UNIVERSITÀ DEGLI STUDI DELLA CALABRIA
FACOLTÀ DI INGEGNERIA
CORSO DI LAUREA SPECIALISTICA IN INGEGNERIA INFORMATICA

Esame di SISTEMI DISTRIBUITI

APPELLO DEL 7 GENNAIO 2004 – SOLUZIONE

Interfaccia dei job

```
import java.io.*;

public interface Job extends Serializable
{     Object run();
      String getID();
}
```

Interfaccia dei controller

```
import java.rmi.*;

public interface Controller extends Remote
{     void aggiungiNodoVicino(String IPnodo) throws RemoteException;
      int getPotenzaDisponibile() throws RemoteException;
      void sottomettiJob(String IPnodo, Job job, int req,
            long timestamp,Processo richiedente) throws RemoteException;
      void consegnaRisultato(String jobID,Object res);
      void segnalaEsecuzioneImpossibile(String jobID);
}
```

Interfaccia dei processi

```
public interface Processo
{     void consegnaRisultato(String jobID, Object res);
      void segnalaEsecuzioneImpossibile(String jobID);
}
```

Server di calcolo

```
public class ComputeEngine
{     private int potenzaDisponibile;

      public ComputeEngine()
      {     potenzaDisponibile=100;
      }

      public int getPotenzaDisponibile()
      {     return potenzaDisponibile;
      }

      public void riservaPotenza(int req)
      {     potenzaDisponibile-=req;
      }

      public void rilasciaPotenza(int req)
      {     potenzaDisponibile+=req;
      }

      public Object esegui(Job j)
      {     return j.run();
      }
}
```

Processo server (*da lanciare su N nodi*)

```
import java.rmi.*;

public class ProcessoServer
{     public static void main(String args[])
      {     try
            {     ControllerImpl cei = new ControllerImpl();
                  Naming.rebind("Controller", cei);
                  System.out.println("ComputeEngine e Controller partiti.");
            }
            catch (Exception e)
            {     System.out.println(e);
            }
   }
}
```

Implementazione dei controller

```
import java.rmi.*;
import java.util.*;

public class ControllerImpl implements Controller
{     private ComputeEngine ce;
      private Vector nodiVicini;
      private HashMap jobProcessi; //mantiene le associazioni
                                   //tra job richiesto e processo locale richiedente

      public ControllerImpl()
      {     super();
            ce=new ComputeEngine();
            nodiVicini=new Vector();
            jobProcessi=new HashMap();
      }

      public void aggiungiNodoVicino(String IPnodo)
      {     nodiVicini.add(IPnodo);
      }

      public int getPotenzaDisponibile()
      {     return ce.getPotenzaDisponibile();
      }

      public void consegnaRisultato(String jobID,Object res)
      {     Processo p=(Processo)jobProcessi.get(jobID);
            p.consegnaRisultato(jobID,res);
      }

      public void segnalaEsecuzioneImpossibile(String jobID)
      {     Processo p=(Processo)jobProcessi.get(jobID);
            p.segnalaEsecuzioneImpossibile(jobID);
      }

      public void sottomettiJob(String IPnodo, Job job, int req,
            long timestamp, Processo richiedente)
      //'richiedente' e' nullo se la richiesta proviene da un altro controller
      {     //verifica timeout(10 minuti)
            if(req > 100 || System.currentTimeMillis()-timestamp > 600000)
            {     if (richiedente!=null) //consegno il timeout al processo locale
                        richiedente.segnalaEsecuzioneImpossibile(job.getID());
                  else //segnalo il timeout al controller sul nodo 'IPnodo'
                        try
                        {     Controller c = (Controller)Naming.lookup("rmi://"+
                                    IPnodo+":1099/Controller");
                              c.segnalaEsecuzioneImpossibile(job.getID());
                        }
                        catch (Exception e)
                        {     System.out.println(e);
                        }
                  return;
            }

            //tengo traccia dell'associazione tra job e processo richiedente
            if(richiedente!=null)
                  jobProcessi.put(job.getID(),richiedente);

            //scelgo il nodo piu' scarico
            String vicinoScarico="";
            int caricoMinimoVicini=100;
```

```
            for(int i=0;i<nodiVicini.size();i++)
            {     String nodo=(String)nodiVicini.get(i);
                  try
                  {     Controller c = (Controller)Naming.lookup("rmi://"+
                              IPnodo+":1099/Controller");
                        int carico=c.getPotenzaDisponibile();
                        if(carico < caricoMinimoVicini)
                        {     caricoMinimoVicini=carico;
                              vicinoScarico=nodo;
                        }
                  }
                  catch (Exception e)
                  {     System.out.println(e);
                  }
            }

            int potenzaLocale=getPotenzaDisponibile();
            if(caricoMinimoVicini > potenzaLocale && potenzaLocale > req)
                  //esecuzione locale
            {     ce.riservaPotenza(req);
                  Object res=ce.esegui(job);
                  ce.rilasciaPotenza(req);
                  if(richiedente!=null) //consegno il risultato al processo locale
                        richiedente.consegnaRisultato(job.getID(),res);
                  else //consegno il risultato al controller sul nodo 'IPnodo'
                        try
                        {     Controller c = (Controller)Naming.lookup("rmi://"+
                                    IPnodo+":1099/Controller");
                              c.consegnaRisultato(job.getID(),res);
                        }
                        catch (Exception e)
                        {     System.out.println(e);
                        }
            }
            else //passo la richiesta a 'vicinoScarico'
            {     try
                  {     Controller c = (Controller)Naming.lookup("rmi://"+
                              vicinoScarico+":1099/Controller");
                        c.sottomettiJob(IPnodo,job,req,timestamp,null);
                  }
                  catch (Exception e)
                  {     System.out.println(e);
                  }
            }
      }
}
```