APPENDIX 1

SAMPLE CODING

A1.1 CLOUD OPTIMIZATION

```java
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Calendar;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.core.SimEntity;
import org.cloudbus.cloudsim.core.SimEvent;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;
public class HybridCloud {

    /** The cloudlet list. */
    private static List<Cloudlet> Task = new LinkedList<Cloudlet>();
    private static List<Cloudlet> cloudletList = new LinkedList<Cloudlet>();
    private static List<Cloudlet> cloudletList1 = new LinkedList<Cloudlet>();
    private static List<Cloudlet> cloudletList2 = new LinkedList<Cloudlet>();
    public static List<Cloudlet> cl = new LinkedList<Cloudlet>();
    public static int SmId, SmId1, SmId2, prid;
    public static Random r = new Random();
    public static Vector grp, g;
    DatacenterBroker broker, broker1, broker2;
    private static List<Vm> vmList;

    /** The vmList. */
    public static List<Vm> vmList = new ArrayList<Vm>(), vmList1, vmList2, prList;

    private static List<Vm> createVM(int userId, int vms) {
        LinkedList<Vm> list = new LinkedList<Vm>();
        int ra[] = {512, 256};
        int mi[] = {250, 200, 150};
        int b[] = {1000, 900, 800};
        Random r = new Random();
        long size = 10000; // image size (MB)
```
String vmm = "Xen";
//create VMs
Vm[] vm = new Vm[vms];

for(int i=0;i<vms;i++){
    int ram = ra[r.nextInt(ra.length)]; //vm memory (MB)
    int mips =mi[r.nextInt(mi.length)];
    long bw = b[r.nextInt(b.length)];
    int pesNumber = 1; //number of cpus
    vm[i] = new Vm(i, userId, mips, pesNumber, ram, bw, size,
    vmm, new CloudletSchedulerTimeShared());
    list.add(vm[i]);
}
return list;

private static List<Cloudlet> createCloudlet(int userId, int cloudlets){
    // Creates a container to store Cloudlets
    LinkedList<Cloudlet> list = new LinkedList<Cloudlet>();

    //cloudlet parameters
    long length = 4000;
    long fileSize = 300;
    long outputSize = 300;
    int pesNumber = 1;
    UtilizationModel utilizationModel = new UtilizationModelFull();
    Cloudlet[] cloudlet = new Cloudlet[cloudlets];
    for(int i=0;i<cloudlets;i++){
        cloudlet[i] = new Cloudlet(i, length, pesNumber, fileSize,
        outputSize, utilizationModel, utilizationModel, utilizationModel);
        // setting the owner of these Cloudlets
        cloudlet[i].setUserId(userId);
        list.add(cloudlet[i]);
    }
    return list;
}

HybridCloud() {
    Log.println("Starting HybridCloud...");
    try {
        textarea.txtaMessage.append("nDataCenter\t Architecture\tOS\tVM");
        textarea.txtaMessage.append("nHL\tX86\tWindows\tOracle");
    }
}
textarea.txtaMessage.append("lnPltx86\uLinux\txtXen");
textarea.txtaMessage.append("lnPU\tx64\tSolaris\tsunX");
    int num_user = 3; // number of grid users
    Calendar calendar = Calendar.getInstance();
    boolean trace_flag = false; // mean trace events
    CloudSim.init(num_user, calendar, trace_flag);

    // Second step: Create Datacenters
    // Datacenters are the resource providers in CloudSim. We need at
    // list one of them to run a CloudSim simulation
    Datacenter HL = createDatacenter("HL");
    Datacenter PI = PrivateDatacenter("PI");
    Datacenter PU = PublicDatacenter("PU");
    int d1id=HL.getId();
    System.out.println(HL.getId());
    broker = createBroker("Hybrid");
    Log.printLine("HybridCloud finished!");
}
}

textarea.txtaMessage.append("\n\nA Path Clustering Heuristic for Scheduling\nVmId	Computation cost\nfor(int i=0;i<vl1.size();i++)
{
    Vm v=vl1.get(i);
    textarea.txtaMessage.append("\n"+v.getId() + "\"\t"+v.getMips()/v.getPesNumber() + "\n");
}
textarea.txtaMessage.append("\nVmId\tComunication cost\nfor (int i = 0; i < cl.size(); i++)
{
    Cloudlet cloudlet =cl.get(i);
    long bw=1000;
    for(int j=0;j<vl1.size();j++)
    {
        Vm v=vl1.get(j);
        if(v.getId()==cloudlet.getVmId())
        {
            bw=v.getBw() ;
        }
    }
textarea.txt.append("\n"+cloudlet.getVmId()+"\t"+cloudlet.getCloudletLength()/bw )
}
textarea.txt.append("\n\nConstructing Group");
textarea.txt.append("\n\nGroup\tTask");
grp=new Vector();
g=new Vector();
DecimalFormat dft = new DecimalFormat("###.##");
for(int j=0;j<vl1.size();j++)
{
    textarea.txt.append("\n"+j);
    Vm v=vl1.get(j);
    double mkspan=0;
    for (int i = 0; i < cl.size(); i++)
    {
        Cloudlet cloudlet =cl.get(i);
        if(v.getId()==cloudlet.getVmId())
        {
            cost=Double.parseDouble(dft.format(cloudlet.getFinishTime()))-
            Double.parseDouble(dft.format(cloudlet.getExecStartTime()));
            //cost+=r.nextInt(100);
            mkspan+=cost;
        }
    }
    g.addElement(j);
}
textarea.txt.append("\nGroup\tMakespan");
for(int i=0;i<grp.size();i++)
{
    textarea.txt.append("\n"+i+
    blk.append(grp.elementAt(i).toString());
}

textarea.txt.append("\nRescheduling to Public Cloud................");
System.out.println(grp.elementAt(0));
double th=(Double.parseDouble(grp.elementAt(0).toString()))/1.5;
textarea.txt.append("\n\n\n\nDeadLine   "+th);
for (int i=0;i<grp.size();i++)
{
    }
double mk = Double.parseDouble(grp.elementAt(i).toString());
if (mk > th) {
    new RamProvisionerSimple(ram),
    new BwProvisionerSimple(bw),
    storage,
    peList1,
    new VmSchedulerTimeShared(peList1)
}
);
// This is our first machine

hostId++;
hostList.add(
    new Host(  
        hostId,  
        new RamProvisionerSimple(ram),  
        new BwProvisionerSimple(bw),  
        storage,
        peList2,
        new VmSchedulerTimeShared(peList2)
    )
);

String arch = "x86";  // system architecture
String os = "Linux";  // operating system
String vmm = "Xen";
double time_zone = 10.0;  // time zone this resource located
double cost = 2.0;  // the cost of using processing in this resource
double costPerMem = 0.03;  // the cost of using memory in this resource
double costPerStorage = 0.05;  // the cost of using storage in this resource
double costPerBw = 0.05;  // the cost of using bw in this resource
LinkedList<Storage> storageList = new LinkedList<Storage>();
DatacenterCharacteristics characteristics = new DatacenterCharacteristics(arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage, costPerBw);

// 6. Finally, we need to create a PowerDatacenter object.
Datacenter datacenter = null;
try {

datacenter = new Datacenter(name, characteristics, new 
VmAllocationPolicySimple(hostList), storageList, 0);
} catch (Exception e) {
    e.printStackTrace();
}

return datacenter;

private static Datacenter PublicDatacenter(String name){
    List<Host> hostList = new ArrayList<Host>();
    List<Pe> peList1 = new ArrayList<Pe>();
    int mips = 1000;
    peList1.add(new Pe(0, new PeProvisionerSimple(mips)));
    peList1.add(new Pe(1, new PeProvisionerSimple(mips)));
    peList1.add(new Pe(2, new PeProvisionerSimple(mips)));
    peList1.add(new Pe(3, new PeProvisionerSimple(mips)));
    List<Pe> peList2 = new ArrayList<Pe>();
    peList2.add(new Pe(0, new PeProvisionerSimple(mips)));
    peList2.add(new Pe(1, new PeProvisionerSimple(mips)));
    int hostId=0;
    int ram = 2048;
    long storage = 1000000;
    int bw = 10000;

    hostList.add(
        new Host(
            hostId,
            new RamProvisionerSimple(ram),
            new BwProvisionerSimple(bw),
            storage,
            peList1,
            new VmSchedulerTimeShared(peList1)
        )
    );

    hostId++;
    hostList.add(
        new Host(
            hostId,
            new RamProvisionerSimple(ram),
            new BwProvisionerSimple(bw),
            storage,
            peList2,
            new VmSchedulerTimeShared(peList2)
        )
    );
}
new BwProvisionerSimple(bw),
storage,
peList2,
new VmSchedulerTimeShared(peList2)
);

String arch = "x86"; // system architecture
String os = "Solaris"; // operating system
String vmm = "sunX";
double time_zone = 10.0; // time zone this resource located
double cost = 1.0; // the cost of using processing in this resource
double costPerMem = 0.02; // the cost of using memory in this resource
double costPerStorage = 0.03; // the cost of using storage in this resource
double costPerBw = 0.03; // the cost of using bw in this resource
LinkedList<Storage> storageList = new LinkedList<Storage>();

DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
textarea.txtaMessage.append("\nCloudlet	Cost	ThroughPut	Utility	Scalability")
textarea.txtaMessage.append("\n"+Main.ta+"\t"+total+"\t"+tput+"\t"+utility+"\t"+scal)
String str="select * from cost where x="+Integer.toString(Main.ta)
ResultSet rs4=db.stat3.executeQuery(str)
if(rs4.next())
{
    str="updatecostsetx="+Main.ta+",y="+total+" where x="+Main.ta;
    db.stat2.executeUpdate(str);
}
else
{
    db.stat2.executeUpdate("insert into cost values(\"+Main.ta+\",\"+total+\")
}

str="select * from utility where x="+Integer.toString(Main.ta)
    rs4=db.stat3.executeQuery(str)
if(rs4.next())
{
    str="update  utility set x="+Main.ta+",y="+utility+" where x="+Main.ta;
    db.stat2.executeUpdate(str);
}
else
{
db.stat2.executeUpdate("insert into utility values("+Main.ta+","utility+")");
}
str="select * from scal where x="+Integer.toString(Main.ta);
rs4=db.stat3.executeQuery(str);
if(rs4.next())
{
str="update scal set x="+Main.ta+",y="+scal+" where x="+Main.ta;
    db.stat2.executeUpdate(str);
}
else
{
    db.stat2.executeUpdate("insert into scal values("+Main.ta+","+scal+")");
}
str="select * from tput where x="+Integer.toString(Main.ta);
    rs4=db.stat3.executeQuery(str);
    if(rs4.next())
    {
        str="update tput set x="+Main.ta+",y="+tput+" where x="+Main.ta;
            db.stat2.executeUpdate(str);
    }
else
{
    db.stat2.executeUpdate("insert into tput values("+Main.ta+","+tput+")");
}
}
}

}catch(Exception e)
{
    e.printStackTrace();
}

A1.2 ENERGY AWARE SCHEDULING

#define HAVE_SSTREAM
#include <stdexcept>  // runtime_error
#include <iostream>   // cout
#include <sstream>  // ostrstream, istrstream
```cpp
#include <fstream>
#include <string.h>
#include <eo>
#include <ga.h>
#include <eo/eoBitOp.h>
using namespace std;
#include <eoInt.h>
#include <eoInit.h>
#include <eoScalarFitness.h>
#include <eval/queenEval.h>
#include <eval/moFullEvalByModif.h>
#include <eval/moFullEvalByCopy.h>
//Neighbors and Neighborhoods
#include <problems/permutation/moShiftNeighbor.h>
#include <neighborhood/moOrderNeighborhood.h>

//Mutation
#include <eoSwapMutation.h>
#include <eoOrderXover.h>
//Algorithm and its components
#include <algo/moFirstImprHC.h>
//mo eval
#include <eval/moFullEvalByCopy.h>
#include <continuator/moTrueContinuator.h>
typedef eoInt<eoMinimizingFitness> Queen; //Permutation (Queen's problem representation)
typedef moShiftNeighbor<Queen> shiftNeighbor; //shift Neighbor
typedef moOrderNeighborhood<shiftNeighbor> orderShiftNeighborhood; //order shift Neighborhood (Indexed)

void main_function(int argc, char **argv) {
    eoParser parser(argc, argv);
    eoValueParam<uint32_t> seedParam(time(0), "seed", "Random number seed", 'S');
    parser.processParam( seedParam );
    unsigned seed = seedParam.value();
    eoValueParam<unsigned int> vecSizeParam(8, "vecSize", "Genotype size", 'V');
    parser.processParam( vecSizeParam, "Representation" );
    unsigned vecSize = vecSizeParam.value();
    string str_status = parser.ProgramName() + ".status"; // default value
    eoValueParam<string> statusParam(str_status.c_str(), "status", "Status file");
    parser.processParam( statusParam, "Persistence" );
    if (parser.userNeedsHelp()) {
```
parser.printHelp(cout);
exit(1);
}
if (statusParam.value() != ")
    ofstream os(statusParam.value().c_str());
    os << parser; // and you can use that file as parameter file
}
rng.reseed(seed);
queenEval<Queen> fullEval;
eoInitPermutation<Queen> init(vecSize);
eoPop<Queen> pop;
Queen tmp;
for (unsigned int i=0; i<20; i++) {
    init(tmp);
    fullEval(tmp);
    pop.push_back(tmp);
}
moFullEvalByCopy<shiftNeighbor> shiftEval(fullEval);
orderShiftNeighborhood orderShiftNH((vecSize-1) * (vecSize-1));

moFirstImprHC<shiftNeighbor> hc(orderShiftNH, fullEval, shiftEval);
eoGenContinue<Queen> EAcont(50);

eoDetTournamentSelect<Queen> selectOne(2);
eoSelectMany<Queen> select(selectOne, 1);
eoOrderXover<Queen> cross;

eoSGATransform<Queen> transform(cross, 0.3, hc, 0.7);
eoGenerationalReplacement<Queen> repl;
eoEasyEA<Queen> hybridAlgo(EAcont, fullEval, select, transform, repl);

std::cout << "INITIAL POPULATION:" << std::endl;
std::cout << "-------------------" << std::endl;

for (unsigned int i=0; i<pop.size(); i++)
    std::cout << pop[i] << std::endl;
hybridAlgo(pop);
std::cout << std::endl;
std::cout << "FINAL POPULATION:" << std::endl;
std::cout << "-------------------" << std::endl;
for (unsigned int i=0; i<pop.size(); i++)
    std::cout << pop[i] << std::endl;
}

int main(int argc, char **argv)
{
    try {
        main_function(argc, argv);
    }
    catch (exception& e) {
        cout << "Exception: " << e.what() << '\n';
    }
    return 1;
}

A1.3 FUZZY OPTIMIZATION (FCL)

// Block definition (there may be more than one block per file)
FUNCTION_BLOCK tipper

// Define input variables
VAR_INPUT
    Application : REAL;
    Application_Type : REAL;
    Catagory : REAL;
    Service : REAL;
END_VAR

// Define output variable
VAR_OUTPUT
    ALGORITHM : REAL;
END_VAR

FUZZIFY Application
    TERM scientific := (0, 1) (1,1) ;
TERM General := (1,0) (5,1);
END_FUZZIFY

FUZZIFY Application_Type
TERM Time:= (0, 1) (1, 1) ;
TERM Utilization := (1,0) (2,1);
TERM Cost:= (2,0) (3,1) ;
TERM Scalablity := (3,0) (4,1);
TERM Energy_Saving := (4,0) (5,1) ;
END_FUZZIFY

FUZZIFY Catagory
TERM Response_Time:= (0,0) (5,1) (10,0) ;
TERM Execution_Time := (10,0) (15,1) (20,0);
TERM Memory:= (20,0) (25,1) (30,0) ;
TERM Bandwidth := (30,0) (35,1) (40,0);
TERM Processor := (40,0) (45,1) (50,0) ;
TERM Profit := (50,0) (55,1) (60,0);
TERM Penalty := (60,0) (65,1) (70,0) ;
END_FUZZIFY

// Fuzzify input variable 'Service'
FUZZIFY Service

TERM HIGH := (0, 1) (4, 0) ;
TERM LOW := (1, 0) (4,1) (6,1) (9,0);
TERM MEDIUM := (6, 0) (9, 1);
END_FUZZIFY

// Defuzzify output variable 'ALGORITHM'
DEFUZZIFY ALGORITHM

TERM Data_Aware_Scheduling := (0,0) (5,1) (10,0);
TERM Load_Balancing_IPSO := (10,0) (15,1) (20,0);
TERM Fuzzy_Nural_Network := (20,0) (25,1) (30,0);
TERM Parallal_Genetic_Algorithm := (30,0) (35,1) (40,0);
TERM MTC_PCH := (40,0) (45,1) (50,0);
TERM Novel_Utility_Accural := (50,0) (55,1) (60,0);
TERM Energy_Aware := (60,0) (65,1) (70,0);

// Use 'Center Of Gravity' defuzzification method
// METHOD : COG;
// Default value is 0 (if no rule activates defuzzifier)
// DEFAULT := 0;
END_DEFUZZIFY

RULEBLOCK No1
  // Use 'min' for 'and' (also implicit use 'max'
  // for 'or' to fulfill DeMorgan's Law)
  AND : MIN;
  // Use 'min' activation method
  ACT : MIN;
  // Use 'max' accumulation method
  ACCU : MAX;

RULE 1 : IF Application IS scientific
  THEN ALGORITHM is Data_Aware_Scheduling;

RULE 2 : IF Application IS General AND Application_Type IS Time AND Catagory IS Response_Time AND Service IS HIGH
  THEN ALGORITHM is Fuzzy_Nural_Network;

RULE 3 : IF Application IS General AND Application_Type IS Time AND Catagory IS Response_Time AND Service IS MEDIUM
  THEN ALGORITHM is Fuzzy_Nural_Network;

RULE 4 : IF Application IS General AND Application_Type IS Time AND Catagory IS Response_Time AND Service IS LOW
  THEN ALGORITHM is Load_Balancing_IPSO;

RULE 5 : IF Application IS General AND Application_Type IS Time AND Catagory IS Execution_Time AND Service IS HIGH
  THEN ALGORITHM is Energy_Aware;

RULE 6 : IF Application IS General AND Application_Type IS Time AND Catagory IS Execution_Time AND Service IS MEDIUM
  THEN ALGORITHM is MTC_PCH;
RULE 7 : IF Application IS General AND Application_Type IS Time AND Catagory IS Execution_Time AND Service IS LOW
   THEN ALGORITHM is Parallal_Genetic_Algorithm;

RULE 8 : IF Application IS General AND Application_Type IS Utilization AND Catagory IS Memory AND Service IS HIGH
   THEN ALGORITHM is Fuzzy_Nural_Network;

RULE 9 : IF Application IS General AND Application_Type IS Utilization AND Catagory IS Memory AND Service IS MEDIUM
   THEN ALGORITHM is Novel_Utility_Accural;

RULE 10 : IF Application IS General AND Application_Type IS Utilization AND Catagory IS Memory AND Service IS LOW
   THEN ALGORITHM is Load_Balancing_IPSO;

RULE 11 : IF Application IS General AND Application_Type IS Utilization AND Catagory IS Bandwidth AND Service IS HIGH
   THEN ALGORITHM is Energy_Aware;

RULE 12 : IF Application IS General AND Application_Type IS Utilization AND Catagory IS Bandwidth AND Service IS MEDIUM
   THEN ALGORITHM is Fuzzy_Nural_Network;

RULE 13 : IF Application IS General AND Application_Type IS Utilization AND Catagory IS Bandwidth AND Service IS LOW
   THEN ALGORITHM is Load_Balancing_IPSO;

RULE 14 : IF Application IS General AND Application_Type IS Utilization AND Catagory IS Processor AND Service IS HIGH
   THEN ALGORITHM is Parallal_Genetic_Algorithm;

RULE 15 : IF Application IS General AND Application_Type IS Utilization AND Catagory IS Processor AND Service IS MEDIUM
   THEN ALGORITHM is Energy_Aware;

RULE 16 : IF Application IS General AND Application_Type IS Utilization AND Catagory IS Processor AND Service IS LOW
   THEN ALGORITHM is MTC_PCH;
RULE 17: IF Application IS General AND Application_Type IS Cost AND Catagory IS Profit
AND Service IS HIGH
    THEN ALGORITHM is MTC_PCH;

RULE 18: IF Application IS General AND Application_Type IS Cost AND Catagory IS Profit
AND Service IS MEDIUM
    THEN ALGORITHM is Novel_Utility_Accural;

RULE 19: IF Application IS General AND Application_Type IS Cost AND Catagory IS Profit
AND Service IS LOW
    THEN ALGORITHM is Energy_Aware;

RULE 20: IF Application IS General AND Application_Type IS Cost AND Catagory IS Penalty
AND Service IS HIGH
    THEN ALGORITHM is Novel_Utility_Accural;

RULE 21: IF Application IS General AND Application_Type IS Cost AND Catagory IS Penalty
AND Service IS MEDIUM
    THEN ALGORITHM is MTC_PCH;

RULE 22: IF Application IS General AND Application_Type IS Cost AND Catagory IS Penalty
AND Service IS LOW
    THEN ALGORITHM is Parallal_Genetic_Algorithm;

RULE 23: IF Application IS General AND Application_Type IS Scalablity AND Service IS
HIGH
    THEN ALGORITHM is MTC_PCH;

RULE 24: IF Application IS General AND Application_Type IS Scalablity AND Service IS
MEDIUM
    THEN ALGORITHM is Fuzzy_Nural_Network;

RULE 25: IF Application IS General AND Application_Type IS Scalablity AND Service IS
LOW
    THEN ALGORITHM is Novel_Utility_Accural;

RULE 26: IF Application IS General AND Application_Type IS Energy_Saving AND Service IS
HIGH
    THEN ALGORITHM is Energy_Aware;
RULE 27: IF Application IS General AND Application_Type IS Energy_Saving AND Service IS MEDIUM
    THEN ALGORITHM is Fuzzy_Nural_Network;

RULE 28: IF Application IS General AND Application_Type IS Energy_Saving AND Service IS LOW
    THEN ALGORITHM is Parallal_Genetic_Algorithm;
END_RULEBLOCK

END_FUNCTION_BLOCK

A1.4 FUZZY OPTIMIZATION (JAVA)

import net.sourceforge.jFuzzyLogic.FIS;
public class FuzzyCloudSystem {
    public static void main(String[] args) throws Exception {
        // Load from 'FCL' file
        String fileName = "cloud.fcl";
        FIS fis = FIS.load(fileName,true);
        // Error while loading?
        if( fis == null ) {
            System.err.println("Can't load file: '
                                + fileName + '");
            return;
        }

        // Show
        fis.chart();
        // Set inputs
        fis.setVariable("Service", 3);
        fis.setVariable("Application", 2);
        fis.setVariable("Application_Type", 5);
        fis.setVariable("Catagory", 7);
        // Evaluate
        fis.evaluate();
        // Show output variable's chart
        fis.getVariable("ALGORITHM").chartDefuzzifier(true);
        // Print ruleSet
        System.out.println(fis);
    }
}