IPAM Performance Test

Motivation

We wanted to be able to test the behavior of the IPAM controller's under various scenarios, by mocking and monitoring the edges that the controller interacts with. This has the following goals:

- Save time on testing
- To simulate various behaviors cheaply
- To observe and model the ideal behavior of the IPAM controller code

Currently the test runs through the 4 different IPAM controller modes for cases where the kube API QPS is a) equal to and b) significantly less than the number of nodes being added to observe and quantify behavior.

How to run

```
# In kubernetes root path
make generated_files

cd test/integration/ipamperf
./test-performance.sh
```

The runner scripts support a few different options:

```
./test-performance.sh -h
usage: ./test-performance.sh [-h] [-d] [-r <pattern>] [-o <filename>]
usage: ./test-performance.sh <options>
 -h display this help message
 -d enable debug logs in tests
 -r <pattern> regex pattern to match for tests
 -o <filename> file to write JSON formatted results to
 -p <id> enable cpu and memory profiles, output written to mem-<id>.out and cpu-
<id>.out
 -c enable custom test configuration
 -a <name> allocator name, one of RangeAllocator, CloudAllocator, IPAMFromCluster,
IPAMFromCloud
 -k <num> api server qps for allocator
 -n <num> number of nodes to simulate
 -m <num> api server qps for node creation
 -l <num> gce cloud endpoint qps
```

The tests follow the pattern TestPerformance/{AllocatorType}-KubeQPS{X}-Nodes{Y}, where AllocatorType is one of

- RangeAllocator
- IPAMFromCluster
- CloudAllocator
- IPAMFromCloud

and X represents the QPS configured for the kubernetes API client, and Y is the number of nodes to create.

The -d flags set the -v level for glog to 6, enabling nearly all of the debug logs in the code.

So to run the test for CloudAllocator with 10 nodes, one can run

```
./test-performance.sh -r /CloudAllocator.*Nodes10$
```

At the end of the test, a JSON format of the results for all the tests run is printed. Passing the -o option allows for also saving this JSON to a named file.

Profiling the code

It's possible to get the CPU and memory profiles of code during test execution by using the `-p` option. The CPU and memory profiles are generated in the same directory with the file names set to `cpu-<id>.out` and `cpu-<id>.out`, where `<id>` is the argument value. Typicall pattern is to put in the number of nodes being simulated as the id, or 'all' in case running the full suite.

Custom Test Configuration

It's also possible to run a custom test configuration by passing the -c option. With this option, it then possible to specify the number of nodes to simulate and the API server qps values for creation, IPAM allocation and cloud endpoint, along with the allocator name to run. The defaults values for the qps parmeters are 30 for IPAM allocation, 100 for node creation and 30 for the cloud endpoint, and the default allocator is the RangeAllocator.

Code Organization

The core of the tests are defined in ipam_test.go, using the t.Run() helper to control parallelism as we want to able to start the master once. cloud.go contains the mock of the cloud server endpoint and can be configured to behave differently as needed by the various modes. The tracking of the node behavior and creation of the test results data is in results.go.