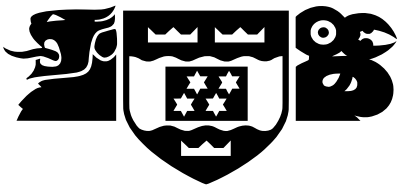
VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui



School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Experiment Instruction

Boxiong Tan

Supervisors: Hui Ma, Yi Mei, Mengjie Zhang

Submitted in partial fulfilment of the requirements for
DEGREE NOT STATED.

Abstract

This document gives intructions to the experiments of Boxiong Tan's PH.D project – *Resource Allocation in Container-based Clouds (RAC)* and its extensions. Three main parts correspond to three main objectives in the thesis, the experiments of the GA-based approaches for off-line *RAC*, GP-based Approaches for On-line *RAC*, and EC for Multi-objective *RAC*. In each experiment section, we introduces how to run the experiment codes and scripts, the algorithm implementations, and the dataset and how to generate them. With this document, one should able to repeat the experiments and to furhter develop these studies.

Contents

1 GA-based approaches for off-line *RAC* **3**
1.1 Dataset . . . 3
1.2 Algorithm Code and Scripts . . . 3
1.2.1 Algorithm Code . . . 3
1.2.2 Data Collection . . . 4

2 GP-based Approaches for On-line *RAC* **5**
2.1 Dataset . . . 5
2.2 Algorithm Code and Scripts . . . 6
2.2.1 Algorithm Code . . . 6
2.2.2 Scripts . . . 6

3 EC for Multi-objective *RAC* **11**
3.1 Dataset . . . 11
3.2 Algorithm Code and Scripts . . . 12
3.2.1 Algorithm Code . . . 12
3.2.2 Scripts . . . 12

Directory Tree

```
~/PH.D_project/
-- data/                                        # data sets
-- algorithm_jar/                               # algorithm jar files
-- containerAllocationConfigurationFiles/       # configuration files for training and testing
-- dataGenerationScripts/                       # scripts for generating test instances
-- gridScripts/                                 # scripts for submiting experiments on the Grid
-- experimentScripts/                           # scripts for running testing, results collection,
and ploting
```

Chapter 1

GA-based approaches for off-line *RAC*

1.1 Dataset

```
-- PH.D_project/data/baseConfig            # PM and VM configurations
-- PH.D_project/data/containerData/static/single-objective/  # test instances

PMConfig_small.csv

13200             # CPU capacity
16000             # Memory capacity
540               # Max energy consumption

VMConfig_ten.csv

#CPU, #Memory
719,2005
917,951
1032,1009
1135,3542
1231,1989
1311,3238
1363,2634
1648,1538
2047,1181
2100,3013

Container1000_VM_twenty.csv

Each row represents a container.

#CPU, #Memory
35.11,323.58
140.4,248.49
353.6,1736.7
41.6,214.35
156.05,379.56
...
```

1.2 Algorithm Code and Scripts

1.2.1 Algorithm Code

```
Directories
-- GA/src/cecGA                                                    # single-chromosome GA
-- GA/src/GroupGA                                                  # group GA
-- GA/src/PermutationBasedGAForContainerAllocation/                # dual-chromosome GA
```

1.2.2 Data Collection

```
-- PH.D_project/experimentScripts/singleObjectiveStaticAllocation/collect.sh
# This is the main collection script. It will execute all the sub-scripts
-- ...sh

collect.sh

Usage:

./collect.sh [alg = groupGA | dualPermutation | cecGA] [vmTypes = ten | twenty]
[appNum = 200 | 500 | 1000 | 1500]
```

Chapter 2

GP-based Approaches for On-line *RAC*

2.1 Dataset

```
-- PH.D_project/data/baseConfig            # PM and VM configurations
-- PH.D_project/data/containerData/dynamic/  # training and test instances
-- PH.D_project/data/InitEnv/              # Initial data center
-- PH.D_project/data/OSData/dynamic/       # The OS requirements for the containers in the test inst
-- PH.D_project/data/OSPro/                    # OS probabilities

InitEnv
        -- container.csv   # The initial containers
        -- os.csv                  # The OS of VMs
        -- pm.csv                  # The initial PMs
        -- vm.csv          # The initial VMs

container.csv

#CPU, #Mem
26,98.3
175.56,425.98
27.73,252.58
6.93,125.61

os.csv

# OS type of each VM
1
2
2
...

pm.csv

# Each row represents a PM. The numbers in each row represent the type of VM in this PM

10,10
6,1
9,2,4,9
2
2
```

```
...

vm.csv

# Each row represents a VM. The numbers in each row represent the indexes of containers.

1,2,3,4
5,6
7,8,9,10
11
12,13,14,15
...

-- PH.D_project/data/OSPro/OS5/probability.csv

# the probability of each type of OS
0.179
0.454
0.236
0.105
0.026
```

2.2 Algorithm Code and Scripts

2.2.1 Algorithm Code

```
Directories
-- ContainerAllocationCCGP # For CCGP training
-- ContainerAllocationGPHH # For GPHH training
-- containerAllocation     # For simulation, or testing
```

2.2.2 Scripts

For Test Instance Generation

```
Directories:
-- PHD_project/dataGenerationScripts/generateTaskCases.R
# For generating individual test instances

-- PHD_project/data/dataGenerationScripts/generateOSCases.R
# For generating the OS requirements for these test instances

-- PHD_project/data/dataGenerationScripts/generateBaseDataCenter.R
# For generating the initial data center
```

generateTaskCases.R

```
Usages:

generateTask(
        whichDataSet=[1:auvergrid | 2:bitbrains],
        whichVMsize=[ 6:ten types | 7:twenty types ],
        size = 2500,
```

```
        testCase, filename='')

Example:

for (i in seq(0,199))
        generateTask(1, 6, 2500, i, paste('./container2500/testCase', i, '.csv', sep=''))")
```

generateOSCases.R

```
generateOScases(
        size=2500,
        numOfOS=[3 | 4 | 5],
        testCase,
        filename='')

Example:

for(i in seq(0, 199))
        generateOScases(2500, 5, i, paste('./OS1Container1000/testCase', i, '.csv', sep=''))")
```

generateBaseDataCenter.R

```
generateBaseDataCenter(
        testCaseSize=2500,
        OSNum=[3 | 4 | 5],
        testCase)

Example:

for(i in seq(0, 199))
        generateOScases(2500, 5, i)
```

For Run Experiments

The grid scripts for GPHH and CCGP training,

```
Directories:
-- PHD_project/gridScripts/CCGP/ccgp.sh   # The configuration scripts
-- PHD_project/gridScripts/CCGP/run.sh    # submit 40 runs

-- PHD_project/gridScripts/GPHH/gphh.sh
-- PHD_project/gridScripts/GPHH/run.sh

Usage:

./run.sh [vmtypes = TEN | TWENTY] [OSNUM = 5] [dataset = BITBRAINS | AUVERGRID]

Example:

./run.sh TEN 5 BITBRAINS
```

The scripts for data collection:

```
-- dynamicContainerAllocation/scriptCCGP/execute.sh
--      ...
-- dynamicContainerAllocation/scriptGPHH/execute.sh
--  ...

Usage:

./collect.sh [alg = CCGP] [osNum=5] [vmNum=TWENTY | TEN] [dataset=AUVERGRID | BITBRAINS]

Example:

./collect.sh CCGP 5 TWENTY AUVERGRID
```

The scripts for testing:

Before test, the simulation must be correctly configured in the program.

The example shows how to test a GPHH rule on the Auvergrid dataset with 10 VM types.

```
In containerAllocation/src/Experiment_selection_creation_combined.java:

        experimentRunner(
                run,
                folder,
                TestDataSet.AUVERGRID_TEN,
                TestCaseSizes.xLarge,
                OperatingSystems.THREE,
                FitnessFunctions.EVO, // vm selection rule
                FitnessFunctions.SUB, // pm selection rule
                VMCreationRules.EVO,
                VMAllocationFramework.ANYFIT,   // vm selection-creation framework, when
                                                // select NONANYFIT framework,
                                                // the VMCreationRules will be
                                                // automatically canceled.
                SelectionRules.FIRSTFIT,        // pm selection framework,
                                                // If we choose FirstFit, then, the fitness functions
                                                // of pm selection rule will be ignored
                PMCreationRules.LARGEST);       // No other choice
    }
```

After compiling the code, you may use the testing script to test. Then, the testing script:

```
--PH.D_project/experimentScripts/dynamicAllocation/experimentRunningScripts/runSingleExperiment.sh
--PH.D_project/experimentScripts/dynamicAllocation/experimentRunningScripts/test30Runs.sh

Usage:

./test30Runs.sh [Path]

Example:

./test30Runs.sh ./Container2500_AUVERGRID_TEN
# where the collected rules are in the Container2500_AUVERGRID_TEN
```

For Data Analysis

```
--PH.D_project/experimentScripts/dynamicAllocation/statScripts/
        -- barPlotEvoCompare.R  # plot bar chart to compare GPHH and CCGP
        -- barPlotPm.R          # plot bar chart of PM number
        -- barplotVmDist.R      # plot bar chart of VM distribution
        -- calVmNum.R           # calculate the number of VMs used
        -- energyAveSd.R        # calculate the mean and sd of a scenario
        -- meanPmRemain.R               # calculate the mean PM resource remaining
        -- meanPmUtil.R                 # calculate the mean PM resource utilization
        -- plotWaste.R          # plot line chart of waste resources
        -- stat_of_30_general.R # statistic significant test
        -- vmDist20.R           # calculate the frequency of 20 VM types
        -- vmDist.R             # calculate the frequency of 10 VM types
```

The usage of each script is record in the scripts and will be printed out when first load them.

Chapter 3

EC for Multi-objective *RAC*

3.1 Dataset

```
Dataset Directories:
-- data/containerData/static/multi_objective
-- data/containerData/static/spaceEstimation
```

The application files describes the information of applications including the number of applications, each application is consist of how many micro-services, and the number of replicas for each micro-services. The application files are named as *application* + *number of application* + .csv.

```
application file

application50.csv
application100.csv
application150.csv
application200.csv
```

The data structure of application files are :

```
micro-service 1, micro-service 2, ..., micro-service 5
2,4,0,0,0
3,0,0,0,0
3,4,4,2,5
3,4,2,3,2
```

Each row represents an application. Each column represents a micro-service. In the current setting, each application has maximum of 5 micro-services. The number of each index denotes the number of replicas of that micro-service. For example, in the first row of the above dataset, [2, 4, 0, 0, 0] denotes application 1 is consist of 2 micro-services. Micro-service 1 has 2 replicas and micro-service 2 has 4 replicas.

The constraint for the number of replicas is defined as following. The lower bound is 2. The upper bound is 5. 0 means the micro-service does not exist.

TestCase Files

The testCase files contain the detailed resource requirement of replicas. The testCases and application files are 1-on-1 mapping. For example, *testCase50.csv* and *application50.csv* are describing the same instance. The application files are named as *application* + *number of application* + .csv. meaning application50.csv

```
# testCase file

testCase50.csv
testCase100.csv
testCase150.csv
testCase200.csv


# testCase data structure

CPU, MEM, applicationId, micro-serviceId, replicaId
53.73,244.39,1,1,1
53.73,244.39,1,1,2
255.54,410.96,1,2,1
255.54,410.96,1,2,2
```

Notice that, in experiments of variable size of replicas, the number of replicas is determined by algorithm. Although we are using the same dataset, the number of replica is not used.

3.2 Algorithm Code and Scripts

3.2.1 Algorithm Code

```
-- GA/src/LNS_FF                # Large-neighbourhood search and FF
-- GA/src/LNS_NS-GGA            # LNS and NS-GGA
-- GA/src/NSGAII_NS-GGA         # NSGAII and NS-GGA
-- GA/src/NSGAII_Spread         # NSGAII and Spread
-- LNBPSO/src/LNS_BPSO          # LNS and BPSO
```

3.2.2 Scripts

Grid

```
-- PH.D_project/gridScripts/multiObj_VariableReplicas/run_ga.sh
-- PH.D_project/gridScripts/multiObj_VariableReplicas/ga.sh
-- PH.D_project/gridScripts/multiObj_VariableReplicas/run_bpso.sh
-- PH.D_project/gridScripts/multiObj_VariableReplicas/bpso.sh


run_ga.sh

Usage:

./run_ga.sh [algorithm = NSGAII_NSGGA | LNS_FF | NSGAII_Spread | LNS_NSGGA ]
[appSize = app100 | app150 | app200] [vmTypes = ten | twenty]


run_bpso.sh

Usage:

./run.sh [algorithm = NSGAII_NSGGA | LNS_FF | NSGAII_Spread | LNS_NSGGA ]
[appSize = app100 | app150 | app200] [vmTypes = ten | twenty]
```

For Data Collection

```
-- PH.D_project/experimentScripts/multiObjectiveStaticAllocation/collect.sh
# This is the main collection script. It will execute all the sub-scripts
-- ...sh

Usage

./collect.sh [alg = multiGroupGA] [vmTypes = ten | twenty] [appNum = 150]
Example:

./collect.sh multiGroupGA ten 150
```

For Data Analysis