Tensorflow on slider introduction

Motivation

- Enable YARN and Slider to have the capability to run tensorflow cluster, which contains a set of "tasks" that participate in the distributed execution of a TensorFlow graph.
- Take the advantage of YARN to manage cluster resources and dynamically generate clusterSpec for tensorflow

Goal

- Support two kinds of running mode
  - Run tensorflow scripts in docker(INCLUDED)
  - Run tensorflow scripts in local machine, need to deploy environment well in advance, Pip Installation (https://www.tensorflow.org/versions/r0.11/get_started/os_setup.html#pip-installation) (INCLUDED)
- Support running tensorflow application on CPU and GPU
  - All component instances are running on CPU (INCLUDED)
  - All component instances are running on GPU (INCLUDED)
  - Part of component instances are running on CPU, the rest are on GPU (TODO)
- Support failover
  - When a component instance(such as ps server) fails and exits, use the new clusterSpec to restart the whole tensorflow cluster (INCLUDED)
  - When a component instance(such as ps server) fails and exits, use the same port to re-launch, continue running; If exceed the limit of failure counts, use the same port to launch on another machine(need to use the YARN DNS to generate clusterSpec at startup) (TODO)
- Support to finish gracefully when succeed, properly handle the exit code of tensorflow script
- Support to view the log and output of tensorflow in YARN web ui
- Support the existing monitoring tools and analytical tools for tensorflow

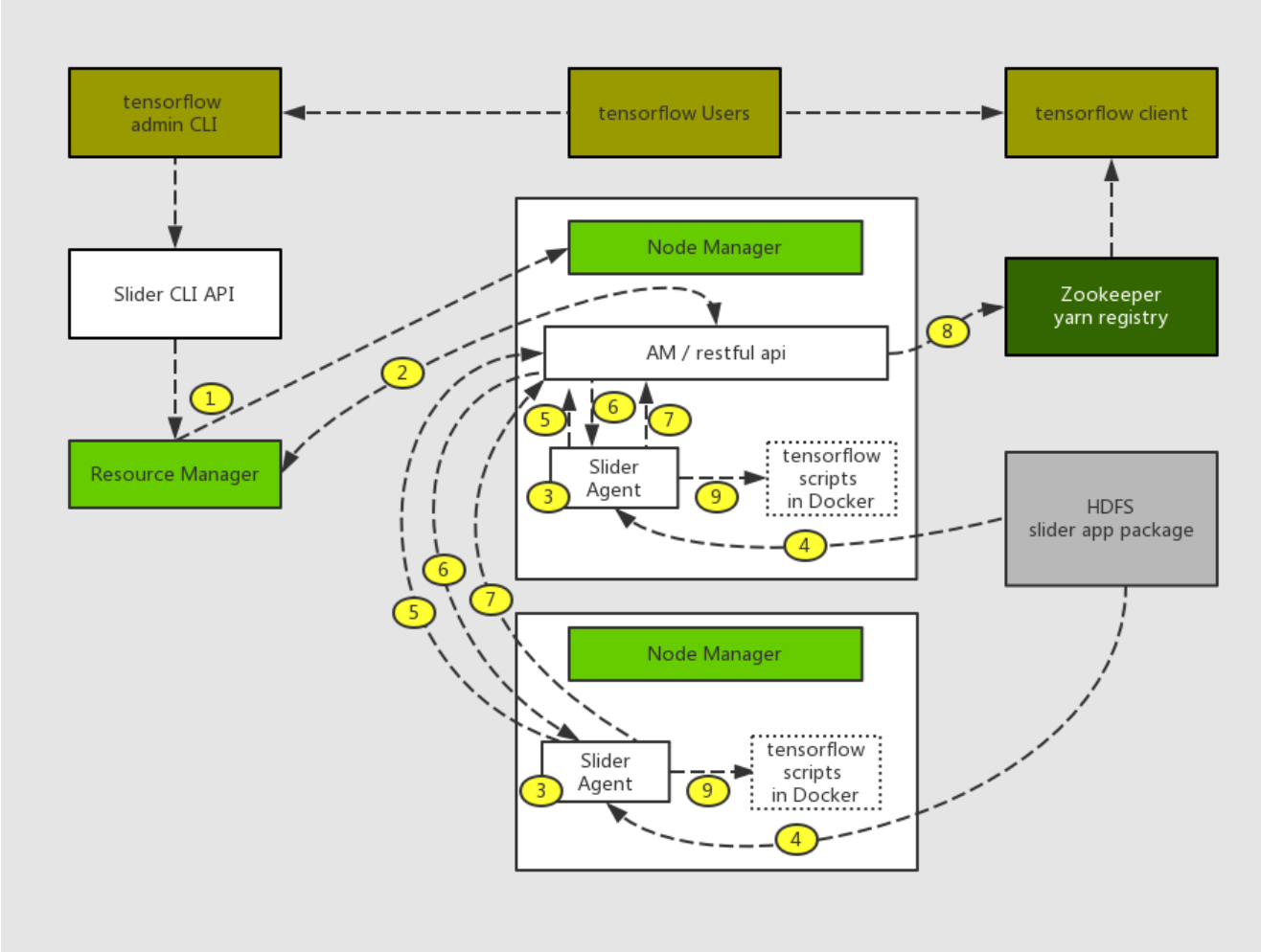
```
"INCLUDED" means that the feature will be done in this JIRA at the first version.
"TODO" means that we have taken them into account and should be discussed more.
```

Requirements

- Add support in the NodeManager to re-launch containers, YARN-3998 (https://issues.apache.org/jira/browse/YARN-3998)

- Add support for GPU as a resource, YARN-4122 (https://issues.apache.org/jira/browse/YARN-4122)
  - A workaround is using Node Label and Anti-affinity
- Simplified discovery of services via DNS mechanisms, YARN-4757 (https://issues.apache.org/jira/browse/YARN-4757)
- Support for short-lived services, SLIDER-494 (https://issues.apache.org/jira/browse/SLIDER-494)
- Support gang scheduling in the AM RM protocol, YARN-624 (https://issues.apache.org/jira/browse/YARN-624)
  - A workaround is waiting for all ports exported

Design



1. Tensorflow User submit application to RM through Slider CLI. Then RM notifies NM to start AM
2. AM ask resources from RM
3. NM launch container, and start Slider Agent
4. Agent fetch app package from HDFS
5. Agent register with AM
6. AM send 'START' command to Agent
7. Agent report the status, port, configurations etc. to AM
8. AM register with YARN registry, including REST addresses and IPC addresses
9. Agent call start method in tensorflow.py and execute tensorflow scripts(user codes)

```
RM: Resource Manager
NM: Node Manager
AM: Application Master
```

Implementation

- Add configuration items in appConfig-default.json to allocate dynamic port, will be exported by REST APIs. All component instances could get the exported ports by REST API, http://{AM_HOST}:1025/ws/v1/slider/publisher/exports

- class Tensorflow(Script)
  - Implement start, stop, status and other methods
  - When start, constantly request the AM REST API to get exported ports. Until all ports are exported, build the clusterSpec to run tensorflow scripts(user codes).

Sample

Scenario 1: Between-graph replication

Commands to run:

```
slider create [app-name] --appdef app-packages/tensorflow --template appConfig-
default.json --resources resources.default.json
```

Package:

```
app-packages/tensorflow/
├── appConfig-default.json
├── metainfo.json
├── package
│   ├── files
│   │   └── trainer.py
│   └── scripts
│       ├── params.py
│       ├── tensorflow_service.py
│       └── tensorflow.py
└── resources.default.json
```

Configuration files:

metainfo.json

```
{
  "schemaVersion": "2.1",
  "application": {
    "name": "tfyarn",
    "exportGroups": [
      {
        "name": "ps",
        "exports": [
          {
            "name": "host_port",
            "value": "${ps_HOST}:${site.global.ps.port}"
          }
        ]
      },
      {
        "name": "worker",
        "exports": [
          {
            "name": "host_port",
            "value": "${worker_HOST}:${site.global.worker.port}"
          }
        ]
      }
    ],
    "components": [
      {
        "name": "ps",
        "compExports": "ps-host_port",
        "commandScript": {
          "script": "scripts/tensorflow.py",
          "scriptType": "PYTHON"
        }
      },
      {
        "name": "worker",
        "compExports": "worker-host_port",
        "commandScript": {
          "script": "scripts/tensorflow.py",
          "scriptType": "PYTHON"
        }
      }
    ],
    "packages": [
      {
        "type": "folder",
        "name": "files"
      }
    ]
  }
}
```

```
appConfig-default.json
{
  "schema": "http://example.org/specification/v2.0.0",
  "metadata": {
  },
  "global": {
    "site.global.ps.port": "${ps.ALLOCATED_PORT}{PER_CONTAINER}",
    "site.global.worker.port": "${worker.ALLOCATED_PORT}{PER_CONTAINER}",
    "site.global.run.mode": "docker",
    "site.global.user.scripts.entry": "trainer.py"
  }
}
```

```
resources.default.json
{
 "schema" : "http://example.org/specification/v2.0.0",
 "metadata" : {
 },
 "global" : {
 },
 "components": {
  "slider-appmaster": {
  },
  "ps": {
    "yarn.role.priority": "2",
    "yarn.component.instances": "2",
    "yarn.memory": "1024"
  },
  "worker": {
    "yarn.role.priority": "1",
    "yarn.component.instances": "2",
    "yarn.memory": "2056"
  }
 }
}
```

Scenario 2: In-graph replication

TODO