impl/tree/master/onironic>.

## Test Tools

Rally was used to drive workload on the cluster, and faults were injected using the OS-Faults plugin. Minor changes were required to the OS-faults plugin to work with this particular deployment, as OS-Faults does not natively support OSA. Salt was installed and used by OS-Faults for running the tests on the cluster’s nodes.

## Test Overview

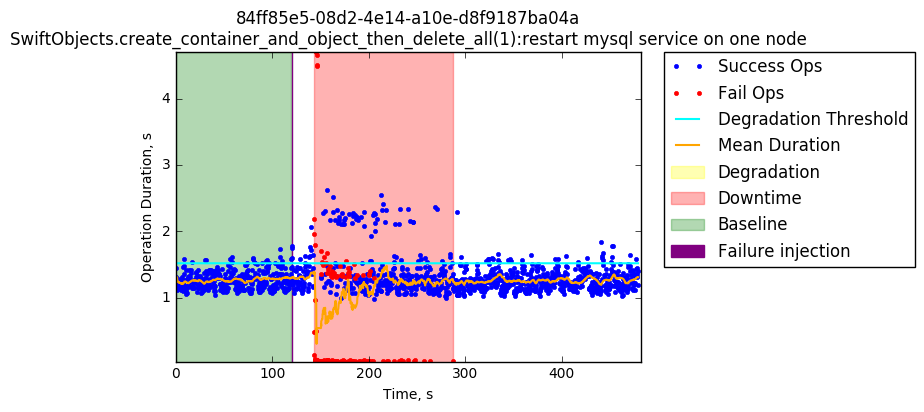
Tests targeted the high availability requirement that the failure of a single service should not cause a failure of the system. Designed to simulate the failure of specific services in the OpenStack environment, the subservices of each of the five primary services were individually either restarted, or killed. The expected behavior was that any service that was killed would be automatically restarted. API failures and performance were measured for tests that did not result in a permanent failure. While the system may enter a degraded state, it is expected that it would automatically return to a baseline state within a nominal amount of time (less than 5 minutes).

## Tested Scenarios

Services received testing for Keystone authentication, MySQL, Memcached, service metadata, and RabbitMQ. Tests specific to the Swift service included tests on the object updated, the proxy service, the object auditor, the object server, and swift container updater. Keystone services received disk and cpu testing. For the Glance service, the registry service received testing. Neutron testing involved killing and restarting the l3-agent, linuxbridge, and dhcp-agent. Nova service testing involved killing and restarting the nova-cert, nova-consoleauth, nova-conductor, and nova-scheduler.

# Results - The graphs

Graphs were created to aid in the analysis of the test results. These graphs were generated using the python package matplotlib (<http://matplotlib.org/> ) and jupyter notebook (<http://jupyter.org/>).

The following example graph provides a visual road map. An example of how to read the test and performance results:

The green section of the graph represents the first 120 seconds. During this time, Rally is running the load on the target system, but no fault has been injected yet. The thin purple line at the extreme left of the green section is the failure injection. This is when the fault was triggered. It is important to note that the time that the fault is triggered is not necessarily the time that it actually takes effect. Using Rally and OS-Faults, it can be tens of seconds between a fault trigger (such as killing a process), and the time when the fault actually takes effect. The time difference appears in the example graph. Approximately 15 seconds after the failure Injection line, we can see that some of the API calls start failing (as indicated by the red dots and red background). The title details displayed on this graph are:

* The Rally task ID, which is the first line of the graph’s title, and the Rally Scenario and injected fault, which is the second line of the graph title.
* The task duration in seconds is represented on the horizontal (x) axis. The Atomic operation duration is represented on the vertical (y) axis.

In the graph legend, the labels represented include:

* Success Ops: The successful API calls conducted by the OpenStack service.
* Fail Ops: The failed API calls experienced after failure injection.
* Degradation Threshold: the 95th percentile of the baseline sample, which is several standard deviations above the mean number of measured operations.
* Mean Duration: the overall operations running mean.
* Degradation: a peak in the performance. Anything above the degradation threshold .
* Downtime: the time frame between the first and last failed operation.
* Baseline: The set of operations taken from the beginning of the task execution until the failure injection.
* Failure injection: the second the failure was injected.

# Results - Summary

# Passing Tests

# Most of the scenarios showed small degradation peaks and were not affected significantly by the failure injection.

## Error Types Identified

### Services Fail to Recover

A severe error observed was a service failing to automatically restart after a complete test, including a failure injection. When the service received a failure injection, and would not restart automatically, a manual restart was required. Only after a manual restart would the OpenStack environment return to service. This type of error was seen specifically when killing Keystone, RabbitMQ, and MySQL.

It should be noted that the common trait connecting these services is that the code running the process is not managed by OpenStack. RabbitMQ and MySQL are third party opensource projects. Keystone, while a critical OpenStack project, is run under Apache.

### Unexpected or Unexplained Test Anomalies - potential quota limits and resource constraints.

Two classes of tests experienced behavior that requires further investigation. First, nearly all Cinder tests experienced an error that may be related exhausting resource limits, such as a quota. A root cause analysis and subsequent fix is necessary for this issue. Second, and an expected outcome due to the test type, API calls fail in the vast majority of neutron tests. Manifested as an decreasing API performance over time until a threshold is reached, this issue may be due to a resource constraints.

### Transient Failures after fault

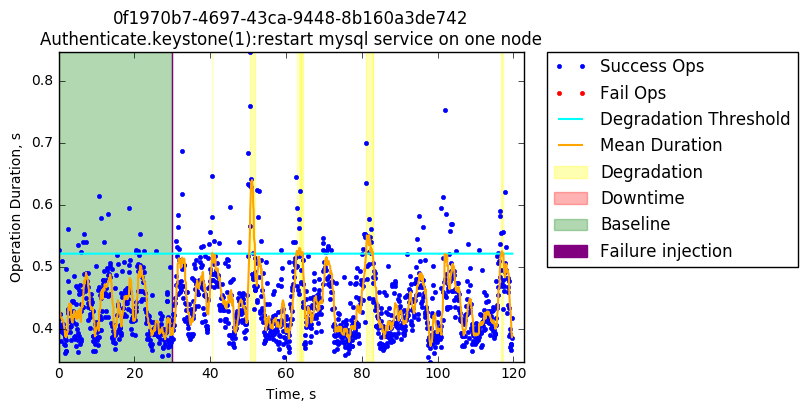
Many of the injected failures resulted in decreased performance, and some number of transient API failures immediately after the injected fault. Tests that restarted Keystone, killed or restarted Nova, or restarted MySQL experienced these failures.

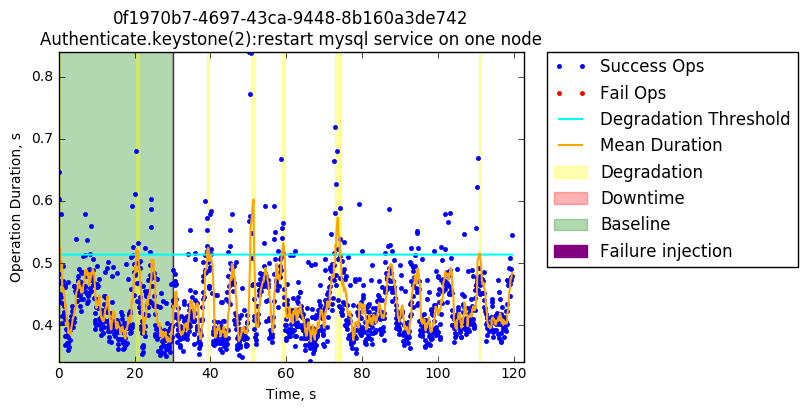
# Tested scenarios - Graph Results:

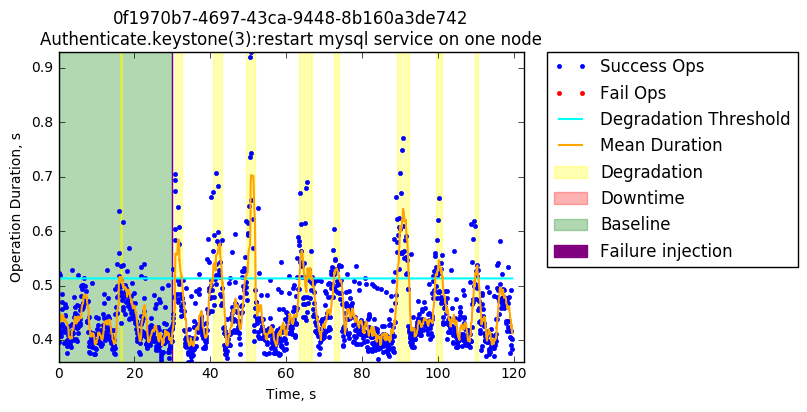
1. Authenticate.Keystone:
   1. Restart MySQL service on one node
   2. Kill memcached service on one node
   3. Stress memory on one node
   4. Restart memcached service on one node
   5. Stress disk on one node
   6. Stress cpu on one node
   7. Restart RabbitMQ service on one node
2. CinderVolumes.list\_volumes:
   1. Restart Keystone service on one node
3. GlanceImages.list\_images:
   1. Kill glance-api service on one node
   2. Restart RabbitMQ service on one node
   3. Restart Keystone service on one node
   4. Restart MySQL service on one node
   5. Restart glance-api service on one node
   6. Kill glance-registry service on one node
   7. Restart memcached service on one node
   8. Restart glance-registry service on one node
   9. Kill memcached service on one node
4. NeutronNetworks.create\_and\_list\_networks:
   1. Restart neutron-metadata-agent service on one node
   2. Kill neutron-metadata-agent service on one node
   3. Kill neutron-metering-agent service on one node
   4. Restart neutron-metering-agent service on one node
   5. Restart neutron-l3-agent service on one node
   6. Kill neutron-l3-agent service on one node
   7. Restart neutron-server service on one node
   8. Kill neutron-server service on one node
   9. Kill neutron-linuxbridge-agent service on one node
   10. Restart neutron-linuxbridge-agent service on one node
   11. Restart neutron-dhcp-agent service on one node
   12. Kill neutron-dhcp-agent service on one node
   13. Restart MySQL service on one node
5. NeutronNetworks.list\_agents:
   1. Restart Keystone service on one node
6. Nova Flavors.list\_flavors:
   1. Restart Keystone service on one node
7. NovaServers.boot\_and\_delete\_server:
   1. Restart memcached service on one node
   2. Kill nova-cert service on one node
   3. Restart nova-scheduler service on one node
   4. Restart nova-api-metadata service on one node
   5. Kill nova-consoleauth service on one node
   6. Restart RabbitMQ service on one node
   7. Restart nova-consoleauth service on one node
   8. Restart nova-compute service on one node
   9. Restart nova-cert service on one node
   10. Kill nova-api-metadata service on one node
   11. Kill nova-api-os-compute service on one node
   12. Kill nova-compute service on one node
   13. Reboot one node with RabbitMQ service
   14. Kill memcached service on one node
   15. Restart MySQL service on one node
   16. Restart nova-api-os-compute service on one node
   17. Restart nova-spicehtml5proxy service on one node
   18. Kill nova-conductor service on one node
   19. Kill nova-scheduler service on one node
8. SwiftObjects.create\_container\_and\_object\_then\_delete\_all:
   1. Restart swift-object-auditor service on one node
   2. Restart swift-object-server service on one node
   3. Restart swift-container-sync service on one node
   4. Kill swift-account-reaper service on one node
   5. Kill swift-container-auditor service on one node
   6. Restart swift-account-reaper service on one node
   7. Restart swift-proxy-server on one node
   8. Restart swift-object-replicator service on one node
   9. Kill swift-object-updater service on one node
   10. Kill swift-container-server service on one node
   11. Restart swift-object-updater service on one node
   12. Kill swift-proxy-server service on one node
   13. Kill swift-account-server service on one node
   14. Kill swift-object-server service on one node
   15. Kill swift-container-replicator service on one node
   16. Restart swift-account-auditor service on one node
   17. Restart swift-container-server service on one node
   18. Kill swift-object-replicator service on one node
   19. Kill swift-object-auditor service on one node
   20. Restart swift-container-auditor service on one node
   21. Restart swift-account-replicator service on one node
   22. Restart swift-container-reconciler service on one node
   23. Restart memcached service on one node
   24. Kill swift-account-auditor service on one node
   25. Restart MySQL service on one node
   26. Kill memcached service on one node
   27. Restart swift-account-server service on one node
   28. Restart swift-container-replicator service on one node
   29. Restart swift-object-expirer service on one node
   30. Kill swift-container-updater service on one node
   31. Kill swift-object-expirer service on one node
   32. Kill swift-container-reconciler service on one node
   33. Kill swift-account-replicator service on one node
   34. Restart RabbitMQ service on one node
   35. Restart swift-container-updater service on one node
   36. Kill swift-container-sync service on one node
9. SwiftObjects.list\_objects:
   1. Restart Keystone service in one node

**1. Authenticate Keystone**

1.1 Restart MySQL service in one node:

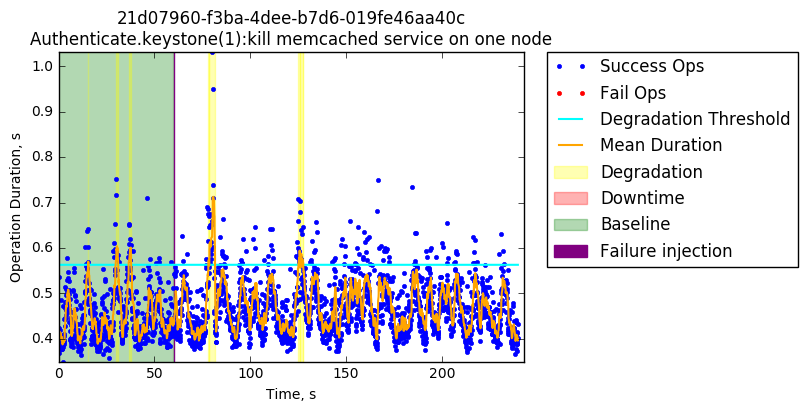
For this test, five runs were executed. In the first chart we can see minimum degradation between operations. As the runs were executed, the degradation started to increase, although still inconsistent for small periods of time. 

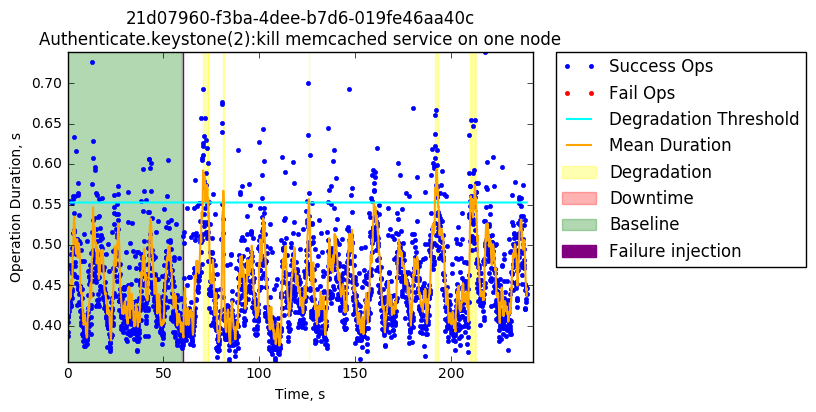


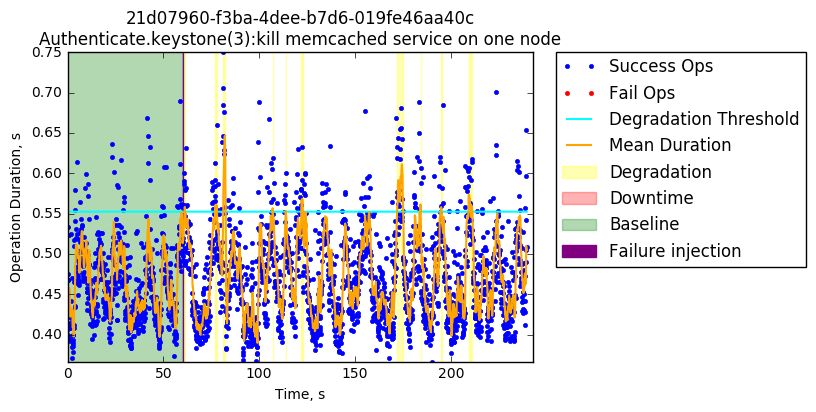


1.2 Kill memcached service in one node:

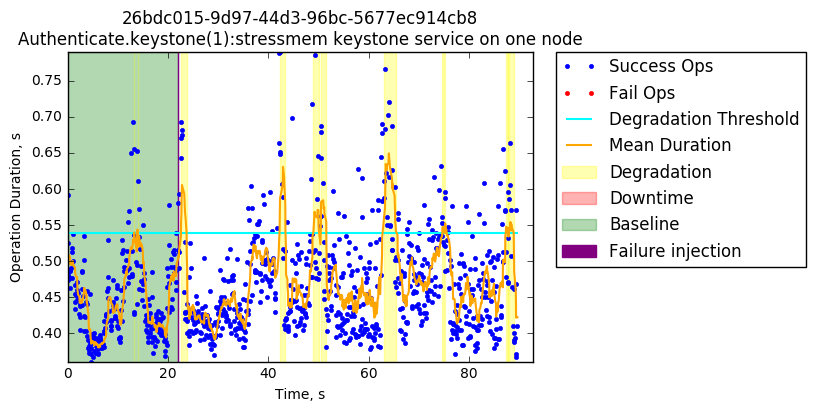
For this test, three runs were executed. From the beginning, we start to see small degradation peaks, but fairly even operations duration distribution. Subsequent runs show more disperse operations duration distribution and higher peaks.





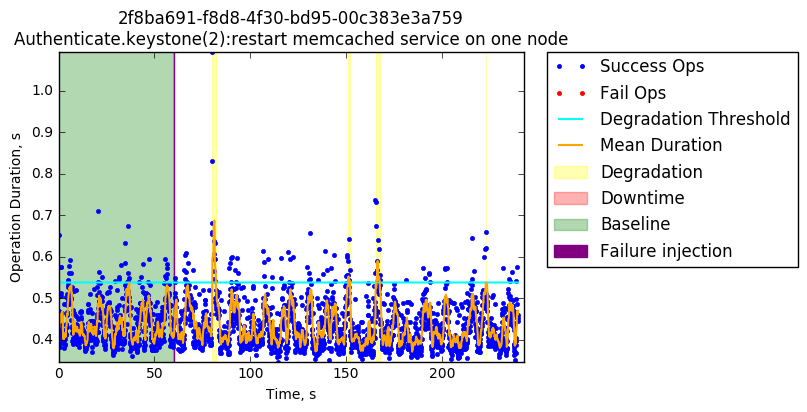


1.3 Stress memory on one node:



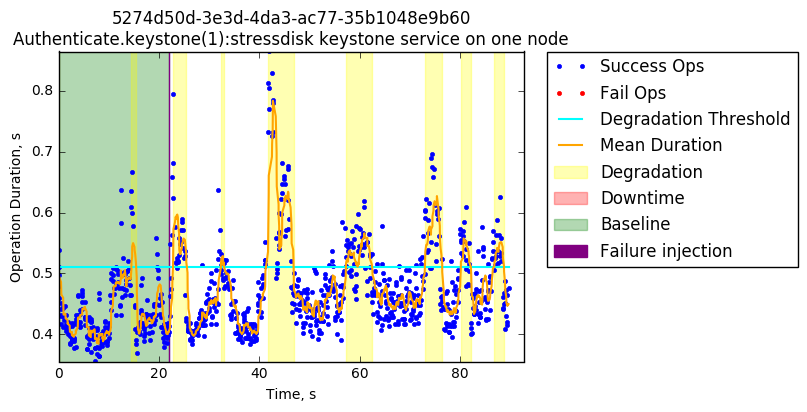
1.4 Restart memcached service on one node

Around 40-50 seconds after the failure injection we can observe small peaks of degradation, which are not significant results, in regards to overall high availability.



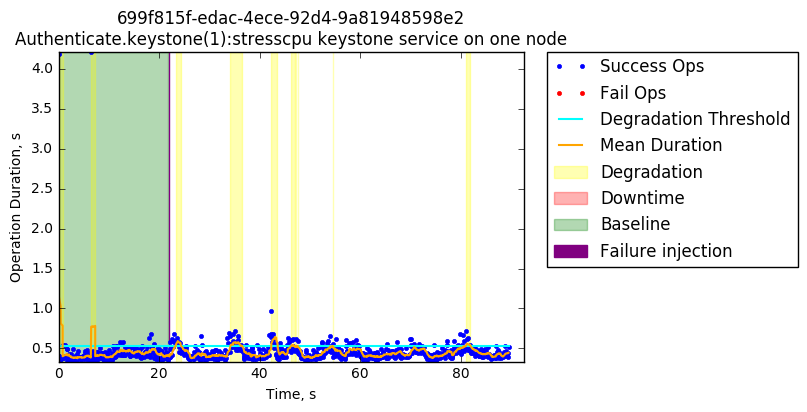
1.5 Stress disk on one node:

Disk stress causes longer degradation periods and higher response times.



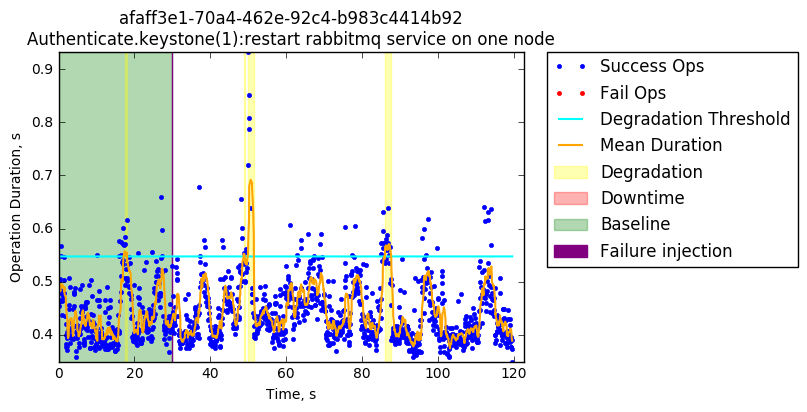
1.6 Stress cpu on one node:

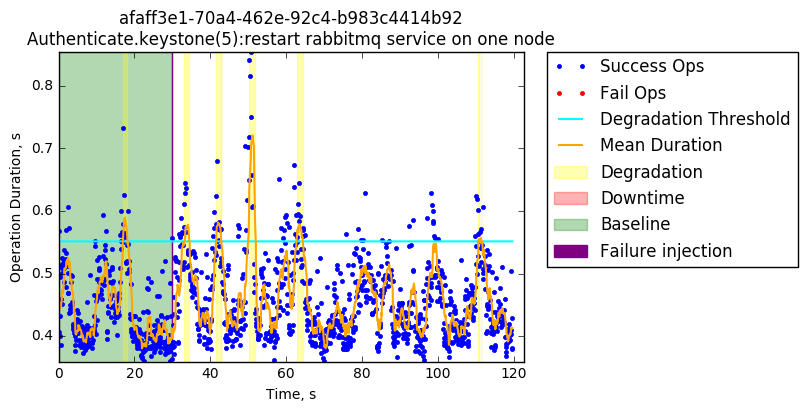
CPU stress did not cause a significant performance degradation.



1.7 Restart RabbitMQ service on one node

Five tasks were executed for this test. Restarting RabbitMQ did not cause failures, but a short and erratic performance degradation appears along the task runtime.

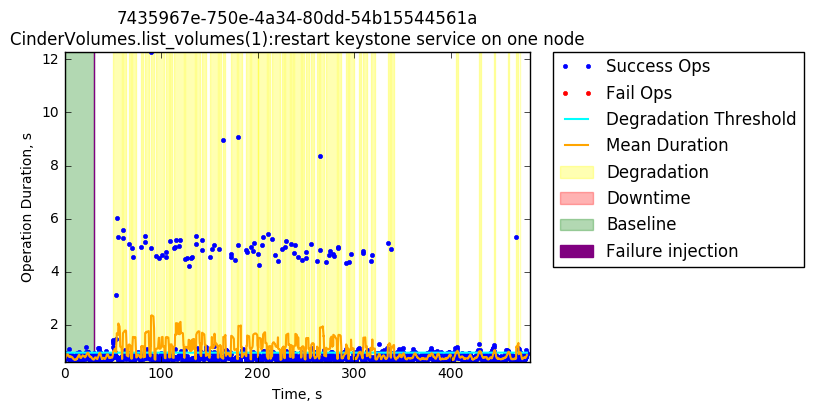




**2.** **CinderVolumes.list\_volumes**

2.1 Restart Keystone service on one node

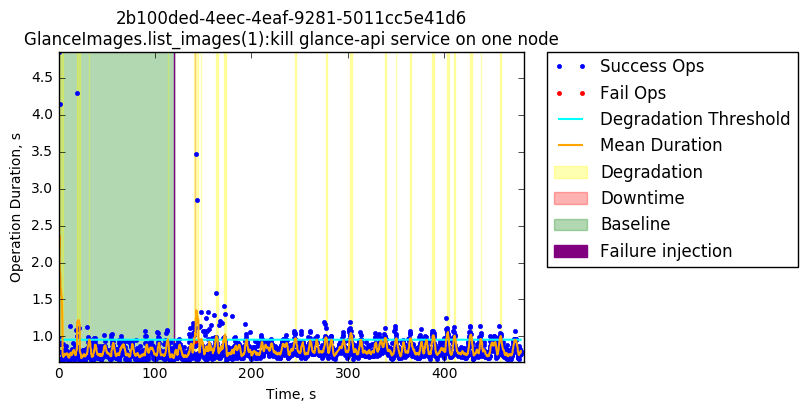
The overall performance degradation time frame on this test is more than the 60% of the overall duration time. In the second execution, some failed operations and a small downtime threshold were observed.



**3 GlanceImages.list\_images**

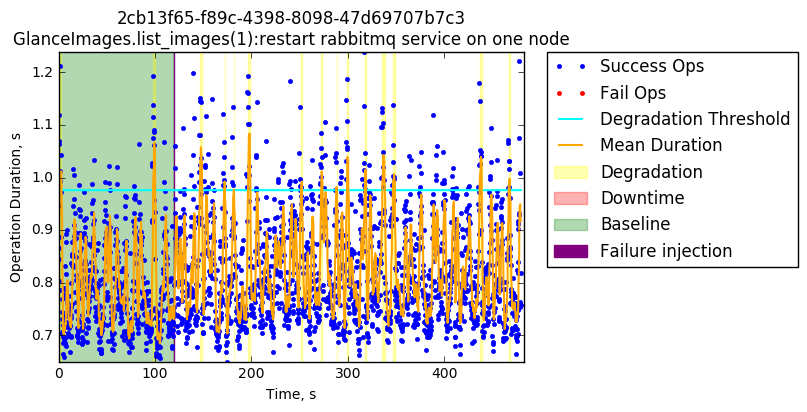
3.1 Kill glance-api service on one node

The baseline sampling starts erratic, with some of operations above the degradation threshold. After approximately 40 seconds after the failure injection we can see a small degradation peak along with some failures. Overall, this failure injection did not harm badly the performance.



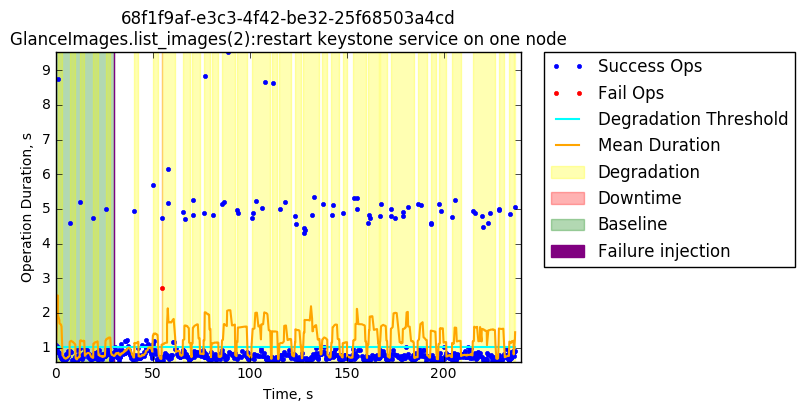
3.2 Restart RabbitMQ service on one node

There is a lot more dispersion in this test, but most of the operations remain under the threshold. After the failure injection, the degradation intervals are more close together.



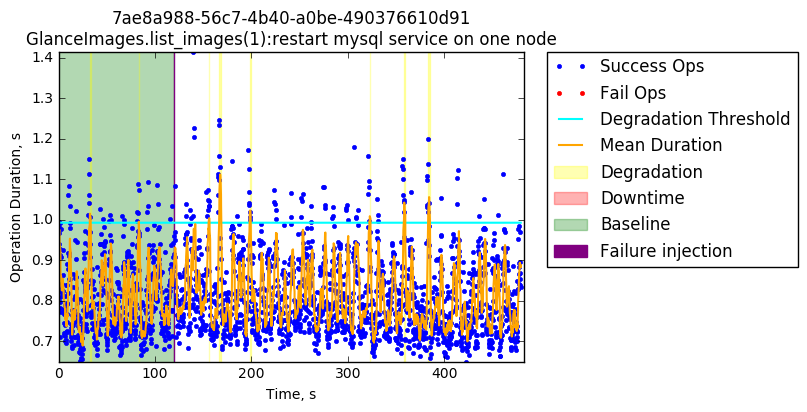
3.3 Restart Keystone service on one node

Similar to Cinder tests, after restarting Keystone in one node the degraded operations start to show up more consistently causing serious degradation time frames



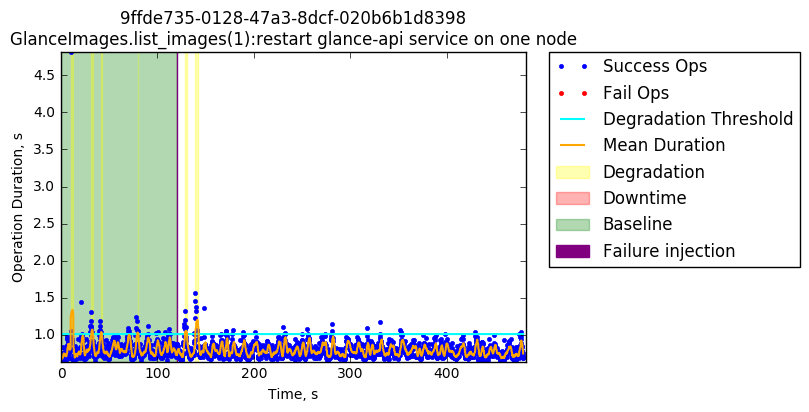
3.4 Restart MySQL service on one node

There is no significant damage to the performance on this test. Injecting failures to background supporting services is nearly transparent to the OpenStack service performance



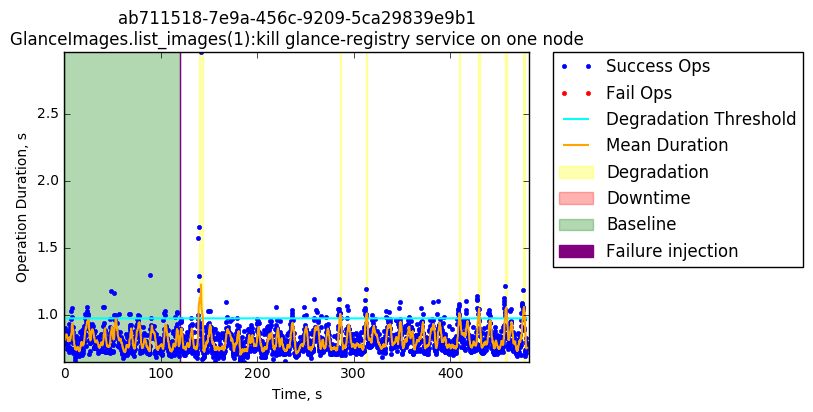
3.5 Restart glance-api service on one node

There are not significant peaks in performance degradation in this test either. Ignoring the peaks of the baseline sample, we can see that after the failure injection there are two small peaks not exceeding the 2 seconds in duration. After these degradation peaks, the service goes back to normal for the rest of the test.



3.6 Kill glance-registry service on one node

After the failure was injected more consistent but small performance degradation peaks appear. Only some operations exceeding the 2 seconds duration.



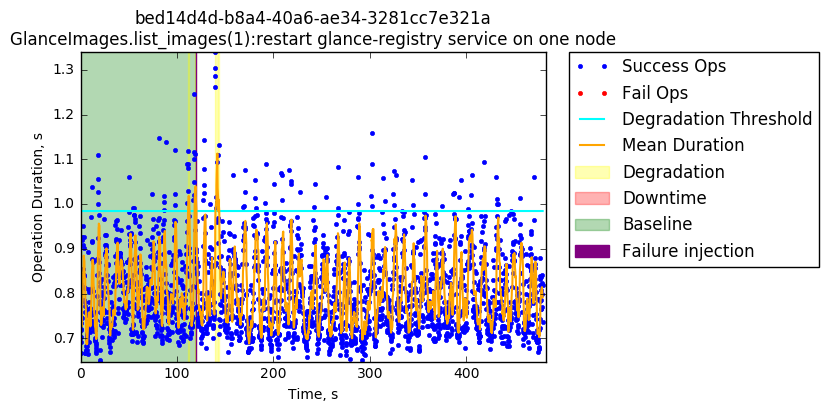
3.7 Restart memcached service on one node

Similar to tests conducted on the MySQL service, there appears to be no significant degradation in this type of test with peaks under 2 seconds in duration per operation.



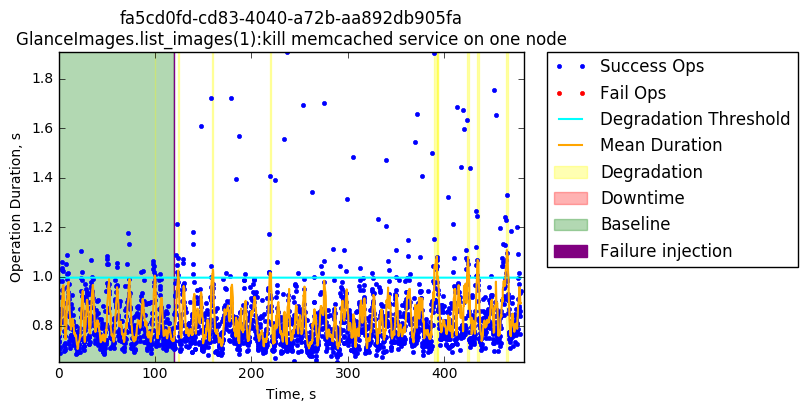
3.8 Restart glance-registry service on one node

Similarly, no significant degradation exceeded 2 seconds in duration per operation.



3.9 Kill memcached service on one node

Similarly, no significant degradation exceeded 2 seconds in duration per operation.

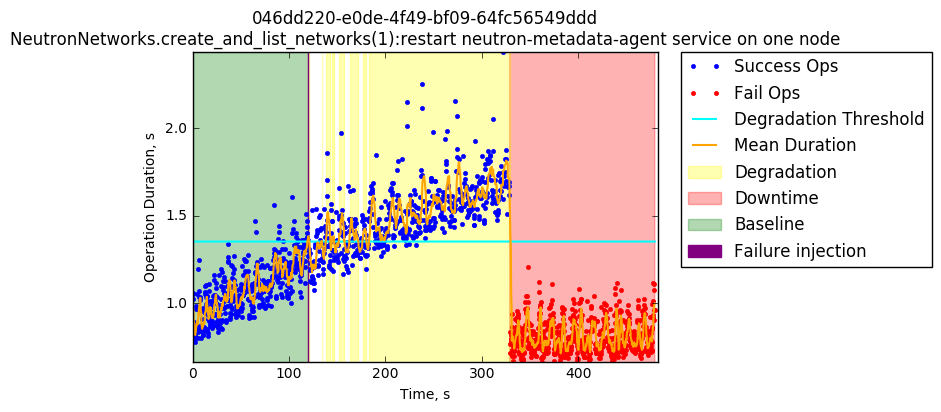


**4 NeutronNetworks.create\_and\_list\_networks**

Neutron tests show similar behavior, despite the failure injection. operation duration seems to increase as the test time increases.

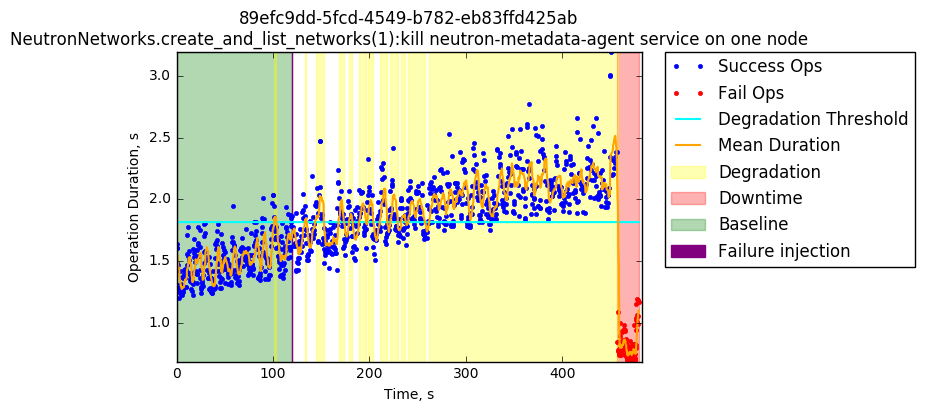
4.1 Restart neutron-metadata-agent service on one node

From the baseline sampling, operation durations consistently starts to increase, which is expected since the testing environment constantly adds new networks for a single user and tenant. The more networks a user owns, the more time it takes to list them all.

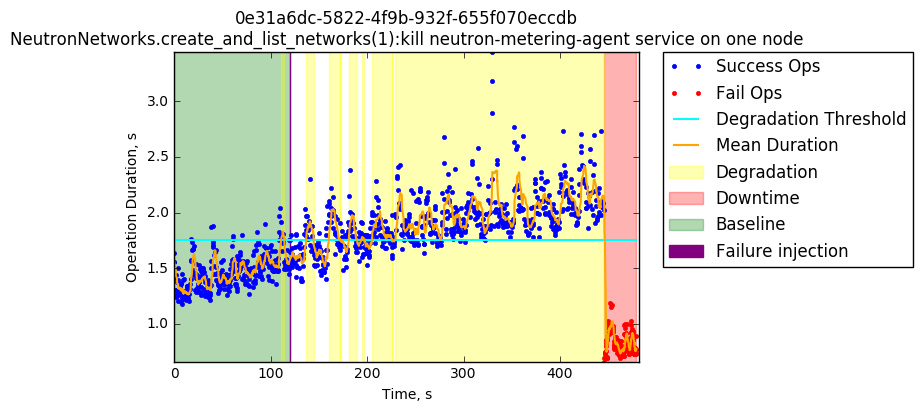


4.2 Kill neutron-metadata-agent service on one node

Similarly, new networks are added for a single user and tenant, and operations expand as result.

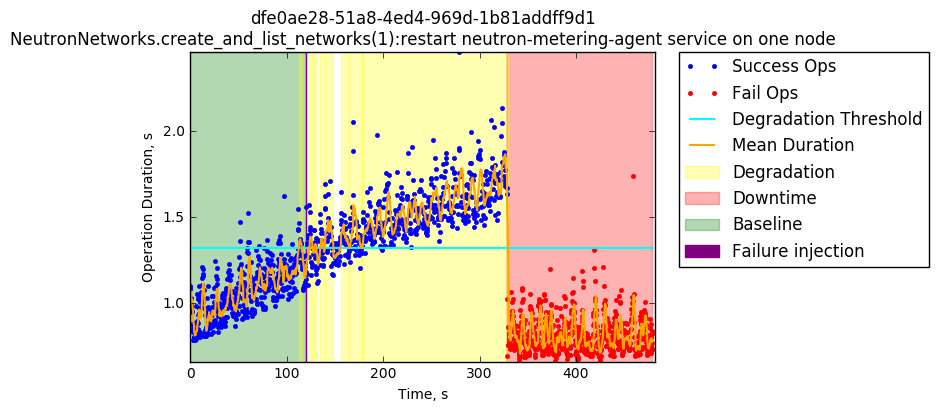


4.3 Kill neutron-metering-agent service on one node



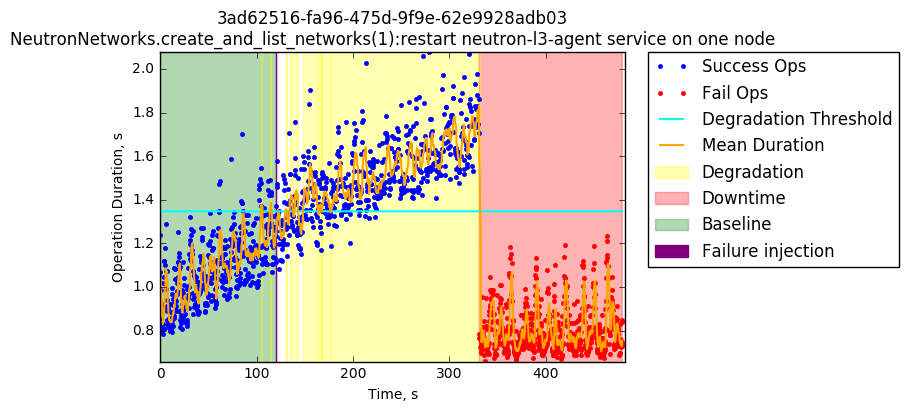
4.4 Restart neutron-metering-agent service on one node

Similarly, new networks are added for a single user and tenant, and operations expand as result.



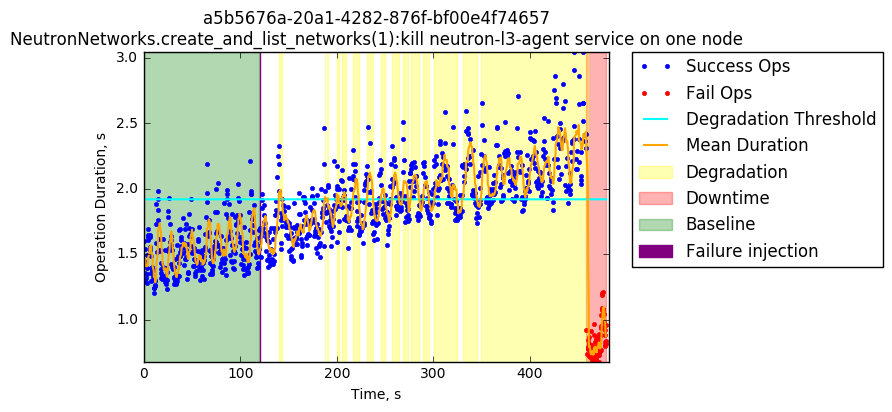
4.5 Restart neutron-l3-agent service on one node

Similarly, new networks are added for a single user and tenant, and operations expand as result.



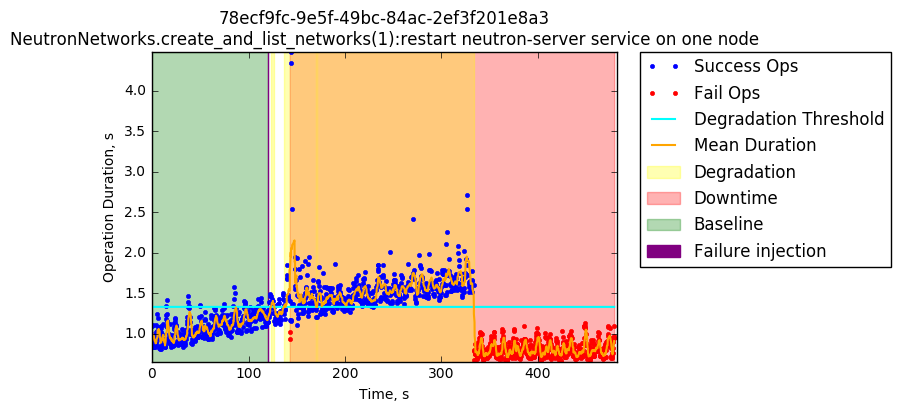
4.6 Kill neutron-l3-agent service on one node

Similarly, new networks are added for a single user and tenant, and operations expand as result.



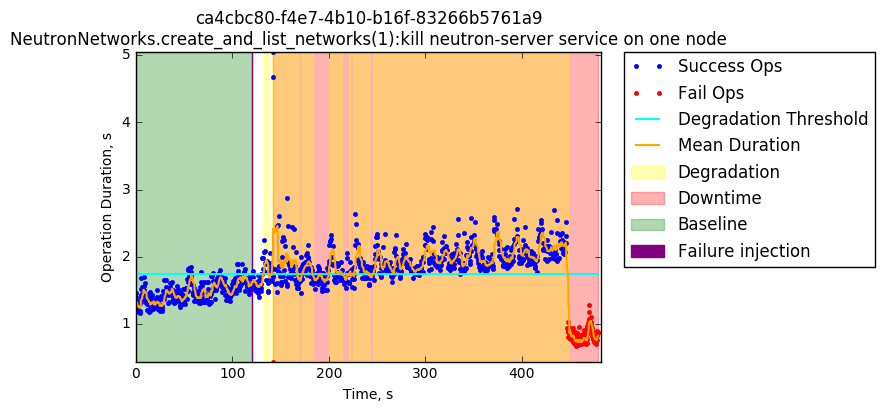
4.7 Restart neutron-server service on one node

In this case we see some failures coexisting with successful operations, which could directly be a result of the failure injection. This does not necessarily means downtime, After approximately 180 seconds we see only failed operations, which indicates downtime.



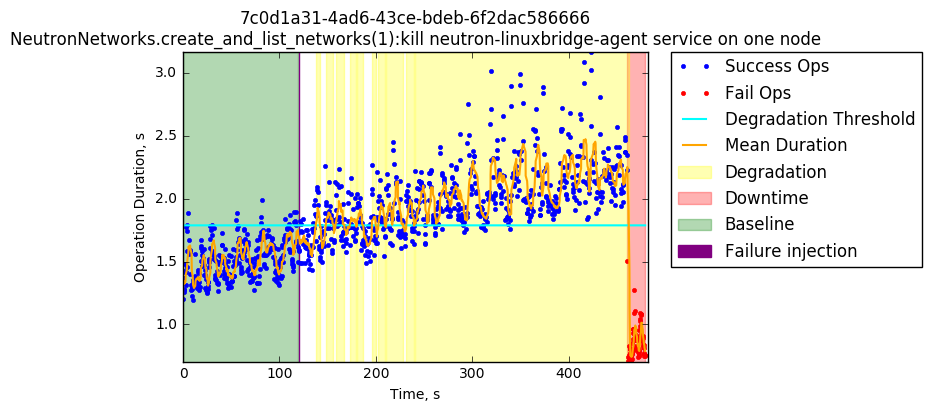
4.8 Kill neutron-server service on one node

In this case we see some failures coexisting with successful operations which does not necessarily means downtime, After approximately 200 seconds we see only failed operations which corresponds to a downtime.

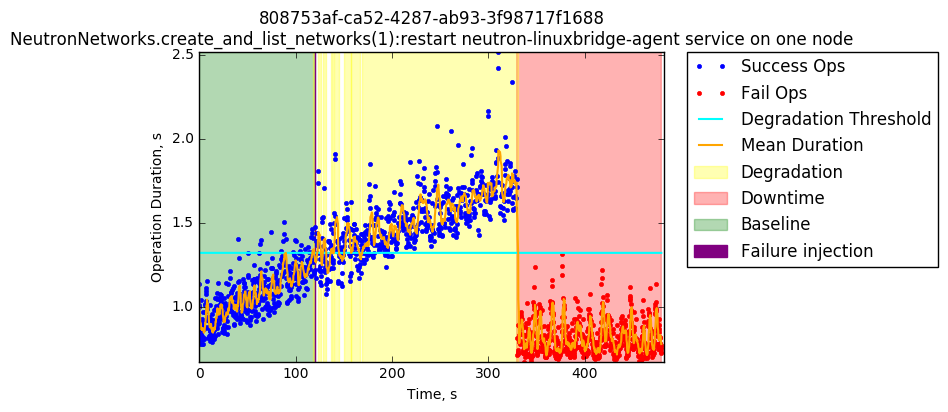


4.9 Kill neutron-linuxbridge-agent service on one node

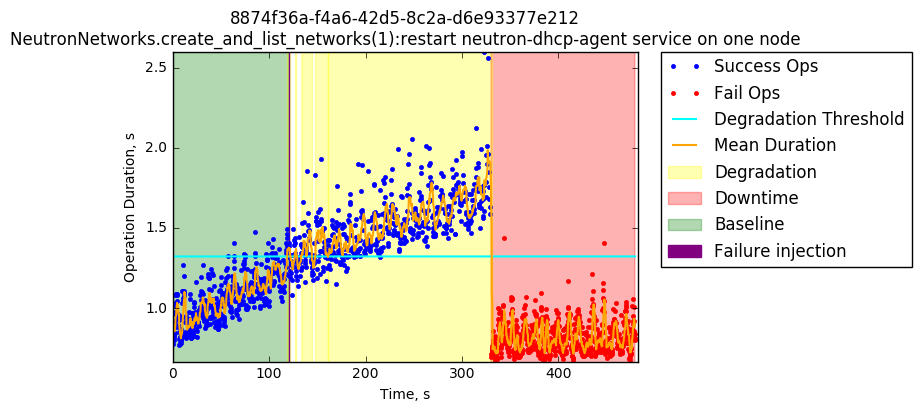
From the baseline sampling we can see how operations’ duration consistently starts to increase. After the failure injection, operations start to exceed the degradation threshold after a few hundred seconds the failure injection/workload starts to hit the system generating a downtime in the service for the rest of the test.



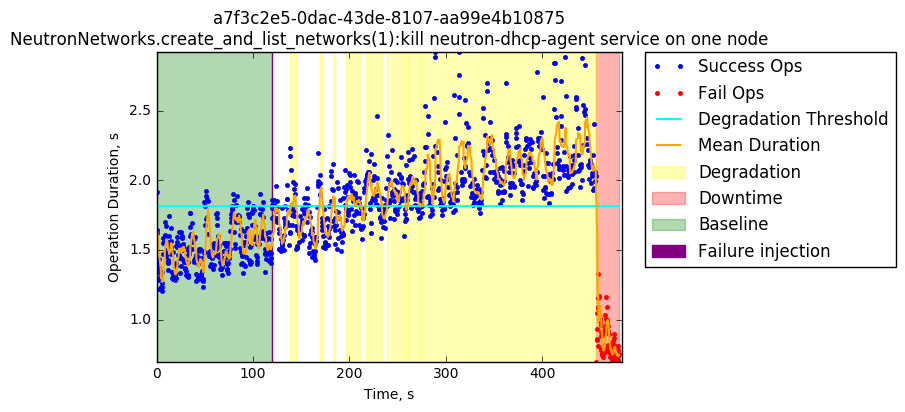
4.10 Restart neutron-linuxbridge-agent service on one node



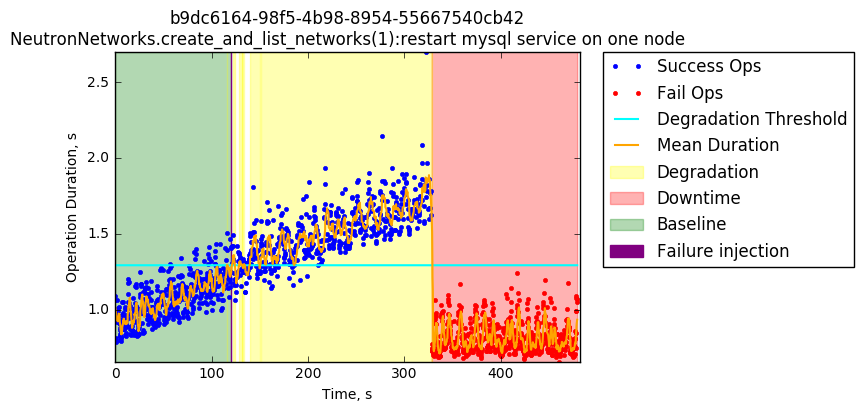
4.11 Restart neutron-dhcp-agent service on one node



4.12 Kill neutron-dhcp-agent service on one node



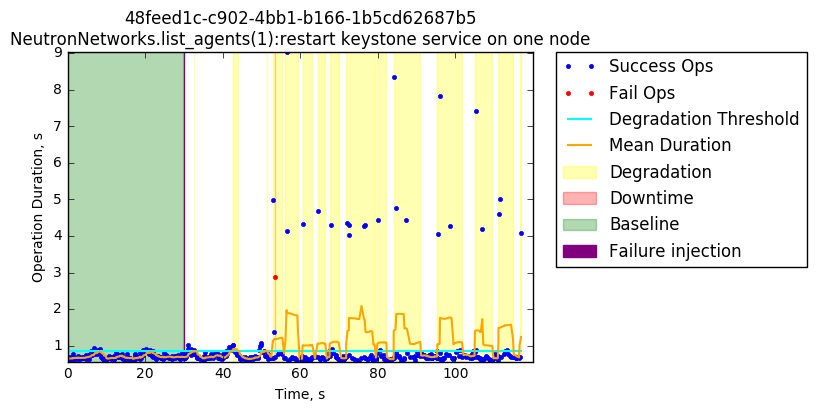
4.13 Restart MySQL service on one node



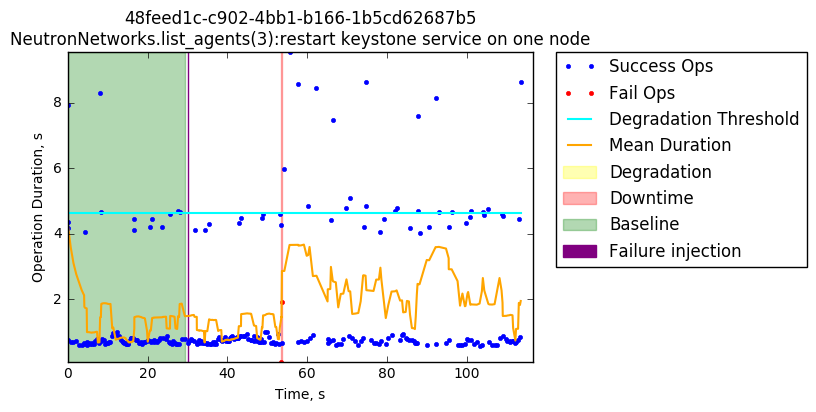
**5 NeutronNetworks.list\_agents**

5.1 Restart Keystone service on one node

Similar to other Keystone service tests the performance after the failure injection causes severe issues, going from less than a second of duration per operation to approximately 9 seconds of duration in some operations. Errors can also be observed.



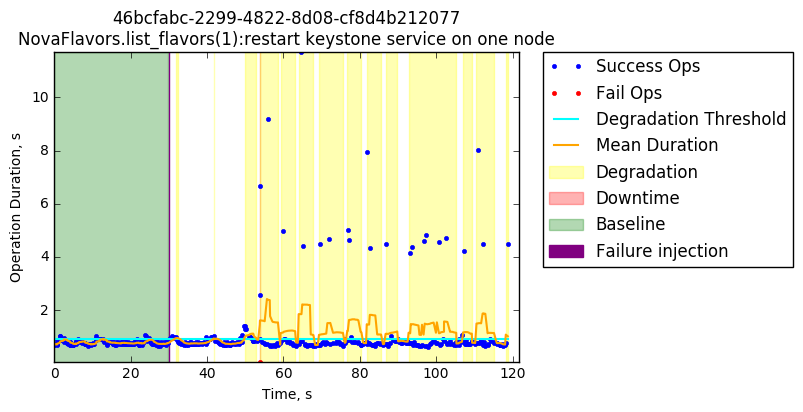
In later executions the sampling starts to have more long operations so the degradation threshold goes up by approximately 4 seconds. So we pass from having most of the operations lasting less than a second to having them now lasting around approximately 4 seconds.

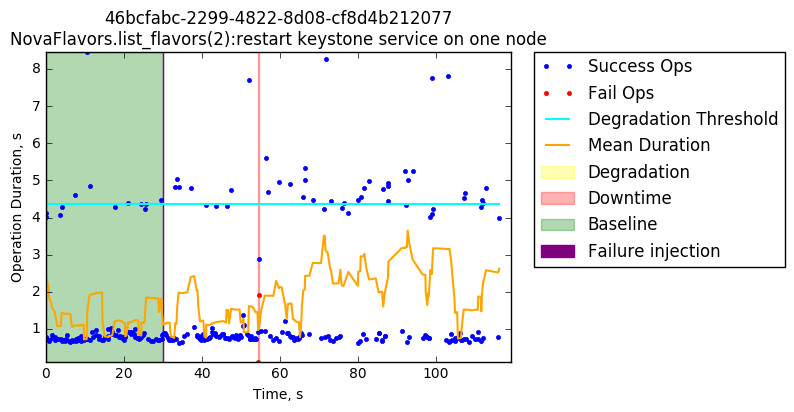


**6 Nova Flavors.list\_flavors**

6.1 restart Keystone service on one node

Similar to other Keystone service tests the performance after the failure injection causes severe issues, going from less than a second of duration per operation to approximately 11 seconds of duration in some operations.



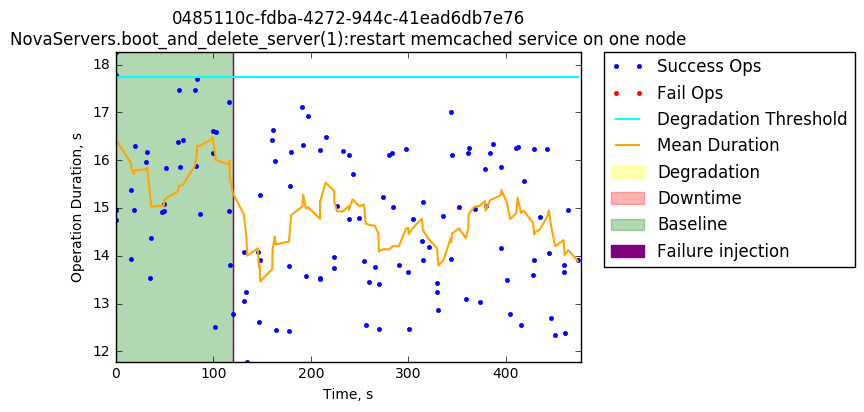


**7 NovaServers.boot\_and\_delete\_server**

Nova tests have fewer baseline samples and fewer operations overall. All of the scenarios seem to behave optimally after the failure injection, only a few of them show failed operations and small downtime windows.

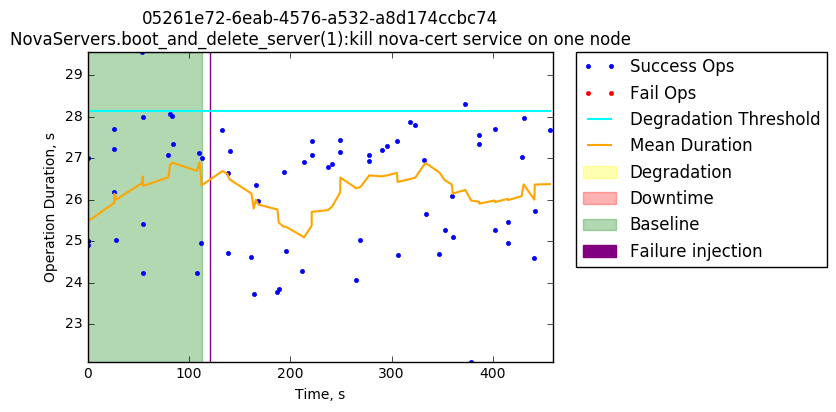
7.1 Restart memcached service on one node

Failure injection does not seem to affect the performance of the service.



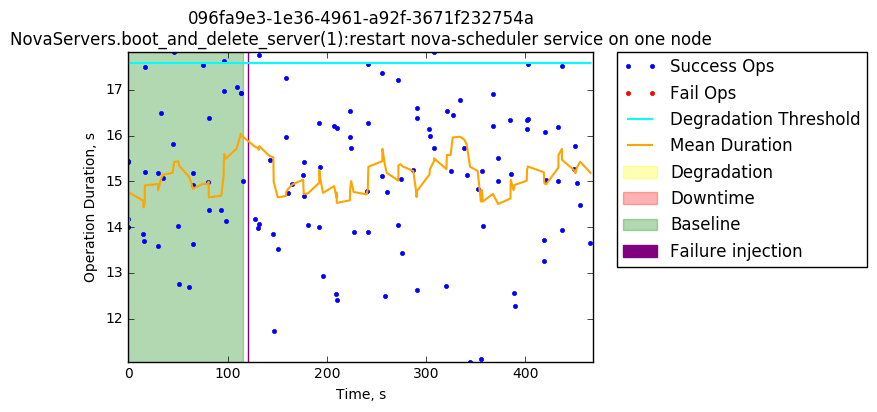
7.2 Kill nova-cert service on one node

Failure injection does not seem to affect the service.



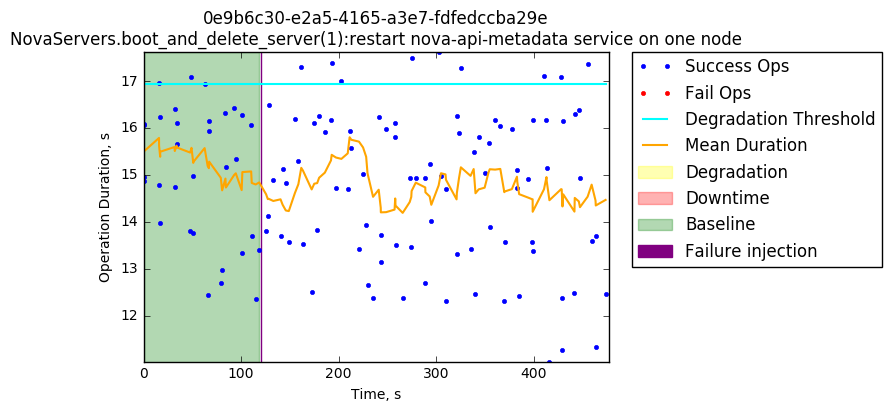
7.3 Restart nova-scheduler service on one node

Failure injection does not seem to affect the service.



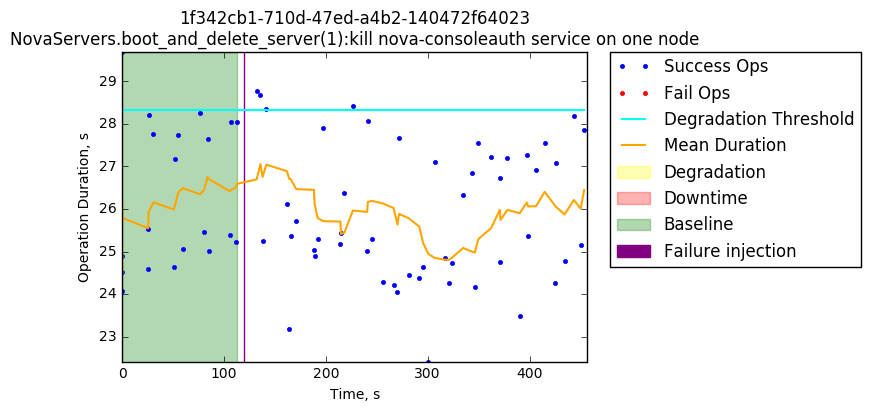
7.4 Restart nova-api-metadata service on one node

Failure injection does not seem to affect the service.



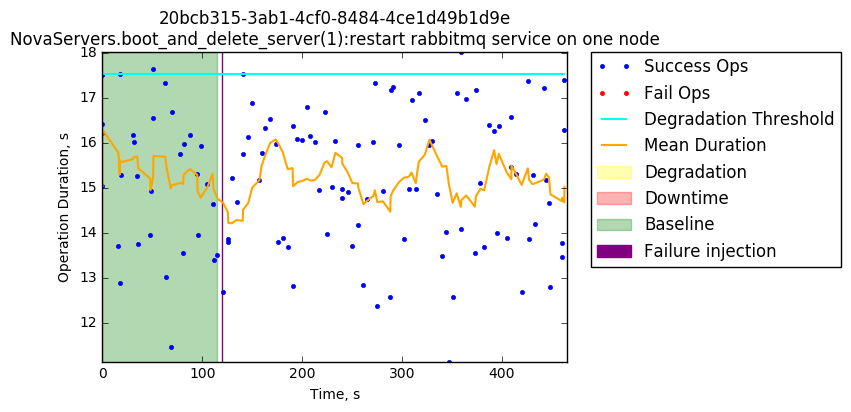
7.5 Kill nova-consoleauth service on one node

Failure injection does not seem to affect the service.



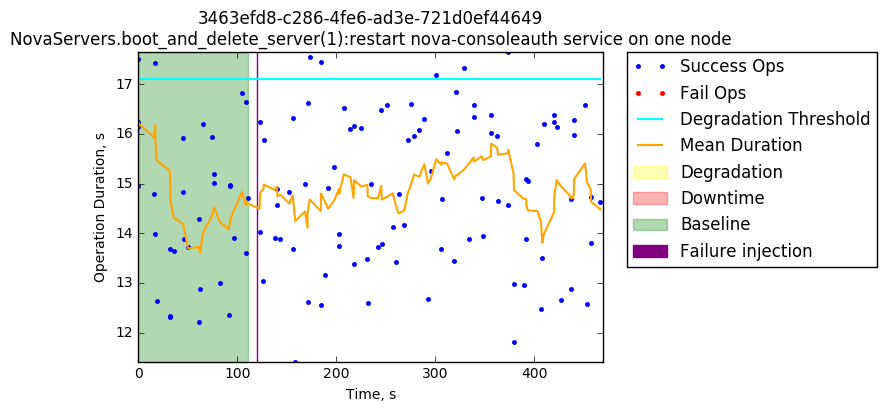
7.6 Restart RabbitMQ service on one node

Failure injection does not seem to affect the service.



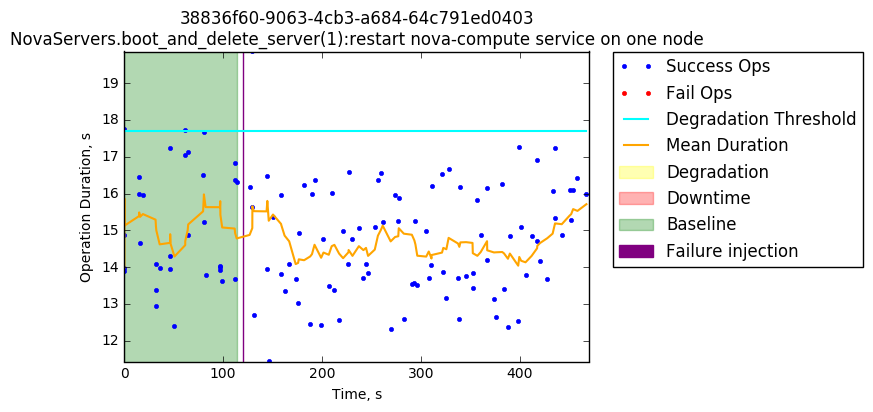
7.7 Restart nova-consoleauth service on one node

Failure injection does not seem to affect the service.



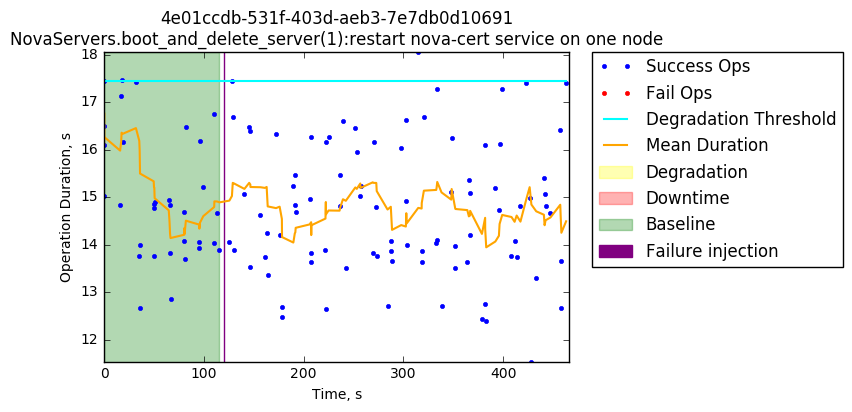
7.8 Restart nova-compute service on one node

Failure injection does not seem to affect the service.



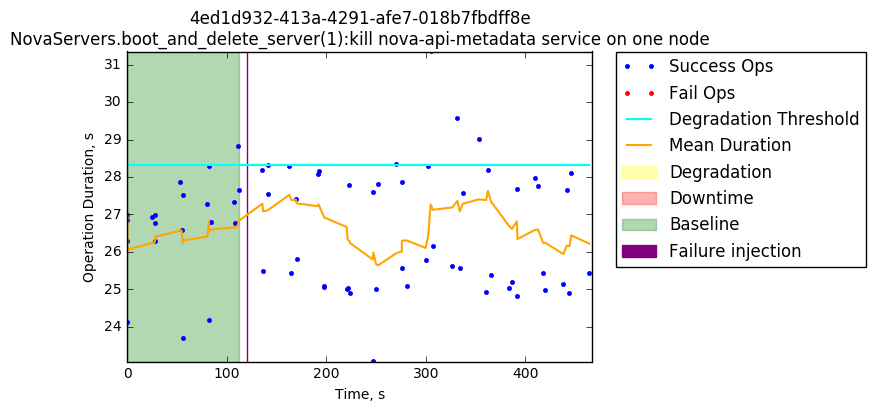
7.9 Restart nova-cert service on one node

Failure injection does not seem to affect the service.



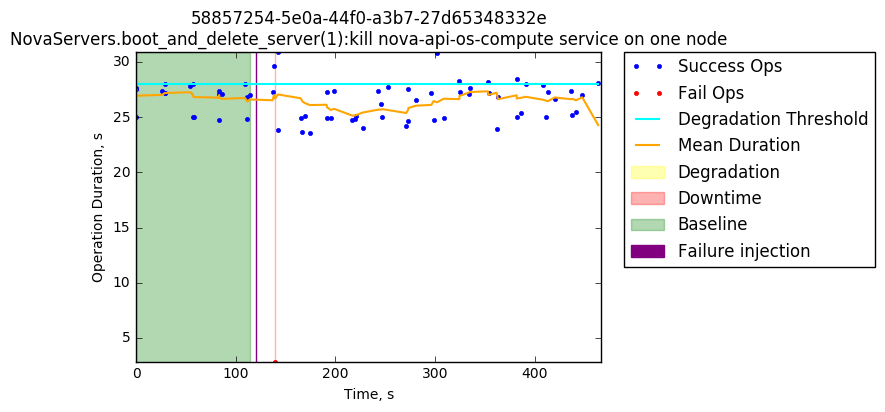
7.10 Kill nova-api-metadata service on one node

Failure injection does not seem to affect the service.



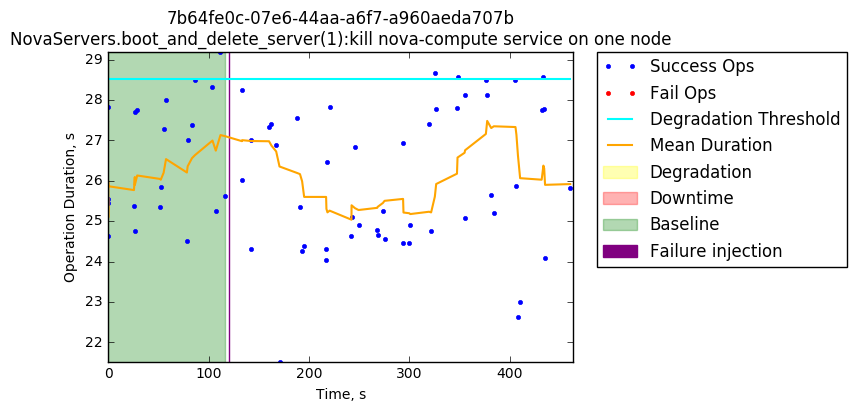
7.11 Kill nova-api-os-compute service on one node

After the failure injection, there is a small downtime window with a failure being reported in the graph.



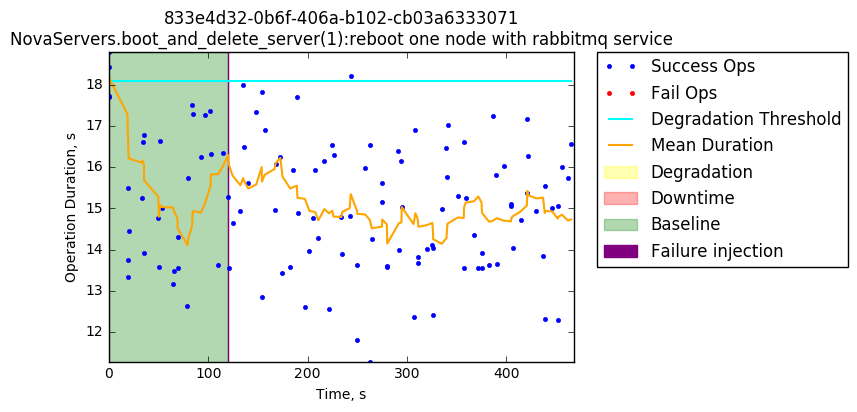
7.12 Kill nova-compute service on one node

Failure injection does not seem to affect the service..



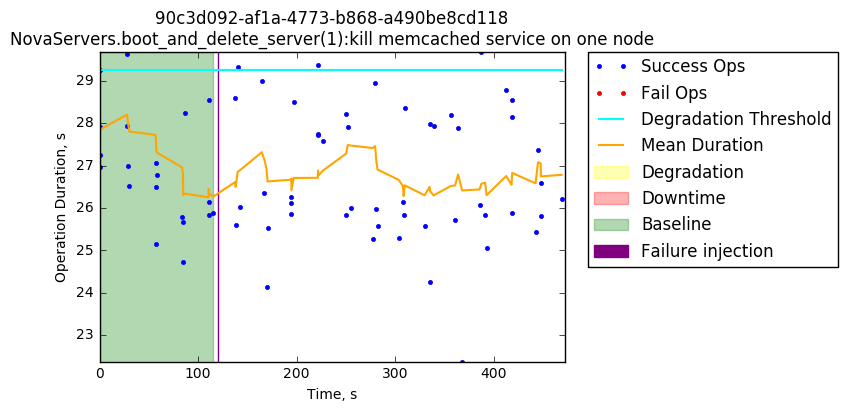
7.13 Reboot one node with RabbitMQ service

Failure injection does not seem to affect the service.



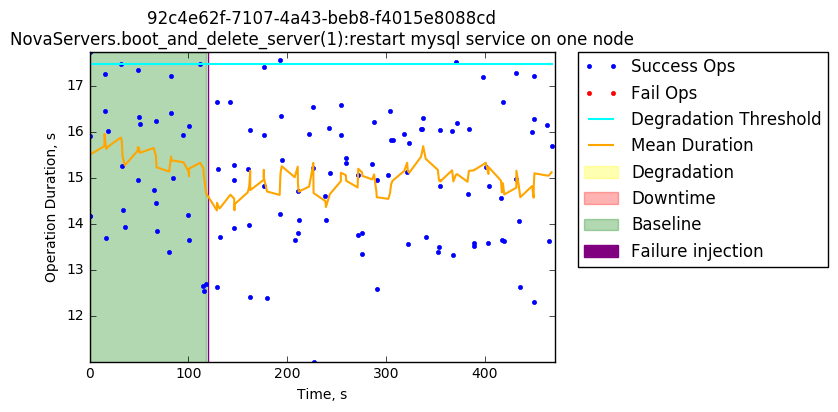
7.14 Kill memcached service on one node

Failure injection does not seem to affect the service.



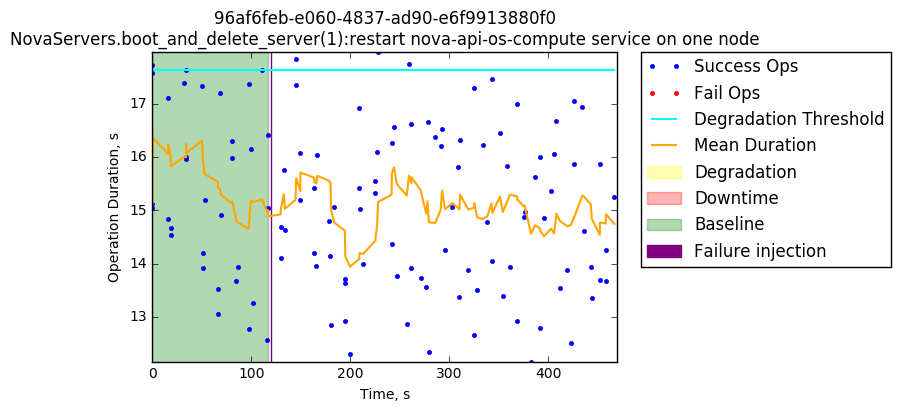
7.15 Restart MySQL service on one node

Failure injection does not seem to affect the service.



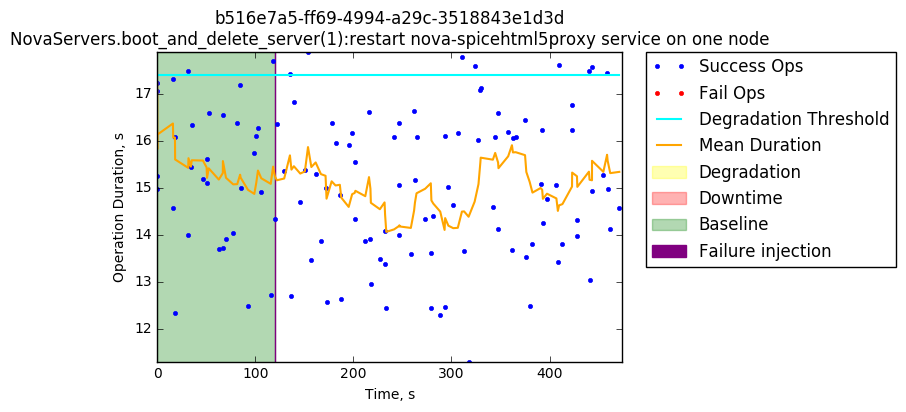
7.16 Restart nova-api-os-compute service on one node

Failure injection does not seem to affect the service.



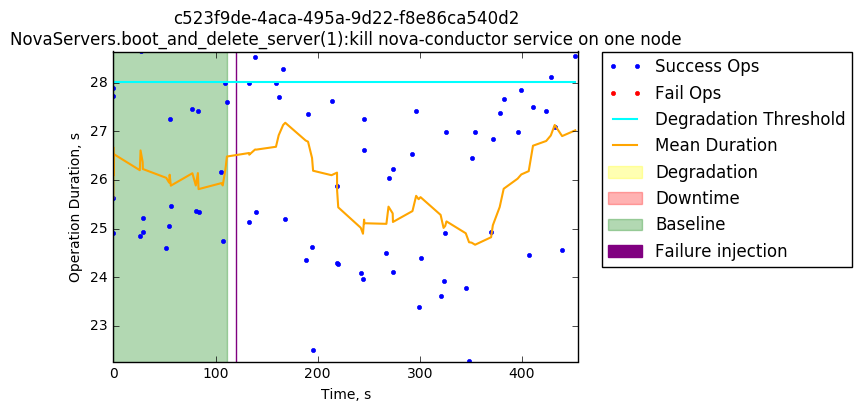
7.17 Restart nova-spicehtml5proxy service on one node

Failure injection does not seem to affect the service.



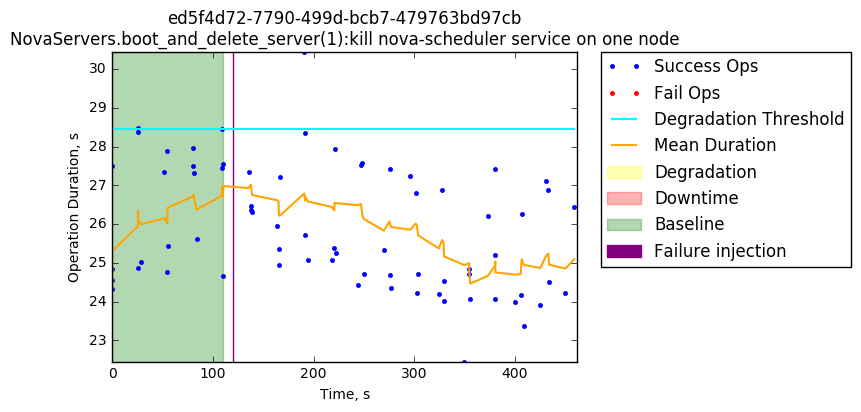
7.18 Kill nova-conductor service on one node

Failure injection does not seem to affect the service.



7.19 Kill nova-scheduler service on one node

Failure injection does not seem to affect the service.

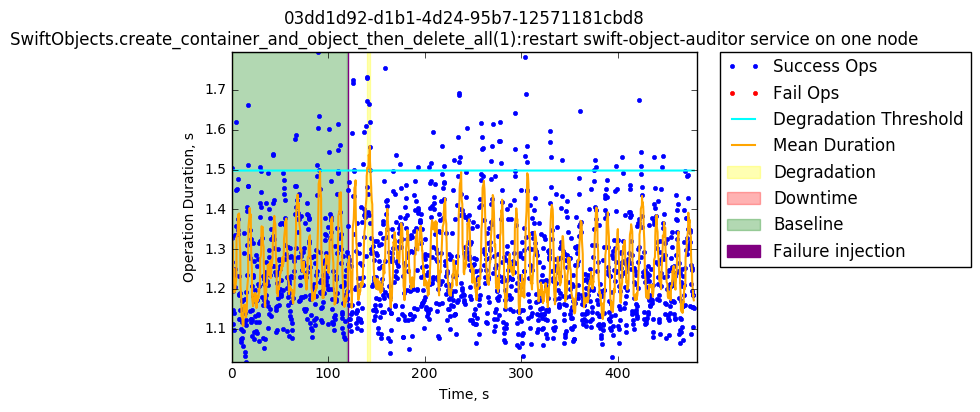


**8 SwiftObjects.create\_container\_and\_object\_then\_delete\_all:**

Most of the following tests presented zero, or almost zero performance degradation failures. In some of the graphs below, the degradation peaks barely cross the threshold after the failure injection, and most of the test completed all the operations successfully.

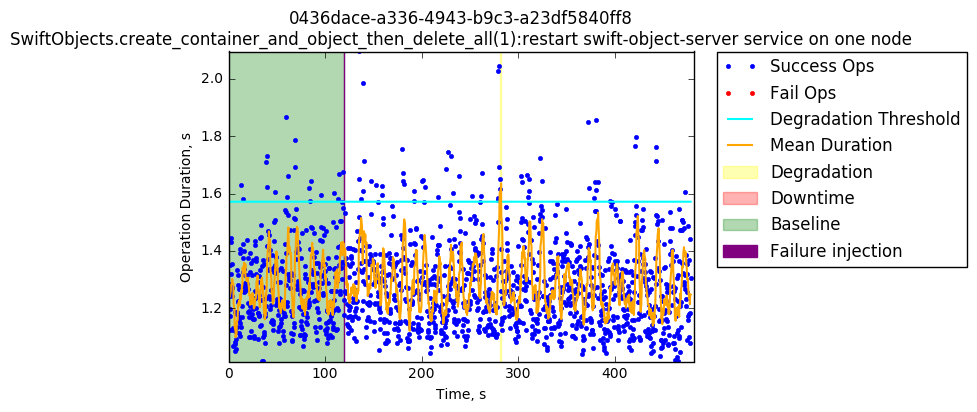
8.1 Restart swift-object-auditor service on one node

The small peak of performance degradation is barely crossing the threshold after the failure injection and after that goes back to normal.



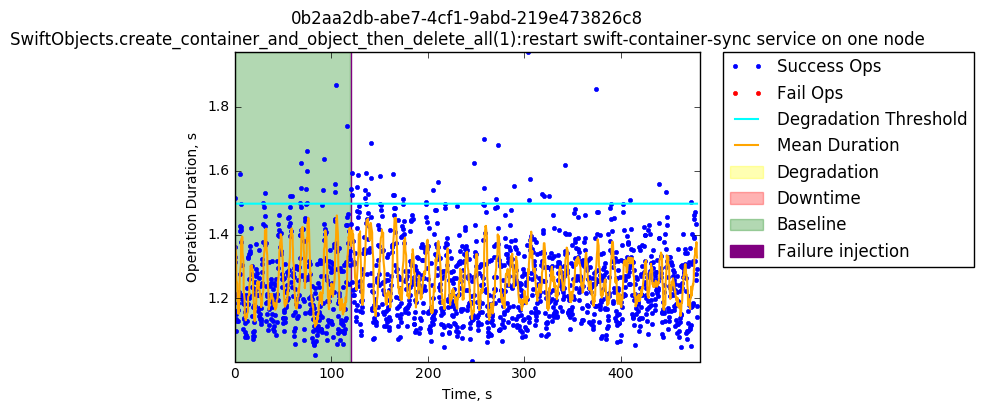
8.2 Restart swift-object-server service on one node

Performance degradation is barely visible. It is difficult to tell if the injected failure actually caused some damage



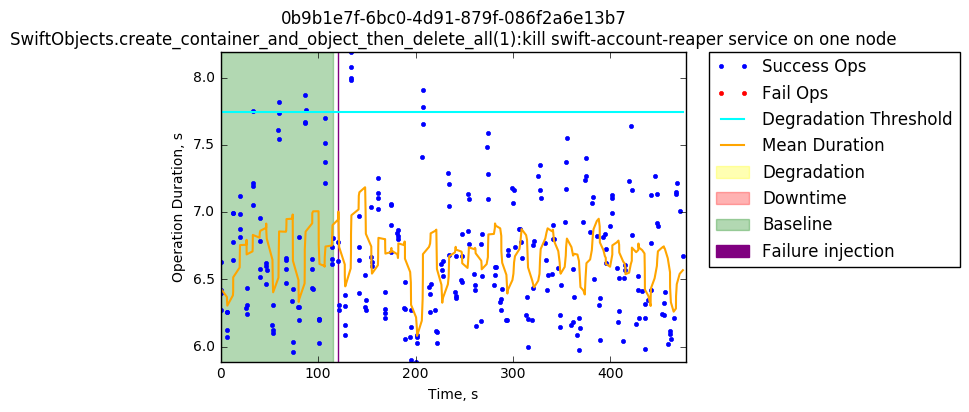
8.3 Restart swift-container-sync service on one node

No failures or degradation present in this test

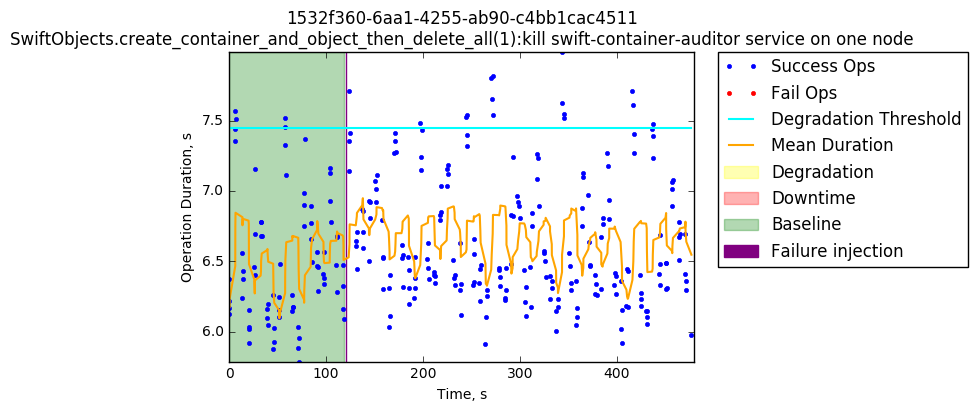


8.4 Kill swift-account-reaper service on one node

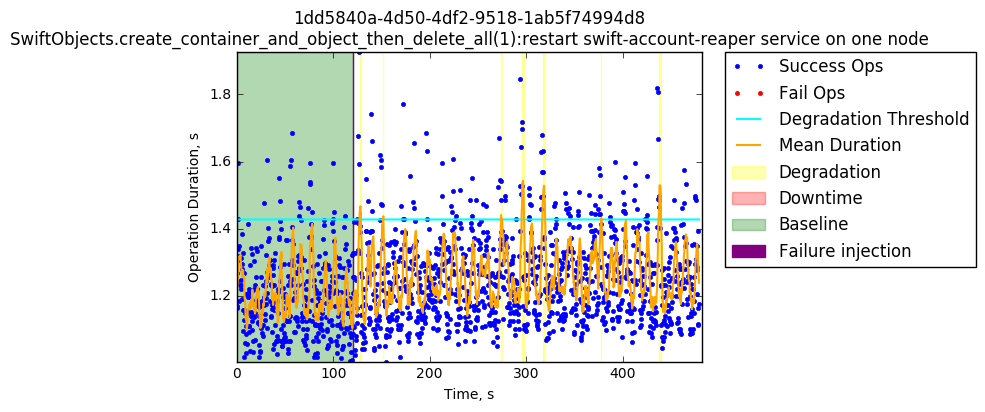
With fewer samples this graph shows broader and wide dispersion between operations but despite that there is no degradation or failures after the failure injection.



8.5 Kill swift-container-auditor service on one node

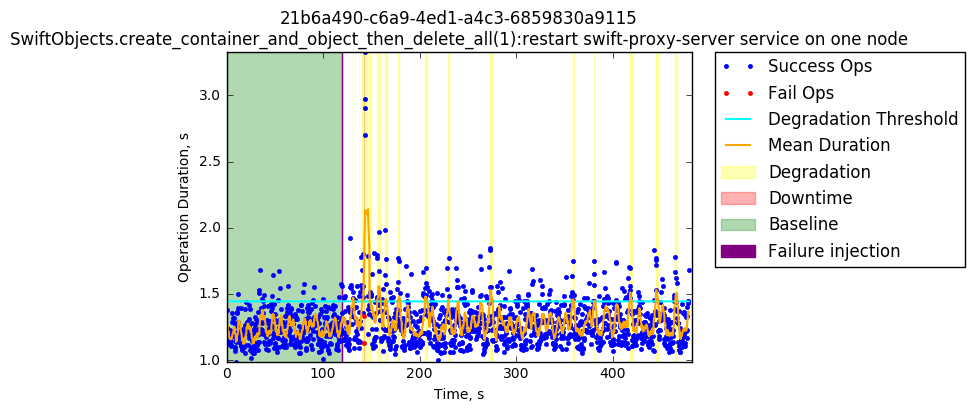


8.6 Restart swift-account-reaper service on one node

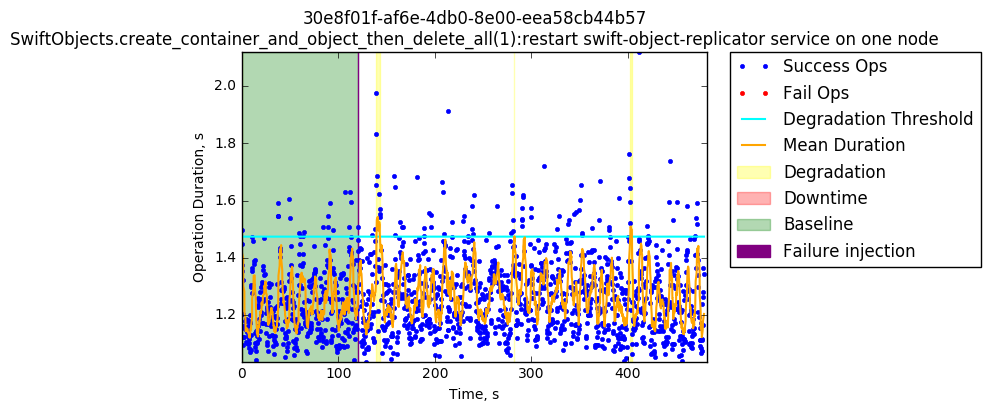


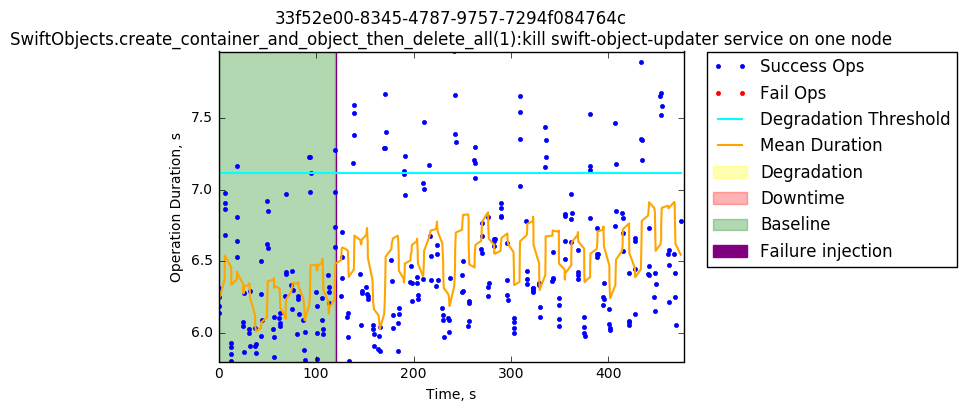
8.7 Restart swift-proxy-server on one node

The issues and damage caused after the failure injection is easier to see. Around approximately 35 seconds after the failure injection performance degradations peak along with some failures. After approximately 10 seconds, the issues disappear, and the performance goes back to barely exceeding the threshold.

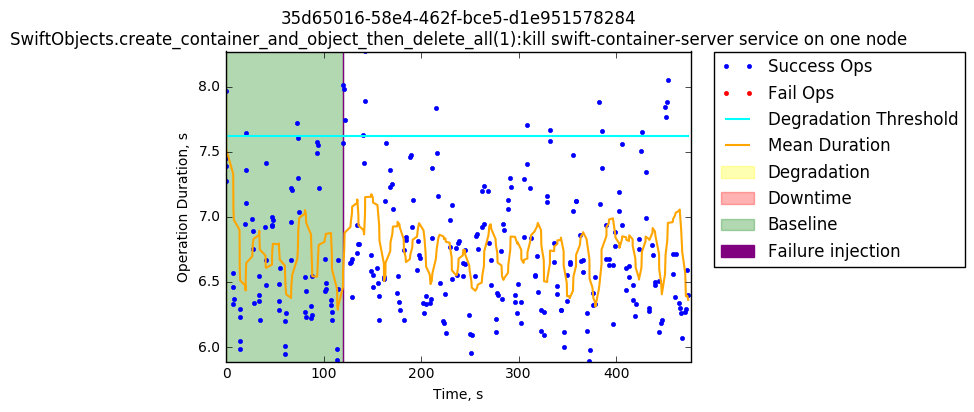


8.8 Restart swift-object-replicator service on one node

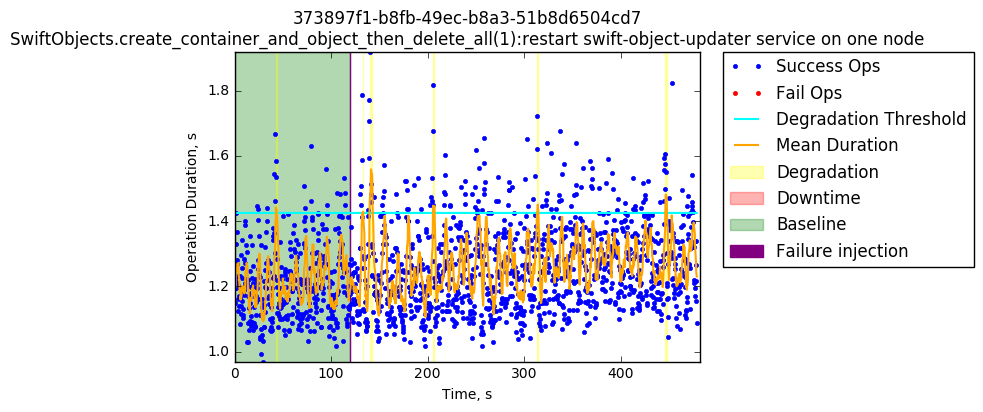


8.9 Kill swift-object-updater service on one node

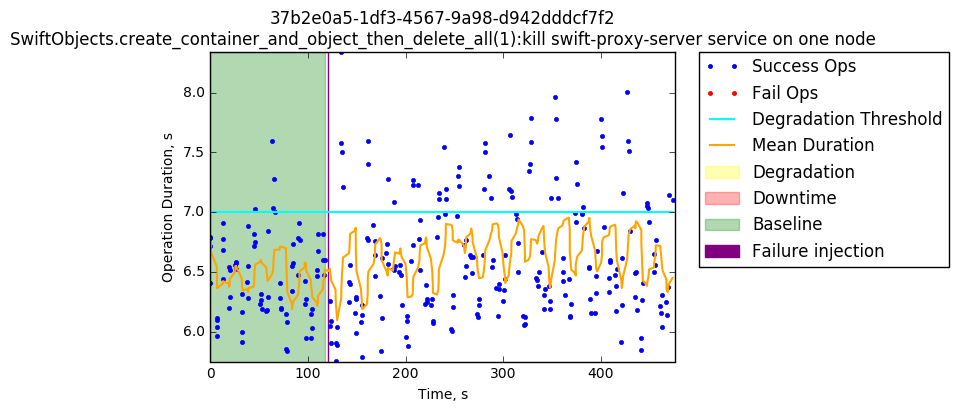
8.10 Kill swift-container-server service on one node



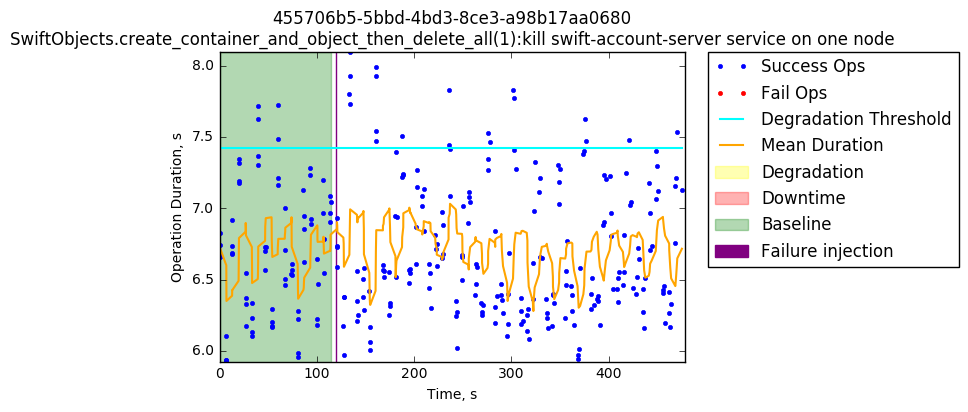
8.11 Restart swift-object-updater service on one node



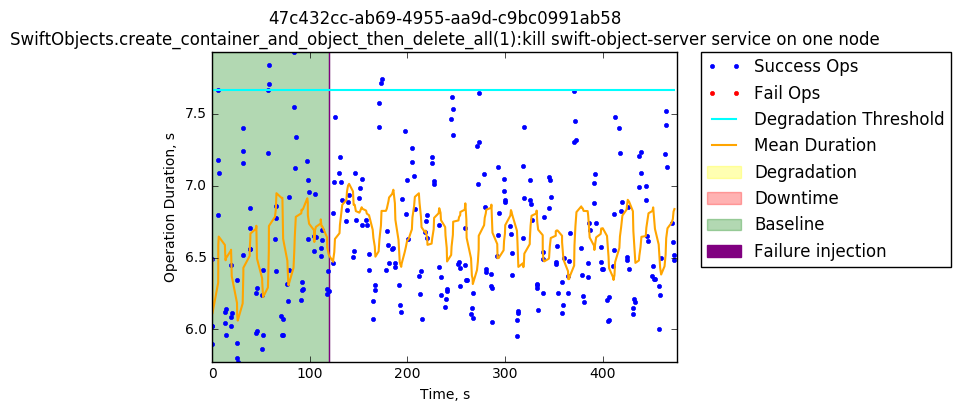
8.12 Kill swift-proxy-server service on one node



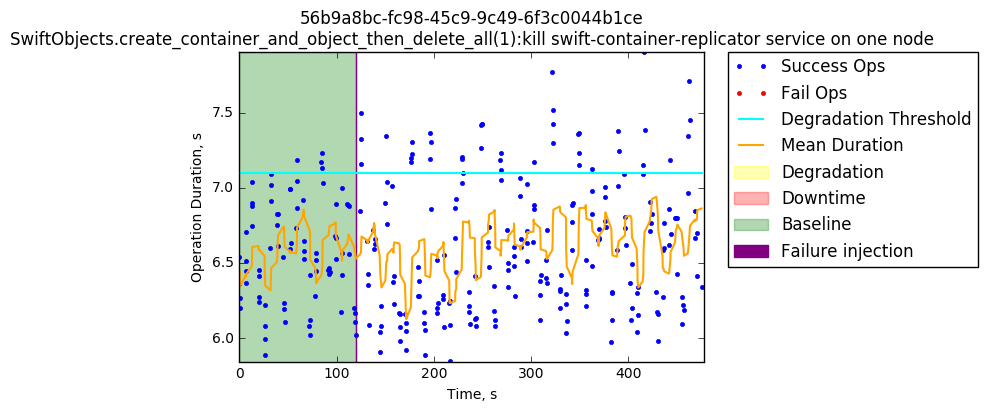
8.13 Kill swift-account-server service on one node



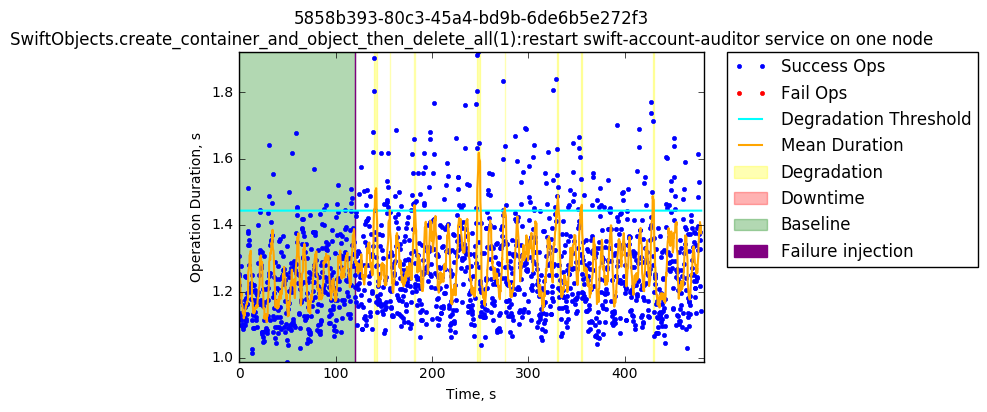
8.14 Kill swift-object-server service on one node



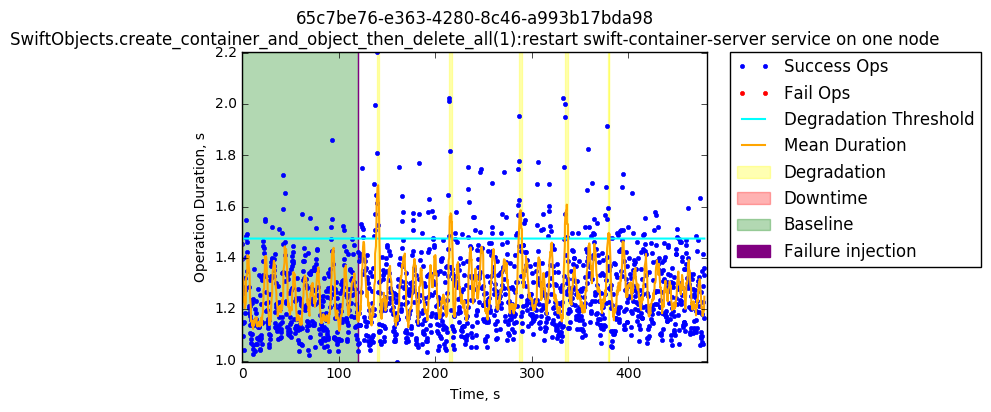
8.15 Kill swift-container-replicator service on one node



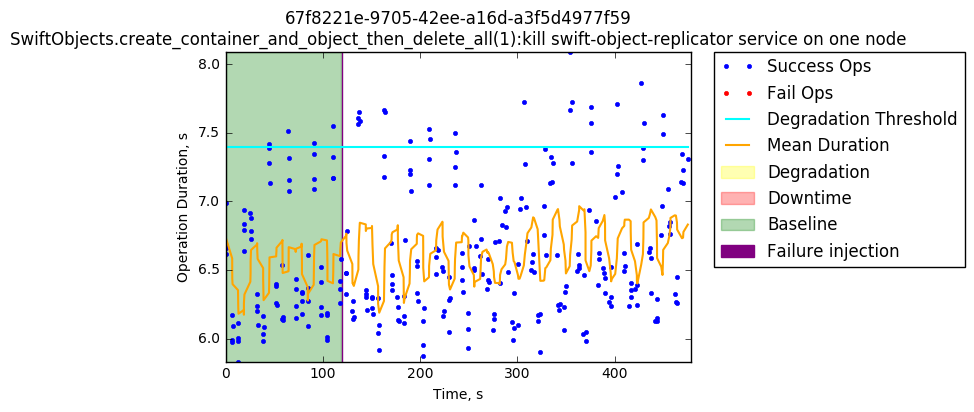
8.16 Restart swift-account-auditor service on one node



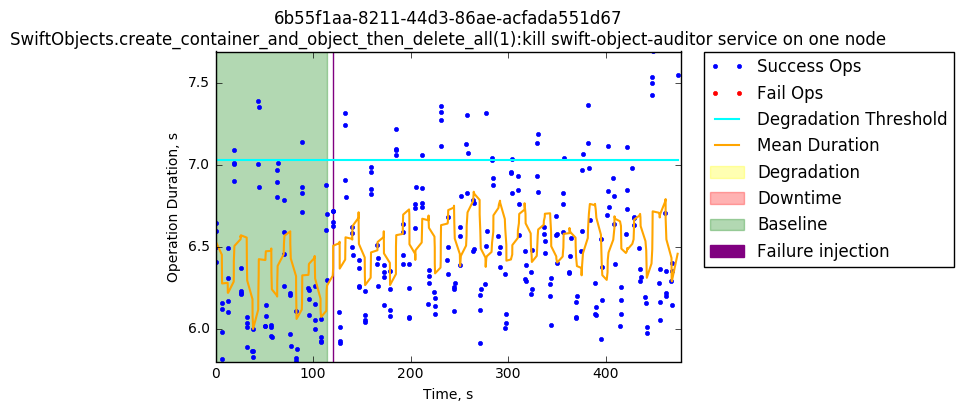
8.17 Restart swift-container-server service on one node



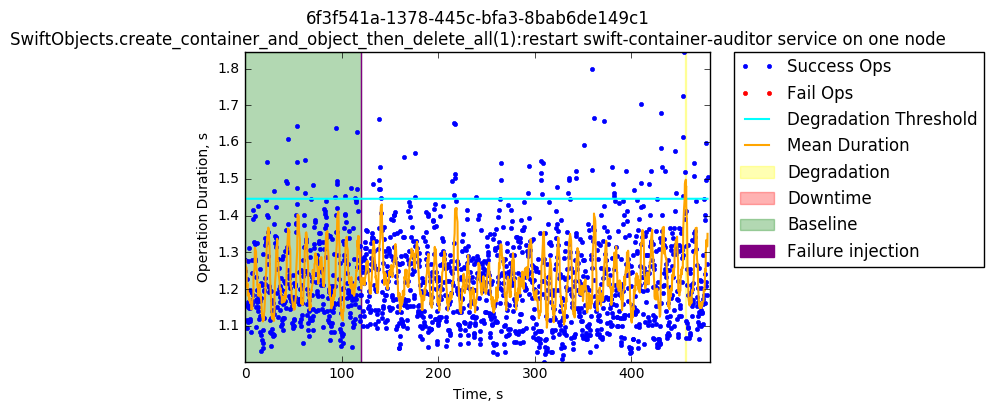
8.18 Kill swift-object-replicator service on one node



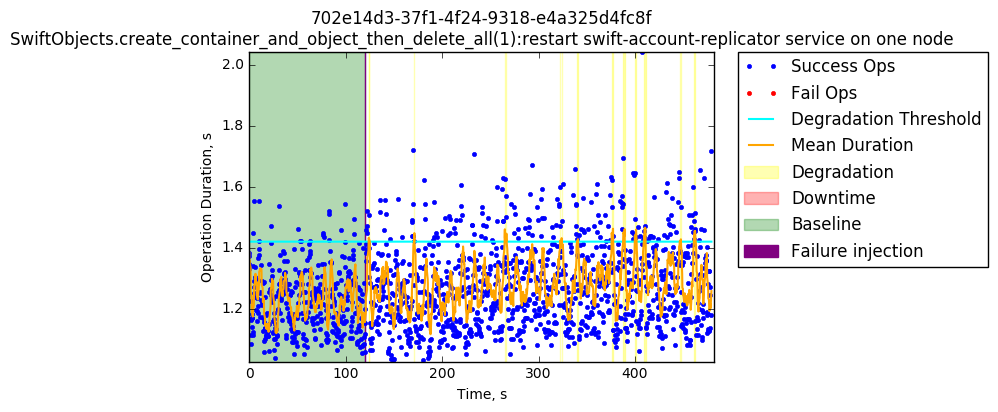
8.19 Kill swift-object-auditor service on one node



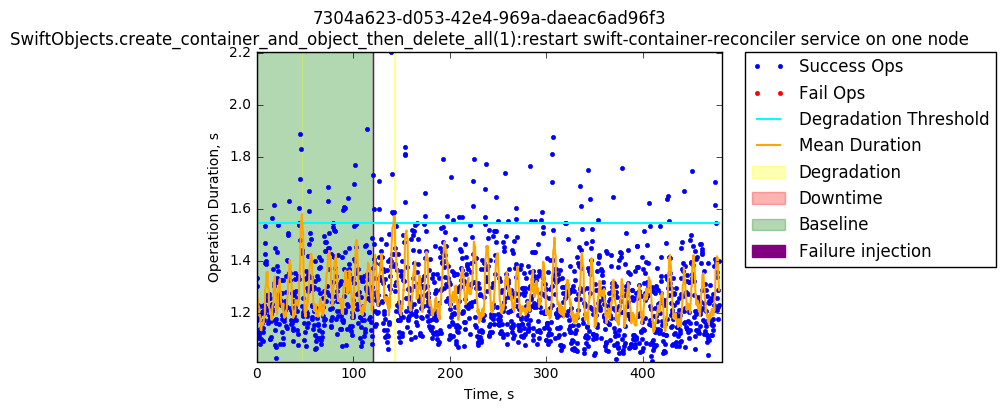
8.20 Restart swift-container-auditor service on one node



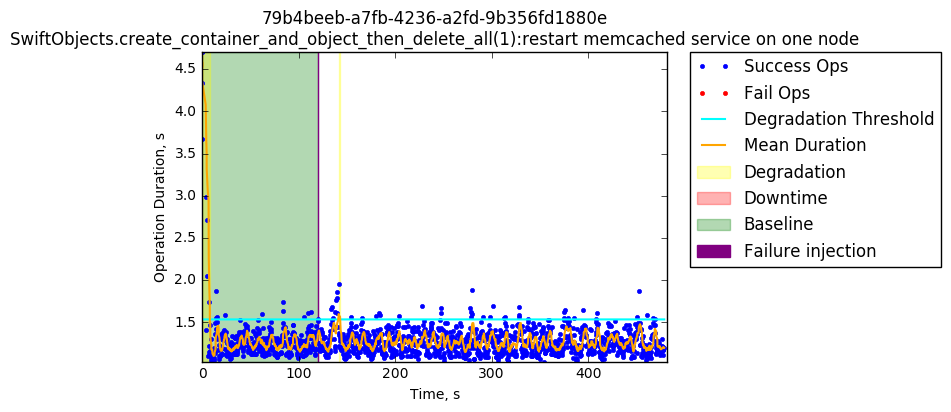
8.21 Restart swift-account-replicator service on one node



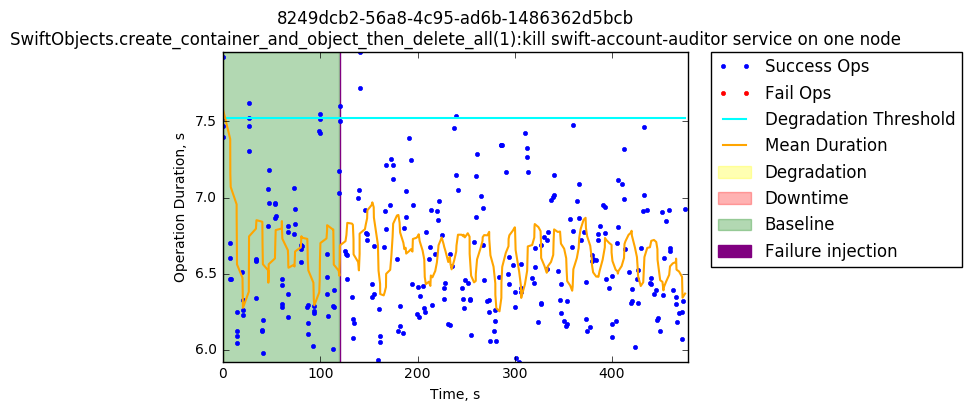
8.22 Restart swift-container-reconciler service on one node



8.23 Restart memcached service on one node

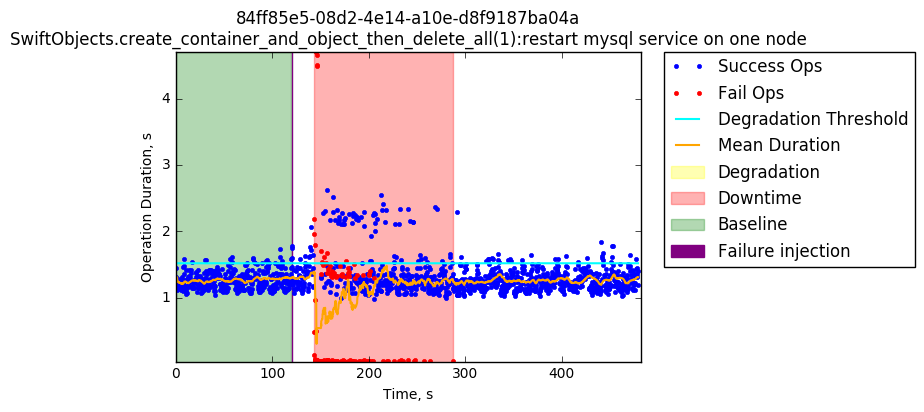


8.24 Kill swift-account-auditor service on one node

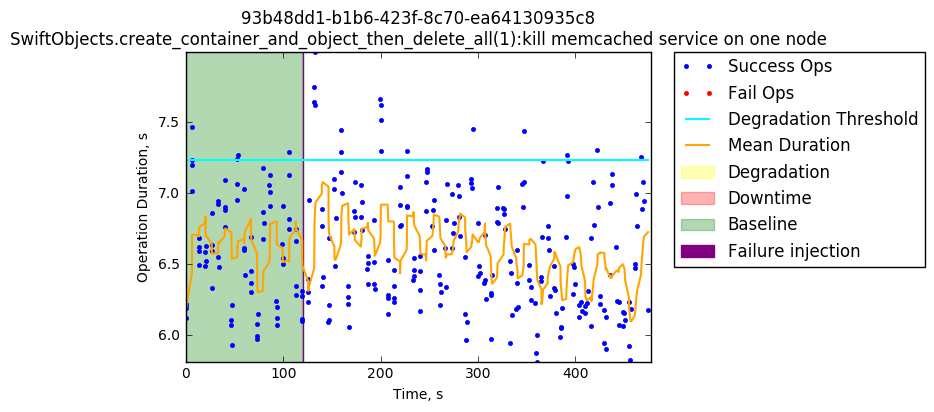


8.25 Restart MySQL service on one node

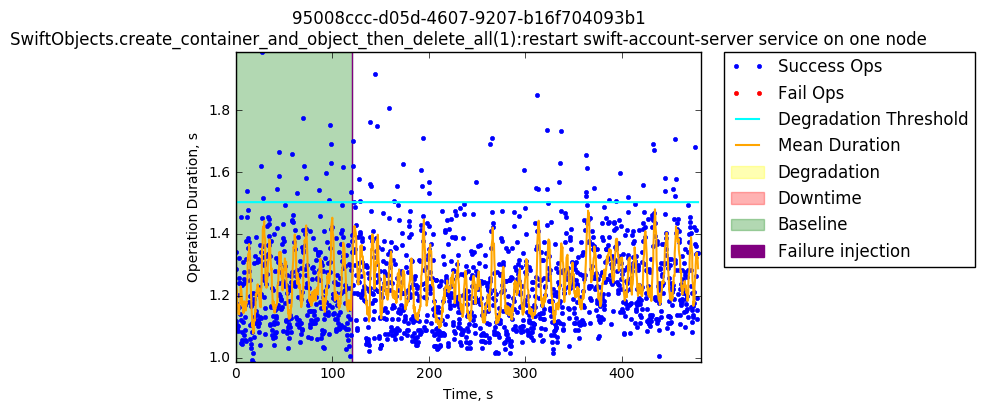
Restarting the MySQL service seems to cause significant damage to swift. After the failure injection, considerable degradation appears, going from approximately 1.5 seconds up to approximately 5 seconds duration along with failure operations. The failures keep showing up for around approximately 100 seconds along with successful operations, which means that there is not full downtime in this case, and performance goes back to normal in a short period of time.



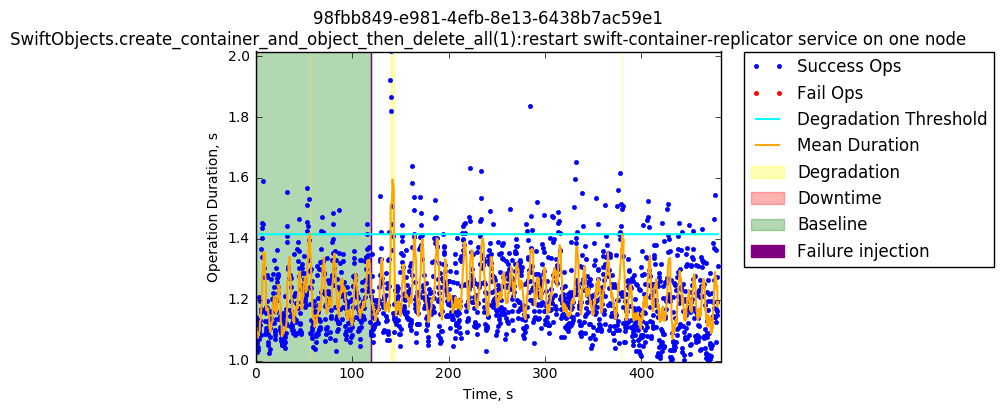
8.26 Kill memcached service on one node



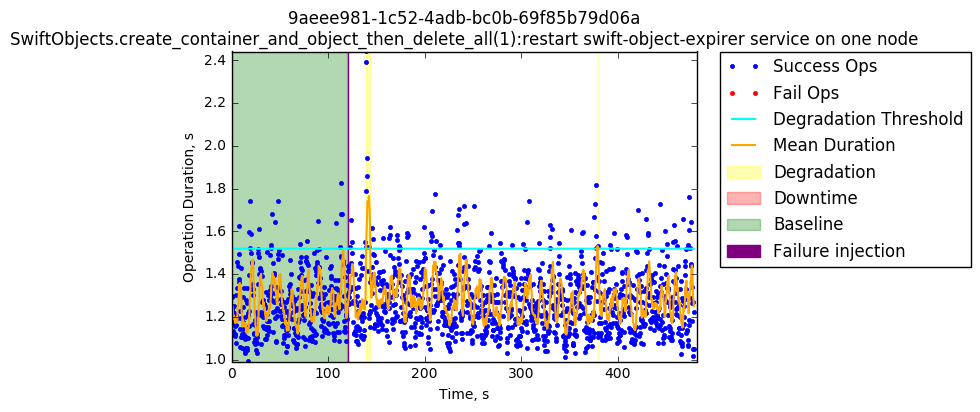
8.27 Restart swift-account-server service on one node



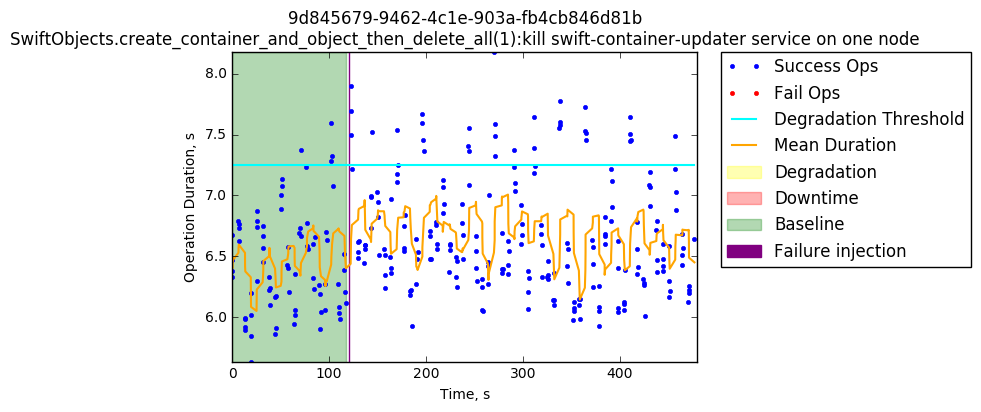
8.28 Restart swift-container-replicator service on one node



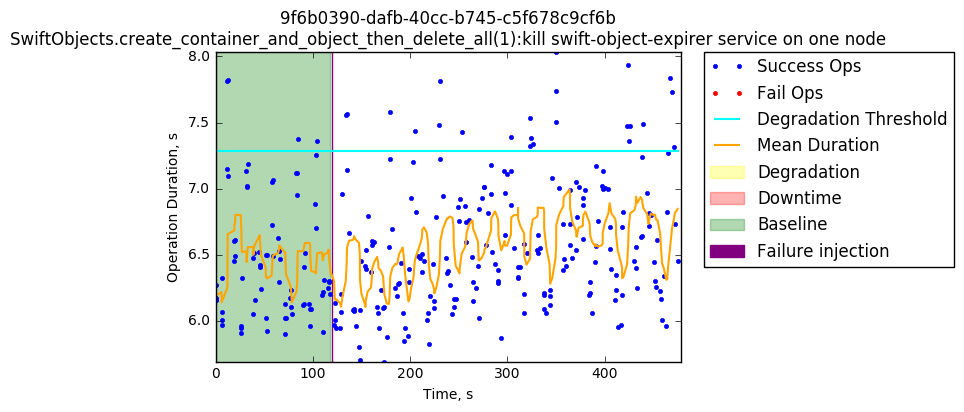
8.29 Restart swift-object-expirer service on one node



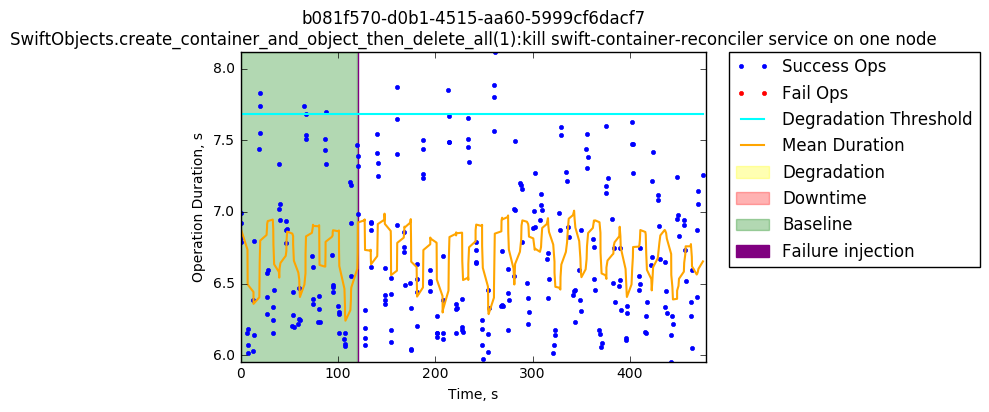
8.30 Kill swift-container-updater service on one node



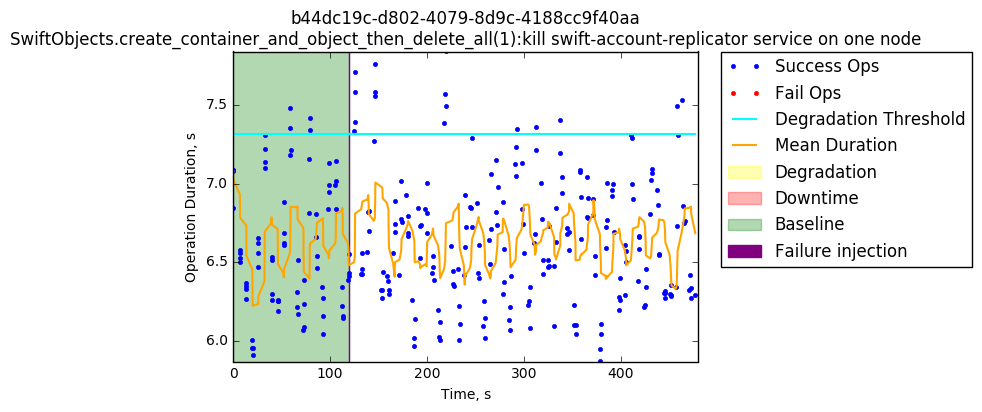
8.31 Kill swift-object-expirer service on one node



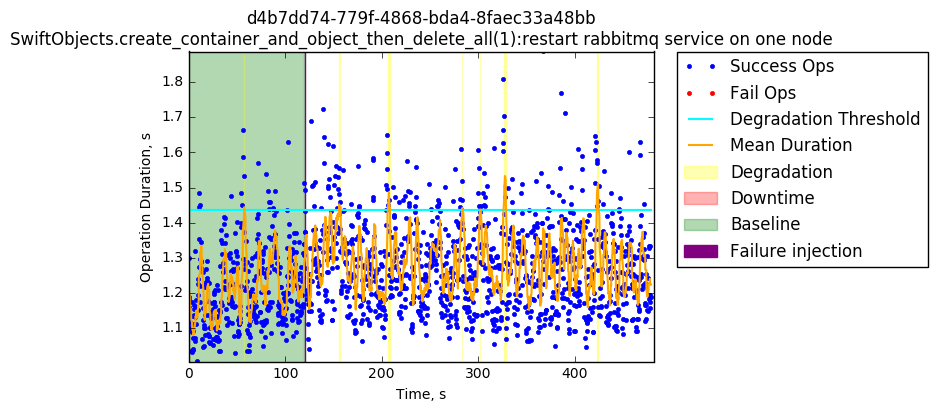
8.32 Kill swift-container-reconciler service on one node



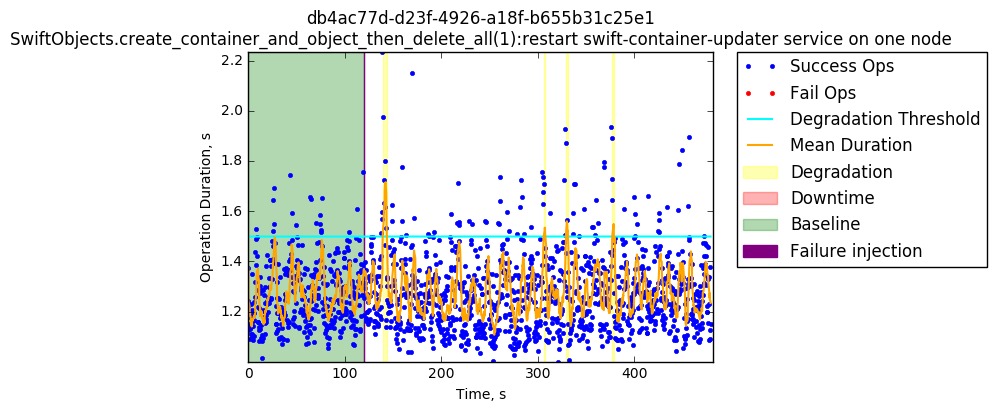
8.33 Kill swift-account-replicator service on one node



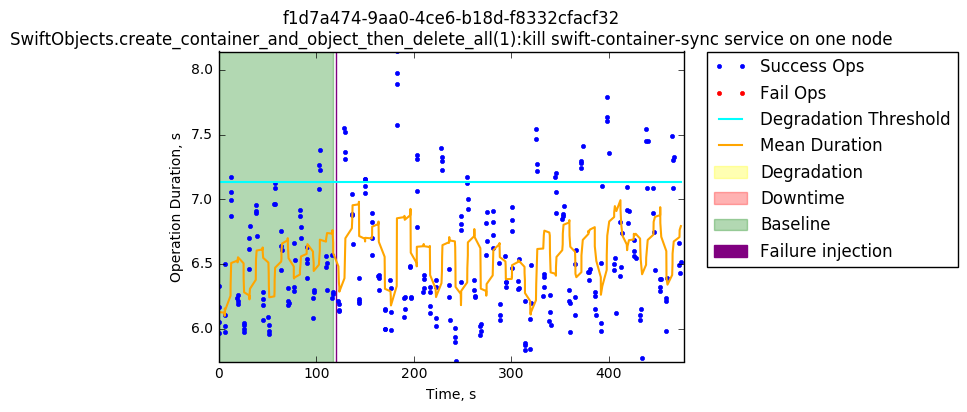
8.34 Restart RabbitMQ service on one node



8.35 Restart swift-container-updater service on one node



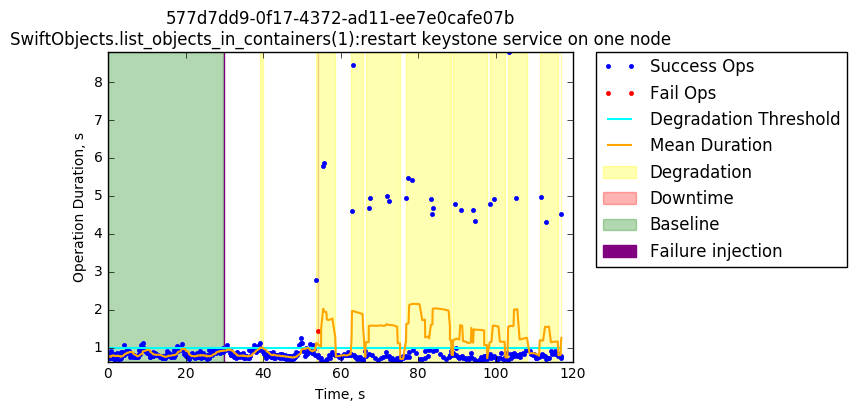
8.36 Kill swift-container-sync service on one node

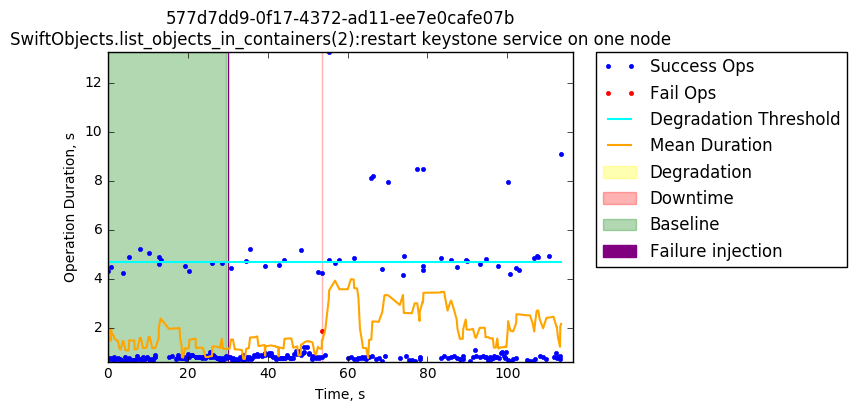


**9 SwiftObjects.list\_objects**

9.1 Restart Keystone service in one node

The same pattern of injecting Keystone restart failure is visible in this chart, causing a noticeable performance degradation to the service along with some failed operations. After the first task execution, the following executions operations with longer duration times happen more frequently which means that the service was not fully recovered.





# Summary and future directions

In summary, results overall indicated mostly positive trends for performance.

Average performance, and operation duration, remained despite the failures applied

to the testing environments.

API calls and service operations resisted failure injections. Mean degradation of services after failure injections indicate that OpenStack users can be certain their environment will run under pressure.

On common sub services (keystone, MySQL, Memcached, service metadata, and RabbitMQ) performance was erratic, with degradation changing rapidly throughout each test.

Common sub services were resistant to failure. The trend indicates OpenStack users can have confidence that commonly recruited service in an OpenStack environment will resist failure at scale.

The Swift Service resists failures. The OpenStack environment continued

operations even when Swift experienced degradation from failures, with some dispersion in results. The swift-account-reaper sub service performance changed after failure testing, with more widely ranging successful operations recorded. Swift-proxy-server results also experienced strong degradation after a failure test, however the service rapidly recovers with high availability. For OpenStack users that require nodes running object storage, the results indicate the swift service is a storage solution. MySQL failures are an ongoing issue, however. Restarting MySQL causes degradation of the swift service, and a slow recovery. The Glance service, similar to the common sub services, showed erratic operation performance and degradation, but ultimately remained running despite failures.

Neutron services remained functional after failure, however, depending on the sub service, eventually reported consistent operation failures. This indicates that while Neutron is resilient, as more performance pressure increases, the probability of failed networking operations also increases. Nova services only showed downtime following failure testing on the nova-api-os-compute sub service. All over operations running under the nova service showed reliable high availability.

Following testing at 22 nodes, OpenStack Users can be confident that their environment will continue operations, and is resilient to failures, with only limited instances of critical errors. The results have demonstrated that failure of a single service should not cause a failure of the system. The next goal for the project is extending the environment to a greater number of nodes, and exploring High Availability in a large scale OpenStack environment.

# Appendix A

## Bugs Filed:

<https://bugs.launchpad.net/openstack-ansible/+bug/1656086>

Issues Addressed:

<https://01.org/jira/browse/OSIC-904>

Neutron tests showed a linearly increasing task duration until failure ultimately resulted. Issue was due to test not removing networks until test completion and reaching limits set on the maximum number of networks allowed in the config. Resolved by modifying test to delete networks at the same rate they’re created, so average network count does not grow without bound during the test.

<https://01.org/jira/browse/OSIC-905>

An issue arose with the cinder services failing to work. Traced issue to NTP client failure. Filed bug [1656086](https://bugs.launchpad.net/openstack-ansible/+bug/1656086) to address this issue.

<https://01.org/jira/browse/OSIC-906>

Issue with attaching cinder volumes was traced to routes not being properly populated on two systems and required dropping and raising the network interfaces.

<https://01.org/jira/browse/OSIC-933>

Notable downtime occurs in certain scenarios where mysql is restarted.

<https://01.org/jira/browse/OSIC-934>

Transitive failure when restarting keystone

<https://01.org/jira/browse/OSIC-935>

This has the same vlxan limit issue as the other tests, and shows a transitive failure when restarting tests

<https://01.org/jira/browse/OSIC-936>

Transitive failures when restarting a service

<https://01.org/jira/browse/OSIC-937>

Transitive failures when restarting a service

<https://01.org/jira/browse/OSIC-938>

RabbitMQ does not automatically get restarted when it is killed.