Manual for using Jarvis system

Reference paper: “Jarvis: Large-scale Server Monitoring with Adaptive Near-Data Processing”

TABLE OF CONTENTS

Introduction

This document contains the instructions to run our system Jarvis on your local machine as well as on AWS. It provides detailed instructions on how to reproduce our measurements from experiments in the paper titled “Jarvis: Large-scale Server Monitoring with Adaptive Near-Data Processing”. For Jarvis, you need to be familiar with Apache NiFi, Apache MiNiFi and RxJava. If you want to run Jarvis on AWS, following are the technologies you would need to be familiar with: AWS EC2, CodeDeploy, RunCommand, AutoScaling Group, Elastic Load Balancing

Latest steps/instructions for all the content in this document can also be found at https://github.com/chopark/CodeDeploy_NiFi

Latest version of Jarvis code can be found at https://gitlab.engr.illinois.edu/sandur2/jarvis-local/-/tree/heuristic_noprobing/

Understand config files for Jarvis MiNiFi data source and Jarvis NiFi stream processor

Environment variable: “$HOME="/home/ubuntu”

- Jarvis MiNiFi config file location on AWS: $HOME/deploy-minifi/minifi/minifi_custom.cfg
  Jarvis NiFi config file location on AWS: $HOME/jarvis_nifi/nifi_custom.cfg
- Sample Jarvis MiNiFi config file on local machine: edge-processor/nifi-edge-processors/*.cfg
- Sample Jarvis NiFi config file on local machine: cloud-processor/nifi-cloud-processors/*.cfg

Relevant Jarvis MiNiFi config entries

- **isTestPhase: yes if on local machine and no if on AWS**
- **“inputFile: path to the monitoring input file to process on each data source node**
- **“warmUpTime”: system warm up time in seconds**
- **“warmUpInputScaleFactor”: input data rate during warm up time. E.g. 10 sends 1/10th data in each epoch**
- **“regularInputScaleFactor”: scaling factor for the input per epoch during experiment time. 1 by default**
- **“groupingReductionFraction”: if doing grouping, number of groups as a fraction of input data records**
- **“workload”: Workload to run. Can be: PingMeshQuery1, PingMeshQuery1HalfSrc, WordCountQuery1, ChainMapQuery1, AllSpQuery, JoinProfile1, ToRIpQueryStatsSummary, ToRIpQueryCountProbe, DnnLogUtilQuery, AllSpQueryDnnLog, FilterSrcDnnLogUtilQuery**
- **“filterOutAnomaly“: filter out anomalies (I.e. more data passed to next operator) or not**
- **“numSubepochs”: number of sub-epochs per epoch**
- **“joinTableSize”: in case of join operator, static table size**
- **“joinKeyStart”: starting join key ID**
- **“drainedRecordsFractionThreshold”: threshold fraction for drained records to flag congestion**
- **“idleToEpochTimeRatioThreshold”: threshold idle to epoch duration ratio for detecting idle state**
- **“idleTimeDeltaRatioThreshold”: threshold for change in idle time across epochs to detect idle state**
- **“firstEpochToStartRuntime”: first epoch when Jarvis runtime starts**
- **“detectionEpochs”: number of detection epochs before adaptation starts**
- **“maxNumAdaptations”: maximum number of adaptations allowed in an experiment run**
- **“coolDownIterations”: number of epochs to wait before triggering next adaptation**
- **“beginLoadFactor”: load factor of operators in query at beginning of adaptation. Relevant if initializeLoadFactor is set to Custom**
- **“initializeLoadFactor”: load factors to initialze for operators when adaptation begins. Can be LP (initialize with output of LP solver), 1, 0, Custom or Bootstrap (load factors just before adaptation began)**
- **“partitioningConfig”: Can be AllSP, AllSrc, HalfSrc, OP or Adapt**
- **“toDrain”: whether draining should be enabled in control proxies**

Relevant Jarvis NiFi config entries

- **isTestPhase: yes if on local machine and no if on AWS**
- **“joinTableSize”: in case of join operator, static table size**
- **“joinKeyStart”: starting join key ID**
- **“workload”: Workload to run. Can be: PingMeshQuery1, WordCountQuery1, ChainMapQuery1, JoinProfile1, ToRIpQueryProfile, DnnLogUtilQuery**

Section 6 experiments on AWS

Prerequisites

- Stream processor node is deployed in EC2 and clones the following repositories
    - Master repository called ***CodeDeploy_NiFi*** (link): this is the location from where all deployment and experiment commands are issued.
    - Jarvis stream processor code ***jarvis_nifi*** (link): contains NiFi code for Jarvis stream processor
    - Code that gets deployed on each data source node instance ***deploy_minifi*** (link): location where MiNiFi code to deploy on each data source node can be changed or updated.
    - AWS CodeDeploy repository from where data source code is deployed ***CodeDeploy_MiNiFi*** (link): pulls code from ***deploy_minifi*** repository above and packages it so that it can be deployed on each data source node
    - Scripts to manage AWS deployment ***autoscale_scripts*** (link): contains scripts for creating auto scale groups in AWS and deploying data source instances
- AWS deployment setup is done (please see Chapter "Deploying Jarvis on AWS")

Deploying data source nodes

If you made code changes to Jarvis data source and have deployed it to EC2 stream processor node (see chapter "Deploying Jarvis setup on AWS"), then

- Your new code will be in deploy_minifi folder
- Run $ bash update_minifi.sh
- Run $ bash push_it.sh <commit message>
- Copy the latest commit ID from CodeDeploy_MiNiFi folder/repository to autoscale_scripts/deploy_single_test_scaleup.sh file in autoscale_scripts folder. Link to line: https://gitlab.com/pch8286/autoscale_scripts/-/blob/master/deploy_single_test_scaleup.sh#L8

Running experiments

- Environment variable: "$HOME="/home/ubuntu"
- Log into EC2 stream processor node.
- From autoscale_scripts folder, run
    - $ bash set_max.sh (the number of groups) (the number of edges)//
        - E.g. $ bash set_max.sh 6 4
        - Above command deploys 6 groups 4 edges (total 24 edges, 4 edges / port)
    - $ bash start_test_scaleup.sh
- Verify the Jarvis config files for stream processor and data source look good.
    - Stream processor config file is located at $HOME/jarvis_nifi/nifi_custom.cfg
    - Data source config file is located at $HOME/deploy-minifi/minifi/minifi_custom.cfg
        - If you make changes just to this config file, and not any Jarvis MiNiFi code, you don't need to re-deploy the whole setup. You can copy the new config file to all data source nodes by running

          $ bash update_minifi_config_file.sh

- If you need to change network bandwidth, set the network bandwidth limit using 'network_limit.sh' in CodeDeploy_Nifi folder.
    - $ bash network_limit.sh 20000 25
    - Above commands sets network bandwidth to 20000 Kbps across 25 target groups
- From CodeDeploy_NiFi folder, run
    - $ bash run_exp.sh <interactive shell?> <experiment time> <the number of target edge group> <number of nodes per group> <number of seconds for system warm up> <CPU resource after 1 minute> <CPU resource after 3 minutes> <CPU resource after 5 minutes>
    - Wait for experiment to finish
    - Getting the metrics from the output logs:
        - Search for “Actual number of finished flowfiles by pipeline: \d+”. The number is the number of input flowfiles processed by the pipeline
        - Search for “log analysis duration in seconds: \d+”. This is the duration during which above input flowfiles were processed
        - Multiple number of flowfiles by the size of each flowfile (e.g. for PingMesh dataset size is 26.2 Mb). Divide the two to get throughput
        - Per-epoch processing latency can be found by searching “latency: (,*) and nifi proc latency is: (.*) , for wm: \d+”
    - To get the convergence time during adaptation of each MiNiFi instance on data source node
        - Go to location of MiNiFi log files that were copied in to stream processor node. The default location in the scripts is set to $HOME/temp-jarvis/minifi_logs"
        - Open one of the log files and look for the search term: “**[Runtime.run] New run. Runtime current state: (.*) and prev runtime state:(.*)**”
        - The sequence of runtime current state values in the log lines above will give the states that Jarvis runtime was in with each epoch
    - Stop the setup using 'bash stop_test_scaleup.sh' in autoscale_scripts folder

Running Jarvis code on local machine

Locating the code

- Jarvis code can be found in the compressed files (edge-processor.tar.gz and cloud-processor.tar.gz). The latest version is also hosted at https://gitlab.engr.illinois.edu/sandur2/jarvis-local/-/tree/heuristic_noprobing/
- Data source code can be found at edge-processor.tar.gz and stream processor code can be found at cloud-processor.tar.gz
- In the Jarvis-Local repository, stream processor code can be found at jarvis-local/myprocessor/cloud-processor. Data source code can be found at jarvis-local/myprocessor/edge-processor.

Running Jarvis

Tests to run Jarvis end-to-end on your local machine:

- First run data source test in edge-processor package: test name is "com.jarvis.processors.edge.MyProcessorTest.TestIntegration". You may need to change the path to writing output flowfiles in this test, depending on the location of stream processor code on your local machine
- After above test completes, run the following for stream processor which is located in cloud-processor package: test name is "com.jarvis.processors.cloud.MyProcessorTest.TestIntegrationPingMesh".

Deploying Jarvis setup on AWS

Deploying new NiFi/MiNiFi dataflow

If you change MiNiFi dataflow, follow the instructions at https://nifi.apache.org/minifi/getting-started.html to get the xml file.

- Download xml from Web UI of NiFi.
- Replace test.xml in deploy-minifi/minifi_conf/script with new xml file.
- Run buildingup_ports_id.sh with argument as ID of first port “From MiNiFi” in the template file deploy-minifi/minifi_conf/script/conf/ from-minifi0

Now you can go to step “Deploying data source nodes” in Chapter “Section 6 experiments on AWS”

Deploying new Jarvis code to AWS

If you change the Jarvis MiNiFi or NiFi code in this repository, you need to redeploy your code by running script build-nars.sh in the submitted compressed file. You can also find the script in Jarvis-Local repository (jarvis-local/myprocessor)

- When running the script, you can provide the destination folder where the compiled binaries need to be sent in EC2 stream processor. So the first argument is the path to the location on stream processor $HOME/jarvis_nifi/lib

- The second argument is the path to the location on EC2 stream processor where Jarvis data source code needs to go i.e. $HOME/deploy-minifi/minifi/minifi-0.5.0/lib

Initial Setup (when deploying AWS setup for first time)

- Install AWS CLI and setup AWS CLI with `aws configure`
- Create AWS Auto Scaling Groups by running 'create autoscale.sh' in autoscale_scripts folder
- Setup AWS IAM groups named 'CodeDeployRole' in AWS console
- Create AWS CodeDeploy Groups by running 'create_deploy_group.sh' in autoscale_scripts folder
- Create and setup repository for AWS CodeDeploy (e.g. Add GitLab token to AWS)
- Setup AWS CodeDeploy repository in deploy_setup.sh of deploy_MiNiFi folder (Directory location)
- Push AWS CodeDeploy repository manually or by push_it.sh in deploy_MiNiFi
- Get commit id and replace it in deploy_test_scaleup.sh in autoscale_scripts folder
- Install openJDK 1.8.0_222 in Cloud side machine