Lease/Reclaim Extension to Yarn

Jessie Yu, Garry Weng, Shuguang Liu

1. Problem

In some clusters outside of Yarn, the machines resources are not fully utilized, e.g., resource usage is quite low at night.
To better utilize the resources while keep the existing SLA of the cluster, Lease/Reclaim Extension to Yarn is introduced.

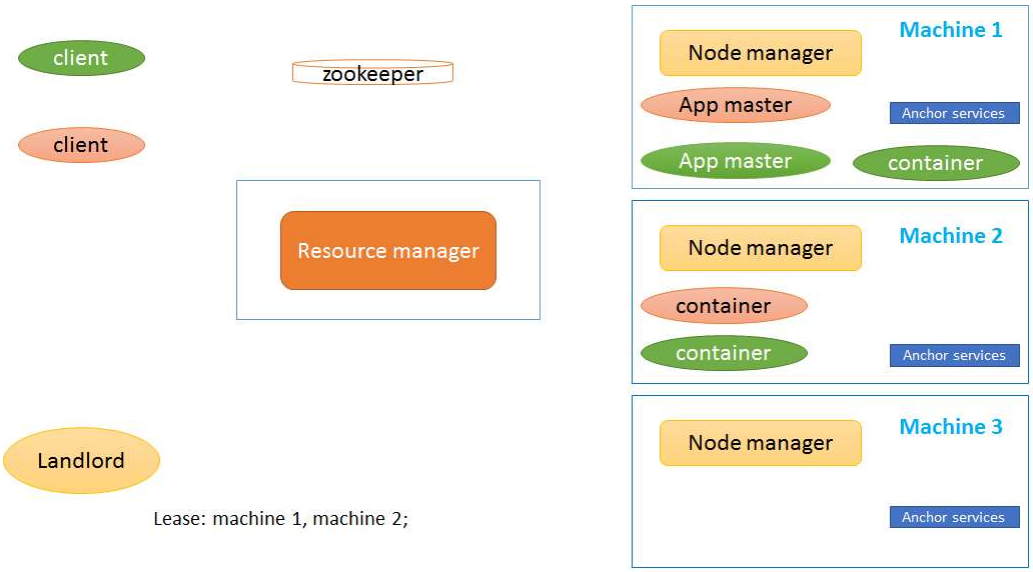
2. Goals

- Define rules for lease/reclaim, the rules could be time based.
- Cluster admin can lease/reclaim machines to Yarn easily, without changing other machines.
- The machines can be scheduled to run yarn tasks after it's leased.
- The leased machines will be recovered after it's reclaimed.

3. Features Details

	Lease/reclaim
Operation	Support command by yarn.cmd
Resource update	Support resource update without restarting NM.
Timeout support	If user leases a NM with a period of time, when timeout, all the resource will be reclaimed.
High availability	Persist all meta-data information about lease for recovery. (Only supports ZooKeeper-based store currently)
Resource Safety	Containers will NOT over-use resources on the node; - When reclaim NM or reduce the NM resources, it will kill containers properly to avoid over-use resources.

4. Architecture



4.1 Zookeeper

Use zookeeper to persist the lease/reclaim related info;

4.2 Landlord

The admin to lease/reclaim NMs, aka landlord;

Landlord leases/reclaims the NMs by:

- Persist the operation request to the configured zookeeper path directly;

4.3 RM

Do some changes for RM:

- Watch zookeeper, and handle lease requests;
- Recover lease info from zookeeper when boot up;
- Use LeaseManager service to maintain Lease status and configured capability for each machine;
- Preempt containers when NM resource is over used;
- Check whether Lease Status and configured capability is changed while receiving NM's HB request, and update resources if needed;

4.4 NM

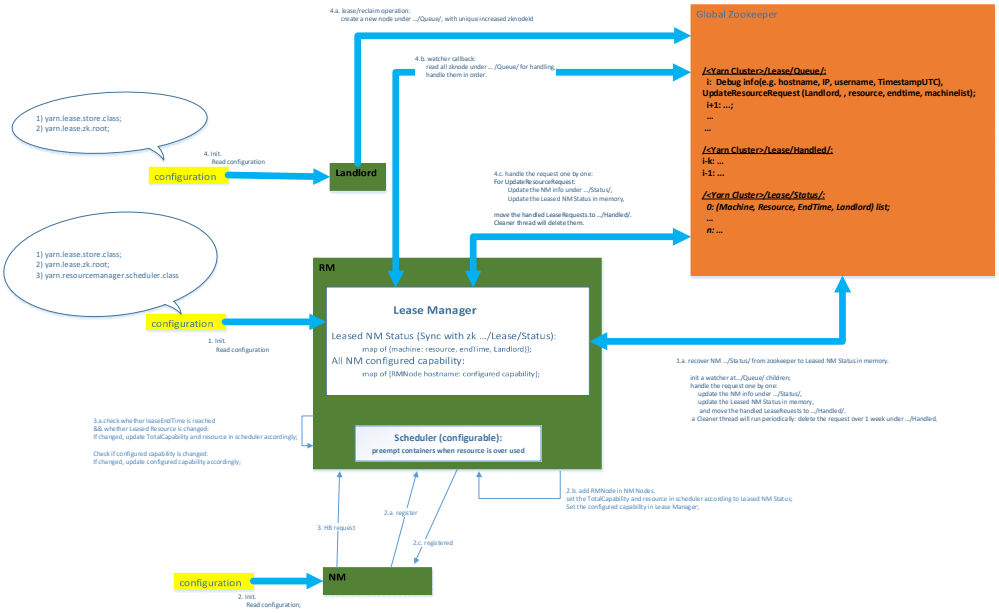
No changes for NM.

NM will run on all the machines.

When a machine is leased, its NM will cooperate with RM to run yarn containers on the machine.

If a machine is not leased or reclaimed, no yarn containers can run on it: make sure the machine's existing services won't impacted.

4.5 Workflow



Lease Manager in RM

Handle lease requests sequentially.

- Handle the unhandled requests when boot up.
- Watch the zookeeper requests' path children, once there're new requests, handle them sequentially.
- Clean the handled requests when they're too old, e.g., over 1 week.

Note:

- It's the only writer for Leased NM Status table (in memory) and /<Yarn Cluster>/Lease/Status/ (in zookeeper);
- Lease Manager will failover with RM, so, only one Lease Manager works at one time.

Lease Info Recover

- All the lease status is persisted in zookeeper.
  No matter RM restarts or failover, the new started RM will recover the lease status when booting up.
- All the lease requests are persisted sequentially in zookeeper, and the handled requests' indexes are moved to …/Handled/ path.
  If RM crashes and exists while handling a lease request, the new started RM will first recover the lease status, then, handle the unhandled requests again. After

the request is handled OK, and the related lease status info is updated in zookeeper successfully, the handled request will be moved to …/Handled/.

Preemptable Schedulers

PreemptCapacityScheduler: a simple preemptable scheduler inherit from capacityScheduler. Preempt containers when containers resource is over used;

5. Operation

5.1 Configuration

Landlord:

1) yarn.lease.store.class=ZookeeperLeaseReclaimStore;
2) yarn.lease.zk.root=/Lease;
3) Use zk configurations defined by “yarn.resourcemanager.zk-”;

RM:

1) yarn.lease.store.class=ZookeeperLeaseReclaimStore;
2) yarn.lease.zk.root=/Lease;
3) Use zk configurations defined by “yarn.resourcemanager.zk-”;
4) Normalize the hostname to avoid mismatching.
   The default method is normalizing the hostname to lowercase.
   User can also use special normalize methods defined by themselves;
5) Use Yarn.resourcemanager.scheduler.class to choose preemptable scheduler:
   Yarn.resourcemanager.scheduler.class=PreemptCapacityScheduler

5.2 Commands

yarn.cmd landlord -leaseNodes <node1,node2,...> <resource> <endtime>
yarn.cmd landlord -reclaimNodes <node1,node2,...>
yarn.cmd landlord -getLeasedNodes -landlord <landlord>
yarn.cmd landlord -getLeasedNodes -all

Note:

Suggest the minimum lease duration is 1 hour.

6. Interfaces

6.1 Proto types stored in zookeeper:

```
message OperationClientInfoProto {
  string username = 1;
  string hostname = 2;
  string address = 3;
  int64  timestamp = 4;
  string note = 5;
}
message UpdateResourceRequestProto {
  string landlordID = 1;
  ResourceProto resource = 2;
  int64 endTime = 3;
  repeated string machines = 4;
}
```

```
message LeaseRequestProto {
  OperationClientInfoProto clientDebugInfo = 0;
  UpdateResourceRequestProto updateResource = 1;
  UpdateIDRequestProto updateID = 2;
}

message LeaseNodeProto {
  ResourceProto resource = 1;
  int64 endTime = 2;
  string landlordID = 3;
}
message MachineLeaseInfoProto {
  string machine = 1;
  LeaseNodeProto status = 2;
}

message MachineLeaseInfoMapProto {
  repeated MachineLeaseInfoProto nodeToStatus = 1;
}
```

6.2 API:

```
// Lease the NM resource to RM.
void lease(Set<String> hostnames, Resource resource, long endTime) throws YarnException,
IOException;

// Reclaim the NM resource from RM.
void reclaim(Set<String> hostnames) throws YarnException, IOException;

// return Leased and not-expired nodes, with each node's lease info
Map<String, LeaseNode> getLeasedNodes(boolean all, String landlord)
    throws YarnException, IOException;

//set landlord Id for identify landlord, if needed
void setLandlordId(String landlordId);
```

7.Future Work

Add configurations at NM side to decide:

When NM is disconnect of RM for a while, whether kill running containers to make the NM resource safety.