```
File: project-fermi/CommandList.txt
--------------------------------COMMANDS LIST----------------------------------------------

       help | -h | --help

       ## list the commands list

-----------------------------------------------------------------------------------------------

       plugman list [-options]

              -b | --type=batch

              -c | --type=cloud

       plugman load [-options] [NAME]

              -b | --type=batch NAME

              -c | --type=cloud NAME

       plugman unload [NAME]

       plugman info [NAME]

-----------------------------------------------------------------------------------------------

       cloudman register [CONF_FILE]

       cloudman dump // List all registerd clouds

        cloudman listhost //List all hosts

       cloudman

-----------------------------------------------------------------------------------------------

       vmman list [options] [NAME]

              -refresh [NAME] //Get the realtime informations from specified cloud and

                            //refresh the vms collection

                            //if cloud name isn't specified,then refresh all the clouds

              -n       [NAME] //vmman list -n Gcloud

              -a              //list available vms

              -r              //running
```

```
              -s              //suspended

       vmman create [-options] [peremeter]

              -h | --help

              -n [VM_NUMS]

              -c [CLOUD]

              ## vmman create -c Gcloud -n 2

              ## default cloud is the last loaded cloud,default nums is 1.

       vmman suspend [id]

       vmman start [id]

       vmman destroy [id]

       vmman show [id]

-----------------------------------------------------------------------------------------------

       check_q | chkq

       check_p | chkp

-----------------------------------------------------------------------------------------------



-----------
-----------
-----------

File: project-fermi/pluginReadme

vcluter > plugin

[USAGE] : plugin <register plugin_name | list>

vcluter >

vcluter > plugin list

Loaded Plugins: 4

  HelloWorld
```

```
  HelloWorld2

  HelloWorld3

  HTCondor

vcluter > plugin register HTCondor

vcluter > condor_status
-----------
-----------
-----------


File: project-fermi/plugins/batch/proxy-HtCondorReadme.txt
##°°This plugin program uses proxy program to communicate with the htcondor system.
-----------
-----------
-----------


File: project-fermi/plugins/cloud/ec2Readme.txt
## INTRODUCTION



The OpenNebula EC2 Query is a web service that enables you to launch and manage virtual machines

in your OpenNebula installation through the  Amazon EC2 Query Interface. In this way, you can use

any EC2 Query tool or utility to access your Private Cloud. The EC2 Query web service is implemented

 upon the new OpenNebula Cloud API (OCA) layer that exposes the full capabilities of an OpenNebula

 private cloud; and  Sinatra, a widely used light web framework.


The current implementation includes the basic routines to use a Cloud, namely: image upload and

registration, and the VM run, describe and terminate operations. The following sections explains

 you how to install and configure the EC2 Query web service on top of a running OpenNebula cloud.




## CLOUD LOADING CONFIGURATION FILE FORMAT
```

```
[cloudelement]

Name =

Interface =

type =

endpoint =

accesskey =

secretkey =

instancetype =

image =

max =

version =

signatureversion =

signaturemethod =


-----------
-----------
-----------


File: project-fermi/plugins/cloud/proxy-opennebulaReadme.txt
This plugin uses proxy program to communicate with a cloud system who runs proxy and opennebula structure.
-----------
-----------
-----------


File: project-fermi/client/vcluser/client/VclusterClient.java
package vcluser.client;


import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.net.MalformedURLException;

import java.rmi.Naming;
```

```
import java.rmi.NotBoundException;
import java.rmi.RemoteException;

import vcluster.ui.remote.RemoteInterface;
/**
 *This class is a vcluster client
 */
public class VclusterClient {

        /**
         *A prompt which is waiting for accepting command line from command shell input.
         */
        private  String promptGen() {

            String userCmd = "";

            /* prompt */
            System.out.print("vcluter@"+"FERMICLOUD"+"> ");

            InputStreamReader input = new InputStreamReader(System.in);
            BufferedReader reader = new BufferedReader(input);

            try {
                   /* get a command string */
               userCmd = reader.readLine();
            }
```

```
            catch(Exception e){return null;}
            /* execution here */

            return userCmd;

        }

        /**
         *
         * @param args
         */
        public static void main(String[] args) {
               // TODO Auto-generated method stub
               try {
                      VclusterClient vc = new VclusterClient();
                      String cm="rmi://"+serverAddr+":"+serverPort+"/command";
                      RemoteInterface ri = (RemoteInterface) Naming.lookup(cm);
                      String more = "";
                      do {
                             more = vc.promptGen();
                             if(more.equalsIgnoreCase("quit")||more.equalsIgnoreCase("exit")){
                                    System.exit(0);
                             }
                             String str = ri.getRemoteCommand(more);
                             System.out.println(str);
                      }while (true);
```

```
		} catch (MalformedURLException | RemoteException | NotBoundException e) {

			// TODO Auto-generated catch block

			e.printStackTrace();

			System.out.println(" Connection refused to server");



		}




	}



	private static String serverAddr = "fermicloudpp005.fnal.gov";

	private static String serverPort = "1099";



}


-----------
-----------
-----------


File: project-fermi/plugins-src/vcluster/plugin/balancer/fermibalancer/Algorithm.java
package vcluster.plugin.balancer.fermibalancer;

import java.util.ArrayList;
import java.util.concurrent.CopyOnWriteArrayList;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;


public class Algorithm {

	private static CopyOnWriteArrayList<Host> privateHostList;
	private static CopyOnWriteArrayList<Host> publicHostList;
	private static CopyOnWriteArrayList<VM> vmWaitQueue;
	private static CopyOnWriteArrayList<VM> vmList;
	public static final int BUFFER_TIME=10000;
	public static boolean ready=false;
```

```
	static{
		privateHostList=new CopyOnWriteArrayList<Host>();
		publicHostList=new CopyOnWriteArrayList<Host>();
		setVmList(new CopyOnWriteArrayList<VM>());
		vmWaitQueue=new CopyOnWriteArrayList<VM>();
	}
	public static CopyOnWriteArrayList<Host> getPrivateHostList() {
		return privateHostList;
	}
	public static void setPrivateHostList(CopyOnWriteArrayList<Host> privateHostList) {
		Algorithm.privateHostList = privateHostList;
	}
	public static CopyOnWriteArrayList<Host> getPublicHostList() {
		return publicHostList;
	}
	public static void setPublicHostList(CopyOnWriteArrayList<Host> publicHostList) {
		Algorithm.publicHostList = publicHostList;
	}

	public static CopyOnWriteArrayList<SystemStat>getEstimateSysStatus(CopyOnWriteArrayList<SystemStat> current, int
max_time_window){
		long currentTime=System.currentTimeMillis();
		currentTime=currentTime/1000*1000;
		boolean find=false;
		int i=0;
		if(!current.isEmpty()){
		for( ;i<current.size();i++){
			if(current.get(i).time==currentTime){
				find=true;
				break;
			}
		}
		}
		CopyOnWriteArrayList<SystemStat> newStat=new CopyOnWriteArrayList<SystemStat>();
		for(int j=0;j<max_time_window;j++){
			if(current.isEmpty()){
				SystemStat s=new SystemStat();
				s.time=currentTime+j*1000;
				s.cpuUtil=0;
				s.ioUtil=0;
				newStat.add(s);

			}else{
			if(find&&i<current.size()){
				newStat.add(current.get(i));
				i++;
			}
			else{
				SystemStat s=current.get(current.size()-1);
				s.time=currentTime+j*1000;
				newStat.add(s);
			}
		}
```

```
                }
                return newStat;
        }
    public static Host BestFitHost(VM v, boolean debug){
        Host h=null;
        VM tmp=new VM();
        tmp.setStartTime(v.getStartTime());
        tmp.setWaitTime(v.getWaitTime());
        int minBootTime=0;;
        int waitTime=0;
        Lock lock=new ReentrantLock();
        try{
                if(lock.tryLock()){
                        VM lastVM=null;
                        if(Algorithm.getVmWaitQueue().size()<2)
                                lastVM=Algorithm.getVmWaitQueue().get(0);
                        else
                            lastVM=Algorithm.getVmWaitQueue().get(Algorithm.getVmWaitQueue().size()-2);
                        long est=lastVM.getStartTime()+lastVM.getWaitTime();
                        v.setStartTime(est);
                        waitTime=(int)(est-tmp.getStartTime());
                        Host tmpHost=estimateDeploy(v,debug);
                        minBootTime=v.getBootTime();
                        h=tmpHost;
                        tmp.setOverhead(v.getOverhead());
                        for(int i=0;i<Algorithm.vmWaitQueue.size()-2;i++){
                                lastVM=Algorithm.vmWaitQueue.get(i);
                                if(lastVM.getStartTime()+lastVM.getWaitTime()+lastVM.getOverhead().transTime()*1000>est){
                                        long
newStart=lastVM.getStartTime()+lastVM.getWaitTime()+lastVM.getOverhead().transTime()*1000;
                                        int wait=(int)(newStart-tmp.getStartTime());
                                        v.setStartTime(newStart);
                                        v.setWaitTime(0);
                                        tmpHost=estimateDeploy(v,debug);
                                        if(debug)
                                                System.out.println("Delay: "+ Integer.toString(wait/1000)+" Boot Time:
"+Integer.toString(v.getBootTime()+wait/1000));
                                        if(v.getBootTime()+wait/1000<=minBootTime){
                                                minBootTime=v.getBootTime()+wait/1000;
                                                waitTime=wait;
                                                h=tmpHost;
                                                tmp.setOverhead(v.getOverhead());
                                        }
                                }
                        }
                        v.setStartTime(tmp.getStartTime());
                        v.setWaitTime(waitTime+tmp.getWaitTime());
                        v.setBootTime(minBootTime);
                        v.setOverhead(tmp.getOverhead());
                        v.setHostId(h.getId());
                        v.setHostName(h.getAddress());
                        h.getVmList().add(v);
                        h.setMemorySize(Math.max(0, h.getMemorySize()-2000));
```

```
                        h.setSysStat(updateHostSysStat(h,v));
                }
        }catch(Exception e){
                e.printStackTrace();
        }finally{
                lock.unlock();
        }
        return h;
    }
    public static CopyOnWriteArrayList<SystemStat> updateHostSysStat(Host h, VM v){
        CopyOnWriteArrayList<SystemStat> temp =getEstimateSysStatus(h.getSysStat(),Host.MAX_TIME_FRAME);
        for(int i=0;i<Host.MAX_TIME_FRAME;i++){
                temp.get(i).cpuUtil=Math.min(1, temp.get(i).cpuUtil+v.getOverhead().getCpuU().get(i));
                temp.get(i).ioUtil=Math.min(1, temp.get(i).ioUtil+v.getOverhead().getIOU().get(i));
        }
        return temp;
    }
        public static Host estimateDeploy(VM v,boolean debug){
                //TODO return the host that with the minimum launching overhead
                int full=0;
                int minLaunchTime=Integer.MAX_VALUE;
                long currentTime=v.getStartTime();
                CopyOnWriteArrayList<SystemStat> sysStat=new CopyOnWriteArrayList<SystemStat>();
                Host minHost=null;
                VmLaunchOverheadModel model=null;
                for(Host h:privateHostList){
                        if(h.getVmList().size()==h.getCpuCount())
                        {
                                full++;
                                continue;
                        }
                        int count=calConcurrentIOUsage(h,currentTime);
                        int count_read=calConcurrentRead(currentTime);
                        //boolean cache=false;
                        boolean cache=ifCached(h,currentTime);
                        if(h.getMemorySize()<=16000)
                                cache=false;
                        CopyOnWriteArrayList<SystemStat> temp =getEstimateSysStatus(h.getSysStat(),Host.MAX_TIME_FRAME);
                        //int imageSize=CloudManager.getCloudList().get(h.getCloudName()).getImageSize();
                        //test
                        int imageSize=2074;
                        String msg="Image size: "+Integer.toString(imageSize)+"Mb";
                        if(debug){
                                System.out.println(msg);
                                FermiBalancer.writeLogFile(msg);
                        }


                        model=new VmLaunchOverheadModel(h.getReadBandwidth(), h.getWriteBandwidth(),h.getCacheBandwidth()
                                        ,Math.min(h.getReadBandwidth(),FermiBalancer.imageRepoReadSpeed/count_read),
h.getWriteBandwidth()/count,
h.getCacheBandwidth()/count,h.getMemorySize(),imageSize,h.getEpsilon(),h.getGamma(),h.getBeta(),h.getBootMin(),
                                        h.getA(),h.getB(),h.getC(),cache);
```

```
                        //model.setImageSize(160000);
                        ArrayList<Double> tempCpu=new ArrayList<Double>();
                        ArrayList<Double> tempIO=new ArrayList<Double>();
                    for(int i=0;i<Host.MAX_TIME_FRAME;i++){
                        double cpu=0;
                        double io=0;
                        if(i<model.getStartTime()){
                                cpu=0;io=0;
                        }else if(i>=model.getStartTime()&&i<=model.transTime()){
                                cpu=model.util_trans(i);
                                io=model.IO(i);
                        }else{
                            if(i==0)
                            {
                                    cpu=model.U_Boot(i, temp.get(0).ioUtil);
                                    io=model.IO(i);
                                }
                            else
                            {
                                cpu=model.U_Boot(i, temp.get(i-1).ioUtil);
                                io=model.IO(i);
                            }
                        }
                        tempCpu.add(cpu);
                        tempIO.add(io);
                        temp.get(i).cpuUtil=Math.min(1, temp.get(i).cpuUtil+cpu);
                        temp.get(i).ioUtil=Math.min(1, temp.get(i).ioUtil+io);
                    }
                    model.setCpuU(tempCpu);
                    model.setIOU(tempIO);
                    //System.out.println(model.dump());
                    int bootTime=model.bootTime();
                    if(debug)
                        System.out.printf("If deploy on host %s, booting time is %d seconds!\n", h.getId(),bootTime);
                    if((bootTime<minLaunchTime)|| (bootTime==minLaunchTime && h.getVmList().size()<minHost.getVmList().size())){
                        minLaunchTime=bootTime;
                        minHost=h;
                        sysStat=temp;

                    }
                }
                if(full<privateHostList.size()){
                        if(minHost!=null){
                                //VM v=new VM();
                                //v.setStartTime(currentTime);
                                v.setBootTime(minLaunchTime);
                                v.setHostId(minHost.getId());
                                v.setOverhead(model);
                                v.setId(minHost.getId()+Integer.toString(minHost.getVmList().size()));
                                //minHost.addVM(v);
                                //minHost.setSysStat(sysStat);
                                //minHost.setMemorySize(Math.max(0, minHost.getMemorySize()-2000));
```

```
                                if(debug)
                                        System.out.printf("[Deploy]: VM %s to Host %s, est launch time: %d\n",
v.getId(),minHost.getId(),v.getBootTime());
                                return minHost;
                        }
                        else
                                System.out.println("[Error]: CANNOT find valid host for VM!\n");
                }else{
                        publicHostList.get(0).addVM(new VM());
                        return publicHostList.get(0);
                }
                return null;
        }

        //Test code for estimate
        public static Host estimateDeployTest(long time){
                int full=0;
                int minLaunchTime=Integer.MAX_VALUE;
                long currentTime=time;
                CopyOnWriteArrayList<SystemStat> sysStat=new CopyOnWriteArrayList<SystemStat>();
                Host minHost=null;
                VmLaunchOverheadModel model=null;
                for(Host h:privateHostList){
                        if(h.getVmList().size()==h.getCpuCount())
                        {
                                full++;
                                continue;
                        }
                        int count=calConcurrentIOUsage(h,currentTime);
                        int count_read=calConcurrentRead(currentTime);
                        int readSpeed=Math.max(1, Math.min(h.getReadBandwidth(),FermiBalancer.imageRepoReadSpeed/count_read));
                        int writeSpeed=Math.max(1, h.getWriteBandwidth()/count);
                        boolean cache=ifCached(h,currentTime);
                        if(h.getMemorySize()<=16000)
                                cache=false;
                        CopyOnWriteArrayList<SystemStat> temp =getEstimateSysStatus(h.getSysStat(),Host.MAX_TIME_FRAME);
                        model=new VmLaunchOverheadModel(h.getReadBandwidth(), h.getWriteBandwidth(),h.getCacheBandwidth()
                                        ,readSpeed, writeSpeed
,h.getCacheBandwidth(),h.getMemorySize(),2704,h.getEpsilon(),h.getGamma(),h.getBeta(),h.getBootMin(),
                                        h.getA(),h.getB(),h.getC(),cache);
                        //model.setImageSize(160000);

        //model.setReadSpeed(Math.min(h.getReadBandwidth(),FermiBalancer.imageRepoReadSpeed/calConcurrentRead(currentTime)));
                        //model.setWriteSpeed(h.getWriteBandwidth()/count);
                        ArrayList<Double> tempCpu=new ArrayList<Double>();
                        ArrayList<Double> tempIO=new ArrayList<Double>();
                    for(int i=0;i<Host.MAX_TIME_FRAME;i++){
                        double cpu=0;
                        double io=0;
                        if(i<model.getStartTime()){
                                cpu=0;io=0;
                        }else if(i>=model.getStartTime()&&i<=model.transTime()){
                                cpu=model.util_trans(i);
```

```
                io=model.IO(i);
            }else{
                if(i==0)
                {
                    cpu=model.U_Boot(i, temp.get(0).ioUtil);
                    io=model.IO(i);
                }
                else
                {
                    cpu=model.U_Boot(i, temp.get(i-1).ioUtil);
                    io=model.IO(i);
                }
            }
            tempCpu.add(cpu);
            tempIO.add(io);
            temp.get(i).cpuUtil=Math.min(1, temp.get(i).cpuUtil+cpu);
            temp.get(i).ioUtil=Math.min(1, temp.get(i).ioUtil+io);
        }
        model.setCpuU(tempCpu);
        model.setIOU(tempIO);
        //System.out.print(model.dump());
        FermiBalancer.writeLogFile(model.dump());
        int bootTime=model.bootTime();
        System.out.printf("If deploy on host %s, booting time is %d seconds!\n", h.getId(),bootTime);
        if((bootTime<minLaunchTime)|| (bootTime==minLaunchTime && h.getVmList().size()<minHost.getVmList().size())){
            minLaunchTime=bootTime;
            minHost=h;
            sysStat=temp;

        }
    }
    if(full<privateHostList.size()){
        if(minHost!=null){
            VM v=new VM();
            v.setStartTime(currentTime);
            v.setBootTime(minLaunchTime);
            v.setHostId(minHost.getId());
            v.setOverhead(model);
            v.setId(minHost.getId()+Integer.toString(minHost.getVmList().size()));
            minHost.addVM(v);
            minHost.setSysStat(sysStat);
            minHost.setMemorySize(Math.max(0, minHost.getMemorySize()-2000));
            System.out.printf("[Deploy]: VM %s to Host %s\n", v.getId(),minHost.getId());
            return minHost;
        }
        else
            System.out.println("[Error]: CANNOT find valid host for VM!");
    }else{
        publicHostList.get(0).addVM(new VM());
        return publicHostList.get(0);
    }
    return null;
}
```

```
public static double aveLaunchTime(){
    int totalTime=0;
    int totalVM=0;
    for(Host h:privateHostList){
        for(VM v:h.getVmList()){
            totalTime+=v.getBootTime()+v.getWaitTime()/1000;
            totalVM+=1;
        }
    }
    return (double)totalTime/(double) totalVM;
}
public static int calConcurrentIOUsage(Host h, long currentTime){
    int count=1;
    for(VM v:h.getVmList()){
        int time=v.getOverhead().transTime()+BUFFER_TIME;
        long timeTemp=(long)v.getStartTime()+(long)time*1000;
        if(timeTemp>currentTime)
            count++;
    }
    return count;
}
public static int calConcurrentRead(long currentTime){
    int count=1;
    for(VM v:vmWaitQueue){
        if((v.getStartTime()+v.getWaitTime()+v.getOverhead().transTime()*1000+BUFFER_TIME)>currentTime)
            count++;
    }
    return count;
}
//TODO: Need a function to count the simultaneous read from image repo
public static boolean ifCached(Host h, long currentTime){
    if(h.getVmList().isEmpty())
        return false;
    else{
    VM v=h.getVmList().get(h.getVmList().size()-1);
    if(v.getOverhead().ifCached())
        return true;
    else{
        long tFinish=v.getStartTime()+v.getOverhead().transTime();
        if(currentTime>=tFinish&&currentTime<=tFinish+BUFFER_TIME)
            return true;
        else
            return false;
    }
    }
}
public static void dump(){
    System.out.println("-----------------------------------------------------------"+System.lineSeparator());
    String tHid=String.format("%-8s", "HostID");
    String tCapacity=String.format("%-10s", "Capacity");
    String tRunVm=String.format("%-8s", "RunVMs");
    String tType=String.format("%-8s", "TYPE");
    String tCloudName=String.format("%-8s", "Cloud");
```

```
        String tHAddress=String.format("%-20s", "Address");
        System.out.println(tHid+tCapacity+tRunVm+tType+tCloudName+tHAddress+System.lineSeparator());
        System.out.println("------------------------------------------------------------"+System.lineSeparator());
        for(Host h:privateHostList){
                String hHid=String.format("%-8s",h.getId() );
                String hCapacity=String.format("%-10s", Integer.toString(h.getCpuCount()));
                String hRunVm=String.format("%-8s", Integer.toString(h.getVmList().size()));
                String hType=String.format("%-8s", "PRIVATE");
                String hCloudName=String.format("%-8s", h.getCloudName());
                String hAddress=String.format("%-20s", h.getAddress());
                System.out.println(hHid+hCapacity+hRunVm+hType+hCloudName+hAddress+System.lineSeparator());
        }
        for(Host h:publicHostList){
                String hHid=String.format("%-8s",h.getId() );
                String hCapacity=String.format("%-10s", Integer.toString(h.getCpuCount()));
                String hRunVm=String.format("%-8s", Integer.toString(h.getVmList().size()));
                String hType=String.format("%-8s", "PUBLIC");
                String hCloudName=String.format("%-8s", h.getCloudName());
                System.out.println(hHid+hCapacity+hRunVm+hType+hCloudName+System.lineSeparator());
        }
        System.out.println("-----------------------VM LIST----------------------------"+System.lineSeparator());
        String tVMid=String.format("%-8s", "VMID");
        String tVmName=String.format("%-20s", "VmName");
        String tVmHost=String.format("%-20s", "Host");
        String tVmBoot=String.format("%-8s", "BootTime");
        System.out.println(tVMid+tVmName+tVmHost+tVmBoot+System.lineSeparator());
        System.out.println("------------------------RUNING---------------------------"+System.lineSeparator());
        for(VM v:vmList){
                String hVmId=String.format("%-8s", v.getId());
                String hVmName=String.format("%-20s", v.getDnsName());
                String hVmHost=String.format("%-20s", v.getHostName());
                String hVmBoot=String.format("%-8s", v.getRealBootTime());
                System.out.println(hVmId+hVmName+hVmHost+hVmBoot+System.lineSeparator());
        }
        System.out.println("-----------------------WAITING----------------------------"+System.lineSeparator());
        for(VM v:vmWaitQueue){
                String hVmId=String.format("%-8s", v.getId());
                String hVmName=String.format("%-20s", v.getDnsName());
                String hVmHost=String.format("%-20s", v.getHostName());
                String hVmBoot=String.format("%-8s", v.getBootTime()+v.getWaitTime()/1000);
                String hVmWait=String.format("%-8s", v.getWaitTime()/1000);
                System.out.println(hVmId+hVmName+hVmHost+hVmBoot+hVmWait+System.lineSeparator());
        }
    }
        public static CopyOnWriteArrayList<VM> getVmList() {
                return vmList;
        }
        public static void setVmList(CopyOnWriteArrayList<VM> vmList) {
                Algorithm.vmList = vmList;
        }
        public static CopyOnWriteArrayList<VM> getVmWaitQueue() {
                return vmWaitQueue;
        }
```

```
        public static void setVmWaitQueue(CopyOnWriteArrayList<VM> vmWaitQueue) {
                Algorithm.vmWaitQueue = vmWaitQueue;
        }

}

-----------
-----------
-----------


File: project-fermi/plugins-src/vcluster/plugin/balancer/fermibalancer/clientInfoCollector.java
package vcluster.plugin.balancer.fermibalancer;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.Socket;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

import vcluster.plugin.balancer.fermibalancer.VM.VmStat;

public class clientInfoCollector implements Runnable {

        private final Socket clientSocket;
        private long bootTime;
        private String hostName;
        private Lock lock;

    public clientInfoCollector(Socket clientSocket) {
        this.clientSocket = clientSocket;
        this.lock=new ReentrantLock();
    }

        @Override
        public void run() {
                // TODO Not work properply, needs to be done.
                try {
                        BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream(), "UTF-8"));
                        String str=null;
                        str=in.readLine();
                        System.out.println(str);
                        Date date = new SimpleDateFormat("EEE MMM d H:m:s zzz yyyy", Locale.ENGLISH).parse(str);
                        bootTime=date.getTime();
                        str=in.readLine().trim();
                        System.out.println(str);
                        hostName=str;
```

```
                        if(lock.tryLock()){
                        for(int i=0;i<Algorithm.getVmWaitQueue().size();i++){
                                if(Algorithm.getVmWaitQueue().get(i).getDnsName().equalsIgnoreCase(str)){
                                        long start=Algorithm.getVmWaitQueue().get(i).getStartTime();
                                        Date startD=new Date(start);
                                        Algorithm.getVmWaitQueue().get(i).setRealBootTime((int)(bootTime-start)/1000);
                                        Algorithm.getVmWaitQueue().get(i).setStatus(VmStat.RUNNING);
                                        Host h=hostInfoCollector.findHost(Algorithm.getVmWaitQueue().get(i).getHostName());
                                        String line = "VM: "+ Algorithm.getVmWaitQueue().get(i).getId()+" Start: " +
startD.toString()+
                                                "Booted: "+ date.toString()+" Estimated Boot Time: "+
Integer.toString(Algorithm.getVmWaitQueue().get(i).getBootTime())
                                                +"s Actual Boot Time:
"+Integer.toString(Algorithm.getVmWaitQueue().get(i).getRealBootTime())+" s";
                                        Algorithm.getVmList().add(Algorithm.getVmWaitQueue().get(i));
                                        Algorithm.getVmWaitQueue().remove(i);
                                        if(h!=null)
                                                h.trainning(false);
                                        FermiBalancer.writeLaunchLogFile(line);
                                        break;
                                        }
                                }
                        }
                        clientSocket.close();
                        Algorithm.dump();
                } catch (IOException e) {
                    e.printStackTrace();
                } catch (ParseException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                finally{
                        lock.unlock();
                }
        }

        public long getBootTime() {
                return bootTime;
        }

        public void setBootTime(long bootTime) {
                this.bootTime = bootTime;
        }

        public String getHostName() {
                return hostName;
        }

        public void setHostName(String hostName) {
                this.hostName = hostName;
        }
}
```

```
-----------
-----------
-----------


File: project-fermi/plugins-src/vcluster/plugin/balancer/fermibalancer/Evaluation.java
package vcluster.plugin.balancer.fermibalancer;

import java.util.ArrayList;


public class Evaluation {

        /**
         * Mean Absolute Scaled Error
         * @param actual
         * @param predict
         * @return
         */
        public static double MASE(ArrayList<Double> actual, ArrayList<Double> predict){
                double sum=0;
                for(int i=1;i<actual.size();i++){
                        double e=(actual.get(i)-predict.get(i));
                        double y=0;
                        for(int j=i;j>0;j--){
                                y=y+Math.abs(predict.get(j)-predict.get(j-1));
                        }
                        double temp=1.00/(100.00-1.00)*y;
                        sum=sum+Math.abs(e/temp);
                }
                return sum/actual.size();
        }

        /**
         * Mean Square Error
         * @param actual
         * @param predict
         * @return
         */
        public static double MSE(ArrayList<Double> actual, ArrayList<Double> predict){
                double sum=0;
                for(int i=0;i<actual.size();i++){
                        double e=actual.get(i)-predict.get(i);
                        sum=sum+Math.pow(e, 2);

                }
                return sum/actual.size();
        }
        /**
         * Mean Absolute Percentage Error
         * @param actual
         * @param predict
         * @return
```

```
	 */
	public static double MAPE(ArrayList<Double> actual, ArrayList<Double> predict){
		double sum=0;
		for(int i=0;i<actual.size();i++){
			double e=actual.get(i)-predict.get(i);
			sum=sum+Math.abs(e/actual.get(i))*100;
		}
		return sum/actual.size();
	}
	/**
	 * Root Mean Square Percentage Error
	 * @param actual
	 * @param predict
	 * @return
	 */
	public static double EMSPE(ArrayList<Double> actual, ArrayList<Double> predict){
		double sum=0;
		for(int i=0;i<actual.size();i++){
			double e=actual.get(i)-predict.get(i);
			sum=sum+Math.pow(e/actual.get(i)*100,2);
		}
		return Math.sqrt(sum/actual.size());
	}
}

-----------
-----------
-----------


File: project-fermi/plugins-src/vcluster/plugin/balancer/fermibalancer/Exp.java
package vcluster.plugin.balancer.fermibalancer;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Random;

public class Exp {

	public static final String fileName="LaunchPattern";

	private static ArrayList<Integer> pattern;

	/**
	 * Randomly generate VM launch pattern
	 * @param n
	 * @param maxInterval
	 */
```

```
	public static void randomPattern(int n, int maxInterval){
		Random r=new Random(System.currentTimeMillis());
		pattern=new ArrayList<Integer>();
		for(int i=0;i<n;i++){
			pattern.add(r.nextInt(maxInterval));
		}
	}

	public static void writePattern(){
		File f=new File(fileName);
		try {
			StringBuilder sb=new StringBuilder();
			for(int i=0;i<pattern.size();i++){
				if(i==pattern.size()-1)
					sb.append(Integer.toString(pattern.get(i)));
				else
					sb.append(Integer.toString(pattern.get(i))+",");
			}
			FileWriter fw=new FileWriter(f,false);
			fw.write(sb.toString());
			fw.close();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

	}

	public static ArrayList<Integer> readPattern(){
		File f=new File(fileName);
		if(!f.exists())
		{
			return null;
		}

	BufferedReader br ;
		try{
			br = new BufferedReader(new FileReader(f));
			String aLine = "";
			pattern=new ArrayList<Integer> ();
			while ((aLine = br.readLine()) != null) {
				String [] sub=aLine.split(",");
				for(int i=0;i<sub.length;i++){
					pattern.add(Integer.parseInt(sub[i]));
				}
			}

			br.close();
		} catch (FileNotFoundException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (IOException e) {
			// TODO Auto-generated catch block
```

```
                e.printStackTrace();
            } catch (Exception e){
                e.printStackTrace();
            }
            return pattern;
        }

        public static  ArrayList<Integer> getPattern() {
            return pattern;
        }

        public static void setPattern(ArrayList<Integer> value) {
            pattern = value;
        }

}

-----------
-----------
-----------


File: project-fermi/plugins-src/vcluster/plugin/balancer/fermibalancer/FermiBalancer.java
package vcluster.plugin.balancer.fermibalancer;


import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.Writer;
import java.util.ArrayList;
import java.util.Date;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

import vcluster.plugInterfaces.LoadBalancer;
import vcluster.plugin.balancer.fermibalancer.VM.VmStat;
import vcluster.elements.*;
import vcluster.executors.*;
import vcluster.managers.*;
import vcluster.ui.*;

public class FermiBalancer implements LoadBalancer{

        /*
         *   There should be several different threads to handle different info
         *   1. update infomation. need to ensure the consistency between load balancer and real info
         *   2. keep tracking of the start time info, training and updating host parameters
         *   3. main threads will be decision making process
         */
        public static final String logFile="log//Balancer_Log";
    public static final String VmLaunchLogFile="log//VmLaunch_Log";
    public static final int port=9738;
    public static final int imageRepoReadSpeed=100;
```

```
    public static final int MAX_CONVERGE_ROUND=10;
    public static final int MIN_CONVERGE_ROUND=5;
    public static final double CONVERGE_THRESHOLD=2;
    public static final double MIN_GRANULARITY=0.00001;
    private Lock lock;
       @Override
       public void activate() {
               // TODO Auto-generated method stub
               //CloudmanExecutor.register(new CmdComb("cloudman register fermi-proxy.conf"));
               //CloudmanExecutor.register(new CmdComb("cloudman register amazon.conf"));
           for(Cloud c:CloudManager.getCloudList().values()){
               CloudmanExecutor.load(new CmdComb("cloudman load "+ c.getCloudName()));
           }
           Thread infoCollectorThread = new Thread(new InfoCollector(port));
        infoCollectorThread.start();
        Thread infoUpdateThread=new Thread(new hostInfoCollector());
        infoUpdateThread.start();
        Thread vmLaunchThread=new Thread(new VmLauncher());
        vmLaunchThread.start();
        lock=new ReentrantLock();

               while (!Algorithm.ready){
                       try {
                               Thread.sleep(10000);
                       } catch (InterruptedException e) {
                               // TODO Auto-generated catch block
                               e.printStackTrace();
                       }
               }
               testDefaultLaunch();
               try {
                       Thread.sleep(30000);
                       } catch (InterruptedException e) {
                               // TODO Auto-generated catch block
                               e.printStackTrace();
                       }
               test();
               /*for(int i=0;i<79;i++){
                       try{
               if(lock.tryLock()){
               //Thread.sleep(30000);
               long currentTime=System.currentTimeMillis();
               VM v=new VM();
               Host h=Algorithm.estimateDeploy(v);
           //String str=CloudManager.getCurrentCloud().createVMonHost(1, h.getId());
           String str=CloudManager.getCurrentCloud().createVM(1, h.getAddress());
               String [] sub=str.split("\\s+");

           v.setId(sub[1]);
           v.setDnsName(sub[2]);
           v.setStartTime(currentTime);
           v.setHostId(h.getId());
           v.setHostName(h.getAddress());
```

```
        Algorithm.getVmWaitQueue().add(v);
        Algorithm.dump();
          }*/
          //Thread.sleep(60000);

 // }catch(Exception e){
 //     e.getStackTrace();
 // }
 //finally{
//      lock.unlock();
 // }
  //}

  }

 private void test(){
        FermiBalancer.writeLaunchLogFile("VM Simulaneous launch with algorithm-------------");
        ArrayList<Integer> timing=Exp.readPattern();
        if (timing==null||timing.isEmpty()){
                System.out.println("[ERROR: ] No pattern available!");
        }
        else{
                lock= new ReentrantLock();
                try{
                        if(lock.tryLock()){
                                long currentTime=System.currentTimeMillis()/1000*1000;
                                int sum=0;
                                for(int i=0;i<timing.size();i++){
                                        sum=timing.get(i)+sum;
                                        VM v=new VM();
                                        v.setStartTime(currentTime+sum*1000);
                                        v.setStatus(VmStat.SCHEDULE);
                                        Algorithm.getVmWaitQueue().add(v);
                                        v=Algorithm.getVmWaitQueue().get(Algorithm.getVmWaitQueue().size()-1);
                                        Host h=Algorithm.BestFitHost(v,false);

                                        v.setStatus(VmStat.PENDING);
                                        //v.setWaitTime(0);
                                        v.setHostName(h.getAddress());

                                }
                                for(VM v:Algorithm.getVmWaitQueue()){
                                        Date d=new Date(v.getStartTime());
                                        System.out.println(d.toString()+" "+Integer.toString(v.getWaitTime()));
                                }
                        }
                }catch (Exception e){
                        e.printStackTrace();
                }finally{
                        lock.unlock();
                }
        }
        while (!Algorithm.getVmWaitQueue().isEmpty()){
```

```
        try {
                        Thread.sleep(100000);
                } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
 }
 for(VM v:Algorithm.getVmList()){
        deleteVM(v);
 }
}

 private void testDefaultLaunch(){
        FermiBalancer.writeLaunchLogFile("VM Simulaneous launch with OpenNebula Defualt Scheduler-------------");
        ArrayList<Integer> timing=Exp.readPattern();
        if (timing==null||timing.isEmpty()){
                System.out.println("[ERROR: ] No pattern available!");
        }
        else{
                lock= new ReentrantLock();
                try{
                        if(lock.tryLock()){
                                long currentTime=System.currentTimeMillis()/1000*1000;
                                int sum=0;
                                for(int i=0;i<timing.size();i++){
                                        sum=timing.get(i)+sum;
                                        VM v=new VM();
                                        v.setStartTime(currentTime+sum*1000);
                                        v.setStatus(VmStat.SCHEDULE);
                                        Algorithm.getVmWaitQueue().add(v);
                                        //v=Algorithm.getVmWaitQueue().get(Algorithm.getVmWaitQueue().size()-1);
                                        //Host h=Algorithm.BestFitHost(v,false);

                                        v.setStatus(VmStat.PENDING);
                                        //v.setWaitTime(0);
                                        v.setHostName("-1");

                                }
                                for(VM v:Algorithm.getVmWaitQueue()){
                                        Date d=new Date(v.getStartTime());
                                        System.out.println(d.toString()+" "+Integer.toString(v.getWaitTime()));
                                }
                        }
                }catch (Exception e){
                        e.printStackTrace();
                }finally{
                        lock.unlock();
                }
        }
        while (!Algorithm.getVmWaitQueue().isEmpty()){
        try {
                        Thread.sleep(100000);
                } catch (InterruptedException e) {
```

```
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }
        for(VM v:Algorithm.getVmList()){
                deleteVM(v);
        }
        }
        public static void writeLogFile(String line){
                try{
                Writer output;
                output = new BufferedWriter(new FileWriter(FermiBalancer.logFile,true));
                Date d=new Date(System.currentTimeMillis());
                String str="["+d.toString()+"] "+ line+System.lineSeparator();
                output.append(str);
                output.close();
        }catch(Exception e){
                e.getStackTrace();
        }
        }

        public static void writeLaunchLogFile(String line){
                try{
                Writer output;
                output = new BufferedWriter(new FileWriter(FermiBalancer.VmLaunchLogFile,true));
                Date d=new Date(System.currentTimeMillis());
                String str="["+d.toString()+"] "+ line+System.lineSeparator();
                output.append(str);
                output.close();
        }catch(Exception e){
                e.getStackTrace();
        }
        }


        private void deleteVM(VM v){

                int index=hostInfoCollector.findHost(v.getHostName(), Algorithm.getPrivateHostList());
                if(index!=-1){
                        String cloudname=Algorithm.getPrivateHostList().get(index).getCloudName();
                        for(Cloud c:CloudManager.getCloudList().values()){
                                if(c.getCloudName().equalsIgnoreCase(cloudname)){
                                        Vm vm=new Vm();
                                        vm.setId(v.getId());
                                        c.destroyVM(vm);
                                        break;
                                }
                        }
                        Algorithm.getPrivateHostList().get(index).getVmList().remove(v);
                        Algorithm.getVmList().remove(v);
                }

        }
```

```
}

-----------
-----------
-----------


File: project-fermi/plugins-src/vcluster/plugin/balancer/fermibalancer/Host.java
package vcluster.plugin.balancer.fermibalancer;

import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.concurrent.CopyOnWriteArrayList;

import vcluster.plugin.balancer.fermibalancer.VM.VmStat;


public class Host {
        private String id;
        private String addr;
        private int port;
        private int memorySize;
        private int cpuCount;
        private CopyOnWriteArrayList<VM> vmList;
        private static final String localImageLocation="/var/lib/one/local/test";
        public static final String configFile="host.conf";
        public static final int MAX_TIME_FRAME = 3600;

        /*
         * sysCpuU and sysIOU map system time (long format) and its corresponding utilization
         */
    private CopyOnWriteArrayList<SystemStat>  sysStat;

    /*
     * System properties for the overhead model
     */
    private String cloudName;
    private int readBandwidth;
    private int writeBandwidth;
    private int cacheBandwidth;
    private double epsilon;
    private double gamma;
    private double beta;
    private double bootMin;
    private double a;
```

```
private double b;
private double c;

private int trainRound;
private boolean trained;

public String getCloudName(){
   return cloudName;
}
public void setCloudName(String value){
   cloudName=value;
}
public String getId(){
   return id;
}
public void setId(String value){
   id=value;
}

public String getAddress(){
   return addr;
}
public void setAddress(String value){
   addr=value;
}

public int getPort(){
   return port;
}
public void setPort(int value){
   port=value;
}

   public int getMemorySize() {
          return memorySize;
   }
   public void setMemorySize(int memorySize) {
          this.memorySize = memorySize;
   }

   public int getCpuCount(){
          return cpuCount;
   }
   public void setCpuCount(int value){
          cpuCount=value;
   }

public CopyOnWriteArrayList<VM> getVmList(){
   return vmList;
}
public void setVmList(CopyOnWriteArrayList<VM> value){
   vmList=value;
}
```

```
public CopyOnWriteArrayList<SystemStat> getSysStat(){
   return sysStat;
}
public void setSysStat(CopyOnWriteArrayList<SystemStat> value){
   sysStat=value;
}


public int getReadBandwidth(){
   return readBandwidth;
}
public void setReadBandwidth(int value){
   readBandwidth=value;
}

public int getWriteBandwidth(){
   return writeBandwidth;
}
public void setWriteBandwidth(int value){
   writeBandwidth=value;
}

public int getCacheBandwidth(){
   return cacheBandwidth;
}
public void setCacheBandwidth(int value){
    cacheBandwidth=value;
}

   public double getEpsilon() {
          return epsilon;
   }
   public void setEpsilon(double epsilon) {
          this.epsilon = epsilon;
   }
   public double getGamma() {
          return gamma;
   }
   public void setGamma(double gamma) {
          this.gamma = gamma;
   }
   public double getBeta() {
          return beta;
   }
   public void setBeta(double beta) {
          this.beta = beta;
   }
   public double getBootMin() {
          return bootMin;
   }
   public void setBootMin(double bootMin) {
          this.bootMin = bootMin;
```

```
    }
    public double getA() {
            return a;
    }
    public void setA(double a) {
            this.a = a;
    }
    public double getB() {
            return b;
    }
    public void setB(double b) {
            this.b = b;
    }
    public double getC() {
            return c;
    }
    public void setC(double c) {
            this.c = c;
    }
    /**
 * Constructor, need id, server address and port number. Initialize the utilization sets(CPU and IO)
     * @param cloudName
     * @param _id
     * @param _addr
     * @param _port
     * @param memSize
     * @param cpuCount
     *
     */
 public Host(String cloudName, String _id, String _addr, int _port, int memSize, int cpuCount) {
    this.cloudName=cloudName;id=_id; addr=_addr; port=_port;memorySize=memSize;this.cpuCount=cpuCount;
    readBandwidth=0;
    writeBandwidth=0;
    cacheBandwidth=0;
    epsilon=0;
    gamma=0;
    beta=0;
    bootMin=0;
    a=0;
    b=0;
    c=0;
    trainRound=0;
    sysStat=new CopyOnWriteArrayList<SystemStat>();
    vmList=new CopyOnWriteArrayList<VM>();
    if(!readConfigFile()){
            if(!benchmarking())
            {
                    String msg="[MSG: ]Trying to bench mark onemore time!";
                    System.out.println(msg);
                    FermiBalancer.writeLogFile(msg);
                    benchmarking();
            }
            if(!readDefualtConfigFile())
```

```
                    defaultParameters();
    }
    if(!checkParameters())
    {

            String line="[Error:] Reading Host Config failure: Missing parameters!\n";
            System.out.println(line);
            FermiBalancer.writeLogFile(line);
                    System.exit(1);
    }
    if(trainRound<FermiBalancer.MAX_CONVERGE_ROUND)
            trained=false;
    else
            trained=true;
    writeConfigFile();
 }
 public boolean readConfigFile(){
    String file="hosts"+File.separator+id;
    File f=new File(file);

    if(!f.exists())
            return false;

    BufferedReader br ;
            try{
                    br = new BufferedReader(new FileReader(file));
                    String aLine = "";
                    while ((aLine = br.readLine()) != null) {
                            if(aLine.contains("#")||!aLine.contains("="))continue;
                            String[] pair = aLine.split("=");
                            if (pair[0].trim().equalsIgnoreCase("ReadBandwidth")) {
                                    readBandwidth = Integer.parseInt(pair[1].trim());
                            }else if(pair[0].trim().equalsIgnoreCase("WriteBandwidth")){
                                    writeBandwidth = Integer.parseInt(pair[1].trim());
                            }else if(pair[0].trim().equalsIgnoreCase("CacheBandwidth")){
                                    cacheBandwidth = Integer.parseInt(pair[1].trim());
                            }else if(pair[0].trim().equalsIgnoreCase("Epsilon")){
                                    epsilon = Double.parseDouble(pair[1].trim());
                            }else if(pair[0].trim().equalsIgnoreCase("Gamma")){
                                    gamma = Double.parseDouble(pair[1].trim());
                            } else if(pair[0].trim().equalsIgnoreCase("Beta")){
                                    beta = Double.parseDouble(pair[1].trim());
                            }else if(pair[0].trim().equalsIgnoreCase("BootMin")){
                                    bootMin = Double.parseDouble(pair[1].trim());
                            } else if(pair[0].trim().equalsIgnoreCase("a")){
                                    a = Double.parseDouble(pair[1].trim());
                            }else if(pair[0].trim().equalsIgnoreCase("b")){
                                    b = Double.parseDouble(pair[1].trim());
                            } else if(pair[0].trim().equalsIgnoreCase("c")){
                                    c = Double.parseDouble(pair[1].trim());
                            } else if(pair[0].trim().equalsIgnoreCase("TRAIN")){
                                    trainRound=Integer.parseInt(pair[1].trim());
                            }
```

```
                }

                br.close();

        }catch(Exception e){
                System.out.println(e.getMessage());
                FermiBalancer.writeLogFile(e.getMessage());
                return false;
        }
    return true;
}

public boolean readDefualtConfigFile(){
    String file=Host.configFile;
    File f=new File(file);
    if(!f.exists())
        return false;

    BufferedReader br ;
        try{
                br = new BufferedReader(new FileReader(file));
                String aLine = "";
                while ((aLine = br.readLine()) != null) {
                        if(aLine.contains("#")||!aLine.contains("="))continue;
                        String[] pair = aLine.split("=");
                        if (pair[0].trim().equalsIgnoreCase("ReadBandwidth")) {
                                if(readBandwidth==0)
                                        readBandwidth = Integer.parseInt(pair[1].trim());
                        }else if(pair[0].trim().equalsIgnoreCase("WriteBandwidth")){
                                if(writeBandwidth==0)
                                        writeBandwidth = Integer.parseInt(pair[1].trim());
                        }else if(pair[0].trim().equalsIgnoreCase("CacheBandwidth")){
                                if(cacheBandwidth==0)
                                        cacheBandwidth = Integer.parseInt(pair[1].trim());
                        }else if(pair[0].trim().equalsIgnoreCase("Epsilon")){
                                epsilon = Double.parseDouble(pair[1].trim());
                        }else if(pair[0].trim().equalsIgnoreCase("Gamma")){
                                gamma = Double.parseDouble(pair[1].trim());
                        } else if(pair[0].trim().equalsIgnoreCase("Beta")){
                                beta = Double.parseDouble(pair[1].trim());
                        }else if(pair[0].trim().equalsIgnoreCase("BootMin")){
                                bootMin = Double.parseDouble(pair[1].trim());
                        } else if(pair[0].trim().equalsIgnoreCase("a")){
                                a = Double.parseDouble(pair[1].trim());
                        }else if(pair[0].trim().equalsIgnoreCase("b")){
                                b = Double.parseDouble(pair[1].trim());
                        } else if(pair[0].trim().equalsIgnoreCase("c")){
                                c = Double.parseDouble(pair[1].trim());
                        }
                }

                br.close();
```

```
        }catch(Exception e){
                System.out.println(e.getMessage());
                FermiBalancer.writeLogFile(e.getMessage());
                return false;
        }
    return true;
}
public boolean benchmarking(){
    String s="Bench Marking Host: "+ this.addr;
    System.out.println(s);
    FermiBalancer.writeLogFile(s);
    writeBandwidth=benchMarkWriteBandwidth();
    String out="Writing Bandwidth : " + Integer.toString(writeBandwidth)+"MB/s";
    System.out.println(out);
    FermiBalancer.writeLogFile(out);
    clearCache();
    readBandwidth=benchMarkReadBandwidth();
    out="Reading Bandwidth : " + Integer.toString(readBandwidth)+"MB/s";
    System.out.println(out);
    FermiBalancer.writeLogFile(s);
    cacheBandwidth=benchMarkReadBandwidth();
    out="Caching Bandwidth : " + Integer.toString(cacheBandwidth)+"MB/s";
    System.out.println(out);
    FermiBalancer.writeLogFile(s);
    cleanBenchMarkFiles();
    if(writeBandwidth!=0&&readBandwidth!=0&&cacheBandwidth!=0)
    {
        String msg="Bench Marking successful!";
        System.out.println(msg);
        FermiBalancer.writeLogFile(msg);
        return true;
    }else{
        String msg="Bench Marking unsuccessful!!";
        System.out.println(msg);
        FermiBalancer.writeLogFile(msg);
        return false;
    }
}

//TODO: Training function to train the parameters for the overhead model
public void trainning(boolean debug){
    /*
     * updating epsilon, gamma, beta and bootmin
     */
    if(trainRound>FermiBalancer.MAX_CONVERGE_ROUND){
        if(!trained){
                writeConfigFile();
                trained=true;
        }
    }else {
        ArrayList<Double> actual=new ArrayList<Double>();
        ArrayList<Double> predict=new ArrayList<Double>();
        for(int i=0;i<vmList.size();i++){
```

```
                if(vmList.get(i).getBootTime()!=-1&&vmList.get(i).getRealBootTime()!=-1){
                        actual.add((double)vmList.get(i).getRealBootTime());
                        predict.add((double)vmList.get(i).getBootTime());
                }
        }
        double error=Evaluation.MASE(actual, predict);
        if(error<=FermiBalancer.CONVERGE_THRESHOLD){
                trainRound++;
        }else{
                boolean terminate=false;
                double granularity=getCurrentGranularity();
                double tmpEpsilon=granularity;
                long start=System.currentTimeMillis();
                long finish=start;
                while (error>FermiBalancer.CONVERGE_THRESHOLD||!terminate){
                        if(granularity<=FermiBalancer.MIN_GRANULARITY){
                                terminate=true;
                                if(debug){
                                        finish=System.currentTimeMillis();
                                        System.out.println("[DEBUG:] Model Tranning Not converged but terminated. Using: "+
                                        Long.toString(finish-start)+" MS.");
                                }
                                break;
                        }
                        else{
                                double minError=error;
                                double minEpsilon=tmpEpsilon;
                                double oriEpsilon=tmpEpsilon;
                                for(int i=0;i<9;i++){
                                        double tmp=calPredictError(tmpEpsilon);
                                        if(tmp<=minError)
                                        {
                                                minError=tmp;
                                                minEpsilon=tmpEpsilon;
                                        }
                                        tmpEpsilon=tmpEpsilon+granularity;
                                }
                                tmpEpsilon=oriEpsilon;
                                for(int i=0;i<9;i++){
                                        tmpEpsilon=tmpEpsilon-granularity;
                                        if(tmpEpsilon<=0)
                                                break;
                                        double tmp=calPredictError(tmpEpsilon);
                                        if(tmp<=minError)
                                        {
                                                minError=tmp;
                                                minEpsilon=tmpEpsilon;
                                        }
                                }
                                tmpEpsilon=minEpsilon;
                                error=minError;
                                granularity=granularity/10;
                        }
```

```
                }
                finish=System.currentTimeMillis();
                if(!terminate&&debug){
                        System.out.println("[DEBUG:] Model Tranning converged and terminated. Using: "+
                                        Long.toString(finish-start)+" MS.");
                }
                if(debug){
                        System.out.println(dumpTrainData());
                        System.out.println(Double.toString(tmpEpsilon));
                }
                this.epsilon=tmpEpsilon;
                trainRound++;
                writeConfigFile();
        }
    }
}

private String dumpTrainData(){
    StringBuilder sb=new StringBuilder();
    String title="--"+this.getAddress()+":Training Data--"+System.lineSeparator();
    sb.append(title);
    String test=String.format("%-8s", "EST.");
    String treal=String.format("%-8s", "REAL");
    sb.append(test+treal+System.lineSeparator());
    for(int i=0;i<vmList.size();i++){
        test=String.format("%-8s",Integer.toString(vmList.get(i).getBootTime()));
        treal=String.format("%-8s", Integer.toString(vmList.get(i).getRealBootTime()));
        sb.append(test+treal+System.lineSeparator());
    }
    sb.append("------------------------------------");
    return sb.toString();
}
public double calPredictError(double epsilon){
    ArrayList<Double> actual=new ArrayList<Double>();
    ArrayList<Double> predict=new ArrayList<Double>();
    for(int i=0;i<vmList.size();i++){
        if(vmList.get(i).getBootTime()!=-1&&vmList.get(i).getRealBootTime()!=-1)
        {
            vmList.get(i).getOverhead().setEpsilon(epsilon);
            actual.add((double)vmList.get(i).getRealBootTime());
            predict.add((double)vmList.get(i).getOverhead().bootTime());

        }
    }
    return Evaluation.MASE(actual, predict);
}

public double getCurrentGranularity(){
    double granularity=0.1;
    while((epsilon-granularity)<=0)
    {
        granularity=granularity/10;
```

```
    }
    return granularity*10;
}

private ArrayList<String> socketToproxy(String cmd){
          String cmdLine=cmd+"\n";
          ArrayList<String> feedBack = new ArrayList<String>();
           Socket socket = null;
           BufferedReader in = null;
           DataOutputStream out = null;
           //System.out.println(cmdLine);
           try {
                 socket = new Socket(addr, port);

               in = new BufferedReader(new InputStreamReader(socket.getInputStream(), "UTF-8"));
               //di = new DataInputStream(socket.getInputStream());
                 out = new DataOutputStream(socket.getOutputStream());
               out.flush();
               /* make an integer to unsigned int */
               int userInput = 5;
               userInput <<= 8;
               userInput |=  1;
               userInput &= 0x7FFFFFFF;

               String s = Integer.toString(userInput);
               byte[] b = s.getBytes();

               out.write(b, 0, b.length);
               out.write(cmdLine.getBytes(), 0, cmdLine.getBytes().length);
               // out.writeBytes("Dump info, test!!!");
              // out.flush();

               String str=null;

               while((str=in.readLine())!=null){
                 str=str.trim();
                 feedBack.add(str);
               }


           } catch (UnknownHostException e) {

                 System.out.print("ERROR: " +e.getMessage());
               closeStream(in, out, socket);
               return feedBack;
           } catch (IOException e) {
                 System.out.print("ERROR: " +e.getMessage());
               closeStream(in, out, socket);
               return feedBack;
           }
           closeStream(in, out, socket);

           return feedBack;
```

```
    }

private static void closeStream(BufferedReader in, DataOutputStream out, Socket socket)
    {
          try {
           if (in != null) in.close();
           if (out != null) out.close();
           if (socket != null) socket.close();
          } catch (IOException e) {
                 e.printStackTrace();
          }
    }

private int benchMarkWriteBandwidth(){
    String cmd="(dd bs=1M count=2000 if=/dev/zero of="+ localImageLocation +" conv=fdatasync)2>&1";
    ArrayList<String> strArr=socketToproxy(cmd);
    for(String s :strArr){
          if (s.contains("copied"))
                 return bandwidthExtract(s);
    }
          return 0;

}

private int benchMarkReadBandwidth(){
    String cmd="(dd bs=1M count=2000 if="+ localImageLocation +" of=/dev/null)2>&1";
    ArrayList<String> strArr=socketToproxy(cmd);
    for(String s :strArr){
          if (s.contains("copied"))
                 return bandwidthExtract(s);
    }
          return 0;

}

private int bandwidthExtract(String s){
    double bandwidth=0;
    String [] sub=s.split(" ");
    String speed=sub[7];
    if(sub[8].equalsIgnoreCase("MB/s"))
          bandwidth=Double.parseDouble(speed);
    else if(sub[8].equalsIgnoreCase("GB/s"))
          bandwidth=Double.parseDouble(speed)*1000;
    return (int)bandwidth;
}

private void clearCache(){
    String cmd="sync; echo 3 > /proc/sys/vm/drop_caches";
    socketToproxy(cmd);
 }
private void cleanBenchMarkFiles(){
    String cmd="rm -f test";
    socketToproxy(cmd);
```

```
    }

        public void dump()
        {
                String lineSeparator = "----------------------------------------------";
                System.out.println(lineSeparator);
                System.out.println(printConfig());
                System.out.println(lineSeparator);
        }

        public String printConfig(){
                StringBuilder sb=new StringBuilder();
                sb.append("Cloud Name = "+cloudName+System.lineSeparator());
                sb.append("Host ID = " + id+System.lineSeparator());
                sb.append("Host Address = "+addr+System.lineSeparator());
                sb.append("Port No. = "+Integer.toString(port)+System.lineSeparator());
                sb.append("Memory Size = "+Integer.toString(memorySize)+System.lineSeparator());
                sb.append("ReadBandwidth = " + Integer.toString(readBandwidth)+System.lineSeparator());
                sb.append("WriteBandwidth = "+Integer.toString(writeBandwidth)+System.lineSeparator());
                sb.append("CacheBandwidth = " + Integer.toString(cacheBandwidth)+System.lineSeparator());
                sb.append("Epsilon = "+ Double.toString(epsilon)+System.lineSeparator());
                sb.append("Gamma = " + Double.toString(gamma)+System.lineSeparator());
                sb.append("Beta = " +Double.toString(beta)+System.lineSeparator());
                sb.append("BootMin = "+ Double.toString(bootMin)+System.lineSeparator());
                sb.append("a = " + Double.toString(a)+System.lineSeparator());
                sb.append("b = "+Double.toString(b)+System.lineSeparator());
                sb.append("c = "+Double.toString(c)+System.lineSeparator());
                sb.append("TRAIN = "+Integer.toString(trainRound)+System.lineSeparator());
                return sb.toString();
        }

        public boolean writeConfigFile(){
                String file="hosts"+File.separator+id;
                File f=new File(file);
                if(f.exists())
                        return false;
                try {
                        FileWriter fw= new FileWriter(f);
                        fw.write(printConfig());
                        fw.flush();
                        fw.close();
                        return true;
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                        return false;
                }
        }

        public boolean checkParameters(){
                return readBandwidth !=0 && writeBandwidth !=0 && cacheBandwidth !=0 && epsilon !=0&&
                                gamma !=0 && beta !=0 && bootMin !=0 && a !=0 && b !=0 && c!=0;
        }
```

```
        public void addVM(VM vm){
                vmList.add(vm);
        }

        /**
         * Only change the status of the vm to be off, keep the record of the vm
         * @param vmid
         */
        public void deleteVM(String vmid){
                for(int i=0;i<vmList.size();i++){
                        if(vmList.get(i).getId().equals(vmid)){
                                //vmList.remove(i);
                                vmList.get(i).setStatus(VmStat.DELETED);
                                System.out.printf("VM %s has been deleted from the host list!\n", vmid);
                                break;
                        }
                }
                System.out.printf("[ERROR : ]CANNOT find VM %s in the host list!\n ",vmid);
        }

        public int runningVMs(){
                int count=0;
                for(int i=0;i<vmList.size();i++){
                        if(vmList.get(i).getStatus()!=VmStat.DELETED&&vmList.get(i).getStatus()!=VmStat.UNKNOWN)
                                count++;
                }
                return count;
        }

        private void defaultParameters(){
                this.epsilon = 0.0045;
                this.gamma = 0.03;
                this.beta = 0.7;
                this.bootMin = 0.02;
                this.a = 0.1;
                this.b = 64;
                this.c = 0.6;
        }
}

-----------
-----------
-----------


File: project-fermi/plugins-src/vcluster/plugin/balancer/fermibalancer/hostInfoCollector.java
package vcluster.plugin.balancer.fermibalancer;

import vcluster.elements.*;

import java.util.concurrent.CopyOnWriteArrayList;
import java.util.concurrent.locks.Lock;
```

```
import java.util.concurrent.locks.ReentrantLock;

public class hostInfoCollector implements Runnable{

	private static final int UPDATE_INTERVAL=60000;
	private int port=9734;
	private Lock lock;
	private CopyOnWriteArrayList<vcluster.elements.Host> vclusterPrivateHostList;
	private CopyOnWriteArrayList<vcluster.elements.Host> vclusterPublicHostList;
	private CopyOnWriteArrayList<vcluster.elements.Vm> vmList;
	public hostInfoCollector(){
		lock=new ReentrantLock();
		vclusterPrivateHostList=new CopyOnWriteArrayList<vcluster.elements.Host> ();
		vclusterPublicHostList=new CopyOnWriteArrayList<vcluster.elements.Host> ();
	}
	@Override
	public void run() {
		// TODO Auto-generated method stub
		while(!vcluster.Vcluster.terminate){
			vclusterPrivateHostList=new CopyOnWriteArrayList<vcluster.elements.Host> ();
			vclusterPublicHostList=new CopyOnWriteArrayList<vcluster.elements.Host> ();
			vmList=new CopyOnWriteArrayList<vcluster.elements.Vm> ();
			for(vcluster.elements.Cloud c:vcluster.managers.CloudManager.getCloudList().values()){
				c.listVMs();
				c.listHost();
				c.matchVMtoHost();
				boolean privateCloud=true;
				for(Vm vm:c.getVmList().values()){
					vmList.add(vm);
				}
				if(c.getCloudType()!=vcluster.elements.Cloud.CloudType.PRIVATE)
					privateCloud=false;
				for(vcluster.elements.Host h:c.getHostList().values()){
					if(h.getStat()!=vcluster.elements.Host.HostStat.ON)
						continue;
					if(privateCloud)
						vclusterPrivateHostList.add(h);
					else
						vclusterPublicHostList.add(h);
				}
			}
			try{
				/*
				 *  1. add new host
				 *  2. delete old one
				 *  3. update the real system info
				 *  4. update VM info
				 */

				if(lock.tryLock()){
					long currentTime=System.currentTimeMillis()/1000*1000;
					if(!Algorithm.getPrivateHostList().isEmpty()){
						for(Host h:Algorithm.getPrivateHostList()){
```

```
						int index=findHost(h,vclusterPrivateHostList);
						if(index==-1){
							Algorithm.getPrivateHostList().remove(h);
							}
						else{

	h.setSysStat(updateSysStat(h,vclusterPrivateHostList.get(index).getCPUUtil(),vclusterPrivateHostList.get(index).getIOUt
il(),currentTime));
							h.setVmList(new CopyOnWriteArrayList<VM>());
							vclusterPrivateHostList.remove(index);
							}
						}
				}
				if(!vclusterPrivateHostList.isEmpty()){
					for(int i=0;i<vclusterPrivateHostList.size();i++){
						Host newHost=new
Host(vclusterPrivateHostList.get(i).getCloudName(),vclusterPrivateHostList.get(i).getId(),vclusterPrivateHostList.get(i).getNam
e(),port,

	(int)vclusterPrivateHostList.get(i).getTMEM(),vclusterPrivateHostList.get(i).getTCPU()/100);
						Algorithm.getPrivateHostList().add(newHost);
					}
				}
				for(Host h:Algorithm.getPublicHostList()){
					h.setVmList(new CopyOnWriteArrayList<VM>());
				}
				updateVMs();
				//Algorithm.dump();
				}
			}catch(Exception e){
				e.printStackTrace();
			}finally{
				Algorithm.ready=true;
				lock.unlock();
			}
			try {
				Thread.sleep(UPDATE_INTERVAL);
			} catch (InterruptedException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
		}

	}
    /**
     * Find a Host in private host list, if find, return the index of position in private host list. If not find, return -1
     * @param h : vcluster host
     * @see vcluster.elements.Host
     * @param privateCloud :if the host is in a private cloud
     * @return
     */
	private static int findHost(vcluster.elements.Host h, boolean privateCloud){
		int find=-1;
```

```
        CopyOnWriteArrayList<Host> hostList=null;
        if(privateCloud)
                hostList=Algorithm.getPrivateHostList();
        else
                hostList=Algorithm.getPublicHostList();
        for(int i=0;i<hostList.size();i++){
                if(h.getName().equalsIgnoreCase(hostList.get(i).getAddress()))
                {
                        find=i;
                        break;
                }
        }
        return find;
    }
    /**
     * Find a Host in private host list, if find, return the index of position in private host list. If not find, return -1
     * @param h : Load balancer Host type
     * @param hostList : list of host
     * @return
     */
    private int findHost(Host h, CopyOnWriteArrayList<vcluster.elements.Host> hostList){
        int find=-1;
        if(hostList.isEmpty())
                return -1;
        for(int i=0;i<hostList.size();i++)
        {
                if(hostList.get(i).getName().equalsIgnoreCase(h.getAddress()))
                {
                        find=i;break;
                }
        }
        return find;
    }
    public static int findHost(String hostName, CopyOnWriteArrayList<Host> hostList){
        int index=-1;
        for(int i=0;i<hostList.size();i++){
                if(Cloud.hostNameMatch(hostName, hostList.get(i).getAddress()))
                        return index=i;
        }
        return index;
    }
    public static Host findHost(String hostName){
        Host h=null;
        for(Host hTmp:Algorithm.getPrivateHostList()){
                if(Cloud.hostNameMatch(hostName, hTmp.getAddress()))
                        return hTmp;
        }
        for(Host hTmp:Algorithm.getPublicHostList()){
                if(Cloud.hostNameMatch(hostName, hTmp.getAddress()))
                        return hTmp;
        }
        return h;
    }
```

```
    private CopyOnWriteArrayList<SystemStat> updateSysStat(Host h, double cpu, double io, long time){
        CopyOnWriteArrayList<SystemStat> tempStat=h.getSysStat();
        int size=Host.MAX_TIME_FRAME;
        if(tempStat.isEmpty()||time>tempStat.get(size-1).time||time<tempStat.get(0).time)
        {
                tempStat=new CopyOnWriteArrayList<SystemStat>();
                for(int i=0;i<size;i++){
                        SystemStat s=new SystemStat();
                        s.cpuUtil=cpu;
                        s.ioUtil=io;
                        s.time=(time/1000+i)*1000;
                        tempStat.add(s);
                }
                return tempStat;
        }
        //else if(time<=tempStat.get(size-1).time&&time>=tempStat.get(0).time){
        else{
                int index=0;
                for(int i=0;i<size;i++){
                        if(tempStat.get(i).time==time){
                                index=i;
                                break;
                        }
                }
                double diffCpu=cpu-tempStat.get(index).cpuUtil;
                double diffIO=io-tempStat.get(index).ioUtil;
                CopyOnWriteArrayList<SystemStat> newStat=new CopyOnWriteArrayList<SystemStat>();
                for(int j=0;j<size;j++){
                        SystemStat s=new SystemStat();
                        s.time=(time/1000+j)*1000;
                        if(j+index<size){
                                s.cpuUtil=Math.min(1, Math.max(tempStat.get(index).cpuUtil+diffCpu, 0));
                                s.ioUtil=Math.min(1, Math.max(tempStat.get(index).ioUtil+diffIO, 0));
                        }
                        else
                        {
                                s.cpuUtil=cpu;
                                s.ioUtil=io;
                        }
                        newStat.add(s);
                }
                return newStat;
        }
    }

  private boolean updateVMs(){
    boolean success=true;
    //CopyOnWriteArrayList<Vm> vmList= new CopyOnWriteArrayList<Vm>(vcluster.managers.VmManager.getVmList().values());
     if(Algorithm.getVmList().size()+Algorithm.getVmWaitQueue().size()!=vmList.size()){
        String line="[Warning:] VM numbers are not matching, some manual operations are made!";
        FermiBalancer.writeLogFile(line);
        System.out.println(line+System.lineSeparator());
     }
```

```
        for(VM v:Algorithm.getVmList()){
            int index=findVM(v,vmList);
            if(index==-1){
                String line="[Warning: ]VM "+v.getId()+" does not exist!";
                FermiBalancer.writeLogFile(line);
                Algorithm.getVmList().remove(v);
            }else{
                if(!Cloud.hostNameMatch(v.getHostName(),vmList.get(index).getHostname())){
                    String line="[Warning: ] VM "+v.getId()+" that suppose to be launched on Host "+v.getHostName()+
                                " is being launched on Host "+vmList.get(index).getHostname();
                    FermiBalancer.writeLogFile(line);
                    v.setRealHostName(vmList.get(index).getHostname());
                    vmList.remove(index);
                }
                else{
                    v.setRealHostName(v.getHostName());
                    vmList.remove(index);
                }
                int privatehostindex=findHost(v.getRealHostName(),Algorithm.getPrivateHostList());
                int publichostindex=findHost(v.getRealHostName(),Algorithm.getPublicHostList());
                if(privatehostindex==-1&&publichostindex==-1){
                    String line="[Error: ] Host "+v.getRealHostName()+" does not exist in the list!";
                    FermiBalancer.writeLogFile(line);
                    success=false;
                }else if(publichostindex==-1){
                Algorithm.getPrivateHostList().get(privatehostindex).addVM(v);}
                else{
                    Algorithm.getPublicHostList().get(publichostindex).addVM(v);
                }
            }
        }
        for(VM v:Algorithm.getVmWaitQueue()){
            int index=findVM(v,vmList);
            if(index==-1){
                String line="[Warning: ]VM "+v.getId()+" does not exist!";
                FermiBalancer.writeLogFile(line);
                Algorithm.getVmList().remove(v);
                success=false;
            }else{
                String temp1=v.getHostName();
                String temp2=vmList.get(index).getHostname();
                if(!Cloud.hostNameMatch(temp1, temp2)){
                    String line="[Warning: ] VM "+v.getId()+" that suppose to be launched on Host "+v.getHostName()+
                                " is being launched on Host "+vmList.get(index).getHostname();
                    FermiBalancer.writeLogFile(line);
                    v.setRealHostName(vmList.get(index).getHostname());
                    vmList.remove(index);
                }
                else{
                    v.setRealHostName(v.getHostName());
                    vmList.remove(index);
                }
```

```
                int privatehostindex=findHost(v.getRealHostName(),Algorithm.getPrivateHostList());
                int publichostindex=findHost(v.getRealHostName(),Algorithm.getPublicHostList());
                if(privatehostindex==-1&&publichostindex==-1){
                    String line="[Error: ] Host "+v.getRealHostName()+" does not exist in the list!";
                    FermiBalancer.writeLogFile(line);
                    success=false;
                }else if(publichostindex==-1){
                Algorithm.getPrivateHostList().get(privatehostindex).addVM(v);}
                else{
                    Algorithm.getPublicHostList().get(publichostindex).addVM(v);
                }
            }
        }
        if(!vmList.isEmpty()){
            for(int i=0;i<vmList.size();i++){
                VM v=new VM();
                v.setId(vmList.get(i).getId());
                v.setHostName(vmList.get(i).getHostname());
                v.setDnsName(vmList.get(i).getDNSName());
                v.setRealHostName(vmList.get(i).getHostname());
                v.setRealBootTime(-1);
                v.setBootTime(-1);
                Algorithm.getVmList().add(v);
                int privatehostindex=findHost(v.getRealHostName(),Algorithm.getPrivateHostList());
                int publichostindex=findHost(v.getRealHostName(),Algorithm.getPublicHostList());
                if(privatehostindex==-1&&publichostindex==-1){
                    String line="[Error: ] Host "+v.getRealHostName()+" does not exist in the list!";
                    FermiBalancer.writeLogFile(line);
                    success=false;
                }else if(publichostindex==-1){
                Algorithm.getPrivateHostList().get(privatehostindex).addVM(v);}
                else{
                    Algorithm.getPublicHostList().get(publichostindex).addVM(v);
                }
            }
        }
        return success;
    }

    private int findVM(VM v, CopyOnWriteArrayList<Vm> vmList){
       int index=-1;
       for(int i=0;i<vmList.size();i++){
            if(vmList.get(i).getId().equalsIgnoreCase(v.getId())&&
                    vmList.get(i).getDNSName().equalsIgnoreCase(v.getDnsName()))
                return index=i;
       }
       return index;
    }
}

-----------
-----------
-----------
```

File: project-fermi/plugins-src/vcluster/plugin/balancer/fermibalancer/InfoCollector.java

```
package vcluster.plugin.balancer.fermibalancer;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
public class InfoCollector implements Runnable{

	private int port;
	public InfoCollector(int portNo){
		port=portNo;}

	public void run(){
		try {
	    ServerSocket serverSocket = new ServerSocket(port);
	    System.out.println("Waiting for clients to connect...");
	    while (!vcluster.Vcluster.terminate) {
		Socket clientSocket = serverSocket.accept();
		Thread clientThread = new Thread(new clientInfoCollector(clientSocket));
		clientThread.start();
		//clientInfoCollector client=new clientInfoCollector(clientSocket);
		//client.run();
	    }
	    serverSocket.close();
	 } catch (IOException e) {
	    System.err.println("Unable to process client request");
	    e.printStackTrace();
	 }
    }


}
```


File: project-fermi/plugins-src/vcluster/plugin/balancer/fermibalancer/SystemStat.java

```
package vcluster.plugin.balancer.fermibalancer;

public class SystemStat {
	public long time;
	public double cpuUtil;
	public double ioUtil;

}
```


File: project-fermi/plugins-src/vcluster/plugin/balancer/fermibalancer/test.java

```
package vcluster.plugin.balancer.fermibalancer;

import java.util.ArrayList;
import java.util.Random;


public class test {
	public static void main(String[] args) throws Exception {
		//testMASE();
		//Thread serverThread = new Thread(new InfoCollector(9738));
	 //serverThread.start();
	 //InfoCollector infoc=new InfoCollector(9738);
	 //infoc.run();
		int id=14;
		int hostNo=303;
		for(int i=0;i<10;i++){
			id=id+1;
			hostNo=hostNo+1;
			String hostname="fgitb"+hostNo+".fnal.gov";
			Host h=new Host("fermicloudpp",Integer.toString(id),hostname,9734,16331456,8);
			Algorithm.getPrivateHostList().add(h);
			h.dump();
		}

		//Host h2=new Host("Amazon","1","aws.amazon.com",9734,0xffffffff,0xffffffff);
		//Algorithm.getPublicHostList().add(h2);
		long currentTime=System.currentTimeMillis()/1000*1000;
		Random r=new Random();
		int t=0;
		long time=currentTime+(long)(t*1000);
		for(int i=0;i<70;i++){
			//System.out.printf("Last Launch: %d CurrentTime: %d Slack: %d seconds\n", currentTime, time, t);
			//currentTime=time;
			//Algorithm.estimateDeployTest(currentTime);
			//Algorithm.dump();
			//t=r.nextInt(30);
			//time=currentTime+(long)(t*1000);

			//long currentTime=System.currentTimeMillis();
		VM v=new VM();
		v.setStartTime(currentTime);
		v.setWaitTime(0);
		Algorithm.getVmWaitQueue().add(v);
		Host h=Algorithm.BestFitHost(v,false);
	    //Host h=Algorithm.estimateDeploy(v,false);
		//String str=CloudManager.getCurrentCloud().createVMonHost(1, h.getId());
	   // String str=CloudManager.getCurrentCloud().createVM(1, h.getAddress());
		//String [] sub=str.split("\\s+");
```

```
            v.setId(Integer.toString(i+1));
            v.setDnsName(Integer.toString(i+1));
            //v.setStartTime(time);
            v.setHostId(h.getId());
            v.setHostName(h.getAddress());
            //h.addVM(v);

//            int op=r.nextInt(100);
//            int real=v.getBootTime();
//            if(op<50)
//            real=real-r.nextInt(30);
//            else
//            real=real+r.nextInt(100);
//                  v.setRealBootTime(real);
//
//                  h.trainning(true);
              }
              Algorithm.dump();
              System.out.printf("Average Launch Time: %.2f seconds\n", Algorithm.aveLaunchTime());

       }
       public static void testMASE(){
              ArrayList<Double> actual=new ArrayList<Double>();
              ArrayList<Double> predict=new ArrayList<Double>();
              Random r=new Random(System.currentTimeMillis());
              for(int i=0;i<100;i++){
                     double a=(double)r.nextInt(100);
                     //double f=(double)r.nextInt(100);
                     double f=a+5;
                     String sa=String.format("%-3.3f", a);
                     String sf=String.format("%-3.3f", f);
                     System.out.println(sa+"  "+sf+System.lineSeparator());
                     actual.add(a);
                     predict.add(f);
              }
              double sum=0;
              for(int i=1;i<actual.size();i++){
                     double e=(actual.get(i)-predict.get(i));
                     double y=0;
                     for(int j=i;j>0;j--){
                            y=y+Math.abs(predict.get(j)-predict.get(j-1));
                     }
                     double temp=1.00/(100.00-1.00)*y;
                     sum=sum+Math.abs(e/temp);
              }
              System.out.println(Double.toString(sum/100));

       }
}

-----------
-----------
-----------
```

```
File: project-fermi/plugins-src/vcluster/plugin/balancer/fermibalancer/TimeSet.java
package vcluster.plugin.balancer.fermibalancer;

public class TimeSet {
       public int start;
       public int finish;
}

-----------
-----------
-----------


File: project-fermi/plugins-src/vcluster/plugin/balancer/fermibalancer/VM.java
package vcluster.plugin.balancer.fermibalancer;



public class VM {
       public static enum VmStat{RUNNING,PENDING,PROLOG,UNKNOWN,DELETED,SCHEDULE}
       private String id;
       private String hostName;
       private VmStat status;
       private long startTime;
       private long actualStartTime;
       private int bootTime;
       private int realBootTime;
       private String realHostName;
       private String hostId;
       private String dnsName;
       private int waitTime;
       private VmLaunchOverheadModel overhead;

       public VM(){
              setStartTime(System.currentTimeMillis()/1000*1000);
              actualStartTime=startTime;
              waitTime=0;
              setStatus(VmStat.PENDING);
              setOverhead(new VmLaunchOverheadModel());
       }

       public String getId() {
              return id;
       }

       public void setId(String id) {
              this.id = id;
       }

       public VmStat getStatus() {
              return status;
```

```
	}

	public void setStatus(VmStat status) {
		this.status = status;
	}

	public long getStartTime() {
		return startTime;
	}

	public void setStartTime(long startTime) {
		this.startTime = startTime;
	}

	public int getBootTime() {
		return bootTime;
	}

	public void setBootTime(int bootTime) {
		this.bootTime = bootTime;
	}

	public VmLaunchOverheadModel getOverhead() {
		return overhead;
	}

	public void setOverhead(VmLaunchOverheadModel overhead) {
		this.overhead = overhead;
	}

	public String getHostId() {
		return hostId;
	}

	public void setHostId(String hostId) {
		this.hostId = hostId;
	}

	public String getHostName() {
		return hostName;
	}

	public void setHostName(String hostName) {
		this.hostName = hostName;
	}

	public int getRealBootTime() {
		return realBootTime;
	}

	public void setRealBootTime(int realBootTime) {
		this.realBootTime = realBootTime;
	}
```

```
	public String getRealHostName() {
		return realHostName;
	}

	public void setRealHostName(String realHostName) {
		this.realHostName = realHostName;
	}

	public String getDnsName() {
		return dnsName;
	}

	public void setDnsName(String dnsName) {
		this.dnsName = dnsName;
	}

	public void setWaitTime(int t){
		waitTime=t;
	}
	public int getWaitTime(){
		return waitTime;
	}

	public long getActualStartTime() {
		return actualStartTime;
	}

	public void setActualStartTime(long actualStartTime) {
		this.actualStartTime = actualStartTime;
	}
}

-----------
-----------
-----------


File: project-fermi/plugins-src/vcluster/plugin/balancer/fermibalancer/vmInfoColloctor.java
package vcluster.plugin.balancer.fermibalancer;

public class vmInfoColloctor implements Runnable{

	@Override
	public void run() {
		// TODO Auto-generated method stub

	}

}

-----------
-----------
```

```
-----------


File: project-fermi/plugins-src/vcluster/plugin/balancer/fermibalancer/VmLauncher.java
package vcluster.plugin.balancer.fermibalancer;

import java.util.Date;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

import vcluster.Vcluster;
import vcluster.managers.CloudManager;
import vcluster.plugin.balancer.fermibalancer.VM.VmStat;

//TODO: TO BE TESTED
public class VmLauncher implements Runnable{

	@Override
	public void run() {
		// TODO Auto-generated method stub
		Lock lock=new ReentrantLock();
		while(!Vcluster.terminate){
			int sleepTime=0;
			boolean launch=false;
			Host h=null;
			VM v=null;
			try{
				if(lock.tryLock()){
					if(Algorithm.getVmWaitQueue().isEmpty()){
						sleepTime=1000;
						launch=false;
					}
					else
					{
						for(int i=0;i<Algorithm.getVmWaitQueue().size();i++){
							if(Algorithm.getVmWaitQueue().get(i).getStatus()==VmStat.PENDING){
								v=Algorithm.getVmWaitQueue().get(i);
								long currentTime=System.currentTimeMillis()/1000*1000;
								long diff=v.getStartTime()-currentTime;

								sleepTime=Math.max(1,
Algorithm.getVmWaitQueue().get(i).getWaitTime()+(int)diff);
								launch=true;
								break;
							}
						}
					}
					}
				}
			catch(Exception e){
				e.printStackTrace();
			}finally{
				lock.unlock();
```

```
}

try{
	Thread.sleep(sleepTime);
}catch(Exception e){
	e.printStackTrace();
}

if(launch&&v!=null){
	if(!v.getHostName().equalsIgnoreCase("-1")){
	int privateIndex=-1;
	int publicIndex=-1;
	privateIndex=hostInfoCollector.findHost(v.getHostName(),Algorithm.getPrivateHostList());
	if(privateIndex==-1){
		publicIndex=hostInfoCollector.findHost(v.getHostName(),Algorithm.getPublicHostList());
	    if(publicIndex==-1){
		System.out.println("[ERROR: ] No host can be found for launching process!");
		 continue;
	    } else
		h=Algorithm.getPublicHostList().get(publicIndex);
	}else
		h=Algorithm.getPrivateHostList().get(privateIndex);
	if(h!=null){
		Date d=new Date(System.currentTimeMillis());
		System.out.println(d.toString()+" Launch a VM!");
		String str=CloudManager.getCurrentCloud().createVMonHost(1, h.getId());

		String [] sub=str.split("\\s+");

		v.setId(sub[1]);
		v.setDnsName(sub[2]);
		v.setHostId(h.getId());
		v.setHostName(h.getAddress());
		v.setStatus(VmStat.PROLOG);
		}
	}else
	{
		Date d=new Date(System.currentTimeMillis());
		System.out.println(d.toString()+" Launch a VM!");
		String str=CloudManager.getCurrentCloud().createVM(1, "-1");

		String [] sub=str.split("\\s+");

		v.setId(sub[1]);
		v.setDnsName(sub[2]);
		//v.setHostId(h.getId());
		//v.setHostName(h.getAddress());
		v.setStatus(VmStat.PROLOG);
	}
	}
}
		}
	}
```

```
-----------
-----------
-----------


File: project-fermi/plugins-src/vcluster/plugin/balancer/fermibalancer/VmLaunchOverheadModel.java
package vcluster.plugin.balancer.fermibalancer;

import java.util.ArrayList;

/*
 * In the model class, the start time is always 0,
 * for each VM, the time line needs to be converted to its' own time line
 */
public class VmLaunchOverheadModel {

	/*
	 * All the parameters to calculate the VM launching overhead
	 */
	private double readBandwidth;
	private double writeBandwidth;
	private double cacheBandwidth;
	private double readSpeed;
	private double writeSpeed;
	private double cacheSpeed;
	private double imageSize;
	private double memorySize;
	private boolean cached;

	private double epsilon;
	private double gamma;
	private double beta;
	private double bootMin;
	private double a;
	private double c;
	private double b;

	private int MAX_TIME_FRAME;
	private ArrayList<Double> CpuUtil;
	private ArrayList<Double> IOUtil;

	private int startTime;

	/**
	 * Default constructor.
	 * Assume Fermi machine specifications: 100m/s read, 400m/s write, 1.2G/s cache, 16G memory
	 */
	public VmLaunchOverheadModel(){
		readBandwidth=100;
		writeBandwidth=400;
		cacheBandwidth=1200;
```

```
		readSpeed=100;
		writeSpeed=100;
		cacheSpeed=1200;
		memorySize=16000;
		epsilon=0.004;
		gamma=0.03;
		beta=0.7;
		bootMin=0.02;
		a=0.1;
		c=0.6;
		b=64;
		cached=false;
		initialize();
	}

	/**
	 * Constructor
	 * @param readB
	 * @param writeB
	 * @param cacheB
	 * @param readS
	 * @param writeS
	 * @param cacheS
	 * @param mem_size
	 * @param imageSize
	 * @param epsilon
	 * @param gamma
	 * @param beta
	 * @param bootMin
	 * @param a
	 * @param b
	 * @param c
	 * @param cache
	 */
	public VmLaunchOverheadModel(int readB, int writeB, int cacheB, int readS, int writeS, int cacheS, int mem_size, int
imageSize,
		double epsilon, double gamma, double beta, double bootMin, double a, double b, double c, boolean cache){
		this.readBandwidth=readB;
		this.writeBandwidth=writeB;
		this.cacheBandwidth=cacheB;
		this.readSpeed=readS;
		this.writeSpeed=writeS;
		this.cacheSpeed=cacheS;
		this.memorySize=mem_size;
		this.imageSize=imageSize;
		this.epsilon=epsilon;
		this.gamma=gamma;
		this.beta=beta;
		this.bootMin=bootMin;
		this.a=a;
		this.b=b;
		this.c=c;
		this.cached=cache;
```

```
        initialize();
}

public int getReadBandwidth(){
        return (int)readBandwidth;
}
public void setReadBandwidth(int value){
        readBandwidth=value;
}

public int getWriteBandwidth(){
        return (int)writeBandwidth;
}
public void setWriteBandwidth(int value){
        writeBandwidth=value;
}

public int getCacheBandwidth(){
        return (int)cacheBandwidth;
}
public void setCacheBandwidth(int value){
        cacheBandwidth=value;
}

public int getReadSpeed(){
        return (int)readSpeed;
}
public void setReadSpeed(int value){
        readSpeed=value;
}

public int getWriteSpeed(){
        return (int)writeSpeed;
}
public void setWriteSpeed(int value){
        writeSpeed=value;
}

public int getCacheSpeed(){
        return (int)cacheSpeed;
}
public void setCacheSpeed(int value){
        cacheSpeed=value;
}

public boolean ifCached(){
        return cached;
}
public void setCached(boolean value){
        cached=value;
}

public double getEpsilon(){
```

```
        return epsilon;
}
public void setEpsilon(double value){
        epsilon=value;
}

public double getGamma(){
        return gamma;
}
public void setGamma(double value){
        gamma=value;
}

public double getBeta(){
        return beta;
}
public void setBeta(double value){
        beta=value;
}

public double getBootMin(){
        return bootMin;
}
public void setBootMin(double value){
        bootMin=value;
}

public int getMaxTimeFrame(){
        return MAX_TIME_FRAME;
}
public void setMaxTimeFrame(int value){
        MAX_TIME_FRAME=value;
}

public ArrayList<Double> getCpuU(){
        return CpuUtil;
}
public void setCpuU(ArrayList<Double> value){
        CpuUtil=value;
}

public ArrayList<Double> getIOU(){
        return IOUtil;
}
public void setIOU(ArrayList<Double> value){
        IOUtil=value;
}
public int getImageSize() {
        return (int)imageSize;
}
public void setImageSize(int imageSize) {
        this.imageSize = imageSize;
}
```

```
        public int getMemorySize() {
                return (int)memorySize;
        }
        public void setMemorySize(int memorySize) {
                this.memorySize = memorySize;
        }
        public int getStartTime(){
                return startTime;
        }
        public void setStartTime(int value){
                startTime=value;
        }
        public double getA() {
                return a;
        }
        public void setA(double a) {
                this.a = a;
        }
        public double getC() {
                return c;
        }
        public void setC(double c) {
                this.c = c;
        }
        public double getB() {
                return b;
        }
        public void setB(double b) {
                this.b = b;
        }

        private ArrayList<TimeSet> writeTimeSet;
    private int TStartWrite;
    private int TFinishWrite;
    private int t_trans;

    private int tStartWrite()
    {
        if (cached)
            return (int)Math.round(startTime+Math.min(30,Math.min(imageSize,memorySize/10)/cacheSpeed));
        else
            return (int)Math.round(startTime+Math.min(30,Math.min(imageSize,memorySize/10)/readSpeed));

    }
    private int tFinishWrite()
    {
       return findWriteTimeSet();
    }

    public int transTime()
    {
        if (cached)
        {
```

```
            double t1 = Math.min(imageSize / cacheSpeed, memorySize / 10 / cacheSpeed);
            if (imageSize <= (memorySize / 10))
                return (int)t1;
            else
            {
                double remain = imageSize - memorySize / 10;
                double t2 = remain / (cacheSpeed - writeSpeed);
                return (int)(t1 + t2);
            }
        }
        else
            return (int)Math.round(imageSize / readSpeed);
    }

    public double util_trans(long t)
    {
        double ftime=t_trans+startTime;
       // double temp=1/(1+Math.pow(Math.E,(-0.5*ftime*(t-startTime))))-1/(1+Math.pow(Math.E,(-0.5*ftime*(t-ftime))));
        if (t >= startTime && t <= ftime)
        {
            if (cached)
                return cacheSpeed / cacheBandwidth;
            else
                return readSpeed / (readBandwidth / beta);
        }
        else
            return 0;
    }

    public int findWriteTimeSet()
    {
        writeTimeSet = new ArrayList<TimeSet>();
        double t1;
        if(!cached)
         t1 = Math.min(Math.min(30,Math.min(imageSize,memorySize/10)/readSpeed)*readSpeed/Math.max(1, (writeSpeed-
readSpeed)),imageSize/writeSpeed);
        else
            t1 = Math.ceil((double)imageSize / (double)writeSpeed);
        TimeSet s1 = new TimeSet();
        s1.start = tStartWrite();
        s1.finish =  s1.start + (int)t1;
        writeTimeSet.add(s1);
        if (cached)
            return s1.finish;
        else {
            double remain = imageSize - t1 * writeSpeed;
            double wt_sleep = 5 * (double)readSpeed /(double)Math.max(1, (writeSpeed - readSpeed));
            double N = Math.ceil(remain/writeSpeed/wt_sleep);
            double lasttime = s1.finish;
            for(int i=1;i<=N;i++)
            {
                TimeSet stmp = new TimeSet();
                if (i != N)
```

```
                {
                    stmp.start = (int) (lasttime + 5);
                    stmp.finish = (int) (stmp.start + wt_sleep);
                    writeTimeSet.add(stmp);
                    lasttime = stmp.finish;
                }
                else
                {
                    stmp.start = (int)(lasttime + 5);
                    stmp.finish = (int) (stmp.start + (remain/writeSpeed)%wt_sleep);
                    writeTimeSet.add(stmp);
                    lasttime = stmp.finish;
                }
            }
            return (int)lasttime;
        }
    }
    public boolean inTimeSet(long t)
    {
        for (int i = 0; i < writeTimeSet.size(); i++)
        {
            if (t <= writeTimeSet.get(i).finish && t >= writeTimeSet.get(i).start)
                return true;
        }
        return false;

    }
    public double IO(int t)
    {
        if (inTimeSet(t))
            return writeSpeed / writeBandwidth* (1 / (1 + Math.pow(Math.E, (-0.5 * TFinishWrite * (t - TStartWrite)))) - 1 / (1
+ Math.pow(Math.E, (-0.5 * TFinishWrite * (t - TFinishWrite)))));
        else
            return 0;
    }

    public double U_IO(int t, int n_cores)
    {
        return IO(t) / n_cores;
    }

    public double U_Boot(long t, double io)
    {

        //return Math.Max(c*Math.Pow(Math.E,-Data.gamma*(1-io)*(t-t_trans)),0.001);
        //double tmp = 1 + Data.boot_min - io;
        double t_mid = Math.ceil(c / a)+startTime+t_trans;
        if(t>=t_mid)
        {
            //double tmp = bootMin * (1 / (gamma + io));
            double time = (t - t_mid);
            return c * Math.pow(Math.E, -time * gamma * (bootMin*(1+io) + b / (time + b)));
        }
```

```
        else
            return (t - startTime - t_trans) * a;
    }

    public void initialize()
    {
       //readSpeed=readBandwidth;
       //writeSpeed=writeBandwidth;
       //cacheSpeed=cacheBandwidth;
       //cached=false;
       startTime=0;
       CpuUtil=new ArrayList<Double>();
       IOUtil=new ArrayList<Double> ();
       writeTimeSet=new ArrayList<TimeSet> ();
        t_trans = transTime();
        TStartWrite = tStartWrite();
        TFinishWrite = tFinishWrite();
    }

    public int bootTime(){
       int time=t_trans;
       try{
              if(CpuUtil.size()<=t_trans)
                     throw new Exception("CPU utilization is not estimated!");
              double t_mid = Math.ceil(c / a)+startTime+t_trans;
              for(int i=(int)t_mid;i<CpuUtil.size();i++)
              {
                     double diff=CpuUtil.get(i)-CpuUtil.get(i+1);
                     if(diff>=0&&diff<epsilon)
                     {
                            time=i;
                            if((IOUtil.get(time-1)-IOUtil.get(time))>=0.05 )
                                   continue;
                            else
                                   break;
                     }
              }

       }
       catch (Exception e){
              System.out.println(e.getMessage());
       }
       return time;
    }

       public String dump(){
              StringBuilder sb=new StringBuilder();
              String tTime=String.format("%-5s", "TIME");
              String tCpu=String.format("%-8s", "CPU");
              String tIO=String.format("%-8s", "IO");
              sb.append("---------------------------------------------"+System.lineSeparator());
              sb.append(tTime+tCpu+tIO+System.lineSeparator());
              sb.append("---------------------------------------------"+System.lineSeparator());
```

```
		for(int i=0;i<300;i++){
			String hTime=String.format("%-5s", Integer.toString(i));
			String temp=String.format("%-3.3f",CpuUtil.get(i)*100);
			String hCpu=String.format("%-8s", temp);
			temp=String.format("%-3.3f", IOUtil.get(i)*100);
			String hIO=String.format("%-8s", temp);
			sb.append(hTime+hCpu+hIO+System.lineSeparator());
		}
		sb.append("--------------------------------------------"+System.lineSeparator());
		return sb.toString();
	}
}

-----------
-----------
-----------


File: project-fermi/plugins-src/vcluster/plugin/ec2/Cloud.java
package vcluster.plugin.ec2;

import java.util.List;
import java.util.StringTokenizer;
/**
 *A class represents a cloud, the instance of this this class involves the information that register to a ec2 interface.
 *
 * Hao: Add three properties to the EC2 cloud: TagKey, TagValue, SecurityGroup so that VM instances can be tagged.
 */
public class Cloud {

	protected Cloud() {
		cloudType = CloudType.NOT_DEFINED;
		endPoint = null;
		maxVMs = -1;
		currentVMs = 0;
		accessKey = null;
		secretKey = null;
		keyName = null;
		instanceType = null;
		imageName = null;
		tagKey=null;
		tagValue=null;
		securityGroup=null;

	}
	/**
	 *Create a instance by a configuration list.
	 */
	Cloud(List<String> configurations){
		for(String aLine : configurations){

			StringTokenizer st = new StringTokenizer(aLine, "=");
```

```
			if (!st.hasMoreTokens()) return;

			/* get a keyword */
			String aKey = st.nextToken().trim();

			/* get a value */
			if (!st.hasMoreTokens()) return;

			String aValue = st.nextToken().trim();

			if (aKey.equalsIgnoreCase("type"))
				setCloudType(aValue);
			else if (aKey.equalsIgnoreCase("endpoint"))
				setEndPoint(aValue);
			else if (aKey.equalsIgnoreCase("accesskey"))
				setAccessKey(aValue);
			else if (aKey.equalsIgnoreCase("secretkey"))
				setSecretKey(aValue);
			else if (aKey.equalsIgnoreCase("instancetype"))
				setInstanceType(aValue);
			else if (aKey.equalsIgnoreCase("keyname"))
				setKeyName(aValue);
			else if (aKey.equalsIgnoreCase("image"))
				setImageName(aValue);
			else if (aKey.equalsIgnoreCase("max"))
				setMaxVMs(Integer.parseInt(aValue));
			else if (aKey.equalsIgnoreCase("version"))
				setVersion(aValue);
			else if (aKey.equalsIgnoreCase("signatureversion"))
				setSignatureVersion(aValue);
			else if (aKey.equalsIgnoreCase("signaturemethod"))
				setSignatureMethod(aValue);
			else if (aKey.equalsIgnoreCase("tagKey"))
				setTagKey(aValue);
			else if (aKey.equalsIgnoreCase("tagValue"))
				setTagValue(aValue);
			else if (aKey.equalsIgnoreCase("securityGroup"))
				setSecurityGroup(aValue);
		}

	}

	public int getCurrentVMs() {
		return currentVMs;
	}

	public void setCurrentVMs(int vms) {
		currentVMs = vms;
	}

	protected void incCurrentVMs(int vms) {
		currentVMs += vms;
	}
```

```
public int getMaxVMs() {
	return maxVMs;
}

protected void setMaxVMs(int vms) {
	maxVMs = vms;
}

public String getEndPoint() {
	return endPoint;
}

protected void setEndPoint(String point) {
	endPoint = point;

	point = point.replaceAll("https://", "");
	point = point.replaceAll("http://", "");
	point = point.replaceAll("/", "");
	shortEndPoint = point.trim();

}

public String getShortEndPoint() {
	return shortEndPoint;
}

public CloudType getCloudType() {
	return cloudType;
}

protected void setCloudType(CloudType type) {
	cloudType = type;
}

protected void setCloudType(String type) {
	if (type.equalsIgnoreCase("private"))
		cloudType = CloudType.PRIVATE;
	else if (type.equalsIgnoreCase("public"))
		cloudType = CloudType.PUBLIC;
	else {
		cloudType = CloudType.NOT_DEFINED;
	}

}

public String getAccessKey() {
	return accessKey;
}

protected void setAccessKey (String key) {
	accessKey = key;
}
```

```
public String getSecretKey() {
	return secretKey;
}

protected void setSecretKey(String key) {
	secretKey = key;
}

public String getInstaceType() {
	return instanceType;
}

protected void setInstanceType(String type) {
	instanceType = type;
}

public String getKeyName() {
	return keyName;
}

protected void setKeyName(String name) {
	keyName = name;
}

protected void setImageName(String name) {
	imageName = name;
}

public String getImageName() {
	return imageName;
}

protected void setVersion(String v) {
	this.version = v;
}

public String getVersion() {
	return version;
}

protected void setSignatureVersion(String v) {
	signatureVersion = v;
}

public String getSignatureVersion() {
	return signatureVersion;
}

protected void setSignatureMethod(String method) {
	signatureMethod = method;
}
```

```
        public String getSignatureMethod() {
                return signatureMethod;
        }

        public String getTagKey(){
                return tagKey;
        }

        protected void setTagKey(String key){
                tagKey=key;
        }

        public String getTagValue(){
                return tagValue;
        }

        protected void setTagValue(String value){
                tagValue=value;
        }

        public String getSecurityGroup(){
                return securityGroup;
        }

        protected void setSecurityGroup(String value){
                securityGroup=value;
        }



        public String stringCloudType() {

                switch(cloudType) {
                case PRIVATE: return "PRIVATE";
                case PUBLIC: return "PUBLIC";
                case NOT_DEFINED: return "NOT_DEFINED";
                }
                return "NOT_DEFINED";
        }

        public void dump() {

                System.out.printf("\tcloud type:\t%s\n", cloudType);
                System.out.printf("\tend point:\t%s\n", endPoint);
                System.out.printf("\tshort point:\t%s\n", shortEndPoint);
                System.out.printf("\tmax vms:\t%d\n", maxVMs);
                System.out.printf("\tcur vms:\t%d\n", currentVMs);
                System.out.printf("\taccess key:\t%s\n", accessKey);
                System.out.printf("\tsecret key:\t%s\n", secretKey);
                System.out.printf("\tkey name:\t%s\n", keyName);
                System.out.printf("\timage name:\t%s\n", imageName);
                System.out.printf("\tinstance type:\t%s\n", instanceType);
                System.out.printf("\tversion:\t%s\n", version);
```

```
                System.out.printf("\tsignature version:\t%s\n", signatureVersion);
                System.out.printf("\tsignature method:\t%s\n", signatureMethod);


        }

        public enum CloudType {PRIVATE, PUBLIC, NOT_DEFINED};
        private CloudType cloudType;
        private String endPoint;
        private String shortEndPoint;
        private int maxVMs;
        private int currentVMs;
        private String accessKey;
        private String secretKey;
        private String keyName;
        private String instanceType;
        private String imageName;

        private String version;
        private String signatureVersion;
        private String signatureMethod;

        private String tagKey;
        private String tagValue;
        private String securityGroup;

}

-----------
-----------
-----------


File: project-fermi/plugins-src/vcluster/plugin/ec2/Command.java
package vcluster.plugin.ec2;

import java.util.ArrayList;
import java.util.List;
import java.util.StringTokenizer;
/**
 *This is a command enum for ec2 interface
 *Hao: add a command for create tag for VM instances
 */
public enum Command {


        RUN_INSTANCE ("RunInstances, runinstance, ri, runinst, runins, run"),
        START_INSTANCE ("StartInstances, startinstance, si, startinst, startins, start"),
        STOP_INSTANCE ("StopInstances, stopinstance, stop"),
        DESCRIBE_INSTANCE ("DescribeInstances, describeinstance, din, dins, descinst, descins"),
        TERMINATE_INSTANCE ("TerminateInstances, terminateinstance, terminate, ti, kill, killins"),
        DESCRIBE_IMAGE ("DescribeImages, describeimage, dim, dimg, descimg"),
        CREATE_TAG ("CreateTage, createtag"),
```

```
        NOT_DEFINED ("not_defined");

        private String command;
        private List<String> cmdList;


        Command( String cmdString) {

                this.cmdList = new ArrayList<String>();
                StringTokenizer st = new StringTokenizer(cmdString, ", ");

                boolean firstToken = true;
                String token = null;

                while(st.hasMoreTokens()) {
                        token = st.nextToken();
                        if (firstToken) {
                                command = token;
                                firstToken = false;
                        }
                        cmdList.add(token);
                }
        }


        public void toPrint()
        {
                System.out.println("Key = " + command);
                for(int i = 0; i < cmdList.size(); i++)
                        System.out.println("\t cmd: " + cmdList.get(i));
        }

        public boolean contains(String aCmd)
        {
                return cmdList.contains(aCmd);
        }

        public String getCommand()
        {
                return command;
        }


}

-----------
-----------
-----------

File: project-fermi/plugins-src/vcluster/plugin/ec2/Ec2.java
```

```
package vcluster.plugin.ec2;


/*

 * Hao: Add: createTag function which create a User-UserName tag for VM instances.

 *      Modified: create VM function, when new VM instances are being created, they are tagged with vcluster

 */



import java.io.BufferedReader;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.io.UnsupportedEncodingException;

import java.net.HttpURLConnection;

import java.net.URL;

import java.net.URLEncoder;

import java.security.InvalidKeyException;

import java.security.NoSuchAlgorithmException;

import java.util.ArrayList;

import java.util.Calendar;

import java.util.Collections;

import java.util.List;


import vcluster.elements.Host;

import vcluster.elements.Vm;

import vcluster.plugInterfaces.CloudInterface;

import vcluster.util.Util;
```

```
import com.sun.xml.bind.StringInputStream;


public class Ec2 implements CloudInterface{


	Cloud cloud = new Cloud();

	private static String makeGETQuery(Cloud cloud, QueryInfo ci)
		throws UnsupportedEncodingException, InvalidKeyException, NoSuchAlgorithmException
		{
			List<String> allKeyNames = new ArrayList<String>(ci.getKeySet());
			Collections.sort(allKeyNames, String.CASE_INSENSITIVE_ORDER);

			String queryString = "";

			StringBuffer stringToSign = new StringBuffer("GET\n"+cloud.getShortEndPoint()+"\n"+"/"+"\n");
			stringToSign.append("AWSAccessKeyId="+cloud.getAccessKey() + "&");

			boolean first = true;
			for (String keyName : allKeyNames) {
		if (first)
			first = false;
		else
			queryString += "&";
```

```
		if(ci.getAttrValue(keyName) == null) {
			System.out.println("Keyname = "+ keyName+" is null");
		}

				queryString += keyName+"="+URLEncoder.encode(ci.getAttrValue(keyName).toString(), "UTF-8");
			}
			//System.out.println("after 22222");
			stringToSign.append(queryString);
		String signature = GetSignature.calculateRFC2104HMAC(new String(stringToSign), cloud.getSecretKey(),
cloud.getSignatureMethod());

			String str = (queryString + "&Signature=" + URLEncoder.encode(signature, "UTF-8")
					+ "&AWSAccessKeyId="+cloud.getAccessKey());
			//System.out.println(str);
			return str;
		}




	public static ArrayList<Vm> executeQuery(Command command,String fullURL, String httpQuery)
	{


		try {
		URL endPoint = new URL(fullURL+"?"+httpQuery);
```

```
		return doHttpQuery(command,endPoint);
	} catch (Exception e)
	{
		e.printStackTrace();

	}
	return null;
}

private static ArrayList<Vm> doHttpQuery(Command command,URL endPoint) throws Exception {
	HttpURLConnection con = (HttpURLConnection) endPoint.openConnection();
	ArrayList<Vm> vmList = null;
	con.setRequestMethod("GET");
	con.setDoOutput(true);
	con.connect();
	String respStr;
//	System.out.println(endPoint.toString());
	if (con.getResponseCode() != HttpURLConnection.HTTP_OK) {
		ResponseDataHandler.handlError(con.getErrorStream());
	} else {
		//System.out.println("MARK 2");
		respStr = getResponseString(con.getInputStream());
		//System.out.println(respStr);
		vmList = ResponseDataHandler.handleResponse(command, new StringInputStream(respStr));
	}
	con.disconnect();
```

```
	return vmList;

}


private static String getResponseString(InputStream queryResp) throws Exception
{
	final InputStreamReader inputStreamReader = new InputStreamReader(queryResp);
	BufferedReader buffReader = new BufferedReader(inputStreamReader);

	String line = new String();
	StringBuffer stringBuff = new StringBuffer();

	while ((line = buffReader.readLine()) != null) {
		stringBuff.append(line);
	}

	return stringBuff.toString();
}

@Override
public boolean RegisterCloud(List<String> conf) {
	// TODO Auto-generated method stub
	cloud = new Cloud(conf);
	return true;
```

```
    }

    @Override
    public ArrayList<Vm> createVM(int maxCount) {

            //System.out.println("Under devoleping.......");
            QueryInfo qi = new QueryInfo();

            qi.putValue("Action", Command.RUN_INSTANCE.getCommand());
            qi.putValue("ImageId", cloud.getImageName());

            String timestamp = Util.getTimestampFromLocalTime(Calendar.getInstance().getTime());

            /* fill the default values */
            qi.putValue("MinCount", "1");
            qi.putValue("MaxCount", Integer.toString(maxCount));
            qi.putValue("InstanceType", cloud.getInstaceType());
     qi.putValue("SecurityGroup.1", cloud.getSecurityGroup());
            /* fill the default values */
     qi.putValue("Timestamp", timestamp);
            qi.putValue("Version", cloud.getVersion());

            if (cloud.getKeyName() != null)
                    qi.putValue("KeyName", cloud.getKeyName());
```

```
            qi.putValue("SignatureVersion", cloud.getSignatureVersion());
            qi.putValue("SignatureMethod", cloud.getSignatureMethod());

            //System.out.println("before");
            String query = null;
            try {
                    query = makeGETQuery(cloud, qi);
                    //System.out.println("before2");
            } catch (Exception e) {
                    e.printStackTrace();
            }
            //System.out.println(cloud.getEndPoint());

            ArrayList<Vm> vmlist =  executeQuery(Command.RUN_INSTANCE, cloud.getEndPoint(), query);
            createTag(vmlist);
            return vmlist;
    }


    public void createTag(ArrayList<Vm> vms)
  {
    QueryInfo qi = new QueryInfo();
            qi.putValue("Action", "CreateTags");
            String timestamp = Util.getTimestampFromLocalTime(Calendar.getInstance().getTime());
            String resource = "ResourceId.";
            for(int i=1; i<=vms.size();i++)
```

```
            {
                    String rid=resource+String.valueOf(i);
                    qi.putValue(rid, vms.get(i-1).getId());
            }


            /* fill the default values */
            qi.putValue("Tag.1.Key", cloud.getTagKey());
            qi.putValue("Tag.1.Value", cloud.getTagValue());
            qi.putValue("SignatureVersion", cloud.getSignatureVersion());
            qi.putValue("SignatureMethod", cloud.getSignatureMethod());
    qi.putValue("Timestamp", timestamp);
    qi.putValue("Version", cloud.getVersion());

            String query = null;
            try {
                    query = makeGETQuery(cloud, qi);
            } catch (Exception e) {
                    e.printStackTrace();
            }

    executeQuery(Command.CREATE_TAG, cloud.getEndPoint(), query);
}
   @Override
   public ArrayList<Vm> listVMs() {
            // TODO Auto-generated method stub
```

```
            //System.out.println("MARK 1.................................");
            //ArrayList<VMelement> vmList = null;
            QueryInfo qi = new QueryInfo();

            //qi.putValue("Action", "DescribeInstances");
            qi.putValue("Action", "DescribeInstances");
            String timestamp = Util.getTimestampFromLocalTime(Calendar.getInstance().getTime());

            /* fill the default values */
    qi.putValue("Timestamp", timestamp);
            qi.putValue("Version", cloud.getVersion());
    qi.putValue("Filter.1.Name", "tag:"+cloud.getTagKey());
    qi.putValue("Filter.1.Value.1", cloud.getTagValue());
            qi.putValue("SignatureVersion", cloud.getSignatureVersion());
            qi.putValue("SignatureMethod", cloud.getSignatureMethod());


            String query = null;
            try {
                    query = makeGETQuery(cloud, qi);
            } catch (Exception e) {
                    e.printStackTrace();
            }
            //System.out.println(query);
            try {
                    Thread.sleep(5000);
```

```
        } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
        return executeQuery(Command.DESCRIBE_INSTANCE,cloud.getEndPoint(), query);
//ArrayList<Vm> vmlist = executeQuery(Command.DESCRIBE_INSTANCE,cloud.getEndPoint(), query);
//createTag(vmlist);
//return vmlist;
        //return new ArrayList<Vm> ();
}

@Override
public ArrayList<Vm> destroyVM(Vm vm) {
        // TODO Auto-generated method stub

        QueryInfo qi = new QueryInfo();
        String timestamp = Util.getTimestampFromLocalTime(Calendar.getInstance().getTime());
        qi.putValue("Action", "TerminateInstances");
        qi.putValue("InstanceId.1", vm.getId());

        /* fill the default values */
        qi.putValue("Version", cloud.getVersion());
        qi.putValue("SignatureVersion", cloud.getSignatureVersion());
        qi.putValue("SignatureMethod", cloud.getSignatureMethod());
 qi.putValue("Timestamp", timestamp);
```

```
        String query = null;
        try {
                query = makeGETQuery(cloud, qi);
        } catch (Exception e) {
                e.printStackTrace();
        }

 return       executeQuery(Command.TERMINATE_INSTANCE, cloud.getEndPoint(), query);

}

@Override
public ArrayList<Vm> startVM(String id) {
        // TODO Auto-generated method stub
        QueryInfo qi = new QueryInfo();

        qi.putValue("Action", "StartInstances");
        qi.putValue("InstanceId.1", id);

        /* fill the default values */
        qi.putValue("SignatureVersion", cloud.getSignatureVersion());
        qi.putValue("SignatureMethod", cloud.getSignatureMethod());

        String timestamp = Util.getTimestampFromLocalTime(Calendar.getInstance().getTime());
        qi.putValue("Timestamp", timestamp);
        qi.putValue("Version", cloud.getVersion());
```

```
        String query = null;
        try {
                query = makeGETQuery(cloud, qi);
        } catch (Exception e) {
                e.printStackTrace();
        }

return executeQuery(Command.TERMINATE_INSTANCE, cloud.getEndPoint(), query);

}

@Override
public ArrayList<Vm> suspendVM(String id) {
        // TODO Auto-generated method stub
        QueryInfo qi = new QueryInfo();

        qi.putValue("Action", "StopInstances");
        qi.putValue("InstanceId.1", id);

        /* fill the default values */
        qi.putValue("SignatureVersion", cloud.getSignatureVersion());
        qi.putValue("SignatureMethod", cloud.getSignatureMethod());
        String timestamp = Util.getTimestampFromLocalTime(Calendar.getInstance().getTime());
        qi.putValue("Timestamp", timestamp);
        qi.putValue("Version", cloud.getVersion());
```

```
        String query = null;
        try {
                query = makeGETQuery(cloud, qi);
        } catch (Exception e) {
                e.printStackTrace();
        }
return executeQuery(Command.TERMINATE_INSTANCE, cloud.getEndPoint(), query);
}

@Override
public boolean hoston(String ipmiID) {
        // TODO Auto-generated method stub
        return false;
}

@Override
public boolean hostoff(String ipmiID) {
        // TODO Auto-generated method stub
        return false;
}

@Override
public boolean migrate(String vmid, String hostid) {
```

```
		// TODO Auto-generated method stub

		return false;

	}









	@Override

	public ArrayList<Vm> createVM(int maxCount, String hostID) {

		// TODO Auto-generated method stub

		return null;

	}









	@Override

	public ArrayList<Host> listHost() {

		// TODO Auto-generated method stub

		return null;

	}


}


-----------
-----------
-----------


File: project-fermi/plugins-src/vcluster/plugin/ec2/GetSignature.java
```

```
package vcluster.plugin.ec2;


import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

import org.apache.axis.encoding.Base64;

public class GetSignature {

	/**
	 * Computes RFC 2104-compliant HMAC signature.
	 *
	 * @param data
	 *            The data to be signed.
	 * @param key
	 *            The signing key.
	 * @return The base64-encoded RFC 2104-compliant HMAC signature.
	 * @throws java.security.SignatureException
	 *             when signature generation fails
	 * @throws NoSuchAlgorithmException
	 * @throws InvalidKeyException
	 */

	public static String calculateRFC2104HMAC(String data, String key, String sigMethod)
	throws NoSuchAlgorithmException, InvalidKeyException {
		String result;
		// get an hmac_sha1 key from the raw key bytes
		SecretKeySpec signingKey = new SecretKeySpec(key.getBytes(), sigMethod);

		// get an hmac_sha1 Mac instance and initialize with the signing key
		Mac mac = Mac.getInstance(sigMethod);
		mac.init(signingKey);


		// compute the hmac on input data bytes
		byte[] rawHmac = mac.doFinal(data.getBytes());
		result = Base64.encode(rawHmac);

		return result;
	}

}
-----------
-----------
-----------


File: project-fermi/plugins-src/vcluster/plugin/ec2/HttpsCertificate.java
```

```
package vcluster.plugin.ec2;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.security.KeyManagementException;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.UnrecoverableKeyException;
import java.security.cert.CertificateException;

import javax.net.ssl.KeyManagerFactory;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManagerFactory;

public class HttpsCertificate
{
    public static final String TRUST_STORE          = "javax.net.ssl.trustStore";
    public static final String TRUST_STORE_PASSWORD = "javax.net.ssl.trustStorePassword";
    public static final String KEY_STORE            = "javax.net.ssl.keyStore";
    public static final String KEY_STORE_PASSWORD   = "javax.net.ssl.keyStorePassword";

    private String _clientCertStoreFile = null;
    private InputStream _clientCertInputStream = null;
```

```
    private String _clientCertStorePswd = null;
    private String _serverCertStoreFile = null;
    private InputStream _serverCertInputStream = null;
    private String _serverCertStorePswd = null;

    private KeyManagerFactory _kmFactory;
    private TrustManagerFactory _tmFactory;

    private static boolean _debug = false;

    /**
     * If we just care server side cert, not client side cert.
     * @param serverCertStoreFile String
     * @param serverCertStorePswd String
     */
    public HttpsCertificate(String serverCertStoreFile, String serverCertStorePswd)
    {
        _serverCertStoreFile = serverCertStoreFile;
        _serverCertStorePswd = serverCertStorePswd;
    }

    public HttpsCertificate(InputStream serverCertInputStream, String serverCertStorePswd)
    {
        _serverCertInputStream = serverCertInputStream;
        _serverCertStorePswd = serverCertStorePswd;
    }
```

```
/**
 * If we care both server side and client side certs.
 * @param clientCertStoreFile String
 * @param clientCertStorePswd String
 * @param serverCertStoreFile String
 * @param serverCertStorePswd String
 */
public HttpsCertificate(String clientCertStoreFile, String clientCertStorePswd,
                        String serverCertStoreFile, String serverCertStorePswd)
{
    _clientCertStoreFile = clientCertStoreFile;
    _clientCertStorePswd = clientCertStorePswd;
    _serverCertStoreFile = serverCertStoreFile;
    _serverCertStorePswd = serverCertStorePswd;
}

public HttpsCertificate(InputStream clientCertInputStream, String clientCertStorePswd,
                        InputStream serverCertInputStream, String serverCertStorePswd)
{
    _clientCertInputStream = clientCertInputStream;
    _clientCertStorePswd = clientCertStorePswd;
    _serverCertInputStream = serverCertInputStream;
    _serverCertStorePswd = serverCertStorePswd;
}
```

```
/**
 * This version works for all stand-alone apps, not for web apps.
 */
public void setCertForStandAlone()
{
    if (_clientCertStoreFile != null)
    {
        System.setProperty(TRUST_STORE, _clientCertStoreFile);
        System.setProperty(TRUST_STORE_PASSWORD, _clientCertStorePswd);
    }
    if (_serverCertStoreFile != null)
    {
        System.setProperty(KEY_STORE, _serverCertStoreFile);
        System.setProperty(KEY_STORE_PASSWORD, _serverCertStorePswd);
    }
}

/**
 * This version works for web apps.
 */
public void setCertForWebApps(Object httpConn)
{
    setClientCert();
    setServerCert();

    try
```

```
    {
        // now create the context
        SSLContext ctx = SSLContext.getInstance("SSL");

        if (_kmFactory != null)
        {
            if (_tmFactory != null)
            {
                ctx.init(_kmFactory.getKeyManagers(), _tmFactory.getTrustManagers(), null);
            }
            else
            {
                ctx.init(_kmFactory.getKeyManagers(), null, null);
            }
        }
        else
        {
            if (_tmFactory != null)
            {
                ctx.init(null, _tmFactory.getTrustManagers(), null);
            }
            else
            {
                return; // if both null, there is no need to set anything at all.
            }
        }
```

```
        // without the following line, we would get the following error:
        // in a web server, hit it the first time, we get this error:
        //     javax.net.ssl.SSLHandshakeException: Received fatal alert: handshake_failure
        // but hit it again, it's fine. Looks like it didn't take what we set for the default.
        // so we set it in this method.
        // Furthermore, for backward compatibility, we have to check these,
        // thanks to IBM and SUN for making this world more miserable!
        System.out.println("httpConn Class=" + httpConn.getClass().getName());

    }
    catch (NoSuchAlgorithmException nsae)
    {
        nsae.printStackTrace();
       // throw new HttpException("error in loading SSL context");
    }
    catch (KeyManagementException kme)
    {
        kme.printStackTrace();
       // throw new HttpException("error on initializing SSL context");
    }
}

private void setClientCert()
{
    debug("set client side cert ...");
```

```
    if (_clientCertStoreFile != null)
    {
        debug("read client side file ...");
        _clientCertInputStream = getKeystoreInputStream(_clientCertStoreFile);
    }

    try
    {
        if (_clientCertInputStream != null)
        {
            debug("setting client side cert ...");
            KeyStore clientCertKeyStore = KeyStore.getInstance(KeyStore.getDefaultType());
            clientCertKeyStore.load(_clientCertInputStream, _clientCertStorePswd.toCharArray());
            _kmFactory = KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
            _kmFactory.init(clientCertKeyStore, _clientCertStorePswd.toCharArray());
        }
        else
        {
            debug("nothing to set for the client cert!");
        }
    }
    catch (IOException ioe)
    {
        ioe.printStackTrace();
       // throw new HttpException("can't load client certificate file");
    }
```

```
    catch (KeyStoreException kse)
    {
        kse.printStackTrace();
       // throw new HttpException("can't find keystore instance");
    }
    catch (NoSuchAlgorithmException nsae)
    {
        nsae.printStackTrace();
        //throw new HttpException("error in loading client cert");
    }
    catch (CertificateException ce)
    {
        ce.printStackTrace();
       // throw new HttpException("error in reading");
    }
    catch (UnrecoverableKeyException uke)
    {
        uke.printStackTrace();
       // throw new HttpException("error in initializing client cert");
    }
}

private void setServerCert()
{
    debug("set server side cert ...");
    if (_serverCertStoreFile != null)
```

```
    {
        debug("read server side cert ...");
        _serverCertInputStream = getKeystoreInputStream(_serverCertStoreFile);
    }
    if (_serverCertInputStream != null)
    {
        try
        {
            debug("setting server side cert ...");
            KeyStore trustStore = KeyStore.getInstance(KeyStore.getDefaultType());
            trustStore.load(_serverCertInputStream, _serverCertStorePswd.toCharArray());
            _tmFactory = TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
            _tmFactory.init(trustStore);
        }
        catch (IOException ioe)
        {
            ioe.printStackTrace();
          //  throw new HttpException("can't load server certificate file");
        }
        catch (KeyStoreException kse)
        {
            kse.printStackTrace();
         //   throw new HttpException("can't find server cert keystore instance");
        }
        catch (NoSuchAlgorithmException nsae)
        {
```

```
            nsae.printStackTrace();
          //  throw new HttpException("error in loading server cert");
        }
        catch (CertificateException ce)
        {
            ce.printStackTrace();
           // throw new HttpException("error in reading server cert");
        }
    }
    else
    {
        debug("Nothing to set for the server cert");
    }
}

private InputStream getKeystoreInputStream(String fileName)
{
    InputStream retInputStream = null;

    if (fileName == null) return retInputStream;

    try
    {
        retInputStream = new FileInputStream(fileName);
        return retInputStream;
    }
```

```
        catch (FileNotFoundException fnfe)

        {

            System.out.println("info: failed to get file: " + fileName + " through filesystem: " + fnfe.getMessage());

            System.out.println("Note: this is just info, not necessarily a fault! Will try with classpath loading.");

            retInputStream = this.getClass().getResourceAsStream(fileName);

        }


        //retInputStream = HTTPSMessage.class.getResourceAsStream(fileName);

        //retInputStream = Thread.currentThread().getContextClassLoader().getResourceAsStream(fileName);

        debug("The input stream is: " + retInputStream);

        return retInputStream;

    }


    private static void debug(String s)

    {

        if (_debug == true && s != null) System.out.println("HttpsCertificate: - " + s);

    }

}



-----------
-----------
-----------


File: project-fermi/plugins-src/vcluster/plugin/ec2/QueryInfo.java
package vcluster.plugin.ec2;

import java.util.HashMap;
import java.util.Map;
import java.util.Set;

//import vcluster.ui.Menu;
```

```
public class QueryInfo {

       public QueryInfo()
       {
              cmdType =  Command.NOT_DEFINED;
           attrMap = new HashMap<String, String>();
       }

       public void putCmdType(Command aCmdType)
       {
              cmdType = aCmdType;
       }

       public Command getCmdType()
       {
              return cmdType;
       }


       public String getAttrValue(String aKeyString)
       {
              return attrMap.get(aKeyString);
       }

       public Set<String> getKeySet()
       {
              return attrMap.keySet();
       }

       public void print()
       {
              System.out.println("--------------------------------------");
              System.out.println("Cmd Type: " + cmdType);

              for (Map.Entry<String, String> e : attrMap.entrySet())
                  System.out.println(e.getKey() + ": " + e.getValue());


              System.out.println("--------------------------------------");
       }

       public void putValue(String keyStr, String valStr)
       {
              attrMap.put(keyStr, valStr);
       }


    private Map <String, String> attrMap;
    private Command cmdType =  Command.NOT_DEFINED;
}

-----------
-----------
```

```
-----------

File: project-fermi/plugins-src/vcluster/plugin/ec2/ResponseDataHandler.java
/*
 * Hao: Add the function to create tags for each virtual machine instances for vcluster.
 *      When describe VM instances, only those taged with vcluster will be listed.
 *      Fixed bug for "stopped" VM status.
 */

package vcluster.plugin.ec2;

import java.io.BufferedReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.io.StringWriter;
import java.util.ArrayList;

import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;

import vcluster.elements.Vm;
import vcluster.managers.VmManager.VMState;

public class ResponseDataHandler {

	public static ArrayList<Vm> handleResponse(Command command, InputStream is) throws Exception
	{
		/*BufferedReader rd = new BufferedReader(new InputStreamReader(is));
		StringBuffer sb = new StringBuffer();
		String line;
		while ((line = rd.readLine()) != null) {
		sb.append(line);
		}
		rd.close();

	System.out.println(sb.toString()+"\n");*/

		Document doc = DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(is);
		ArrayList<Vm> vmList = new ArrayList<Vm>();

		switch(command) {
```

```
		case DESCRIBE_INSTANCE:
			vmList =  describeInstanceResponse(doc);
			break;
		case DESCRIBE_IMAGE:
			describeImageResponse(doc);
			break;
		case RUN_INSTANCE:
			vmList =  operateResponse(doc);
			break;
		case TERMINATE_INSTANCE:
			vmList =  operateResponse(doc);
			break;
		case STOP_INSTANCE:
			vmList =  operateResponse(doc);
			break;
		case START_INSTANCE:
			vmList =  operateResponse(doc);
			break;
		case CREATE_TAG:
			createTagResponse(doc);
		default:
			break;
		}

		saveResponse(doc);
		return vmList;
	}


	private static ArrayList<Vm> operateResponse(Document doc) {
		// TODO Auto-generated method stub
		 ArrayList<Vm> vmList = new ArrayList<Vm>();
		Element instNodeList = findFirstMatchedNode(doc, "instancesSet");

		if (instNodeList == null) {
			System.out.println("[Error] : cannot find instanceSet element!");

		}

		String instanceId = "";
		String instanceState = "";
		for(int i = 0; i < instNodeList.getElementsByTagName("item").getLength();i++) {
			Vm vm = new Vm();

			Element anItem = (Element)instNodeList.getElementsByTagName("item").item(i);

				instanceId = getTextValue(anItem,"instanceId");
				if (instanceId == null || instanceId.equals(""))
					continue;

				instanceState = getTextValue(anItem,"name");
				if (instanceState == null || instanceState.equals(""))
					continue;
```

```
                        //System.out.println(instanceId+"\t\t"+instanceState);
                        vm.setId(instanceId);
                        if(instanceState.equalsIgnoreCase("running")){
                                vm.setState(VMState.RUNNING);
                        }else if(instanceState.equalsIgnoreCase("stopped")){
                                vm.setState(VMState.STOP);
                        }else if(instanceState.equalsIgnoreCase("Pending")){
                                vm.setState(VMState.PENDING);
                        }else if(instanceState.equalsIgnoreCase("Prolog")){
                                vm.setState(VMState.PROLOG);
                        }else if(instanceState.equalsIgnoreCase("Suspended")){
                                vm.setState(VMState.SUSPEND);
                        }else if(instanceState.equalsIgnoreCase("terminated")){
                                vm.setState(VMState.FAILED);
                        }
                        else{
                                vm.setState(VMState.NOT_DEFINED);
                        }
                        vmList.add(vm);
                }
                //System.out.println("----------------------------------------");
                return vmList;
        }

        private static ArrayList<Vm> describeInstanceResponse(Document doc) throws Exception
        {

                 ArrayList<Vm> vmList = new ArrayList<Vm>();
                Element reservSet = findFirstMatchedNode(doc, "reservationSet");

                if (reservSet == null) {
                        System.out.println("[Error] : cannot find instanceSet element!");
                }

                //get a nodelist of  elements
                NodeList instNodeList = reservSet.getElementsByTagName("instancesSet");

                if(instNodeList == null || instNodeList.getLength() <= 0) {
                        System.out.println("[Error] : but no item element!");
                }

                //System.out.println("----------------------------------------");
                //System.out.println("Inst ID\t\tStatus");
                //System.out.println("----------------------------------------");

                String instanceId = "";
                String instanceState = "";
                String privateIP = "";
                String publicIP = "";
                String lunTime ="";
                String dnsName="";
```

```
                for(int i = 0 ; i < instNodeList.getLength();i++) {

                        Element anElement = (Element)instNodeList.item(i);
                        NodeList itemNodeList = anElement.getElementsByTagName("item");
                        for(int j = 0; j < itemNodeList.getLength(); j++) {

                                Element anItem = (Element)itemNodeList.item(j);
                                Vm vm = new Vm();
                                instanceId = getTextValue(anItem,"instanceId");
                                if(instanceId=""||instanceId==null){
                                        continue;
                                }
                                String ips = getTextValue(anItem,"privateIpAddress");
                                if(ips==null)continue;
                                String [] ipStr = ips.split(",");
                                privateIP = ipStr[0].trim();
                                if(ipStr.length>1){publicIP = ipStr[1].trim();}
                                else{try {
                                        publicIP =getTextValue(anItem,"ipAddress").split(",")[0].trim();
                                } catch (Exception e) {
                                        // TODO Auto-generated catch block
                                        publicIP = "";
                                }}
                                lunTime = getTextValue(anItem,"launchTime");
                                if (instanceId == null || instanceId.equals(""))
                                        continue;

                                instanceState = getTextValue(anItem,"name");
                                if (instanceState == null || instanceState.equals(""))
                                        continue;

                                dnsName = getTextValue(anItem,"dnsName");
                                if (dnsName == null || dnsName.equals(""))
                                        continue;
                                vm.setId(instanceId);
                                vm.setPrivateIP(privateIP);
                                vm.setPubicIP(publicIP);
                                vm.setTime(lunTime);
                                vm.setHostname("host1");
                                vm.setDNSName(dnsName);
                                if(instanceState.equalsIgnoreCase("running")){
                                        vm.setState(VMState.RUNNING);
                                }else if(instanceState.equalsIgnoreCase("stopped")){
                                        vm.setState(VMState.STOP);
                                }else if(instanceState.equalsIgnoreCase("Pending")){
                                        vm.setState(VMState.PENDING);
                                }else if(instanceState.equalsIgnoreCase("Prolog")){
                                        vm.setState(VMState.PROLOG);
                                }else if(instanceState.equalsIgnoreCase("Suspended")){
                                        vm.setState(VMState.SUSPEND);
                                }else if(instanceState.equalsIgnoreCase("terminated")){
                                        vm.setState(VMState.FAILED);
                                }else{
```

```
                        vm.setState(VMState.NOT_DEFINED);
                }
                vmList.add(vm);
            }
        }
        //System.out.println("----------------------------------------");
        saveResponse(doc);
        return vmList;

    }

    private static void createTagResponse(Document doc) throws Exception
    {
        Element returnEle = findFirstMatchedNode(doc, "return");
        NodeList nl = returnEle.getChildNodes();
        String textVal = nl.item(0).getNodeValue();

        if (!textVal.equalsIgnoreCase("true"))
        {
            System.out.println("[Error] : cannot tag all the VM instances!");
        }
    }
    private static void describeImageResponse(Document doc) throws Exception
    {

        Element imgSetEle = findFirstMatchedNode(doc, "imagesSet");

        if (imgSetEle == null) {
            System.out.println("[Error] : cannot find imageSet element!");
            return;
        }

        //get a nodelist of  elements
        NodeList nl = imgSetEle.getElementsByTagName("item");

        if(nl == null || nl.getLength() <= 0) {
            System.out.println("[Error] : but no item element!");
        }

        System.out.println("----------------------------------------");
        System.out.println("Img Id\t\tOwner\t\tisPublic?");
        System.out.println("----------------------------------------");

        for(int i = 0 ; i < nl.getLength();i++) {

            String imageId = "";
            String ownerId = "";
            String isPublic = "";

            /* get an item element */
            Element el = (Element)nl.item(i);

            /* get image id */
```

```
            imageId = getTextValue(el,"imageId");

            if (imageId == null || imageId.equals(""))
                continue;

            /* get owner id */
            ownerId = getTextValue(el,"imageOwnerId");

            if (ownerId == null || ownerId.equals(""))
                continue;

            /* get owner id */
            isPublic = getTextValue(el,"isPublic");

            if (isPublic == null || isPublic.equals(""))
                continue;

            System.out.println(imageId+"\t"+ownerId+"\t\t"+isPublic);
        }

        System.out.println("----------------------------------------");

        //saveResponse(doc);
    }


    private static Element findFirstMatchedNode(Document doc, String nodeName)
    {
        /* got a root element */
        Element docEle = doc.getDocumentElement();

        NodeList nl = docEle.getElementsByTagName(nodeName);
        return (Element)nl.item(0);
    }

    public static void handlError(InputStream is) throws Exception
    {

        BufferedReader rd = new BufferedReader(new InputStreamReader(is));
            StringBuffer sb = new StringBuffer();
            String line;
            while ((line = rd.readLine()) != null) {
            sb.append(line);
            }
            rd.close();
        System.out.println(sb.toString()+"\n");
        Document doc = DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(is);

        /* get the root element */
        Element docEle = doc.getDocumentElement();
        /* get a nodelist of  elements */
        NodeList nl = docEle.getElementsByTagName("Error");
        if(nl == null || nl.getLength() <= 0) {
```

```
                    System.out.println("[Error] : hard to explain....");
                    System.out.println("        : copy and paste the below end point and see..why it is difficult to
explain....");
                    // System.out.println("        : " + QueryExecutor.getEndPoint());
             } else {
                    for(int i = 0 ; i < nl.getLength();i++) {
                           //get an error element
                           Element el = (Element)nl.item(i);
                           String msg = getTextValue(el,"Message");
                           System.out.println("[Error] : "+msg);
                    }
             }
             saveResponse(doc);
       }

       private static String getTextValue(Element ele, String tagName) {
             String textVal = null;
             NodeList nl = ele.getElementsByTagName(tagName);
             if(nl != null && nl.getLength() > 0) {
                    Element el = (Element)nl.item(0);
                    textVal = el.getFirstChild().getNodeValue();
             }
             return textVal;
       }

       private static void saveResponse(Document doc) throws Exception
       {
             Transformer transformer = TransformerFactory.newInstance().newTransformer();
             transformer.setOutputProperty(OutputKeys.INDENT, "yes");
             //initialize StreamResult with File object to save to file
             StreamResult result = new StreamResult(new StringWriter());
             DOMSource source = new DOMSource(doc);
             transformer.transform(source, result);
             String xmlString = result.getWriter().toString();

        try {
                    FileWriter outFile = new FileWriter("amazonResponse.xml");
                    PrintWriter out = new PrintWriter(outFile);
                    out.println(xmlString);
                    outFile.close();
             } catch (IOException e){
                    e.printStackTrace();
             }
       }
}

-----------
-----------
-----------


File: project-fermi/plugins-src/vcluster/plugin/ec2/Signature.java
```

```
package vcluster.plugin.ec2;


import java.security.SignatureException;

import javax.crypto.Mac;

import javax.crypto.spec.SecretKeySpec;

import com.amazonaws.util.*;


/**

* This class defines common routines for generating

* authentication signatures for AWS Platform requests.

*/

public class Signature {

    private static final String HMAC_SHA256_ALGORITHM = "HmacSHA256";


    /**

     * Computes RFC 2104-compliant HMAC signature.

     * * @param data

     * The signed data.

     * @param key

     * The signing key.

     * @return

     * The Base64-encoded RFC 2104-compliant HMAC signature.

     * @throws

     * java.security.SignatureException when signature generation fails

     */

    public static String calculateRFC2104HMAC(String data, String key)
```

```
    throws java.security.SignatureException

    {

        String result;

        try {


            // Get an hmac_sha256 key from the raw key bytes.

            SecretKeySpec signingKey = new SecretKeySpec(key.getBytes("UTF8"), HMAC_SHA256_ALGORITHM);


            // Get an hmac_sha256 Mac instance and initialize with the signing key.

            Mac mac = Mac.getInstance(HMAC_SHA256_ALGORITHM);

            mac.init(signingKey);


            // Compute the hmac on input data bytes.

            byte[] rawHmac = mac.doFinal(data.getBytes("UTF8"));


            // Base64-encode the hmac by using the utility in the SDK

            result = BinaryUtils.toBase64(rawHmac);




        } catch (Exception e) {

            throw new SignatureException("Failed to generate HMAC : " + e.getMessage());

        }

        return result;

    }

}
-----------
-----------
-----------
```

```
File: project-fermi/plugins-src/vcluster/plugin/proxy/fermicloud/FermiCloud.java
package vcluster.plugin.proxy.fermicloud;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.List;
import java.util.StringTokenizer;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

import vcluster.Vcluster;
import vcluster.elements.Host;
import vcluster.elements.Slot;
import vcluster.elements.Vm;
import vcluster.managers.VmManager.VMState;
import vcluster.plugInterfaces.CloudInterface;

public class FermiCloud implements CloudInterface {


        private static void closeStream(BufferedReader in, DataOutputStream out, Socket socket)
        {
                try {
                 if (in != null) in.close();
                 if (out != null) out.close();
                 if (socket != null) socket.close();
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }

        private ArrayList<String> socketToproxy(String cmd){
                String cmdLine=cmd+"\n";
                ArrayList<String> feedBack = new ArrayList<String>();
                 Socket socket = null;
                 BufferedReader in = null;
                 DataOutputStream out = null;
                 //System.out.println(cmdLine);
```

```
        try {
              socket = new Socket(addr, port);

            in = new BufferedReader(new InputStreamReader(socket.getInputStream(), "UTF-8"));
            out = new DataOutputStream(socket.getOutputStream());
            out.flush();
            /* make an integer to unsigned int */
            int userInput = 5;
            userInput <<= 8;
            userInput |=  1;
            userInput &= 0x7FFFFFFF;

            String s = Integer.toString(userInput);
            byte[] b = s.getBytes();

            out.write(b, 0, b.length);
            out.write(cmdLine.getBytes(), 0, cmdLine.getBytes().length);
            out.flush();

            String str=null;
              while((str=in.readLine())!=null)
              {
                     str=str.trim();
                     feedBack.add(str);
              }
            /*char[] cbuf = new char[4096];
              String temp = null;
              while (in.read(cbuf, 0, 4096) != -1) {
              String str = new String(cbuf);
               str = str.trim();
               if (!str.equals(temp)){
                     //System.out.println(str);
                      feedBack.add(str);
               }

               //cbuf[0] = '\0';
              temp = str;
            }*/

        } catch (UnknownHostException e) {
              System.out.print("ERROR: " +e.getMessage());
            closeStream(in, out, socket);
            return feedBack;
        } catch (IOException e) {
              System.out.print("ERROR: " +e.getMessage());
            closeStream(in, out, socket);
            return feedBack;
        }

        closeStream(in, out, socket);
        return feedBack;
}
```

```
private ArrayList<String> socketToproxy(String cmd, String address){
       String cmdLine=cmd+"\n";
       ArrayList<String> feedBack = new ArrayList<String>();
        Socket socket = null;
        BufferedReader in = null;
        DataOutputStream out = null;
        //System.out.println(cmdLine);
        try {
              socket = new Socket(address, port);

            in = new BufferedReader(new InputStreamReader(socket.getInputStream(), "UTF-8"));
            out = new DataOutputStream(socket.getOutputStream());
            out.flush();
            /* make an integer to unsigned int */
            int userInput = 5;
            userInput <<= 8;
            userInput |=  1;
            userInput &= 0x7FFFFFFF;

            String s = Integer.toString(userInput);
            byte[] b = s.getBytes();

            out.write(b, 0, b.length);
            out.write(cmdLine.getBytes(), 0, cmdLine.getBytes().length);
            out.flush();

            String str=null;
              while((str=in.readLine())!=null)
              {
                     str=str.trim();
                     feedBack.add(str);
              }
            /*char[] cbuf = new char[4096];
              String temp = null;
              while (in.read(cbuf, 0, 4096) != -1) {
              String str = new String(cbuf);
               str = str.trim();
               if (!str.equals(temp)){
                     //System.out.println(str);
                      feedBack.add(str);
               }

               //cbuf[0] = '\0';
              temp = str;
            }*/

        } catch (UnknownHostException e) {
              String msg="ERROR: " +address+" "+e.getMessage();
              System.out.print(msg);
              Vcluster.writeSysLogFile(msg);
            closeStream(in, out, socket);
            return feedBack;
        } catch (IOException e) {
```

```
                String msg="ERROR: " +address+" "+e.getMessage();
                System.out.print(msg);
                Vcluster.writeSysLogFile(msg);
            closeStream(in, out, socket);
            return feedBack;
        }

        closeStream(in, out, socket);
        return feedBack;
}
@Override
public boolean RegisterCloud(List<String> configurations) {
        // TODO Auto-generated method stub

        for(String aLine : configurations){

                StringTokenizer st = new StringTokenizer(aLine, "=");

                if (!st.hasMoreTokens()) return false;

                /* get a keyword */
                String aKey = st.nextToken().trim();

                /* get a value */
                if (!st.hasMoreTokens()) return false;

                String aValue = st.nextToken().trim();

                if (aKey.equalsIgnoreCase("username")) {
                } else if (aKey.equalsIgnoreCase("endpoint"))
                        this.addr = aValue;
                else if (aKey.equalsIgnoreCase("port"))
                        this.port = Integer.parseInt(aValue);
                else if (aKey.equalsIgnoreCase("template")){
                        this.template = aValue;
                }
                else if (aKey.equalsIgnoreCase("ipmiParas")){
                        this.ipmiParas = aValue;
                }
                else if (aKey.equalsIgnoreCase("cluster")){
                        this.clusterName=aValue;
                }
                else if (aKey.equalsIgnoreCase("Name")){
                        this.cloudName=aValue;
                }

        }
        ArrayList<String> dateR = socketToproxy("date -R");
        //if(dateR.get(0).split(regex))

        try {
                if(dateR!=null&&!dateR.isEmpty()) {
                }
```

```
        } catch (NumberFormatException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }

        return true;
}

public String getVmNames(String id)
{
 String cmdLine = "onehostname "+id;
 //System.out.println(cmdLine);
        ArrayList<String> feedBack = socketToproxy(cmdLine);
        if(feedBack!=null&&!feedBack.isEmpty())
                        return feedBack.get(0);
        else
        {
                System.out.println("[Error:] Cannot get hostname!");
                return null;
        }
}

@Override
public ArrayList<Vm> createVM(int maxCount) {
        // TODO Auto-generated method stub
        String cmdLine="onetemplate instantiate "+template +" -m "+maxCount + " -n vcluster_worker";
        System.out.println(cmdLine);
        ArrayList<Vm> vmList = new ArrayList<Vm>();
        ArrayList<String> feedBack = socketToproxy(cmdLine);
        if(feedBack!=null&&!feedBack.isEmpty()&&feedBack.get(0).contains("ID:")){
                for(int i = 0;i<feedBack.size();i++){
                        if(!feedBack.get(i).contains("ID:"))
                                continue;
                        String [] vmEle = feedBack.get(i).split("\\s+");
                        Vm vm = new Vm();
                        vm.setId(vmEle[2]);
                        vm.setState(VMState.PROLOG);
                        vm.setDNSName(getVmNames(vmEle[2]));
                        vmList.add(vm);
                }
        }else{
                System.out.println(feedBack.get(0));
                return null;
        }
                return vmList;
}

@Override
public ArrayList<Vm> createVM(int maxCount, String hostID)
{
        String cmdLine="onetemplate instantiate "+template +" -m "+maxCount + " -n vcluster_worker";
        System.out.println(cmdLine);
        ArrayList<Vm> vmList = new ArrayList<Vm>();
```

```
		ArrayList<String> feedBack = socketToproxy(cmdLine);
		if(feedBack!=null&&!feedBack.isEmpty()&&feedBack.get(0).contains("ID:")){
			for(int i = 0;i<feedBack.size();i++){
				if(!feedBack.get(i).contains("ID:"))
					continue;
				String [] vmEle = feedBack.get(i).split("\\s+");
				Vm vm = new Vm();
				vm.setId(vmEle[2]);
				vm.setState(VMState.PROLOG);
				vm.setDNSName(getVmNames(vmEle[2]));
  vmList.add(vm);
			}
		}
		if(vmList.isEmpty()){
			System.out.println("[Error:] No vm has been created!!");
			return null;
		}
		for(int j=0;j<vmList.size();j++)
		{
			cmdLine="onevm deploy "+ vmList.get(j).getId()+" "+ hostID;
			System.out.println(cmdLine);
			socketToproxy(cmdLine);
		}
			return vmList;
	}

	private boolean getVminf(Vm vm){
		ArrayList<String> feedBack = socketToproxy("onevm show "+vm.getId()+" | grep IP");


		if(feedBack!=null&&!feedBack.isEmpty()){

			for(String str : feedBack){
				try {
					if(str.contains("IP_PUBLIC")){
						vm.setPubicIP(str.split("=")[1].replace(",", "").replaceAll("\"",
"").trim());
					}else if(str.contains("IP=")){
						vm.setPrivateIP(str.split("=")[1].trim().replace(",", "").replaceAll("\"",
""));
					}
					else if(str.contains("IP_PRIVATE")){
						vm.setPrivateIP(str.split("=")[1].replace(",", "").replaceAll("\"", ""));
					}
				} catch (Exception e) {
					// TODO Auto-generated catch block
					return false;
				}
			}
			return true;
		}
		return true;
	}
```

```
	@Override
	public ArrayList<Vm> listVMs() {
		// TODO Auto-generated method stub
		String cmdLine="onevm list|grep vcluster";
		ArrayList<Vm> vmList = new ArrayList<Vm>();
		ArrayList<String> feedBack = socketToproxy(cmdLine);
		boolean flag = true;
		while(flag){
		if(feedBack!=null&&!feedBack.isEmpty()){
			for(int i = 0;i<feedBack.size();i++){
				//System.out.println(feedBack.get(i));
				String [] vmEle = feedBack.get(i).split("\\s+");
				if(vmEle.length<9){
					continue;
				}
				Vm vm = new Vm();
				try{
					vm.setId(vmEle[0]);
					//vm.setState(vmEle[4]);
					getVminf(vm);
					vm.setDNSName(getVmNames(vmEle[0]));
					vm.setUser(vmEle[1]);
					vm.setGroup(vmEle[2]);

					if(vmEle[4].equalsIgnoreCase("runn")){
						vm.setState(VMState.RUNNING);
					}else if(vmEle[4].equalsIgnoreCase("stop")){
						vm.setState(VMState.STOP);
					}else if(vmEle[4].equalsIgnoreCase("Pend")){
						vm.setState(VMState.PENDING);
					}else if(vmEle[4].equalsIgnoreCase("Prol")){
						vm.setState(VMState.PROLOG);
					}else if(vmEle[4].equalsIgnoreCase("Susp")){
						vm.setState(VMState.SUSPEND);
					}else{
						vm.setState(VMState.NOT_DEFINED);
					}
					vm.setUcpu(vmEle[5]);
					vm.setMemory(vmEle[6]);
					if(vm.getState()==VMState.STOP){vm.setHostname("");}else{
						vm.setHostname(vmEle[7]);
					}
				}catch(Exception e){
					e.printStackTrace();
					continue;
				}

				vmList.add(vm);

			}
			flag = false;
		}else if(feedBack==null||feedBack.isEmpty())
```

```
                return null;
        else if(feedBack.get(0).contains("ReadTimeout")){
                try {
                        Thread.sleep(5000);
                } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                System.out.println("ReadTimeout, waiting 5s... ...");

        }else{
                System.out.println(feedBack.get(0));
                flag = false;
                return null;
        }
        }
        //System.out.println("opennebula plugin:"+vmList.size());
        return vmList;
}

@Override
public ArrayList<Vm> destroyVM(Vm vm) {
        // TODO Auto-generated method stub

        String cmdLine = "/cloud/login/hwu28/scripts/deletevm "+vm.getId();
        System.out.println(cmdLine);
        ArrayList<Vm> vmList = new ArrayList<Vm>();
        ArrayList<String> feedBack = socketToproxy(cmdLine);
        if(feedBack!=null&&!feedBack.isEmpty()){
                System.out.println(feedBack.get(0));
                return null;
        }

        Vm vm_1 = new Vm();
        vm_1.setId(vm.getId());
        vm_1.setState(VMState.STOP);
        vmList.add(vm_1);
        return vmList;
}

@Override
public ArrayList<Vm> startVM(String id) {
        // TODO Auto-generated method stub
        String cmdLine="onevm resume "+id;
        ArrayList<Vm> vmList = new ArrayList<Vm>();
        ArrayList<String> feedBack = socketToproxy(cmdLine);
        if(feedBack!=null&&!feedBack.isEmpty()){
                System.out.println(feedBack.get(0));
                return null;
        }
        Vm vm = new Vm();
        vm.setId(id);
        vm.setState(VMState.PROLOG);
```

```
        vmList.add(vm);
        return vmList;
}

@Override
public ArrayList<Vm> suspendVM(String id) {
        // TODO Auto-generated method stub
        String cmdLine = "onevm suspend "+id;
        ArrayList<Vm> vmList = new ArrayList<Vm>();
        ArrayList<String> feedBack = socketToproxy(cmdLine);
        if(feedBack!=null&&!feedBack.isEmpty()){
                System.out.println(feedBack.get(0));
                return null;
        }
        Vm vm = new Vm();
        vm.setId(id);
        vm.setState(VMState.SUSPEND);
        vmList.add(vm);
        return vmList;
}



@Override
public boolean migrate(String vmid, String hostid) {
        // TODO Auto-generated method stub
        String cmdLine = "onevm migrate "+ vmid + " "+ hostid;

    socketToproxy(cmdLine);

    return true;
}

@Override
public boolean hoston(String ipmiID) {
        // TODO Auto-generated method stub
        String cmdLine = "ipmitool" + ipmiParas + " -H "+ipmiID+ " power on";

    ArrayList<String> feedback = socketToproxy(cmdLine);
    if(feedback.get(0).equalsIgnoreCase("Chassis Power Control: Up/On")){

       return true;
    }else{
       return false;
    }
}

@Override
public boolean hostoff(String ipmiID) {
        // TODO Auto-generated method stub
        String cmdLine = "ipmitool" + ipmiParas + " -H "+ipmiID+ " power off";
```

```
        ArrayList<String> feedback = socketToproxy(cmdLine);
        if(feedback.get(0).equalsIgnoreCase("Chassis Power Control: Down/Off")){
           return true;
        }
           return false;
    }


private String addr;
   private int port;
   private String template;
   private String clusterName;
   private String ipmiParas;
   private String cloudName;


   /*
    * Added functions to list hosts machine information
    * @see vcluster.plugInterfaces.CloudInterface#listHost()
    */
@Override
   public ArrayList<Host> listHost() {
          //ArrayList<Host> tempHosts=getHosts();
          return getHosts();
   }

public Document getXmlHost(){
   Document doc = null;
          File f = new File("host.xml");
          String cmdLine2 ="onehost list -x";
          ArrayList<String> arr = socketToproxy(cmdLine2);
          try {
                 FileWriter fw = new FileWriter(f);
                 fw.write("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
                 for(String line : arr){
                        //if (line.contains("MESSAGE"))
                        //     continue;
                        fw.write(line);

                        //fw.flush();
                 }
                 fw.close();
                 DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
       //factory.setValidating(true);
                 DocumentBuilder dBuilder = factory.newDocumentBuilder();
                 doc = dBuilder.parse(f);


          } catch (Exception e) {
                 // TODO Auto-generated catch block
                 e.printStackTrace();
```

```
          }

          return doc;
}


public ArrayList<Host> getHosts(){

   ArrayList<Host> hosts=new ArrayList<Host>();
          Document doc = getXmlHost();
          Element hostPool=(Element)doc.getElementsByTagName("HOST_POOL").item(0);
          NodeList nlHost =  hostPool.getElementsByTagName("HOST");
          for(int i = 0;i<nlHost.getLength();i++){
                 if (!nlHost.item(i).getParentNode().getNodeName().equals("HOST_POOL"))
                        continue;
                 NodeList nlCluster=((Element)nlHost.item(i)).getElementsByTagName("TEMPLATE");
                 nlCluster=nlCluster.item(0).getChildNodes();
                 boolean cluster=false;
                 String cName="";
                 String hostName="";
                 for(int k=0;k < nlCluster.getLength();k++)
                 {
                        if (nlCluster.item(k).getNodeName().equals("CLUSTER"))
                        {
                               cName=nlCluster.item(k).getFirstChild().getTextContent();
                               if (!cName.equalsIgnoreCase(clusterName))
                                      break;
                               cluster=true;
                        }
                        else if (nlCluster.item(k).getNodeName().equals("HOSTNAME"))
                        {
                               hostName=nlCluster.item(k).getFirstChild().getTextContent();
                        }
                 }

                 if (!cluster) continue;

                 Element e= (Element)((Element) nlHost.item(i)).getElementsByTagName("ID").item(0);
                 String hId=e.getTextContent();
                 e=(Element)((Element) nlHost.item(i)).getElementsByTagName("STATE").item(0);
                 Host.HostStat hStat=getHostStat(e.getTextContent());
                 Host tempHost=new Host(1,hId,hostName,this.cloudName);
                 tempHost.setStat(hStat);
                 NodeList attribute=((Element)nlHost.item(i)).getElementsByTagName("HOST_SHARE").item(0).getChildNodes();
                 for(int j=0;j<attribute.getLength();j++)
                 {
                        e = (Element)((Element)attribute.item(j));
                        if (e.getNodeName().equals("USED_MEM"))
                         tempHost.setAMEM(Double.parseDouble(e.getTextContent()));
                        else if (e.getNodeName().equals("USED_CPU"))
                         tempHost.setACPU(Integer.parseInt(e.getTextContent()));
                        else if (e.getNodeName().equals("MAX_MEM"))
                         tempHost.setTMEM(Double.parseDouble(e.getTextContent()));
```

```
                        else if (e.getNodeName().equals("MAX_CPU"))
                            tempHost.setTCPU(Integer.parseInt(e.getTextContent()));
                        else if (e.getNodeName().equals("FREE_MEM"))
                            tempHost.setFMEM(Double.parseDouble(e.getTextContent()));
                        else if (e.getNodeName().equals("FREE_CPU"))
                            tempHost.setFCPU(Integer.parseInt(e.getTextContent()));

                    }
                            tempHost.setMaxVmNum(tempHost.getTCPU()/100);
                            hosts.add(tempHost);
                }
         getHostsRealTimeInfo(hosts);
                return hosts;
        }

    public Host.HostStat getHostStat(String stat){
        switch (stat){
        case "2":
                return Host.HostStat.ON;
        case "3":
                return Host.HostStat.ERR;
        case "4":
                return Host.HostStat.OFF;
        default:
                return Host.HostStat.UNKNOW;
        }
    }

    public void getHostCpuUtil(Host h){
        String cmd="sar 1 1|tail -1";
        ArrayList<String> arr=socketToproxy(cmd,h.getName());
        String[] subStr=arr.get(0).split("\\s+");
        h.setCPUUtil(100-Double.parseDouble(subStr[7]));
    }

    public void getHostIoUtil(Host h){
        String cmd="iostat -xk 1 1|tail -2";
        ArrayList<String> arr=socketToproxy(cmd,h.getName());
        String[] subStr=arr.get(0).split("\\s+");
        h.setIOUtil(Double.parseDouble(subStr[11]));
    }

    public void getHostsRealTimeInfo(ArrayList<Host> hlist){
        for(Host h:hlist){
                if(h.getStat()==Host.HostStat.ON){
                        try{
                        getHostCpuUtil(h);
                        getHostIoUtil(h);
                        }
                        catch(Exception e){
                                e.printStackTrace();
                        }
                }
```

```
        }
    }
}

-----------
-----------
-----------


File: project-fermi/plugins-src/vcluster/plugin/proxy/fermihtcondor/CheckCondor.java
/*
 * Hao: Need a function to deregister the slots in condor pool,
 *      Command: condor_off hostname
 */

package vcluster.plugin.proxy.fermihtcondor;


import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.StringTokenizer;

import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;

import vcluster.Vcluster;
import vcluster.elements.PoolStatus;
import vcluster.elements.QStatus;
import vcluster.elements.Slot;
import vcluster.elements.Job;
import vcluster.managers.CloudManager;

public class CheckCondor {


        private ArrayList<Slot> slots;
        private QStatus q;
        private PoolStatus pool;
        private ArrayList<Job> jobs;
        private String addr;
        private int port;
```

```
	public CheckCondor() {
		this.slots = new ArrayList<Slot> ();
		this.jobs = new ArrayList<Job> ();
		this.addr = Vcluster.CONDOR_IPADDR;
		this.port = Vcluster.PORTNUM;
		//setQ();
		//setPool();
		//System.out.println("condor instance has generated!");
	}


	public QStatus getQ() {
		this.setQ();
		q.setJobs(jobs);
		this.setJobHost();
		return q;
	}


	public PoolStatus getPool() {
	//	this.setPool();
		pool = new PoolStatus();
		getSlots();
		pool.setSlotList(slots);
		int claimed=0;
		int unclaimed=0;
		int total = pool.getSlotList().size();
		for(Slot s : pool.getSlotList()){
			if(s.getActivity()==1)claimed++;
		}
		unclaimed = total - claimed;
		pool.setTotalSlot(total);
		pool.setUnClaimedSlot(unclaimed);
		pool.setClaimedSlot(claimed);
		return pool;
	}


	public void setQ() {

		ArrayList<String> status = socketToproxy("condor_q");
		for(String str : status){
		 try {
			if(str.contains("Submitter")||str.contains("OWNER")||str.equals(""))
				continue;
		 if (str.contains("jobs")) {
			extractQInfo(str);
		 }
		 else
			extractJobInfo(str);
```

```
			} catch (Exception e) {
				// TODO Auto-generated catch block
				continue;
			}
		}
	}

	public void setJobHost(){
		ArrayList<String> status = socketToproxy("condor_q -run");
		for(String str : status){
		 try {
			if(str.contains("Submitter")||str.contains("OWNER")||str.equals(""))
				continue;
		 extractJobHost(str);
			} catch (Exception e) {
				// TODO Auto-generated catch block
				continue;
			}
		}
	}

	public boolean extractJobHost(String str){
StringTokenizer st = new StringTokenizer(str, ";, ");

		String jId="";
	    String hostName="";

		String token = null;

		if (!st.hasMoreTokens()) {
		System.out.println("ERROR: no more token.");
			return false;
		}

		/* get job ID */
		token = st.nextToken();

		jId=token;

		if (!st.hasMoreTokens()) {
		System.out.println("ERROR: no more token");
			return false;
		}

		/* get owner */
		token = st.nextToken();

		if (!st.hasMoreTokens()) {
		System.out.println("ERROR: no more token");
			return false;
		}

		/* get submitted time */
```

```
        token = st.nextToken();

        if (!st.hasMoreTokens()) {
        System.out.println("ERROR: no more token");
                return false;
        }

        token = st.nextToken();

        if (!st.hasMoreTokens()) {
        System.out.println("ERROR: no more token");
                return false;
        }

        /* get RUN Time */
        token = st.nextToken();

        if (!st.hasMoreTokens()) {
        System.out.println("ERROR: no more token");
                return false;
        }

        /* get HostName */
        token = st.nextToken();

        hostName = token;

        if(!q.setJobHost(jId, hostName))
        {
                System.out.println("[ERROR: ] CANNOT FIND HOST FOR RUNNING JOB");
                return false;
        }
        return true;
}
private String getDomain(String domainStr){
        String domain="";
        for(String cn : CloudManager.getCloudList().keySet()){
                if(domainStr.toLowerCase().contains(cn.toLowerCase())){
                        return cn;
                }
        }
        return domain;
}
public void setPool() {
        String cmd = "condor_status -total";
        ArrayList<String> status = socketToproxy(cmd);

        for(String str : status){
         try {
                        if (str.contains("Total")&&!str.contains("Owner")){
                                this.extractPoolInfo(str);
                                //System.out.println("hanlding pool extraction");
                        }
```

```
                } catch (Exception e) {
                        // TODO Auto-generated catch block
                        continue;
                }
        }
}


public boolean extractJobInfo(String qStatus)
{
 StringTokenizer st = new StringTokenizer(qStatus, ";, ");

        Job j=new Job();

        String token = null;

        if (!st.hasMoreTokens()) {
        System.out.println("ERROR: no more token.");
                return false;
        }

        /* get job ID */
        token = st.nextToken();

        j.setID(token);

        if (!st.hasMoreTokens()) {
        System.out.println("ERROR: no more token");
                return false;
        }

        /* get owner */
        token = st.nextToken();

        j.setOwner(token);

        if (!st.hasMoreTokens()) {
        System.out.println("ERROR: no more token");
                return false;
        }

        /* get submitted time */
        token = st.nextToken();

        if (!st.hasMoreTokens()) {
        System.out.println("ERROR: no more token");
                return false;
        }
        token=token + " " + st.nextToken();
        j.setSubmit(token);
        if (!st.hasMoreTokens()) {
        System.out.println("ERROR: no more token");
```

```
                return false;
        }

        /* get RUN Time */
        token = st.nextToken();

        j.setRunT(token);

        if (!st.hasMoreTokens()) {
        System.out.println("ERROR: no more token");
                return false;
        }

        /* get Status */
        token = st.nextToken();

        j.setStat(token);

        if (!st.hasMoreTokens()) {
        System.out.println("ERROR: no more token");
                return false;
        }

        /* get priority  */
        token = st.nextToken();

        j.setPri(token);

        if (!st.hasMoreTokens()) {
        System.out.println("ERROR: no more token");
                return false;
        }

        /* get size */
        token = st.nextToken();

        j.setSize(token);

        if (!st.hasMoreTokens()) {
        System.out.println("ERROR: no more token");
                return false;
        }
        jobs.add(j);
        return true;
}

public boolean extractQInfo(String qStatus)
{
        StringTokenizer st = new StringTokenizer(qStatus, ";, ");

        int numJob = 0;
        String token = null;
```

```
        if (!st.hasMoreTokens()) {
                System.out.println("ERROR: no more token.");
                return false;
        }

        /* get total jobs */
        token = st.nextToken();

        numJob = Integer.parseInt(token);
        if (!st.hasMoreTokens()) return false;
        if (!st.nextToken().equalsIgnoreCase("jobs")) {
                System.out.println("ERROR: token, jobs, expected, but not");
                return false;
        }
        int totalJob = numJob;

        if (!st.hasMoreTokens()) {
                System.out.println("ERROR: no more token");
                return false;
        }

        /* get completed jobs */
        token = st.nextToken();

        numJob = Integer.parseInt(token);

        if (!st.hasMoreTokens()) return false;
        if (!st.nextToken().equalsIgnoreCase("completed")) {
                System.out.println("ERROR: token, completed, expected, but not");
                return false;
        }
        int completedJob = numJob;



        if (!st.hasMoreTokens()) {
                System.out.println("ERROR: no more token");
                return false;
        }

        /* get removed jobs */
        token = st.nextToken();

        numJob = Integer.parseInt(token);

        if (!st.hasMoreTokens()) return false;
        if (!st.nextToken().equalsIgnoreCase("removed")) {
                System.out.println("ERROR: token, removed, expected, but not");
                return false;
        }
        int removedJob = numJob;

        if (!st.hasMoreTokens()) {
```

```
                System.out.println("ERROR: no more token");
                return false;
        }

        /* get idle jobs */
        token = st.nextToken();

        numJob = Integer.parseInt(token);

        if (!st.hasMoreTokens()) return false;
        if (!st.nextToken().equalsIgnoreCase("idle")) {
                System.out.println("ERROR: token, idle, expected, but not");
                return false;
        }
        int idleJob = numJob;

        if (!st.hasMoreTokens()) {
                System.out.println("ERROR: no more token");
                return false;
        }

        /* get running jobs */
        token = st.nextToken();
        numJob = Integer.parseInt(token);

        if (!st.hasMoreTokens()) return false;
        if (!st.nextToken().equalsIgnoreCase("running")) {
                System.out.println("ERROR: token, running, expected, but not");
                return false;
        }
        int runningJob = numJob;

        if (!st.hasMoreTokens()) {
                System.out.println("ERROR: no more token");
                return false;
        }

        /* get held jobs */
        token = st.nextToken();
        numJob = Integer.parseInt(token);

        if (!st.hasMoreTokens()) return false;
        if (!st.nextToken().equalsIgnoreCase("held")) {
                System.out.println("ERROR: token, held, expected, but not");
                return false;
        }
        int heldJob = numJob;

        if (!st.hasMoreTokens()) {
                System.out.println("ERROR: no more token");
                return false;
        }
```

```
        /* get suspended jobs */
        token = st.nextToken();
        numJob = Integer.parseInt(token);

        if (!st.hasMoreTokens()) return false;
        if (!st.nextToken().equalsIgnoreCase("suspended")) {
                System.out.println("ERROR: token, held, expected, but not");
                return false;
        }
        int suspendedJob = numJob;

        /* check the number of jobs */
        numJob = idleJob + runningJob + heldJob;
        q = new QStatus(totalJob, completedJob, removedJob, idleJob, runningJob, heldJob, suspendedJob);
        if (totalJob != numJob) {
                System.out.println("ERROR: Number of jobs does not mache, total="
                                +totalJob+", sum="+numJob);
                return false;
        }

        return true;
}

public boolean getSlots(){
        Document doc = getXmlPool();
        Element classads = (Element) doc.getElementsByTagName("classads").item(0);
        NodeList nl = classads.getElementsByTagName("c");
        for(int i = 0;i<nl.getLength();i++){
                Slot s = new Slot();
                Element e = (Element) nl.item(i);
                NodeList nolst = e.getElementsByTagName("a");
                for(int j = 0;j<nolst.getLength();j++){
                        Element ele = (Element) nolst.item(j);
                        String eleName = ele.getAttribute("n");
                        Element subele = (Element) ele.getElementsByTagName("s").item(0);
                        if(subele==null)continue;
                        //subele.get
                        String value = subele.getTextContent();
                        if(eleName.equalsIgnoreCase("Activity")){
                                if(value.equalsIgnoreCase("Idle")){
                                        s.setActivity(0);
                                }else{
                                        s.setActivity(1);
                                }
                        }else if(eleName.equalsIgnoreCase("Name")){
                                s.setHostName(value);
                                String domain = getDomain(value);
                                //System.out.println(value);
                                s.setDomain(domain);
                        }else if(eleName.equalsIgnoreCase("MyAddress")){
                                String regexString=".*(\\d{3}(\\.\\d{1,3}){3}).*";
                                String ip = value.replaceAll(regexString,"$1");
                                //System.out.println("ip");
```

```
                        s.setIdentifier(ip);
                }else if(eleName.equalsIgnoreCase("MyType")){
                        //System.out.println(value);
                }
            }
            s.setIdType(Slot.IdType.PRIVATEIP);
            slots.add(s);
        }

        return true;

    }

    public boolean extractPoolInfo(String poolStatus)
    {
        StringTokenizer st = new StringTokenizer(poolStatus, " \t");

        int numSlot = 0;
        String token = null;

        if (!st.hasMoreTokens()) {
            System.out.println("[Error] : no more token");
            return false;
        }

        /*
         *
         *         Total Owner Claimed Unclaimed Matched Preempting Backfill
         *
         *   Total    16     2       0        14       0          0        0
         */

        /* skip token Total */
        token = st.nextToken();
        // System.out.println("token = "+token);

        if (!st.hasMoreTokens()) return false;

        /* get total */
        token = st.nextToken();
        //System.out.println("Token = "+token);
        numSlot = Integer.parseInt(token);
        int totalSlot = numSlot;
        if (!st.hasMoreTokens()) return false;

        /* get owner */
        token = st.nextToken();
        //System.out.println("Token = "+token);
        numSlot = Integer.parseInt(token);
        int ownerSlot = numSlot;
        if (!st.hasMoreTokens()) return false;
```

```
        /* get claimed */
        token = st.nextToken();
        //System.out.println("Token = "+token);
        numSlot = Integer.parseInt(token);
        int claimedSlot = numSlot;
        if (!st.hasMoreTokens()) return false;


        /* get unclaimed */
        token = st.nextToken();
        //System.out.println("Token = "+token);
        numSlot = Integer.parseInt(token);
        int unClaimedSlot = numSlot;
        if (!st.hasMoreTokens()) return false;


        /* get matched */
        token = st.nextToken();
        //System.out.println("Token = "+token);
        numSlot = Integer.parseInt(token);
        int matchedSlot = numSlot;
        if (!st.hasMoreTokens()) return false;


        /* get preempting */
        token = st.nextToken();
        //System.out.println("Token = "+token);
        numSlot = Integer.parseInt(token);
        int preemptingSlot = numSlot;
        if (!st.hasMoreTokens()) return false;


        /* get backfill */
        token = st.nextToken();
        //System.out.println("Token = "+token);
        numSlot = Integer.parseInt(token);
        int backfillSlot = numSlot;
        //if (!st.hasMoreTokens()) return false;


        /* check the number of jobs */
        numSlot = ownerSlot + claimedSlot + unClaimedSlot + matchedSlot + preemptingSlot + backfillSlot;
        pool = new PoolStatus(totalSlot, ownerSlot, claimedSlot, unClaimedSlot, matchedSlot, preemptingSlot,
backfillSlot);
        if (totalSlot != numSlot) {
            System.out.println("[Error] : Number of jobs does not mache");
            System.out.println("        : Total = "+totalSlot+", Sum = "+numSlot);
            return false;
        }

        return true;
    }
```

```
/*
 * Function to deregister workers from the condor pool
 */
public boolean deregisterWorker(String dnsName){
	ArrayList<String> status = socketToproxy("condor_off "+dnsName);
	if(status.get(0).contains("Sent"))
		return true;
	else
		return false;
}

private ArrayList<String> socketToproxy(String cmd){
	String cmdLine=cmd+"\n";
	ArrayList<String> feedBack = new ArrayList<String>();
		Socket socket = null;
	 BufferedReader in = null;
	 DataOutputStream out = null;
	 //System.out.println(cmdLine);
	 try {
		socket = new Socket(addr, port);

		in = new BufferedReader(new InputStreamReader(socket.getInputStream(), "UTF-8"));
		out = new DataOutputStream(socket.getOutputStream());
		out.flush();
		/* make an integer to unsigned int */
		int userInput = 5;
		userInput <<= 8;
		userInput |=  1;
		userInput &= 0x7FFFFFFF;

		String s = Integer.toString(userInput);
		byte[] b = s.getBytes();

		out.write(b, 0, b.length);
		out.write(cmdLine.getBytes(), 0, cmdLine.getBytes().length);
		// out.flush();
		// out.close();
		 //char[] cbuf = new char[1024];
		   //String temp = null;
		   String str=null;
		   while((str=in.readLine())!=null)
		   {
			str=str.trim();
			feedBack.add(str);
		   }
		   /*while (in.read(cbuf, 0, 1024) != -1) {
		   //String str = new String(cbuf);
		   //str.replaceAll("\u0000", " ");
		    //str = str.trim();
		    String str=trimNullChar(cbuf);
		    str=str.trim();
		    if (!str.equals(temp)){
			//System.out.println(str);
```

```
		feedBack.add(str);
		    }

		   // cbuf[0] = '\0';
		   temp = str;
		}*/

	 } catch (UnknownHostException e) {
		System.out.print("ERROR: " +e.getMessage());
		closeStream(in, out, socket);
		return feedBack;
	 } catch (IOException e) {
		System.out.print("ERROR: " +e.getMessage());
		closeStream(in, out, socket);
		return feedBack;
	 }

	 closeStream(in, out, socket);
	 return feedBack;
}
private Document getXmlPool(){
	Document doc = null;

	File f = new File("condor.xml");
	/*try {
		FileReader fr=new FileReader(f);
		char[] buffer =new char[512];
		fr.read(buffer);
		for(char c:buffer){
		System.out.printf("U+%04x ", (int) c);
		}
		fr.close();
	} catch (FileNotFoundException e1) {
		// TODO Auto-generated catch block
		e1.printStackTrace();
	} catch (IOException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}*/
	//String cmdLine2 ="condor_status -xml -attributes Name,Activity,MyAddress";
	String cmdLine2 ="condor_status -xml";
	ArrayList<String> arr = socketToproxy(cmdLine2);
	try {
		FileWriter fw = new FileWriter(f);
		for(String line : arr){
			//line=line.trim();
			fw.write(line);
			fw.flush();
		}
		fw.close();
		DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    factory.setValidating(true);
		doc = factory.newDocumentBuilder().parse(f);
```

```
		} catch (Exception e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

		return doc;
	}
	private void closeStream(BufferedReader in, DataOutputStream out, Socket socket)
	{
		try {
		 if (in != null) in.close();
		 if (out != null) out.close();
		 if (socket != null) socket.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	/*
	 * Remove all the null chars in a string
	 */
	private String trimNullChar(String s){
		StringBuilder sb=new StringBuilder();
		for(char c:s.toCharArray())
		{
			if(c!='\0')
				sb.append(c);
		}
		return sb.toString();
	}
	private String trimNullChar(char [] cArray){
		StringBuilder sb=new StringBuilder();
		for(char c:cArray)
		{
			if(c!='\0')
				sb.append(c);
		}
		return sb.toString();
	}
}

-----------
-----------
-----------


File: project-fermi/plugins-src/vcluster/plugin/proxy/fermihtcondor/ProxyHTCondor.java
package vcluster.plugin.proxy.fermihtcondor;

import java.io.File;

import vcluster.elements.PoolStatus;
```

```
import vcluster.elements.QStatus;
import vcluster.plugInterfaces.BatchInterface;

public class ProxyHTCondor implements BatchInterface{



	@Override
	public boolean ConnectTo(File conf) {
		// TODO Auto-generated method stub
		return false;
	}


	@Override
	public PoolStatus getPoolStatus() {
		// TODO Auto-generated method stub
		return new CheckCondor().getPool();
	}


	@Override
	public QStatus getQStatus() {
		// TODO Auto-generated method stub
		return new CheckCondor().getQ();
	}


	@Override
	public boolean deregisterWorker(String dnsname) {
		// TODO Auto-generated method stub
		return new CheckCondor().deregisterWorker( dnsname);

	}

}

-----------
-----------
-----------


File: project-fermi/plugins-src/vcluster/plugin/proxy/htcondor/CheckCondor.java
package vcluster.plugin.proxy.htcondor;



import java.io.BufferedReader;

import java.io.DataOutputStream;

import java.io.File;
```

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.StringTokenizer;

import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;

import vcluster.Vcluster;
import vcluster.elements.PoolStatus;
import vcluster.elements.QStatus;
import vcluster.elements.Slot;
import vcluster.managers.CloudManager;

public class CheckCondor {



	private ArrayList<Slot> slots;
	private QStatus q;
	private PoolStatus pool;
```

```
	private String addr;
	private int port;

	public static void main(String[] arg) throws Exception{
		CheckCondor chc = new CheckCondor();

		chc.getSlots();
		for(Slot s : chc.slots){
			System.out.println(s.getDomain()+" : "+s.getIdentifier()+" : "+s.getActivity());
		}

	}

	public CheckCondor() {
		this.slots = new ArrayList<Slot> ();
		this.addr = Vcluster.CONDOR_IPADDR;
		this.port = Vcluster.PORTNUM;
		//setQ();
		//setPool();
		//System.out.println("condor instance has generated!");
	}


	public QStatus getQ() {
		this.setQ();
		return q;
```

```
}

public PoolStatus getPool() {
//      this.setPool();
        pool = new PoolStatus();
        getSlots();
        pool.setSlotList(slots);
        int claimed=0;
        int unclaimed=0;
        int total = pool.getSlotList().size();
        for(Slot s : pool.getSlotList()){
                if(s.getActivity()==1)claimed++;
        }
        unclaimed = total - claimed;
        pool.setTotalSlot(total);
        pool.setUnClaimedSlot(unclaimed);
        pool.setClaimedSlot(claimed);
        return pool;
}

public void setQ() {

        ArrayList<String> status = socketToproxy("condor_q");
        for(String str : status){
```

```
         try {
         if (str.contains("jobs")) {
                extractQInfo(str);
         }
                } catch (Exception e) {
                        // TODO Auto-generated catch block
                        continue;
                }
        }
}

private String getDomain(String domainStr){
        String domain="";
        for(String cn : CloudManager.getCloudList().keySet()){
                if(domainStr.toLowerCase().contains(cn.toLowerCase())){
                        return cn;
                }
        }
        return domain;
}
public void setPool() {
        String cmd = "condor_status -total";
        ArrayList<String> status = socketToproxy(cmd);

        for(String str : status){
         try {
```

```
                if (str.contains("Total")&&!str.contains("Owner")){
                        this.extractPoolInfo(str);
                        //System.out.println("hanlding pool extraction");
                }

            } catch (Exception e) {
                // TODO Auto-generated catch block
                continue;
            }
        }
    }


    public boolean extractQInfo(String qStatus)
    {
        StringTokenizer st = new StringTokenizer(qStatus, ";, ");

        int numJob = 0;
        String token = null;

        if (!st.hasMoreTokens()) {
            System.out.println("ERROR: no more token.");
            return false;
        }

        /* get total jobs */
```

```
        token = st.nextToken();

        numJob = Integer.parseInt(token);
        if (!st.hasMoreTokens()) return false;
        if (!st.nextToken().equalsIgnoreCase("jobs")) {
            System.out.println("ERROR: token, jobs, expected, but not");
            return false;
        }
        int totalJob = numJob;

        if (!st.hasMoreTokens()) {
            System.out.println("ERROR: no more token");
            return false;
        }

        /* get completed jobs */
        token = st.nextToken();

        numJob = Integer.parseInt(token);

        if (!st.hasMoreTokens()) return false;
        if (!st.nextToken().equalsIgnoreCase("completed")) {
            System.out.println("ERROR: token, completed, expected, but not");
            return false;
        }
        int completedJob = numJob;
```

```
if (!st.hasMoreTokens()) {
        System.out.println("ERROR: no more token");
        return false;
}

/* get removed jobs */
token = st.nextToken();

numJob = Integer.parseInt(token);

if (!st.hasMoreTokens()) return false;
if (!st.nextToken().equalsIgnoreCase("removed")) {
        System.out.println("ERROR: token, removed, expected, but not");
        return false;
}
int removedJob = numJob;

if (!st.hasMoreTokens()) {
        System.out.println("ERROR: no more token");
        return false;
}

/* get idle jobs */
```

```
token = st.nextToken();

numJob = Integer.parseInt(token);

if (!st.hasMoreTokens()) return false;
if (!st.nextToken().equalsIgnoreCase("idle")) {
        System.out.println("ERROR: token, idle, expected, but not");
        return false;
}
int idleJob = numJob;

if (!st.hasMoreTokens()) {
        System.out.println("ERROR: no more token");
        return false;
}

/* get running jobs */
token = st.nextToken();
numJob = Integer.parseInt(token);

if (!st.hasMoreTokens()) return false;
if (!st.nextToken().equalsIgnoreCase("running")) {
        System.out.println("ERROR: token, running, expected, but not");
        return false;
}
int runningJob = numJob;
```

```
        if (!st.hasMoreTokens()) {
                System.out.println("ERROR: no more token");
                return false;
        }

        /* get held jobs */
        token = st.nextToken();
        numJob = Integer.parseInt(token);

        if (!st.hasMoreTokens()) return false;
        if (!st.nextToken().equalsIgnoreCase("held")) {
                System.out.println("ERROR: token, held, expected, but not");
                return false;
        }
        int heldJob = numJob;

        if (!st.hasMoreTokens()) {
                System.out.println("ERROR: no more token");
                return false;
        }

        /* get suspended jobs */
        token = st.nextToken();
        numJob = Integer.parseInt(token);
```

```
        if (!st.hasMoreTokens()) return false;
        if (!st.nextToken().equalsIgnoreCase("suspended")) {
                System.out.println("ERROR: token, held, expected, but not");
                return false;
        }
        int suspendedJob = numJob;

        /* check the number of jobs */
        numJob = idleJob + runningJob + heldJob;
        q = new QStatus(totalJob, completedJob, removedJob, idleJob, runningJob, heldJob, suspendedJob);
        if (totalJob != numJob) {
                System.out.println("ERROR: Number of jobs does not mache, total="
                                +totalJob+", sum="+numJob);
                return false;
        }

        return true;
}

public boolean getSlots(){
        Document doc = getXmlPool();
        Element classads = (Element) doc.getElementsByTagName("classads").item(0);
        NodeList nl = classads.getElementsByTagName("c");
        for(int i = 0;i<nl.getLength();i++){
                Slot s = new Slot();
                Element e = (Element) nl.item(i);
```

```
NodeList nolst = e.getElementsByTagName("a");
for(int j = 0;j<nolst.getLength();j++){
	Element ele = (Element) nolst.item(j);
	String eleName = ele.getAttribute("n");
	Element subele = (Element) ele.getElementsByTagName("s").item(0);
	if(subele==null)continue;
	//subele.get
	String value = subele.getTextContent();
	if(eleName.equalsIgnoreCase("Activity")){
		if(value.equalsIgnoreCase("Idle")){
			s.setActivity(0);
		}else{
			s.setActivity(1);
		}
	}else if(eleName.equalsIgnoreCase("Name")){
		String domain = getDomain(value);
		//System.out.println(value);
		s.setDomain(domain);
	}else if(eleName.equalsIgnoreCase("MyAddress")){
		String regexString=".*(\\d{3}(\\.\\d{1,3}){3}).*";
		String ip = value.replaceAll(regexString,"$1");
		//System.out.println("ip");
		s.setIdentifier(ip);
	}else if(eleName.equalsIgnoreCase("MyType")){
		//System.out.println(value);
	}
```

```
		}
		s.setIdType(Slot.IdType.PRIVATEIP);
		slots.add(s);
	}

	return true;

}

public boolean extractPoolInfo(String poolStatus)
{
	StringTokenizer st = new StringTokenizer(poolStatus, " \t");

	int numSlot = 0;
	String token = null;

	if (!st.hasMoreTokens()) {
		System.out.println("[Error] : no more token");
		return false;
	}

	/*
	 *
	 *		Total Owner Claimed Unclaimed Matched Preempting Backfill
	 *
	 *	Total	16	2	0	14	0	0	0
```

```
 */

/* skip token Total */
token = st.nextToken();
// System.out.println("token = "+token);

if (!st.hasMoreTokens()) return false;

/* get total */
token = st.nextToken();
//System.out.println("Token = "+token);
numSlot = Integer.parseInt(token);
int totalSlot = numSlot;
if (!st.hasMoreTokens()) return false;

/* get owner */
token = st.nextToken();
//System.out.println("Token = "+token);
numSlot = Integer.parseInt(token);
int ownerSlot = numSlot;
if (!st.hasMoreTokens()) return false;


/* get claimed */
token = st.nextToken();
//System.out.println("Token = "+token);
```

```
numSlot = Integer.parseInt(token);
int claimedSlot = numSlot;
if (!st.hasMoreTokens()) return false;


/* get unclaimed */
token = st.nextToken();
//System.out.println("Token = "+token);
numSlot = Integer.parseInt(token);
int unClaimedSlot = numSlot;
if (!st.hasMoreTokens()) return false;


/* get matched */
token = st.nextToken();
//System.out.println("Token = "+token);
numSlot = Integer.parseInt(token);
int matchedSlot = numSlot;
if (!st.hasMoreTokens()) return false;


/* get preempting */
token = st.nextToken();
//System.out.println("Token = "+token);
numSlot = Integer.parseInt(token);
int preemptingSlot = numSlot;
```

```
		if (!st.hasMoreTokens()) return false;



		/* get backfill */
		token = st.nextToken();
		//System.out.println("Token = "+token);
		numSlot = Integer.parseInt(token);
		int backfillSlot = numSlot;
		//if (!st.hasMoreTokens()) return false;



		/* check the number of jobs */
		numSlot = ownerSlot + claimedSlot + unClaimedSlot + matchedSlot + preemptingSlot + backfillSlot;
		pool = new PoolStatus(totalSlot, ownerSlot, claimedSlot, unClaimedSlot, matchedSlot, preemptingSlot,
backfillSlot);
		if (totalSlot != numSlot) {
			System.out.println("[Error] : Number of jobs does not mache");
			System.out.println("        : Total = "+totalSlot+", Sum = "+numSlot);
			return false;
		}

		return true;
	}

	private ArrayList<String> socketToproxy(String cmd){
		String cmdLine=cmd;
		ArrayList<String> feedBack = new ArrayList<String>();
```

```
			Socket socket = null;
		BufferedReader in = null;
		DataOutputStream out = null;
		//System.out.println(cmdLine);
		try {
			socket = new Socket(addr, port);

		    in = new BufferedReader(new InputStreamReader(socket.getInputStream(), "UTF-8"));
		    out = new DataOutputStream(socket.getOutputStream());
		    out.flush();
		    /* make an integer to unsigned int */
		    int userInput = 5;
		    userInput <<= 8;
		    userInput |=  1;
		    userInput &= 0x7FFFFFFF;

		    String s = Integer.toString(userInput);
		    byte[] b = s.getBytes();

		    out.write(b, 0, b.length);
		    out.write(cmdLine.getBytes(), 0, cmdLine.getBytes().length);
		   // out.flush();
		   // out.close();
		    char[] cbuf = new char[1024];
			String temp = null;
			while (in.read(cbuf, 0, 1024) != -1) {
```

```
                String str = new String(cbuf);
                 str = str.trim();
                 if (!str.equals(temp)){
                        //System.out.println(str);
                         feedBack.add(str);
                 }

                 cbuf[0] = '\0';
                temp = str;
            }

        } catch (UnknownHostException e) {
                System.out.print("ERROR: " +e.getMessage());
            closeStream(in, out, socket);
            return feedBack;
        } catch (IOException e) {
                System.out.print("ERROR: " +e.getMessage());
            closeStream(in, out, socket);
            return feedBack;
        }

        closeStream(in, out, socket);
        return feedBack;
}
private Document getXmlPool(){
       Document doc = null;
```

```
       File f = new File("condor.xml");
       String cmdLine2 ="condor_status -xml -attributes Name,MyAddress,Activity";
       ArrayList<String> arr = socketToproxy(cmdLine2);
       try {
              FileWriter fw = new FileWriter(f);
              for(String line : arr){
                     line=line.trim();
                     fw.write(line);
                     fw.flush();
              }
              fw.close();
              DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
     factory.setValidating(true);
              doc = factory.newDocumentBuilder().parse(f);

       } catch (Exception e) {
              // TODO Auto-generated catch block
              e.printStackTrace();
       }

       return doc;
}
private void closeStream(BufferedReader in, DataOutputStream out, Socket socket)
{
       try {
        if (in != null) in.close();
```

```
                if (out != null) out.close();
                if (socket != null) socket.close();
               } catch (IOException e) {
                      e.printStackTrace();
               }
        }
}

-----------
-----------
-----------

File: project-fermi/plugins-src/vcluster/plugin/proxy/htcondor/ProxyHTCondor.java
package vcluster.plugin.proxy.htcondor;


import java.io.File;


import vcluster.elements.PoolStatus;
import vcluster.elements.QStatus;
import vcluster.plugInterfaces.BatchInterface;


public class ProxyHTCondor implements BatchInterface{




        @Override
        public boolean ConnectTo(File conf) {
               // TODO Auto-generated method stub
               return false;



        }


        @Override
        public PoolStatus getPoolStatus() {
               // TODO Auto-generated method stub
               return new CheckCondor().getPool();
        }


        @Override
        public QStatus getQStatus() {
               // TODO Auto-generated method stub
               return new CheckCondor().getQ();
        }


        @Override
        public boolean deregisterWorker(String dnsname) {
               // TODO Auto-generated method stub
               return false;
        }

}

-----------
-----------
-----------
```

File: project-fermi/plugins-src/vcluster/plugin/proxy/opennebula/ProxyOpennebula.java

```
package vcluster.plugin.proxy.opennebula;


import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.DataOutputStream;

import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

import java.io.InputStreamReader;

import java.net.Socket;

import java.net.UnknownHostException;

import java.util.ArrayList;

import java.util.List;

import java.util.StringTokenizer;


import vcluster.elements.Host;

import vcluster.elements.Vm;

import vcluster.managers.VmManager.VMState;

import vcluster.plugInterfaces.CloudInterface;


public class ProxyOpennebula implements CloudInterface {


       public static void main(String[] arg){


           String cmdLine = "";
```

```
           ProxyOpennebula proxyOpennebula = new ProxyOpennebula();

           //proxyOpennebula.RegisterCloud(new ArrayList<String>());

           proxyOpennebula.addr="fermicloudpp-ui.fnal.gov";

           proxyOpennebula.port=9734;

           File f = new File("e:"+File.separator+"vmlist.txt");

           /* prompt */

          do{

                   System.out.print("proxyTest > ");


                   InputStreamReader input = new InputStreamReader(System.in);

                   BufferedReader reader = new BufferedReader(input);


                   try {

                          /* get a command string */

                      cmdLine = reader.readLine();

                      //reader.close();

                      System.out.println(cmdLine);

                   }

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

                  ArrayList<String> feedback = proxyOpennebula.socketToproxy(cmdLine);


                  if(!f.exists()){

                         try {

                             f.createNewFile();

                      } catch (IOException e) {

                             // TODO Auto-generated catch block
```

```
                    e.printStackTrace();
                }
            }
            try {
                FileWriter fw=new FileWriter(f);
                BufferedWriter bw=new BufferedWriter(fw);
                for(String line : feedback){
                        System.out.println(line);
                        bw.write(line);
                    bw.newLine();//
                }
                bw.close();
                fw.close();

        } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }


    }while(!cmdLine.equals("quit"));
}

private static void closeStream(BufferedReader in, DataOutputStream out, Socket socket)
{
        try {
```

```
         if (in != null) in.close();
         if (out != null) out.close();
         if (socket != null) socket.close();
        } catch (IOException e) {
                e.printStackTrace();
        }
}

private ArrayList<String> socketToproxy(String cmd){
        String cmdLine=cmd;
        ArrayList<String> feedBack = new ArrayList<String>();
         Socket socket = null;
         BufferedReader in = null;
         DataOutputStream out = null;
         //System.out.println(cmdLine);
         try {
                socket = new Socket(addr, port);

             in = new BufferedReader(new InputStreamReader(socket.getInputStream(), "UTF-8"));
             out = new DataOutputStream(socket.getOutputStream());
             out.flush();
             /* make an integer to unsigned int */
             int userInput = 5;
             userInput <<= 8;
             userInput |=  1;
             userInput &= 0x7FFFFFFF;
```

```
            String s = Integer.toString(userInput);
            byte[] b = s.getBytes();

            out.write(b, 0, b.length);
            out.write(cmdLine.getBytes(), 0, cmdLine.getBytes().length);
            out.flush();
            char[] cbuf = new char[1024];
              String temp = null;
              while (in.read(cbuf, 0, 1024) != -1) {
              String str = new String(cbuf);
               str = str.trim();
               if (!str.equals(temp)){
                     //System.out.println(str);
                      feedBack.add(str);
               }

               cbuf[0] = '\0';
              temp = str;
            }

        } catch (UnknownHostException e) {
              System.out.print("ERROR: " +e.getMessage());
            closeStream(in, out, socket);
            return feedBack;
        } catch (IOException e) {
```

```
              System.out.print("ERROR: " +e.getMessage());
            closeStream(in, out, socket);
            return feedBack;
        }

        closeStream(in, out, socket);
        return feedBack;
}


@Override
public boolean RegisterCloud(List<String> configurations) {
       // TODO Auto-generated method stub
       /*configurations.add("username=amol");
       configurations.add("endpoint=fcl301.fnal.gov");
       configurations.add("port=9734");
       configurations.add("template = OpenNebula/clean.one");*/
       for(String aLine : configurations){

              StringTokenizer st = new StringTokenizer(aLine, "=");

              if (!st.hasMoreTokens()) return false;

              /* get a keyword */
              String aKey = st.nextToken().trim();
```

```
		/* get a value */
		if (!st.hasMoreTokens()) return false;

		String aValue = st.nextToken().trim();

		if (aKey.equalsIgnoreCase("username")) {
		} else if (aKey.equalsIgnoreCase("endpoint"))
			this.addr = aValue;
		else if (aKey.equalsIgnoreCase("port"))
			this.port = Integer.parseInt(aValue);
		else if (aKey.equalsIgnoreCase("template")){
			this.template = aValue;
		}
		else if (aKey.equalsIgnoreCase("ipmiParas")){
			this.ipmiParas = aValue;
		}

	}
	ArrayList<String> dateR = socketToproxy("date -R");
	//if(dateR.get(0).split(regex))

	try {
		if(dateR!=null&&!dateR.isEmpty()) {
		}
	} catch (NumberFormatException e) {
		// TODO Auto-generated catch block
```

```
		e.printStackTrace();
	}

	return true;
}

@Override
public ArrayList<Vm> createVM(int maxCount) {
	// TODO Auto-generated method stub
	String cmdLine="onevm create "+template +" -m "+maxCount;
	System.out.println(cmdLine);
	String hostName = template.replace(".one", "");
	ArrayList<Vm> vmList = new ArrayList<Vm>();
	ArrayList<String> feedBack = socketToproxy(cmdLine);
	if(feedBack!=null&&!feedBack.isEmpty()&&feedBack.get(0).contains("ID:")){
		for(int i = 0;i<feedBack.size();i++){

			String [] vmEle = feedBack.get(i).split("\\s+");
			Vm vm = new Vm();
			vm.setId(vmEle[1]);
			vm.setState(VMState.PROLOG);
			vm.setHostname(hostName);
			vmList.add(vm);
		}
	}else{
		System.out.println(feedBack.get(0));
```

```
                return null;
            }
                return vmList;
        }

        @Override
        public ArrayList<Vm> createVM(int maxCount, String hostID)
        {
            ArrayList<Vm> vmList = new ArrayList<Vm>();

            return vmList;
        }
        private boolean getVminf(Vm vm){
            ArrayList<String> feedBack = socketToproxy("onevm show "+vm.getId()+" | grep IP");



            if(feedBack!=null&&!feedBack.isEmpty()){

                for(String str : feedBack){
                        try {
                            if(str.contains("IP_PUBLIC")){
                                vm.setPubicIP(str.split("=")[1].replace(",", "").replaceAll("\"",
"").trim());
                            }else if(str.contains("IP=")){
                                vm.setPrivateIP(str.split("=")[1].trim().replace(",", "").replaceAll("\"",
""));
                            }
```

```
                            else if(str.contains("IP_PRIVATE")){
                                vm.setPrivateIP(str.split("=")[1].replace(",", "").replaceAll("\"", ""));
                            }
                        } catch (Exception e) {
                            // TODO Auto-generated catch block
                            return false;
                        }
                    }
                return true;
            }
            return true;
        }

        @Override
        public ArrayList<Vm> listVMs() {
            // TODO Auto-generated method stub
            String cmdLine="onevm list";
            ArrayList<Vm> vmList = new ArrayList<Vm>();
            ArrayList<String> feedBack = socketToproxy(cmdLine);
            boolean flag = true;
            while(flag){
            if(feedBack!=null&&!feedBack.isEmpty()&&feedBack.get(0).contains("ID")){
                for(int i = 1;i<feedBack.size();i++){
                    //System.out.println(feedBack.get(i));
                    String [] vmEle = feedBack.get(i).split("\\s+");
                    if(vmEle.length<9){
```

```
                        continue;
                }
                Vm vm = new Vm();
                try{
                        vm.setId(vmEle[0]);
                        //vm.setState(vmEle[4]);
                        getVminf(vm);
                        if(vmEle[4].equalsIgnoreCase("runn")){
                                vm.setState(VMState.RUNNING);
                        }else if(vmEle[4].equalsIgnoreCase("stop")){
                                vm.setState(VMState.STOP);
                        }else if(vmEle[4].equalsIgnoreCase("Pend")){
                                vm.setState(VMState.PENDING);
                        }else if(vmEle[4].equalsIgnoreCase("Prol")){
                                vm.setState(VMState.PROLOG);
                        }else if(vmEle[4].equalsIgnoreCase("Susp")){
                                vm.setState(VMState.SUSPEND);
                        }else{
                                vm.setState(VMState.NOT_DEFINED);
                        }
                        if(vm.getState()==VMState.STOP){vm.setHostname("");}else{
                                vm.setHostname(vmEle[7]);
                        }
                }catch(Exception e){
                        e.printStackTrace();
                        continue;
```

```
                }

                vmList.add(vm);

        }
        flag = false;
    }else if(feedBack.get(0).contains("ReadTimeout")){
        try {
                Thread.sleep(5000);
        } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
        System.out.println("ReadTimeout, waiting 5s... ...");

    }else{
        System.out.println(feedBack.get(0));
        flag = false;
        return null;
    }
    }
    //System.out.println("opennebula plugin:"+vmList.size());
    return vmList;
}

@Override
```

```
public ArrayList<Vm> destroyVM(Vm vm) {
	// TODO Auto-generated method stub

	String cmdLine = "./rmvm "+vm.getPrivateIP()+" "+vm.getId();
	ArrayList<Vm> vmList = new ArrayList<Vm>();
	ArrayList<String> feedBack = socketToproxy(cmdLine);
	if(feedBack!=null&&!feedBack.isEmpty()){
		System.out.println(feedBack.get(0));
		return null;
	}
	Vm vm_1 = new Vm();
	vm_1.setId(vm.getId());
	vm_1.setState(VMState.STOP);
	vmList.add(vm_1);
	return vmList;
}

@Override
public ArrayList<Vm> startVM(String id) {
	// TODO Auto-generated method stub
	String cmdLine="onevm resume "+id;
	ArrayList<Vm> vmList = new ArrayList<Vm>();
	ArrayList<String> feedBack = socketToproxy(cmdLine);
	if(feedBack!=null&&!feedBack.isEmpty()){
		System.out.println(feedBack.get(0));
		return null;
```

```
	}
	Vm vm = new Vm();
	vm.setId(id);
	vm.setState(VMState.PROLOG);
	vmList.add(vm);
	return vmList;
}

@Override
public ArrayList<Vm> suspendVM(String id) {
	// TODO Auto-generated method stub
	String cmdLine = "onevm suspend "+id;
	ArrayList<Vm> vmList = new ArrayList<Vm>();
	ArrayList<String> feedBack = socketToproxy(cmdLine);
	if(feedBack!=null&&!feedBack.isEmpty()){
		System.out.println(feedBack.get(0));
		return null;
	}
	Vm vm = new Vm();
	vm.setId(id);
	vm.setState(VMState.SUSPEND);
	vmList.add(vm);
	return vmList;
}
```

```
@Override
public boolean migrate(String vmid, String hostid) {
        // TODO Auto-generated method stub
        String cmdLine = "onevm migrate "+ vmid + " "+ hostid;

    socketToproxy(cmdLine);

    return true;
}

@Override
public boolean hoston(String ipmiID) {
        // TODO Auto-generated method stub
        String cmdLine = "ipmitool" + ipmiParas + " -H "+ipmiID+ " power on";

    ArrayList<String> feedback = socketToproxy(cmdLine);
    if(feedback.get(0).equalsIgnoreCase("Chassis Power Control: Up/On")){

        return true;
    }else{
        return false;
    }
}
```

```
@Override
public boolean hostoff(String ipmiID) {
        // TODO Auto-generated method stub
        String cmdLine = "ipmitool" + ipmiParas + " -H "+ipmiID+ " power off";

    ArrayList<String> feedback = socketToproxy(cmdLine);
    if(feedback.get(0).equalsIgnoreCase("Chassis Power Control: Down/Off")){
        return true;
    }
        return false;
}

private String addr;
private int port;
private String template;
private String ipmiParas;
@Override
public ArrayList<Host> listHost() {
        // TODO Auto-generated method stub
        return null;
}

}
```

```
-----------
-----------
-----------


File: project-fermi/README.md
project
=======

vcluster project

Source code for the vcluster in FermiCloud

Update 8-14-2014

1. Overhead aware best fit algorithm --DONE
2. VM release pattern -- need config file --DONE
3. Model traning --DONE
4. Readings from worker nodes --DONE
5. Auto-launch based on model --DONE

Update 7-28-2014

1. BUG for empty plugman command -- fixed
2. VM name for EC2 VMs -- fixed
3. Bugs for empty job queue -- fixed
4. VM overhead model:
   1. Host configuration read -- DONE
   2. Host configuration write -- DONE
   3. Host parameters check -- DONE
   4. Model calculation -- DONE
   5. Overhead aware best fit algorithm
   6. Host benchmarking --DONE
   7. Host constructor --DONE
   8. Model test --DONE
   9. VM release pattern -- need config file
   10. Model traning
   11. Readings from worker nodes
5. Auto-launch based on model
6. Bugs on proxy server. Read redundant messages from the client side. --FIXED

Update 7-18-2014

Add Functions

1. Enable list host infomation from private cloud
2. Enable getting real time infomation for hosts (CPU Utilization, IO Utilization)
3. Match VMs to Hosts, counting running VMs. (Readings from OpenNebula are not accurate)
4. When load hosts, filters used to select hosts from given cluster
5. Enable list condor jobs in details (Owner, Start time, Status, Slot, etc.)
6. EC2 feature:
   a. Lable worker nodes when instantiate VM on EC2
   b. Describe only vCluster worker nodes
```

```
   c. "listhost, hostlist" commands only show hosts informations


-----------
-----------
-----------


File: project-fermi/src/vcluster/elements/Cloud.java
package vcluster.elements;

import java.util.ArrayList;
import java.util.List;
import java.util.StringTokenizer;
import java.util.TreeMap;

import vcluster.Vcluster;
import vcluster.executors.BatchExecutor;
import vcluster.executors.PlugmanExecutor;
import vcluster.managers.CloudManager;
import vcluster.managers.PluginManager;
import vcluster.managers.VmManager;
import vcluster.managers.VmManager.VMState;
import vcluster.plugInterfaces.CloudInterface;
import vcluster.ui.CmdComb;
import vcluster.elements.Host;


/**
 *A class representing a cloud
 *
 */
public class Cloud{

	/**
	 *The constructor without any member initiation.
	 *
	 */
	public Cloud() {
		// TODO Auto-generated constructor stub
	}
	/**
	 *The constructor, by using this constructor some of the member value will be initiated.
	 *@param conf, a List of the cloud configuration value. Such as the cloud name, interface name, etc.
	 *
	 */
	public Cloud(List<String> conf) {
		this.conf = conf;
		for(String aLine : conf){

			StringTokenizer st = new StringTokenizer(aLine, "=");

			if (!st.hasMoreTokens()) break;
```

```
                        /* get a keyword */
                        String aKey = st.nextToken().trim();

                        /* get a value */
                        if (!st.hasMoreTokens()) break;

                        String aValue = st.nextToken().trim();

                        if (aKey.equalsIgnoreCase("type")){
                                setCloudType(aValue);

                        }else if((aKey.equalsIgnoreCase("Interface"))){
                                setCloudpluginName(aValue);
                                cp = (CloudInterface)PluginManager.pluginList.get(cloudpluginName).getInstance();
                        }else if((aKey.equalsIgnoreCase("Name"))){
                                //System.out.println("name");
                                setCloudName(aValue);
                        //      System.out.println(aValue);
                        }else if(aKey.equalsIgnoreCase("hosts")){
                        //      System.out.println(aValue);
                                hostList = new TreeMap<String,Host> ();
                                String [] hostlist = aValue.split(",");
                                //System.out.println("host");
                                for(int i = 0 ; i<hostlist.length;i++){
                                        String [] hostStr = hostlist[i].split("/");
                                        String hostId = hostStr[0];
                                        String hostname = hostStr[1];
                                        String MaxVMNum = hostStr[2];
                                        String ipmiID = hostStr[3];
                                        Host host = new Host(Integer.parseInt(MaxVMNum),hostId,hostname,this.cloudName);
                                        host.setIpmiID(ipmiID);
                                        //System.out.println(host.getId()+ i + "");
                                        hostList.put(hostname,host);
                                }

                        }else if (aKey.equalsIgnoreCase("imagesize")){
                                setImageSize(Integer.parseInt(aValue));
                        }else if (aKey.equalsIgnoreCase("cluster")){
                                setCluster(aValue);
                        }
                }
                isLoaded = false;

        }

        /**
         *This function is for loading a cloud into vcluster.
         *Inovke this function the vcluster would connect the corresponding cloud and get the vms'
         *information.
         */
        public String load(){
                StringBuffer str = new StringBuffer();
```

```
                if(cloudName==null||cloudType==null||cloudpluginName==null){
                        return null;
                }
                if(!PluginManager.isLoaded(cloudpluginName))PlugmanExecutor.load(new CmdComb("plugman load -c
"+cloudpluginName));
                cp = (CloudInterface)PluginManager.pluginList.get(cloudpluginName).getInstance();
                this.listVMs();
                this.listHost();
                matchVMtoHost();
                if(getVmList()==null)return null;
                for(Vm vm : getVmList().values()){
                        Integer id = new Integer(VmManager.getcurrId());
                        VmManager.getVmList().put(id, vm);
                }

                CloudManager.setCurrentCloud(this);
                isLoaded = true;
                String fName = String.format("%-12s", getCloudName());
                String fInterface =String.format("%-20s", getCloudpluginName());
                String fType = String.format("%-12s", getCloudType());
                String fVMs = String.format("%-8s", vmList.size());
                String fHosts = String.format("%-8s", hostList.size());
                str.append(fName+fInterface+fType+fVMs+fHosts);
                //HandleXML.setCloudAttribute(cloudName,"isLoaded", "true");
                //System.out.println(str);
                return str.toString();
        }

        public int getCurrentVMs() {
                return currentVMs;
        }

        public void setCurrentVMs(int vms) {
                currentVMs = vms;
        }

        protected void incCurrentVMs(int vms) {
                currentVMs += vms;
        }

        /**
         *Get the configurations of the cloud
         *@return A list of configurations, such as cloud name, interface name connection configurations.
         */
        public List<String> getConf() {
                return conf;
        }

        /**
         *Set the configurations of the cloud
         *@param conf, a list a the configurations.
         */
        public void setConf(List<String> conf) {
```

```
		this.conf = conf;
	}

	public void dump(){
		for(String aLine : conf){
			System.out.println(aLine);
		}
	}

	/**
	 * To create a virtual machine.
	 * Through the fucntion creates specified number of virtual machines on the specific host of the cloud.
	 * @param maxCount, the number of virtual machines that you want to create.
	 * @param hostId, the host ID where you want to create vms on.
	 */
	public String createVM(int maxCount,String hostName) {
		// TODO Auto-generated method stub
		StringBuffer str = new StringBuffer();
		/*if(!hostName.equalsIgnoreCase("host1")){
			int i = 100;
			for(int j =0;j<conf.size();j++){
				if(conf.get(j).contains("template")){
					i = j;
					break;
				}
			}
			//if(i!=100)conf.set(i, "template = templates/"+hostName+".one");
		}*/
		cp.RegisterCloud(conf);
		ArrayList<Vm> vmlist = cp.createVM(maxCount);
		if(vmlist==null || vmlist.isEmpty()){
			str.append("Operation failed!");
			return str.toString();
		}
		for(Vm vm : vmlist){
			vm.setCloudName(cloudName);
			vm.setHostname(hostName);
			this.vmList.put(vm.getId(), vm);
			Integer uId = VmManager.getcurrId();
			vm.setuId(uId);
			VmManager.getVmList().put(uId, vm);
			str.append(cloudName+"   "+vm.getId()+"   "+vm.getDNSName()+System.getProperty("line.separator"));
		}
		if(maxCount==1){
			str.append(maxCount + " virture machine has been created
successfully"+System.getProperty("line.separator"));
		}else{
			str.append(maxCount +" virture machines have been created
successfully"+System.getProperty("line.separator"));
		}
		//System.out.println(str);
		Vcluster.writeSysLogFile(str.toString());
		return str.toString();
```

```
	}

	public String createVMonHost(int maxCount, String hostID){
		StringBuffer str = new StringBuffer();
		cp.RegisterCloud(conf);
		ArrayList<Vm> vmlist = cp.createVM(maxCount,hostID);
		if(vmlist==null || vmlist.isEmpty()){
			str.append("Operation failed!");
			return str.toString();
		}
		for(Vm vm : vmlist){
			vm.setCloudName(cloudName);
			//vm.setHostname(hostName);
			this.vmList.put(vm.getId(), vm);
			Integer uId = VmManager.getcurrId();
			vm.setuId(uId);
			VmManager.getVmList().put(uId, vm);
			str.append(cloudName+"   "+vm.getId()+"   "+vm.getDNSName()+System.getProperty("line.separator"));
		}
		if(maxCount==1){
			str.append(maxCount + " virture machine has been created
successfully"+System.getProperty("line.separator"));
		}else{
			str.append(maxCount +" virture machines have been created
successfully"+System.getProperty("line.separator"));
		}
		//System.out.println(str);
		return str.toString();
	}
	/**
	 *List up the virtual machine that is running on the cloud.
	 *
	 */
	public boolean listVMs() {
		// TODO Auto-generated method stub
		cp.RegisterCloud(conf);
	//	HashMap<String,VMelement> vms = new HashMap<String,VMelement>();
		ArrayList<Vm> cVmList = cp.listVMs();
		//System.out.println(cVmList.size());
		vmList = new TreeMap<String,Vm>();
		//int i = 1;
		if(cVmList==null||cVmList.size()==0)return false;
		for(Vm vm : cVmList){
			if(this.cloudType==CloudType.PUBLIC){
				vm.setHostname("host1");
			}
			vm.setCloudName(getCloudName());
			vmList.put(vm.getId(), vm);
		}
		/*
		 * Why need to mapping the activity at the time load the cloud?
		 */
		//BatchExecutor.mapingActivityToVm();
```

```
        return true;
}

/**
 * Terminate the given virtual machine on the cloud.
 * @param id, the given virtual machine's id.
 */
public boolean destroyVM(Vm vm) {
        // TODO Auto-generated method stub
        cp.RegisterCloud(conf);
        cp.destroyVM(vm);

         return true;
}

/**
 *Resume a given virtual machine form suspend.
 *@param id, the given virtual machine's id.
 *@return boolean.
 */
public boolean startVM(String id) {
        // TODO Auto-generated method stub
        cp.RegisterCloud(conf);
        ArrayList<Vm> vmlist = cp.startVM(id);

        if(vmlist==null || vmlist.isEmpty()){
                System.out.println("Operation failed!");
                return false;
        }
        for(Vm vm : vmlist){
                vmList.get(vm.getId()).setState(vm.getState());
                System.out.println(cloudName+"   "+vm.getId()+"   "+vm.getState());
        }
        return true;
}

/**
 * Suspend a given virtual machine from running.
 * @param id, the virtual machine's id.
 *
 */

public boolean suspendVM(String id) {
        // TODO Auto-generated method stub
        cp.RegisterCloud(conf);
        ArrayList<Vm> vmlist = cp.suspendVM(id);
        if(vmlist==null || vmlist.isEmpty()){
                System.out.println("Operation failed!");
                return false;
        }
        for(Vm vm : vmlist){
                vmList.get(vm.getId()).setState(vm.getState());
                System.out.println(cloudName+"   "+vm.getId()+"   "+vm.getState());
```

```
        }
         return true;
}

/**
 *Get the type of the cloud :private or public.
 *@return cloudType.
 */
public CloudType getCloudType() {

        return cloudType;
}

/**
 * Set the type of a cloud
 * @see class CloudType.
 * @param cloudType,private or public.
 */
public void setCloudType(CloudType type) {
        cloudType = type;
}

public void setCloudType(String type) {
        if (type.equalsIgnoreCase("private"))
                cloudType = CloudType.PRIVATE;
        else if (type.equalsIgnoreCase("public"))
                cloudType = CloudType.PUBLIC;
        else {
                System.out.println("undefined type, "+type+", found");
                cloudType = CloudType.NOT_DEFINED;
        }

}

public String stringCloudType() {

        switch(cloudType) {
        case PRIVATE: return "PRIVATE";
        case PUBLIC: return "PUBLIC";
        case NOT_DEFINED: return "NOT_DEFINED";
        }
        return "NOT_DEFINED";
}

/**
 *Get the cloud name .
 *@return cloudName.
 */
public String getCloudName() {
        return cloudName;
}

/**
```

```
 *Set the cloud name.
 *@param cloudName , the name of the cloud.
 */
public void setCloudName(String cloudName) {
	this.cloudName = cloudName;
}


/**
 * Get the name of cloud plugin
 * @return cloudpluginName, a string of the cloud plugin name.
 */
public String getCloudpluginName() {
	return cloudpluginName;
}

/**
 * Set the name of cloud plugin
 * @param cloudpluginName, The name of cloud plugin
 */
public void setCloudpluginName(String cloudpluginName) {
	this.cloudpluginName = cloudpluginName;
}



/**
 * Get the virtual machines list
 * @return the instances collection of virtual machines
 *
 */
public TreeMap<String,Vm> getVmList() {
	return vmList;
}

/**
 * Set the virtual machines list
 * @param vmList, a TreeMap collection of virtual machine's instances,the key set is virtual machines' id
 */
public void setVmList(TreeMap<String, Vm> vmList) {
	this.vmList = vmList;
}



/**
 * Judge the cloud is loaded or not.
 * @return isLoaded, the load status of the cloud.
 */
public boolean isLoaded() {
	return isLoaded;
}
```

```
/**
 * Set the load status of the cloud
 * @param isLoaded, boolean type.
 */
public void setIsLoaded(boolean isLoaded) {
	this.isLoaded = isLoaded;
}

public boolean listHost(){
	hostList=new TreeMap<String,Host>();
	if (this.cloudType==Cloud.CloudType.PRIVATE)
	{
		cp.RegisterCloud(conf);
		ArrayList<Host> hlist=cp.listHost();
		if(hlist!=null){
			for (Host h:hlist)
				hostList.put(h.getId(),h);
		}
		return true;
	}
	return false;
}
/**
 * Get the host list of the cloud
 * @return a TreeMap of host instance list, as key set is the host's id.
 */
public TreeMap<String, Host> getHostList() {

	return hostList;
}

/**
 * Set the host list
 * @param hostList. a TreeMap of host instances' list
 */
public void setHostList(TreeMap<String, Host> hostList) {
	this.hostList = hostList;
}

/**
 * Set the usage priority of the cloud, the smaller number is higher priority.
 * @param int i, is the value of the priority.
 */
public void setPriority(int i) {
	// TODO Auto-generated method stub
	this.priority = i;
}

/**
 * Get the priority of the cloud
 * @return a number of priotiry.
 *
```

```
 */
public int getPriority() {
	return priority;
}

/**
 * Migrate a given virtual machine to a specific host.
 * @param vmId , hostid, The virtual machine's id and the target host id.
 */
public String migrate(String vmID,String hostid) {
	// TODO Auto-generated method stub
/*	if(!vmList.keySet().contains(vmID)||!hostList.keySet().contains(hostid)){
		System.out.println("This virtual machine or host does not exist!");
		return false;
	}*/
	String str="";
	cp.RegisterCloud(conf);
	if(cp.migrate(vmID,hostid))
	str = "virtual machine "+vmID+" is being migrated to the host "+hostid+"... ...";
	else str = "Migration failed";
	return str;
}

/**
 * Physically turn off the given host.
 * @param hostID, The given host's id.
 */
public boolean hostoff(String hostID) {
	// TODO Auto-generated method stub
	Host host = hostList.get(hostID);
	if(host.getPowerStat()==0){
		System.out.println("Host current status is power off!!");
		return false;
	}
	if(!host.getVmList().isEmpty()){
		System.out.println("Host still has vms is running,cannot be shutted down!");
		return false;
	}
	cp.RegisterCloud(conf);
	cp.hostoff(host.getIpmiID());
	host.setPowerStat(0);
	return true;
}

/**
 * Physically turn on the given host.
 * @param the given host's id.
 */
public boolean hoston(String hostID) {
	// TODO Auto-generated method stub
	Host host = hostList.get(hostID);
	if(host.getPowerStat()==1){
		System.out.println("Host current status is power on!!");
```

```
		return false;
	}

	host.setPowerStat(3);
	return true;
}

public int getImageSize()
{return imageSize;}

public void setImageSize(int size)
{imageSize=size;}

public String getCluster(){
	return cluster;
}

public void setCluster(String cName){
	cluster=cName;
}

/**
 *  Need to override this function, all the VMs should match using dnsnames
 */
public String slotNameToVMId(String slotName){
	String vmId="";
	if(cloudName.equals("Gcloud")){
		if(slotName.contains(".")){
			vmId=slotName.replace(".kisti", "").split("-")[1].replace("vm", "").trim();
		}else{
			vmId=slotName.split("-")[1].replace("vm", "");
		}

	}else if(cloudName.equalsIgnoreCase("fermicloud")){

	}else if(cloudName.equalsIgnoreCase("amazon")){

		String ip = slotName.replace(".amaz", "").replaceAll("-", ".");
		System.out.println(ip);
		for(Vm vm : vmList.values()){
			if(vm.getPrivateIP().equalsIgnoreCase(ip)){
				vmId = vm.getId();
			}
		}
	}
	return vmId;
}
class HostControlRunner implements Runnable{

	@Override
	public void run() {
		// TODO Auto-generated method stub
```

```
		}

	}

	public enum CloudType {PRIVATE, PUBLIC, NOT_DEFINED};


	private String cloudName;
	private String cloudpluginName;
	private List<String> conf;
	private int currentVMs;
	private CloudType cloudType;
	private CloudInterface cp;
	private TreeMap<String, Vm> vmList;
	private boolean isLoaded;
	private TreeMap<String,Host> hostList;
	private int priority;

	/*
	 * Added functions by Hao
	 */
	/*
	 * The size of image, this attribute is temp here. Maybe future in VM class
	 */
	private int imageSize;
	/*
	 * cluster name of the cloud host machine
	 */
	private String cluster;

	/**
	 * To match short hostname and full dns hostnames
	 */
	public static boolean hostNameMatch(String n1, String n2){
		String [] sn1=n1.split("\\.");
		String [] sn2=n2.split("\\.");
		return (n1.equals(n2)||sn1[0].equals(sn2[0]));
	}

	/**
	 * Add virtual machines to its host
	 */
	public boolean matchVMtoHost(){
		if(vmList.isEmpty()||hostList.isEmpty())
			return false;
		for (Vm v:vmList.values())
		{
			String str=null;
			if((str=v.getHostname())!=null)
			{
				for(Host h:hostList.values())
				{
					if(hostNameMatch(str,h.getName())){
```

```
h.getVmList().put(v.getId(), v);
if (v.getState()==VMState.RUNNING)
	h.setCurrVmNum(h.getCurrVmNum()+1);
break;
}
}
}
}
}
return true;
}
}
```


File: project-fermi/src/vcluster/elements/Host.java

```
package vcluster.elements;


import java.util.TreeMap;


import vcluster.managers.CloudManager;


/**

 * A class that presents a host of cloud.

 * Update with some one host features -- Hao

 */

public class Host {

	private TreeMap<String,Vm> vmList;

	private String cloudName;

	private int maxVmNum;

	private int currVmNum;

	private int remainingVmNum;

	private String id;
```

```
private String name;
private String ipmiID;
private int powerStat = 1;
//host total cpu
private int tcpu;

//host free cpu
private int fcpu;

//host allocated cpu
private int acpu;

//host total memory
private double tmem;

//host free memory
private double fmem;

//host allocated memory
private double amem;

//host status
private HostStat stat;

//host current cpu utilization, [0,100]
private double cpuUtil;
```

```
//host current io utilization,[0,100]
private double ioUtil;

//enumeration of the host states
public static enum HostStat{ON, OFF, ERR, UNKNOW};

public String getName() {
	return name;
}


public void setName(String name) {
	this.name = name;
}


/**
 * Get the status of power,on or off
 * @return power Status
 */
public int getPowerStat() {
	return powerStat;
}
```

```
/**
 * Set the status of the power,on or off.
 * @param the power status.
 */
public void setPowerStat(int powerStat) {
	this.powerStat = powerStat;
}


/**
 * Get the ipmiID
 * @return the string of ipmiID
 */
public String getIpmiID() {
	return ipmiID;
}


/**
 * Set the ipmi ID
 * @param ipmiID.
 */
public void setIpmiID(String ipmiID) {
	this.ipmiID = ipmiID;
}
```

```
/**
 * Get the remaining capability of creating virtual machines.
 * @return The number of how many virtual machines can be created on this host.
 */
public int getRemainingVmNum() {
	return remainingVmNum;
}


/**
 * Set the  remaining capability of creating virtual machines.
 * @param  The number of how many virtual machines can be created on this host.
 */
public void setRemainingVmNum() {
	this.remainingVmNum = maxVmNum-currVmNum;
}

/**
 * Get the cloud name that this host belongs to.
 * @return cloud name
 */
public String getCloudName() {
	return cloudName;
}
```

```
/**
 *Set the cloud name that this host belongs to.
 *@param cloud name
 */
public void setCloudName(String cloudName) {
	this.cloudName = cloudName;
}

/**
 * Constructor that specify the id,cloud name and the max capability of virtual machine creation.
 */

public Host(int maxVmNum, String id,String name, String cloudName) {
	this.maxVmNum = maxVmNum;
	this.id = id;
	this.cloudName = cloudName;
	vmList = new TreeMap<String,Vm>();
	this.name = name;
}



/**
 * Get the virtual machines' list that running on this host.
 * @return A TreeMap of the instances of virtual machines. as the key set is vitual machine's id.
 */
public TreeMap<String, Vm> getVmList() {
```

```
	if ( CloudManager.getCloudList().get(cloudName).getCloudType()==Cloud.CloudType.PUBLIC){
	vmList = new TreeMap<String,Vm> ();
	for(Vm vm : CloudManager.getCloudList().get(cloudName).getVmList().values()){
		if(vm.getHostname().equalsIgnoreCase(id)){
			//System.out.println(vm.getHostname()+" : "+ id + "   mark");
			vmList.put(vm.getId(), vm);
		}
	}
	currVmNum = vmList.size();
	return vmList;
	}
	else {
		return vmList;
	}
}

/**
 * Set the virtual machines list that running on the host.
 * @param vmList, a treeMap of virtual machines' instances
 */
public void setVmList(TreeMap<String, Vm> vmList) {
	this.vmList = vmList;
}

/**
 * Get the max capability of virtual machine creation.
```

```
 * @return maxVmNum, the max number of virtual machines that can be launched on this host.
 */
public int getMaxVmNum() {
	return maxVmNum;
}

/**
 * Set the max capability of virtual machine creation.
 * @param maxVmNum, the max number of virtual machines that can be launched on this host.
 */
public void setMaxVmNum(int maxVmNum) {
	this.maxVmNum = maxVmNum;
}
/**
 * Get the number of running virtual machines.
 * @return the number of virtual machines that already be launched on this host.
 */
public int getCurrVmNum() {
	return getVmList().size();
}
/**
 * Set the number of running virtual machines.
 * @param the number of virtual machines that already be launched on this host.
 */
public void setCurrVmNum(int currVmNum) {
	this.currVmNum = currVmNum;
```

```
}

/**
 * Get the host id
 * @return the id of the host.
 */
public String getId() {
	return id;
}

/**
 * Set the host id.
 * @param the id of the host.
 */
public void setId(String id) {
	this.id = id;
}

//Get and set the total number of cpus
public  int getTCPU()
{return tcpu;}
public void setTCPU(int n)
{tcpu=n;}

//Get and set the free number of cpus
public  int getFCPU()
```

```
{return fcpu;}
public void setFCPU(int n)
{fcpu=n;}

//Get and set the allocated number of cpus
public  int getACPU()
{return acpu;}
public void setACPU(int n)
{acpu=n;}

//Get and set the total memory
public  double getTMEM()
{return tmem;}
public void setTMEM(double n)
{tmem=n;}

//Get and set the free memory
public  double getFMEM()
{return fmem;}
public void setFMEM(double n)
{fmem=n;}

//Get and set the allocated memory
public  double getAMEM()
{return amem;}
public void setAMEM(double n)
```

```
{amem=n;}

//Get and set the host states
public  HostStat getStat()
{return stat;}
public void setStat(HostStat s)
{stat=s;}

public void setStat(String s)
{
	if(s.equalsIgnoreCase("on"))
		stat = HostStat.ON;
	else if(s.equalsIgnoreCase("off"))
		stat = HostStat.OFF;
	else
		stat = HostStat.ERR;
}

//Get and set the current cpu utilization on host
public double getCPUUtil()
{return cpuUtil;}
public void setCPUUtil(double u)
{cpuUtil= u;}

//Get and set the current io utilization on host
```

```
        public double getIOUtil()

        {return ioUtil;}

        public void setIOUtil(double u)

        {ioUtil=u;}



}


-----------
-----------
-----------


File: project-fermi/src/vcluster/elements/Job.java
package vcluster.elements;

public class Job {

        private String id;
        private String owner;
        private String submitted;
        private String runt;
        private String st;
        private String pri;
        private String size;
        private String host;

        public Job()
        {
                id="";
                owner="";
                submitted="";
                runt="";
                st="";
                pri="";
                size="";
                host="";
        }

        public String getID()
        {return id;}
        public void setID(String i)
        {id=i;}

        public String getOwner()
        {return owner;}
        public void setOwner(String o)
        {owner=o;}
```

```
        public String getSubmit()
        {return submitted;}
        public void setSubmit(String s)
        {submitted=s;}

        public String getRunT()
        {return runt;}
        public void setRunT(String rt)
        {runt=rt;}

        public String getStat()
        {return st;}
        public void setStat(String s)
        {st=s;}

        public String getPri()
        {return pri;}
        public void setPri(String p)
        {pri=p;}

        public String getSize()
        {return size;}
        public void setSize(String s)
        {size=s;}

        public String getHost(){
                return host;
        }

        public void setHost(String hostName)
        {host=hostName;}
}

-----------
-----------
-----------


File: project-fermi/src/vcluster/elements/Plugin.java
package vcluster.elements;



/**

 *A class represents plug-in

 */

public class Plugin {
```

```
private String pluginName;
private String pluginStatus;
private String pluginType;
private Object instance;

/**
 *Get the name of the plug-in
 *@return pluginName, the name of the plug-in.
 */
public String getPluginName() {
       return pluginName;
}
/**
 * Set the name of the plug-in
 * @param pluginName, the name of the plug-in
 */
public void setPluginName(String pluginName) {
       this.pluginName = pluginName;
}

/**
 *Get the plug-in status,loaded or unloaded into vcluster.
 *@return pluginStatus.
 */
public String getPluginStatus() {
```

```
       return pluginStatus;
}

/**
 *Set the plug-in status
 *@param pluginStatus, the plug-in loaded/unloaded status.
 */
public void setPluginStatus(String pluginStatus) {
       this.pluginStatus = pluginStatus;
}

/**
 *Get the plug-in type, There are three types of plug-in, cloud,batch system and load balancer.
 *@return pluginType, the type of the plug-in.
 */
public String getPluginType() {
       return pluginType;
}

/**
 *Set the type of the plug-in, There are three types of plug-in, cloud,batch system and load balancer.
 *@param pluginType, the type of the plug-in.
 */
public void setPluginType(String pluginType) {
       this.pluginType = pluginType;
}
```

```
	/**
	 *Get the instance that this class represents.
	 *@return instance, cloud be an instance of cloud plug-in, batch system plug-in or load balancer plug-in.
	 */
	public Object getInstance() {
		return instance;
	}

	/**
	 *Set the instance that this class represents.
	 * @param instance, cloud be an instance of cloud plug-in, batch system plug-in or load balancer plug-in.
	 */
	public void setInstance(Object instance) {
		this.instance = instance;
	}




}

-----------
-----------
-----------

File: project-fermi/src/vcluster/elements/PoolStatus.java
package vcluster.elements;

import java.util.ArrayList;
/**
 * A class representing the pool status of the batch system.
 *
```

```
 */
public class PoolStatus {

	/**
	 * Constructor,initialize the pool status
	 */
	public PoolStatus(int totalSlot, int ownerSlot,
			int claimedSlot, int unClaimedSlot, int matchedSlot,
			int preemptingSlot, int backfillSlot) {
		super();
		this.totalSlot = totalSlot;
		this.ownerSlot = ownerSlot;
		this.claimedSlot = claimedSlot;
		this.unClaimedSlot = unClaimedSlot;
		this.matchedSlot = matchedSlot;
		this.preemptingSlot = preemptingSlot;
		this.backfillSlot = backfillSlot;
	}
/**
 *Constructor, not initialize anything of the pool status
 *
 */
	public PoolStatus() {
		// TODO Auto-generated constructor stub
	}

	/**
	 *
	 *Set the number of unclaimed slots
	 *@param unClaimedSlot, the number of unclaimed slots
	 */
	public void setUnClaimedSlot(int unClaimedSlot) {
		this.unClaimedSlot = unClaimedSlot;
	}
	/**
	 * Get the number of nuclaimed slots
	 * @return the number of unclaimed slots
	 */
	public int getUnClaimedSlot() {
		return unClaimedSlot;
	}

	/**
	 *Set the number of  total  slots
	 * @param totalSlot ,the number of totalSlot.
	 */
	public void setTotalSlot(int totalSlot) {
		this.totalSlot = totalSlot;
	}

	/**
```

```
 *Get the number of total slots.
 *@return the number of total slots.
 */
public int getTotalSlot()
{
	return totalSlot;
}

/**
 *Set the number of owner slots
 *@param ownerSlot, the number of owner slotsf
 *
 */
public void setOwnerSlot(int ownerSlot) {
	this.ownerSlot = ownerSlot;
}



/**
 *Get the number of owner slots
 *@return the number of owner slots;
 */
public int getOwnerSlot()
{
	return ownerSlot;
}

/**
 *Set the number of Claimed slots.
 *@param the number of claimed slots.
 */
public void setClaimedSlot(int claimedSlot) {
	this.claimedSlot = claimedSlot;
}


/**
 *Get the number of Claimed slots.
 *@return the number of claimed slots.
 */
public int getClaimedSlot()
{
	return claimedSlot;
}

/**
 * Set the number of Matched slots
 *@param matchedSlot ,the number of matched slots.
 */
public void setMatchedSlot(int matchedSlot) {
	this.matchedSlot = matchedSlot;
}
```

```
/**
 *Get the  the number of Matched slots.
 *@return the number of Matched slots.
 *
 */
public int getMatchedSlot()
{
	return matchedSlot;
}

/**
 *Set the number of preempting slots .
 *@param preemptingSlot,the number of preempting slots.
 */
public void setPreemptingSlot(int preemptingSlot) {
	this.preemptingSlot = preemptingSlot;
}

/**
 *Get the number of preempting slots.
 *@return the number of preempting slots.
 */
public int getPreemptingSlot()
{
	return preemptingSlot;
}

/**
 *Set the number of Backfill slots .
 *@param Backfill,the number of Backfill slots.
 */
public void setBackfillSlot(int backfillSlot) {
	this.backfillSlot = backfillSlot;
}

/**
 *Get the number of Backfill slots.
 *@return the number of Backfill slots.
 */
public int getBackfillSlot()
{
	return backfillSlot;
}

/**
 *Get a list of all of the slots.
 *@return  An array list of slots
 */
public ArrayList<Slot> getSlotList() {
	return slotList;
}
```

```
	/**
	 *Set a list of all of the slots.
	 *@param  An array list of slots
	 */
	public void setSlotList(ArrayList<Slot> slotList) {
		this.slotList = slotList;
	}


	/**
	 *A function to output the pool status,for command line console,print the result on screen.
	 */
	public String printPoolStatus()
	{
		StringBuffer str = new StringBuffer();
		str.append("----------------------------------------"+System.getProperty("line.separator"));
		str.append("Pool Status"+System.getProperty("line.separator"));
		str.append("----------------------------------------"+System.getProperty("line.separator"));
		str.append(" Total Slots : " + getTotalSlot()+System.getProperty("line.separator"));
		str.append(""+System.getProperty("line.separator"));
		str.append("       Owner : " + getOwnerSlot()+System.getProperty("line.separator"));
		str.append("     Claimed : " + getClaimedSlot()+System.getProperty("line.separator"));
		str.append("   Unclaimed : " + getUnClaimedSlot()+System.getProperty("line.separator"));
		str.append("     Matched : " + getMatchedSlot()+System.getProperty("line.separator"));
		str.append("  Preempting : " + getPreemptingSlot()+System.getProperty("line.separator"));
		str.append("    Backfill : " + getBackfillSlot()+System.getProperty("line.separator"));
		str.append("----------------------------------------"+System.getProperty("line.separator"));

		return str.toString();
	}


	private ArrayList<Slot> slotList;
	private int totalSlot = 0;
	private int ownerSlot = 0;
	private int claimedSlot = 0;
	private int unClaimedSlot = 0;
	private int matchedSlot = 0;
	private int preemptingSlot = 0;
	private int backfillSlot = 0;

}

-----------
-----------
-----------


File: project-fermi/src/vcluster/elements/QStatus.java
package vcluster.elements;

import java.util.ArrayList;
```

```
/**
 * A class representing the job queue status of the batch system.
 *
 */
public class QStatus {

	/**
	 *The constructor, to initialize the QStatus.
	 */
	public QStatus(int totalJob, int completedJob, int removedJob, int idleJob,
		       int runningJob, int heldJob, int suspendedJob) {
		super();
		this.totalJob = totalJob;
		this.completedJob = completedJob;
		this.removedJob = removedJob;
		this.idleJob = idleJob;
		this.runningJob = runningJob;
		this.heldJob = heldJob;
		this.suspendedJob = suspendedJob;
	}

	/**
	 * Get the number of total jobs
	 *@return the number of total jobs  *
	 */
	public int getTotalJob() {
		return totalJob;
	}

	/**
	 *Set the number of total jobs
	 *@param totaljob, the number of total jobs.
	 */
	public void setTotalJob(int totalJob) {
		this.totalJob = totalJob;
	}

	/**
	 * Get the number of completed jobs
	 *@return the number of completed jobs     *
	 */
	public int getCompletedJob() {
		return completedJob;
	}

	/**
	 *Set the number of completed jobs
	 *@param completedJob, the number of completed jobs.
	 */
	public void setCompletedJob(int completedJob) {
		this.completedJob = completedJob;
	}
```

```
/**
 * Get the number of removed jobs
 *@return the number of removed jobs       *
 */
public int getRemovedJob() {
       return removedJob;
}

/**
 *Set the number of removed jobs
 *@param removedJob, the number of removed jobs.
 */
public void setRemovedJob(int removedJob) {
       this.removedJob = removedJob;
}
/**
 * Get the number of idle jobs
 *@return the number of idle jobs   *
 */
public int getIdleJob() {
       return idleJob;
}

/**
 *Set the number of idle jobs
 *@param idleJob, the number of idle jobs.
 */
public void setIdleJob(int idleJob) {
       this.idleJob = idleJob;
}

/**
 * Get the number of running jobs
 *@return the number of running jobs       *
 */
public int getRunningJob() {
       return runningJob;
}

/**
 *Set the number of running jobs
 *@param runningdJob, the number of running jobs.
 */
public void setRunningJob(int runningJob) {
       this.runningJob = runningJob;
}

/**
 * Get the number of held jobs
 *@return the number of held jobs   *
 */
public int getHeldJob() {
       return heldJob;
```

```
}

/**
 *Set the number of held jobs
 *@param heldJob, the number of held jobs.
 */
public void setHeldJob(int heldJob) {
       this.heldJob = heldJob;
}

/**
 * Get the number of suspended jobs
 *@return the number of suspended jobs     *
 */
public int getSuspendedJob() {
       return suspendedJob;
}

/**
 *Set the number of suspended jobs
 *@param suspendedJob, the number of suspended jobs.
 */
public void setSuspendedJob(int suspendedJob) {
       this.suspendedJob = suspendedJob;
}

/**
 *This is a function the output the job queue status,for command line console,print the result on the screeen.
 */

public ArrayList<Job> getJobs(){
       return jobs;
}
public void setJobs(ArrayList<Job> j){
       jobs=j;
}

public boolean setJobHost(String jID, String hostName)
{
       for(int i=0;i<jobs.size();i++)
       {
              if(jobs.get(i).getID().equalsIgnoreCase(jID))
              {
                     jobs.get(i).setHost(hostName);
                     return true;
              }
}
       return false;
}

public String printQStatus()
{
       StringBuffer str = new StringBuffer();
```

```
		str.append("--------------------------------------"+System.getProperty("line.separator"));
		str.append("Queue Status"+System.getProperty("line.separator"));
		str.append("--------------------------------------"+System.getProperty("line.separator"));
		str.append(" Total Jobs : " + getTotalJob()+System.getProperty("line.separator"));
		str.append(""+System.getProperty("line.separator"));
		str.append("  Completed : " + completedJob+System.getProperty("line.separator"));
		str.append("    Removed : " + removedJob+System.getProperty("line.separator"));
		str.append("       Idle : " + getIdleJob()+System.getProperty("line.separator"));
		str.append("    Running : " + getRunningJob()+System.getProperty("line.separator"));
		str.append("       Held : " + getHeldJob()+System.getProperty("line.separator"));
		str.append("  Suspended : " + suspendedJob+System.getProperty("line.separator"));
		str.append("--------------------------------------"+System.getProperty("line.separator"));
		//System.out.println(str);
		return str.toString();
	}

	public String printJobs()
	{
		StringBuffer flag=new StringBuffer();
		String cID = String.format("%-7s", "ID");
		String cOwner =String.format("%-12s", "OWENER");
		String cSubmit = String.format("%-16s", "SUBMITTED");
		String cRunTime = String.format("%-12s", "RUN_TIME");
		String cStatus = String.format("%-7s", "STATUS");
		String cHost = String.format("%-20s", "HOST");
		flag.append("-------------------------------------------------------"+System.getProperty("line.separator"));
		flag.append(cID+cOwner+cSubmit+cRunTime+cStatus+cHost+System.getProperty("line.separator"));
		flag.append("-------------------------------------------------------"+System.getProperty("line.separator"));
		for(int i = 0;i<jobs.size();i++){
			try {
				String tID = String.format("%-7s", jobs.get(i).getID());
				String tOwner =String.format("%-12s", jobs.get(i).getOwner());
				String tSubmit = String.format("%-16s", jobs.get(i).getSubmit());
				String tRunTime = String.format("%-12s", jobs.get(i).getRunT());
				String tStatus = String.format("%-7s", jobs.get(i).getStat());
				String tHost = String.format("%-20s", jobs.get(i).getHost());
				flag.append(tID+tOwner+tSubmit+tRunTime+tStatus+tHost+System.getProperty("line.separator"));
			} catch (NullPointerException e) {
				// TODO Auto-generated catch block
				//e.printStackTrace();
				flag.append("Jobs doesn't exist!"+System.getProperty("line.separator"));
			}
		}
		flag.append("-------------------------------------------------------"+System.getProperty("line.separator"));
		return flag.toString();
	}
	private int totalJob = 0;
	private int completedJob = 0;
	private int removedJob = 0;
	private int idleJob = 0;
	private int runningJob = 0;
	private int heldJob = 0;
```

```
	private int suspendedJob = 0;
	private ArrayList<Job> jobs;

}
```


File: project-fermi/src/vcluster/elements/Slot.java

```
package vcluster.elements;

/**

 * A class representing a slot of the batch system.

 *

 */

public class Slot {

	private String domain;

	private String identifier;

	private IdType idType;

	private int activity;

	private String hostname;


    /*

     * Set and get the hostname of the slots.

     */

	public String gethostName(){

		return hostname;

	}


	public void setHostName(String hn){

		hostname=hn;
```

```
}
/**
 *Get the domain of the slot, identify where(or which cloud) is the slot from.
 * @return the domain of the slot.
 */
public String getDomain() {
	return domain;
}

/**
 *Set the domain of the slot.
 *@param domain, the domain of the slot
 */
public void setDomain(String domain) {
	this.domain = domain;
}

/**
 *Get the identifier of the slot,
 *cloud be a string of IP address, or virtual machine's ID, etc.
 *@return the identifier of the slot.
 */
public String getIdentifier() {
	return identifier;
}
```

```
/**
 *Set the identifier of the slot.
 *@return a string of the identifier type of the slot.
 */
public void setIdentifier(String identifier) {
	this.identifier = identifier;
}

/**
 *Get the identifier type of the slot,
 *for example: IP address, virtual machine's ID
 *@return the identifier type of the slot.
 */
public IdType getIdType() {
	return idType;
}

/**
 *Set the identifier type of the slot.
 *@return the identifier type of the slot.
 */
public void setIdType(IdType idType) {
	this.idType = idType;
}
```

```
	/**
	 *Get the activity status of a slot
	 *There are three kinds of status for a slot
	 *0 represents the slot is idle
	 *1 represents the slot is busy
	 *3 represents the status is not determined.
	 *@return the activity status of a slot.
	 */
	public int getActivity() {
		return activity;
	}

	/**
	 *Set the activity status of a slot
	 *@param activity, only can be one of the number "0, 1, 3" .
	 */
	public void setActivity(int act) {
		this.activity = act;
	}

	/**
	 *An enum to define the identifier types of a slot.
	 *Five types is defined, which are "PRIVATEIP, PUBLICIP, VMID, VMHOSTNAME,NOTDEFINED ".
	 */
	public static enum IdType {PRIVATEIP, PUBLICIP, VMID, VMHOSTNAME,NOTDEFINED };
	public static enum Activity {IDLE,BUSY, NOTDETERMINED};
```

```
}

-----------
-----------
-----------

File: project-fermi/src/vcluster/elements/Vm.java
package vcluster.elements;


import java.util.ArrayList;


import vcluster.managers.VmManager.VMState;
import vcluster.elements.Job;
/**
 * A class representing a virtual machine
 */
public class Vm {
	private Integer uId;
	private String id;
	private VMState state;
	private String memory;
	private String user;
	private String hostname;
	private String time;
	private String ucpu;
	private String group;
	private String cloudName;
	private String privateIP;
	private String pubicIP;
```

```
private int isIdle;
private ArrayList<Job> jobs;

private String dnsname;
/**
 *Constructor, initialize an instance of virtual machine as the default members' value.
 */
public Vm() {
	this.uId = new Integer(0);
	this.id = "";
	this.state = VMState.PENDING;
	this.memory = "";
	this.user = "";
	this.hostname = "";
	this.time = "";
	this.ucpu = "";
	this.group = "";
	this.cloudName = "";
	this.privateIP="";
	this.pubicIP="";
	this.isIdle = 3;
	this.dnsname="";
}

/**
 *Get the global Id
```

```
 *@return uId, the global id in vcluster
 */
public Integer getuId() {
	return uId;
}

 /**
  *Set the global Id
  *@param uId, the global id in vcluster.
  */
public void setuId(Integer uId) {
	this.uId = uId;
}

/**
 *Get the private IP of the virtual machine.
 *@return privateIp, the private IP of the virtual machine.
 */
public String getPrivateIP() {
	return privateIP;
}


/**
 *Set the private IP of the virtual machine.
 *@param privateIp, the private IP of the virtual machine.
```

```
	 */
	public void setPrivateIP(String privateIP) {
		this.privateIP = privateIP;
	}



	/**
	 *Get the public IP of the virtual machine.
	 *@return publicIP, the public IP of the virtual machine.
	 */
	public String getPubicIP() {
		return pubicIP;
	}



	/**
	 *Set the public IP of the virtual machine.
	 *@param publicIP, the public IP of the virtual machine.
	 */
	public void setPubicIP(String pubicIP) {
		this.pubicIP = pubicIP;
	}

	/**
	 *Get the internal ID of virtual machine, this is the ID that identify the virtual machine in the real cloud system.
It's generated by cloud.
	 *@return id, the internal ID of the virtual machine
```

```
	 */
	public String getId() {
		return id;
	}

	/**
	 *Set the internal ID of virtual machine, this is the ID that identify the virtual machine in the real cloud system.
It's generated by cloud.
	 *@param id, the internal ID of the virtual machine
	 */
	public void setId(String id) {
		this.id = id;
	}

	/**
	 *Get the virtual machine's status
	 *@return state, the status of a virtual machine,
	 *@see VMState
	 */
	public VMState getState() {
		return state;
	}

	/**
	 *Set the virtual machine's status
	 *@param state, the status of a virtual machine.
	 *@see VMState
```

```
 */
public void setState(VMState state) {
	this.state = state;
}

/**
 *Get the memory size of the virtual machine.
 *@return memory, the memory size of the virtual machine.
 */
public String getMemory() {
	return memory;
}

/**
 *Set the memory size of the virtual machine.
 *@param memory, the memory size of the virtual machine.
 */
public void setMemory(String memory) {
	this.memory = memory;
}

/**
 *Get the user of the virtual machine.
 *@return user, the user of the virtual machine.
 */
public String getUser() {
```

```
	return user;
}
/**
 *Set the user of the virtual machine.
 *@param user, the user of the virtual machine.
 */
public void setUser(String user) {
	this.user = user;
}

/**
 *Get the host name of the virtual machine.
 *@return hostname, the host name of the virtual machine.
 *@see hostname.
 */
public String getHostname() {
	return hostname;
}

/**
 *Set the host name of the virtual machine.
 *@param hostname, the host name of the virtual machine.
 *@see hostname.
 */
public void setHostname(String hostname) {
	this.hostname = hostname;
```

```
}

/**
 *Get the launch time of the virtual machine.
 *@return time, the launch time of the virtual machine.
 *@see time.
 */
public String getTime() {
	return time;
}

/**
 *Set the launch time of the virtual machine.
 *@param time, the launch time of the virtual machine.
 *@see time.
 */
public void setTime(String time) {
	this.time = time;
}

/**
 *Get the usage percentage of the virtual machine.
 *@return ucpu, the usage percentage of the virtual machine.
 *@see ucpu.
 */
public String getUcpu() {
```

```
	return ucpu;
}

/**
 *Set the usage percentage of the virtual machine.
 *@param ucpu, the usage percentage of the virtual machine.
 *@see ucpu.
 */
public void setUcpu(String ucpu) {
	this.ucpu = ucpu;
}

/**
 *Get the group that the virtual machine belongs to.
 *@return group, the group of the virtual machine belongs to.
 *@see group.
 */
public String getGroup() {
	return group;
}

/**
 *Set the group that the virtual machine belongs to.
 *@param group, the group of the virtual machine belongs to.
 *@see group.
 */
```

```
	public void setGroup(String group) {
		this.group = group;
	}

	/**
	 *Get the name of the cloud that the virtual machine belongs to.
	 *@return cloudName, the name of the cloud of the virtual machine belongs to.
	 *@see cloudName.
	 *@see Cloud
	 */
	public String getCloudName() {
		return cloudName;
	}

	/**
	 *Set the name of the cloud that the virtual machine belongs to.
	 *@param cloudName, the name of the cloud of the virtual machine belongs to.
	 *@see cloudName.
	 *@see Cloud
	 */
	public void setCloudName(String cloudName) {
		this.cloudName = cloudName;
	}


	/**
```

```
	 *Get the activity status of the virtual machine,idle or busy.
	 *@return isIdle, the value is a integer number, 0 represents idle, 1 represents busy and 3 represents vcluster failed
to get the information.
	 */
	public int isIdle() {
		// TODO Auto-generated method stub
		return this.isIdle;
	}

	/**
	 *Set the activity status of the virtual machine,idle or busy.
	 *@param isIdle, the value is a integer number, 0 represents idle, 1 represents busy and 3 represents vcluster failed
to get the information.
	 */
	public void setIsIdle(int isIdle) {
		this.isIdle = isIdle;
	}

	/*
	 * Get the VM's DNS name
	 */
	public String getDNSName(){
		return dnsname;
	}

	/*
	 * Set the VM's DNS name
```

```
	 */

	public void setDNSName(String name){
		dnsname=name;
	}

	/*
	 * Get the image size of the VM
	 */

	/*
	 * Get the jobs running on the VM
	 */

	public ArrayList<Job> getJobs(){
		return jobs;
	}

	/*
	 * Set the jobs on the VM
	 */

	public void setJobs(ArrayList<Job> j)
	{
		jobs=j;
	}
```

```
	/**
	 *	Return virtual machine's status as String form.
	 */

	public String stringVMState() {

		switch(state) {
		case PENDING: return "PENDING";
		case RUNNING: return "RUNNING";
		case STOP: return "STOP";
		case SUSPEND: return "SUSPEND";
		case PROLOG: return "PROLOG";
		case FAILED: return "FAILED";
		case NOT_DEFINED: return "NOT_DEFINED";
		}
		return "NOT_DEFINED";
	}

}

-----------
-----------
-----------

File: project-fermi/src/vcluster/executors/BatchExecutor.java
package vcluster.executors;


import java.io.File;
```

```
import java.util.ArrayList;


import vcluster.elements.Cloud;

import vcluster.elements.PoolStatus;

import vcluster.elements.QStatus;

import vcluster.elements.Slot;

import vcluster.elements.Vm;

import vcluster.elements.Job;

import vcluster.managers.CloudManager;

import vcluster.managers.PluginManager;

/**

 * A class representing a batch system executor, which involves the functions to operate on batch system, such as get the pool
status and get queue status.

 */


public class BatchExecutor {

	/**

	 * Mapping between virtual machines and slots. This function pushes the slots' activity status to the corresponding
virtual machines.

	 */

	public static void mapingActivityToVm(){

		ArrayList<Slot> slotList = getPoolStatus().getSlotList();

		for(Cloud c : CloudManager.getCloudList().values()){

			if(c.getVmList()==null)continue;

			for(Vm vm : c.getVmList().values()){

				for(Slot s : slotList){

					if(s.getDomain()!=null){
```

```

 
 
 
```

```
        }

        public static boolean ConnectTo(File conf){
                return PluginManager.current_proxyExecutor.ConnectTo(conf);
        }



        /**
         * Get the pool status
         * @return A instance of PoolStatus
         * @see PoolStatus
         */
        public static PoolStatus getPoolStatus(){
                //BatchExecutor.mapingActivityToVm();
                return PluginManager.current_proxyExecutor.getPoolStatus();
        }

        /**
         * Get the queue status
         * @return A instance of the queue status
         * @see QStatus
         */
        public static QStatus getQStatus(){
                return PluginManager.current_proxyExecutor.getQStatus();
        }
```

```
}

-----------
-----------
-----------

File: project-fermi/src/vcluster/executors/CloudmanExecutor.java
package vcluster.executors;


import java.util.ArrayList;
import java.util.TreeMap;


import org.opennebula.client.host.Host;


import vcluster.Vcluster.uiType;
import vcluster.elements.*;
import vcluster.managers.CloudManager;
import vcluster.ui.CmdComb;
/**
 *A class representing Cloud manager executor. Cloud-related commands would be sent to this class and the commands will be
analyzed,
 *then corresponding functions will be invoked according the commands.
 *
 */
public class CloudmanExecutor {



        /**
         *Execute the command of register clouds on vcluster
```

```
 *step 1. analyze the command, extract the parameters
 *step 2. invoke the corresponding function in cloud manager class and send the parameters.
 *@param cmdLine, the command line of register cloud from a conf file.
 */
public static String register(CmdComb cmd) {
	// TODO Auto-generated method stub
	StringBuffer str = new StringBuffer();
	if (cmd.getParaset().size()==0) {
		str.append("[USAGE] : cloudman register <cloudelement conf file>");
		System.out.println(str);
		return str.toString();
	}
	String confFile = cmd.getParaset().get(0);
	//System.out.println(token);
	String result =  CloudManager.registerCloud(confFile);
	if(cmd.getUi().equals(uiType.CMDLINE))System.out.println(result);
	return result;
}



/**
 * Load a cloud into vcluster. command line would be analyzed and the corresponding function in cloud manager would be
invoked.
 */
public static String load(CmdComb cmd) {
	// TODO Auto-generated method stub
	if (cmd.getParaset().size()==0) {
```

```
		System.out.println("[ERROR : ] Expect a cloud name!");
		return "[ERROR : ] Expect a cloud name!";
	}
	else{
		ArrayList<String> para = cmd.getParaset();
		int size = para.size();
		String [] arg =(String [])para.toArray(new String[size]);

		String result =  CloudManager.loadCloud(arg);
		if(cmd.getUi().equals(uiType.CMDLINE))System.out.println(result);
		return result;
	}
}

/**
 * list up all the registered clouds, the relevant information would be list up.
 */
public static TreeMap<String,Cloud> getCloudList() {
	// TODO Auto-generated method stub
	return CloudManager.getCloudList();
}

public static boolean undefined(CmdComb cmd) {
	// TODO Auto-generated method stub
	/* skip the command */
```

```
        if (cmd.getParaset().size()==0) {
                System.out.println("[USAGE] : cloudman dump [<private | public>>]");
                System.out.println("        : cloudman register <cloudelement conf file>");
                System.out.println("        : cloudman set <private | public> <cloud num>");
                return false;
        }
        return true;
}

/**
 *unload a cloud from vcluster. the cloud status would be change to "unloaded",
 *and the connection between vcluster and related real cloud system would be cut off.
 */
public static boolean unload(CmdComb cmd) {
        // TODO Auto-generated method stub

        if (cmd.getParaset().size()==0) {
                System.out.println("[ERROR : ] Expect a cloud name!");
                return false;
        }else{
                String [] arg = (String[])cmd.getParaset().toArray();

                boolean result =  CloudManager.unLoadCloud(arg);
                return result;
        }
```

```
}

/**
 *Turn on a physical host
 */
public static boolean hoston(CmdComb cmd) {
        // TODO Auto-generated method stub

        if (cmd.getParaset().size()==0) {
                System.out.println("[ERROR : ] Expect a cloud name!");
                return false;
        }
        String cloudname = cmd.getParaset().get(0);
        if (cmd.getParaset().size()<2) {
                System.out.println("[ERROR : ] Expect a host id!");
                return false;
        }
        String hostID = cmd.getParaset().get(1).trim();
        Cloud cloud = CloudManager.getCloudList().get(cloudname);


        return cloud.hoston(hostID);

        }

/**
```

```
 * Shut down a physical host.
 */
public static boolean hostoff(CmdComb cmd) {
	// TODO Auto-generated method stub
	if (cmd.getParaset().size()==0) {
		System.out.println("[ERROR : ] Expect a cloud name!");
		return false;
	}
	String cloudname = cmd.getParaset().get(0).trim();
	if (cmd.getParaset().size()<2) {
		System.out.println("[ERROR : ] Expect a host id!");
		return false;
	}
	String hostID = cmd.getParaset().get(1).trim();

	Cloud cloud = CloudManager.getCloudList().get(cloudname);



	return cloud.hostoff(hostID);

}



public static String dump(CmdComb cmd) {
	// TODO Auto-generated method stub
	StringBuffer str = new StringBuffer();
```

```
	if(cmd.getParaset().size()==0){
		System.out.println("[ERROR : ] Expect a Cloud name!");
		return str.append("[ERROR : ] Expect a Cloud name!").toString();
	}
	Cloud c = CloudManager.getCloudList().get(cmd.getParaset().get(0));
	c.dump();
	String tId=  String.format("%-8s","ID");
	String tName=  String.format("%-20s", "Name");
	String tMax = String.format("%-6s", "Max");
	String tStat =  String.format("%-6s","stat");
	String tip = String.format("%-20s","Private IP");
	str.append("-------------------------------------------------------"+System.getProperty("line.separator"));
	str.append(tId+tName+tMax+tStat+tip+System.getProperty("line.separator"));
	str.append("-------------------------------------------------------"+System.getProperty("line.separator"));
	if(!(c.getHostList()==null||c.getHostList().size()==0)){
	for(vcluster.elements.Host h : c.getHostList().values()){
		String id = String.format("%-8s", h.getId());
		String name = String.format("%-20s",h.getName());
		String max = String.format("%-6s",h.getMaxVmNum()+"");
		String stat = String.format("%-6s",h.getStat().toString());
		String ip = String.format("%-20s", h.getIpmiID());
		if(h.getPowerStat()==0)
		stat = String.format("%-6s","OFF");
		str.append(id+name+max+stat+ip+System.getProperty("line.separator"));
	}
	}
```

```
		str.append("-----------------------------------------------------"+System.getProperty("line.separator"));
		if(cmd.getUi()==uiType.CMDLINE)System.out.println(str);
		return str.toString();
	}



	public static String hostlist(CmdComb cmd) {
		// TODO Auto-generated method stub
		StringBuffer str=new StringBuffer();
		String tId=  String.format("%-8s","ID");
		String tName=  String.format("%-20s", "Name");
		String tStat =  String.format("%-6s","stat");
		String tTcpu = String.format("%-8s","TCPU");
		String tAcpu = String.format("%-8s", "ACPU");
		String tFcpu = String.format("%-8s", "FCPU");
		String tTmem = String.format("%-12s","TMEM");
		String tAmem = String.format("%-12s", "AMEM");
		String tFmem = String.format("%-12s", "FMEM");
		String tVms = String.format("%-4s", "VMs");
		String tRunVms=String.format("%-8s", "RunVMs");
		String tCpuU = String.format("%-8s", "CPU_U");
		String tIoU = String.format("%-8s", "IO_U");
		str.append("-----------------------------------------------------------------------------------------------------
"+System.getProperty("line.separator"));

	str.append(tId+tName+tStat+tTcpu+tAcpu+tFcpu+tTmem+tAmem+tFmem+tVms+tRunVms+tCpuU+tIoU+System.getProperty("line.separat
or"));
```

```
		str.append("-----------------------------------------------------------------------------------------------------
"+System.getProperty("line.separator"));
		for (Cloud c : CloudManager.getCloudList().values()){
			if (!c.isLoaded())
				c.load();
			for(vcluster.elements.Host h: c.getHostList().values()
			{
				String hId=  String.format("%-8s", h.getId());
				String hName= String.format("%-20s", h.getName());
				String hStat = String.format("%-6s", h.getStat().toString());
				String hTcpu = String.format("%-8s",Integer.toString(h.getTCPU()));
				String hAcpu = String.format("%-8s",Integer.toString(h.getACPU()));
				String hFcpu = String.format("%-8s",Integer.toString(h.getFCPU()));
				String hTmem = String.format("%-12s",Integer.toString((int)h.getTMEM()));
				String hAmem = String.format("%-12s",Integer.toString((int)h.getAMEM()));
				String hFmem = String.format("%-12s",Integer.toString((int)h.getFMEM()));
				String hVms = String.format("%-4s", Integer.toString(h.getVmList().size()));
				String hRunVms = String.format("%-8s", Integer.toString(h.getCurrVmNum()));
				String hCpuU = String.format("%-8s", String.format("%3.2f", h.getCPUUtil())+"%");
				String hIoU = String.format("%-8s", String.format("%3.2f",h.getIOUtil())+"%");

	str.append(hId+hName+hStat+hTcpu+hAcpu+hFcpu+hTmem+hAmem+hFmem+hVms+hRunVms+hCpuU+hIoU+System.getProperty("line.separat
or"));
			}
		}
		str.append("-----------------------------------------------------------------------------------------------------
"+System.getProperty("line.separator"));
		if(cmd.getUi()==uiType.CMDLINE)System.out.println(str);
```

```
		return str.toString();

	}




	public static String showhost(CmdComb cmd) {

		// TODO Auto-generated method stub

		return null;

	}






}


-----------
-----------
-----------


File: project-fermi/src/vcluster/executors/CmdCataloger.java
package vcluster.executors;

import vcluster.Vcluster.uiType;
import vcluster.elements.PoolStatus;
import vcluster.elements.QStatus;
import vcluster.managers.CloudManager;
import vcluster.managers.PluginManager;
import vcluster.managers.VmManager;
import vcluster.ui.CmdComb;
import vcluster.ui.remote.CmdServer;

/**
 * @author Seo-Young Noh, Modified by Dada Huang
 * This class resolves the command and invoke the right function.
 *
 */

public class CmdCataloger {

	/**
	 *
	 *Deliver the command the different group executor according the group of the command.
```

```
	 */
	public static Object execute(CmdComb cmd)
	{
		switch (cmd.getCmdGroup()) {
		case VMMAN: return executeVMMAN(cmd);
		case CLOUDMAN: return executeCLOUDMAN(cmd);
		case PLUGMAN: return executePLUGMAN(cmd);
		case VCLMAN: return executeVCLMAN(cmd);
		default:
			break;
		}
		return "";
	}

	/**
	 *Invoke the corresponding function according the command.
	 */
	private static Object executeCLOUDMAN(CmdComb cmd) {
		// TODO Auto-generated method stub
	switch (cmd.getCmd()) {
		case REGISTER:
			return CloudmanExecutor.register(cmd);
		case LOADCLOUD:
			return CloudmanExecutor.load(cmd);
		case LISTCLOUD:
			String str = CloudManager.dump();
			if(cmd.getUi()==uiType.CMDLINE)System.out.println(str);
			return str;
		case UNLOADCLOUD:
			return CloudmanExecutor.unload(cmd);
		case HOSTON:
			return CloudmanExecutor.hoston(cmd);
		case HOSTOFF:
			return CloudmanExecutor.hostoff(cmd);
		case DUMP:
			return CloudmanExecutor.dump(cmd);
		case LISTHOST:
			return CloudmanExecutor.hostlist(cmd);
		case SHOWHOST:
			return CloudmanExecutor.showhost(cmd);
		default:
			return CloudmanExecutor.undefined(cmd);
		}
	}

	/**
	 *Invoke the corresponding function according the command.
	 */
	private static String executePLUGMAN(CmdComb cmd) {

		switch (cmd.getCmd()) {
		case LOAD:
			return PlugmanExecutor.load(cmd);
```

```
		case UNLOAD:
			return PlugmanExecutor.unload(cmd);
		case INFO:
			return PlugmanExecutor.getInfo(cmd);
		case LIST:
			return PlugmanExecutor.list(cmd);
		default:
			return PlugmanExecutor.undefined(cmd);
		}

	}

	/**
	 *Invoke the corresponding function according the command.
	 */
	private static Object executeVCLMAN(CmdComb cmd)
	{

		switch (cmd.getCmd()) {
		case SERVER_MODE:
			CmdServer.initiate();
			break;

		case CHECK_P:
			PoolStatus pstat =  BatchExecutor.getPoolStatus();
			if(cmd.getUi().equals(uiType.CMDLINE))System.out.println(pstat.printPoolStatus());
			return pstat.printPoolStatus();
		case CHECK_Q:
		QStatus qstat =  PluginManager.current_proxyExecutor.getQStatus();
		if(cmd.getUi().equals(uiType.CMDLINE)){
			System.out.println(qstat.printQStatus());
			System.out.println(qstat.printJobs());
		}
		return qstat.printQStatus();
		case LOADBALANCERSTART:
			if(PluginManager.current_loadbalancer!=null)
				PluginManager.current_loadbalancer.activate();
			else{
				System.out.println("[Error] : No Load Balancer availble!");
			}
		case AUTOSERVER:
			CmdComb cmdLine=new CmdComb("cloudman register fermi-proxy.conf");
			cmdLine.setUi(uiType.CMDLINE);
			CloudmanExecutor.register(cmdLine);
			//cmdLine=new CmdComb("cloudman register amazon.conf");
			//cmdLine.setUi(uiType.CMDLINE);
			//CloudmanExecutor.register(cmdLine);
			cmdLine=new CmdComb("plugman load -l fermibalancer");
			cmdLine.setUi(uiType.CMDLINE);
			PlugmanExecutor.load(cmdLine);
			PluginManager.current_loadbalancer.activate();
		default:
			break;
```

```
		}

		return null;
	}

	/**
	 *Invoke the corresponding function according the command.
	 */
	private static String executeVMMAN(CmdComb cmd)
	{

		switch (cmd.getCmd()) {
		case SHOW: return VmManager.showVM(cmd);
		case CREATE: return VmManager.createVM(cmd);
		case LISTVM: return VmManager.listVM(cmd);
		case DESTROY: return VmManager.destroyVM(cmd);
		case SUSPEND: return VmManager.suspendVM(cmd);
		case START: return VmManager.startVM(cmd);
		case MIGRATE:
			return VmManager.migrate(cmd);
		default:System.out.println("command is not defined");
			break;
		}

		return null;
	}


}

-----------
-----------
-----------


File: project-fermi/src/vcluster/executors/PlugmanExecutor.java
package vcluster.executors;



import java.io.File;

import java.util.ArrayList;

import java.util.List;

import java.util.StringTokenizer;



import vcluster.Vcluster.uiType;
```

```
import vcluster.elements.Plugin;

import vcluster.managers.PluginManager;

import vcluster.ui.CmdComb;

/**

 * A class representing a plug-in manager executor.In this class involves the functions which are charge of manage the plug-
ins.

 */

public class PlugmanExecutor {




	/**

	 *List up all the plug-ins located in plug-in folder

	 */

	public static String list(CmdComb cmd) {

		// TODO Auto-generated method stub

		StringBuffer str = new StringBuffer();

		if(cmd.getParaset().size()>0){

		String para2 = cmd.getParaset().get(0);

		if(para2.equals("-b")){

		}else if(para2.equalsIgnoreCase("-c")){

		}else{

			str.append("[ERROR : ] Wrong parameter!"+System.getProperty("line.separator"));

		}

	}

		String cName = String.format("%-20s", "Name");

		String cStat =String.format("%-12s", "Status");
```

```
		String cType = String.format("%-12s", "Type");

		str.append("List the plugins in plugin directory :"+System.getProperty("line.separator"));

		str.append("----------------------------------------"+System.getProperty("line.separator"));

		str.append(cName+cStat+cType+System.getProperty("line.separator"));

		str.append("----------------------------------------"+System.getProperty("line.separator"));


		for(Plugin plugin : PluginManager.pluginList.values()){

			String name = String.format("%-20s", plugin.getPluginName());

			String stat=String.format("%-12s", plugin.getPluginStatus());

			String type=String.format("%-12s", plugin.getPluginType());

			str.append(name+stat+type+System.getProperty("line.separator"));

		}

		str.append("----------------------------------------"+System.getProperty("line.separator"));

		if(cmd.getUi()==uiType.CMDLINE)System.out.println(str);

		return str.toString();

	}



	/**

	 *Load a plug-in into vcluster runtime, the plug-in package would stay in the memory.

	 */

	public static String load(CmdComb cmd) {

		// TODO Auto-generated method stub

		StringBuffer str = new StringBuffer();

		String pluginPath = "";

		if(cmd.getParaset().size()==0){
```

```
                str.append( "expected a plugin type!"+System.getProperty("line.separator"));
                str.append("[USAGE] : plugman <load -pluginType pluginName>"+System.getProperty("line.separator"));
                return str.toString();
            }
            String pluginType = cmd.getParaset().get(0);
            if(pluginType.equalsIgnoreCase("-b")){
                File dir = new File(System.getProperty("user.dir") + File.separator
                            + PluginManager.BATCH_PLUGIN_DIR);
                pluginPath = dir.getPath();
            }else if(pluginType.equalsIgnoreCase("-c")){
                File dir = new File(System.getProperty("user.dir") + File.separator
                            + PluginManager.CLOUD_PLUGIN_DIR);
                pluginPath = dir.getPath();
            }else if(pluginType.equalsIgnoreCase("-l")){
                File dir = new File(System.getProperty("user.dir") + File.separator
                            + PluginManager.LOADBALANCER_PLUGIN_DIR);
                pluginPath = dir.getPath();
            }

            List<String> pluginNames = new ArrayList<String>();
            if(cmd.getParaset().size()>1) {
                pluginNames.add(cmd.getParaset().get(1));

            }else{
                str.append( "expected a plugin name!"+System.getProperty("line.separator"));
                str.append("[USAGE] : plugin <register plugin_name | list>"+System.getProperty("line.separator"));
```

```
                return str.toString();
            }
        if(pluginType.equalsIgnoreCase("-c")){

            int i = 1;
                while(i<cmd.getParaset().size()){
                    pluginNames.add(cmd.getParaset().get(i));
                    i++;
                }

        }
                for(String pluginName:pluginNames){
                    //str.append(pluginName);
                    if(PluginManager.pluginList.keySet().contains(pluginName)){

                        try {
                            //str.append(pluginPath + File.separator + pluginName);
                            PluginManager.LoadPlugin(pluginPath + File.separator +
pluginName+".jar",pluginType);

                        } catch (ClassNotFoundException e) {
                            // TODO Auto-generated catch block
                            str.append("[ERROR:] No such a plugin,please check your
input!"+System.getProperty("line.separator"));
                            return str.toString();
                        }
```

```
                        }else{
                                str.append("[ERROR:] No such a plugin,please check your
input!"+System.getProperty("line.separator"));
                        }
                }

                if(cmd.getUi()==uiType.CMDLINE)System.out.println(str);
                return str.toString();
        }

        /**
         *Unload a plug-in from vcluster, all the references to the instance of plug-in would be removed,
         *the plug-in would be removed from memory by garbage collection
         */
        public static String unload(CmdComb cmd) {
                // TODO Auto-generated method stub
                StringBuffer str = new StringBuffer();
                if(cmd.getParaset().size()==0){
                        str.append( "expected a plugin name!"+System.getProperty("line.separator"));
                        str.append("[USAGE] : plugman <unload pluginName>"+System.getProperty("line.separator"));
                        return str.toString();
                }
                int i = 0;
                while(i<cmd.getParaset().size()){
                        try {
                                PluginManager.UnloadPlugin(cmd.getParaset().get(i));
                        } catch (Exception e) {
```

```
                                // TODO Auto-generated catch block
                                str.append(e.getMessage()+System.getProperty("line.separator"));
                                return str.toString();
                        }
                }
                if(cmd.getUi()==uiType.CMDLINE)System.out.println(str);
                return str.toString();
        }

        /**
         *Get the introduction of the plug-in
         *
         */
        public static String getInfo(CmdComb cmd) {
                // TODO Auto-generated method stub
                StringBuffer str = new StringBuffer();
                if(cmd.getParaset().size()==0){
                        str.append( "expected a plugin name!"+System.getProperty("line.separator"));
                        str.append("[USAGE] : plugman <unload pluginName>"+System.getProperty("line.separator"));
                        return str.toString();
                }
                String pluginName = cmd.getParaset().get(0);

                str.append(PluginManager.getInfo(pluginName)+System.getProperty("line.separator"));

                return str.toString();
```

```
	}


	public static String undefined(CmdComb cmd){

		StringBuffer str = new StringBuffer();

		if(cmd.getParaset().isEmpty())

			str.append("need a parameter!"+System.getProperty("line.separator"));

		else{

			str.append( "no such a parameter like \""+cmd.getParaset().get(0)+"\"
!"+System.getProperty("line.separator"));

		}

		str.append(getUsage()+System.getProperty("line.separator"));

		if(cmd.getUi()==uiType.CMDLINE)System.out.println(str);

		return str.toString();




	}

	/**

	 *The information about command line for plug-ins

	 *@return A string of help information.

	 */

	public static String getUsage(){

		StringBuffer usage = new StringBuffer();

		usage.append("[USAGE] :plugman"+System.getProperty("line.separator"));

		usage.append("                    -h : list all options and usages"+System.getProperty("line.separator"));

		usage.append("                  list : list all the plugins"+System.getProperty("line.separator"));

		usage.append("                       -c : list all cloud plugins "+System.getProperty("line.separator"));
```

```
		usage.append("                       -b : list all batch plugins "+System.getProperty("line.separator"));


		usage.append("                       -l : list all loaded plugins "+System.getProperty("line.separator"));

		usage.append("                  load : "+System.getProperty("line.separator"));

		usage.append("                       -c pluginname-1,pluginname-2......,pluginname-n : load n cloud plugins
"+System.getProperty("line.separator"));

		usage.append("                       -b pluginname : load a batch plugin
"+System.getProperty("line.separator"));

		usage.append("                unload : unload plugins"+System.getProperty("line.separator"));

		usage.append("                  info : prints the information about a plugin
"+System.getProperty("line.separator"));

		return usage.toString();


	}


}

-----------
-----------
-----------


File: project-fermi/src/vcluster/managers/CloudManager.java
package vcluster.managers;


import java.io.BufferedReader;

import java.io.FileReader;

import java.util.ArrayList;

import java.util.TreeMap;


import vcluster.elements.Cloud;
```

```
/**
 * A class that presents functions to manage the clouds. Since vcluster only has one cloudmanager,
 * all it's functions are static.
 */
public class CloudManager  {

	/**
	 * Set the current activity cloud, all the operations should be performed on the current cloud.
	 * @param instance of cloud.
	 */
	public static void setCurrentCloud(Cloud cloud)
	{
		currentCloud = cloud;
	}

	/**
	 * Get the current activity cloud.
	 * @return An instance of a cloud.
	 */
	public static Cloud getCurrentCloud()
	{
		return currentCloud;
	}
```

```
	private static boolean chkConf(ArrayList<String> conf){
		boolean flag = true;
		if(conf.size()<3){
			System.out.println("[ERROR : ] Configuration file format is incorrect,please check the format! ==>
"+conf.get(0));
			flag = false;
			return false;
		}
		String[] l1 = conf.get(0).split("=");
		String[] l2 = conf.get(1).split("=");
		String[] l3 = conf.get(2).split("=");
		if(l1.length<2||l2.length<2||l3.length<2){
			System.out.println("[ERROR : ] Configuration file format is incorrect,please check the format! ==>
"+conf.get(0));
			flag = false;
			return flag;
		}

	if(!l1[0].trim().equalsIgnoreCase("name")||!l2[0].trim().equalsIgnoreCase("interface")||l3[0].trim().equalsIgnoreCase(
"type")){
			System.out.println("[ERROR : ] Configuration file format is incorrect,please check the format! ==>
"+conf.get(0));
			flag = false;
			return flag;
		}
		return flag;
	}

	private static ArrayList<ArrayList<String>> handleConf(String confFile){
```

```
        ArrayList<ArrayList<String>> confList = new  ArrayList<ArrayList<String>>();
        BufferedReader br ;
        try{
                br = new BufferedReader(new FileReader(confFile));
                String aLine = "";
                ArrayList<String> conf = new ArrayList<String>() ;
                while ((aLine = br.readLine()) != null) {
                        if (aLine.equalsIgnoreCase("[cloudelement]")) {
                                if(!conf.isEmpty()){
                                        if(chkConf(conf)){
                                                confList.add(conf);
                                        }
                                        conf = new ArrayList<String>();
                                }

                        }else{
                                if(!aLine.trim().isEmpty())conf.add(aLine);
                                //System.out.println(aLine);
                        }
                }
                if(chkConf(conf)){
                        confList.add(conf);
                }
                br.close();
```

```
        }catch(Exception e){
                System.out.println("Configuration file doesn't exist ,please check it!");
                return null;
        }

        return confList;
}

/**
 * Register a cloud on vcluster use the given configurations
 * @param confFile, the configuration file path.
 */
public static String registerCloud(String confFile)
{
        StringBuffer str = new StringBuffer();
        ArrayList<ArrayList<String>> confList = handleConf(confFile);
        if(confList==null)return "";
        ArrayList<Cloud> tempCL = new ArrayList<Cloud>();
        for(ArrayList<String> conf:confList){
                String name = conf.get(0).split("=")[1].trim();
                boolean flag = false;
                if(cloudList!=null&&!cloudList.isEmpty()){
                        for(Cloud c : getCloudList().values()){
                                if(name.equals(c.getCloudName())){
                                        flag = true;
```

```
                                break;
                            }
                        }
                        if(flag)continue;
                    }
                    Cloud cloud = new Cloud(conf);
                    if(cloud.getCloudName()!=null&cloud.getCloudpluginName()!=null&cloud.getCloudType()!=null)
                    cloudList.put(cloud.getCloudName(), cloud);
                    tempCL.add(cloud);
            }
            String cName = String.format("%-12s", "Name");
            String cInterface =String.format("%-20s", "Interface");
            String cType = String.format("%-12s", "Type");
            String cVMs = String.format("%-16s", "VMs");
            str.append("-------------------------------------------------------"+System.getProperty("line.separator"));
            str.append(cName+cInterface+cType+cVMs+System.getProperty("line.separator"));
            str.append("-------------------------------------------------------"+System.getProperty("line.separator"));

            for(Cloud cloud:tempCL){
                    String vmnum = "";
                    if(cloud.isLoaded()){
                            vmnum = cloud.getVmList().size()+"";
                    }else{
                            vmnum = "undeployed";
                    }
                    String fName = String.format("%-12s", cloud.getCloudName());
```

```
                    String fInterface =String.format("%-20s", cloud.getCloudpluginName());
                    String fType = String.format("%-12s", cloud.getCloudType());
                    String fVMs = String.format("%-16s", vmnum);
                    str.append(fName+fInterface+fType+fVMs+System.getProperty("line.separator"));

            }

            str.append("-------------------------------------------------------"+System.getProperty("line.separator"));

            return str.toString();
    }

    /**
     *Get the registered clouds list
     */
    public static TreeMap<String, Cloud> getCloudList(){

            return cloudList;
    }


    public static String dump(){
            StringBuffer str = new StringBuffer();
            String cName = String.format("%-12s", "Name");
            String cInterface =String.format("%-20s", "Interface");
            String cType = String.format("%-12s", "Type");
```

```
		String cVMs = String.format("%-16s", "VMs");
		str.append("-----------------------------------------------------"+System.getProperty("line.separator"));
		str.append(cName+cInterface+cType+cVMs+System.getProperty("line.separator"));
		str.append("-----------------------------------------------------"+System.getProperty("line.separator"));

		for(Cloud cloud:cloudList.values()){
			String vmnum = "";
			if(cloud.isLoaded()){
				vmnum = cloud.getVmList().size()+"";
			}else{
				vmnum = "undeployed";
			}
			String fName = String.format("%-12s", cloud.getCloudName());
			String fInterface =String.format("%-20s", cloud.getCloudpluginName());
			String fType = String.format("%-12s", cloud.getCloudType());
			String fVMs = String.format("%-16s", vmnum);
			str.append(fName+fInterface+fType+fVMs+System.getProperty("line.separator"));

		}

		str.append("-----------------------------------------------------"+System.getProperty("line.separator"));

		return str.toString();
	}
	/**
	 * Load the given clouds in vcluster. vlcusetr would communicate with the real clouds system and get the virtual
machines information.
```

```
	 * @param An array of the given clouds' names
	 */
	public static String loadCloud(String[] name) {
		// TODO Auto-generated method stub
		StringBuffer flag = new StringBuffer();
		String cName = String.format("%-12s", "Name");
		String cInterface =String.format("%-20s", "Interface");
		String cType = String.format("%-12s", "Type");
		String cVMs = String.format("%-8s", "VMs");
		String cHosts = String.format("%-8s", "Hosts");
		flag.append("-------------------------------------------------------------
"+System.getProperty("line.separator"));
		flag.append(cName+cInterface+cType+cVMs+cHosts+System.getProperty("line.separator"));
		flag.append("-------------------------------------------------------------
"+System.getProperty("line.separator"));
		for(int i = 0;i<name.length;i++){
			try {
				flag.append(cloudList.get(name[i].trim()).load()+System.getProperty("line.separator"));
			} catch (NullPointerException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
				flag.append(name[i]+" doesn't exist!"+System.getProperty("line.separator"));
			}
		}
		flag.append("-------------------------------------------------------------
"+System.getProperty("line.separator"));
		return flag.toString();
	}
```

```
	/**
	 * Unload the given clouds from vcluster. the status of the unloaded clouds would be switched to "unloaded"
	 * the virtual machines that belongs to those clouds would be removed from data structure.
	 * @param  An array of the given clouds' names
	 */
	public static boolean unLoadCloud(String[] name) {
		// TODO Auto-generated method stub
		boolean flag = true;
		for(int i = 0;i<name.length;i++){
			try {
				Cloud c = cloudList.get(name[i].trim());
				if(!c.getVmList().isEmpty()){
					System.out.println(name[i]+" still have vms are running!");
					return false;
				}
				CloudManager.cloudList.remove(name[i]);
			} catch (NullPointerException e) {
				// TODO Auto-generated catch block

				System.out.println(name[i]+" Cloud doesn't exist!");
				e.printStackTrace();
			}
		}
```

```
		return flag;
	}


	/* this is only used when executing commands from command line */
	private static Cloud currentCloud = null;
	private static TreeMap<String,Cloud> cloudList = new TreeMap<String,Cloud>();


}
-----------
-----------
-----------

File: project-fermi/src/vcluster/managers/CustomClassLoader.java
package vcluster.managers;

import java.io.File;
import java.io.IOException;
import java.net.URL;
import java.net.URLClassLoader;
import java.util.Enumeration;
import java.util.jar.JarEntry;
import java.util.jar.JarFile;

import vcluster.plugInterfaces.BatchInterface;
import vcluster.plugInterfaces.CloudInterface;
import vcluster.plugInterfaces.LoadBalancer;

/**
 * This class extends java.lang.ClassLoader,which is responsible for loading java classes into the JVM dynamically.In this
class,we override the method "loadClass"
 * to make it can load class files into the JVM from the plugins forder.
 * @version 1.0,2013-05-31
 * @since JDK1.7
 */


public class CustomClassLoader extends ClassLoader {
	/**
	 * Specifys the directory where the plugin files are located.
```

```
	 */

	private JarFile jarFile;
	private URLClassLoader loader;

	/**
	 * Invoke the superclass's constructor and initialize the plugins directory.
	 * @param dir The plugins directory
	 * */
	// @Override
	public CustomClassLoader(){
		super(CustomClassLoader.class.getClassLoader());
	}

	/**
	 * Loads a specified Class to the JVM from a jar file.if the JAR file doesn't include a eligible class(implements the
interface of ProxyExecutor),it will be dropped.
	 * @param jarName The name of the target jar file,without extension .jar.
	 * @return Class<?>
	 * @exception ClassNotFoundException
	 *
	 */
	public Class<?> loadClass(String jarName){

		try {

			File f = new File(jarName);
			jarFile = new JarFile(f);
			URL url = f.toURI().toURL();
			//@test

			loader = new URLClassLoader(new URL[]{url});
			Enumeration<JarEntry> es = jarFile.entries();
			while (es.hasMoreElements()) {
			 JarEntry jarEntry = (JarEntry) es.nextElement();
			 String name = jarEntry.getName();
			 if(name != null && name.endsWith(".class")){
				Class<?> c = loader.loadClass(name.replace("/", ".").substring(0,name.length() - 6));
				    Class<?>[] intfs = c.getInterfaces();
				    for (Class<?> intf : intfs) {
					if
(intf.getName().equals(BatchInterface.class.getName())||intf.getName().equals(CloudInterface.class.getName())||intf.getName().e
quals(LoadBalancer.class.getName())) {
						return c;
					}
				    }
				// return c;
			 }
			}

			jarFile.close();
			loader.close();
```

```
		}catch(IOException e){e.printStackTrace();return null;}
		catch(ClassNotFoundException e){e.printStackTrace();return null;}
		return null;
	}
}

-----------
-----------
-----------


File: project-fermi/src/vcluster/managers/LoadManager.java
package vcluster.managers;

public class LoadManager {

	private static boolean load;
	private static boolean start;
	static{
		load=false;
		start=false;
	}
	public static boolean isLoad() {
		return load;
	}
	public static void setLoad(boolean load) {
		LoadManager.load = load;
	}
	public static boolean isStart() {
		return start;
	}
	public static void setStart(boolean start) {
		LoadManager.start = start;
	}

}

-----------
-----------
-----------


File: project-fermi/src/vcluster/managers/PluginManager.java
package vcluster.managers;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.TreeMap;
```

```
import vcluster.elements.Plugin;
import vcluster.executors.CmdCataloger;
import vcluster.plugInterfaces.BatchInterface;
import vcluster.plugInterfaces.CloudInterface;
import vcluster.plugInterfaces.LoadBalancer;
import vcluster.ui.CmdComb;
import vcluster.ui.Command;
import vcluster.util.HandleXML;

/**
 *  This class is responsible for managing plug-ins,including loading plug-ins into the JVM,unloading plugins,maintaining a
plugin list.
 * @author kjwon15 modified by Dada Huang
 *
 */
public class PluginManager {

	public static BatchInterface current_proxyExecutor;
	public static LoadBalancer current_loadbalancer;
	public static final String CLOUD_PLUGIN_DIR = "plugins"+File.separator+"cloud";
	public static final String BATCH_PLUGIN_DIR = "plugins"+File.separator+"batch";
	public static final String LOADBALANCER_PLUGIN_DIR = "plugins"+File.separator+"balancer";

	public static TreeMap<String,Plugin> pluginList;
	private static CustomClassLoader cl;


	static {
		// TODO Auto-generated constructor stub
		cl = new CustomClassLoader();
		pluginList = new TreeMap<String,Plugin>();
		getCloudPluginList();
		List<String> btl = getBatchPluginList();
		getBalancerPluginList();
		String batchplugin = btl.get(0);
		CmdComb cmd = new CmdComb("plugman load -b "+batchplugin);
		CmdCataloger.execute(cmd);
	}

	/**
	 * Loads a specified plugin(JAR file) into the JVM from the plugin directory.
	 * @param name JAR file name without the extension ".jar",case sensitive.
	 * @throws ClassNotFoundException
	 */
	public static void LoadPlugin(String path,String type) throws ClassNotFoundException {
		File f = new File(path);
		String name = f.getName().replace(".jar", "");
		//System.out.println(path);
		if (isLoaded(name)) {
				//System.out.println(name + " : has already been loaded!");
				return;
			}
			Class<?> plugin = cl.loadClass(path);
```

```
		if(type.equalsIgnoreCase("-c")){
			try {
				CloudInterface ce = (CloudInterface) plugin.newInstance();
				pluginList.get(name).setInstance(ce);
				pluginList.get(name).setPluginStatus("loaded");
			} catch (InstantiationException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			} catch (IllegalAccessException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}

		}else if(type.equalsIgnoreCase("-b")){
				try {
					for(Plugin pli : pluginList.values()){

	if(pli.getPluginType().equals("batch")&&pli.getPluginStatus().equals("loaded")){
								UnloadPlugin(pli.getPluginName());
						}
					}
					BatchInterface pe = (BatchInterface) plugin.newInstance();
					current_proxyExecutor = pe;
					pluginList.get(name).setInstance(pe);
					pluginList.get(name).setPluginStatus("loaded");
					HandleXML.loadBatchPlugin(name);
				} catch (InstantiationException e) {
					// TODO Auto-generated catch block
					e.printStackTrace();
				} catch (IllegalAccessException e) {
					// TODO Auto-generated catch block
					e.printStackTrace();
				}

		}else if(type.equalsIgnoreCase("-l")){
			try {
				LoadBalancer lbe = (LoadBalancer) plugin.newInstance();
				current_loadbalancer = lbe;
				pluginList.get(name).setInstance(lbe);
				pluginList.get(name).setPluginStatus("loaded");
				//HandleXML.loadBatchPlugin(name);
			} catch (InstantiationException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			} catch (IllegalAccessException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}

	}
	}
```

```
/**
 * Loads all plugins from the plugin directory.
 * @see #LoadPlugin(String)
 */
public static void LoadAllCloudPlugins() {
	File dir = new File(System.getProperty("user.dir") + File.separator
			+ CLOUD_PLUGIN_DIR);
	List<String> list = getCloudPluginList();
	for (String filename : list) {
		try {
			String name = filename.substring(dir.toString().length() + 1,
				filename.indexOf(".")).replace(File.separatorChar, '.');
			LoadPlugin(name,"-c");
		}catch (ClassNotFoundException e) {
			System.err.println(filename
				+ " does not contain valid class data");
		}
	}
}


/**
 * Scan the cloud plugins directiory get all the paths of JAR file
 * @see #LoadAllPlugins()
 * @return a list of paths.
 */
public static List<String> getCloudPluginList(){
	File dir = new File(System.getProperty("user.dir") + File.separator
			+ CLOUD_PLUGIN_DIR);
	File f = new File(dir.getPath());
	File[] list = f.listFiles();
	List<String> jars = new ArrayList<String>();
		for (File file : list) {
			if (file.getPath().endsWith(".jar")) {
				Plugin pi = new Plugin();
				String name = file.getName().replace(".jar", "");
				String status = "unloaded";
				jars.add(name);
				pi.setPluginName(name);
				pi.setPluginStatus(status);
				pi.setPluginType("cloud");
				pluginList.put(name, pi);
			}
		}
	return jars;
}


/**
 * Scan the load balancer plugins directiory get all the paths of JAR file
 * @see #LoadAllPlugins()
 * @return a list of paths.
 */
```

```
public static List<String> getBalancerPluginList(){
	File dir = new File(System.getProperty("user.dir") + File.separator
			+ LOADBALANCER_PLUGIN_DIR);
	File f = new File(dir.getPath());
	File[] list = f.listFiles();
	List<String> jars = new ArrayList<String>();
		for (File file : list) {
			if (file.getPath().endsWith(".jar")) {
				Plugin pi = new Plugin();
				String name = file.getName().replace(".jar", "");
				String status = "unloaded";
				jars.add(name);
				pi.setPluginName(name);
				pi.setPluginStatus(status);
				pi.setPluginType("balancer");
				pluginList.put(name, pi);
			}
		}
	return jars;
}

/**
 * Scan the batch plugins directiory get all the paths of JAR file
 * @see #LoadAllPlugins()
 * @return a list of paths.
 */
public static List<String> getBatchPluginList(){

	File dir = new File(System.getProperty("user.dir") + File.separator
			+ BATCH_PLUGIN_DIR);
	File f = new File(dir.getPath());
	File[] list = f.listFiles();
	List<String> jars = new ArrayList<String>();
	if(list.length>0){
		for (File file : list) {

			if (file.getPath().endsWith(".jar")) {
				Plugin pi = new Plugin();
				String name = file.getName().replace(".jar", "");
				String status="unloaded";
				jars.add(name);
				pi.setPluginName(name);
				pi.setPluginStatus(status);
				pi.setPluginType("batch");
				pluginList.put(name, pi);
			}
		}
	}
	return jars;
}

/**
 * Unload a specified plugin.remove the plugin form the plugins collection
```

```
	 * @param name a plugin name to be unloaded.
	 * @throws ClassNotFoundException
	 */
	public static void UnloadPlugin(String name) throws ClassNotFoundException {
		if (!isLoaded(name)) {
			throw new ClassNotFoundException("Plugin didn't loaded: " + name);
		}
		pluginList.get(name).setInstance(null);
		pluginList.get(name).setPluginStatus("unloaded");
		if(getBatchPluginList().contains(name)){
			HandleXML.unloadBatchPlugin();
		}
	}

	/**
	 *Get the introduction information about a plugin
	 */
	public static String getInfo(String pluginName) {
		// TODO Auto-generated method stub
		StringBuffer info =new StringBuffer("");
		String path = "";
		if(getCloudPluginList().contains(pluginName))path=CLOUD_PLUGIN_DIR;
		if(getBatchPluginList().contains(pluginName))path=BATCH_PLUGIN_DIR;
		try {
			BufferedReader br = new BufferedReader(new FileReader(new
File(path+File.separator+pluginName+"Readme.txt")));
			String aLine = "";
			while ((aLine = br.readLine()) != null) {
				System.out.println(aLine);
			}
			br.close();

		} catch (IOException e) {
			// TODO Auto-generated catch block
			System.out.println("Readme file doesn't exist ,please check it on the plugin directory!");
		}

		return info.toString();
	}

	/**
	 *Get the plugin status, loaded or not
	 *@return boolean
	 */
	public static boolean isLoaded(String name){

		if (pluginList.get(name).getPluginStatus().equals("loaded")) {
			return true;
		}else{
			return false;
		}
	}
}
```

```
-----------
-----------
-----------


File: project-fermi/src/vcluster/managers/VmManager.java
package vcluster.managers;

import java.util.ArrayList;
import java.util.TreeMap;

import vcluster.Vcluster.uiType;
import vcluster.elements.Cloud;
import vcluster.elements.Vm;
import vcluster.ui.CmdComb;
import vcluster.ui.Command;

/**
 *A class representing the virtual machine manager.
 *Since there only can be one vmmanager, all the functions in this class are static.
 */
public class VmManager extends Thread {

	/**
	 *A static block, initialize the instance of vmlist and the virtual machine's id
	 *@see vmList
	 */
	static{
		vmList = new TreeMap<Integer, Vm>();
		id = 0;
	}


	/**
	 * Get the virtual machine's status
	 * @param vmState, virtual machine's status described in string
	 * @see VMState
	 */
	public static VMState getVMState(String vmState) {

		if(vmState == null) return VMState.NOT_DEFINED;

		if(vmState.equalsIgnoreCase("runn")) return VMState.RUNNING;
		if(vmState.equalsIgnoreCase("penn")) return VMState.PENDING;
		if(vmState.equalsIgnoreCase("prol")) return VMState.PROLOG;

		return VMState.NOT_DEFINED;
	}

	/**
	 * Create certain number of virtual machines on specific cloud by following the command line
	 * @param cmdLine, the string of command line,
	 * @see Command
```

```
	 *
	 */
	public static String createVM(CmdComb cmd) {
		// TODO Auto-generated method stub
		StringBuffer strBuff = new StringBuffer();
		Cloud currCloud = CloudManager.getCurrentCloud();
		int vms = 1;
		ArrayList<String> cmdList = cmd.getParaset();

		if(cmd.getParaset().get(0).equalsIgnoreCase("--help")||cmd.getParaset().get(0).equalsIgnoreCase("-h")){
			strBuff.append("[USAGE : ]"+System.getProperty("line.separator"));
			strBuff.append("vmman"+System.getProperty("line.separator"));
			strBuff.append("    create"+System.getProperty("line.separator"));
			strBuff.append("        [ --help] | "+System.getProperty("line.separator"));
			strBuff.append("        [ -n NUM_VMS | --num=NUM_VMS ] | "+System.getProperty("line.separator"));
			strBuff.append("        [ -c CLOUDNAME | --name=CLOUDNAME]"+System.getProperty("line.separator"));
			if(cmd.getUi().equals(uiType.CMDLINE))System.out.println(strBuff);
			return strBuff.toString();
		}
		if(cmdList.size()==0){
			cmdList.add("-c");
			cmdList.add(currCloud.getCloudName());
			cmdList.add("-n");
			cmdList.add("1");
		}
		if(!cmdList.get(0).equals("-c")){
			cmdList.add(0,"-c");
			cmdList.add(1,currCloud.getCloudName());
		}

		if(cmdList.size()<4||!cmdList.get(2).equalsIgnoreCase("-n")){
			cmdList.add(2, "-n");
			cmdList.add(3, "1");
		}
		if(cmdList.size()<6 || !cmdList.get(4).equalsIgnoreCase("-h")){
			cmdList.add(4, "-h");
			cmdList.add(5, "host1");
		}

		System.out.println("");
		currCloud = CloudManager.getCloudList().get(cmdList.get(1));
		if(currCloud==null){
			strBuff.append("[ERROR : ] Cloud doesn't exist! please check the cloud
name..."+System.getProperty("line.separator"));
			if(cmd.getUi().equals(uiType.CMDLINE))System.out.println(strBuff);
			return strBuff.toString();
		}
		try {
			vms = Integer.parseInt(cmdList.get(3));
		} catch (NumberFormatException e) {
			// TODO Auto-generated catch block
			strBuff.append("Number format is not correct!"+System.getProperty("line.separator"));
			return strBuff.toString();
```

```
		}
		String result = currCloud.createVM(vms,cmdList.get(5));
		if(cmd.getUi().equals(uiType.CMDLINE))System.out.println(result);
			return result;

	}

	/**
	 * List up the virtual machines by following the command line
	 * @param cmdLine, the string of command line
	 * @see Command
	 */
	public static String listVM(CmdComb cmd) {
		// TODO Auto-generated method
		StringBuffer str = new StringBuffer();
		String filter ="";
		String cloudFilter="";
		if( cmd.getParaset().size()>0){
			String tmp = cmd.getParaset().get(0);
			TreeMap<String,Cloud> cc = new TreeMap<String,Cloud>();
			if(tmp.equalsIgnoreCase("-refresh")){
				String name="";
				if(cmd.getParaset().size()>1){
					name = cmd.getParaset().get(1);
					if(!CloudManager.getCloudList().keySet().contains(name)){
						str.append("[ERROR : ] "+ name+ " has not been
loaded!"+System.getProperty("line.separator"));
						System.out.println(str);
						return str.toString();
					}
					cc.put(name, CloudManager.getCloudList().get(name));
					cloudFilter = name;
				}else{
					cc = CloudManager.getCloudList();
				}
				TreeMap<Integer,Vm> temp = new TreeMap<Integer,Vm>();
				for(Cloud cloud : cc.values()){
					if(cloud.isLoaded()){
						cloud.listVMs();
						if(getVmList()==null)continue;
						for(Vm vm : cloud.getVmList().values()){
							boolean flag = true;
							for(Integer id:vmList.keySet()){
								if(vm.getId().equals(vmList.get(id).getId())){
									temp.put(id, vm);
									flag = false;
									break;
								}
							}
							if(flag){
								Integer uId = new Integer(VmManager.getcurrId());
									vm.setuId(uId);
									temp.put(uId, vm);
```

```
                                        }
                                }
                        }
                }

                for(Integer id:temp.keySet()){

                        vmList.put(id, temp.get(id));


                }
                ArrayList<Integer> ids = new ArrayList<Integer>();

                for(Integer id:vmList.keySet()){
                        boolean flag = true;
                        boolean flagx = false;
                        for(Cloud cloud:CloudManager.getCloudList().values()){
                                if(cloud.isLoaded()){
                                        for(Vm vm : cloud.getVmList().values()){
                                                if(vm.getId().equals(vmList.get(id).getId())){
                                                        flag = false;
                                                        flagx = true;

                                                        break;
                                                }
                                        }
                                        if(flagx){flagx = false;break;}
                        }
                        }
                        if(flag)ids.add(id);
                }
                for(Integer id:ids){
                        vmList.remove(id);
                }

            }else if(tmp.equalsIgnoreCase("-a")||tmp.equalsIgnoreCase("--type=available")){
                filter="RUNNING";
            }else if(tmp.equalsIgnoreCase("-r")||tmp.equalsIgnoreCase("--type=running")){
                filter="RUNNING";
            }else if(tmp.equalsIgnoreCase("-s")||tmp.equalsIgnoreCase("--type=suspended")){
                filter="SUSPEND";
            }else if(tmp.equalsIgnoreCase("-n")){
                if(cmd.getParaset().size()>1){
                        cloudFilter = cmd.getParaset().get(1);
                }else{
                        str.append("[ERROR : ] Expected a cloud name!"+System.getProperty("line.separator"));
                        System.out.println(str);
                        return str.toString();
                }
            }
        }

        String tName = String.format("%-10s", "CLOUD");
```

```
        String tHost = String.format("%-10s", "Host");
        String tID =String.format("%-5s", "ID");
        String tStat = String.format("%-12s", "STATUS");
        String tPrivateIp =  String.format("%-18s", "PrivateIP");
        String tPublicIp =  String.format("%-18s", "PublicIP");
        String tActivity =  String.format("%-10s", "Activity");
        String tInternalID =  String.format("%-12s", "IntnlID");
        String tDNSName = String.format("%-25s", "VMName");
        String line =  "-----------------------------------------------------------------------------------------------------
";
        str.append(line+System.getProperty("line.separator"));

    str.append(tID+tInternalID+tStat+tActivity+tPrivateIp+tPublicIp+tHost+tName+tDNSName+System.getProperty("line.separator
"));
        str.append(line+System.getProperty("line.separator"));
            int runNum = 0;
            int totalNum = 0;
            for(Integer id : vmList.keySet()){
                Vm vm = vmList.get(id);
                String acti = "busy";
                if(vm.isIdle()==0){acti = "idle";}else if(vm.isIdle()==3){acti="none";}
                if(!cloudFilter.equals("")&&vmList.get(id).getCloudName().equals(cloudFilter)){
                        String fName = String.format("%-10s", vmList.get(id).getCloudName());
                        String fID =String.format("%-5s", id);
                        String fStat = String.format("%-12s", vmList.get(id).getState());
                        String fPrivateIp =  String.format("%-18s", vm.getPrivateIP());
                        String fPublicIp =  String.format("%-18s", vm.getPubicIP());
                        String fActivity =  String.format("%-10s", acti);
                        String fInternalID =  String.format("%-12s", vm.getId());
                        String fHost = String.format("%-10s", vm.getHostname());
                        String fDnsName = String.format("%-20s", vm.getDNSName());


                        if(vmList.get(id).getState().equals(VMState.RUNNING))runNum++;

    str.append(fID+fInternalID+fStat+fActivity+fPrivateIp+fPublicIp+fHost+fName+fDnsName+System.getProperty("line.separator
"));
                        totalNum++;
                }
                if(!filter.equals("")&&vmList.get(id).getState().toString().equals(filter)){
                        String fName = String.format("%-10s", vmList.get(id).getCloudName());
                        String fID =String.format("%-5s", id);
                        String fStat = String.format("%-12s", vmList.get(id).getState());
                        String fPrivateIp =  String.format("%-18s", vm.getPrivateIP());
                        String fPublicIp =  String.format("%-18s", vm.getPubicIP());
                        String fActivity =  String.format("%-10s", acti);
                        String fInternalID =  String.format("%-12s", vm.getId());
                        String fHost = String.format("%-10s", vm.getHostname());
                        String fDnsName = String.format("%-20s", vm.getDNSName());
                        if(vmList.get(id).getState().equals(VMState.RUNNING))runNum++;

    str.append(fID+fInternalID+fStat+fActivity+fPrivateIp+fPublicIp+fHost+fName+fDnsName+System.getProperty("line.separator
"));
```

```
                                        totalNum++;
                                }
                                if(filter.equals("")&cloudFilter.equals("")){
                                        String fName = String.format("%-10s", vmList.get(id).getCloudName());
                                        String fID =String.format("%-5s", id);
                                        String fStat = String.format("%-12s", vmList.get(id).getState());
                                        String fPrivateIp =  String.format("%-18s", vm.getPrivateIP());
                                        String fPublicIp =  String.format("%-18s", vm.getPubicIP());
                                        String fActivity =  String.format("%-10s", acti);
                                        String fInternalID =  String.format("%-12s", vm.getId());
                                        String fHost = String.format("%-10s", vm.getHostname());
                                        String fDnsName = String.format("%-20s", vm.getDNSName());
                                        if(vmList.get(id).getState().equals(VMState.RUNNING))runNum++;

        str.append(fID+fInternalID+fStat+fActivity+fPrivateIp+fPublicIp+fHost+fName+fDnsName+System.getProperty("line.separator
"));
                                        totalNum++;
                                }
                        }
                        str.append(line+System.getProperty("line.separator"));
                        str.append("      Running VMs :  "+runNum+"/"+totalNum+System.getProperty("line.separator"));
                        str.append(line+System.getProperty("line.separator"));
                        if(cmd.getUi().equals(uiType.CMDLINE))System.out.println(str);
                return str.toString();
        }

        /**
         * Show the virtual machine's information in detail,such as ID, public IP, private IP, launch time, etc.
         * @param cmdLine, the command line string.
         * @see Command
         */
        public static String showVM(CmdComb cmd) {
                // TODO Auto-generated method stub

                String vmID = "";
                Integer vID;
                ArrayList<String> paras = cmd.getParaset();
                StringBuffer str = new StringBuffer();

                if(paras.size()==0){
                                str.append("[ERROR : ] Expected a VM ID");
                                System.out.println(str);
                                return str.toString();
                        }
                String tmp = paras.get(0);

                if(tmp.equalsIgnoreCase("-h")||tmp.equalsIgnoreCase("--help")){
                        str.append("[USAGE : ]");
                        str.append("vmman");
                        str.append("    show VM_ID");
                        str.append("        [ --help] |-h ");
                        System.out.println(str);
                        return str.toString();
```

```
                }else{
                        vmID = tmp;
                        try{
                                vID =new Integer(vmID);
                        }catch (NumberFormatException e){
                                str.append("[ERROR : ] VM ID has to be a number!");
                                System.out.println(str);
                                return str.toString();
                        }
                        if(!vmList.keySet().contains(vID)){

                                str.append("[ERROR : ] VM ID doesn't exist!");
                                System.out.println(str);
                                return str.toString();
                        }

                }
                String tName = String.format("%-11s", "CLOUD");
                String tID =String.format("%-11s", "ID ");
                String tStat = String.format("%-11s", "STATUS");
                String tpriIP = String.format("%-11s", "privateIP");
                String tpubIP = String.format("%-11s", "publicIP");
                String tinID = String.format("%-11s", "internalID");
                String tlauTime = String.format("%-11s", "launchTime");
                String activity = String.format("%-11s", "Activity");
                String tDNSName = String.format("%-11s", "VmName");
                Integer id = vID;

                String fName = String.format("%-16s", vmList.get(id).getCloudName());
                String fID =String.format("%-16s", id);
                String fStat = String.format("%-16s", vmList.get(id).getState());
                String fpriIP = String.format("%-16s", vmList.get(id).getPrivateIP());
                String fpubIP =String.format("%-16s", vmList.get(id).getPubicIP());
                String finID = String.format("%-16s", vmList.get(id).getId());
                String flauTime = String.format("%-16s", vmList.get(id).getTime());
                String factivity = String.format("%-16s", vmList.get(id).isIdle());
                String fDNSName = String.format("%-25s", vmList.get(id).getDNSName());

                str.append("--------------------------------------"+System.getProperty("line.separator"));
                str.append(tID+" : "+fID+System.getProperty("line.separator"));
                str.append(tName+" : "+fName+System.getProperty("line.separator"));
                str.append(tinID+" : "+finID+System.getProperty("line.separator"));
                str.append(tDNSName+" : "+fDNSName+System.getProperty("line.separator"));
                str.append(tStat+" : "+fStat+System.getProperty("line.separator"));
                str.append(tpriIP+" : "+fpriIP+System.getProperty("line.separator"));
                str.append(tpubIP+" : "+fpubIP+System.getProperty("line.separator"));
                str.append(activity+" : "+factivity+System.getProperty("line.separator"));
                str.append(tlauTime+" : "+flauTime+System.getProperty("line.separator"));
                str.append("--------------------------------------"+System.getProperty("line.separator"));

                if(cmd.getUi().equals(uiType.CMDLINE))System.out.println(str);
                return str.toString();
        }
```

```java
	/**
	 * Terminate the given virtual machine by following the command line.
	 * @param cmdLine, the string of command line
	 * @see Command
	 */
	public static String destroyVM(CmdComb cmd) {
		// TODO Auto-generated method stub
		StringBuffer str = new StringBuffer();

		String vmID = "";
		Integer vID;

		if(cmd.getParaset().size()==0){
			str.append("[USAGE : ]");
			str.append("vmman");
			str.append("    destroy VM_ID");
			str.append("        [ --help] |-h ");
			if(cmd.getUi().equals(uiType.CMDLINE))System.out.println(str);
			return str.toString();
		}

		String tmp = cmd.getParaset().get(0);

		if(tmp.equalsIgnoreCase("-h")||tmp.equalsIgnoreCase("--help")){
			str.append("[USAGE : ]");
			str.append("vmman");
			str.append("    destroy VM_ID");
			str.append("        [ --help] |-h ");
			System.out.println(str);
			return str.toString();
		}else{
			vmID = tmp;
			try{
				vID =new Integer(vmID);
			}catch (NumberFormatException e){
				str.append("[ERROR : ] VM ID has to be a number!");
				return str.toString();
			}
			if(!vmList.keySet().contains(vID)){
				str.append("[ERROR : ] VM ID doesn't exist!");
				System.out.println(str);
				return str.toString();
			}

		}
		String cloudName = vmList.get(vID).getCloudName();
		Vm vm = vmList.get(vID);
		Cloud cloud = CloudManager.getCloudList().get(cloudName);
		cloud.destroyVM(vm);
		vmList.remove(vID);
		return str.toString();
	}
```

```java
	/**
	 * Suspend the given virtual machine by following the command line.the virtual machine's original status must be
running.
	 * @param cmdLine, the string of command line
	 * @see Command
	 */
	public static String suspendVM(CmdComb cmd) {
		// TODO Auto-generated method stub
		StringBuffer str = new StringBuffer();
		String cloudName = "";
		String vmID = "";
		if(cmd.getParaset().size()==0){
			str.append("[USAGE : ]"+System.getProperty("line.separator"));
			str.append("vmman"+System.getProperty("line.separator"));
			str.append("    suspend VM_ID"+System.getProperty("line.separator"));
			str.append("        [ --help] |-h "+System.getProperty("line.separator"));
			System.out.println(str);
			return str.toString();
		}
		Integer vID;

		String tmp = cmd.getParaset().get(0);

		if(tmp.equalsIgnoreCase("-h")||tmp.equalsIgnoreCase("--help")){
			str.append("[USAGE : ]"+System.getProperty("line.separator"));
			str.append("vmman"+System.getProperty("line.separator"));
			str.append("    suspend VM_ID"+System.getProperty("line.separator"));
			str.append("        [ --help] |-h "+System.getProperty("line.separator"));
			System.out.println(str);
			return str.toString();
		}else{
			vmID = tmp;
			try{
				vID =new Integer(vmID);
			}catch (NumberFormatException e){
				str.append("[ERROR : ] VM ID has to be a number!");
				System.out.println(str);
				return str.toString();
			}
			if(!vmList.keySet().contains(vID)){
				str.append("[ERROR : ] VM ID doesn't exist!");
				System.out.println(str);
				return str.toString();
			}

		}
		cloudName = vmList.get(vID).getCloudName();
		String inId = vmList.get(vID).getId();
		Cloud cloud = CloudManager.getCloudList().get(cloudName);
		cloud.suspendVM(inId);
		if(cmd.getUi().equals(uiType.CMDLINE))System.out.println(str);
```

```
        return str.toString();
}


/**
 * Start the given suspended virtual machine.
 * @param cmdLine, the string of command line.
 * @see Command
 */
public static String startVM(CmdComb cmd) {
        // TODO Auto-generated method stub
        StringBuffer str = new StringBuffer();
        String cloudName = "";
        String vmID = "";

        if(cmd.getParaset().size()==0){
                str.append("[USAGE : ]"+System.getProperty("line.separator"));
                str.append("vmman"+System.getProperty("line.separator"));
                str.append("    start VM_ID"+System.getProperty("line.separator"));
                str.append("        [ --help] |-h "+System.getProperty("line.separator"));
                System.out.println(str);
                return str.toString();
        }
        Integer vID;

        String tmp = cmd.getParaset().get(0);

        if(tmp.equalsIgnoreCase("-h")||tmp.equalsIgnoreCase("--help")){
                str.append("[USAGE : ]"+System.getProperty("line.separator"));
                str.append("vmman"+System.getProperty("line.separator"));
                str.append("    start VM_ID"+System.getProperty("line.separator"));
                str.append("        [ --help] |-h "+System.getProperty("line.separator"));
                System.out.println(str);
                return str.toString();
        }else{
                vmID = tmp;
                try{
                        vID =new Integer(vmID);
                }catch (NumberFormatException e){
                        str.append("[ERROR : ] VM ID has to be a number!");
                        System.out.println(str);
                        return str.toString();
                }
                if(!vmList.keySet().contains(vID)){
                        str.append("[ERROR : ] VM ID doesn't exist!");
                        System.out.println(str);
                        return str.toString();
                }

        }
        cloudName = vmList.get(vID).getCloudName();
        String inId = vmList.get(vID).getId();
        Cloud cloud = CloudManager.getCloudList().get(cloudName);
```

```
        cloud.startVM(inId);

        return str.toString();
}


/**
 * Get the current Id of virtual machine.when a virtual machine is created, this function would be invoked.
 * @return id,
 */
public static int getcurrId() {
        return id++;
}


/**
 * Get the virtual machines list
 * @return vmList, a TreeMap of virtual machine's instance, the key set is the virtual machine's id.
 */
public static TreeMap<Integer,Vm> getVmList() {
        return vmList;
}

/**
 *Migrate a given virtual machine to a specific host by following the command line
 *@param cmdLine, the string of command line.
 */
public static String migrate(CmdComb cmd) {
        // TODO Auto-generated method stub
        StringBuffer  str = new StringBuffer();
        if (cmd.getParaset().size()==0) {
                str.append("[ERROR : ] Expect a cloud name!");
                System.out.println(str);
                return str.toString();
        }
        String cloudname = cmd.getParaset().get(0).trim();
        if (cmd.getParaset().size()<2) {
                str.append("[ERROR : ] Expect the source VM id!");
                System.out.println(str);
                return str.toString();
        }
        String vmID = cmd.getParaset().get(1);
        if (cmd.getParaset().size()<3) {
                str.append("[ERROR : ] Expect a target host id!");
                System.out.println(str);
                return str.toString();
        }
        String hostID =cmd.getParaset().get(2);


        Cloud cloud = CloudManager.getCloudList().get(cloudname);
```

```
		Integer vID;
		try{
			vID =new Integer(vmID);
		}catch (NumberFormatException e){
			str.append("[ERROR : ] VM ID has to be a number!");
			System.out.println(str);
			return str.toString();
		}
		if(!vmList.keySet().contains(vID)){
			str.append("[ERROR : ] VM ID doesn't exist!");
			System.out.println(str);
			return str.toString();
		}

	String inId = vmList.get(vID).getId();

		String result = cloud.migrate(inId, hostID);
		if(cmd.getUi().equals(uiType.CMDLINE))System.out.println(result);
		return result;
	}

	/**
	 *Definition of the virtual machine's status
	 */
	public enum VMState {STOP, PENDING, RUNNING, SUSPEND, PROLOG, NOT_DEFINED, FAILED };

	/**
	 *The virtual machine's id, it's the global id in vcluster,  cloud independent.
	 */
	private static int id;
	/**
	 * The virtual machines's list, the key is the id of the virtual machine, the value is the instance of a virtual
machine.
	 */
	private static TreeMap <Integer, Vm> vmList;

   }


-----------
-----------
-----------


File: project-fermi/src/vcluster/plugInterfaces/BatchInterface.java
package vcluster.plugInterfaces;


import java.io.File;
```

```
import vcluster.elements.PoolStatus;

import vcluster.elements.QStatus;


/**

 * All the batch system plug-ins should implement this interface.

 *

 */

public interface BatchInterface{

	/**

	 *Connect to batch system central manager,

	 *@param conf, A text file includes the connection informations.

	 */

	public boolean ConnectTo(File conf);

	/**

	 *Get the pool status .

	 *@return PoolStatus, return a instance of pool status, which includes all the pool status information.

	 *@see PoolStatus.

	 */

	public PoolStatus getPoolStatus();

	/**

	 * Get the queue status.

	 * @return QStatus, A instance of the job queue status, which includes the queue status information.

	 * @see QStatus.

	 */

	public QStatus getQStatus();
```

```
        public boolean deregisterWorker(String dnsname);



}

-----------
-----------
-----------

File: project-fermi/src/vcluster/plugInterfaces/CloudInterface.java
package vcluster.plugInterfaces;


import java.util.ArrayList;

import java.util.List;


import vcluster.elements.Vm;

import vcluster.elements.Host;

/**

 *An interface for cloud plug-in, all the cloud plugins implement this interface.

 */

public interface CloudInterface  {

        /**

         * Connect to a cloud system.

         * @param Configurations, a collection of cloud connection informations.

         */

        public boolean RegisterCloud(List<String> configurations);

        /**

         *Create given numbers of virtual machines.
```

```
         *@param maxCount, the number of virtual machines that's suppose to be created.

         */

        public ArrayList<Vm> createVM(int maxCount);


        /**

         * Create given number of virtual machine on given host

         * @param maxCount, the number of virtual machines that to be launched

         * @param hostID, the id of the host

         */

        public ArrayList<Vm> createVM(int maxCount, String hostID);

        /**

         *Get the list of virtual machines.

         *@return ArrayList<Vm>, an array list of virtual machine instances .

         */

        public ArrayList<Vm> listVMs();

        /**

         *Terminate a given virtual machine.

         *@param id, the id that's suppose to be terminated.

         *

         */

        public ArrayList<Vm> destroyVM(Vm vm);

        /**

         *Start a given virtual machine from suspend status.

         *@param id, the id of the virtual machine that is suppose to start.

         */

        public ArrayList<Vm> startVM(String id);
```

```
        /**
         *Suspend a given virtual machine.
         *@param id, the id of the victual machine that's suppose to be suspended.
         */
        public ArrayList<Vm> suspendVM(String id);
        /**
         * Turn on a given host
         * @param ipmiID, the ipmi id of the host.
         */
        public boolean hoston(String ipmiID);
        /**
         *Turn the the give host's power off
         *@param ipmiID, the ipmi id of the host.
         */
        public boolean hostoff(String ipmiID);
        /**
         *Migrate a given virtual machine to the target host.
         *@param vmid, the specified virtual machine's id
         *@param hostid, the target host's id.virtual machine would be migrated to it.
         */
        public boolean migrate(String vmid,String hostid);


        /**
         * List host machine information
         */
        public ArrayList<Host> listHost();
```

```
}

-----------
-----------
-----------

File: project-fermi/src/vcluster/plugInterfaces/LoadBalancer.java
package vcluster.plugInterfaces;

/**
 *Load balancer interface, all the load balancer plug-ins implement this interface.
 */
public interface LoadBalancer {


        /**
         *Vcluster invoke this function to activate the load balancer in a new thread.
         */
        public void activate();
}

-----------
-----------
-----------

File: project-fermi/src/vcluster/ui/CmdComb.java
package vcluster.ui;


import java.util.ArrayList;
import java.util.StringTokenizer;
import vcluster.Vcluster.uiType;
import vcluster.ui.Command.CMD_GROUP;
```

```
/**
 *This class represents a command set, a command set  consist of command group, command,and the parameters set
 */
public class CmdComb {

	/**
	 *Construct a command combination by the command line.
	 */
	public CmdComb(String cmdLine){
		this.cmdLine = cmdLine;
		paraSet = new ArrayList<String>();
		cmdLineParser(cmdLine);
	 ui=uiType.CMDLINE;

	}

	/**
	 * Resolve the command line string,separate the  command group, command ,and command parameters set.
	 */
	private void cmdLineParser(String cmdLine){
		StringTokenizer st = new StringTokenizer(cmdLine);
		String strGroup = null;

		if(st.hasMoreTokens()){
			strGroup = st.nextToken().trim();
```

```
		}else{

		}
		switch (strGroup){
			case "plugman":
				cmdGroup = CMD_GROUP.PLUGMAN;
				break;

			case "cloudman":
				cmdGroup = CMD_GROUP.CLOUDMAN;
				break;
			case "vmman":
				cmdGroup = CMD_GROUP.VMMAN;
				break;
			default:
				cmdGroup = CMD_GROUP.VCLMAN;

				break;
		}
		String cmdStr ="";
		if(cmdGroup==CMD_GROUP.VCLMAN){
			cmdStr = strGroup;
			cmd = Command.getCommand(cmdGroup, cmdStr);
		}else if(st.hasMoreTokens()){
			cmdStr = st.nextToken();
			cmd = Command.getCommand(cmdGroup, cmdStr);
```

```
        }else{
                cmd = Command.VHELP;
        }
        boolean flag = true;
        while(flag){
                if(st.hasMoreTokens()){
                        paraSet.add(st.nextToken());
                }else{
                        flag = false;
                }
        }


}
/**
 *Get the command line string.
 *@return String cmdLine
 */
public String getCmdLine() {
        return cmdLine;
}

/**
 *Set the command line string
 *@param String cmdLine
 */
```

```
public void setCmdLine(String cmdLine) {
        this.cmdLine = cmdLine;
}

/**
 *Get the parameters set
 *@return ArrayList of parameters
 */
public ArrayList<String> getParaset() {
        return paraSet;
}

/**
 * Set the parameters set
 * @param ArrayList of parameters
 */
public void setParas(ArrayList<String> paraset) {
        this.paraSet = paraset;
}

/**
 * Get the command launch time.
 * @return launch time
 */
public long getLaunchTime() {
        return launchTime;
```

```
}

/**
 * Set the launch time of the command
 * @param value of time in long format
 */
public void setLaunchTime(long launchTime) {
	this.launchTime = launchTime;
}


/**
 * Get the group the command belongs to
 */
public CMD_GROUP getCmdGroup() {
	return cmdGroup;
}

/**
 * Set the group the command belongs to
 */
public void setCmdGroup(CMD_GROUP cmdGroup) {
	this.cmdGroup = cmdGroup;
}

/**
```

```
 * Get the command
 */
public Command getCmd() {
	return cmd;
}

/**
 * Set the command
 */
public void setCmd(Command cmd) {
	this.cmd = cmd;
}


/**
 * Get the ip address where command from
 */
public String getSourceIp() {
	return sourceIp;
}

/**
 * Set the ip address where the command from
 */
public void setSourceIp(String sourceIp) {
	this.sourceIp = sourceIp;
```

```
	}

	/**
	 * Get the UI type where the command from
	 */
	public uiType getUi() {
		return ui;
	}

	/**
	 * Set the UI type where the command from
	 */
	public void setUi(uiType ui) {
		this.ui = ui;
	}



	private String cmdLine;
	private ArrayList<String> paraSet;
	private long launchTime;
	private CMD_GROUP cmdGroup;
	private Command cmd;
	private String sourceIp;
	private uiType ui;
```

```
}

-----------
-----------
-----------

File: project-fermi/src/vcluster/ui/Command.java
package vcluster.ui;

import java.util.ArrayList;
import java.util.List;
import java.util.StringTokenizer;
/**
 * @author S.Y. Noh
 * This is the enum of commands. all the vcluster command is listed here
 *
 */
public enum Command {
	/** Commands of vclman Category*/
	VHELP(CMD_GROUP.VCLMAN,"-h,--help,help"),
	QUIT (CMD_GROUP.VCLMAN, "quit, exit"),
	CHECK_P (CMD_GROUP.VCLMAN, "chkp,check_p"),
	CHECK_Q (CMD_GROUP.VCLMAN, "chkq,check_q"),
	SERVER_MODE(CMD_GROUP.VCLMAN,"servermode"),
	LOADBALANCERSTART(CMD_GROUP.VCLMAN,"loadbalancer, lb"),
	AUTOSERVER(CMD_GROUP.VCLMAN,"auto"),


	/*Commands of cloudman Category*/
	REGISTER(CMD_GROUP.CLOUDMAN,"register,rgst"),
	LISTCLOUD(CMD_GROUP.CLOUDMAN,"list,ls"),
	LOADCLOUD(CMD_GROUP.CLOUDMAN,"load,ld"),
	UNLOADCLOUD(CMD_GROUP.CLOUDMAN,"unload,unld"),
	DUMP(CMD_GROUP.CLOUDMAN,"show,dump"),
	HOSTON(CMD_GROUP.CLOUDMAN,"hoston"),
	HOSTOFF(CMD_GROUP.CLOUDMAN,"hostoff"),
	LISTHOST(CMD_GROUP.CLOUDMAN,"listhost,hostlist"),
	SHOWHOST(CMD_GROUP.CLOUDMAN,"showhost"),


	/*Commands of vmman Category*/
	SHOW(CMD_GROUP.VMMAN,"show,sh"),
	CREATE (CMD_GROUP.VMMAN, "create,crt"),
	START (CMD_GROUP.VMMAN, "start,st"),
```

```
        SUSPEND (CMD_GROUP.VMMAN, "suspend,ssp"),
        LISTVM (CMD_GROUP.VMMAN, "list,ls"),
        DESTROY (CMD_GROUP.VMMAN, "destroy,dt"),
        MIGRATE(CMD_GROUP.VMMAN,"migrate,mig"),


        /*Commands of plugman Category*/
        LOAD (CMD_GROUP.PLUGMAN, "load,ld"),
        UNLOAD (CMD_GROUP.PLUGMAN, "unload,uld"),
        LIST (CMD_GROUP.PLUGMAN, "list,ls"),
        INFO (CMD_GROUP.PLUGMAN, "info,ifo"),



        NOT_DEFINED (CMD_GROUP.NOT_DEFINED, "not_defined");


        private String command;
        private CMD_GROUP cmdGroup;
        private List<String> cmdList;

        /**
         *Constructor of command, specify the command name and group
         *@param group specify the command group
         *@param cmdString is the command line
         *
         */
        Command(CMD_GROUP group, String cmdString) {

                cmdGroup = group;
                this.cmdList = new ArrayList<String>();
                StringTokenizer st = new StringTokenizer(cmdString, ", ");

                boolean firstToken = true;
                String token = null;

                while(st.hasMoreTokens()) {
                        token = st.nextToken();
                        if (firstToken) {
                                command = token;
                                firstToken = false;
                        }
                        cmdList.add(token);
                }
        }

        public CMD_GROUP getCmdGroup()
        {
                return cmdGroup;
        }

        public void toPrint()
        {
```

```
                System.out.println("Key = " + command);
                for(int i = 0; i < cmdList.size(); i++)
                        System.out.println("\t cmd: " + cmdList.get(i));
        }

        public boolean contains(String aCmd)
        {
                return cmdList.contains(aCmd);
        }


        public String getCommand()
        {
                return command;
        }

        public static Command getCommand(CMD_GROUP cmdGroup, String aCmdLine)
        {
                StringTokenizer st = new StringTokenizer(aCmdLine);
                String aCmd = st.nextToken().trim();
        if (cmdGroup==null){
              for (Command cmd : Command.values()){
                if (cmd.contains(aCmd)) return cmd;
              }
        }
         for (Command cmd : Command.values()){
                if (cmd.getCmdGroup().equals(cmdGroup)&cmd.contains(aCmd)) return cmd;
         }
         return Command.NOT_DEFINED;
        }

        /**
         *This is enum of CMD_GROUP, all the command groups are defined here.
         *
         */
        public enum CMD_GROUP {
                VCLMAN,
                VMMAN,
                CLOUDMAN,
                PLUGMAN,
                NOT_DEFINED;


        }

}

------------
------------
------------


File: project-fermi/src/vcluster/ui/remote/CmdServer.java
```

```
package vcluster.ui.remote;


import java.io.IOException;

import java.net.MalformedURLException;

import java.rmi.Naming;

import java.rmi.RemoteException;

import java.rmi.registry.LocateRegistry;

import java.rmi.server.RMISocketFactory;


import vcluster.Vcluster;

/**

 *This class starts the vcluster as the server mode

 */

public class CmdServer {

	/**

	 *This function initiates vcluster server

	 */

	public static void initiate(){

		try {

			System.setProperty("java.rmi.server.hostname",Vcluster.SERVER_ADDR);

			RMISocketFactory.setSocketFactory (new MyRMISocket());

			LocateRegistry.createRegistry(Vcluster.SERVER_PORT1);

			RemoteCmd rcmd = new RemoteCmd("command");

			Naming.rebind("command", rcmd);

			System.out.println("Vcluster server is running... ...");
```

```
		} catch (RemoteException e) {

			// TODO Auto-generated catch block

			e.printStackTrace();

		} catch (MalformedURLException e) {

			// TODO Auto-generated catch block

			e.printStackTrace();

		} catch (IOException e) {

			// TODO Auto-generated catch block

			e.printStackTrace();

		}

	}

}

-----------
-----------
-----------

File: project-fermi/src/vcluster/ui/remote/MyRMISocket.java
package vcluster.ui.remote;


import java.rmi.server.*;

import java.io.*;

import java.net.*;


import vcluster.Vcluster;



public class MyRMISocket extends RMISocketFactory

{
```

```
        public Socket createSocket(String host, int port) throws IOException
        {
                return new Socket(host,port);
        }

        public ServerSocket createServerSocket(int port) throws IOException
        {
                 if (port == 0)
                    port = Vcluster.SERVER_PORT2;
                return new ServerSocket(port);
        }

}
-----------
-----------
-----------

File: project-fermi/src/vcluster/ui/remote/RemoteCmd.java
package vcluster.ui.remote;


import java.net.InetAddress;
import java.net.UnknownHostException;
import java.rmi.RemoteException;
import java.rmi.server.RemoteServer;
import java.rmi.server.ServerNotActiveException;
import java.rmi.server.UnicastRemoteObject;


import vcluster.Vcluster.uiType;
import vcluster.executors.CmdCataloger;
```

```
import vcluster.ui.CmdComb;
import vcluster.ui.Command;

/**
 *This class involve the function that can receive the command from remote client.
 */
public class RemoteCmd extends UnicastRemoteObject implements RemoteInterface{



        private static final long serialVersionUID = 1L;

        public RemoteCmd(String cmd) throws RemoteException {
        }

        /**
         * This function gets the command from the remote client.
         */
        public String getRemoteCommand(String command) {
                // TODO Auto-generated method stub
                CmdComb cmd = new CmdComb(command);

            /*
             * if command is not defined, no more action
             */
            if (cmd.getCmd() == Command.NOT_DEFINED) {
                return "command is not supported!";
```

```
            }

                cmd.setUi(uiType.REMOTECLIENT);
                String result = CmdCataloger.execute(cmd).toString();
                String clienthost = "";
                String clientIP="";
                try {
                        clienthost =RemoteServer.getClientHost();
                        InetAddress ia = InetAddress.getByName(clienthost);
                        clientIP = ia.getHostAddress();
                } catch (ServerNotActiveException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                } catch (UnknownHostException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                StringBuffer str = new StringBuffer();

                String uistr = String.format("%-9s", "UIType");
                String clientstr = String.format("%-9s", "clientIP");
                String cmdstr = String.format("%-9s", "Command");



                String uiType = String.format("%-16s", "RemoteClient");
```

```
                String clientAddr = String.format("%-16s", clientIP);
                String fcmd = String.format("%-16s", command);
                str.append(System.getProperty("line.separator")+"Request from client: "+System.getProperty("line.separator"));
                str.append("--------------------------------------------"+System.getProperty("line.separator"));
                str.append(uistr+" : "+uiType+System.getProperty("line.separator"));
                str.append(clientstr+" : "+clientAddr+System.getProperty("line.separator"));
                str.append(cmdstr+" : "+fcmd+System.getProperty("line.separator"));
                str.append("--------------------------------------------"+System.getProperty("line.separator"));

                System.out.println(str);
                return result;
        }

}

-----------
-----------
-----------

File: project-fermi/src/vcluster/ui/remote/RemoteInterface.java
package vcluster.ui.remote;


import java.rmi.Remote;
import java.rmi.RemoteException;
/**
 *This is the interface that the server program need to be implemented
 */
public interface RemoteInterface extends Remote{
        /**
```

```
         *The remote client invokes this function.

         */

        public String getRemoteCommand(String command) throws RemoteException;



}


-----------
-----------
-----------


File: project-fermi/src/vcluster/ui/UIMain.java
package vcluster.ui;

import java.io.BufferedReader;
import java.io.InputStreamReader;

import vcluster.Vcluster;
import vcluster.Vcluster.uiType;
import vcluster.executors.CmdCataloger;



/**
 * This is the main class of vcluster, the main function is involved in this class.
 * @author S.Y.Noh
 * @version 1.0
 *
 */
public class UIMain {


        public UIMain()
        {
                Vcluster.init();
        }


        /**
         * Judge if the input command line is empty or not.Space character is regarded as empty.
         * @return boolean type.
         */
        private boolean isEmpty(String aCmd)
        {
                if (aCmd == null) return true;

                String cmd = aCmd.replaceAll(" ", "");
                return cmd.isEmpty();
```

```
        }

        /**
         * Generate "vcluster" prompt, get the input command line and send it to the executor.
         *
         * @return true if it has to be continue; otherwise false
         */
        private  boolean promptGen() {

            String userCmd = "";

            /* prompt */
            System.out.print("vcluter > ");

            InputStreamReader input = new InputStreamReader(System.in);
            BufferedReader reader = new BufferedReader(input);

            try {
                    /* get a command string */
               userCmd = reader.readLine();
               //userCmd+=" --cmdLine";
            }
            catch(Exception e){return true;}

            /* check emptiness, then just no more action */
            if (isEmpty(userCmd))
               return true;

            /* extract the command from the command string */
            CmdComb command = new CmdComb(userCmd);
            command.setUi(uiType.CMDLINE);
            /* if quit, then forcedly quit */
            if (command.getCmd() == Command.QUIT) {
               vcluster.Vcluster.terminate=true;
               /* forcedly exit */
               System.exit(0);
            }

            /*
             * if command is not defined, no more action
             */
            if (command.getCmd() == Command.NOT_DEFINED) {
               System.out.println("command is not supported!");
               return true;
            }

            /* execution here */
            CmdCataloger.execute(command);

            return true;

        }
```

```
	private static boolean startCmdLine(String cmd){
		String userCmd = cmd;


	    /* extract the command from the command string */
	    CmdComb command = new CmdComb(userCmd);
	    command.setUi(uiType.CMDLINE);
	    /* if quit, then forcedly quit */
	    if (command.getCmd() == Command.QUIT) {
	       vcluster.Vcluster.terminate=true;
	       /* forcedly exit */
	       System.exit(0);
	    }

	    /*
	     * if command is not defined, no more action
	     */
	    if (command.getCmd() == Command.NOT_DEFINED) {
	       System.out.println("command is not supported!");
	       return true;
	    }

	    /* execution here */
	    CmdCataloger.execute(command);

	    return true;
	}

	/**
	 *This is the main function, loops the promptGen function, keep staying at the "vcluster " prompt line, wait for
inputing the command.
	 *
	 */
	public static void main(String[] args) throws Exception {
		UIMain uimain = new UIMain();



		if(args.length>=1){
			startCmdLine(args[0]);

		}else{
			boolean more = false;
			do {
				more = uimain.promptGen();
			}while (more == true);
		}
	}



}
```

```
-----------
-----------
-----------


File: project-fermi/src/vcluster/util/GetSignature.java
package vcluster.util;


import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import org.apache.axis.encoding.Base64;

public class GetSignature {

	/**
	 * Computes RFC 2104-compliant HMAC signature.
	 *
	 * @param data
	 *            The data to be signed.
	 * @param key
	 *            The signing key.
	 * @return The base64-encoded RFC 2104-compliant HMAC signature.
	 * @throws java.security.SignatureException
	 *             when signature generation fails
	 * @throws NoSuchAlgorithmException
	 * @throws InvalidKeyException
	 */

	public static String calculateRFC2104HMAC(String data, String key, String sigMethod)
	throws NoSuchAlgorithmException, InvalidKeyException {
		String result;
		// get an hmac_sha1 key from the raw key bytes
		SecretKeySpec signingKey = new SecretKeySpec(key.getBytes(), sigMethod);

		// get an hmac_sha1 Mac instance and initialize with the signing key
		Mac mac = Mac.getInstance(sigMethod);
		mac.init(signingKey);


		// compute the hmac on input data bytes
		byte[] rawHmac = mac.doFinal(data.getBytes());
		result = Base64.encode(rawHmac);

		return result;
	}

}
-----------
-----------
```

```
-----------

File: project-fermi/src/vcluster/util/HandleXML.java
package vcluster.util;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Calendar;

import java.util.Date;

import java.util.List;


import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.ParserConfigurationException;

import javax.xml.transform.Transformer;

import javax.xml.transform.TransformerConfigurationException;

import javax.xml.transform.TransformerException;

import javax.xml.transform.TransformerFactory;

import javax.xml.transform.dom.DOMSource;

import javax.xml.transform.stream.StreamResult;


import org.w3c.dom.Document;

import org.w3c.dom.Element;

import org.xml.sax.SAXException;
```

```
import vcluster.elements.Cloud;

import vcluster.executors.PlugmanExecutor;

import vcluster.managers.CloudManager;

import vcluster.ui.CmdComb;


public class HandleXML {

	private static DocumentBuilderFactory factory;

	private static DocumentBuilder builder;

	private static Document document;

	private static Element cloudman;

	private static Element root;

	private static Element time;

	private static File file;

	private static Element batchPlugin;


	public static void main(String[] arg){


		//it.getElement(e, name, attr, attrName)

	}


	static {

		try {

			file = new File("vclusterStatus.xml");

		 if(!file.exists()){

			file.createNewFile();

			firstTime();
```

```
		}
		factory = DocumentBuilderFactory.newInstance();
		builder = factory.newDocumentBuilder();
		document = builder.parse(file);
		document.setXmlVersion("1.0");
		root = (Element) document.getElementsByTagName("root").item(0);
		cloudman = (Element) document.getElementsByTagName("cloudman").item(0);
		time = (Element) document.getElementsByTagName("LastModifiedTime").item(0);
		batchPlugin = (Element)document.getElementsByTagName("batchPlugin").item(0);
	} catch (ParserConfigurationException | IOException | SAXException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}
}
public static boolean restore(){

	/*restore clouds status */

	for(int i = 0; i<cloudman.getChildNodes().getLength();i++){
		ArrayList<String> conf = new ArrayList<String> ();
		Element e = (Element) cloudman.getChildNodes().item(i);
		for(int j = 0 ; j<e.getAttributes().getLength();j++){
			String str = e.getAttributes().item(j).getNodeName() +
"="+e.getAttributes().item(j).getNodeValue();
			conf.add(str);
		}
		Cloud cloud = new Cloud(conf);
```

```
		boolean isloaded = Boolean.parseBoolean(e.getAttribute("isloaded"));
		cloud.setIsLoaded(isloaded);
		CloudManager.getCloudList().put(cloud.getCloudName(), cloud);
		if(cloud.isLoaded())cloud.load();

	}

	/*restore batchPlugin status */
	String name = batchPlugin.getAttribute("name");
	if(!name.equals("")&&name!=null){
		PlugmanExecutor.load(new CmdComb("plugman load -b "+name));
	}


	return true;
}

public static boolean loadBatchPlugin(String name){
	batchPlugin.setAttribute("name", name);
	writeXML();
	return true;
}
public static boolean unloadBatchPlugin(){
	batchPlugin.removeAttribute("name");
	writeXML();
	return true;
```

```
}

public static boolean addCloudElement(List<String> conf){
	String name = conf.get(0).split("=")[1].trim();
	Element e = document.createElement(name);
	for(String item : conf){
		String str1 = item.split("=")[0].trim();
		String str2 = item.split("=")[1].trim();
		e.setAttribute(str1, str2);
	}
	e.setAttribute("isLoaded", "false");
	e.setAttribute("vms", "0");
	cloudman.appendChild(e);
	writeXML();
	return true;
}

public static boolean setCloudAttribute(String name,String att,String value){
	Element e = (Element)cloudman.getElementsByTagName(name).item(0);
	e.setAttribute(att, value);
	writeXML();
	return true;
}

public static boolean firstTime(){
	factory = DocumentBuilderFactory.newInstance();
```

```
	try {
		file.createNewFile();
		builder = factory.newDocumentBuilder();
	} catch (IOException | ParserConfigurationException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}

	document = builder.newDocument();
	document.setXmlVersion("1.0");
	java.text.DateFormat format1 = new java.text.SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
		Calendar cal = Calendar.getInstance();
		Date dt = cal.getTime();
		String t = format1.format(dt);

		root = document.createElement("vCluster");
		cloudman = document.createElement("cloudman");
		batchPlugin = document.createElement("batchPlugin");
		time = document.createElement("LastModifiedTime");
		time.setAttribute("value", t);
		document.appendChild(root);
		root.appendChild(time);
		root.appendChild(cloudman);
		root.appendChild(batchPlugin);

	return creaXML();
```

```
}



public static boolean creaXML(){
	try {
		TransformerFactory transFactory = TransformerFactory.newInstance();
		Transformer transFormer = transFactory.newTransformer();
		DOMSource domSource = new DOMSource(document);
		 FileOutputStream out = new FileOutputStream(file);
		 StreamResult xmlResult = new StreamResult(out);
		 transFormer.transform(domSource, xmlResult);



	} catch (IOException | TransformerException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
		return false;
	}

	return true;
}
public static boolean readXML(){

	return true;
}
```

```
public static boolean writeXML(){
	java.text.DateFormat format1 = new java.text.SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
	Calendar cal = Calendar.getInstance();
	Date dt = cal.getTime();
	String t = format1.format(dt);
	time.setAttribute("value", t);
	TransformerFactory transFactory = TransformerFactory.newInstance();
	Transformer transFormer = null;
	try {
		transFormer = transFactory.newTransformer();
	} catch (TransformerConfigurationException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
		return false;
	}

	DOMSource domSource = new DOMSource(document);

	 if(!file.exists()){
		 try {
			file.createNewFile();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
			return false;
		}
```

```
		}

		FileOutputStream out = null;

		try {

			out = new FileOutputStream(file);

		} catch (FileNotFoundException e) {

			// TODO Auto-generated catch block

			e.printStackTrace();

			return false;

		}

		StreamResult xmlResult = new StreamResult(out);

		try {

			transFormer.transform(domSource, xmlResult);

		} catch (TransformerException e) {

			// TODO Auto-generated catch block

			e.printStackTrace();

			return false;

		}


		return true;

	}


}


-----------
-----------
-----------


File: project-fermi/src/vcluster/util/Util.java
package vcluster.util;
```

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.StringTokenizer;
import java.util.TimeZone;
import java.util.TreeMap;

import vcluster.Vcluster;
/**
 * This class involves the global utilizable functions.
 */
public class Util {

	public static TreeMap<String, Boolean> listVSlist(ArrayList<String> arr1,ArrayList<String> arr2){
		TreeMap<String,Boolean> hsm = new TreeMap<String,Boolean>();
		ArrayList<String> temp = new ArrayList<String>();
		for(String str1 : arr1){
			for(String str2 : arr2){
				if(str1.equals(str2)){
					temp.add(str1);
				}

			}
		}

		arr1.removeAll(temp);
		arr2.removeAll(temp);
		for(String str : arr1){
			hsm.put(str, new Boolean(false));
		}
		for(String str : arr2){
			hsm.put(str, new Boolean(true));
		}
		return hsm;
	}

	public static boolean loadConfig(String configFile)
	{
		/* set config file */
		Vcluster.configFile = configFile;

		try {
			BufferedReader br = new BufferedReader(new FileReader(Vcluster.configFile));

			String aLine = "";
			while ((aLine = br.readLine()) != null) {
				lineProcess(aLine);
			}
			br.close();
```

```
            } catch (Exception e) {
               //PrintMsg.print(DMsgType.ERROR, "while loading, "+Config.configFile+" file!");
               return false;
            }

               return true;
        }

        private static void lineProcess(String aLine)
        {
               StringTokenizer st = new StringTokenizer(aLine, "=");

               if (!st.hasMoreTokens()) return;

               /* get a keyword */
               String aKey = st.nextToken().trim();

               /* get a value */
               if (!st.hasMoreTokens()) return;

               String aValue = st.nextToken().trim();

               if (aKey.equalsIgnoreCase("condor"))
                      Vcluster.CONDOR_IPADDR = aValue;
        }


        public static byte[] intToBytes(int i)
        {
               byte[] result = new byte[5];

               result[0] = (byte) (i >> 24);
               result[1] = (byte) (i >> 16);
               result[2] = (byte) (i >> 8);
               result[3] = (byte) (i /*>> 0*/);
               result[4] = '\0';

               return result;
        }

        public static String getTimestampFromLocalTime(Date date) {
                SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS'Z'");
                format.setTimeZone(TimeZone.getTimeZone("GMT"));
                return format.format(date);
        }

        public static String formatTimestamp() {
               Calendar cal = Calendar.getInstance();
               DateFormat df = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss'Z'");
               cal.setTimeInMillis(cal.getTimeInMillis() - cal.get(Calendar.ZONE_OFFSET) -
                              cal.get(Calendar.DST_OFFSET));
               return df.format(cal.getTime());
        }
```

```
        public static void print(StringBuffer arg){
               print(arg.toString());
        }
        public static void print(String arg){
               switch(Vcluster.currUI) {
               case CMDLINE:
                      System.out.println(arg);
                      break;
               default:
                      break;
               }
        }
}

-----------
-----------
-----------


File: project-fermi/src/vcluster/Vcluster.java
package vcluster;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.Writer;
import java.util.Date;


/**
 *This class initiate the parameters of vcluster.
 */
public class Vcluster {

        public static String CONDOR_IPADDR;
        public static int PORTNUM;
        public static String SERVER_ADDR;
        public static int SERVER_PORT1;
        public static int SERVER_PORT2;
        public static String configFile = "vcluster.conf";
        public static enum uiType {CMDLINE, REMOTECLIENT, WEBUI, LOADBALANCER};
        public static uiType currUI;
        public static String xmlFile;
        public static boolean terminate;
        public static final String logFile="log"+File.separator+"Balancer_Log";
     public static final String VmLaunchLogFile="log"+File.separator+"VmLaunch_Log";
     public static final String sysLogFile="log"+File.separator+"Sys_Log";

        /**
         * When vcluser start this function will be involked, the parameters will be read from a configuration file.
         */
```

```
	public static void init(){
		terminate=false;
		BufferedReader br ;
		try{
			br = new BufferedReader(new FileReader(configFile));
			String aLine = "";
			while ((aLine = br.readLine()) != null) {
				if(aLine.contains("#")||!aLine.contains("="))continue;
				String[] pair = aLine.split("=");
				if (pair[0].trim().equalsIgnoreCase("CONDOR_IPADDR")) {
					CONDOR_IPADDR = pair[1].trim();
				}else if(pair[0].trim().equalsIgnoreCase("CONDOR_PORTNUM")){
					PORTNUM = Integer.parseInt(pair[1].trim());
				}else if(pair[0].trim().equalsIgnoreCase("SERVER_IPADDR")){
					SERVER_ADDR = pair[1].trim();
				}else if(pair[0].trim().equalsIgnoreCase("SERVER_PORT1")){
					SERVER_PORT1 = Integer.parseInt(pair[1].trim());
				}else if(pair[0].trim().equalsIgnoreCase("SERVER_PORT2")){
					SERVER_PORT2 = Integer.parseInt(pair[1].trim());
				}
			}

			br.close();

		}catch(Exception e){
			System.out.println("Configuration file doesn't exist ,please check it!");
		}
	}

	public static void writeLogFile(String line){
		try{
		Writer output;
		output = new BufferedWriter(new FileWriter(Vcluster.logFile,true));
		Date d=new Date(System.currentTimeMillis());
		String str="["+d.toString()+"] "+ line+System.lineSeparator();
		output.append(str);
		output.close();
	}catch(Exception e){
			e.getStackTrace();
	}
	}

	public static void writeLaunchLogFile(String line){
		try{
		Writer output;
		output = new BufferedWriter(new FileWriter(Vcluster.VmLaunchLogFile,true));
		Date d=new Date(System.currentTimeMillis());
		String str="["+d.toString()+"] "+ line+System.lineSeparator();
		output.append(str);
		output.close();
	}catch(Exception e){
			e.getStackTrace();
	}
```

```
	}

	public static void writeSysLogFile(String line){
		try{
		Writer output;
		output = new BufferedWriter(new FileWriter(Vcluster.sysLogFile,true));
		Date d=new Date(System.currentTimeMillis());
		String str="["+d.toString()+"] "+ line+System.lineSeparator();
		output.append(str);
		output.close();
	}catch(Exception e){
			e.getStackTrace();
	}
	}
}
-----------
-----------
-----------

File: proxy-master/README.md
virtual cluster proxy server

this server should be running on condor master and cloud front end in order to communicate with vcluster
-----------
-----------
-----------


File: proxy-master/include/vcluster_proxy.h
/*
 * kissman_svr.h
 *
 *  Created on: Mar 16, 2011
 *      Author: rsyoung
 */

#ifndef KISSMAN_SVR_H_
#define KISSMAN_SVR_H_

#define OFFSET_SERVER_TYPE		8
#define SVR_SAM_STATION		0x1
#define SVR_PBS_BATCH_SYSTEM		0x2

/* for opennebula cloud system */
#define SVR_ONE_CLOUD_SYSTEM		0x5

/* for cloudgate test purpose */
#define SVR_CONDOR_BATCH_SYSTEM		0x10

#define BUF_SIZE		1024
#define CMD_LINE_LEN		1024
```

```
#endif /* KISSMAN_SVR_H_ */

-----------
-----------
-----------


File: proxy-master/src/main.c
/*
 * main.c
 *
 *  Created on: Mar 16, 2011
 *      Author: rsyoung
 */

#include <sys/types.h>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/time.h>
#include <sys/ioctl.h>

#include "service.h"


static unsigned int get_server_num(const unsigned int svr_code);
static unsigned int get_service_type(const unsigned int svr_code);
static void make_new_connection(const int server_sockfd, fd_set* masterfds, int* fdmax);
static void *get_in_addr(struct sockaddr *sa);
static void set_server_addr(struct sockaddr_in* svr_addr);
static int get_request(const int fd, struct  service_struct* svc_struct);
static void process_request(const int fd);
static int get_service_code(const int fd, unsigned int* svc_code);
static int get_command_line(const int fd, struct service_struct* svc_struct);

static int check_server_num(int svr_num);
static int check_service_type(struct service_struct *svc_struct);

static void dump_svc(struct service_struct* svc_struct);

static int train_command(char *orginal, char *cmd);

extern int execute_service(const int fd, const struct service_struct svc_struct);

static int server_num;

int main(int argc, char* argv[])
{

        int server_sockfd;
        socklen_t server_len;;

        struct sockaddr_in server_address;
```

```
        int ret;
        fd_set masterfds, readfds;

        /* arguments should be 2 for correct execution */
        /*
         * temporally blocked
        if ( argc != 2 ) {
         printf( "usage: %s <server number>\n", argv[0] );
         exit(-1);
    }
        server_num = atoi(argv[1]);
        */



        /* create a socket */
        server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
        printf("[MSG] : server_sockfd = %d\n", server_sockfd);


        /* setsockopt: Handy debugging trick that lets
        * us rerun the server immediately after we kill it;
        * otherwise we have to wait about 20 secs.
        * Eliminates "ERROR on binding: Address already in use" error.
        */
        int optval = 1;
        setsockopt(server_sockfd, SOL_SOCKET, SO_REUSEADDR,
                (const void *)&optval , sizeof(int));

        set_server_addr(&server_address);
        server_len = sizeof(server_address);



        /* bind the socket to an address */
        ret = bind(server_sockfd, (struct sockaddr *)&server_address, server_len);
        if (ret != 0) {
                printf("[Error] : failure to bind at %d, %s\n", __LINE__, __FILE__);
                exit(1);
        }

        /* listen from connection */
        ret = listen(server_sockfd, MAX_QUEUE_LEN);

        if (ret != 0) {
                perror("Error: fail to listen\n");
                exit(1);
        }

        FD_ZERO(&readfds);
        FD_ZERO(&masterfds);
        FD_SET(server_sockfd, &masterfds);
```

```
	int fdmax = server_sockfd;

	while(1)
	{
		int fd = -1;

		readfds = masterfds;

		/* checks to see if any sockets are ready for reading */
		ret = select(fdmax+1, &readfds, (fd_set *)0,
								(fd_set *)0, (struct timeval *)0);
		if(ret < 1) {
			printf("[Error] : failure to select fd at %d, %s\n", __LINE__, __FILE__);
			exit(-1);
		}

		/* check if there is a fd to read */

		for(fd = 0; fd <= fdmax; fd++) {

			/* if fd is not set in readfds, then skip the rest code */
			if(!FD_ISSET(fd, &readfds)) continue;

			// printf("fd = %d\n", fd);

			if (fd == server_sockfd) {
				make_new_connection(server_sockfd, &masterfds, &fdmax);
			} else {
				process_request(fd);

				FD_CLR(fd, &masterfds);
				close(fd);
				printf("[MSG] : fd = %d closed\n", fd);
			}
		}
	}

	close(server_sockfd);

	return 0;
}

/**
 * set server address
 *
 * @param svr_addr sockaddr_in struct to be filled with values
 */
static void set_server_addr(struct sockaddr_in* svr_addr)
{
	svr_addr->sin_family = AF_INET;
	svr_addr->sin_addr.s_addr = htonl(INADDR_ANY);
	svr_addr->sin_port = htons(PORT_NUM);
```

```
}

/**
 * get sockaddr for IPv4 or IPv6
 *
 * @param sa socket address
 *  */
static void *get_in_addr(struct sockaddr *sa)
{
	if (sa->sa_family == AF_INET) {
		return &(((struct sockaddr_in*)sa)->sin_addr);
	}

	return &(((struct sockaddr_in6*)sa)->sin6_addr);
}

/**
 * make a new connection from a client
 *
 * @param server_sockfd server socket fd
 * @param masterfds master file descriptor set
 * @param fdmax max file descriptor to return
 */
static void make_new_connection(const int server_sockfd, fd_set* masterfds, int* fdmax)
{

	int client_sockfd;
	socklen_t client_len;
	struct sockaddr_storage client_address;

	client_len = sizeof(client_address);
	client_sockfd = accept(server_sockfd,
			(struct sockaddr *)&client_address, &client_len);

	if(client_sockfd < 0) {
		printf("[Error] : cannot accept a new connection at %d, %s\n", __LINE__, __FILE__);
		return;
	}

	FD_SET(client_sockfd, masterfds);
	char clientIP[INET6_ADDRSTRLEN];

	printf("[MSG] : new connection from %s on socket %d\n",
				inet_ntop(client_address.ss_family,
					get_in_addr((struct sockaddr*)&client_address),
					clientIP, INET6_ADDRSTRLEN),
					client_sockfd);

	if(client_sockfd > *fdmax) *fdmax = client_sockfd;

}

/**
```

```
 * process a request from a client. It gets a service code, identifies a service
 * and execute service according to the service code.
 *
 * @param fd file descriptor to communicate.
 */
static void process_request(const int fd)
{
	unsigned int service_code = 0;
	struct service_struct svc_struct;

	/* try to get service code. If fail, then just return */
	if (-1 == get_request(fd, &svc_struct)) {
		printf("[Error] : failed to get service code = %d\n", service_code);
		return;
	}

	/* execute service */
	execute_service(fd, svc_struct);
}



/**
 * receive a request from a client. The request may consist of two command
 * 1) service code + (-) 2) command.
 *
 * Particularly onevm, condor* releated commands are being sent from a client with
 * service code + its shell command
 *
 * @param fd file descriptor to receive service code and command
 * @param svc_struct contains a server, service type and command
 *
 * @return 1 if successfully received; otherwise -1
 */
static int get_request(const int fd, struct  service_struct* svc_struct)
{

	unsigned int svc_code;

	if (-1 == get_service_code(fd, &svc_code)) return -1;

	/* get server number  and check if it is correct */
	svc_struct->server_num = get_server_num(svc_code);

	/*
	 * temporally blocked
	if (-1 == check_server_num(svc_struct->server_num)) return -1;
	*/


	/* get service type and check if it is correct */
	svc_struct->service_type = get_service_type(svc_code);
	if (-1 == check_service_type(svc_struct)) return -1;
```

```
	/* get shell command if there is */
	if (-1 == get_command_line(fd, svc_struct)) return -1;

	dump_svc(svc_struct);

	return 1;
}

/**
 * get service code from fd
 *
 * @param fd file descriptor being used to communicated with a client
 * @param svc_code service code to receive from the client
 * @return 1 if success; otherwise -1
 */
static int get_service_code(const int fd, unsigned int* svc_code)
{
	int ret;
	char buf[4];

	if ((ret = recv(fd, buf, 4, 0)) <= 0) {
		if(ret == 0)
			/* connection closed */
			printf("[MSG] : socket %d hung up\n", fd);
		else
			printf("[Error] : recv() error lol at %d, %s.\n", __LINE__, __FILE__);

		return -1;
	}

	*svc_code = atoi(buf);
	if(*svc_code == 0) return -1;

	return 1;

}


/**
 * check the server number if requested server and current running one are same.
 * It is to prevent from executing commands which are not supported by
 * running server
 *
 * @param svr_num server number which is requested
 * @return 1 if correct; otherwise -1
 */
static int check_server_num(int svr_num)
{
	if (svr_num != server_num) {
		printf("[Error] : Server number is not matched: cur_svr[%u] != req_svr[%u]\n",
		                                                                        server_num, svr_num);
		return -1;
```

```
        }

        return 1;
}

/**
 * receive a shell command line.
 *
 * @param fd file descriptor to receive command line from a client
 * @svc_struct data structure to contain the command line
 * @return 1 if success; otherwise -1
 */
static int get_command_line(const int fd, struct service_struct* svc_struct)
{
        char cmdbuf[CMD_LINE_LEN];
        int ret;
        int svr_num = svc_struct->server_num;

        /* copy command */
        switch (svr_num) {
        case SVR_ONE_CLOUD_SYSTEM:
        case SVR_CONDOR_BATCH_SYSTEM:
                if ((ret = recv(fd, cmdbuf, CMD_LINE_LEN, 0)) <= 0) {
                        if(ret == 0)
                                /* connection closed */
                                printf("[MSG] : socket %d hung up\n", fd);
                        else
                                printf("[Error] : recv() error lol at %d, %s.\n", __LINE__, __FILE__);

                        return -1;
                }

                bzero(svc_struct->cmdline, CMD_LINE_LEN);
                train_command(cmdbuf,svc_struct->cmdline);
                  //strncpy(svc_struct->cmdline, cmdbuf, CMD_LINE_LEN);
        }
         //printf("[Debug] : %s", cmdbuf);
        return 1;
}


/**
 * check service type
 *
 * @param svc_struct service structure containing service type
 * @return 1 if valid; otherwise -1
 */
static int check_service_type(struct service_struct *svc_struct)
{
        int svr_num = svc_struct->server_num;
        int svc_type = svc_struct->service_type;

        switch (svr_num) {
```

```
        case SVR_SAM_STATION:
                if (svc_type == 0 || svc_type >= SVC_SAM_MAX) {
                        printf("[Error] : service number is wrong: server[%u], service [%u]"
                                        " at %d, %s\n", svr_num, svc_type, __LINE__, __FILE__);
                        return -1;
                }
                break;

        default:
                break;
        }

        return 1;
}


/**
 * dump service structure
 *
 * @param svc_struct service structure to dump
 */
static void dump_svc(struct service_struct* svc_struct)
{
        printf("[MSG] : contents of service_struct\n");
        printf("        server num  : [%d]\n", svc_struct->server_num);
        printf("        service type: [%d]\n", svc_struct->service_type);
        printf("        command     : %s\n", svc_struct->cmdline);
}

/**
 * get a server number from a given code. A server number was 8 bit
 * left shifted. So, in order to get the right server number, it has to be
 * right shifted properly. For example, server number is 1, then it is encoded
 * as 100000000.
 *
 * @param svr_code a code given from a client which contains a service number
 * and service type by bit operation.
 *
 * @return sever number
 */

static unsigned int get_server_num(const unsigned int svr_code)
{
        return svr_code >> OFFSET_SERVER_TYPE;
}


/**
 * get a service type from a given code. A service type is added to a service
 * number which has been 8 bit left shifted. So, in order get to the service
 * type, the code has to be masked with only low 8 bits having "1".
 *
 * @param svr_code a code given from a client which contains a service number
```

```
 * and service type by bit operation.
 *
 * @return a service type
 */
static unsigned int get_service_type(const unsigned int svr_code)
{
       return svr_code & ((1 << OFFSET_SERVER_TYPE)-1);
}

static int train_command(char *str, char *cmd)
{
       int index=0;
       int line=0;
       int i=0;
       for( ;i<CMD_LINE_LEN;i++){
            if(str[i]=='\n'||str[i]=='\r')
               {
                   line=1;
                   index=i;break;
               }
       }
       if(line!=1){
         printf("[ERROR] : Incomplete command!");
         return -1;
         }
       strncpy(cmd,str,index);
       return 1;
}

-----------
-----------
-----------


File: proxy-master/src/service.c
/*
 * service.c
 *
 *  Created on: Jul 18, 2011
 *      Author: rsyoung
 */

#include "service.h"


const char sam_cmd[][CMD_LINE_LEN] = {
              "dummy command", /* never called */
              "/bin/ls /home/rsyoung/wallpaper/",
              "/bin/ls /home/rsyoung/wallpaper/",
              "/bin/ls /home/rsyoung/wallpaper/",
              "/bin/ls /home/rsyoung/wallpaper/",
              "/bin/cat /proc/cpuinfo",
              "/bin/ls /etc/",
```

```
};


/*
const char condor_cmd[][CMD_LINE_LEN] = {
              "dummy command",
              "condor_status",
              "condor_q",
};


const char opennebula_cmd[][CMD_LINE_LEN] = {
              "dummy command",
              "onevm list",
};
*/


int get_svc_cmdline(const struct service_struct svc_struct, char* cmdline);

/**
 * execute a service based on server number and service type. It first gets
 * a command for service, create a pipe for the command, redirect the output
 * of execution to sending file descriptor fd.
 *
 * @param fd file descriptor to send data
 * @param svc_struct service struct which cotains server number and service type
 *
 * @return 1 if successfully execute a service; otherwise -1
 */
int execute_service(const int fd, const struct service_struct svc_struct)
{
       char cmdline[CMD_LINE_LEN];
       bzero(cmdline, CMD_LINE_LEN);

       FILE *fp;
       char buffer[BUF_SIZE];
       int nbytes = -1;

       /* get service command */
       if (get_svc_cmdline(svc_struct, cmdline) < 0)
              return -1;
        //printf("%s",cmdline);
        //printf("one command!");

       fp = popen(cmdline, "r");
       if (fp == NULL) {
              printf("[Error] : failure to run command at %d, %s\n", __LINE__, __FILE__);
              return -1;
       }

       bzero(buffer, BUF_SIZE);
        int i=0;
```

```
        while (fgets(buffer, BUF_SIZE, fp) != NULL) {
                buffer[BUF_SIZE-1]='\n';
                  nbytes = send(fd, buffer, BUF_SIZE, 0);
                  //i++;
                  //printf("This is a Line: ");
                printf("%s", buffer);

                if (nbytes < 0)
                        printf("[Error] : bytes are less than 0 at %d, %s\n", __LINE__, __FILE__);
                bzero(buffer, BUF_SIZE);
        }
        pclose(fp);

        return 1;
}


int execute_service_2(const int fd, const struct service_struct svc_struct)
{
        char cmdline[CMD_LINE_LEN];
        bzero(cmdline, CMD_LINE_LEN);

        FILE *fp;
        char buffer[BUF_SIZE];
        int nbytes = -1;

        /* get service command */
        if (get_svc_cmdline(svc_struct, cmdline) < 0)
                return -1;

        fp = popen(cmdline, "r");
        if (fp == NULL) {
                printf("[Error] : failure to run command at %d, %s\n", __LINE__, __FILE__);
                return -1;
        }

        bzero(buffer, BUF_SIZE);

    FILE *fpout;
    size_t count = 0;

    fpout = fopen("output.txt", "w");
    if(fpout == NULL) {
        perror("failed to open sample.txt");
        return -1;
    }

        while (fgets(buffer, BUF_SIZE, fp) != NULL) {
            printf("This is a Line: ");
             printf("%s", buffer);
            count += fwrite(buffer, 1, strlen(buffer), fpout);
                bzero(buffer, BUF_SIZE);
```

```
        }
        pclose(fp);
        fclose(fpout);

    fpout = fopen("output.txt", "r");
    if(fpout == NULL) {
        perror("failed to open sample.txt");
        return -1;
    }


        bzero(buffer, BUF_SIZE);
        while (fgets(buffer, BUF_SIZE, fpout) != NULL) {
                nbytes = send(fd, buffer, BUF_SIZE, 0);

                printf("%s", buffer);
                if (nbytes < 0)
                        printf("[Error] : bytes are less than 0 at %d, %s\n", __LINE__, __FILE__);
                bzero(buffer, BUF_SIZE);
        }

        fclose(fpout);


        return 1;
}

/**
 * get a command line for a server number and service type
 *
 * @param svc_struct service struct holding server number and service type
 * @param cmdline command line to be returned
 *
 * @return 1 if successfully get service command; otherwise -1
 */
int get_svc_cmdline(const struct service_struct svc_struct, char* cmdline)
{
        const unsigned int svr_num = svc_struct.server_num;
        const unsigned int svc_type = svc_struct.service_type;

        switch (svr_num) {
        case SVR_SAM_STATION:
                if (strcpy(cmdline, sam_cmd[svc_type]) != NULL)
                        return 1;

        case SVR_CONDOR_BATCH_SYSTEM:
        case SVR_ONE_CLOUD_SYSTEM:
                if (strcpy(cmdline, svc_struct.cmdline) != NULL)
                        return 1;

        default:
                break;
        }
```

```
        printf("[Error] : error while copying service command at %d, %s\n",
                        __LINE__, __FILE__);

        return -1;

}

-----------
-----------
-----------


File: proxy-master/src/service.h
/*
 * service.h
 *
 *  Created on: Jul 18, 2011
 *      Author: rsyoung
 */

#ifndef SERVICE_H_
#define SERVICE_H_

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>

#include <./include/vcluster_proxy.h>


#define MAX_QUEUE_LEN              10
#define PORT_NUM                        9734


enum svc_sam {
        SVC_SAM_LOGFILE = 1,
        SVC_SAM__LOGFILE,
        SVC_SAM_CPUINFO,
        SVC_SAM_MEMINFO,
        SVC_SAM_PROCESSINFO,
        SVC_SAM_CACHE_STATUS,
        SVC_SAM_MAX
};


struct service_struct {
        unsigned int server_num;
        unsigned int service_type;
        char cmdline[CMD_LINE_LEN];
};
```

```
#endif /* SERVICE_H_ */

-----------
-----------
-----------
```