xSpark-dagsymb

A tool exploiting symbolic execution techniques to safely run multi-dag applications in xSpark. (https://github.com/gioenn/xSpark-dagsymb.git). It combines two distinct functionalities, application profiling and application execution, which are part of symbolic executor enabled xSpark applications lifecycle, in one integrated tool called **xSpark-dagsymb**.

The tool is composed by ten principal modules: **xSpark_dagsymb.py**, **launch.py**, **run.py**, **log.py**, **plot.py**, **metrics.py**, **configure.py**, **processing.py**, **average_runs.py**, **process_on_server.py**, in addition to the configuration files **credentials.json**, **setup.json**, **control.json**.

Core Functionality

The **launch.py** module manages the startup of spot request instances on *Amazon EC2* or virtual machines on *Microsoft Azure* and waits until the instances are created and are reachable from the network via their public ip's. Subsequently the **run.py** module receives as input the instances on which to configure the cluster (*HDFS* or *Spark*), configures and runs the applications to be executed and waits for the applications to complete. The module **log.py** downloads and saves the logs created by the applications run. The **plot.py** and **metrics.py** modules respectively generate graphs and calculate metrics. The **process_on_server.py** module can be called to remotely execute the log analysis, graphs generation and metrics calculation on the xSpark master server, and download the results to the client. This option is very useful to speed-up the processing especially in case of sizeable logfiles.

Cloud Environment Configuration

The Cloud environment must be properly initialized in order to allow **xSpark_dagsymb** to access and modify resources in the cloud.

Azure

Follow the instructions to create an identity called service principal and assign to it all the required permissions:

1) Check that your account has the required permissions to create an identity.
2) Create an Azure Active Directory application
3) Get the *Application ID* and an *Authentication Key*. The *Application ID* and *Authentication Key* values replace respectively the $<$ *AZ-APP-ID* $>$ and the $<$ *AZ-SECRET* $>$ values in the credentials.json file described in the next paragraph.

Tool Configuration

The **configure.py** module contains the Config class used to instantiate configuration objects that are initialized with default values. The **credentials.json** file contains *Amazon EC2* and/or *Microsoft Azure* credential information. The **setup.json** contains Cloud environment and *Amazon EC2* and/or *Microsoft Azure* image parameters. The **control.json** file contains xSpark controller configuration parameters. Information in the **credentials.json**, **setup.json** and **control.json** files are used to customize the configuration object used by other modules during the application execution.

- AWS and/or MS-Azure Credentials: Open the *credentials_template.json* file and add the credentials for **xSpark_dagsymb** (see instructions below to retrieve missing credentials):

  { “AzTenantId”: “< AZ-TENANT-ID >”, “AzSubscriptionId”: “< AZ-SUBSCRIPTION-ID >”, “AzApplicationId”: “< AZ-APP-ID >”, “AzSecret”: “< AZ-SECRET >”, “AzPubKeyPath”: “< AZ-PUB-KEY-PATH >”, “AzPrvKeyPath”: “< AZ-PRV-KEY-PATH >”, “AwsAccessId”: “< KEY-ID >”, “AwsSecretId”: “< ACCESS-KEY >”, “KeyPairPath”: “< KEY-PAIR-PATH >” }

Save the file as *credentials.json*.

- How to retrieve your Azure credentials (using the Azure Command Line Interface):

Install the Azure CLI. Launch the following command from a console terminal:

```
$ az login
Note, we have launched a browser for you to login. For old experience with device code, use
```

a browser authentication windows is open to allow you to login to the Azure portal. If login is successful, you should get an output similar to the following:

```
You have logged in. Now let us find all the subscriptions to which you have access...
[
  {
    "cloudName": "AzureCloud",
    "id": "< AZ-SUBSCRIPTION-ID >",
    "isDefault": true,
    "name": "Microsoft Azure Sponsorship xx",
    "state": "Enabled",
    "tenantId": "< AZ-TENANT-ID >",
    "user": {
      "name": "*your_username*",
      "type": "user"
    }
  }
]
```

where you can pick the < *AZ-SUBSCRIPTION-ID* > and < *AZ-TENANT-ID* > parameters to be written in the *credentials.json* file.

Launch the following command from a console terminal to create the private and public RSA cryptography keys:

```
$ ssh-keygen -t rsa
```

Save the generated files in your favorite folder and replace the values < *AZ-PUB-KEY-PATH* > and < *AZ-PRV-KEY-PATH* > in the *credentials.json* file respectively with the fully qualified file name of the public and the private key.

- Setup the xSpark and the Virtual Machine Cloud environment: Edit the setup.json file to set the values to your need. The following is an example using Microsoft Azure VM Cloud Service:

  { “Provider”: “AZURE”, “VM”: { “Core”: 16, “Memory”: “100g” }, “ProcessOnServer”: true, “InstallPython3”: false, “Aws”: { “SecurityGroup”: “spark-cluster”, “Region”: “us-west-2”, “EbsOptimized”: true, “Price”: “0.015”, “InstanceType”: “m3.medium”, “AwsRegions”: { “eu-west-1”: {“ami”:“ami-bf61fbc8”, “az”: “eu-west-1c”, “keypair”: “simone”, “price”: “0.0035” }, “us-west-2”: {“ami”: “ami-7f5ff81f”, “snapid”: “snap-4f38bf1c”, “az”: “us-west-2c”, “keypair”: “simone2”, “price”: “0.015”} } }, “Azure”: { “ResourceGroup”: “xspark-davide-ap”, “SecurityGroup”: “cspark-securitygroup2”, “StorageAccount”: { “Sku”: “standard_lrs”, “Kind”: “storage”, “Name”: “xsparkstoragedavide1” }, “Subnet”: “default”, “NodeSize”: “Standard_D14_v2_Promo”, “Network”: “cspark-vnet2”, “Location”: “australiaeast”, “NodeImage”: { “BlobContainer”: “vhd”, “StorageAccount”: “xsparkstoragedavide1”, “Name”: “vm2-os.vhd” } }, “Spark”: { “ExternalShuffle”: “true”, “Home”: “/opt/spark/”, “LocalityWaitRack”: 0, “CpuTask”: 1, “LocalityWaitProcess”: 1, “LocalityWait”: 0, “LocalityWaitNode”: 0 }, “xSpark”: { “Home”: “/usr/local/spark/” }, “SparkSeq”: { “Home”: “/opt/spark-seq/” }

- Setup the Spark Controller parameters: Edit the control.json file to set the values to your need. The following is an example:

  {
  “Alpha”: 1.0, “Beta”: 0.3, “OverScale”: 2, “K”: 50, “Ti”: 12000, “TSample”: 500, “Heuristic”: “CONTROL_UNLIMITED”, “CoreQuantum”: 0.05, “CoreMin”: 0, “CpuPeriod”: 100000 }

Application Profiling

Profiling is the first logical phase of the performance testing lifecycle. In profiling mode, Benchmarks are run using the “vanilla” Spark version. Then the **processing.py** module is called to analyze the logs and create the “application profile”, that is a JSON file containing the annotated DAG of the executed stages plus additional information intended to be used by the controller in the execution phase. The **average__runs.py** module is called to create a JSON profile called *dagsymbmarkname>-app.json containing the average values of the "n" profiles obtained by running the same application "n" times. Finally, the file with the average profile is uploaded to the xSpark configuration directory on the master server.

Application Execution

Benchmarks are executed using xSpark, and require the application profile *dagsymbmarkname*-app.json to be present in the xSpark configuration directory. The name of the application and the parameters to modify its default configuration can either be specified as commandline arguments to the *submit* command, or can be inserted into JSON format "experiment files" and passed as commandline arguments to the *launch_exp* command. As an example, an experiment files for Pagerank , one for KMeans and one for AggregateByKey are shown here below:

PageRank experiment file example:

```
{
    "Deadline": 148080,
    "BenchmarkName": "PageRank",
    "BenchmarkConf": {
            "NumOfPartitions": 1000,
            "NumV": 35000000,
            "Mu": 3.0,
            "Sigma": 0.0,
            "MaxIterations": 1,
            "NumTrials": 1
        }
}
```

KMeans experiment file example:

```
{
    "Deadline": 116369,
    "BenchmarkName": "KMeans",
    "BenchmarkConf": {
            "NumOfPartitions": 1000,
            "NumOfPoints": 100000000,
            "NumOfClusters": 10,
            "Dimensions": 20,
            "Scaling": 0.6,
            "MaxIterations": 1
        }
 }
```

AggregateByKey experiment file example:

```
{
    "Deadline": 124000,
    "BenchmarkName": "scala-agg-by-key",
    "BenchmarkConf": {
            "ScaleFactor": 5
            }
 }
```

Download & Requirements

```
$ git clone https://github.com/gioenn/xSpark-dagsymb.git
$ cd xSpark-dagsymb
$ pip3 install -r requirements.txt"
```

xSpark-dagsymb commands

xSpark-dagsymb run command syntax:

```
$ cd xSpark-dagsymb
$ python3 xSpark_dagsymb.py *command [*args*]*
```

*command [*args*]* syntax:

```
[setup | reboot | terminate | log | profiling | time_analysis | check |
 profile | submit | launch_exp] [*args*]
```

where *args* is a set of command-specific arguments list or options.

setup command syntax:

```
setup [hdfs | spark | all | generic] {[-n | --num-instances] *numinstances*}
        {[-y |  --assume-yes]}
```

where *numinstances* is the number of nodes to add to the specified cluster (default is 5), *-y* or *–assume-yes* option sets default affirmative answer to interactive confirmation requests.

reboot command syntax:

```
reboot [hdfs | spark | all | generic]
```

reboots all nodes in the specified cluster.

terminate command syntax:

```
terminate [hdfs | spark | all | generic]
```

deletes (destroyes) all nodes and their connected resources in the specified cluster.

check command syntax:

```
check [hdfs | spark | all | generic]
```

checks the status of all nodes in the specified cluster.

profile command syntax:

```
profile [*exp_file_paths*] {[-r | --num-runs] *numruns*} {[-R | --reuse-dataset]}
         {[-q | --spark-seq]}
```

where *exp_file_paths* is a non-empty space separated list of experiment file paths, *numruns* is the number of times to repeat the profiling for each experiment file (default is 1), *-R* or *–reuse-dataset* option instructs xSpark to reuse (not to delete) application data in hdfs master node, *-q* or *–spark-seq* option instructs xSpark to use Spark data sequencing home directory.

submit command syntax:

```
submit [*exp_file_paths*] {[-r | --num-runs] *numruns*} {[-R | --reuse-dataset]}
```

where *exp_file_paths* is a non-empty space separated list of experiment file paths, *numruns* is an integer specifying the number of times to repeat the profiling for each experiment file (default is 1), *-R* or *–reuse-dataset* option instructs xSpark to reuse (not to delete) application data in hdfs master node.

_launch_exp_ command syntax:

```
launch_exp {[-e | --executors] *numexecutors*} {[-b | --application] [pagerank | kmeans |
             sort_by_key]} {[-v | --variable-parameter] *bpar*}
             {[-n | --num-instances] *numinstances*}
```

where *numexecutors* is an integer specifying the maximum number of executors to be used in the experiments, *bpar* is a variable parameter representing num_v for pagerank, num_of_points for kmeans, scale_factor for sort_by_key, *-r* or *–num-runs* is the number of times the specified application is run, *-p* or *–num-partitions* is the number of partitions for each task, *-P* or *–profile* instructs xSpark to perform the profiling at the end of the experiments, *-R* or *–reuse-dataset* option instructs xSpark to reuse (do not delete) application data in hdfs master node.

_log_profiling_ command syntax:

```
log_profiling {[-L | --local]}
```

where *-L* or *–local* option instructs xSpark use default local output folders.

_time_analysis_ command syntax:

```
time_analysis {[-i | --input-dir] *dir*}
```

where *dir* is the directory where the log files are located.

Example: Profile and Test PageRank

1) Create the credential.json file as instructed above.
2) Configure the setup.json file as instructed above.
3) Configure the control.json file as instructed above.
4) Create and initialize a hdfs cluster with 5 nodes:

   ```
   $ python3 xSpark_dagsymb.py setup hdfs -n 5
   ```

5) Create and initialize a spark cluster with 5 nodes:

   ```
   $ python3 xSpark_dagsymb.py setup spark -n 5
   ```

6) Create *experiment.json* file with the the following contents:

   ```
   { “Deadline”: 148080, “BenchmarkName”: “PageRank”, “BenchmarkConf”:
   { “NumOfPartitions”: 1000, “NumV”: 35000000, “Mu”: 3.0, “Sigma”: 0.0,
   “MaxIterations”: 1, “NumTrials”: 1
   } }
   ```

7) Run the Profiling with 6 iterations:

   ```
   $ python3 xSpark_dagsymb.py profile experiment.json -r 6 -R
   ```

8) Run the Application Test with 5 iterations:

   ```
   $ python3 xSpark_dagsymb.py submit experiment.json -r 5 -R
   ```

TODO

☐ complete this file with installation instructions for AWS
☐ clean-up code