Quota Update Design Doc

NOTE: This is a public document.

Authors	Zhitao Li (zhitaoli.cs@gmail.com)
Shepherd	Joris Van Remoortere (joris@mesosphere.io) Alex Rukletsov (rukletsov@gmail.com)
Revision	0.4
Last Revision	September 14, 2016 (after 1.0 release)
Status	Ready for dev@ review

Overview

Quota is introduced in Mesos 0.27. One caveat of the initial implementation is that operator cannot update an existing quota for a particular role. This is a short design doc of how to implement this feature.

JIRA issue: https://issues.apache.org/jira/browse/MESOS-4941

Use Case

To change an existing quota without a quota update API, an operator has to 1) delete the existing quota and 2) create a new quota for the given role.

Consider the case of updating quota for a role *R* in a contentious cluster. Because 1) delete and 2) (re)create are not done in an atomic operation, resources previously allocated to role *R* could be allocated to other roles between 1) and 2), and there is no guarantee that Mesos could claim enough resources to really satisfy the new quota.

Therefore, we are proposing to implement an atomic update of an existing quota.

Proposed Implementation

HTTP Endpoint

As proposed in original quota design doc, updating quota uses PUT to `/quota` endpoint:

```
$ curl -d resources="[{
         {
           "name" : "cpus",
           "type" : "SCALAR",
           "scalar" : { "value" : 8 },
           "role" : "ads",
         }
       }]"
       -X PUT http://<master-ip>:<port>/quota/ads
```

Similar to Set Quota operation, the operator will receive one of the following HTTP response codes:

- 200 OK: Success (the update request was successful).
- 400 BadRequest: Invalid arguments (e.g., quota for role not exists, invalid JSON, non-scalar resources included).
- 401 Unauthorized: Unauthenticated request.
- 403 Forbidden: Unauthorized request.
- 409 Conflict: The capacity heuristic check failed due to insufficient resources

Note: We will allow both add and update in PUT method, and deprecate POST method (which only support adding new quota, but not updating existing quota). This behavior is consistent with

HTTP methods on the “weights” endpoint, and keeps current action unchanged to avoid surprising mistake.

Authorization Change

~~We will also introduce another authorization action `authorization::UPDATE_QUOTA_WITH_ROLE`, and use it to authorize update quota case.~~

~~Open question: Should we start to consolidate `SET_QUOTA_WITH_ROLE` and `DESTROY_QUOTA_WITH_PRINCIPAL` into the new `UPDATE_QUOTA_WITH_ROLE` action? This will require upgrade doc to instruct cluster operators to set same content in their ACL for these actions to prepare for the upcoming deprecation.~~

After Mesos 1.0, quota related ACLs have been redesigned and a single `` `UpdateQuota` `` ACL has been introduced to cover both Set/Update/Delete, so no additional change to Authorization part is necessary.

Capacity Check

If the “force” flag is set, capacity check will be bypassed.

Otherwise: if the scalar value of a resource type is larger than the existing quota, or a new resource type is included in the quota update request, the existing capacity heuristic check will be invoked to verify that the total quota, including the increased resources in updated request, does not exceed the sum of total non-statically-reserved cluster resources.

Offer Rescinding

If the scalar value of any resource type is larger than the existing quota, or a new resource type is included in the quota update request, the following offer rescinding operations will happen (similar to setting new quota case):

- Rescind at least as many resources required to fulfill the increased part of the updated quota request.
- If at least one offer is to be rescinded from an agent, all offers from this agent are rescinded. This is done in order to make the potential offer bigger, which increases the chances that a quota'ed framework will be able to use the offer.

Note: because current form of quota is “guarantee” only, we decide not to rescind offers from the quota’ed role for decreased quota case. This could result in over-allocation until offers are rejected or timed out, or manual churn triggered by operator. The potential fair sharing issue will be addressed with future work on quota like implementing limit support.

Storing New Quota

We will create a new `HierarchicalAllocatorProcess::updateQuota()` function to store the newly updated quota.

Storing a new quota in registrar is already implemented and tested in `RegistrarTest.UpdateQuota()`.

Track Overallocation above Quota

In Mesos 1.0, we introduced two metrics for each (role, resource) in a quota:

- `allocator/mesos/quota/roles/<role>/resources/<resource>/offered_or_allocated`
- `allocator/mesos/quota/roles/<role>/resources/<resource>/guarantee`

The metrics.cpp helper class will be modified so that these metrics gauges are updated when a quota guarantee is changed.

In first implementation, we will not have code to actively correct over-allocation above guarantee. Instead, we will advise in quota.md documentation that it is the responsibility of cluster operator to monitor over-allocation through `/metrics/snapshot` for any over-allocation above quota guarantee, and perform one of the following actions:

- Wait for tasks to finish so that the resources can be re-allocated, or
- Manually kill some tasks in the over-allocated role
    - (how do we recommend to do this?) Once we have preemption support in Mesos master, this can be simplified with automatic preemption.

~~The information of `/quota` endpoint will be modified to also include a “resources” field containing current allocation for each roles with a quota. Right now this information is only in `/roles` endpoint. Cluster operator can query this endpoint and generate alarms for any allocation above quota, and perform proper actions to claim resources from the given role.~~

```
~~{~~
~~    "infos": [~~
~~    {~~
~~      "guarantee": [~~
~~        {~~
~~          "name": "cpus",~~
~~          "role": "*",~~
~~          "scalar": {~~
~~            "value": 1.0~~
~~          },~~
~~          "type": "SCALAR"~~
~~        }~~
```

```
      ],
      "resources": [
        {
          "name": "cpus",
          "role": "*",
          "scalar": {
            "value": 2.0
          },
          "type": "SCALAR"
        }
      ],
      "role": "test-role"
    }
  ]
}
```

The above example indicates cpus resource is overallocated for “test-role”.

Impact to HierarchicalAllocatorProcess::allocate()

No code change is expect to happen in `HierarchicalAllocatorProcess::allocate()` to implement feature proposed in this doc, because the first stage for satisfying quota guarantee in current implementation is sufficient for increased/new resources.

NOTE: we will not attempt to correct over-allocation in first implementation. It will be up to the operator to ensure quota allocation falls within quota.