PERF COUNTERS

The perf counters provide generic internal infrastructure for gauges and counters. The counted values can be both integer and float. There is also an “average” type (normally float) that combines a sum and num counter which can be divided to provide an average.

The intention is that this data will be collected and aggregated by a tool like `collectd` or `statsd` and fed into a tool like `graphite` for graphing and analysis.

ACCESS

The perf counter data is accessed via the admin socket. For example:

```
ceph --admin-daemon /var/run/ceph/ceph-osd.0.asok perf schema
ceph --admin-daemon /var/run/ceph/ceph-osd.0.asok perf dump
```

COLLECTIONS

The values are grouped into named collections, normally representing a subsystem or an instance of a subsystem. For example, the internal `throttle` mechanism reports statistics on how it is throttling, and each instance is named something like:

```
throttle-msgr_dispatch_throttler-hbserver
throttle-msgr_dispatch_throttler-client
throttle-filestore_bytes
...
```

SCHEMA

The `perf schema` command dumps a json description of which values are available, and what their type is. Each named value as a `type` bitfield, with the following bits defined.

bit	meaning
1	floating point value
2	unsigned 64-bit integer value
4	average (sum + count pair)
8	counter (vs gauge)

Every value with have either bit 1 or 2 set to indicate the type (float or integer). If bit 8 is set (counter), the reader may want to subtract off the previously read value to get the delta during the previous interval.

If bit 4 is set (average), there will be two values to read, a sum and a count. If it is a counter, the average for the previous interval would be sum delta (since the previous read) divided by the count delta. Alternatively, dividing the values outright would provide the lifetime average value. Normally these are used to measure latencies (number of requests and a sum of request latencies), and the average for the previous interval is what is interesting.

Here is an example of the schema output:

```
{
  "throttle-msgr_dispatch_throttler-hbserver" : {
     "get_or_fail_fail" : {
        "type" : 10
     },
     "get_sum" : {
        "type" : 10
     },
     "max" : {
        "type" : 10
     },
     "put" : {
        "type" : 10
     },
     "val" : {
        "type" : 10
     },
     "take" : {
        "type" : 10
     },
     "get_or_fail_success" : {
        "type" : 10
     },
     "wait" : {
        "type" : 5
     },
     "get" : {
        "type" : 10
     },
     "take_sum" : {
        "type" : 10
     },
     "put_sum" : {
        "type" : 10
     }
  },
  "throttle-msgr_dispatch_throttler-client" : {
     "get_or_fail_fail" : {
        "type" : 10
     },
     "get_sum" : {
        "type" : 10
     },
     "max" : {
        "type" : 10
     },
     "put" : {
        "type" : 10
     },
     "val" : {
        "type" : 10
     },
     "take" : {
        "type" : 10
     },
     "get_or_fail_success" : {
        "type" : 10
     },
     "wait" : {
        "type" : 5
     },
     "get" : {
        "type" : 10
     },
     "take_sum" : {
        "type" : 10
     },
     "put_sum" : {
        "type" : 10
     }
  }
}
```

DUMP

The actual dump is similar to the schema, except that average values are grouped. For example:

```
{
  "throttle-msgr_dispatch_throttler-hbserver" : {
     "get_or_fail_fail" : 0,
     "get_sum" : 0,
     "max" : 104857600,
     "put" : 0,
     "val" : 0,
     "take" : 0,
     "get_or_fail_success" : 0,
     "wait" : {
        "avgcount" : 0,
        "sum" : 0
     },
     "get" : 0,
     "take_sum" : 0,
     "put_sum" : 0
  },
  "throttle-msgr_dispatch_throttler-client" : {
     "get_or_fail_fail" : 0,
     "get_sum" : 82760,
     "max" : 104857600,
     "put" : 2637,
     "val" : 0,
     "take" : 0,
     "get_or_fail_success" : 0,
     "wait" : {
        "avgcount" : 0,
        "sum" : 0
     },
     "get" : 2637,
     "take_sum" : 0,
     "put_sum" : 82760
  }
}
```