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

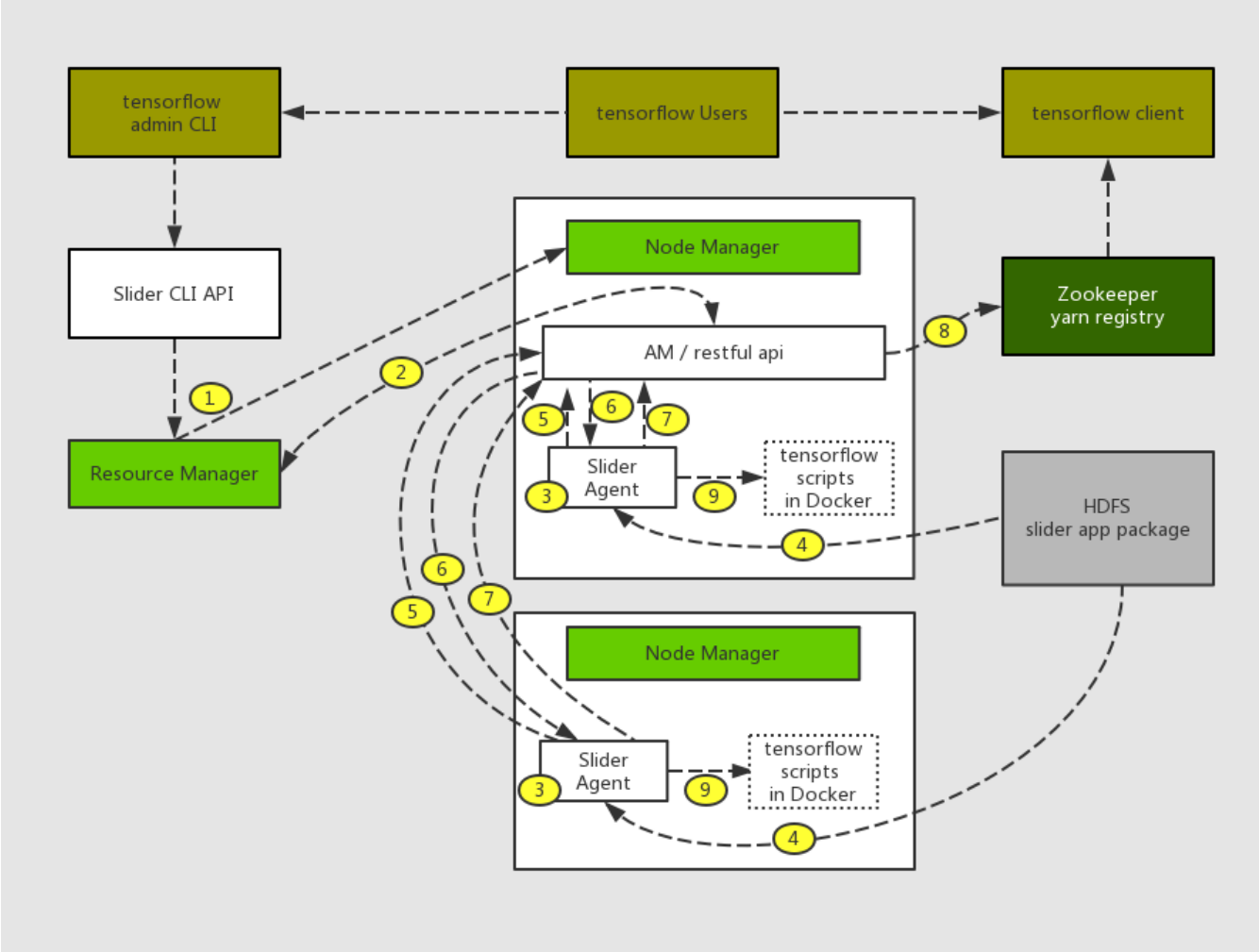
- Support two kinds of running mode
  - Run tensorflow scripts in docker (INCLUDED)
  - Run tensorflow scripts in local machine, need to deploy environment well in advance (TODO)
- Support running tensorflow application on CPU and GPU
  - All component instances are running on CPU (INCLUDED)
  - All component instances are running on GPU (TODO)
  - Part of component instances are running on CPU, the rest are on GPU (TODO)
- Support failover
  - When a component instance(such as ps/worker) fails and exits, slider agent relaunch it with the same configuration, restore from checkpoint and continue to run (INCLUDED)
  - When slider agent fails and exits, it may be rescheduled to other machine. We should use the same host:port to re-launch, restore from checkpoint and continue to run.(need to use the YARN DNS to generate clusterSpec at startup) (TODO)
- Support to finish gracefully when succeed
  - User could stop tensorflow cluser manually (INCLUDED)
  - Use CLI-RM protocol / restAPI to stop tensorflow cluster when all workers finish successfully (INCLUDED)
  - Properly handle the exit code of tensorflow script and finish the tensorflow cluster automatically (TODO)
- Support to view the log of tensorflow in YARN web ui (INCLUDED)
- Support the existing monitoring tools and analytical tools for tensorflow (such as TensorBoard) (INCLUDED)

```
"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 for GPU as a resource, YARN-4122 (https://issues.apache.org/jira/browse/YARN-4122)
  - A workaround is using Node Label and Anti-affinity
- Add support in the NodeManager to re-launch containers, YARN-3998 (https://issues.apache.org/jira/browse/YARN-3998)
- 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)
  - A workaroud is use CLI-RM protocol to stop cluster
- Support gang scheduling in the AM RM protocol, YARN-624 (https://issues.apache.org/jira/browse/YARN-624)
  - A workaround is holding on to wait for all ports exported in slider agent

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://{RM_HOST:PORT}/proxy/{applicationId}/ws/v1/slider/publisher/exports

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

Dockerfile

https://github.com/tensorflow/ecosystem/blob/master/docker/Dockerfile.hdfs

```
FROM tensorflow/tensorflow:nightly

# Install java
RUN add-apt-repository -y ppa:openjdk-r/ppa && \
    apt-get update && \
    apt-get install -y --no-install-recommends \
    openjdk-8-jdk openjdk-8-jre-headless && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

ENV HADOOP_VERSION 2.7.3
RUN curl -O http://www-us.apache.org/dist/hadoop/common/hadoop-${HADOOP_VERSION}/hadoop-${HA
    tar xfz hadoop-${HADOOP_VERSION}.tar.gz && \
    mv hadoop-${HADOOP_VERSION} /usr/local/hadoop && \
    rm hadoop-${HADOOP_VERSION}.tar.gz

ENV JAVA_HOME /usr/lib/jvm/java-8-openjdk-amd64
ENV HADOOP_INSTALL /usr/local/hadoop
ENV HADOOP_HDFS_HOME $HADOOP_INSTALL
ENV HADOOP_COMMON_LIB_NATIVE_DIR $HADOOP_INSTALL/lib/native
ENV HADOOP_OPTS "-Djava.library.path=$HADOOP_INSTALL/lib/native"
ENV LD_LIBRARY_PATH $LD_LIBRARY_PATH:$JAVA_HOME/jre/lib/amd64/server

ENV CLASSPATH /usr/local/hadoop/etc/hadoop:/usr/local/hadoop/share/hadoop/common/lib/httpcor
```

Slider config

Scenario 1: Between-graph replication

Use slider 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
├── bin
│   ├── ytensorflow
│   └── ytensorflow.py
├── config.json
├── Dockerfile
├── metainfo.json
├── package
│   ├── files
│   │   └── mnist.py
│   └── scripts
│       ├── functions.py
│       ├── params.py
│       ├── tensorflow.py
│       └── tensorflow_service.py
└── resources.default.json
```

Configuration files:
metainfo.json

```
{
  "schemaVersion": "2.1",
  "application": {
    "name": "tensorflow",
    "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}"
          }
        ]
      },
      {
        "name": "tensorboard",
        "exports": [
          {
            "name": "url",
            "value": "http://${tensorboard_HOST}:${site.global.tensor.board.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"
        }
      },
      {
```

```
        "name": "tensorboard",
        "compExports": "tensorboard-url",
        "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.hadoop.conf": "/etc/hadoop/conf",
    "site.global.user.name": "${USER_NAME}",
    "site.global.zookeeper.quorum": "${ZK_HOST}",
    "site.global.user.scripts.entry": "mnist.py",
    "site.global.user.data.dir": "hdfs://hdpalt/user/danrtsey.wy/mnist-data",
    "site.global.user.checkpoint.dir": "hdfs://hdpalt/user/${USER_NAME}/.slider/cluster",
    "site.global.ps.port": "${ps.ALLOCATED_PORT}{PER_CONTAINER}",
    "site.global.worker.port": "${worker.ALLOCATED_PORT}{PER_CONTAINER}",
    "site.global.tensor.board.port": "${tensorboard.ALLOCATED_PORT}{PER_CONTAINER}"
  }
}
```

resources.default.json

```
{
 "schema" : "http://example.org/specification/v2.0.0",
 "metadata" : {
 },
 "global" : {
 },
 "components": {
  "slider-appmaster": {
  },
  "ps": {
    "yarn.role.priority": "1",
    "yarn.component.instances": "2",
    "yarn.memory": "2056",
    "yarn.vcores": "1"
  },
  "worker": {
    "yarn.role.priority": "2",
    "yarn.component.instances": "5",
    "yarn.memory": "2056",
    "yarn.vcores": "1"
  },
  "tensorboard": {
    "yarn.role.priority": "3",
    "yarn.component.instances": "1",
    "yarn.memory": "1024",
    "yarn.vcores": "1"
  }
 }
}
```

Scenario 2: In-graph replication

TODO