GreenCloud Simulator

Generated by Doxygen 1.8.11

Contents

1 Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

2 Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

3 File Index

3.1 File List

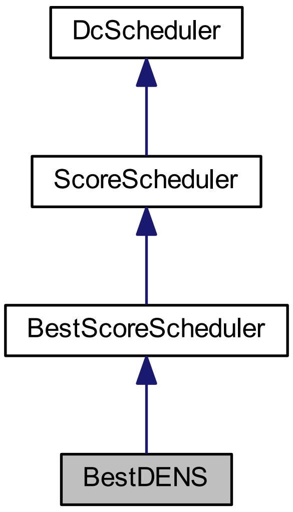
Here is a list of all files with brief descriptions:

4 Class Documentation

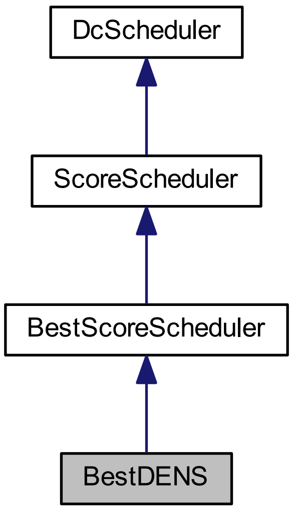
4.1 BestDENS Class Reference

```
#include <bestdens.h>
```

Inheritance diagram for BestDENS:



Collaboration diagram for BestDENS:



Public Member Functions

- BestDENS ()
- virtual ∼BestDENS ()
- virtual TskComAgent ∗ scheduleTask (CloudTask ∗task, std::vector< ResourceProvider ∗ > providers)

Private Member Functions

- virtual double calculateScore (ResourceProvider ∗rp)
- double densLoadFactor (double load, double epsilon)
- double linkLoadFactor (double load)

Private Attributes

- double epsilon

4.1.1 Detailed Description

MultiDENS scheduler. To meaningfully use it, enableDVFS on the resource providers used by this scheduler. TO↩
DO: add networking part of DENS.

Definition at line 21 of file bestdens.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 BestDENS::BestDENS ( )

Definition at line 11 of file bestdens.cc.

```
                      : epsilon(0.1){

}
```

4.1.2.2 BestDENS::∼BestDENS ( ) [virtual]

Definition at line 15 of file bestdens.cc.

```
                       {

}
```

4.1.3 Member Function Documentation

4.1.3.1 double BestDENS::calculateScore (ResourceProvider ∗ *rp*) [private], [virtual]

Implements BestScoreScheduler.

Definition at line 23 of file bestdens.cc.

```
                                                    {
                double result = 0;
                double load;
                for(int i = FirstResType; i <= LastResType ; i++){
                                load = rp->getResTypeUtil(static_cast<res_type>(i));
                                result+= densLoadFactor(load,
     epsilon);
                }
                result=result/(LastResType+1); // normalize according to the number of
      dimensions
//              std::cerr << "Res_prov " << rp->id_ << "\n";
//              DcHost* host = rp->getRootHost();
//              DcRack* rack = host->rack_;
//              std::cerr << "Rack " << rack->rack_id_ << "\n";
//              double ll = rack->link_load;
//              std::cerr << "ll " << ll << "\n";
                result *= pow(linkLoadFactor(rp->getRootHost()->
     rack_->link_load),2);
//              result += linkLoadFactor(ll);


//              double load = rp->getResTypeUtil(Computing);
//              result = densLoadFactor(load,0.1);
//              std::cerr << "Host " << rp->id_ /*<< "CPU load: " << load*/ << "\tDENS score: " << result
      << "\n";

                return result;
}
```

4.1.3.2 double BestDENS::densLoadFactor (double *load*, double *epsilon*) [private]

Definition at line 48 of file bestdens.cc.

```
                                                          {
                return 1/(1+exp(-10*(load-0.5))) - 1/(1+exp((-10/epsilon)*(load-(1-(
     epsilon/2)))));
}
```

4.1.3.3 double BestDENS::linkLoadFactor (double *load*) [private]

Definition at line 52 of file bestdens.cc.

```
                                             {
                return exp(-(load*load));
}
```

4.1.3.4 TskComAgent ∗ BestDENS::scheduleTask (CloudTask ∗ *task*, std::vector< ResourceProvider ∗ > *providers*) [virtual]

Reimplemented from BestScoreScheduler.

Definition at line 19 of file bestdens.cc.

```
                                                                                   {
                return BestScoreScheduler::scheduleTask(task,providers);
}
```

4.1.4 Member Data Documentation

4.1.4.1 double BestDENS::epsilon [private]

Definition at line 27 of file bestdens.h.

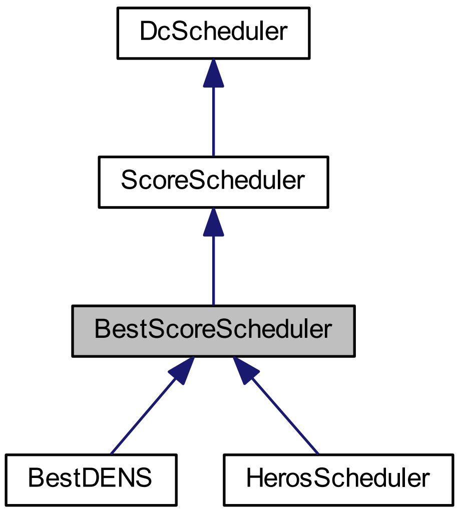
The documentation for this class was generated from the following files:

- bestdens.h
- bestdens.cc

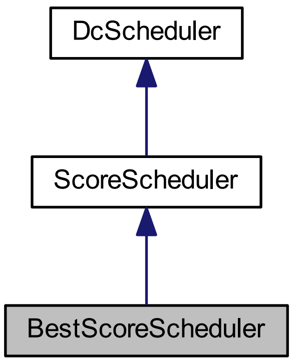
4.2 BestScoreScheduler Class Reference

`#include <bestscorescheduler.h>`

Inheritance diagram for BestScoreScheduler:



Collaboration diagram for BestScoreScheduler:



Public Member Functions

- BestScoreScheduler ()
- virtual ∼BestScoreScheduler ()
- virtual TskComAgent ∗ scheduleTask (CloudTask ∗task, std::vector< ResourceProvider ∗ > providers)

Private Member Functions

- virtual double calculateScore (ResourceProvider ∗rp)=0

4.2.1 Detailed Description

Definition at line 14 of file bestscorescheduler.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 BestScoreScheduler::BestScoreScheduler ( )

Definition at line 10 of file bestscorescheduler.cc.

```
                                       {


}
```

4.2.2.2 BestScoreScheduler::∼BestScoreScheduler ( ) [virtual]

Definition at line 15 of file bestscorescheduler.cc.

```
                                        {

}
```

4.2.3 Member Function Documentation

4.2.3.1 virtual double BestScoreScheduler::calculateScore (ResourceProvider ∗ *rp*) [private], [pure virtual]

Implements ScoreScheduler.

Implemented in HerosScheduler, and BestDENS.

4.2.3.2 TskComAgent ∗ BestScoreScheduler::scheduleTask (CloudTask ∗ *task,* std::vector< ResourceProvider ∗ > *providers*) [virtual]

Implements DcScheduler.

Reimplemented in HerosScheduler, and BestDENS.

Definition at line 19 of file bestscorescheduler.cc.

```
                                                                                                    {
                vector<ProviderScore> scored_providers_;
                vector <ResourceProvider*>::iterator iter;
                for (iter = providers.begin(); iter!=providers.end(); iter++)
                {
                                if ((*iter)->testSchedulingPossibility(task)){
                                                scored_providers_.push_back(
    ProviderScore((*iter),calculateScore((*iter))));
                                } else {
//                                              std::cerr << "Provider full!\n";
                                }
                }
                if(scored_providers_.empty()){
                                return NULL;
                } else {
                                ProviderScore best = (*max_element(scored_providers_.begin(),
    scored_providers_.end()));
                                scored_providers_.clear();
                                return best.provider_->getTskComAgent();
                }
}
```

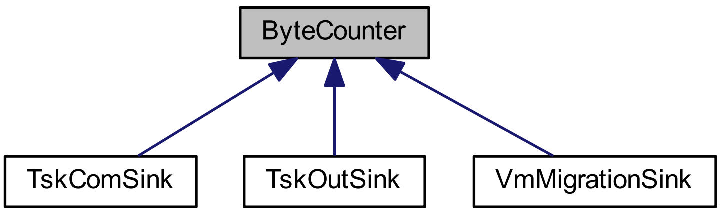
The documentation for this class was generated from the following files:

- bestscorescheduler.h
- bestscorescheduler.cc

4.3 ByteCounter Class Reference

```
#include <bytecounter.h>
```

Inheritance diagram for ByteCounter:



Public Member Functions

- ByteCounter ()
- virtual ∼ByteCounter ()
- int resetBytesSince ()
- double getLastBytesSinceTime ()

Protected Attributes

- int bytes_since_
- double last_bytes_since_

4.3.1 Detailed Description

Definition at line 13 of file bytecounter.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 ByteCounter::ByteCounter ( )

Definition at line 10 of file bytecounter.cc.

```
                            {
                    resetBytesSince();

}
```

4.3.2.2 ByteCounter::∼ByteCounter ( ) [virtual]

Definition at line 15 of file bytecounter.cc.

```
                            {

}
```

4.3.3 Member Function Documentation

4.3.3.1 double ByteCounter::getLastBytesSinceTime ( )

Returns the time of the last check.

Definition at line 26 of file bytecounter.cc.

```
                                          {
                  return last_bytes_since_;
}
```

4.3.3.2 int ByteCounter::resetBytesSince ( )

Returns the value of bytes since last check.

Definition at line 19 of file bytecounter.cc.

```
                             {
                  int result =  this->bytes_since_;
                  this->bytes_since_ = 0;
                  this->last_bytes_since_=Scheduler::instance().clock();
                  return result;
}
```

4.3.4 Member Data Documentation

4.3.4.1 int ByteCounter::bytes_since_ [protected]

Definition at line 26 of file bytecounter.h.

4.3.4.2 double ByteCounter::last_bytes_since_ [protected]

Definition at line 27 of file bytecounter.h.

The documentation for this class was generated from the following files:

- bytecounter.h
- bytecounter.cc

4.4 Capacity Class Reference

`#include <resource.h>`

Public Member Functions

- Capacity ()
- Capacity (int i)
- Capacity (double d)
- Capacity & operator= (const double &val)
- Capacity & operator= (const Capacity &c)
- operator double ()
- Capacity & operator+= (const Capacity &rhs)
- Capacity & operator-= (const Capacity &rhs)
- Capacity & operator+= (const double &rhs)
- Capacity & operator-= (const double &rhs)
- double getValueRecursive ()

Public Attributes

- double value
- std::vector< Capacity ∗ > virtual_capacities

4.4.1 Detailed Description

Definition at line 28 of file resource.h.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 Capacity::Capacity ( )

Definition at line 11 of file resource.cc.

```
                                    : value(0.0) {

                  }
```

4.4.2.2 Capacity::Capacity (int *i*)

Definition at line 15 of file resource.cc.

```
                                         : value(i){

                  }
```

4.4.2.3 Capacity::Capacity (double *d*)

Definition at line 19 of file resource.cc.

```
                                            : value(d){

                  }
```

4.4.3 Member Function Documentation

4.4.3.1 double Capacity::getValueRecursive ( )

Definition at line 59 of file resource.cc.

```
59                              {
60                  double result = value;
61                  std::vector<Capacity*>::iterator iter;
62                  if(!virtual_capacities.empty()){
63                                  for(iter = virtual_capacities.begin(); iter!=
      virtual_capacities.end(); iter ++){
64                                                  result += (*iter)->getValueRecursive();
65                                  }
66                  }
67                  return result;
68 }
```

4.4.3.2 Capacity::operator double ( )

Definition at line 37 of file resource.cc.

```
37                           {
38                  return value;
39 }
```

4.4.3.3 Capacity & Capacity::operator+= (const Capacity & *rhs*)

Definition at line 41 of file resource.cc.

```
41                                                 {
42                  this->value += rhs.value;
43                  return *this;
44 }
```

4.4.3.4 Capacity & Capacity::operator+= (const double & *rhs*)

Definition at line 50 of file resource.cc.

```
50                                               {
51                  this->value += rhs;
52                  return *this;
53 }
```

4.4.3.5 Capacity & Capacity::operator-= (const Capacity & *rhs*)

Definition at line 45 of file resource.cc.

```
45                                                 {
46                  this->value -= rhs.value;
47                  return *this;
48 }
```

4.4.3.6 Capacity & Capacity::operator-= (const double & *rhs*)

Definition at line 54 of file resource.cc.

```
                                                {
                    this->value -= rhs;
                    return *this;
}
```

4.4.3.7 Capacity & Capacity::operator= (const double & *val*)

Definition at line 23 of file resource.cc.

```
                                                {
                    this->value = val;
                    return *this;    // Return ref for multiple assignment
}
```

4.4.3.8 Capacity & Capacity::operator= (const Capacity & *c*)

Definition at line 28 of file resource.cc.

```
                                                {
                    if (this != &c) {  // make sure it is not the same object
                    this->value = c.value;
                    this->virtual_capacities.clear();
                    this->virtual_capacities = c.
      virtual_capacities;
                    }
                    return *this;    // Return ref for multiple assignment
}
```

4.4.4 Member Data Documentation

4.4.4.1 double Capacity::value

Definition at line 33 of file resource.h.

4.4.4.2 std::vector< Capacity∗ > Capacity::virtual_capacities

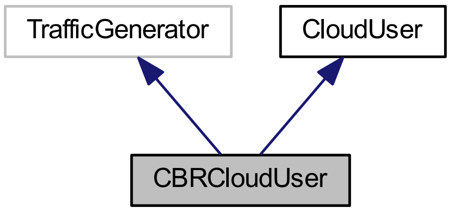
Definition at line 34 of file resource.h.

The documentation for this class was generated from the following files:

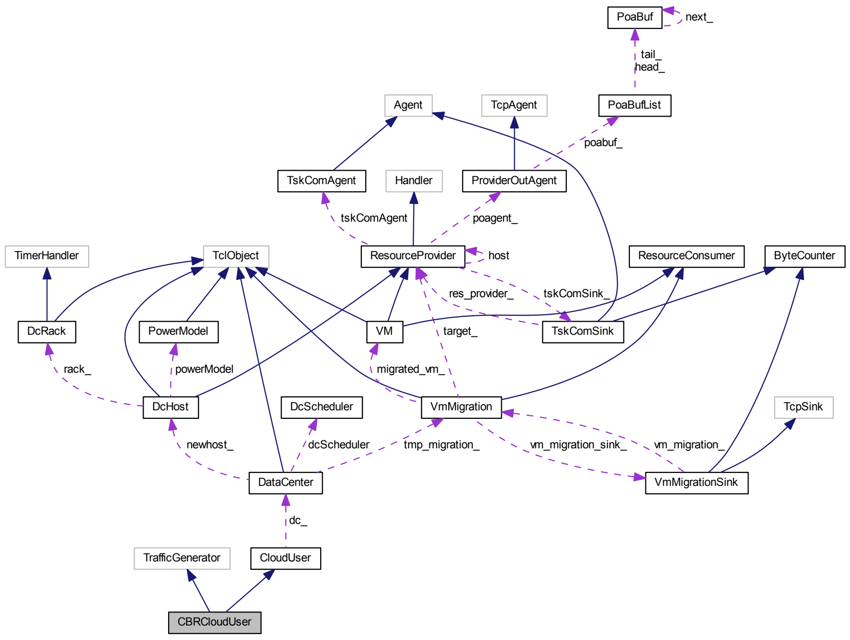
- resource.h
- resource.cc

4.5 CBRCloudUser Class Reference

Inheritance diagram for CBRCloudUser:



Collaboration diagram for CBRCloudUser:



Public Member Functions

- CBRCloudUser ()
- virtual double next_interval (int &)
- double interval ()
- virtual void timeout ()
- int command (int argc, const char ∗const ∗argv)
- void addDataCenterPointer (DataCenter ∗joindc_)

Protected Member Functions

- virtual void start ()
- void init ()

Protected Attributes

- double rate_
- double interval_
- double random_
- int seqno_
- int maxpkts_

Additional Inherited Members

4.5.1 Detailed Description

Definition at line 18 of file cbrclouduser.cc.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 CBRCloudUser::CBRCloudUser ( )

Definition at line 62 of file cbrclouduser.cc.

```
                              : seqno_(0)
{
                bind_time("random_tskmips_",random_tskmips_.avgp());
                bind_bw("rate_", &rate_);
                bind("random_", &random_);
                bind("packetSize_", &size_);
                bind("maxpkts_", &maxpkts_);

                // Bind CloudUser variables
                bind("id_", &id_);
                bind("tskmips_", &tskmips_);
                bind("tsksize_", &tsksize_);
                bind("tskmaxduration_", &tskmaxduration_);
                bind("mean_response_time_", &mean_response_time_);
                bind("sd_response_time_", &sd_response_time_);
                bind("unfinished_tasks_", &unfinished_tasks_);
}
```

4.5.3 Member Function Documentation

4.5.3.1 void CBRCloudUser::addDataCenterPointer (DataCenter ∗ *joindc_*)

4.5.3.2 int CBRCloudUser::command (int *argc*, const char ∗const ∗ *argv*)

Definition at line 47 of file cbrclouduser.cc.

```
                                                        {

                if(argc==3){
                                if (strcmp(argv[1], "join-datacenter") == 0) {
                                                DataCenter *dc = dynamic_cast<
     DataCenter*> (TclObject::lookup(argv[2]));
                                                if(dc){
                                                                dc_ = dc;
                                                                return (TCL_OK);
                                                }
                                                return (TCL_ERROR);
                                }
                }
                return Application::command(argc,argv);
}
```

4.5.3.3 void CBRCloudUser::init ( ) [protected]

Definition at line 80 of file cbrclouduser.cc.

```
{
                // compute inter-packet interval
                interval_ = (double)(size_ << 3)/(double)rate_;
                if (agent_)
                                if (agent_->get_pkttype() != PT_TCP &&
                                                                agent_->get_pkttype() != PT_TFRC)
                                                agent_->set_pkttype(PT_CBR);
}
```

4.5.3.4 double CBRCloudUser::interval ( ) [inline]

Definition at line 23 of file cbrclouduser.cc.

```
{ return (interval_); }
```

4.5.3.5 double CBRCloudUser::next_interval (int & *size*) [virtual]

Definition at line 97 of file cbrclouduser.cc.

```
{
                // Recompute interval in case rate_ or size_ has changes
                interval_ = (double)(size_ << 3)/(double)rate_;
                double t = interval_;
                if (random_)
                                t += interval_ * Random::uniform(-0.5, 0.5);
                size = size_;
                if (++seqno_ < maxpkts_)
                                return(t);
                else
                                return(-1);
}
```

4.5.3.6 void CBRCloudUser::start ( ) [protected], [virtual]

Definition at line 90 of file cbrclouduser.cc.

```
{
                init();
                running_ = 1;
                timeout();
}
```

4.5.3.7 void CBRCloudUser::timeout ( ) [virtual]

Definition at line 111 of file cbrclouduser.cc.

```
{
                if (! running_)
                                return;

                /* send a packet */
                dc_->receivedTsk(size_, createTask());
                /* figure out when to send the next one */
                nextPkttime_ = next_interval(size_);
                /* schedule it */
                if (nextPkttime_ > 0)
                                timer_.resched(nextPkttime_);
                else
                                running_ = 0;
}
```

4.5.4 Member Data Documentation

4.5.4.1 double CBRCloudUser::interval_ [protected]

Definition at line 32 of file cbrclouduser.cc.

4.5.4.2 int CBRCloudUser::maxpkts_ [protected]

Definition at line 35 of file cbrclouduser.cc.

4.5.4.3 double CBRCloudUser::random_ [protected]

Definition at line 33 of file cbrclouduser.cc.

4.5.4.4 double CBRCloudUser::rate_ [protected]

Definition at line 31 of file cbrclouduser.cc.

4.5.4.5 int CBRCloudUser::seqno_ [protected]

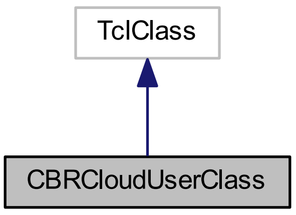
Definition at line 34 of file cbrclouduser.cc.

The documentation for this class was generated from the following file:

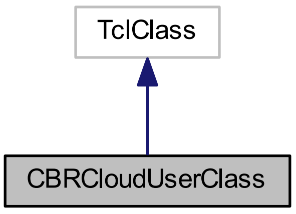
- cbrclouduser.cc

4.6 CBRCloudUserClass Class Reference

Inheritance diagram for CBRCloudUserClass:



Collaboration diagram for CBRCloudUserClass:



Public Member Functions

- CBRCloudUserClass ()
- TclObject ∗ create (int, const char ∗const ∗)

4.6.1 Detailed Description

Definition at line 39 of file cbrclouduser.cc.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 CBRCloudUserClass::CBRCloudUserClass ( ) [inline]

Definition at line 41 of file cbrclouduser.cc.

```
: TclClass("Application/Traffic/CBRCloudUser") {}
```

4.6.3 Member Function Documentation

4.6.3.1 TclObject∗ CBRCloudUserClass::create (int , const char ∗const ∗) [inline]

Definition at line 42 of file cbrclouduser.cc.

```
                                                    {
                                return (new CBRCloudUser());
                }
```

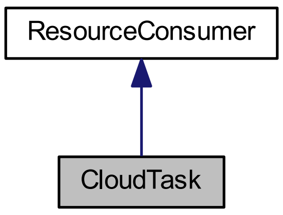
The documentation for this class was generated from the following file:

- cbrclouduser.cc

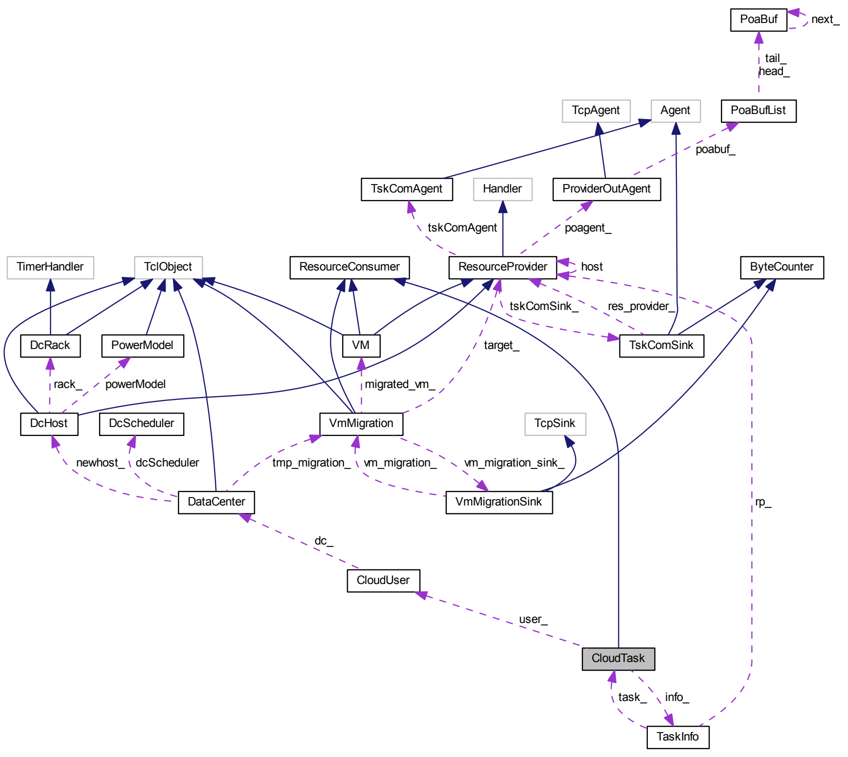
4.7 CloudTask Class Reference

`#include <cloudtask.h>`

Inheritance diagram for CloudTask:



Collaboration diagram for CloudTask:



Public Member Functions

- CloudTask ()
- CloudTask (unsigned int size, double duration, std::vector< Resource ∗ > demand, CloudUser ∗clouduser)
- virtual ∼CloudTask ()
- int getID ()
- double getMIPS (int rd, int cap)
- double getDeadline ()
- int getOutput ()
- void setMIPS (int rd, int cap, double mips)
- void setID (int id)
- void setDeadline (double deadline)
- void setOutput (int output)
- void setIntercom (int intercom)
- bool isFinished ()
- void printCompCapacites ()
- void removeTaskAlloc (TaskAlloc ∗ta)
- void fail (ResourceProvider ∗provider)
- void releaseAllTaskAllocs ()

Public Attributes

- int id_
- bool scheduled_

- bool started_
- double deadline_
- bool failed_
- std::vector< TaskAlloc ∗ > task_allocations_
- TaskInfo ∗ info_

Protected Member Functions

- void handler (Event ∗)

Protected Attributes

- CloudUser ∗ user_
- int output_
- int intercom_

4.7.1 Detailed Description

Definition at line 15 of file cloudtask.h.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 CloudTask::CloudTask ( )

Definition at line 16 of file cloudtask.cc.

```
16                      : id_(0),   scheduled_(false), started_(false),
      failed_(false), output_(0), intercom_(0)
17 {
18              isTask = true;
19 }
```

4.7.2.2 CloudTask::CloudTask (unsigned int *size,* double *duration,* std::vector< Resource ∗ > *demand,* CloudUser ∗ *clouduser*)

Definition at line 22 of file cloudtask.cc.

```
22                                                                                  :
23 ResourceConsumer(size, demand, true, false),id_(0),
      scheduled_(false), started_(false), failed_(false),
      user_(clouduser), output_(0), intercom_(0)
24 {
25
26              currProcRate_=0.0;
27              deadline_ = Scheduler::instance().clock() + duration;
28              isTask = true;
29              for(unsigned int rd = 0; rd < res_demands.size();rd++){
30                              if(res_demands.at(rd)->getType()==
      Computing){
31                                              for(unsigned int cap = 0; cap <
      res_demands.at(rd)->capacity.size();cap++){
32
      task_allocations_.push_back(new TaskAlloc(this,rd,cap));
33                                              }
34                              }
35              }
36
37 }
```

4.7.2.3 CloudTask::∼CloudTask ( ) [virtual]

Definition at line 39 of file cloudtask.cc.

```
{
                std::vector <ResDemand*>::iterator iter;
                for (iter = res_demands.begin() ; iter!=res_demands.end(); iter++)
                {
                                delete (*iter);
                }
}
```

4.7.3 Member Function Documentation

4.7.3.1 void CloudTask::fail (ResourceProvider * *provider*)

Perform failure - related actions. Typically used after failure of one of the task allocations is detected.

Definition at line 59 of file cloudtask.cc.

```
                                {
//              std::cerr << "Task\t" << id_ <<"\tfailed on provider:\t" << provider->id_ <<"\n";
                failed_ = true;
                if(this->started_==true){
                provider->releaseAllocation(this);
                }
                provider->tskFailed_++;
                releaseAllTaskAllocs();
                task_allocations_.clear();
}
```

4.7.3.2 double CloudTask::getDeadline ( ) [inline]

Definition at line 23 of file cloudtask.h.

```
{return deadline_;};
```

4.7.3.3 int CloudTask::getID ( ) [inline]

Definition at line 21 of file cloudtask.h.

```
{return id_;};
```

4.7.3.4 double CloudTask::getMIPS (int *rd,* int *cap*)

Definition at line 52 of file cloudtask.cc.

```
                                {
                if(res_demands.at(rd)->getType() != Computing){
                                std::cerr << "MIPS requested for non-Computing resource";
                                abort();
                }
                return res_demands.at(rd)->capacity.at(cap);}
```

4.7.3.5 int CloudTask::getOutput ( ) [inline]

Definition at line 24 of file cloudtask.h.

```
24 {return output_;};
```

4.7.3.6 void CloudTask::handler (Event *) [protected]

4.7.3.7 bool CloudTask::isFinished ( )

Definition at line 83 of file cloudtask.cc.

```
83                         {
84                 std::vector <ResDemand*>::iterator u_res;
85 
86                 /* //Check if all computational load is finished */
87                 if(task_allocations_.size()>0){
88                                 return false;
89                 } else {
90                                 return true;
91                 }
92 }
```

4.7.3.8 void CloudTask::printCompCapacites ( )

Definition at line 95 of file cloudtask.cc.

```
95                                 {
96                 std::vector <ResDemand*>::iterator u_res;
97                 std::cerr << "Capacities: ";
98                 for (u_res = res_demands.begin() ; u_res!=res_demands.end(); u_res++)
99                 {
100                                 if((*u_res)->getType()==Computing){
101                                                 std::vector <Capacity>::iterator cap;
102                                                 for(cap=(*u_res)->capacity.begin();cap!=(*u_res)->capacity.
      end();cap++){
103                                                                 std::cerr << (*cap) << " ";
104                                                 }
105                                 }
106                 }
107                 std::cerr << "\n";
108 }
```

4.7.3.9 void CloudTask::releaseAllTaskAllocs ( )

Remove all task allocations from the schedulers.

Definition at line 70 of file cloudtask.cc.

```
70                                   {
71                 std::vector<TaskAlloc*>::iterator iter;
72                 for(iter = task_allocations_.begin(); iter!=
      task_allocations_.end(); iter++){
73                                 if((*iter)->getCoreScheduler()!=NULL){
74                                 (*iter)->getCoreScheduler()->removeTaskAlloc((*iter));
75                                 }
76                 }
77 }
```

4.7.3.10 void CloudTask::removeTaskAlloc (TaskAlloc ∗ *ta*)

Removes task allocation.

Definition at line 78 of file cloudtask.cc.

```
                                                {
                task_allocations_.erase(remove(
    task_allocations_.begin(),task_allocations_.end(),ta),
                                                  task_allocations_.end()); /*erase-remove
     idiom*/
 }
```

4.7.3.11 void CloudTask::setDeadline (double *deadline*) [inline]

Definition at line 28 of file cloudtask.h.

```
 {deadline_ = deadline;};
```

4.7.3.12 void CloudTask::setID (int *id*) [inline]

Definition at line 27 of file cloudtask.h.

```
 {id_ = id;};
```

4.7.3.13 void CloudTask::setIntercom (int *intercom*) [inline]

Definition at line 30 of file cloudtask.h.

```
 {intercom_ = intercom;};
```

4.7.3.14 void CloudTask::setMIPS (int *rd,* int *cap,* double *mips*)

Definition at line 48 of file cloudtask.cc.

```
                                                      {
                res_demands.at(rd)->capacity.at(cap) = mips;
 }
```

4.7.3.15 void CloudTask::setOutput (int *output*) [inline]

Definition at line 29 of file cloudtask.h.

```
 {output_ = output;};
```

4.7.4 Member Data Documentation

4.7.4.1 double CloudTask::deadline_

task deadline

Definition at line 54 of file cloudtask.h.

4.7.4.2 bool CloudTask::failed_

Definition at line 55 of file cloudtask.h.

4.7.4.3 int CloudTask::id_

Definition at line 51 of file cloudtask.h.

4.7.4.4 TaskInfo* CloudTask::info_

Definition at line 61 of file cloudtask.h.

4.7.4.5 int CloudTask::intercom_ [protected]

amount of data in bytes to be transferred to another data center application.

Definition at line 68 of file cloudtask.h.

4.7.4.6 int CloudTask::output_ [protected]

amount of data in bytes sent out of the data center upon task completion.

Definition at line 67 of file cloudtask.h.

4.7.4.7 bool CloudTask::scheduled_

true if task has been scheduled

Definition at line 52 of file cloudtask.h.

4.7.4.8 bool CloudTask::started_

true if task has started its execution

Definition at line 53 of file cloudtask.h.

4.7.4.9 std::vector<TaskAlloc*> CloudTask::task_allocations_

Task allocations are objects that create many-to-many relationship between computational capacities of a task and core schedulers.

Definition at line 60 of file cloudtask.h.

4.7.4.10 CloudUser∗ CloudTask::user_ [protected]

The cloud user that created the task.

Definition at line 65 of file cloudtask.h.

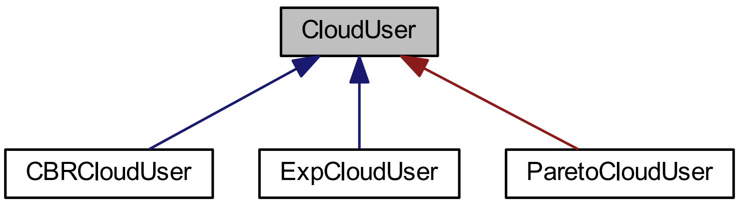
The documentation for this class was generated from the following files:

- cloudtask.h
- cloudtask.cc

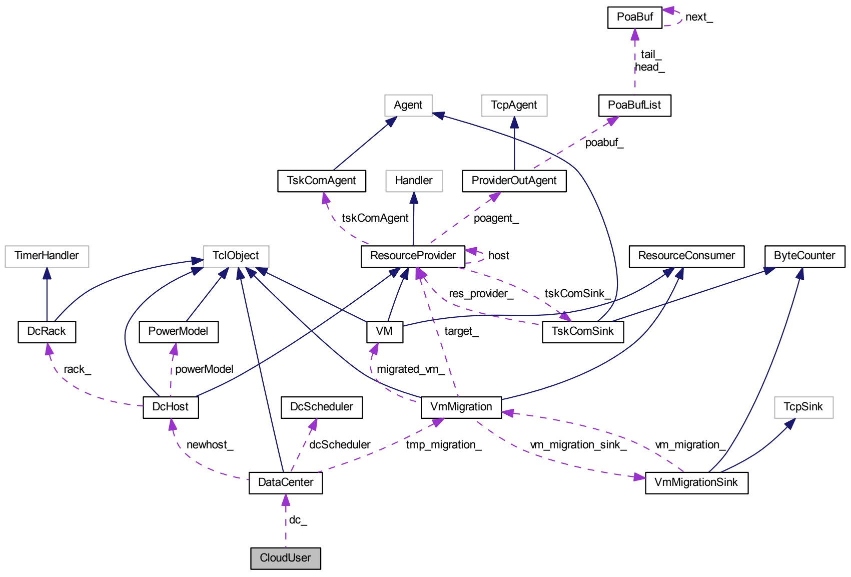
4.8 CloudUser Class Reference

`#include <clouduser.h>`

Inheritance diagram for CloudUser:



Collaboration diagram for CloudUser:



Public Member Functions

- CloudUser ()
- virtual ∼CloudUser ()
- CloudTask ∗ createTask ()
- void setRandomized (int i)
- int process_command (int argc, const char ∗const ∗argv)

Public Attributes

- int id_
- double tskmips_
- double memory_
- double storage_
- unsigned int tsksize_
- double tskmaxduration_
- int toutputsize_
- int tintercom_
- int randomized_
- double mean_response_time_
- double sd_response_time_
- int unfinished_tasks_

Protected Member Functions

- void printTasksStatus ()
- void postSimulationTestTasks ()
- void calculateStatistics ()

Protected Attributes

- DataCenter ∗ dc_
- int taskcounter_
- ExponentialRandomVariable random_tskmips_
- std::vector< TaskInfo ∗ > tasks_info_

4.8.1 Detailed Description

Definition at line 19 of file clouduser.h.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 CloudUser::CloudUser ( )

Definition at line 9 of file clouduser.cc.

```
9                       : id_(0), tskmips_(0) , memory_(0.0),
      storage_(0.0), tsksize_(0), tskmaxduration_(0.0),
10 toutputsize_ (0), tintercom_(0),randomized_(0),
      mean_response_time_(-1), sd_response_time_(-1),
11 unfinished_tasks_(-1), dc_(NULL),taskcounter_(0),
      random_tskmips_(0.0)
12 {
13 }
```

4.8.2.2 CloudUser::∼CloudUser ( ) [virtual]

Definition at line 15 of file clouduser.cc.

```
{
}
```

4.8.3 Member Function Documentation

4.8.3.1 void CloudUser::calculateStatistics ( ) [protected]

Definition at line 141 of file clouduser.cc.

```
                                    {
                std::vector<TaskInfo*>::iterator i;
                double sum = 0;
                int counter = 0;
                unfinished_tasks_ = 0;

                //mean calculation
                if(!tasks_info_.empty()){
                for(i = tasks_info_.begin(); i < tasks_info_.end(); i++){
                                if((*i)->getDcExitTime() != -1){
                                                sum+= (*i)->getDcExitTime() - (*i)->getReleaseTime();
                                                counter++;
                                } else {
                                                unfinished_tasks_++;
                                }
                }

                mean_response_time_ = sum / counter;

                // sd calculation
                sum = 0;
                double tmp;
                for(i = tasks_info_.begin(); i < tasks_info_.end(); i++){
                                if((*i)->getDcExitTime() != -1){
                                                tmp = pow( ((*i)->getDcExitTime() - (*i)->getReleaseTime()
      ) - mean_response_time_, 2.0f);

                                                sum += tmp;
                                }
                }
                sd_response_time_ = sqrt(sum/counter);
                } else {
                                std::cerr << "WARNING: No tasks generated by the cloud user: " <<
      id_ << " (normally it should not happen).\n";
                }
}
```

4.8.3.2 CloudTask ∗ CloudUser::createTask ( )

Definition at line 19 of file clouduser.cc.

```
{
                std::vector<Resource*> task_demand;

                std::vector<Capacity> task_proc_cap;
                double mips;
                int processes_number = 1;
                for(int i = 0 ; i < processes_number; i++){
                                if(!randomized_){
                                                mips = tskmips_/processes_number;
                                } else {
                                                do{
                                                mips = random_tskmips_.value()/
      processes_number;
                                                }while(mips > (tskmips_/processes_number)*
      tskmaxduration_*0.98);
                                }
                                task_proc_cap.push_back(mips);
35                  }
36                  task_demand.push_back(new Resource(Computing,1.0,task_proc_cap));
37
38                  if(memory_!=0){
39                  std::vector<Capacity> task_memory_cap;
40                  task_memory_cap.push_back(memory_);
41                  task_demand.push_back(new Resource(Memory,1.0,task_memory_cap));
42                  }
43
44                  if(storage_!=0){
45                  std::vector<Capacity> task_storage_cap;
46                  task_storage_cap.push_back(storage_);
47                  task_demand.push_back(new Resource(Storage,1.0,task_storage_cap));
48                  }
49
50                  //  std::cerr << "MIPS:" << tskmips_ << "\tMEM:" << memory_<< "\tSTO:" <<storage_ << "\n";
51                  // TODO: LEAK OCCURS: the created tasks are never released... However, they exist only
      until the end of a simulation.
52
53                  CloudTask *pTskObj = new CloudTask(tsksize_,
     tskmaxduration_,task_demand, this);
54                  pTskObj->setID(taskcounter_);
55 //               std::cout <<"Task generated, id: "<< pTskObj->id_ << "\n";
56                  pTskObj->setOutput(toutputsize_);
57                  pTskObj->setIntercom(tintercom_);
58                  TaskInfo* tmp_info_ =  new TaskInfo(pTskObj,Scheduler::instance().clock(),
     Scheduler::instance().clock() + tskmaxduration_);
59                  tasks_info_.push_back(tmp_info_);
60                  pTskObj->info_= tmp_info_;
61                  taskcounter_++;
62
63                  return pTskObj;
64 }
```

4.8.3.3 void CloudUser::postSimulationTestTasks ( ) [protected]

Definition at line 116 of file clouduser.cc.

```
116                                     {
117                  bool ok = true;
118                  std::vector<TaskInfo*>::iterator i;
119                  unfinished_tasks_ = 0;
120                  for(i = tasks_info_.begin(); i < tasks_info_.end(); i++){
121                                  if(false){
122 //                               if((*i)->getDcExitTime() == -1){
123                                                  ok = false;
124                                                  unfinished_tasks_++;
125                                                  std::cout << "Cloud User:\t" <<
     id_ << "\t";
126                                  std::cout << fixed << setprecision(2) << "Task unfinished, id: " << (*i)->
     getTaskId() <<
127                                                                  " Rel: "<< (*i)->getReleaseTime() <<
128                                                                  " Ser: "<< (*i)->getServerFinishTime() <<
129                                                                  " Ext: "<< (*i)->getDcExitTime() <<
130                                                                  " Due: " << (*i)->getDueTime() <<
131                                                                  "\n";
132                                  }
133                  }
134                  if(ok){
135                                  std::cout << "Cloud User:\t" << id_ << "\t: all tasks finished sucesfully.
     \n";
136                  } else {
137                                  std::cout << "Cloud User:\t" << id_ << "\t:\t"<<
     unfinished_tasks_ << "\ttasks did NOT exit datacenter.\n";
138                  }
139 }
```

4.8.3.4 void CloudUser::printTasksStatus ( ) [protected]

Definition at line 103 of file clouduser.cc.

```
103                              {
104                  std::vector<TaskInfo*>::iterator i;
105                  std::cout << "Cloud User:\t" << id_ << "\n";
106                  for(i = tasks_info_.begin(); i < tasks_info_.end(); i++){
107                                  std::cout << fixed << setprecision(2) << "T: " << (*i)->getTaskId() <<
```

```
                                                        " Rel: "<< (*i)->getReleaseTime() <<
                                                        " Ser: "<< (*i)->getServerFinishTime() <<
                                                        " Ext: "<< (*i)->getDcExitTime() <<
                                                        " Due: " << (*i)->getDueTime() <<
                                                        "\n";
                }
}
```

4.8.3.5 int CloudUser::process_command (int *argc,* const char ∗const ∗ *argv*)

Definition at line 73 of file clouduser.cc.

```
                                                    {
                if(argc==2){
                                if (strcmp(argv[1], "print-tasks-status") == 0) {
                                                printTasksStatus();
                                                return (TCL_OK);
                                } else if(strcmp(argv[1], "post-simulation-test-tasks") == 0){
                                                postSimulationTestTasks();
                                                return (TCL_OK);
                                } else if(strcmp(argv[1], "calculate-statistics") == 0){
                                                calculateStatistics();
                                                return (TCL_OK);
                                }


                } else if(argc==3){
                                if (strcmp(argv[1], "join-datacenter") == 0) {
                                                DataCenter *dc = dynamic_cast<
     DataCenter*> (TclObject::lookup(argv[2]));
                                                if(dc){
                                                                dc_ = dc;
                                                                return (TCL_OK);
                                                }
                                                return (TCL_ERROR);
                                } else          if (strcmp(argv[1], "set-randomized") == 0) {
                                                setRandomized(atoi(argv[2]));
                                                return (TCL_OK);
                                }
                }
                return -1;
}
```

4.8.3.6 void CloudUser::setRandomized (int *i*)

Definition at line 66 of file clouduser.cc.

```
                                {
                randomized_  =i;
                if(i!=0){
                random_tskmips_.setavg(tskmips_);
                }
}
```

4.8.4 Member Data Documentation

4.8.4.1 DataCenter∗ CloudUser::dc_ [protected]

Definition at line 45 of file clouduser.h.

4.8.4.2 int CloudUser::id_

Task ID

Definition at line 26 of file clouduser.h.

4.8.4.3 double CloudUser::mean_response_time_

Definition at line 40 of file clouduser.h.

4.8.4.4 double CloudUser::memory_

Task memory_ demand

Definition at line 30 of file clouduser.h.

4.8.4.5 ExponentialRandomVariable CloudUser::random_tskmips_ [protected]

Definition at line 47 of file clouduser.h.

4.8.4.6 int CloudUser::randomized_

Definition at line 38 of file clouduser.h.

4.8.4.7 double CloudUser::sd_response_time_

Definition at line 41 of file clouduser.h.

4.8.4.8 double CloudUser::storage_

Generated task computing demand

Definition at line 31 of file clouduser.h.

4.8.4.9 int CloudUser::taskcounter_ [protected]

Definition at line 46 of file clouduser.h.

4.8.4.10 std::vector<TaskInfo*> CloudUser::tasks_info_ [protected]

Definition at line 48 of file clouduser.h.

4.8.4.11 int CloudUser::tintercom_

Size of inter-task communication

Definition at line 36 of file clouduser.h.

4.8.4.12 int CloudUser::toutputsize_

Task output size in bytes (sent out of the data center)

Definition at line 35 of file clouduser.h.

4.8.4.13 double CloudUser::tskmaxduration_

Task execution deadline

Definition at line 33 of file clouduser.h.

4.8.4.14 double CloudUser::tskmips_

Generated task computing demand

Definition at line 29 of file clouduser.h.

4.8.4.15 unsigned int CloudUser::tsksize_

Size of task description sent to a server for execution in bytes

Definition at line 32 of file clouduser.h.

4.8.4.16 int CloudUser::unfinished_tasks_

Definition at line 42 of file clouduser.h.

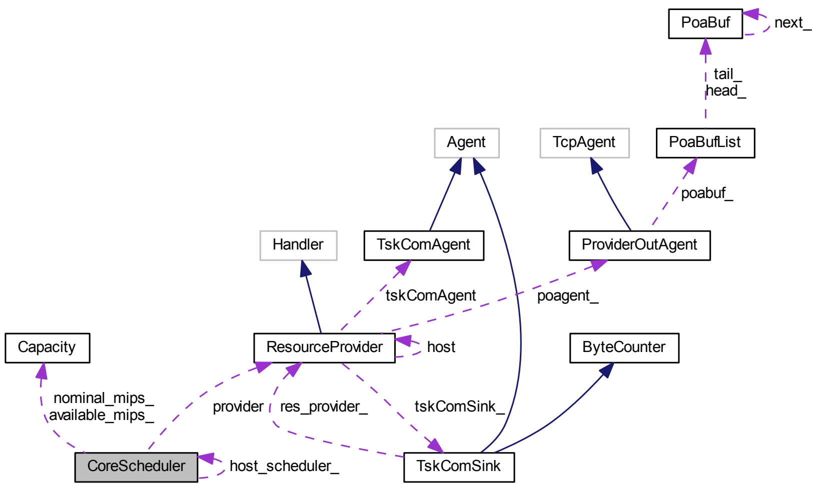
The documentation for this class was generated from the following files:

- clouduser.h
- clouduser.cc

4.9 CoreScheduler Class Reference

```
#include <corescheduler.h>
```

Collaboration diagram for CoreScheduler:



Public Member Functions

- CoreScheduler (Capacity ∗nominal_mips_, Capacity ∗available_mips_)
- virtual ∼CoreScheduler ()
- void setProvider (ResourceProvider ∗provider)
- void setDVFS (int eDVFS_enabled_)
- double getCurrentMIPS ()
- double getCurrentMIPSRecursive ()
- double getNominalMIPS ()
- double getAvailableMIPS ()
- ResourceProvider ∗ getProvider ()
- void updateTskList (double c_mips)
- void updateTskList ()
- void removeCompleted ()
- CloudTask ∗ removeTaskAlloc (std::vector< TaskAlloc ∗ >::iterator &iter, bool executed)
- void removeTaskAlloc (TaskAlloc ∗ta)
- void removeFailedTaskAlloc (std::vector< TaskAlloc ∗ >::iterator &iter, bool executed)
- void updateTskComputingRates (double c_mips)
- double getMostUrgentTaskRate ()
- void setComputingRate ()
- int getAllTasksNumber ()
- void assignTask (TaskAlloc ∗tskobj)
- void executeTask (TaskAlloc ∗tskobj)
- bool removeFromAssginedList (TaskAlloc ∗tskobj)
- bool removeAllocationsFromAssginedList (CloudTask ∗tskobj)
- void startTaskExecution (CloudTask ∗tskobj)
- void addVcoreScheduler (CoreScheduler ∗cs)
- void removeVcoreScheduler (CoreScheduler ∗cs)
- CoreScheduler ∗ getHostScheduler ()

Private Attributes

- Capacity ∗ nominal_mips_
- Capacity ∗ available_mips_
- double current_mips_
- ResourceProvider ∗ provider
- std::vector< CoreScheduler ∗ > hosted_vcores_schedulers
- std::vector< TaskAlloc ∗ > tasks_alloc_list_
- std::vector< TaskAlloc ∗ > tasks_alloc_assigned_
- CoreScheduler ∗ host_scheduler_
- int eDVFS_enabled_
- int tskAllocFailed_

4.9.1 Detailed Description

Definition at line 26 of file corescheduler.h.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 CoreScheduler::CoreScheduler (Capacity ∗ *nominal_mips_*, Capacity ∗ *available_mips_*)

Definition at line 12 of file corescheduler.cc.

```
                                                                                  :
    current_mips_(0.0), host_scheduler_(NULL),
    tskAllocFailed_(0) {
                 this->nominal_mips_=nominal_mips_;
                 this->available_mips_=available_mips_;
                 tasks_alloc_list_.clear();
                 tasks_alloc_assigned_.clear();
                 hosted_vcores_schedulers.clear();

 }
```

4.9.2.2 CoreScheduler::∼CoreScheduler ( ) [virtual]

Definition at line 23 of file corescheduler.cc.

```
                               {
                 std::vector <TaskAlloc*>::iterator task;
                 for(task = tasks_alloc_list_.begin(); task!=
    tasks_alloc_list_.end() ;task++){
                                 delete (*task);
                 }
                 tasks_alloc_list_.~vector();
                 for(task = tasks_alloc_assigned_.begin(); task!=
    tasks_alloc_assigned_.end() ;task++){
                                 delete (*task);
                 }
                 tasks_alloc_assigned_.~vector();
 }
```

4.9.3 Member Function Documentation

4.9.3.1 void CoreScheduler::addVcoreScheduler (CoreScheduler ∗ *cs*)

Definition at line 308 of file corescheduler.cc.

```
                                                      {
                 hosted_vcores_schedulers.push_back(cs);
                 cs->host_scheduler_ = this;
 }
```

4.9.3.2 void CoreScheduler::assignTask (TaskAlloc ∗ *tskobj*)

Definition at line 71 of file corescheduler.cc.

```
                                              {
                 tasks_alloc_assigned_.push_back(tskobj);
                 tskobj->setCoreScheduler(this);
 }
```

4.9.3.3 void CoreScheduler::executeTask (TaskAlloc ∗ *tskobj*)

Definition at line 76 of file corescheduler.cc.

```
                                                {
                tasks_alloc_list_.push_back(tskobj);           // add to the active tasks
      links
                tskobj->setCoreScheduler(this);
}
```

4.9.3.4 int CoreScheduler::getAllTasksNumber ( )

Definition at line 67 of file corescheduler.cc.

```
                                     {
                return tasks_alloc_list_.size() +
     tasks_alloc_assigned_.size();
}
```

4.9.3.5 double CoreScheduler::getAvailableMIPS ( )

Definition at line 59 of file corescheduler.cc.

```
                                       {
                return *available_mips_;
}
```

4.9.3.6 double CoreScheduler::getCurrentMIPS ( )

Definition at line 43 of file corescheduler.cc.

```
                                     {
                return current_mips_;
}
```

4.9.3.7 double CoreScheduler::getCurrentMIPSRecursive ( )

Definition at line 46 of file corescheduler.cc.

```
                                              {
                double result = getCurrentMIPS();
                std::vector<CoreScheduler*>::iterator iter;
                int i = 1;
                for(iter = hosted_vcores_schedulers.begin(); iter !=
     hosted_vcores_schedulers.end();iter++){
                                result += (*iter)->getCurrentMIPSRecursive();
                                i++;
                }
                return result;
}
```

4.9.3.8 CoreScheduler ∗ CoreScheduler::getHostScheduler ( )

Definition at line 319 of file corescheduler.cc.

```
                                               {
                return host_scheduler_;
}
```

4.9.3.9 double CoreScheduler::getMostUrgentTaskRate ( )

Definition at line 201 of file corescheduler.cc.

```
{
                std::vector<TaskAlloc*>::iterator iter;

                /* update what is already computing */

                for (iter = tasks_alloc_list_.begin(); iter !=
    tasks_alloc_list_.end(); iter++)
                {
                                (*iter)->updateMIPS();          // update what is already computing
                }
                /* remove completed */
                removeCompleted();

                /* Compute highest MIPS/deadline ratio */
                double maxrate = 0.0;

                /* get most urgent task rate from the execution list */
                for (iter = tasks_alloc_list_.begin(); iter !=
    tasks_alloc_list_.end();)
                {
                                if((double)((*iter)->getDeadline() - Scheduler::instance().clock())>0 &&
    (*iter)->cloudTask->failed_==false){
                                                double rate = (double)(*iter)->getMIPS()/(double)((*iter)->
    getDeadline() - Scheduler::instance().clock());
                                                if(rate>maxrate){
                                                                maxrate = rate;
                                                }
                                                iter++;
                                }
                                else {
                                                removeFailedTaskAlloc(iter,true);
                                }
                }

                /* get most urgent task rate from the in-fly list */
                for (iter = tasks_alloc_assigned_.begin(); iter !=
    tasks_alloc_assigned_.end();)
                {
                                if((double)((*iter)->getDeadline() - Scheduler::instance().clock())>0 &&
    (*iter)->cloudTask->failed_==false){
                                                double rate = (double)(*iter)->getMIPS()/(double)((*iter)->
    getDeadline() - Scheduler::instance().clock());
                                                if(rate>maxrate){
                                                                maxrate = rate;
                                                }
                                                iter++;
                                } else {
                                                removeFailedTaskAlloc(iter,false);
                                }
                }
                if (maxrate > getAvailableMIPS()){return
    getAvailableMIPS();}
                else{return maxrate;}
}
```

4.9.3.10 double CoreScheduler::getNominalMIPS ( )

Definition at line 56 of file corescheduler.cc.

```
                                    {
                return *nominal_mips_;
}
```

4.9.3.11 ResourceProvider * CoreScheduler::getProvider ( )

Definition at line 63 of file corescheduler.cc.

```
                                            {
                return provider;
}
```

4.9.3.12 bool CoreScheduler::removeAllocationsFromAssginedList (CloudTask ∗ *tskobj*)

Definition at line 94 of file corescheduler.cc.

```
94                                                                  {
95                  vector<TaskAlloc*>::iterator iter;
96                  bool found = false;
97                  for (iter = tasks_alloc_assigned_.begin(); iter !=
      tasks_alloc_assigned_.end();)
98                  {
99                                  /* task received remove from in-fly list */
100                                  if ((*iter)->cloudTask->id_ == tskobj->id_) {
101                                                  iter = tasks_alloc_assigned_.erase(
      iter);
102                                                  found = true;
103                                  } else {
104                                                  iter++;
105                                  }
106                  }
107                  return found;
108 }
```

4.9.3.13 void CoreScheduler::removeCompleted ( )

Definition at line 173 of file corescheduler.cc.

```
174 {
175                  std::vector<TaskAlloc*>::iterator iter;
176 
177                  /* remove completed tasks from the execution list */
178                  for (iter = tasks_alloc_list_.begin(); iter !=
      tasks_alloc_list_.end();)
179                  {
180                                  /* task should be completed and remove it from the list */
181                                  if ((*iter)->getMIPS() <= 1) {
182                                                  CloudTask* ct =
      removeTaskAlloc(iter,true);
183                                                  //check if finished:
184                                                  if(ct->isFinished()){
185                                                                  provider->
      releaseAllocation(ct);
186                                                                  provider->
      sendTaskOutput(ct);
187                                                  }
188                                  }
189                                  /*Task run over its deadline, remove it and mark as failed. */
190                                  else if(Scheduler::instance().clock() >= (*iter)->cloudTask->getDeadline())
       {
191                                                  removeFailedTaskAlloc(iter,true);
192                                  } else if ((*iter)->cloudTask->failed_==true){
193                                                  std::cerr << "This should not happen!\n";
194                                  }
195                                  else {
196                                                  iter++;
197                                  }
198                  }
199 }
```

4.9.3.14 void CoreScheduler::removeFailedTaskAlloc (std::vector< TaskAlloc ∗ >::iterator & *iter,* bool *executed*)

Definition at line 158 of file corescheduler.cc.

```
159 {
160                  if((*iter)->cloudTask->failed_==false){
161                                  (*iter)->cloudTask->fail(this->provider);
162                  }
163                  // REMOVE ALL TASK ALLOCATIONS FOR THE CLOUDTASK OF THIS TASK_ALLOC
164                  if(executed){
165                                  iter = tasks_alloc_list_.begin();
166                  } else {
167                                  iter = tasks_alloc_assigned_.begin();
168                  }
169 }
```

4.9.3.15 bool CoreScheduler::removeFromAssginedList (TaskAlloc ∗ *tskobj*)

Definition at line 81 of file corescheduler.cc.

```
81                                                        {
82                 vector<TaskAlloc*>::iterator iter;
83                 for (iter = tasks_alloc_assigned_.begin(); iter !=
     tasks_alloc_assigned_.end();iter++)
84                 {
85                                 /* task received remove from in-fly list */
86                                 if ((*iter) == tskobj) {
87                                                 tasks_alloc_assigned_.erase(iter);
88                                                 return true;
89                                 }
90                 }
91                 return false;
92 }
```

4.9.3.16 CloudTask∗ CoreScheduler::removeTaskAlloc (std::vector< TaskAlloc ∗ >::iterator & *iter*, bool *executed*)

4.9.3.17 void CoreScheduler::removeTaskAlloc (TaskAlloc ∗ *ta*)

Definition at line 148 of file corescheduler.cc.

```
149 {
150                 this->provider->updateEnergyAndConsumption();
151                 tskAllocFailed_++;
152                 tasks_alloc_list_.erase(remove(
     tasks_alloc_list_.begin(),tasks_alloc_list_.end(),ta),
153                                                 tasks_alloc_list_.end()); /*erase-remove
      idiom*/
154                 tasks_alloc_assigned_.erase(remove(
     tasks_alloc_assigned_.begin(),tasks_alloc_assigned_.end(),ta),
155                                                 tasks_alloc_assigned_.end()); /*
     erase-remove idiom*/
156 }
```

4.9.3.18 void CoreScheduler::removeVcoreScheduler (CoreScheduler ∗ *cs*)

Definition at line 313 of file corescheduler.cc.

```
313                                                          {
314                 hosted_vcores_schedulers.erase(remove(
     hosted_vcores_schedulers.begin(),
     hosted_vcores_schedulers.end(),cs),
315                                                 hosted_vcores_schedulers.end()); /*
     erase-remove idiom*/
316                 cs->host_scheduler_ = NULL;
317 }
```

4.9.3.19 void CoreScheduler::setComputingRate ( )

Definition at line 294 of file corescheduler.cc.

```
295 {
296                 if (eDVFS_enabled_) {
297                                 /* Max requested rate times the number of active taks */
298                                 current_mips_ =
     getMostUrgentTaskRate()* tasks_alloc_list_.size();
299                 } else {
300                                 /* no energy saving */
301                                 if (tasks_alloc_list_.size() != 0){
     current_mips_ = getAvailableMIPS();}
302                                 else {(current_mips_) = 0;}
303                 }
304                 /* new computing rate, report it to tasks */
305                 updateTskComputingRates(current_mips_);
306 }
```

4.9.3.20 void CoreScheduler::setDVFS (int *eDVFS_enabled_*)

Definition at line 39 of file corescheduler.cc.

```
                                    {
                this->eDVFS_enabled_ = eDVFS_enabled_;
}
```

4.9.3.21 void CoreScheduler::setProvider (ResourceProvider ∗ *provider*)

Definition at line 35 of file corescheduler.cc.

```
                                                     {
                this->provider = provider;
}
```

4.9.3.22 void CoreScheduler::startTaskExecution (CloudTask ∗ *tskobj*)

Definition at line 110 of file corescheduler.cc.

```
                                                     {
                vector<TaskAlloc*>::iterator iter;
                for (iter = tasks_alloc_assigned_.begin(); iter !=
     tasks_alloc_assigned_.end();)
                {
                                /* task received remove from in-fly list */
                                if ((*iter)->cloudTask->id_ == tskobj->id_) {
                                                executeTask((*iter));
                                                (*iter)->setExecTime(Scheduler::instance().clock());
                                                iter = tasks_alloc_assigned_.erase(
     iter);

                                } else {
                                                iter++;
                                }
                }
                if(!tskobj->started_){
                                tskobj->started_ = true;
                }
                this->updateTskList();
}
```

4.9.3.23 void CoreScheduler::updateTskComputingRates (double *c_mips*)

Definition at line 251 of file corescheduler.cc.

```
{
                vector<TaskAlloc*>::iterator iter;


                for (iter = tasks_alloc_list_.begin(); iter !=
     tasks_alloc_list_.end(); iter++)
                {
                                /* each task with then update mips left */
                                (*iter)->setComputingRate((double)c_mips/
     tasks_alloc_list_.size());
                }
}
```

4.9.3.24 void CoreScheduler::updateTskList (double *c_mips*)

Definition at line 263 of file corescheduler.cc.

```
{

                vector<TaskAlloc*>::iterator iter;
                if (tasks_alloc_list_.size()==0) return;
//              std::cout << "Prov: " << provider->id_ << "\n";
                /* update task computing rates to see which tasks are completed */
                updateTskComputingRates(c_mips);
                removeCompleted();

                /* set server computing rate */
                setComputingRate();

                /* compute next deadline */
                double nextDeadline = DBL_MAX;
                for (iter = tasks_alloc_list_.begin(); iter !=
  tasks_alloc_list_.end(); iter++)
                {
                                if (nextDeadline > (*iter)->execTime()){
                                                nextDeadline = (*iter)->execTime();
                                }
                }


                provider->scheduleNextExent(nextDeadline);

}
```

4.9.3.25 void CoreScheduler::updateTskList ( )

Definition at line 290 of file corescheduler.cc.

```
                                 {
                updateTskList(this->current_mips_);
}
```

4.9.4 Member Data Documentation

4.9.4.1 Capacity* CoreScheduler::available_mips_ [private]

MIPS unavailable for this scheduler (reserved for hosted vms)

Definition at line 64 of file corescheduler.h.

4.9.4.2 double CoreScheduler::current_mips_ [private]

MIPS currently used by this scheduler

Definition at line 65 of file corescheduler.h.

4.9.4.3 int CoreScheduler::eDVFS_enabled_ [private]

DVFS flag, influences the scheduling policy

Definition at line 72 of file corescheduler.h.

4.9.4.4 CoreScheduler∗ CoreScheduler::host_scheduler_ [private]

The scheduler that hosts this (next level) scheduler

Definition at line 71 of file corescheduler.h.

4.9.4.5 std::vector<CoreScheduler ∗> CoreScheduler::hosted_vcores_schedulers [private]

List of schedulers of hosted vcores

Definition at line 68 of file corescheduler.h.

4.9.4.6 Capacity∗ CoreScheduler::nominal_mips_ [private]

Maximal available MIPS from ResourceSpec

Definition at line 63 of file corescheduler.h.

4.9.4.7 ResourceProvider∗ CoreScheduler::provider [private]

The resorce provider that uses the scheduler

Definition at line 67 of file corescheduler.h.

4.9.4.8 std::vector<TaskAlloc ∗> CoreScheduler::tasks_alloc_assigned_ [private]

in-fly list

Definition at line 70 of file corescheduler.h.

4.9.4.9 std::vector<TaskAlloc ∗> CoreScheduler::tasks_alloc_list_ [private]

execution list

Definition at line 69 of file corescheduler.h.

4.9.4.10 int CoreScheduler::tskAllocFailed_ [private]

Number of TaskAllocations that failed on this scheduler (not used yet)

Definition at line 73 of file corescheduler.h.

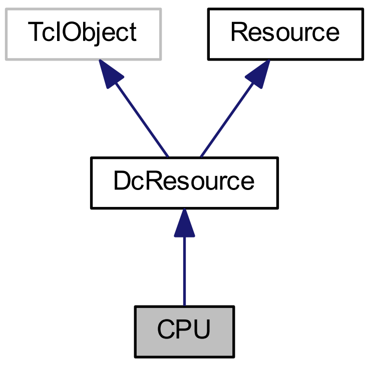
The documentation for this class was generated from the following files:

- corescheduler.h
- corescheduler.cc

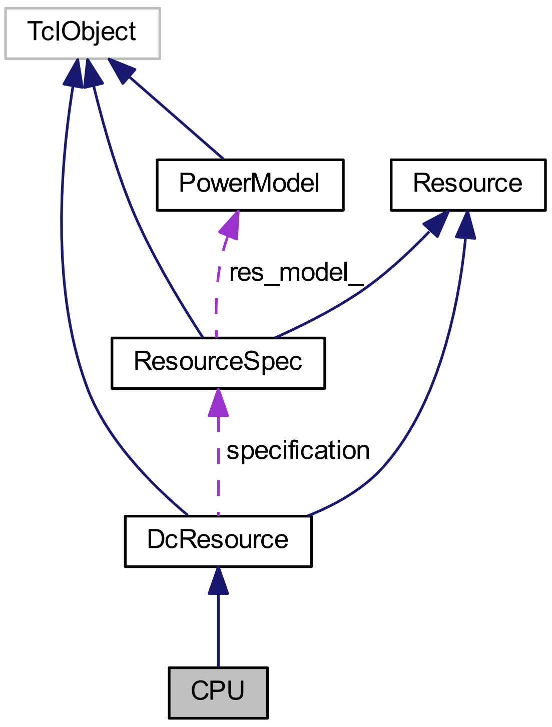
4.10 CPU Class Reference

`#include <cpu.h>`

Inheritance diagram for CPU:



Collaboration diagram for CPU:



Public Member Functions

- CPU ()
- virtual ∼CPU ()
- double getCurrentMIPS ()
- double getNominalMIPS ()
- virtual int setSpecification (ResourceSpec ∗resspec)
- void setProvider (ResourceProvider ∗provider)
- void setDVFS (int eDVFS_enabled_)
- virtual double getUtilization ()
- virtual void print ()
- virtual int command (int argc, const char ∗const ∗argv)

Public Attributes

- std::vector< CoreScheduler ∗ > cores_schedulers_

Private Member Functions

- void getMIPS ()

Private Attributes

- double nominal_mips_

Additional Inherited Members

4.10.1 Detailed Description

Definition at line 18 of file cpu.h.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 CPU::CPU ( )

Definition at line 18 of file cpu.cc.

```
18         {
19
20 }
```

4.10.2.2 CPU::∼CPU ( ) [virtual]

Definition at line 22 of file cpu.cc.

```
22          {
23
24 }
```

4.10.3 Member Function Documentation

4.10.3.1 int CPU::command (int *argc,* const char ∗const ∗ *argv*) [virtual]

Reimplemented from DcResource.

Definition at line 45 of file cpu.cc.

```
                                                {

                 if (argc == 2) {
                                 if (strcmp(argv[1], "print") == 0) {
                                                 this->print();
                                                 return (TCL_OK);
                                 } else if (strcmp(argv[1], "getMIPS") == 0) {
                                                 this->getMIPS();
                                                 return (TCL_OK);
                                 }
                 }
                 return (CPU::command(argc, argv));
 }
```

4.10.3.2 double CPU::getCurrentMIPS ( )

Definition at line 110 of file cpu.cc.

```
                          {
                 std::vector<CoreScheduler*>::iterator iter;
                 double result = 0;
                 for(iter= cores_schedulers_.begin();
                                                 iter != cores_schedulers_.end();
                                                 iter++){

                                 result += (*iter)->getCurrentMIPSRecursive();
                 }
                 return result;
 }
```

4.10.3.3 void CPU::getMIPS ( ) [private]

Definition at line 31 of file cpu.cc.

```
                  {
                 double result = 0;
                 std::vector <Capacity>::iterator iter;
                 for (iter = this->specification->capacity.begin(); iter!=this->
      specification->capacity.end(); iter++)
                 {
                                 result += (*iter);
                 }
                 char out[100];
                 sprintf(out,"set tmp_cpu_mips %.0f",result);
                 Tcl& tcl = Tcl::instance();
                 tcl.eval(out);

 }
```

4.10.3.4 double CPU::getNominalMIPS ( )

Definition at line 121 of file cpu.cc.

```
                          {
                 return nominal_mips_;
 }
```

4.10.3.5 double CPU::getUtilization ( ) [virtual]

Reimplemented from DcResource.

Definition at line 125 of file cpu.cc.

```
                          {
                return getCurrentMIPS()/getNominalMIPS();
}
```

4.10.3.6 void CPU::print ( ) [virtual]

Reimplemented from DcResource.

Definition at line 26 of file cpu.cc.

```
               {
                std::cerr << "CPU: ";
                DcResource::print();
}
```

4.10.3.7 void CPU::setDVFS (int *eDVFS_enabled_*)

Definition at line 98 of file cpu.cc.

```
                                  {
                if(cores_schedulers_.empty()){
                                std::cerr << "No core schedulers!\n";
                }
                std::vector<CoreScheduler*>::iterator iter;
                for(iter= cores_schedulers_.begin();
                                                iter != cores_schedulers_.end();
                                                iter++){
                                (*iter)->setDVFS(eDVFS_enabled_);
                }
}
```

4.10.3.8 void CPU::setProvider (ResourceProvider * *provider*)

Sets the resource provider of the resource.

Definition at line 85 of file cpu.cc.

```
                                        {
                if(cores_schedulers_.empty()){
                                std::cerr << "No core schedulers!\n";
                }
                std::vector<CoreScheduler*>::iterator iter;
                for(iter= cores_schedulers_.begin();
                                                iter != cores_schedulers_.end();
                                                iter++){
                                (*iter)->setProvider(provider);
                }

}
```

4.10.3.9 int CPU::setSpecification (ResourceSpec ∗ *resspec*) [virtual]

Sets the resource specification of the resource.

Reimplemented from DcResource.

Definition at line 61 of file cpu.cc.

```
                                           {
                if(resspec==NULL){
                                std::cerr << "ERROR: Null pointer passed as ResourceSpec.";
                                return 1;
                }
                this->DcResource::setSpecification(resspec);
                std::vector<Capacity>::iterator iter_nominal;
                std::vector<Capacity>::iterator iter_reserved;
                for(iter_nominal= resspec->capacity.begin(),iter_reserved=this->
      capacity.begin();
                                                iter_nominal != resspec->
      capacity.end();
                                                iter_nominal++,iter_reserved++){
                                cores_schedulers_.push_back(new
      CoreScheduler(&(*iter_nominal),&(*iter_reserved)));
                }
                std::vector<CoreScheduler*>::iterator iter;
                nominal_mips_ = 0;
                for(iter= cores_schedulers_.begin();
                                                iter != cores_schedulers_.end();
                                                iter++){
                                nominal_mips_ += (*iter)->getNominalMIPS();
                }

                return 0;
}
```

4.10.4 Member Data Documentation

4.10.4.1 std::vector<CoreScheduler ∗> CPU::cores_schedulers_

List of cores schedulers of the CPU

Definition at line 37 of file cpu.h.

4.10.4.2 double CPU::nominal_mips_ [private]

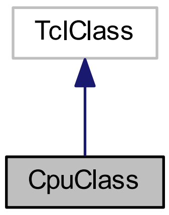
Definition at line 41 of file cpu.h.

The documentation for this class was generated from the following files:

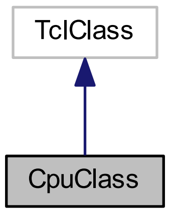
- cpu.h
- cpu.cc

4.11 CpuClass Class Reference

Inheritance diagram for CpuClass:



Collaboration diagram for CpuClass:



Public Member Functions

- CpuClass ()
- TclObject ∗ create (int argc, const char ∗const ∗argv)

4.11.1 Detailed Description

Definition at line 10 of file cpu.cc.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 CpuClass::CpuClass ( ) [inline]

Definition at line 12 of file cpu.cc.

```
12 : TclClass("CPU") {}
```

4.11.3 Member Function Documentation

4.11.3.1 TclObject∗ CpuClass::create (int *argc*, const char ∗const ∗ *argv*) [inline]

Definition at line 13 of file cpu.cc.

```
                                                                   {
                                    return (new CPU());
                   }
```

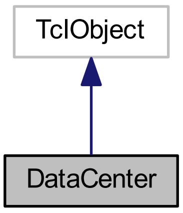
The documentation for this class was generated from the following file:

- cpu.cc

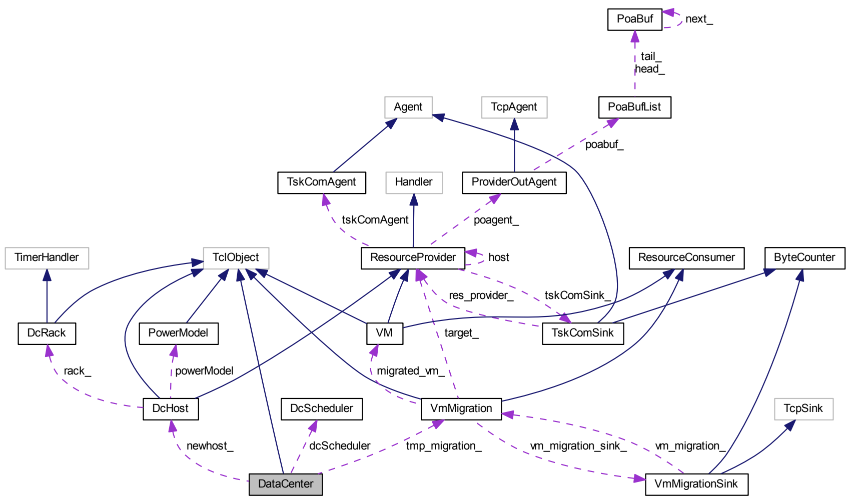
4.12 DataCenter Class Reference

`#include <datacenter.h>`

Inheritance diagram for DataCenter:



Collaboration diagram for DataCenter:



Public Member Functions

- DataCenter ()
- virtual ∼DataCenter ()
- void clear ()
- void addHostPointer (DcHost ∗newhst)
- void addVmPointer (VM ∗newvm)
- void addHostTaskAgentPointer (TskComAgent ∗newagnt)
- void addVmTaskAgentPointer (TskComAgent ∗newagnt)
- void addResourceSpecificationPointer (ResourceSpec ∗newresspec)
- void addVirtualResourceSpecificationPointer (ResourceSpec ∗newresspec)
- void addPModelPointer (PowerModel ∗newPModel)
- int initiallyConfigureVms ()
- int setScheduler (const char ∗scheduler_name)
- void migrateVm (VM ∗vm, ResourceProvider ∗target)
- int configureResource (DcResource ∗confRes, const char ∗spec_name)
- int configureVirtualResource (DcResource ∗confRes, const char ∗spec_name)
- void printResourceSpecs ()
- virtual void receivedTsk (int tsksize, CloudTask ∗pTask, const char ∗flags=0)
- virtual int command (int argc, const char ∗const ∗argv)

Public Attributes

- int tskSubmitted_
- int tskFailed_
- double avgLoad_
- double avgLoadMem_
- double avgLoadStor_
- double avgPower_

Protected Member Functions

- TskComAgent ∗ scheduleRoundRobin (CloudTask ∗tsk)
- TskComAgent ∗ scheduleRoundRobin (CloudTask ∗tsk, std::vector< TskComAgent ∗ > agent_list)
- TskComAgent ∗ scheduleGreen (CloudTask ∗tsk)
- TskComAgent ∗ scheduleGreenVmOnly (CloudTask ∗tsk)
- void computeLoad ()
- void setVmScheduling (bool scheduleOnVms)

Protected Attributes

- vector< ResourceProvider ∗ > host_list
- vector< TskComAgent ∗ > host_agent_list
- vector< ResourceProvider ∗ > vm_list
- vector< TskComAgent ∗ > vm_agent_list
- vector< PowerModel ∗ > power_model_list
- DcScheduler ∗ dcScheduler
- vector< ResourceSpec ∗ > resource_specification_list
- vector< ResourceSpec ∗ > virt_resource_specification_list
- DcHost ∗ newhost_
- int numHostTskAgents_
- int numVmTskAgents_
- VmMigration ∗ tmp_migration_
- bool scheduleOnVms_

Private Member Functions

- void createNewMigration ()

4.12.1 Detailed Description

Definition at line 27 of file datacenter.h.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 DataCenter::DataCenter ( )

Definition at line 22 of file datacenter.cc.

```
                        : tskSubmitted_ (0),tskFailed_ (0),
      avgLoad_(0.0), avgLoadMem_(0.0),avgLoadStor_(0.0),
      avgPower_(0.0), dcScheduler(NULL),  numHostTskAgents_(0),
      numVmTskAgents_(0),scheduleOnVms_(false)

{
                bind("tskSubmitted_", &tskSubmitted_);
                bind("avgLoad_", &avgLoad_);
                bind("avgLoadMem_", &avgLoadMem_);
                bind("avgLoadStor_", &avgLoadStor_);
                bind("avgPower_", &avgPower_);
                bind("tskFailed_", &tskFailed_);
//              dcScheduler = new GreenScheduler();
}
```

4.12.2.2 DataCenter::∼DataCenter ( ) [virtual]

Definition at line 35 of file datacenter.cc.

```
{
                clear();
}
```

4.12.3 Member Function Documentation

4.12.3.1 void DataCenter::addHostPointer (DcHost ∗ *newhst*)

Definition at line 65 of file datacenter.cc.

```
{
                host_list.push_back(newhst);
}
```

4.12.3.2 void DataCenter::addHostTaskAgentPointer (TskComAgent ∗ *newagnt*)

Definition at line 74 of file datacenter.cc.

```
{
                newagnt->set_pkttype(PT_CLOUD_USER);
                host_agent_list.push_back(newagnt);
}
```

4.12.3.3 void DataCenter::addPModelPointer (PowerModel ∗ *newPModel*)

Register power model in Data Center.

Definition at line 91 of file datacenter.cc.

```
                                         {
                power_model_list.push_back(newPModel);
}
```

4.12.3.4 void DataCenter::addResourceSpecificationPointer (ResourceSpec ∗ *newresspec*)

Registers resource used in Data Center.

Definition at line 86 of file datacenter.cc.

```
{
                resource_specification_list.push_back(newresspec);
}
```

4.12.3.5 void DataCenter::addVirtualResourceSpecificationPointer (ResourceSpec ∗ *newresspec*)

Registers virtual resource used in Data Center.

Definition at line 96 of file datacenter.cc.

```
{
                virt_resource_specification_list.push_back(newresspec);
}
```

4.12.3.6 void DataCenter::addVmPointer (VM ∗ *newvm*)

Definition at line 70 of file datacenter.cc.

```
                              {
                vm_list.push_back(newvm);
}
```

4.12.3.7 void DataCenter::addVmTaskAgentPointer (TskComAgent ∗ *newagnt*)

Definition at line 80 of file datacenter.cc.

```
{
                newagnt->set_pkttype(PT_CLOUD_USER);
                vm_agent_list.push_back(newagnt);
}
```

4.12.3.8 void DataCenter::clear ( )

Definition at line 40 of file datacenter.cc.

```
40                      {
41
42              //TODO: this is the stub of the function, it should be finished to fully clear a DataCenter
      object.
43              host_list.clear();
44              host_agent_list.clear();
45              vector <ResourceSpec*>::iterator rs;
46              for(rs = resource_specification_list.begin();rs!=
      resource_specification_list.end(); rs++){
47                      delete (* rs);
48              }
49              resource_specification_list.clear();
50              vector <ResourceProvider*>::iterator iter;
51              for(iter = host_list.begin();iter!= host_list.end(); iter++){
52                      delete (* iter);
53              }
54              for(iter = vm_list.begin();iter!= vm_list.end(); iter++){
55                      delete (* iter);
56              }
57              host_list.clear();
58              vm_list.clear();
59 }
```

4.12.3.9 int DataCenter::command (int *argc,* const char *const * *argv*) [virtual]

Definition at line 221 of file datacenter.cc.

```
222 {
223                 if (argc == 2) {
224                         if (strcmp(argv[1], "compute-load") == 0) {
225                                 computeLoad();
226                                 return (TCL_OK);
227                         } else          if (strcmp(argv[1], "print-resspec") == 0) {
228                                 printResourceSpecs();
229                                 return (TCL_OK);
230                         } else if (strcmp(argv[1], "initially-configure-vms") == 0) {
231                                 if(initiallyConfigureVms() == 0){
232                                         return (TCL_OK);
233                                 } else {
234                                         return (TCL_ERROR);
235                                 }
236                         } else          if (strcmp(argv[1], "clear") == 0) {
237                                 clear();
238                                 return (TCL_OK);
239                         } else          if (strcmp(argv[1], "schedule-on-vms") == 0) {
240                                 setVmScheduling(true);
241                                 return (TCL_OK);
242                         }
243                 } else if (argc == 3) {
244                         if (strcmp(argv[1], "add-dchost") == 0) {
245                                 DcHost *hst = dynamic_cast<
      DcHost*> (TclObject::lookup(argv[2]));
246                                 if(hst){
247                                         addHostPointer(hst);
248                                         return (TCL_OK);
249                                 }
250                                 return (TCL_ERROR);
251                         }               else            if (strcmp(argv[1], "add-vm") == 0) {
252                                 VM *vm = dynamic_cast<VM*> (TclObject::lookup(argv[2]))
      ;
253                                 if(vm){
254                                         addVmPointer(vm);
255                                         return (TCL_OK);
256                                 }
257                                 return (TCL_ERROR);
258                         }
259                         else if (strcmp(argv[1], "add-hosttaskagent") == 0) {
260                                 TskComAgent *agnt = dynamic_cast<
      TskComAgent*> (TclObject::lookup(argv[2]));
261                                 if(agnt){
262
      addHostTaskAgentPointer(agnt);
263
      numHostTskAgents_ = (int) host_agent_list.size();
```

```
                                                return (TCL_OK);
                                        }
                                        return (TCL_ERROR);
                        } else if (strcmp(argv[1], "add-vmtaskagent") == 0) {
                                        TskComAgent *agnt = dynamic_cast<
    TskComAgent*> (TclObject::lookup(argv[2]));
                                        if(agnt){

    addVmTaskAgentPointer(agnt);
                                                numVmTskAgents_= (int)
    vm_agent_list.size();
                                                return (TCL_OK);
                                        }
                                        return (TCL_ERROR);
                        }
                        else if (strcmp(argv[1], "add-resspec") == 0) {
                                        ResourceSpec *resspec = dynamic_cast<
    ResourceSpec*> (TclObject::lookup(argv[2]));
                                        if(resspec){

    addResourceSpecificationPointer(resspec);
                                                return (TCL_OK);
                                        }
                                        return (TCL_ERROR);
                        }               else if (strcmp(argv[1], "add-vresspec") == 0) {
                                        ResourceSpec *resspec = dynamic_cast<
    ResourceSpec*> (TclObject::lookup(argv[2]));
                                        if(resspec){

    addVirtualResourceSpecificationPointer(resspec);
                                                return (TCL_OK);
                                        }
                                        return (TCL_ERROR);
                        }               else if (strcmp(argv[1], "add-pmodel") == 0) {
                                        PowerModel *pmodel = dynamic_cast<
    PowerModel*> (TclObject::lookup(argv[2]));
                                        if(pmodel){
                                                addPModelPointer(pmodel);
                                                return (TCL_OK);
                                        }
                                        return (TCL_ERROR);
                        }  else if (strcmp(argv[1], "get-newest-migration") == 0) {
                                        VmMigration *mig = dynamic_cast<
    VmMigration*> (TclObject::lookup(argv[2]));
                                        if(mig){
                                                tmp_migration_=mig;
                                                return (TCL_OK);
                                        }
                                        return (TCL_ERROR);
                        } else if (strcmp(argv[1], "set-scheduler") == 0) {
                                        if(setScheduler(argv[2])){
                                                return (TCL_OK);
                                        }
                                        return (TCL_ERROR);
                        }
                } else if (argc == 4) {
                        if (strcmp(argv[1], "configure-resource") == 0) {
                                        DcResource *res = dynamic_cast<
    DcResource*> (TclObject::lookup(argv[2]));
                                        if(res){
                                                if(
    configureResource(res,argv[3])==0){
                                                        return (TCL_OK);
                                                } else {
                                                        return (TCL_ERROR);
                                                }
                                        }
                                        return (TCL_ERROR);
                        } else          if (strcmp(argv[1], "configure-vresource") == 0) {
                                        DcResource *res = dynamic_cast<
    DcResource*> (TclObject::lookup(argv[2]));
                                        if(res){
                                                if(
    configureVirtualResource(res,argv[3])==0){
                                                        return (TCL_OK);
                                                } else {
                                                        return (TCL_ERROR);
                                                }
                                        }
                                        return (TCL_ERROR);
                        } else          if (strcmp(argv[1], "migrate-vm") == 0) {
                                        VM *vm = dynamic_cast<VM*> (TclObject::lookup(argv[2]))
    ;
                                        ResourceProvider *target = dynamic_cast<
    ResourceProvider*> (TclObject::lookup(argv[3]));
                                        if(vm && target){
                                                migrateVm( vm, target);
```

```
                                                        return (TCL_OK);
                                        } else {
                                                        return (TCL_ERROR);
                                        }
                        }
                        return (TCL_ERROR);


        }
        return (DataCenter::command(argc, argv));
}
```

4.12.3.10 void DataCenter::computeLoad ( ) [protected]

Definition at line 348 of file datacenter.cc.

```
{
        /* Traverse servers and compute their average load */
        vector <ResourceProvider*>::iterator iter;

        double avgLoad = 0;
        double avgLoadMem = 0;
        double avgLoadStor = 0;
        double avgPower = 0;
        for (iter = host_list.begin(); iter!=host_list.end(); iter++)
        {

                        avgLoad         += (*iter)->getResTypeUtil(
   Computing);
                        avgLoadMem      += (*iter)->getResTypeUtil(
   Memory);
                        avgLoadStor += (*iter)->getResTypeUtil(Storage);
                        avgPower += ((DcHost*)(*iter))->eCurrentConsumption_;

        }

        for (iter = vm_list.begin(); iter!=vm_list.end(); iter++)
        {
                                                        (*iter)->getResTypeUtil(
   Computing);
                                                        (*iter)->getResTypeUtil(
   Memory);
                                                        (*iter)->getResTypeUtil(
   Storage);
        }



        avgLoad_ = avgLoad / host_list.size();
        avgLoadMem_= avgLoadMem / host_list.size();
        avgLoadStor_= avgLoadStor / host_list.size();
        avgPower_= avgPower / host_list.size();
}
```

4.12.3.11 int DataCenter::configureResource (DcResource * confRes, const char * spec_name)

Definition at line 391 of file datacenter.cc.

```
                                                                {
        vector <ResourceSpec*>::iterator iter;
        int result = 1;
        std::string test = spec_name;
        for (iter = resource_specification_list.begin(); iter!=
   resource_specification_list.end(); iter++)
        {
                        if((*iter)->name_==test){
                                        result = 0;
                                        break;
                        }
        }
        if(result==0){
                        confRes->setSpecification(*iter);
        } else {
                        std::cerr << "ERROR: The requested resource specification is not registered
    in the data center!\n";
        }
        return result;
}
```

4.12.3.12 int DataCenter::configureVirtualResource (DcResource * *confRes*, const char * *spec_name*)

Definition at line 410 of file datacenter.cc.

```
                                                                          {
                vector <ResourceSpec*>::iterator iter;
                int result = 1;
                std::string test = spec_name;
                for (iter = virt_resource_specification_list.begin(); iter!=virt_resource_specification_list.end(); iter++)
                {
                                if((*iter)->name_==test){
                                                result = 0;
                                                break;
                                }
                }
                if(result==0){
                                confRes->setSpecification(*iter);
                } else {
                                std::cerr << "ERROR: The requested virtual resource specification is not registered in the data center!\n";
                }
                return result;
}
```

4.12.3.13 void DataCenter::createNewMigration ( ) [private]

Definition at line 142 of file datacenter.cc.

```
                                 {
                // Create and allocate VmMigration object
                Tcl& tcl = Tcl::instance();
                tcl.evalf("set vmmigration_($next_migration_id) [new VmMigration]");
                tcl.evalf("$vmmigration_($next_migration_id) set-id $next_migration_id");
                tcl.evalf(
                                                "$DCenter get-newest-migration $vmmigration_($next_migration_id)");
}
```

4.12.3.14 int DataCenter::initiallyConfigureVms ( )

TODO: Dynamically configure the initial state of VMs.

Definition at line 101 of file datacenter.cc.

```
{
                char output[100];
                // Create the output for the Tcl interpreter.
                sprintf(output, "puts \"Configuring VMs...\"");
                Tcl& tcl = Tcl::instance();
                tcl.eval(output);
                return 0;
}
```

4.12.3.15 void DataCenter::migrateVm (VM * *vm*, ResourceProvider * *target*)

Definition at line 151 of file datacenter.cc.

```
                                                                      {
                createNewMigration(); // Creates and sets new migrations as tmp_migration_
                tmp_migration_->initalizeMigration(vm,target);

                tmp_migration_->startMigration();
                // The finalization of migration will be done by the migration object.
                // The data center (probably) does not need to be informed about that, as the vm agent will be re-attached to the new location.
                tmp_migration_ = NULL;
}
```

4.12.3.16 void DataCenter::printResourceSpecs ( )

Definition at line 382 of file datacenter.cc.

```
                                {
                vector <ResourceSpec*>::iterator iter;
                for (iter = resource_specification_list.begin(); iter!=
     resource_specification_list.end(); iter++)
                {
                                (*iter)->print();
                }

}
```

4.12.3.17 void DataCenter::receivedTsk (int *tsksize*, CloudTask * *pTask*, const char * *flags* = 0) [virtual]

Definition at line 161 of file datacenter.cc.

```
{
                /* Update Stats */
                tskSubmitted_ ++;

                /* Schedule it */
                TskComAgent *tagent_ ;
                if(scheduleOnVms_)// You can switch to another scheduler or providers list
      using this variable.
                {
                                tagent_ = dcScheduler->scheduleTask((
     CloudTask *) pTask,vm_list); }
                else {
                                tagent_ = dcScheduler->scheduleTask((
     CloudTask *) pTask,host_list);
                }
                if(tagent_==NULL){
                                tskFailed_++;
                } else {
                                tagent_->sendmsg(tsksize, pTask, flags);
                }
}
```

4.12.3.18 TskComAgent * DataCenter::scheduleGreen (CloudTask * *tsk*) [protected]

Definition at line 188 of file datacenter.cc.

```
{
                vector <ResourceProvider*>::iterator iter;

                for (iter = host_list.begin(); iter!=host_list.end(); iter++)
                {
                                if ((*iter)->trySchedulingTsk(tsk))
                                                return (*iter)->getTskComAgent();
//                                              return (host_agent_list.at((*iter)->id_));
                }
                return NULL;
}
```

4.12.3.19 TskComAgent * DataCenter::scheduleGreenVmOnly (CloudTask * *tsk*) [protected]

Definition at line 208 of file datacenter.cc.

```
{
                vector <ResourceProvider*>::iterator iter;

                for (iter = vm_list.begin(); iter!=vm_list.end(); iter++)
                {
                                if ((*iter)->trySchedulingTsk(tsk))
                                                return (*iter)->getTskComAgent();
//                                              return (vm_agent_list.at((*iter)->id_));
                }
                return NULL;
}
```

4.12.3.20 TskComAgent ∗ DataCenter::scheduleRoundRobin (CloudTask ∗ *tsk*) [protected]

Definition at line 181 of file datacenter.cc.

```
{
                int j = tskSubmitted_% numHostTskAgents_;

                return (host_agent_list.at(j));
}
```

4.12.3.21 TskComAgent ∗ DataCenter::scheduleRoundRobin (CloudTask ∗ *tsk*, std::vector< TskComAgent ∗ > *agent_list*) [protected]

Definition at line 201 of file datacenter.cc.

```
{
                int j = tskSubmitted_% agent_list.size();

                return (agent_list.at(j));
}
```

4.12.3.22 int DataCenter::setScheduler (const char ∗ *scheduler_name*)

Release old DcScheduler, and create and set new by name

Definition at line 111 of file datacenter.cc.

```
                                                        {
                if(dcScheduler!=NULL){
                delete dcScheduler;
                }
                std::cout << "Selected DC scheduler: " << scheduler_name << "\n";
                if(strcmp(scheduler_name, "Green") == 0){
                                dcScheduler = new GreenScheduler();
                                return 1;
                } else if(strcmp(scheduler_name, "RoundRobin") == 0){
                                dcScheduler = new
      RoundRobinsScheduler();
                                return 1;
                } else if(strcmp(scheduler_name, "Random") == 0){
                                dcScheduler = new RandomScheduler();
                                return 1;
                } else if(strcmp(scheduler_name, "RandDENS") == 0){
                                dcScheduler = new RandDENS();
                                return 1;
                } else if(strcmp(scheduler_name, "BestDENS") == 0){
                                dcScheduler = new BestDENS();
                                return 1;
                } else if(strcmp(scheduler_name, "HEROS") == 0){
                                dcScheduler = new HerosScheduler();
                                return 1;
                }


                std::cerr << "Unknown scheduler type: " << scheduler_name;
                abort();

}
```

4.12.3.23 void DataCenter::setVmScheduling (bool *scheduleOnVms*) [protected]

Definition at line 61 of file datacenter.cc.

```
                                                   {
                this->scheduleOnVms_ = scheduleOnVms;
}
```

4.12.4 Member Data Documentation

4.12.4.1 double DataCenter::avgLoad_

Definition at line 54 of file datacenter.h.

4.12.4.2 double DataCenter::avgLoadMem_

Definition at line 55 of file datacenter.h.

4.12.4.3 double DataCenter::avgLoadStor_

Definition at line 56 of file datacenter.h.

4.12.4.4 double DataCenter::avgPower_

Definition at line 57 of file datacenter.h.

4.12.4.5 DcScheduler∗ DataCenter::dcScheduler [protected]

Definition at line 66 of file datacenter.h.

4.12.4.6 vector<TskComAgent∗> DataCenter::host_agent_list [protected]

Definition at line 61 of file datacenter.h.

4.12.4.7 vector<ResourceProvider∗> DataCenter::host_list [protected]

Definition at line 60 of file datacenter.h.

4.12.4.8 DcHost∗ DataCenter::newhost_ [protected]

Definition at line 73 of file datacenter.h.

4.12.4.9 int DataCenter::numHostTskAgents_ [protected]

Definition at line 75 of file datacenter.h.

4.12.4.10 int DataCenter::numVmTskAgents_ [protected]

Definition at line 76 of file datacenter.h.

4.12.4.11 vector<PowerModel∗> DataCenter::power_model_list [protected]

Definition at line 64 of file datacenter.h.

4.12.4.12 vector< ResourceSpec∗ > DataCenter::resource_specification_list [protected]

Definition at line 70 of file datacenter.h.

4.12.4.13 bool DataCenter::scheduleOnVms_ [protected]

Definition at line 87 of file datacenter.h.

4.12.4.14 VmMigration∗ DataCenter::tmp_migration_ [protected]

Definition at line 78 of file datacenter.h.

4.12.4.15 int DataCenter::tskFailed_

Definition at line 53 of file datacenter.h.

4.12.4.16 int DataCenter::tskSubmitted_

Definition at line 52 of file datacenter.h.

4.12.4.17 vector< ResourceSpec∗ > DataCenter::virt_resource_specification_list [protected]

Definition at line 71 of file datacenter.h.

4.12.4.18 vector<TskComAgent∗> DataCenter::vm_agent_list [protected]

Definition at line 63 of file datacenter.h.

4.12.4.19 vector<ResourceProvider∗> DataCenter::vm_list [protected]

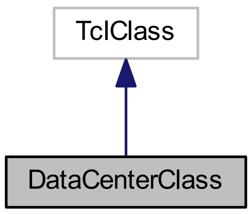
Definition at line 62 of file datacenter.h.

The documentation for this class was generated from the following files:

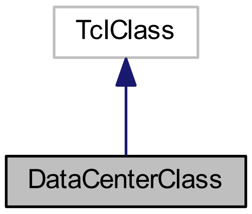
- datacenter.h
- datacenter.cc

4.13 DataCenterClass Class Reference

Inheritance diagram for DataCenterClass:



Collaboration diagram for DataCenterClass:



Public Member Functions

- DataCenterClass ()
- TclObject ∗ create (int argc, const char ∗const ∗argv)

4.13.1 Detailed Description

Definition at line 14 of file datacenter.cc.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 DataCenterClass::DataCenterClass ( ) [inline]

Definition at line 16 of file datacenter.cc.

```
: TclClass("DataCenter") {}
```

4.13.3 Member Function Documentation

4.13.3.1 TclObject∗ DataCenterClass::create (int *argc,* const char ∗const ∗ *argv*) [inline]

Definition at line 17 of file datacenter.cc.

```
                                                                 {
                                   return (new DataCenter());
                  }
```

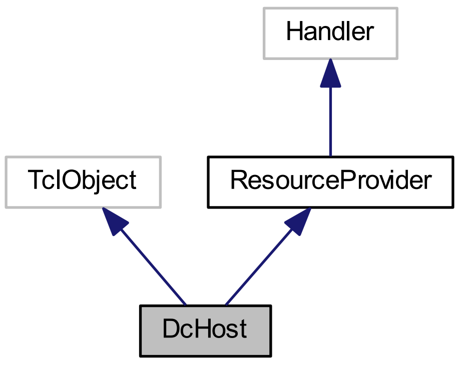
The documentation for this class was generated from the following file:

- datacenter.cc

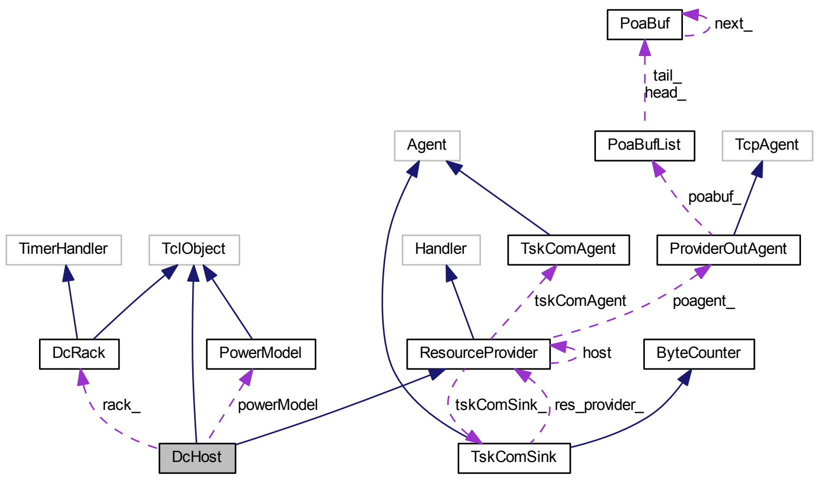
4.14 DcHost Class Reference

```
#include <dchost.h>
```

Inheritance diagram for DcHost:



Collaboration diagram for DcHost:



Public Member Functions

- DcHost ()
- virtual ∼DcHost ()
- virtual void print ()
- virtual void printTasklist ()
- virtual int command (int argc, const char ∗const ∗argv)
- virtual void updateEnergyAndConsumption ()

Public Attributes

- DcRack ∗ rack_
- PowerModel ∗ powerModel
- double eConsumed_
- double eNominalrate_
- double eCurrentConsumption_
- int eDNS_enabled_

Protected Member Functions

- void setCurrentConsumption ()
- void eUpdate ()
- virtual void addResource (DcResource ∗res)
- void setPowerModel (PowerModel ∗pModel)

Protected Attributes

- double eLastUpdateTime_

Additional Inherited Members

4.14.1 Detailed Description

Definition at line 27 of file dchost.h.

4.14.2 Constructor & Destructor Documentation

4.14.2.1 DcHost::DcHost ( )

Definition at line 16 of file dchost.cc.

```
                : eConsumed_(0.0), eNominalrate_ (0.0),
      eCurrentConsumption_ (0.0),   eDNS_enabled_(0.0) ,
      eLastUpdateTime_(0.0)
{
                bind("id_", &id_);
                bind("ntasks_", &ntasks_);
                bind("currentLoad_", &currentLoad_);
                bind("currentLoadMem_", &currentLoadMem_);
                bind("currentLoadStor_", &currentLoadStor_);
                bind("tskFailed_", &tskFailed_);

                bind("eConsumed_", &eConsumed_);
      /* total W of energy consumed */
                bind("eNominalrate_", &eNominalrate_);
                bind("eCurrentConsumption_", &eCurrentConsumption_);
      /* current consumption rate */

                bind("eDVFS_enabled_", &eDVFS_enabled_);
      /* ON when DVFS is enabled */
                bind("eDNS_enabled_", &eDNS_enabled_);
      /* ON when DNS is enabled */

}
```

4.14.2.2 DcHost::∼DcHost ( ) [virtual]

Definition at line 34 of file dchost.cc.

```
{
                delete powerModel;
}
```

4.14.3 Member Function Documentation

4.14.3.1 void DcHost::addResource (DcResource ∗ *res*) [protected], [virtual]

Reimplemented from ResourceProvider.

Definition at line 130 of file dchost.cc.

```
                                        {
                ResourceProvider::addResource(res);
                if(res->specification->getPowerModel()!=NULL){
                                powerModel->addComponent(res);
                }
}
```

4.14.3.2 int DcHost::command (int *argc,* const char ∗const ∗ *argv*) [virtual]

Reimplemented from ResourceProvider.

Definition at line 44 of file dchost.cc.

```
{
                Tcl& tcl = Tcl::instance();

                if (argc == 2) {
                                if (strcmp(argv[1], "start") == 0) {

                                                /* start counting energy consumed */
                                                setCurrentConsumption();
                                                eLastUpdateTime_ = Scheduler::instance().
      clock();
                                                started_ = true;
                                                return (TCL_OK);
                                } else if (strcmp(argv[1], "stop") == 0) {
                                                /* update total energy consumed */
                                                updateEnergyAndConsumption();
                                                return (TCL_OK);
                                } else if (strcmp(argv[1], "print") == 0) {
                                                /* print general info */
                                                print();
                                                return (TCL_OK);
                                }
                } else if (argc == 3) {
                                if (strcmp(argv[1], "set-power-model") == 0) {
                                                PowerModel* pModel = (
      PowerModel*) TclObject::lookup(argv[2]);
                                                if (pModel == NULL) {
                                                                tcl.resultf("no such power model %s", argv[
      2]);
                                                                return(TCL_ERROR);
                                                }
                                                setPowerModel(pModel);
                                                return(TCL_OK);
                                }
                }
                return (ResourceProvider::command(argc, argv));
}
```

4.14.3.3 void DcHost::eUpdate ( ) [protected]

Definition at line 120 of file dchost.cc.

```
121 {
122                 /* Get time spent since last update */
123                 double etime = (Scheduler::instance().clock() - eLastUpdateTime_)/3600;
      /* time in hours */
124 
125                 eConsumed_ += etime * eCurrentConsumption_;
126                 eLastUpdateTime_ = Scheduler::instance().clock();
127 
128 }
```

4.14.3.4 void DcHost::print ( ) [virtual]

Implements ResourceProvider.

Definition at line 79 of file dchost.cc.

```
79                  {
80                 std::cout << "DcHost:\t";
81                 std::cout << id_;
82                 std::cout << "\n";
83                 std::cout << "Resources:\n";
84                 std::vector <std::vector<DcResource*> >::iterator iter_out;
85                 for(iter_out = resource_list.begin(); iter_out!=
      resource_list.end() ;iter_out++){
86                                 std::vector <DcResource*>::iterator iter;
87                                 for (iter = iter_out->begin(); iter!=iter_out->end(); iter++)
88                                 {
89                                                 (*iter)->print();
90                                 }
91                 }
92                 std::cout << "\n";
93 
94 }
```

4.14.3.5 void DcHost::printTasklist ( ) [virtual]

Reimplemented from ResourceProvider.

Definition at line 96 of file dchost.cc.

```
96                          {
97                 std::vector<CloudTask *>::iterator iter;
98                 std::cout <<"Host " <<this->id_  << "\n";
99                 ResourceProvider::printTasklist();
100 }
```

4.14.3.6 void DcHost::setCurrentConsumption ( ) [protected]

Definition at line 102 of file dchost.cc.

```
103 {
104                 double * predictors = new double[LastResType+1];
105                 bool idle = true;
106                 for(int i = Computing; i <= LastResType; i++){
107                                 predictors[i]=updateResTypeUtil(static_cast<res_type>(i));
108                                 if(predictors[i]!=0){
109                                                 idle=false;
110                                 }
111                 }
112                 if((eDNS_enabled_) && (idle)){
113                                 eCurrentConsumption_ = 0;
114                 } else {
115                                 eCurrentConsumption_ =
      powerModel->estimate(4,predictors);
116                 }
117                 delete[] predictors;
118 }
```

4.14.3.7 void DcHost::setPowerModel (PowerModel ∗ *pModel*) [protected]

Definition at line 39 of file dchost.cc.

```
                                       {
                powerModel = pModel;
 }
```

4.14.3.8 void DcHost::updateEnergyAndConsumption ( ) [virtual]

Implements ResourceProvider.

Definition at line 138 of file dchost.cc.

```
                                   {
                setCurrentConsumption();
                eUpdate();
 }
```

4.14.4 Member Data Documentation

4.14.4.1 double DcHost::eConsumed_

total W of energy consumed

Definition at line 39 of file dchost.h.

4.14.4.2 double DcHost::eCurrentConsumption_

current consumption rate

Definition at line 41 of file dchost.h.

4.14.4.3 int DcHost::eDNS_enabled_

ON when dynamic shutdown is enabled

Definition at line 44 of file dchost.h.

4.14.4.4 double DcHost::eLastUpdateTime_ [protected]

Definition at line 57 of file dchost.h.

4.14.4.5 double DcHost::eNominalrate_

nominal consumption rate at full load at max CPU frequency

Definition at line 40 of file dchost.h.

4.14.4.6 PowerModel∗ DcHost::powerModel

Definition at line 34 of file dchost.h.

4.14.4.7 DcRack∗ DcHost::rack_

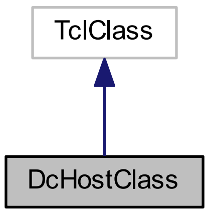
Definition at line 33 of file dchost.h.

The documentation for this class was generated from the following files:

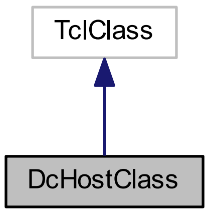
- dchost.h
- dchost.cc

4.15 DcHostClass Class Reference

Inheritance diagram for DcHostClass:



Collaboration diagram for DcHostClass:



Public Member Functions

- DcHostClass ()
- TclObject ∗ create (int argc, const char ∗const ∗argv)

4.15.1 Detailed Description

Definition at line 8 of file dchost.cc.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 DcHostClass::DcHostClass ( ) [inline]

Definition at line 10 of file dchost.cc.

```
: TclClass("DcHost") {}
```

4.15.3 Member Function Documentation

4.15.3.1 TclObject* DcHostClass::create (int *argc*, const char ∗const ∗ *argv*) [inline]

Definition at line 11 of file dchost.cc.

```
                                                              {
                                  return (new DcHost());
                  }
```

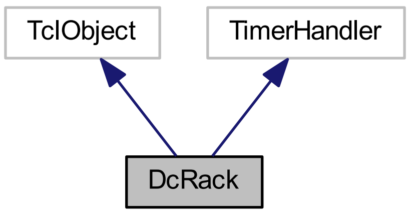
The documentation for this class was generated from the following file:

- dchost.cc

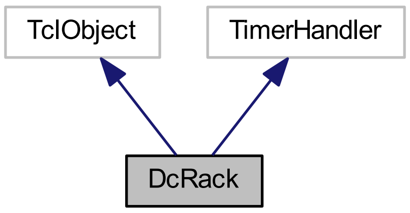
4.16 DcRack Class Reference

`#include <dcrack.h>`

Inheritance diagram for DcRack:



Collaboration diagram for DcRack:



Public Member Functions

- DcRack ()
- virtual ∼DcRack ()
- virtual int command (int argc, const char ∗const ∗argv)

Public Attributes

- int rack_id_
- double stat_interval
- double link_load
- double uplink_B

Protected Member Functions

- void updatestats ()
- virtual void expire (Event ∗e)
- void addHost (DcHost ∗hst)

Protected Attributes

- vector< DcHost ∗ > hosts_list_
- vector< QueueMonitor ∗ > qmon_uplink_list
- int breceived_
- int breceived_old_

4.16.1 Detailed Description

Definition at line 9 of file dcrack.h.

4.16.2 Constructor & Destructor Documentation

4.16.2.1 DcRack::DcRack ( )

Definition at line 15 of file dcrack.cc.

```
               : rack_id_(0), stat_interval (0.0),
      breceived_(0), breceived_old_(0)
{
                bind("rack_id_", &rack_id_);
                bind("breceived_", &breceived_);
                bind("stat_interval", &stat_interval);
                bind("uplink_B", &uplink_B);
}
```

4.16.2.2 DcRack::∼DcRack ( ) [virtual]

Definition at line 23 of file dcrack.cc.

```
{
                hosts_list_.~vector();
                qmon_uplink_list.~vector();
}
```

4.16.3 Member Function Documentation

4.16.3.1 void DcRack::addHost (DcHost ∗ *hst*) [protected]

Definition at line 60 of file dcrack.cc.

```
                        {
        hosts_list_.push_back(hst);
        hst->rack_=this;
}
```

4.16.3.2 int DcRack::command (int *argc,* const char ∗const ∗ *argv*) [virtual]

Definition at line 65 of file dcrack.cc.

```
{
        if (argc == 2) {
                if (strcmp(argv[1], "update-stats") == 0) {
                        updatestats();
                        return (TCL_OK);
                } else if (strcmp(argv[1], "start") == 0) {
                        expire(new Event());
                        return (TCL_OK);
                }
        } else if (argc == 3) {
                if (strcmp(argv[1], "add-dchost") == 0) {
                        DcHost *hst = dynamic_cast<
   DcHost*> (TclObject::lookup(argv[2]));
                        if(hst){
                                addHost(hst);
                                return (TCL_OK);
                        }
                        return (TCL_ERROR);
                }
                else if (strcmp(argv[1], "add-uplink-qmon") == 0) {
                        QueueMonitor *uplinkqmon = dynamic_cast<QueueMonitor*> (
   TclObject::lookup(argv[2]));
                        if(uplinkqmon){
                                qmon_uplink_list.push_back(
   uplinkqmon);

                                return (TCL_OK);
                        }
                        return (TCL_ERROR);
                }
        }
        return (DcRack::command(argc, argv));
}
```

4.16.3.3 void DcRack::expire (Event ∗ *e*) [protected], [virtual]

Definition at line 54 of file dcrack.cc.

```
                        {

        updatestats();
        this->resched(stat_interval);
}
```

4.16.3.4 void DcRack::updatestats ( ) [protected]

Definition at line 29 of file dcrack.cc.

```
29                           {
30 
31                   vector <QueueMonitor*>::iterator iter;
32                   breceived_old_ = breceived_;
33                   breceived_ = 0;
34 //                link_load = 0;
35 
36                   /* Update statistics on the number of bytes received */
37                   for (iter = qmon_uplink_list.begin(); iter!=
      qmon_uplink_list.end(); iter++)
38                   {
39                                   breceived_ += (*iter)->bdepartures_tot();
40                   }
41                   if(breceived_-breceived_old_<0){
42                                   std::cerr << "ERROR in DcRack.cc: Byte counter overflow, consider resetting
       bdepartures_tot in qmon";
43                                   abort();
44                   }
45                   link_load  =(breceived_-breceived_old_)/(
      qmon_uplink_list.size()*stat_interval*uplink_B);
46                   //TODO: fix the problem
47                   link_load *=qmon_uplink_list.size();
48                   if(link_load > 1 ){
49                                   link_load=1;
50                   }
51 //                std:cerr << "Rack " << this->rack_id_ << " link load\t" << link_load <<"\n";
52 }
```

4.16.4 Member Data Documentation

4.16.4.1 int DcRack::breceived_ [protected]

bytes received

Definition at line 32 of file dcrack.h.

4.16.4.2 int DcRack::breceived_old_ [protected]

Definition at line 33 of file dcrack.h.

4.16.4.3 vector<DcHost*> DcRack::hosts_list_ [protected]

list of hosts in a rack

Definition at line 23 of file dcrack.h.

4.16.4.4 double DcRack::link_load

Definition at line 17 of file dcrack.h.

4.16.4.5 vector<QueueMonitor*> DcRack::qmon_uplink_list [protected]

uplink queue monitors

Definition at line 24 of file dcrack.h.

4.16.4.6 int DcRack::rack_id_

Definition at line 15 of file dcrack.h.

4.16.4.7 double DcRack::stat_interval

Definition at line 16 of file dcrack.h.

4.16.4.8 double DcRack::uplink_B

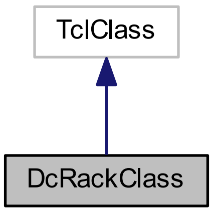
Definition at line 18 of file dcrack.h.

The documentation for this class was generated from the following files:

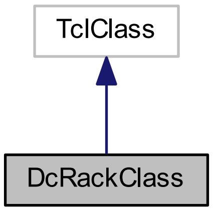
- dcrack.h
- dcrack.cc

4.17 DcRackClass Class Reference

Inheritance diagram for DcRackClass:



Collaboration diagram for DcRackClass:



Public Member Functions

- DcRackClass ()
- TclObject ∗ create (int argc, const char ∗const ∗argv)

4.17.1 Detailed Description

Definition at line 7 of file dcrack.cc.

4.17.2 Constructor & Destructor Documentation

4.17.2.1 DcRackClass::DcRackClass ( ) [inline]

Definition at line 9 of file dcrack.cc.

```
9 : TclClass("DcRack") {}
```

4.17.3 Member Function Documentation

4.17.3.1 TclObject∗ DcRackClass::create (int *argc*, const char ∗const ∗ *argv*) [inline]

Definition at line 10 of file dcrack.cc.

```
                                                   {
                                 return (new DcRack());
                  }
```

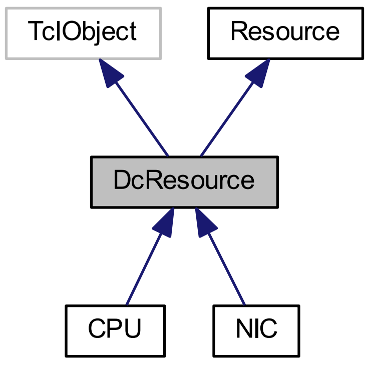
The documentation for this class was generated from the following file:

- dcrack.cc

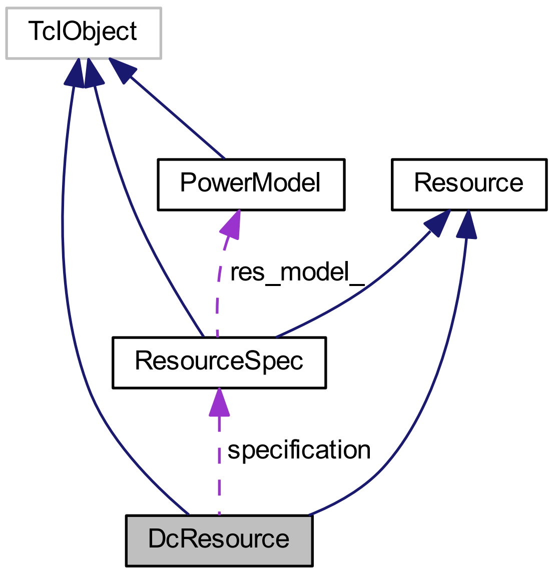
4.18 DcResource Class Reference

`#include <dcresource.h>`

Inheritance diagram for DcResource:



Collaboration diagram for DcResource:



Public Member Functions

- DcResource ()
- virtual ∼DcResource ()
- virtual int command (int argc, const char ∗const ∗argv)
- virtual int setSpecification (ResourceSpec ∗resspec)
- virtual double getUtilization ()
- virtual double getPower ()
- virtual double getMaxPower ()
- virtual void print ()

Public Attributes

- ResourceSpec ∗ specification

Private Attributes

- int used_power_state_
- double total_cap

Additional Inherited Members

4.18.1 Detailed Description

Definition at line 20 of file dcresource.h.

4.18.2 Constructor & Destructor Documentation

4.18.2.1 DcResource::DcResource ( )

Definition at line 19 of file dcresource.cc.

```
19                          {
20
21 }
```

4.18.2.2 DcResource::∼DcResource ( ) [virtual]

Definition at line 23 of file dcresource.cc.

```
23                          {
24
25 }
```

4.18.3 Member Function Documentation

4.18.3.1 int DcResource::command (int *argc*, const char ∗const ∗ *argv*) [virtual]

Reimplemented in CPU.

Definition at line 98 of file dcresource.cc.

```
98                                                   {
99
100
101                 if (argc == 2) {
102                                 if (strcmp(argv[1], "print") == 0) {
103                                                 this->print();
104                                                 return (TCL_OK);
105                                 }
106                 }
107                 return (DcResource::command(argc, argv));
108 }
```

4.18.3.2 double DcResource::getMaxPower ( ) [virtual]

Definition at line 68 of file dcresource.cc.

```
68                               {
69                int n = 1;
70                double* utilization = new double[n];
71                utilization[0] = 1;
72                double result =  specification->getPowerModel()->
     estimate(n,utilization);
73                delete utilization;
74                return result;
75 }
```

4.18.3.3 double DcResource::getPower ( ) [virtual]

Definition at line 58 of file dcresource.cc.

```
                        {
                int n = 1;
                double* utilization = new double[n];
                utilization[0] = getUtilization();
                double result =  specification->getPowerModel()->
    estimate(n,utilization);
                delete utilization;
                return result;
}
```

4.18.3.4 double DcResource::getUtilization ( ) [virtual]

Reimplemented in CPU, and NIC.

Definition at line 46 of file dcresource.cc.

```
                              {
                double free = 0;

                std::vector<Capacity>::iterator iter;
                for(iter=capacity.begin(); iter!= capacity.end();iter++){
                                free += iter->getValueRecursive();

                }
                return 1-(free/total_cap);
}
```

4.18.3.5 void DcResource::print ( ) [virtual]

Reimplemented in CPU.

Definition at line 81 of file dcresource.cc.

```
                   {
                std::cout << "DcResource";
                std::cout << "\n";
                specification->print();
                std::cout << "Available capacities:\t";
                std::vector <Capacity>::iterator iter;
                for  (iter = capacity.begin(); iter!=capacity.end(); iter++)
                {
                                std::cout << (*iter) << ",";
                }
                std::cout << "\n";
                std::cout << "Used power state:\t" << used_power_state_;

                std::cout << "\n";
}
```

4.18.3.6 int DcResource::setSpecification (ResourceSpec * *resspec*) [virtual]

Reimplemented in CPU.

Definition at line 28 of file dcresource.cc.

```
                                                {
                if(resspec==NULL){
                                std::cerr << "ERROR: Null pointer passed as ResourceSpec.";
                                return 1;
                }
                specification=resspec;
                capacity = resspec->capacity;
                used_power_state_ = specification->
    power_states.at(0);
                type=resspec->type;
                arch=resspec->arch;
                total_cap = 0;
                std::vector<Capacity>::iterator iter;
                for(iter = resspec->capacity.begin();iter!= resspec->
    capacity.end();iter++){
                                total_cap += iter->value;
                }
                return 0;
}
```

4.18.4 Member Data Documentation

4.18.4.1 ResourceSpec∗ DcResource::specification

Definition at line 34 of file dcresource.h.

4.18.4.2 double DcResource::total_cap [private]

Definition at line 37 of file dcresource.h.

4.18.4.3 int DcResource::used_power_state_ [private]

Used power state

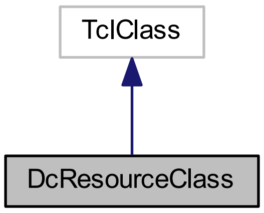
Definition at line 36 of file dcresource.h.

The documentation for this class was generated from the following files:

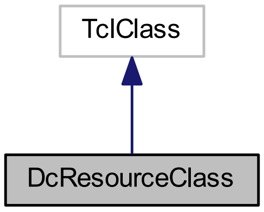
- dcresource.h
- dcresource.cc

4.19 DcResourceClass Class Reference

Inheritance diagram for DcResourceClass:



Collaboration diagram for DcResourceClass:



Public Member Functions

- DcResourceClass ()
- TclObject ∗ create (int argc, const char ∗const ∗argv)

4.19.1 Detailed Description

Definition at line 10 of file dcresource.cc.

4.19.2 Constructor & Destructor Documentation

4.19.2.1 DcResourceClass::DcResourceClass ( ) [inline]

Definition at line 12 of file dcresource.cc.

```
: TclClass("DcResource") {}
```

4.19.3 Member Function Documentation

4.19.3.1 TclObject∗ DcResourceClass::create (int *argc*, const char ∗const ∗ *argv*) [inline]

Definition at line 13 of file dcresource.cc.

```
                                                   {
                                return (new DcResource());
                }
```

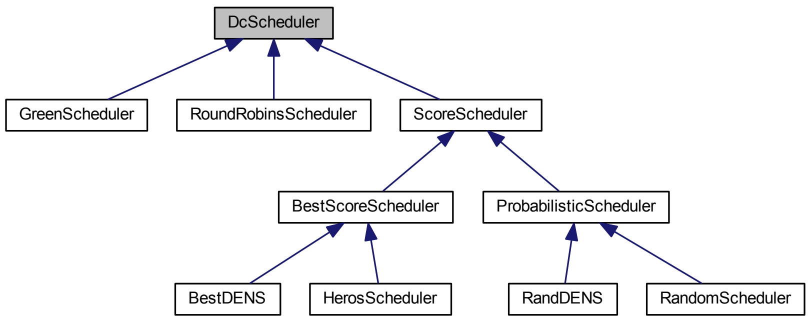
The documentation for this class was generated from the following file:

- dcresource.cc

4.20 DcScheduler Class Reference

`#include <dcscheduler.h>`

Inheritance diagram for DcScheduler:



Public Member Functions

- DcScheduler ()
- virtual ∼DcScheduler ()
- virtual TskComAgent ∗ scheduleTask (CloudTask ∗task, std::vector< ResourceProvider ∗ > providers)=0

4.20.1 Detailed Description

Definition at line 15 of file dcscheduler.h.

4.20.2 Constructor & Destructor Documentation

4.20.2.1 DcScheduler::DcScheduler ( )

Definition at line 10 of file dcscheduler.cc.

```
10                              {
11 
12 
13 }
```

4.20.2.2 DcScheduler::∼DcScheduler ( ) [virtual]

Definition at line 15 of file dcscheduler.cc.

```
15                               {
16 
17 }
```

4.20.3 Member Function Documentation

4.20.3.1 virtual TskComAgent∗ DcScheduler::scheduleTask (CloudTask ∗ *task,* std::vector< ResourceProvider ∗ > *providers*) [pure virtual]

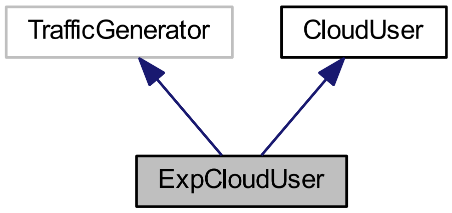
Implemented in HerosScheduler, BestDENS, RandDENS, ProbabilisticScheduler, RandomScheduler, BestScore←Scheduler, GreenScheduler, and RoundRobinsScheduler.

The documentation for this class was generated from the following files:

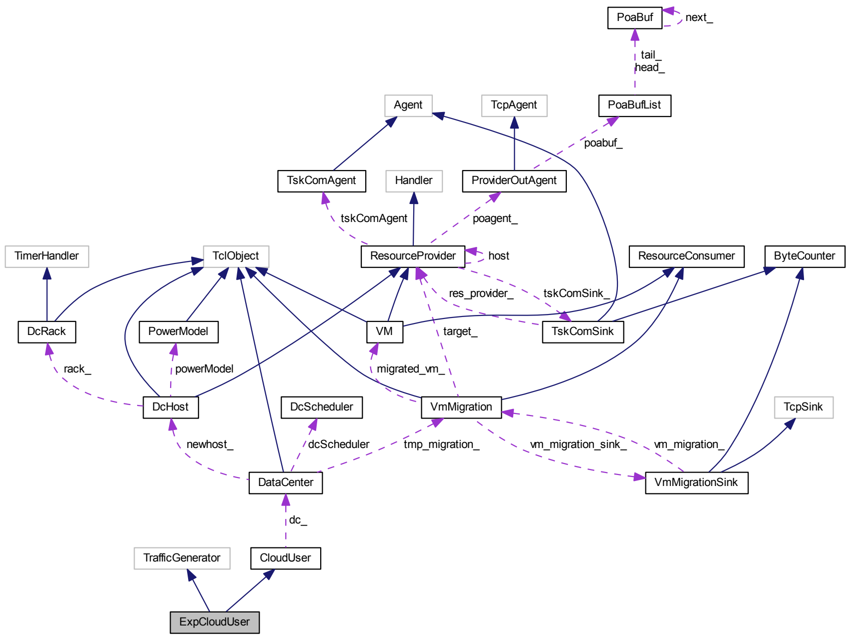
- dcscheduler.h
- dcscheduler.cc

4.21 ExpCloudUser Class Reference

Inheritance diagram for ExpCloudUser:



Collaboration diagram for ExpCloudUser:



Public Member Functions

- ExpCloudUser ()
- virtual double next_interval (int &)
- virtual void timeout ()
- int command (int argc, const char ∗const ∗argv)
- void addDataCenterPointer (DataCenter ∗joindc_)

Protected Member Functions

- void init ()

Protected Attributes

- double ontime_
- double offtime_
- double rate_
- double interval_
- unsigned int rem_
- ExponentialRandomVariable burstlen_
- ExponentialRandomVariable Offtime_

Additional Inherited Members

4.21.1 Detailed Description

Definition at line 20 of file expclouduser.cc.

4.21.2 Constructor & Destructor Documentation

4.21.2.1 ExpCloudUser::ExpCloudUser ( )

Definition at line 84 of file expclouduser.cc.

```
84                              : burstlen_(0.0), Offtime_(0.0)
85 {
86
87                 bind_time("random_tskmips_",random_tskmips_.avgp());
88                 bind_time("burst_time_", &ontime_);
89                 bind_time("idle_time_", Offtime_.avgp());
90                 bind_bw("rate_", &rate_);
91                 bind("packetSize_", &size_);
92
93                 // Bind CloudUser variables
94                 bind("id_", &id_);
95                 bind("tskmips_", &tskmips_);
96                 bind("memory_", &memory_);
97                 bind("storage_", &storage_);
98                 bind("tsksize_", &tsksize_);
99                 bind("tskmaxduration_", &tskmaxduration_);
100                 bind("toutputsize_", &toutputsize_);
101                 bind("tintercom_", &tintercom_);
102                 bind("mean_response_time_", &mean_response_time_);
103                 bind("sd_response_time_", &sd_response_time_);
104                 bind("unfinished_tasks_", &unfinished_tasks_);
105 }
```

4.21.3 Member Function Documentation

4.21.3.1 void ExpCloudUser::addDataCenterPointer (DataCenter ∗ *joindc_*)

4.21.3.2 int ExpCloudUser::command (int *argc,* const char ∗const ∗ *argv*)

Definition at line 57 of file expclouduser.cc.

```
                                                            {
                int result = CloudUser::process_command(argc,argv);
                if(result==-1){
                if(argc==3){
                                if (strcmp(argv[1], "use-rng") == 0) {
                                                burstlen_.seed((char *)argv[2]);
                                                Offtime_.seed((char *)argv[2]);
                                                return (TCL_OK);
                                }
                                //ADDED CODE
                                else if (strcmp(argv[1], "set-rate") == 0) {
                                                int new_rate = atoi(argv[2]);
                                                if(1){
                                                                rate_=new_rate;
                                                                interval_ = (double)(size_ << 3)/(
      double)rate_;
                                                                return (TCL_OK);
                                                }
                                                return (TCL_ERROR);
                                }
                                //ADDED CODE
                }
                return Application::command(argc,argv);
                } else {
                                return result;
                }
}
```

4.21.3.3 void ExpCloudUser::init ( ) [protected]

Definition at line 107 of file expclouduser.cc.

```
{
                /* compute inter-packet interval during bursts based on
                 * packet size and burst rate.  then compute average number
                 * of packets in a burst.
                 */
                interval_ = (double)(size_ << 3)/(double)rate_;
                burstlen_.setavg(ontime_/interval_);
                rem_ = 0;
}
```

4.21.3.4 double ExpCloudUser::next_interval (int & *size*) [virtual]

Definition at line 118 of file expclouduser.cc.

```
{
                double t = interval_;

                if (rem_ == 0) {
                                /* compute number of packets in next burst */
                                rem_ = int(burstlen_.value() + .5);
                                /* make sure we got at least 1 */
                                if (rem_ == 0)
                                                rem_ = 1;
                                /* start of an idle period, compute idle time */
                                t += Offtime_.value();
                }
                rem_--;

                size = size_;
                //TODO: add change factor (for decay the change should be >1):interval_ = interval_ *
       change_ ;
                // OR: use change by constant term as below:
                //              interval_ = interval_ + 0.000005;
                //   if(interval_<=0) interval_ = 0.000005;
                return(t);
}
```

4.21.3.5 void ExpCloudUser::timeout ( ) [virtual]

Definition at line 141 of file expclouduser.cc.

```
{
                if (! running_)
                                return;

                if (nextPkttime_ != interval_ || nextPkttime_ == -1){
                                dc_->receivedTsk(size_, createTask(), "NEW_BURST");
                }
                else {
                                dc_->receivedTsk(size_,
      createTask());
                }


                /* figure out when to send the next one */
                nextPkttime_ = next_interval(size_);
                /* schedule it */
                if (nextPkttime_ > 0)
                                timer_.resched(nextPkttime_);
}
```

4.21.4 Member Data Documentation

4.21.4.1 ExponentialRandomVariable ExpCloudUser::burstlen_ [protected]

Definition at line 39 of file expclouduser.cc.

4.21.4.2 double ExpCloudUser::interval_ [protected]

packet inter-arrival time during burst (sec)

Definition at line 35 of file expclouduser.cc.

4.21.4.3 double ExpCloudUser::offtime_ [protected]

average length of idle time (sec)

Definition at line 33 of file expclouduser.cc.

4.21.4.4 ExponentialRandomVariable ExpCloudUser::Offtime_ [protected]

Definition at line 40 of file expclouduser.cc.

4.21.4.5 double ExpCloudUser::ontime_ [protected]

average length of burst (sec)

Definition at line 32 of file expclouduser.cc.

4.21.4.6 double ExpCloudUser::rate_ [protected]

send rate during on time (bps)

Definition at line 34 of file expclouduser.cc.

4.21.4.7 unsigned int ExpCloudUser::rem_ [protected]

number of packets left in current burst

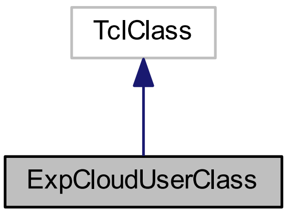
Definition at line 36 of file expclouduser.cc.

The documentation for this class was generated from the following file:

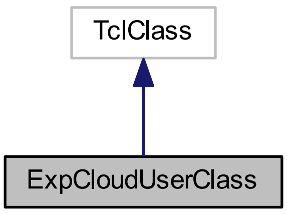
- expclouduser.cc

4.22 ExpCloudUserClass Class Reference

Inheritance diagram for ExpCloudUserClass:



Collaboration diagram for ExpCloudUserClass:



Public Member Functions

- ExpCloudUserClass ()
- TclObject ∗ create (int, const char ∗const ∗)

4.22.1 Detailed Description

Definition at line 44 of file expclouduser.cc.

4.22.2 Constructor & Destructor Documentation

4.22.2.1 ExpCloudUserClass::ExpCloudUserClass ( ) [inline]

Definition at line 46 of file expclouduser.cc.

```
: TclClass("Application/Traffic/ExpCloudUser") {}
```

4.22.3 Member Function Documentation

4.22.3.1 TclObject∗ ExpCloudUserClass::create (int , const char ∗const ∗) [inline]

Definition at line 47 of file expclouduser.cc.

```
                                                {
                                return (new ExpCloudUser());
                }
```

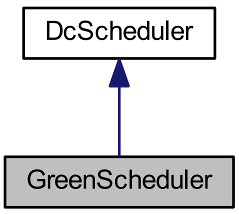
The documentation for this class was generated from the following file:

- expclouduser.cc

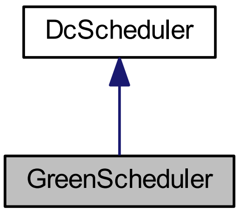
4.23 GreenScheduler Class Reference

`#include <greenscheduler.h>`

Inheritance diagram for GreenScheduler:



Collaboration diagram for GreenScheduler:



Public Member Functions

- GreenScheduler ()
- virtual ∼GreenScheduler ()
- virtual TskComAgent ∗ scheduleTask (CloudTask ∗task, std::vector< ResourceProvider ∗ > providers)

4.23.1 Detailed Description

Definition at line 13 of file greenscheduler.h.

4.23.2 Constructor & Destructor Documentation

4.23.2.1 GreenScheduler::GreenScheduler ( )

Definition at line 11 of file greenscheduler.cc.

```
                                    {


 }
```

4.23.2.2 GreenScheduler::∼GreenScheduler ( ) [virtual]

Definition at line 16 of file greenscheduler.cc.

```
                                     {

 }
```

4.23.3 Member Function Documentation

4.23.3.1 TskComAgent ∗ GreenScheduler::scheduleTask (CloudTask ∗ *task,* std::vector< ResourceProvider ∗ > *providers*) [virtual]

Implements DcScheduler.

Definition at line 20 of file greenscheduler.cc.

```
                                                                                    {
            vector <ResourceProvider*>::iterator iter;

            for (iter = providers.begin(); iter!=providers.end(); iter++)
            {
                        if ((*iter)->trySchedulingTsk(task))
                                    return (*iter)->getTskComAgent();
            }
            return NULL;
}
```

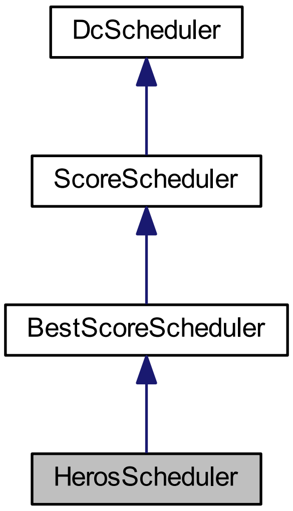
The documentation for this class was generated from the following files:

- greenscheduler.h
- greenscheduler.cc

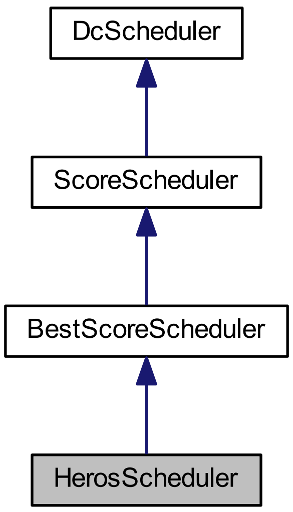
4.24 HerosScheduler Class Reference

```
#include <herosscheduler.h>
```

Inheritance diagram for HerosScheduler:



Collaboration diagram for HerosScheduler:



Public Member Functions

- HerosScheduler ()
- virtual ∼HerosScheduler ()
- virtual TskComAgent ∗ scheduleTask (CloudTask ∗task, std::vector< ResourceProvider ∗ > providers)

Static Public Member Functions

- static double performancePerWattMax (ResourceProvider ∗rp)

Private Member Functions

- virtual double calculateScore (ResourceProvider ∗rp)

Static Private Member Functions

- static double densLoadFactor (double load, double epsilon)
- static double linkLoadFactor (double load)
- static double performancePerWatt (ResourceProvider ∗rp)
- static double herosTransformation (ResourceProvider ∗rp, double alpha, double beta, double gamma)

Private Attributes

- double epsilon

4.24.1 Detailed Description

Definition at line 24 of file herosscheduler.h.

4.24.2 Constructor & Destructor Documentation

4.24.2.1 HerosScheduler::HerosScheduler ( )

Definition at line 11 of file herosscheduler.cc.

```
                               : epsilon(0.1){


}
```

4.24.2.2 HerosScheduler::∼HerosScheduler ( ) [virtual]

Definition at line 16 of file herosscheduler.cc.

```
                                {

}
```

4.24.3 Member Function Documentation

4.24.3.1 double HerosScheduler::calculateScore (ResourceProvider ∗ *rp*) [private], [virtual]

Implements BestScoreScheduler.

Definition at line 21 of file herosscheduler.cc.

```
                                                         {
                double result = 0;

                result = herosTransformation(rp,110,0.90,1.2);
//              std::cerr << "Heros transformation result" << result << "\n";
                result *= pow(linkLoadFactor(rp->getRootHost()->
     rack_->link_load),2);
//              std::cerr << "Final result" << result << "\n";
                return result;
}
```

4.24.3.2 double HerosScheduler::densLoadFactor (double *load,* double *epsilon*) [static], [private]

Definition at line 32 of file herosscheduler.cc.

```
                                                                {
                return 1/(1+exp(-10*(load-0.5))) - 1/(1+exp((-10/epsilon)*(load-(1-(
     epsilon/2)))));
}
```

4.24.3.3 double HerosScheduler::herosTransformation (ResourceProvider * *rp*, double *alpha*, double *beta*, double *gamma*) [static], [private]

Definition at line 51 of file herosscheduler.cc.

```
51                                                                                                    {
52
53               double maxl = rp->getTotalCap(Computing);
54               //             std::cerr << "Ppw current: " <<performancePerWatt(rp) << "\n";
55               double result =  performancePerWatt(rp);
56                                                     if(rp->getResTypeUtil(
     Computing) > beta/2){
57                                                     result -= gamma *
     performancePerWatt(rp) *
58                                                     1/
59                                                     (
60                                                                                1+exp
61                                                                                (
62
     -(alpha/maxl)*(rp->getResTypeUtil(Computing)*maxl - (beta * maxl ) )
63                                                                                )
64                                                     );
65                                                     }
66
67               return result;
68 }
```

4.24.3.4 double HerosScheduler::linkLoadFactor (double *load*) [static], [private]

Definition at line 70 of file herosscheduler.cc.

```
70                                                     {
71               return exp(-(pow(2*load,2)));
72 }
```

4.24.3.5 double HerosScheduler::performancePerWatt (ResourceProvider * *rp*) [static], [private]

Definition at line 36 of file herosscheduler.cc.

```
36                                                                {
37               if(rp->getRootHost()->eCurrentConsumption_==0)
38               {
39                              return 0;
40               }
41               return (rp->getResTypeUtil(Computing)) * rp->
     getTotalCap(Computing) / rp->getRootHost()->
     eCurrentConsumption_ ;
42 }
```

4.24.3.6 double HerosScheduler::performancePerWattMax (ResourceProvider * *rp*) [static]

Definition at line 44 of file herosscheduler.cc.

```
44                                                                   {
45
46               double result =  rp->getTotalCap(Computing) / rp->
     getRootHost()->powerModel->getMaxPower();
47               //             std::cerr << "Ppw max: " <<result << "\n";
48               return result;
49 }
```

4.24.3.7 TskComAgent ∗ HerosScheduler::scheduleTask (CloudTask ∗ *task,* std::vector< ResourceProvider ∗ > *providers*) [virtual]

Reimplemented from BestScoreScheduler.

Definition at line 76 of file herosscheduler.cc.

```
                                                                                                    {
                //              std::cerr<< "HEROS is making decision:\n";
                vector<ProviderScore> scored_providers_;
                vector <ResourceProvider*>::iterator iter;
                for (iter = providers.begin(); iter!=providers.end(); iter++)
                {
                                if ((*iter)->testSchedulingPossibility(task)){
                                                scored_providers_.push_back(
    ProviderScore((*iter),calculateScore((*iter)),
    linkLoadFactor((*iter)->getRootHost()->rack_->link_load)));
                                }
                }
                if(scored_providers_.empty()){
                                return NULL;
                } else {
                                vector <ProviderScore>::iterator sp;
                                sort(scored_providers_.begin(),scored_providers_.end(),
    herosComparator);
                                vector<ProviderScore>::reverse_iterator rsp = scored_providers_.rbegin();
                                ProviderScore best = *rsp;
                                int max_n = 0;
                                for (; rsp != scored_providers_.rend(); rsp++ ) {
                                                if(!herosComparator((*rsp),best)){
                                                                ++max_n;
                                                } else {
                                                                break;
                                                }
                                 }
                                 if(max_n!=1){
                                 int selected = (double)rand() / (double)RAND_MAX * max_n +1;
                                                 best = scored_providers_.at(scored_providers_.size()-
    selected);
                                 }

                                 scored_providers_.clear();
//                               std::cerr<< "Selected prov: " << best.provider_->id_ << "\tScore:" <<
     best.score_ <<"\tSelected task:" << task->id_ << "\n";

                                 return best.provider_->getTskComAgent();
                 }
}
```

4.24.4 Member Data Documentation

4.24.4.1 double HerosScheduler::epsilon [private]

Definition at line 38 of file herosscheduler.h.

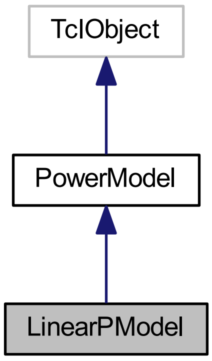
The documentation for this class was generated from the following files:

- herosscheduler.h
- herosscheduler.cc

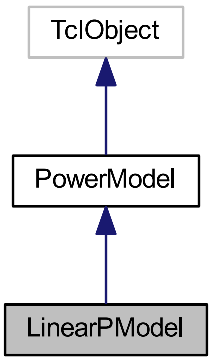
4.25 LinearPModel Class Reference

`#include <linearpmodel.h>`

Inheritance diagram for LinearPModel:



Collaboration diagram for LinearPModel:



Public Member Functions

- LinearPModel ()
- virtual ∼LinearPModel ()
- virtual int command (int argc, const char ∗const ∗argv)
- virtual double estimate (int size, double ∗predictors)
- virtual double getMaxPower ()
- virtual void addComponent (DcResource ∗component)
- virtual void print ()

Private Member Functions

- void setCoefficient (const char ∗coef, double value)
- void setCoefficientNumeric (const char ∗coef, double value)
- void updateInit ()
- void setCoefNumber (int number)

Private Attributes

- double ∗ coefficients
- bool ∗ initialized
- bool ready
- int coef_number

Additional Inherited Members

4.25.1 Detailed Description

Definition at line 13 of file linearpmodel.h.

4.25.2 Constructor & Destructor Documentation

4.25.2.1 LinearPModel::LinearPModel ( )

Definition at line 20 of file linearpmodel.cc.

```
                              {

                  /*LastResType+2 = +1 for elements number, +1 for the intercept:*/
                  coefficients = new double[LastResType+2];
                  initialized = new bool[LastResType+2];
                  setCoefNumber(LastResType+2);
                  for(int i = 0; i < LastResType+2; i++){
                                  coefficients[i]= 0;
                                  initialized[i]=false;
                  }

 }
```

4.25.2.2 LinearPModel::∼LinearPModel ( ) [virtual]

Definition at line 33 of file linearpmodel.cc.

```
                               {

                  delete[] coefficients;
                  delete[] initialized;
                  name_.clear();
 }
```

4.25.3 Member Function Documentation

4.25.3.1 void LinearPModel::addComponent (DcResource ∗ *component*) [virtual]

Implements PowerModel.

Definition at line 52 of file linearpmodel.cc.

```
                                                  {
                /* Linear power model does not accept components*/
                return;
}
```

4.25.3.2 int LinearPModel::command (int *argc*, const char ∗const ∗ *argv*) [virtual]

Reimplemented from PowerModel.

Definition at line 128 of file linearpmodel.cc.

```
{

                if (argc == 2) {
                                if (strcmp(argv[1], "print") == 0) {
                                                /* print general info */
                                                print();
                                                return (TCL_OK);
                                }
                } else if (argc==3){
                                if (strcmp(argv[1], "set-name") == 0) {
                                                setName(argv[2]);
                                                return(TCL_OK);
                                } else if (strcmp(argv[1], "set-coef-number") == 0) {
                                                setCoefNumber(atoi(argv[2]));
                                                return(TCL_OK);
                                } else {
                                                return(TCL_ERROR);
                                }
                }
                else if (argc == 4) {
                                if (strcmp(argv[1], "set-coefficient") == 0) {
                                                setCoefficient(argv[2],atof(argv[3]));
                                                return(TCL_OK);
                                } else if (strcmp(argv[1], "set-coefficient-numeric") == 0) {
                                                setCoefficientNumeric(argv[2],atof(
      argv[3]));
                                                return(TCL_OK);
                                } else {
                                                return(TCL_ERROR);
                                }
                }
                return (TCL_ERROR);
}
```

4.25.3.3 double LinearPModel::estimate (int *size*, double ∗ *predictors*) [virtual]

Implements PowerModel.

Definition at line 57 of file linearpmodel.cc.

```
                                                          {
                if(ready){
                                if(size!=  coef_number - 1){
                                                std::cerr <<"Incorrect size of predictors array!\n";
                                }
                                double result = coefficients[size];
                                for(int i = 0; i < size; i++){
                                                result += predictors[i] *
      coefficients[i];
                                }
                                return result;
                } else {
                                std::cerr << "The model is not correctly initalized.\n" ;
                                print();
                                std::cerr << "Aborting simulation";
                                abort();
                }
}
```

4.25.3.4 double LinearPModel::getMaxPower ( ) [virtual]

Implements PowerModel.

Definition at line 75 of file linearpmodel.cc.

```
75                              {
76                double * load = new double[coef_number];
77                for(int i = 0; i < coef_number; i++){
78                                load[i] = 1;
79                }
80                return estimate(coef_number,load);
81 }
```

4.25.3.5 void LinearPModel::print ( ) [virtual]

Reimplemented from PowerModel.

Definition at line 85 of file linearpmodel.cc.

```
85                        {
86                std::cout << "Linear model: "<< name_ << "\n";
87                if(ready){
88                                std::cout << "Coefficients:\n";
89                                for(int i = 0; i < coef_number ; i++){
90                                                std::cout << i << ": " <<
      coefficients[i] << "\n";
91                                }
92                } else {
93                                std::cout << "Model not initalized properly\n";
94                }
95 }
```

4.25.3.6 void LinearPModel::setCoefficient (const char ∗ *coef,* double *value*) [private]

Definition at line 97 of file linearpmodel.cc.

```
97                                                        {
98                if(strcmp(coef, "Intercept") != 0){
99                                res_type type = Resource::translateType(coef
      );
100                               coefficients[type]=value;
101                               initialized[type]=true;
102               } else {
103                               coefficients[coef_number-1]= value;
104                               initialized[coef_number-1]=true;
105               }
106               updateInit();
107 }
```

4.25.3.7 void LinearPModel::setCoefficientNumeric (const char ∗ *coef,* double *value*) [private]

Definition at line 109 of file linearpmodel.cc.

```
109                                                               {
110               if(strcmp(coef, "Intercept") != 0){
111                               int i = atoi(coef);
112                               coefficients[i]=value;
113                               initialized[i]=true;
114               } else {
115                               coefficients[coef_number-1]= value;
116                               initialized[coef_number-1]=true;
117               }
118               updateInit();
119 }
```

4.25.3.8 void LinearPModel::setCoefNumber (int *number*) [private]

Definition at line 40 of file linearpmodel.cc.

```
                                        {
 
               coef_number = number;
               if(coefficients != NULL){
               delete[] coefficients;
               delete[] initialized;
               }
               coefficients = new double[coef_number];
               initialized = new bool[coef_number];
 }
```

4.25.3.9 void LinearPModel::updateInit ( ) [private]

Definition at line 120 of file linearpmodel.cc.

```
                          {
               bool result = true;
               for(int i = 0; i < coef_number-1; i++){
                              result = result && initialized[i];
               }
               ready = result;
 }
```

4.25.4 Member Data Documentation

4.25.4.1 int LinearPModel::coef_number [private]

Definition at line 29 of file linearpmodel.h.

4.25.4.2 double∗ LinearPModel::coefficients [private]

Slopes as the resource types. The last element is the intercept

Definition at line 23 of file linearpmodel.h.

4.25.4.3 bool∗ LinearPModel::initialized [private]

Initialization flag

Definition at line 24 of file linearpmodel.h.

4.25.4.4 bool LinearPModel::ready [private]

Model initialized and ready for usage

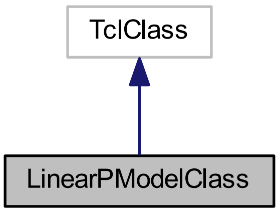
Definition at line 25 of file linearpmodel.h.

The documentation for this class was generated from the following files:

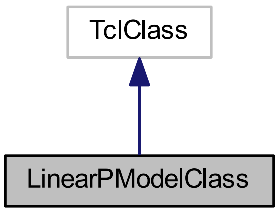
- linearpmodel.h
- linearpmodel.cc

4.26 LinearPModelClass Class Reference

Inheritance diagram for LinearPModelClass:



Collaboration diagram for LinearPModelClass:



Public Member Functions

- LinearPModelClass ()
- TclObject ∗ create (int argc, const char ∗const ∗argv)

4.26.1 Detailed Description

Definition at line 11 of file linearpmodel.cc.

4.26.2 Constructor & Destructor Documentation

4.26.2.1 LinearPModelClass::LinearPModelClass ( ) [inline]

Definition at line 13 of file linearpmodel.cc.

```
13 : TclClass("LinearPModel") {}
```

4.26.3 Member Function Documentation

4.26.3.1 TclObject* LinearPModelClass::create (int *argc*, const char *const * *argv*) [inline]

Definition at line 14 of file linearpmodel.cc.

```
                                                        {
                                  return (new LinearPModel());
                    }
```

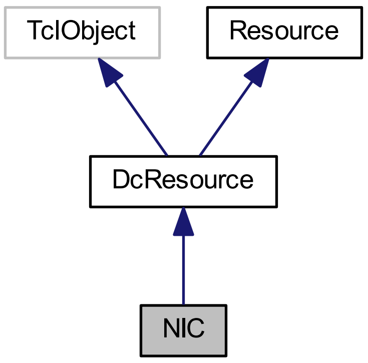
The documentation for this class was generated from the following file:

- linearpmodel.cc

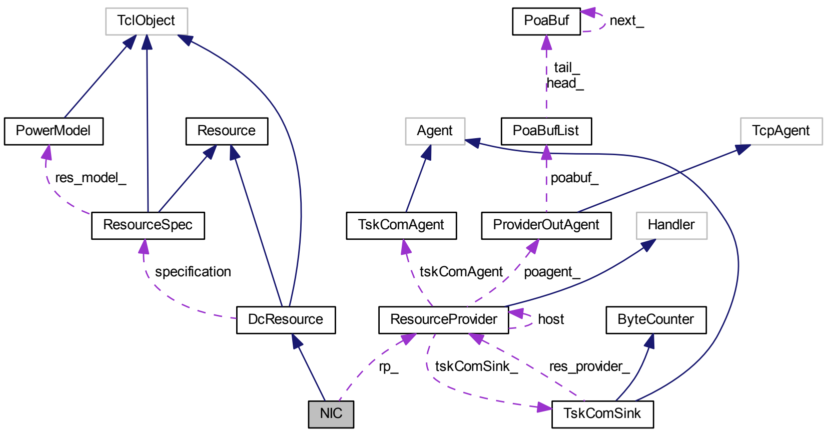
4.27 NIC Class Reference

`#include <nic.h>`

Inheritance diagram for NIC:



Collaboration diagram for NIC:



Public Member Functions

- NIC ()
- virtual ~NIC ()
- void setRp (ResourceProvider *rp)
- virtual double getUtilization ()

Private Attributes

- ResourceProvider * rp_

Additional Inherited Members

4.27.1 Detailed Description

Definition at line 14 of file nic.h.

4.27.2 Constructor & Destructor Documentation

4.27.2.1 NIC::NIC ( )

Definition at line 19 of file nic.cc.

```
          {


}
```

4.27.2.2 NIC::~NIC ( ) [virtual]

Definition at line 24 of file nic.cc.

```
           {

}
```

4.27.3 Member Function Documentation

4.27.3.1 double NIC::getUtilization ( ) [virtual]

Reimplemented from DcResource.

Definition at line 32 of file nic.cc.

```
                          {
               return rp_->getResTypeUtil(Networking);
}
```

4.27.3.2 void NIC::setRp (ResourceProvider ∗ *rp*)

Definition at line 28 of file nic.cc.

```
                             {
                 rp_ = rp;
 }
```

4.27.4 Member Data Documentation

4.27.4.1 ResourceProvider∗ NIC::rp_ [private]

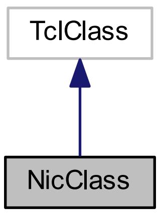
Definition at line 21 of file nic.h.

The documentation for this class was generated from the following files:

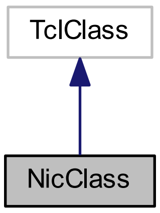
- nic.h
- nic.cc

4.28 NicClass Class Reference

Inheritance diagram for NicClass:



Collaboration diagram for NicClass:



Public Member Functions

- NicClass ()
- TclObject ∗ create (int argc, const char ∗const ∗argv)

4.28.1 Detailed Description

Definition at line 11 of file nic.cc.

4.28.2 Constructor & Destructor Documentation

4.28.2.1 NicClass::NicClass ( ) [inline]

Definition at line 13 of file nic.cc.

```
: TclClass("NIC") {}
```

4.28.3 Member Function Documentation

4.28.3.1 TclObject∗ NicClass::create (int *argc*, const char ∗const ∗ *argv*) [inline]

Definition at line 14 of file nic.cc.

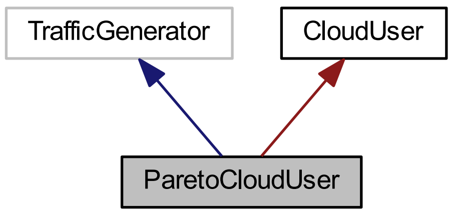
```
                                              {
                          return (new NIC());
            }
```

The documentation for this class was generated from the following file:

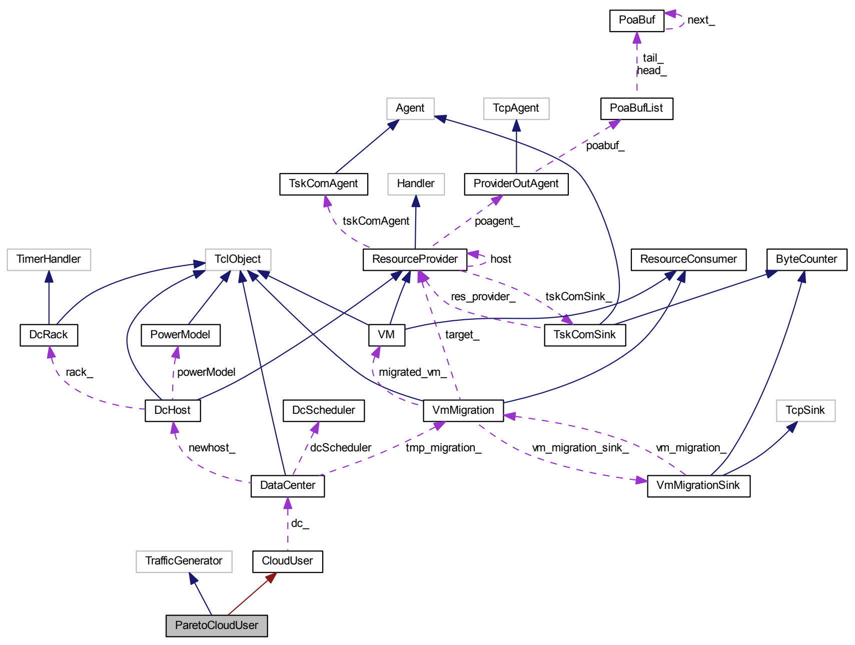
- nic.cc

4.29 ParetoCloudUser Class Reference

Inheritance diagram for ParetoCloudUser:



Collaboration diagram for ParetoCloudUser:



Public Member Functions

- ParetoCloudUser ()
- virtual double next_interval (int &)
- virtual void timeout ()
- int on ()
- int command (int argc, const char ∗const ∗argv)

Protected Member Functions

- void init ()

Protected Attributes

- double ontime_
- double offtime_
- double rate_
- double interval_
- double burstlen_
- double shape_
- unsigned int rem_
- double p1_
- double p2_
- int on_
- RNG ∗ rng_

Additional Inherited Members

4.29.1 Detailed Description

Definition at line 14 of file paretoclouduser.cc.

4.29.2 Constructor & Destructor Documentation

4.29.2.1 ParetoCloudUser::ParetoCloudUser ( )

Definition at line 83 of file paretoclouduser.cc.

```
83                                   : rng_(NULL)
84 {
85                 bind_time("burst_time_", &ontime_);
86                 bind_time("idle_time_", &offtime_);
87                 bind_bw("rate_", &rate_);
88                 bind("shape_", &shape_);
89                 bind("packetSize_", &size_);
90 }
```

4.29.3 Member Function Documentation

4.29.3.1 int ParetoCloudUser::command (int *argc*, const char ∗const ∗ *argv*)

Definition at line 57 of file paretoclouduser.cc.

```
57                                                          {
58
59                 Tcl& tcl = Tcl::instance();
60                 if(argc==3){
61                                 if (strcmp(argv[1], "use-rng") == 0) {
62                                                 rng_ = (RNG*)TclObject::lookup(argv[2]);
63                                                 if (rng_ == 0) {
64                                                                 tcl.resultf("no such RNG %s", argv[2]);
65                                                                 return(TCL_ERROR);
66                                                 }
67                                                 return (TCL_OK);
68                                 }
69                                 //ADDED CODE
70                                 else if (strcmp(argv[1], "join-datacenter") == 0) {
71                                                 DataCenter *dc = dynamic_cast<
      DataCenter*> (TclObject::lookup(argv[2]));
72                                                 if(dc){
73                                                                 dc_ = dc;
74                                                                 return (TCL_OK);
75                                                 }
76                                                 return (TCL_ERROR);
77                                 }
78                                 //ADDED CODE
79                 }
80                 return Application::command(argc,argv);
81 }
```

4.29.3.2 void ParetoCloudUser::init ( ) [protected]

Definition at line 92 of file paretoclouduser.cc.

```
93 {
94                 interval_ = (double)(size_ << 3)/(double)rate_;
95                 burstlen_ = ontime_/interval_;
96                 rem_ = 0;
97                 on_ = 0;
98                 p1_ = burstlen_ * (shape_ - 1.0)/shape_;
99                 p2_ = offtime_ * (shape_ - 1.0)/shape_;
100 }
```

4.29.3.3 double ParetoCloudUser::next_interval (int & *size*) [virtual]

Definition at line 102 of file paretoclouduser.cc.

```
{

                double t = interval_;

                on_ = 1;
                if (rem_ == 0) {
                                /* compute number of packets in next burst */
                                if(rng_ == 0){
                                                rem_ = int(Random::pareto(p1_, shape_) + .5);
                                }
                                else{
                                                // Added by Debojyoti Dutta 13th October 2000
                                                rem_ = int(rng_->pareto(p1_, shape_) + .5);
                                }
                                /* make sure we got at least 1 */
                                if (rem_ == 0)
                                                rem_ = 1;
                                /* start of an idle period, compute idle time */
                                if(rng_ == 0){
                                                t += Random::pareto(p2_, shape_);
                                }
                                else{
                                                // Added by Debojyoti Dutta 13th October 2000
                                                t += rng_->pareto(p2_, shape_);
                                }
                                on_ = 0;
                }
                rem_--;

                size = size_;
                return(t);

}
```

4.29.3.4 int ParetoCloudUser::on ( ) [inline]

Definition at line 19 of file paretoclouduser.cc.

```
{ return on_ ; }
```

4.29.3.5 void ParetoCloudUser::timeout ( ) [virtual]

Definition at line 137 of file paretoclouduser.cc.

```
{
                if (! running_)
                                return;

                /* send a packet */
                dc_->receivedTsk(size_, createTask());
                /* figure out when to send the next one */
                nextPkttime_ = next_interval(size_);
                /* schedule it */
                if (nextPkttime_ > 0)
                                timer_.resched(nextPkttime_);
                else
                                running_ = 0;
}
```

4.29.4 Member Data Documentation

4.29.4.1 double ParetoCloudUser::burstlen_ [protected]

Definition at line 28 of file paretoclouduser.cc.

4.29.4.2 double ParetoCloudUser::interval_ [protected]

Definition at line 27 of file paretoclouduser.cc.

4.29.4.3 double ParetoCloudUser::offtime_ [protected]

Definition at line 25 of file paretoclouduser.cc.

4.29.4.4 int ParetoCloudUser::on_ [protected]

Definition at line 37 of file paretoclouduser.cc.

4.29.4.5 double ParetoCloudUser::ontime_ [protected]

Definition at line 24 of file paretoclouduser.cc.

4.29.4.6 double ParetoCloudUser::p1_ [protected]

Definition at line 31 of file paretoclouduser.cc.

4.29.4.7 double ParetoCloudUser::p2_ [protected]

Definition at line 34 of file paretoclouduser.cc.

4.29.4.8 double ParetoCloudUser::rate_ [protected]

Definition at line 26 of file paretoclouduser.cc.

4.29.4.9 unsigned int ParetoCloudUser::rem_ [protected]

Definition at line 30 of file paretoclouduser.cc.

4.29.4.10 RNG∗ ParetoCloudUser::rng_ [protected]

Definition at line 40 of file paretoclouduser.cc.

4.29.4.11 double ParetoCloudUser::shape_ [protected]

Definition at line 29 of file paretoclouduser.cc.

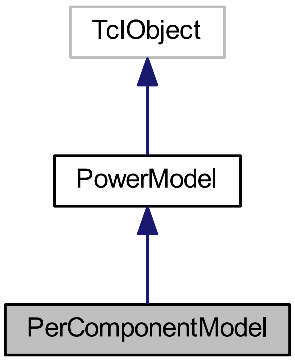
The documentation for this class was generated from the following file:

- paretoclouduser.cc

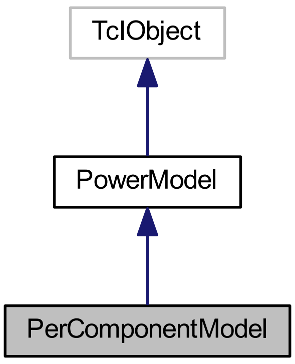
4.30 PerComponentModel Class Reference

```
#include <percomponentmodel.h>
```

Inheritance diagram for PerComponentModel:



Collaboration diagram for PerComponentModel:



Public Member Functions

- PerComponentModel ()
- virtual ∼PerComponentModel ()
- virtual void print ()
- virtual int command (int argc, const char ∗const ∗argv)
- virtual double estimate (int size, double ∗predictors)
- virtual double getMaxPower ()
- virtual void addComponent (DcResource ∗component)

Public Attributes

- std::vector< DcResource * > modeled_components_

4.30.1 Detailed Description

Definition at line 17 of file percomponentmodel.h.

4.30.2 Constructor & Destructor Documentation

4.30.2.1 PerComponentModel::PerComponentModel ( )

Definition at line 19 of file percomponentmodel.cc.

```
                                       {


}
```

4.30.2.2 PerComponentModel::∼PerComponentModel ( ) [virtual]

Definition at line 24 of file percomponentmodel.cc.

```
                                        {

}
```

4.30.3 Member Function Documentation

4.30.3.1 void PerComponentModel::addComponent (DcResource * *component*) [virtual]

Implements PowerModel.

Definition at line 28 of file percomponentmodel.cc.

```
                                                         {
                modeled_components_.push_back(component);
}
```

4.30.3.2 int PerComponentModel::command (int *argc*, const char *const * *argv*) [virtual]

Reimplemented from PowerModel.

Definition at line 32 of file percomponentmodel.cc.

```
                                                              {
                if (argc == 2) {
                                if (strcmp(argv[1], "print") == 0) {
                                                /* print general info */
                                                print();
                                                return (TCL_OK);
                                }
                } else if (argc==3){
                                if (strcmp(argv[1], "set-name") == 0) {
                                                this->setName(argv[2]);
                                                return(TCL_OK);
                                } else {
                                                return(TCL_ERROR);
                                }
                }
                return(TCL_ERROR);

}
```

4.30.3.3 double PerComponentModel::estimate (int *size,* double ∗ *predictors*) [virtual]

Implements PowerModel.

Definition at line 50 of file percomponentmodel.cc.

```
                                                {
        std::vector<DcResource*>::iterator iter;
        double result = 0;
        for(iter = modeled_components_.begin(); iter !=
 modeled_components_.end(); iter++){
                result+=(*iter)->getPower();
        }
        return result;
}
```

4.30.3.4 double PerComponentModel::getMaxPower ( ) [virtual]

Implements PowerModel.

Definition at line 59 of file percomponentmodel.cc.

```
                            {
        std::vector<DcResource*>::iterator iter;
        double result = 0;
        for(iter = modeled_components_.begin(); iter !=
 modeled_components_.end(); iter++){
                result+=(*iter)->getMaxPower();
        }
        return result;
}
```

4.30.3.5 void PerComponentModel::print ( ) [virtual]

Reimplemented from PowerModel.

Definition at line 69 of file percomponentmodel.cc.

```
                        {
        std::cout << "Per component power model.\nModeled components:\n";
        std::vector<DcResource*>::iterator iter;
        for(iter = modeled_components_.begin(); iter !=
 modeled_components_.end(); iter++){
                std::cout << (*iter)->specification->getPowerModel()->name_ << "\n";
        }

}
```

4.30.4 Member Data Documentation

4.30.4.1 std::vector<DcResource∗> PerComponentModel::modeled_components_

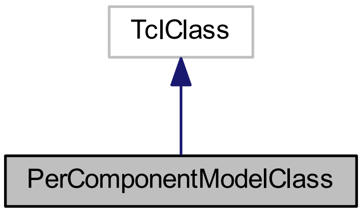
Definition at line 26 of file percomponentmodel.h.

The documentation for this class was generated from the following files:

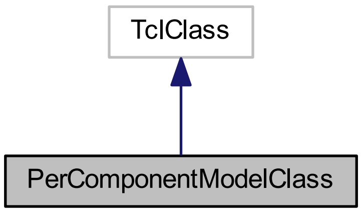
- percomponentmodel.h
- percomponentmodel.cc

4.31 PerComponentModelClass Class Reference

Inheritance diagram for PerComponentModelClass:



Collaboration diagram for PerComponentModelClass:



Public Member Functions

- PerComponentModelClass ()
- TclObject ∗ create (int argc, const char ∗const ∗argv)

4.31.1 Detailed Description

Definition at line 11 of file percomponentmodel.cc.

4.31.2 Constructor & Destructor Documentation

4.31.2.1 PerComponentModelClass::PerComponentModelClass ( ) [inline]

Definition at line 13 of file percomponentmodel.cc.

```
13 : TclClass("PerComponentModel") {}
```

4.31.3 Member Function Documentation

4.31.3.1 TclObject∗ PerComponentModelClass::create (int *argc,* const char ∗const ∗ *argv*) [inline]

Definition at line 14 of file percomponentmodel.cc.

```
                                                  {
                                  return (new PerComponentModel());
                  }
```

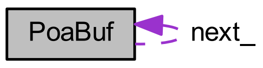
The documentation for this class was generated from the following file:

- percomponentmodel.cc

4.32 PoaBuf Class Reference

`#include <provideroutagent.h>`

Collaboration diagram for PoaBuf:



Public Member Functions

- PoaBuf (void ∗c, int nbytes)
- ∼PoaBuf ()
- void ∗ pointer ()
- int bytes ()

Protected Attributes

- void ∗ pointer_
- int nbytes_
- PoaBuf ∗ next_

Friends

- class PoaBufList

4.32.1 Detailed Description

Definition at line 17 of file provideroutagent.h.

4.32.2 Constructor & Destructor Documentation

4.32.2.1 PoaBuf::PoaBuf (void ∗ *c*, int *nbytes*)

Definition at line 63 of file provideroutagent.cc.

```
{
                nbytes_ = nbytes;
                pointer_=c;
                next_ = NULL;
}
```

4.32.2.2 PoaBuf::∼PoaBuf ( ) [inline]

Definition at line 20 of file provideroutagent.h.

```
                          {
                }
```

4.32.3 Member Function Documentation

4.32.3.1 int PoaBuf::bytes ( ) [inline]

Definition at line 23 of file provideroutagent.h.

```
{ return nbytes_; }
```

4.32.3.2 void∗ PoaBuf::pointer ( ) [inline]

Definition at line 22 of file provideroutagent.h.

```
{ return pointer_; }
```

4.32.4 Friends And Related Function Documentation

4.32.4.1 friend class PoaBufList [friend]

Definition at line 27 of file provideroutagent.h.

4.32.5 Member Data Documentation

4.32.5.1 int PoaBuf::nbytes_ [protected]

Total length of this transmission

Definition at line 29 of file provideroutagent.h.

4.32.5.2 PoaBuf∗ PoaBuf::next_ [protected]

Definition at line 30 of file provideroutagent.h.

4.32.5.3 void∗ PoaBuf::pointer_ [protected]

Definition at line 28 of file provideroutagent.h.

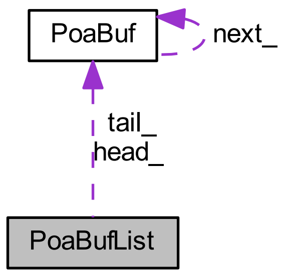
The documentation for this class was generated from the following files:

- provideroutagent.h
- provideroutagent.cc

4.33 PoaBufList Class Reference

`#include <provideroutagent.h>`

Collaboration diagram for PoaBufList:



Public Member Functions

- PoaBufList ()
- ∼PoaBufList ()
- void insert (PoaBuf ∗poabuf)
- PoaBuf ∗ detach ()

Protected Attributes

- PoaBuf ∗ head_
- PoaBuf ∗ tail_

4.33.1 Detailed Description

Definition at line 35 of file provideroutagent.h.

4.33.2 Constructor & Destructor Documentation

4.33.2.1 PoaBufList::PoaBufList ( ) [inline]

Definition at line 38 of file provideroutagent.h.

```
: head_(NULL), tail_(NULL) {}
```

4.33.2.2 PoaBufList::∼PoaBufList ( )

Definition at line 70 of file provideroutagent.cc.

```
{
                while (head_ != NULL) {
                                tail_ = head_;
                                head_ = head_->next_;
                                delete tail_;
                }
}
```

4.33.3 Member Function Documentation

4.33.3.1 PoaBuf ∗ PoaBufList::detach ( )

Definition at line 89 of file provideroutagent.cc.

```
{
                if (head_ == NULL)
                                return NULL;
                PoaBuf *p = head_;
                if ((head_ = head_->next_) == NULL)
                                tail_ = NULL;
                return p;
}
```

4.33.3.2 void PoaBufList::insert (PoaBuf ∗ *poabuf*)

Definition at line 79 of file provideroutagent.cc.

```
{
                if (tail_ == NULL)
                                head_ = tail_ = poabuf;
                else {
                                tail_->next_ = poabuf;
                                tail_ = poabuf;
                }
}
```

4.33.4 Member Data Documentation

4.33.4.1 PoaBuf∗ PoaBufList::head_ [protected]

Definition at line 45 of file provideroutagent.h.

4.33.4.2 PoaBuf∗ PoaBufList::tail_ [protected]

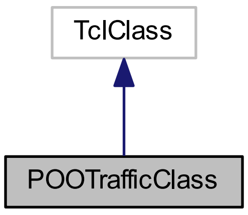
Definition at line 46 of file provideroutagent.h.

The documentation for this class was generated from the following files:

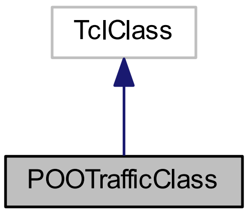
- provideroutagent.h
- provideroutagent.cc

4.34 POOTrafficClass Class Reference

Inheritance diagram for POOTrafficClass:



Collaboration diagram for POOTrafficClass:



Public Member Functions

- POOTrafficClass ()
- TclObject ∗ create (int, const char ∗const ∗)

4.34.1 Detailed Description

Definition at line 44 of file paretoclouduser.cc.

4.34.2 Constructor & Destructor Documentation

4.34.2.1 POOTrafficClass::POOTrafficClass ( ) [inline]

Definition at line 46 of file paretoclouduser.cc.

```
: TclClass("Application/Traffic/ParetoCloudUser") {}
```

4.34.3 Member Function Documentation

4.34.3.1 TclObject∗ POOTrafficClass::create (int , const char ∗const ∗) [inline]

Definition at line 47 of file paretoclouduser.cc.

```
                                                        {
                                 return (new ParetoCloudUser());
                 }
```

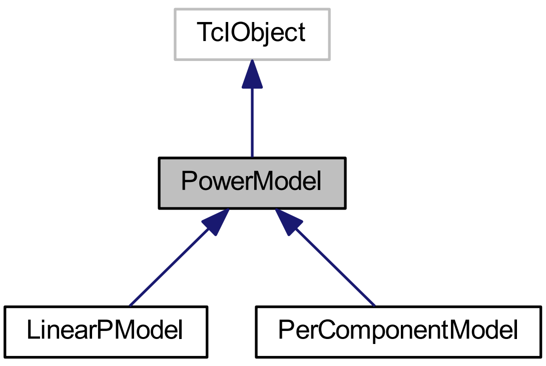
The documentation for this class was generated from the following file:

- paretoclouduser.cc

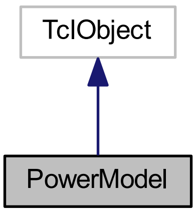
4.35 PowerModel Class Reference

```
#include <powermodel.h>
```

Inheritance diagram for PowerModel:



Collaboration diagram for PowerModel:



Public Member Functions

- PowerModel ()
- virtual ∼PowerModel ()
- virtual void print ()
- virtual int command (int argc, const char ∗const ∗argv)
- virtual double estimate (int size, double ∗predictors)=0
- virtual double getMaxPower ()=0
- virtual void addComponent (DcResource ∗component)=0
- void setName (const char ∗name)

Public Attributes

- std::string name_

4.35.1 Detailed Description

Definition at line 17 of file powermodel.h.

4.35.2 Constructor & Destructor Documentation

4.35.2.1 PowerModel::PowerModel ( )

Definition at line 11 of file powermodel.cc.

```
                        {

}
```

4.35.2.2 PowerModel::∼PowerModel ( ) [virtual]

Definition at line 15 of file powermodel.cc.

```
                         {

}
```

4.35.3 Member Function Documentation

4.35.3.1 virtual void PowerModel::addComponent (DcResource ∗ *component*) [pure virtual]

Implemented in PerComponentModel, and LinearPModel.

4.35.3.2 int PowerModel::command (int *argc*, const char ∗const ∗ *argv*) [virtual]

Reimplemented in PerComponentModel, and LinearPModel.

Definition at line 26 of file powermodel.cc.

```
{

        if (argc == 2) {
                if (strcmp(argv[1], "print") == 0) {

                        /* print general info */
                        print();
                        return (TCL_OK);
                }
        }
        return (PowerModel::command(argc, argv));
}
```

4.35.3.3 virtual double PowerModel::estimate (int *size*, double ∗ *predictors*) [pure virtual]

Implemented in PerComponentModel, and LinearPModel.

4.35.3.4 virtual double PowerModel::getMaxPower ( ) [pure virtual]

Implemented in PerComponentModel, and LinearPModel.

4.35.3.5 void PowerModel::print ( ) [virtual]

Reimplemented in LinearPModel, and PerComponentModel.

Definition at line 22 of file powermodel.cc.

```
                {
        std::cout << "Abstract power model";
}
```

4.35.3.6 void PowerModel::setName (const char ∗ *name*)

Definition at line 19 of file powermodel.cc.

```
                        {
        name_=name;
}
```

4.35.4 Member Data Documentation

4.35.4.1 std::string PowerModel::name_

Definition at line 27 of file powermodel.h.

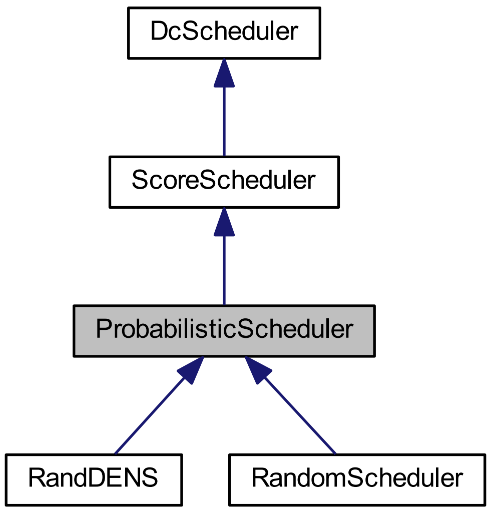
The documentation for this class was generated from the following files:

- powermodel.h
- powermodel.cc

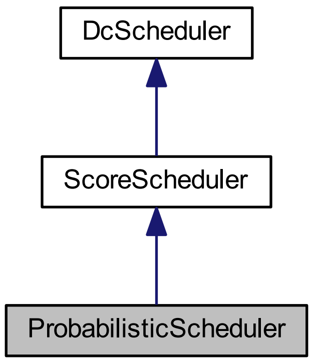
4.36 ProbabilisticScheduler Class Reference

```
#include <probabilisticscheduler.h>
```

Inheritance diagram for ProbabilisticScheduler:



Collaboration diagram for ProbabilisticScheduler:



Public Member Functions

- ProbabilisticScheduler ()
- virtual ∼ProbabilisticScheduler ()
- virtual TskComAgent ∗ scheduleTask (CloudTask ∗task, std::vector< ResourceProvider ∗ > providers)

Private Member Functions

- virtual double calculateScore (ResourceProvider ∗rp)=0

4.36.1 Detailed Description

Definition at line 15 of file probabilisticscheduler.h.

4.36.2 Constructor & Destructor Documentation

4.36.2.1 ProbabilisticScheduler::ProbabilisticScheduler ( )

Definition at line 10 of file probabilisticscheduler.cc.

```
                                              {


 }
```

4.36.2.2 ProbabilisticScheduler::∼ProbabilisticScheduler ( ) [virtual]

Definition at line 15 of file probabilisticscheduler.cc.

```
                                               {

 }
```

4.36.3 Member Function Documentation

4.36.3.1 virtual double ProbabilisticScheduler::calculateScore (**ResourceProvider** ∗ *rp*) [private], [pure virtual]

Implements ScoreScheduler.

Implemented in RandDENS, and RandomScheduler.

4.36.3.2 **TskComAgent** ∗ ProbabilisticScheduler::scheduleTask (**CloudTask** ∗ *task,* std::vector< **ResourceProvider** ∗ > *providers*) [virtual]

Implements DcScheduler.

Reimplemented in RandDENS, and RandomScheduler.

Definition at line 19 of file probabilisticscheduler.cc.

```
                                                                                          {
                //1. calculate mDENS score array
                //2. generate random number in [0:summed mDENS score]
                //3. binary search on the array
                //4. return the selected host
                TskComAgent* selected = NULL;
                std::vector<double>  mdens_score(providers.size());
                std::vector<double>   mdens_score_cumulative(providers.size());
                vector <ResourceProvider*>::iterator res_p;
                vector <double>::iterator score;
                vector <double>::iterator cumul;
                for (res_p = providers.begin(), score = mdens_score.begin(), cumul = mdens_score_cumulative
     .begin();
                                                res_p!=providers.end();
                                                res_p++, score++, cumul++)
                {
                                (*score) = calculateScore((*res_p));
                                if(cumul==mdens_score_cumulative.begin()){
                                                (*cumul) = (*score);
                                } else {
                                                (*cumul) = (*score) + (*(cumul-1));
                                }
//                              std::cout << "score:" << (*score)<< " cumul: " << (*cumul) << "\n";
                }
//              srand(time(0));
                double r = ( (double)rand() / (double)RAND_MAX ) * mdens_score_cumulative.at(
     mdens_score_cumulative.size()-1);
//              std::cerr << "Random: " << r  << "\n";
                vector <double>::iterator lb = lower_bound(mdens_score_cumulative.begin(),
     mdens_score_cumulative.end(),r);
                int sel_ind = lb - mdens_score_cumulative.begin();
//              std::cerr << "Lb: "<< (*lb) << " ind: " << sel_ind << "\n";
//              abort();
                selected = providers.at(sel_ind)->getTskComAgent();
                return selected;
}
```

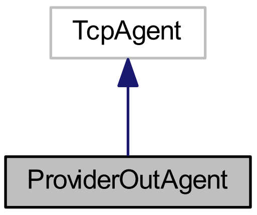
The documentation for this class was generated from the following files:

- probabilisticscheduler.h
- probabilisticscheduler.cc

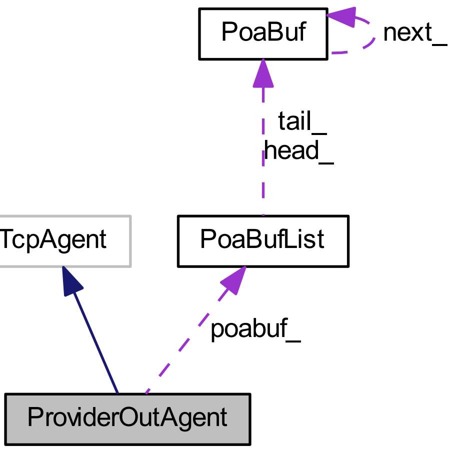
4.37 ProviderOutAgent Class Reference

```
#include <provideroutagent.h>
```

Inheritance diagram for ProviderOutAgent:



Collaboration diagram for ProviderOutAgent:



Public Member Functions

- ProviderOutAgent ()
- virtual ∼ProviderOutAgent ()
- int updateAgentDataBytes ()
- double updateTime ()
- void sendmsg (int nbytes, void ∗pointer)
- void tryToSend ()

Protected Attributes

- int lastTrackedBytes_
- double lastTrackedTime_
- PoaBufList poabuf_

4.37.1 Detailed Description

Definition at line 50 of file provideroutagent.h.

4.37.2 Constructor & Destructor Documentation

4.37.2.1 ProviderOutAgent::ProviderOutAgent ( )

Definition at line 19 of file provideroutagent.cc.

```
                                    : lastTrackedBytes_(0),
      lastTrackedTime_(0.0){

                  lastTrackedTime_  = Scheduler::instance().clock();
   //             lastTrackedBytes = this->getNdatabytes();
   }
```

4.37.2.2 ProviderOutAgent::∼ProviderOutAgent ( ) [virtual]

Definition at line 25 of file provideroutagent.cc.

```
                                      {
                  PoaBuf* tmp;
                  while(true){
                  tmp = poabuf_.detach();
                  if(tmp!=NULL){
                                  delete tmp;
                  } else {
                                  break;}
                  }
   }
```

4.37.3 Member Function Documentation

4.37.3.1 void ProviderOutAgent::sendmsg (int *nbytes*, void ∗ *pointer*)

Definition at line 48 of file provideroutagent.cc.

```
                                                      {
                  poabuf_.insert(new PoaBuf( pointer,nbytes));
                  tryToSend();
   }
```

4.37.3.2 void ProviderOutAgent::tryToSend ( )

Definition at line 53 of file provideroutagent.cc.

```
                        {
        if(current_pointer_==NULL){
                PoaBuf* cb = poabuf_.detach();
                if(cb!=NULL){
                        TcpAgent::sendmsg(cb->bytes(),(void*) cb->
pointer());
                        delete cb;
                }
        }
}
```

4.37.3.3 int ProviderOutAgent::updateAgentDataBytes ( )

Definition at line 36 of file provideroutagent.cc.

```
                                {
        int result =  getNdatabytes() - lastTrackedBytes_;
        lastTrackedBytes_ = getNdatabytes();
        return result;
}
```

4.37.3.4 double ProviderOutAgent::updateTime ( )

Definition at line 42 of file provideroutagent.cc.

```
                        {
        double result = lastTrackedTime_;
        lastTrackedTime_ = Scheduler::instance().clock();
        return result;
}
```

4.37.4 Member Data Documentation

4.37.4.1 int ProviderOutAgent::lastTrackedBytes_ [protected]

Definition at line 59 of file provideroutagent.h.

4.37.4.2 double ProviderOutAgent::lastTrackedTime_ [protected]

Definition at line 60 of file provideroutagent.h.

4.37.4.3 PoaBufList ProviderOutAgent::poabuf_ [protected]

Definition at line 61 of file provideroutagent.h.

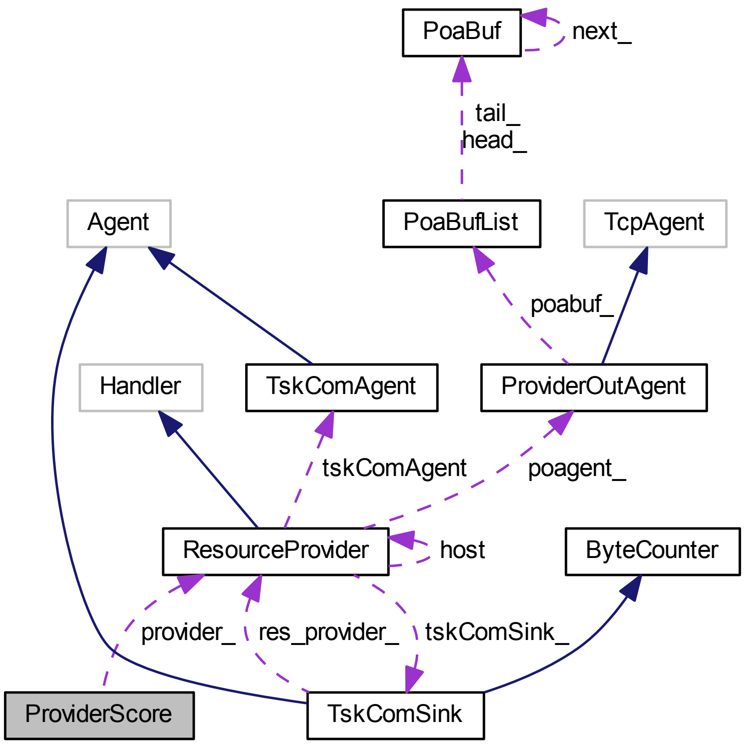
The documentation for this class was generated from the following files:

- provideroutagent.h
- provideroutagent.cc

4.38 ProviderScore Class Reference

```
#include <providerscore.h>
```

Collaboration diagram for ProviderScore:



Public Member Functions

- ProviderScore (ResourceProvider ∗provider, double score)
- ProviderScore (ResourceProvider ∗provider, double score, double comm_potential_)
- virtual ∼ProviderScore ()
- bool operator< (const ProviderScore &other) const

Public Attributes

- ResourceProvider ∗ provider_
- double score_
- double comm_potential_

4.38.1 Detailed Description

Definition at line 11 of file providerscore.h.

4.38.2 Constructor & Destructor Documentation

4.38.2.1 ProviderScore::ProviderScore (ResourceProvider ∗ *provider,* double *score*)

Definition at line 10 of file providerscore.cc.

```
                                                                   : provider_(provider),
     score_(score){


 }
```

4.38.2.2 ProviderScore::ProviderScore (ResourceProvider ∗ *provider,* double *score,* double *comm_potential_*)

Definition at line 15 of file providerscore.cc.

```
                                                                                           :
     provider_(provider),score_(score), comm_potential_(
     comm_potential_){


 }
```

4.38.2.3 ProviderScore::∼ProviderScore ( ) [virtual]

Definition at line 22 of file providerscore.cc.

```
                             {

 }
```

4.38.3 Member Function Documentation

4.38.3.1 bool ProviderScore::operator< (const ProviderScore & *other*) const

Definition at line 26 of file providerscore.cc.

```
                                                              {
     return this->score_ < other.score_;
 }
```

4.38.4 Member Data Documentation

4.38.4.1 double ProviderScore::comm_potential_

Definition at line 18 of file providerscore.h.

4.38.4.2 ResourceProvider∗ ProviderScore::provider_

Definition at line 16 of file providerscore.h.

4.38.4.3 double ProviderScore::score_

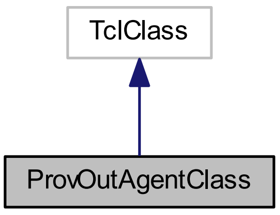
Definition at line 17 of file providerscore.h.

The documentation for this class was generated from the following files:

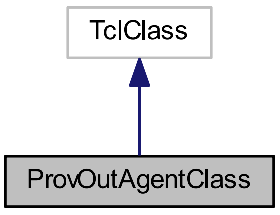
- providerscore.h
- providerscore.cc

4.39 ProvOutAgentClass Class Reference

Inheritance diagram for ProvOutAgentClass:



Collaboration diagram for ProvOutAgentClass:



Public Member Functions

- ProvOutAgentClass ()
- TclObject ∗ create (int, const char ∗const ∗)

4.39.1 Detailed Description

Definition at line 11 of file provideroutagent.cc.

4.39.2 Constructor & Destructor Documentation

4.39.2.1 ProvOutAgentClass::ProvOutAgentClass ( ) [inline]

Definition at line 13 of file provideroutagent.cc.

```
: TclClass("Agent/TCP/ProvOutAgent") {}
```

4.39.3 Member Function Documentation

4.39.3.1 TclObject∗ ProvOutAgentClass::create (int , const char ∗const ∗) [inline]

Definition at line 14 of file provideroutagent.cc.

```
                                           {
                                 return (new ProviderOutAgent());
                  }
```

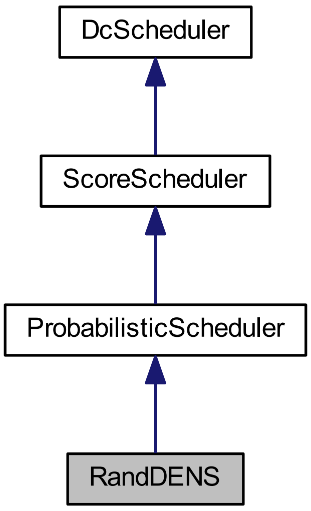
The documentation for this class was generated from the following file:

- provideroutagent.cc

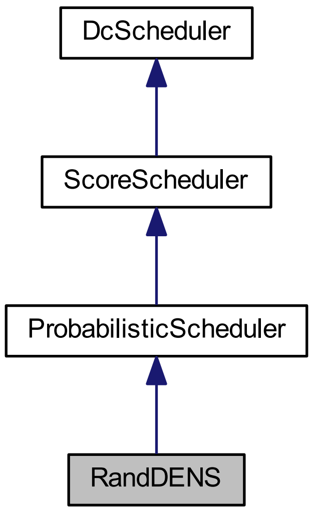
4.40 RandDENS Class Reference

`#include <randdens.h>`

Inheritance diagram for RandDENS:



Collaboration diagram for RandDENS:



Public Member Functions

- RandDENS ()
- virtual ∼RandDENS ()
- virtual TskComAgent ∗ scheduleTask (CloudTask ∗task, std::vector< ResourceProvider ∗ > providers)

Private Member Functions

- virtual double calculateScore (ResourceProvider ∗rp)
- double densLoadFactor (double load, double epsilon)
- double linkLoadFactor (double load)

Private Attributes

- double epsilon

4.40.1 Detailed Description

MultiDENS scheduler. To meaningfully use it, enableDVFS on the resource providers used by this scheduler. TO←DO: add networking part of DENS.

Definition at line 21 of file randdens.h.

4.40.2 Constructor & Destructor Documentation

4.40.2.1 RandDENS::RandDENS ( )

Definition at line 11 of file randdens.cc.

```
11                      : epsilon(0.1){
12
13 }
```

4.40.2.2 RandDENS::∼RandDENS ( ) [virtual]

Definition at line 15 of file randdens.cc.

```
15                       {
16
17 }
```

4.40.3 Member Function Documentation

4.40.3.1 double RandDENS::calculateScore (ResourceProvider ∗ *rp*) [private], [virtual]

Implements ProbabilisticScheduler.

Definition at line 23 of file randdens.cc.

```
23                                                  {
24                 double result = 0;
25                 double load;
26                 for(int i = FirstResType; i <= LastResType ; i++){
27                                 load = rp->getResTypeUtil(static_cast<res_type>(i));
28                                 result+= densLoadFactor(load,
     epsilon);
29                 }
30                 result=result/(LastResType+1); // normalize according to the number of
      dimensions
31
32                 result += linkLoadFactor(rp->getRootHost()->
     rack_->link_load);
33
34                 return result;
35 }
```

4.40.3.2 double RandDENS::densLoadFactor (double *load,* double *epsilon*) [private]

Definition at line 37 of file randdens.cc.

```
37                                                        {
38                 return 1/(1+exp(-10*(load-0.5))) - 1/(1+exp((-10/epsilon)*(load-(1-(
     epsilon/2)))));
39 }
```

4.40.3.3 double RandDENS::linkLoadFactor (double *load*) [private]

Definition at line 41 of file randdens.cc.

```
41                                         {
42                 return exp(-(load*load));
43 }
```

4.40.3.4 TskComAgent ∗ RandDENS::scheduleTask (CloudTask ∗ *task,* std::vector< ResourceProvider ∗ > *providers*) [virtual]

Reimplemented from ProbabilisticScheduler.

Definition at line 19 of file randdens.cc.

```
                                                                                    {
                  return ProbabilisticScheduler::scheduleTask(task,
     providers);
 }
```

4.40.4 Member Data Documentation

4.40.4.1 double RandDENS::epsilon [private]

Definition at line 27 of file randdens.h.

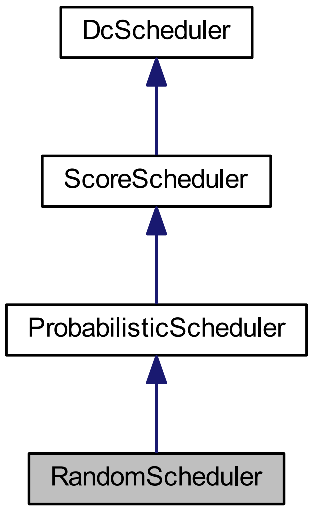
The documentation for this class was generated from the following files:

- randdens.h
- randdens.cc

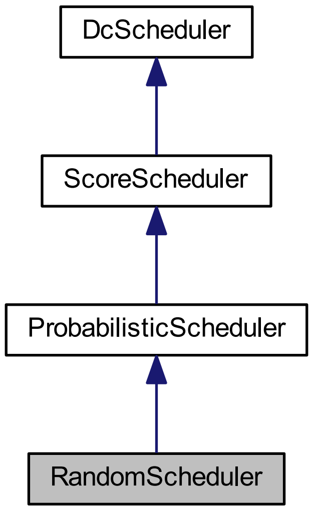
4.41 RandomScheduler Class Reference

`#include <randomscheduler.h>`

Inheritance diagram for RandomScheduler:



Collaboration diagram for RandomScheduler:



Public Member Functions

- RandomScheduler ()
- virtual ∼RandomScheduler ()
- virtual TskComAgent $*$ scheduleTask (CloudTask $*$task, std::vector$<$ ResourceProvider $*$ $>$ providers)

Private Member Functions

- virtual double calculateScore (ResourceProvider $*$rp)

4.41.1 Detailed Description

Definition at line 15 of file randomscheduler.h.

4.41.2 Constructor & Destructor Documentation

4.41.2.1 RandomScheduler::RandomScheduler ( )

Definition at line 10 of file randomscheduler.cc.

```
                                       {


}
```

4.41.2.2 RandomScheduler::∼RandomScheduler ( ) [virtual]

Definition at line 15 of file randomscheduler.cc.

```
                                {

}
```

4.41.3 Member Function Documentation

4.41.3.1 double RandomScheduler::calculateScore (**ResourceProvider** ∗ *rp*) [private], [virtual]

Implements ProbabilisticScheduler.

Definition at line 23 of file randomscheduler.cc.

```
                                                            {
                double result = 1;
                return result;
}
```

4.41.3.2 **TskComAgent** ∗ RandomScheduler::scheduleTask (**CloudTask** ∗ *task*, std::vector< **ResourceProvider** ∗ > *providers*) [virtual]

Reimplemented from ProbabilisticScheduler.

Definition at line 19 of file randomscheduler.cc.

```
                                                                                                          {
                return ProbabilisticScheduler::scheduleTask(task,
     providers);
}
```

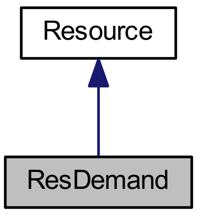
The documentation for this class was generated from the following files:

- randomscheduler.h
- randomscheduler.cc

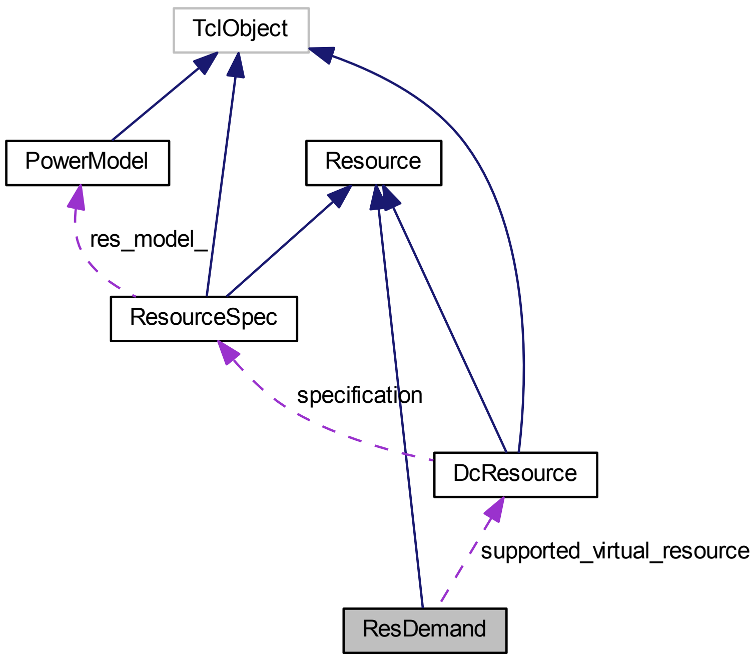
4.42 ResDemand Class Reference

`#include <resdemand.h>`

Inheritance diagram for ResDemand:



Collaboration diagram for ResDemand:



Public Member Functions

- ResDemand (Resource &res)
- ResDemand (Resource &res, DcResource ∗svr)

Public Attributes

- std::vector< Capacity ∗ > capacity_location
- std::vector< double > current_performance
- DcResource ∗ supported_virtual_resource

Additional Inherited Members

4.42.1 Detailed Description

Used to represent the status of execution. (Where, how fast, and how much left)

Definition at line 18 of file resdemand.h.

4.42.2 Constructor & Destructor Documentation

4.42.2.1 ResDemand::ResDemand (Resource & *res*)

Definition at line 11 of file resdemand.cc.

```
                                 : Resource( res){
                capacity_location = std::vector <Capacity *>(res.
    capacity.size(),NULL) ;
                current_performance = std::vector <double>(res.
    capacity.size(),0);
                supported_virtual_resource = NULL;

}
```

4.42.2.2 ResDemand::ResDemand (Resource & *res,* DcResource ∗ *svr*)

Definition at line 18 of file resdemand.cc.

```
                                                 : Resource( res){
                capacity_location = std::vector <Capacity *>(res.
    capacity.size(),NULL) ;
                current_performance = std::vector <double>(res.
    capacity.size(),0);
                if(svr != NULL){
                                std::vector<Capacity>::iterator i_res;
                                std::vector<Capacity>::iterator i_dem;
                                for(i_res = svr->capacity.begin(), i_dem = this->
    capacity.begin();
                                                                i_res != svr->
    capacity.end();
                                                                i_res++, i_dem++){
                                                i_dem->virtual_capacities.push_back(&(*i_res));
                                }
                }
                supported_virtual_resource = svr;
}
```

4.42.3 Member Data Documentation

4.42.3.1 std::vector<Capacity ∗> ResDemand::capacity_location

Currently used capacities for "static" resources (other than CPU and Networking)

Definition at line 21 of file resdemand.h.

4.42.3.2 std::vector< double > ResDemand::current_performance

Current allocation (i.e. processing rate) rate for each of the resources

Definition at line 23 of file resdemand.h.

4.42.3.3 DcResource∗ ResDemand::supported_virtual_resource

Definition at line 28 of file resdemand.h.

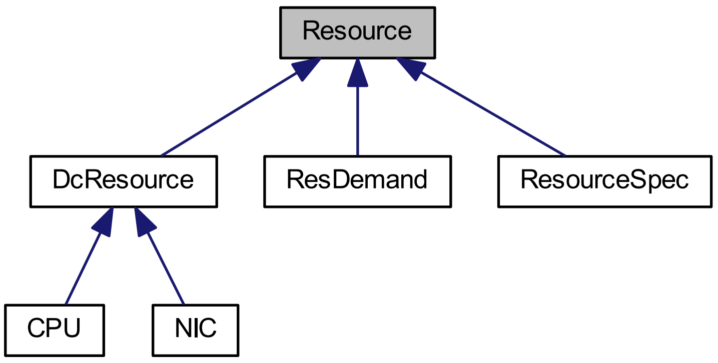
The documentation for this class was generated from the following files:

- resdemand.h
- resdemand.cc

4.43 Resource Class Reference

```
#include <resource.h>
```

Inheritance diagram for Resource:



Public Member Functions

- Resource (res_type t, double a, std::vector< Capacity > cap)
- void print ()
- virtual ∼Resource ()
- res_type getType ()
- double getArch ()

Static Public Member Functions

- static res_type translateType (const char ∗t)

Public Attributes

- std::vector< Capacity > capacity

Protected Member Functions

- Resource ()
- Resource & operator= (const Resource &r)
- int setType (const char ∗t)
- int setCapacity (std::vector< Capacity > cap)
- void sortCapacity ()

Protected Attributes

- res_type type
- double arch

4.43.1 Detailed Description

Definition at line 82 of file resource.h.

4.43.2 Constructor & Destructor Documentation

4.43.2.1 Resource::Resource (res_type *t*, double *a*, std::vector< Capacity > *cap*)

Definition at line 74 of file resource.cc.

```
                                                              {
                type = t;
                arch = a;
                this->setCapacity(cap);
 }
```

4.43.2.2 Resource::∼Resource ( ) [virtual]

Definition at line 80 of file resource.cc.

```
                    {

 }
```

4.43.2.3 Resource::Resource ( ) [protected]

Definition at line 70 of file resource.cc.

```
                   {

 }
```

4.43.3 Member Function Documentation

4.43.3.1 double Resource::getArch ( ) [inline]

Definition at line 90 of file resource.h.

```
{return arch;};
```

4.43.3.2 res_type Resource::getType ( ) [inline]

Definition at line 89 of file resource.h.

```
{return type;};
```

4.43.3.3 Resource & Resource::operator= (const Resource & *r*) [protected]

Definition at line 119 of file resource.cc.

```
                                                {
                if (this != &r) {  // make sure it is not the same object
                                capacity.clear();
                                capacity = r.capacity;
                                arch = r.arch;
                                type = r.type;
                }
                return *this;    // Return ref for multiple assignment
}
```

4.43.3.4 void Resource::print ( )

Definition at line 129 of file resource.cc.

```
                    {
                std::cerr << "Type:\t"<< type;
                std::cerr << "\n";
                std::cerr << "Architecture:\t"<< arch;
                std::cerr << "\n";
                std::cerr << "Capacities:\t";
                std::vector <Capacity>::iterator iter;
                for (iter = capacity.begin(); iter!=capacity.end(); iter++)
                {
                                std::cerr << (*iter) << ",";
                }

}
```

4.43.3.5 int Resource::setCapacity (std::vector< Capacity > *cap*) [protected]

Definition at line 110 of file resource.cc.

```
                                                  {
                capacity = cap;
                return 0;
}
```

4.43.3.6 int Resource::setType (const char ∗ *t*) [protected]

Definition at line 103 of file resource.cc.

```
                        {
            type = translateType(t);
            return 0;
}
```

4.43.3.7 void Resource::sortCapacity ( ) [protected]

Definition at line 115 of file resource.cc.

```
                   {
            std::sort(capacity.begin(),capacity.end());
}
```

4.43.3.8 res_type Resource::translateType (const char ∗ *t*) [static]

Definition at line 85 of file resource.cc.

```
                                       {
            res_type type;
            if(strcmp(t, "Computing") == 0){
                    type=Computing;
            } else if(strcmp(t, "Memory") == 0){
                    type=Memory;
            } else if(strcmp(t, "Storage") == 0){
                    type=Storage;
            } else if(strcmp(t, "Networking") == 0){
                    type=Networking;
            } else {
                    std::cerr << "Unknown resource type" << t;
                    abort();

            }
            return type;
}
```

4.43.4 Member Data Documentation

4.43.4.1 double Resource::arch [protected]

Definition at line 99 of file resource.h.

4.43.4.2 std::vector<Capacity> Resource::capacity

Capacities of resources, e.g. MIPS for each core, B for each disk in an array

Definition at line 87 of file resource.h.

4.43.4.3 res_type Resource::type [protected]

Type of resource, see enum (e.g. CPU, Memory, Disk, Network Interface)

Definition at line 95 of file resource.h.

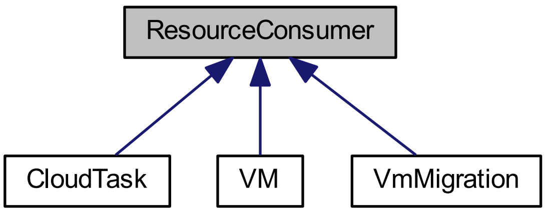
The documentation for this class was generated from the following files:

- resource.h
- resource.cc

4.44 ResourceConsumer Class Reference

```
#include <resourceconsumer.h>
```

Inheritance diagram for ResourceConsumer:



Public Member Functions

- ResourceConsumer ()
- ResourceConsumer (unsigned int size, std::vector< Resource ∗ > dem, bool isTask, bool isVM)
- virtual ∼ResourceConsumer ()
- unsigned int getSize ()
- void setSize (unsigned int size)
- void setCurrentPerformance (std::vector< double > newPerf)
- void addUsedCapacity (double ∗cap)

Public Attributes

- bool isTask
- bool isVM
- double size_
- double currProcRate_
- std::vector< ResDemand ∗ > res_demands

4.44.1 Detailed Description

Definition at line 20 of file resourceconsumer.h.

4.44.2 Constructor & Destructor Documentation

4.44.2.1 ResourceConsumer::ResourceConsumer ( )

Definition at line 10 of file resourceconsumer.cc.

```
                                {
        res_demands.clear();
}
```

4.44.2.2 ResourceConsumer::ResourceConsumer (unsigned int *size*, std::vector< Resource * > *dem*, bool *isTask*, bool *isVM*)

Definition at line 14 of file resourceconsumer.cc.

```
                                                                                                                    :
     isTask(isTask), isVM(isVM), size_(size){

                 res_demands.clear();
                 res_demands = std::vector<ResDemand *>(demand.size(),NULL);
                 std::vector <Resource*>::iterator iter;
                 std::vector <ResDemand*>::iterator iter2;
                 for (iter = demand.begin(), iter2=res_demands.begin(); iter!=demand.end(); iter+
     +,iter2++)
                 {
                                 (*iter2)=new ResDemand(*(*iter));
                 }

                 for (iter = demand.begin() ; iter!=demand.end(); iter++)
                 {
                                 delete (*iter);
                 }
   }
```

4.44.2.3 ResourceConsumer::∼ResourceConsumer ( ) [virtual]

Definition at line 31 of file resourceconsumer.cc.

```
                                     {
                 std::vector <ResDemand*>::iterator iter2;
                 for (iter2 = res_demands.begin() ; iter2!=res_demands.end(); iter2++)
                 {
                                 delete (*iter2);
                 }
   }
```

4.44.3 Member Function Documentation

4.44.3.1 void ResourceConsumer::addUsedCapacity (double * *cap*)

4.44.3.2 unsigned int ResourceConsumer::getSize ( )

Definition at line 39 of file resourceconsumer.cc.

```
   {return size_;};
```

4.44.3.3 void ResourceConsumer::setCurrentPerformance (std::vector< double > *newPerf*)

4.44.3.4 void ResourceConsumer::setSize (unsigned int *size*)

Definition at line 42 of file resourceconsumer.cc.

```
   {size_ = size;};
```

4.44.4 Member Data Documentation

4.44.4.1 double ResourceConsumer::currProcRate_

current processing rate of the task (determined by the server)

Definition at line 34 of file resourceconsumer.h.

4.44.4.2 bool ResourceConsumer::isTask

Definition at line 27 of file resourceconsumer.h.

4.44.4.3 bool ResourceConsumer::isVM

Definition at line 28 of file resourceconsumer.h.

4.44.4.4 std::vector<**ResDemand** ∗> ResourceConsumer::res_demands

Initial demand for resources

Definition at line 37 of file resourceconsumer.h.

4.44.4.5 double ResourceConsumer::size_

amount of bytes transferred to servers for task execution

Definition at line 31 of file resourceconsumer.h.

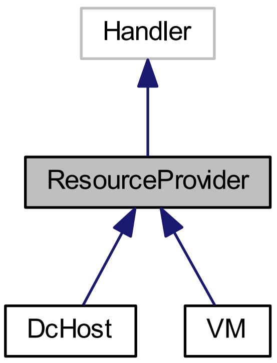
The documentation for this class was generated from the following files:

- resourceconsumer.h
- resourceconsumer.cc

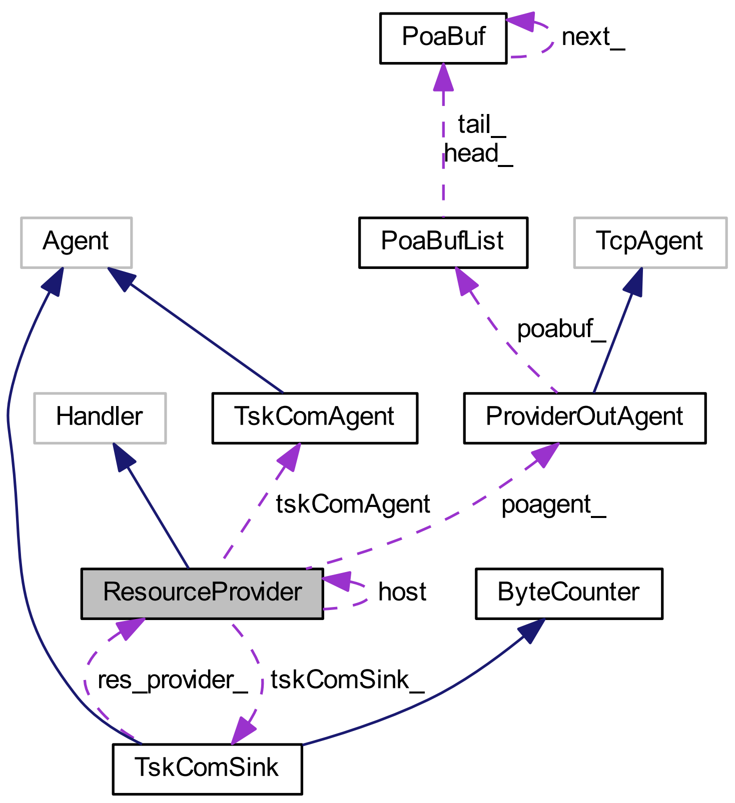
4.45 ResourceProvider Class Reference

```
#include <resourceprovider.h>
```

Inheritance diagram for ResourceProvider:



Collaboration diagram for ResourceProvider:



Public Types

- enum EventStatus

Public Member Functions

- ResourceProvider ()
- virtual ∼ResourceProvider ()
- void setTskComSink (TskComSink ∗tcs)
- int tryToAllocate (ResourceConsumer ∗rc)
- bool releaseAllocation (ResourceConsumer ∗rc)
- bool addVM (VM ∗newVm)
- bool removeVM (VM ∗vm)
- ResourceProvider ∗ getHost ()
- DcHost ∗ getRootHost ()
- void recv (ResourceConsumer ∗rcobj)
- virtual int command (int argc, const char ∗const ∗argv)
- double getResTypeUtil (res_type type)
- virtual void print ()=0
- virtual void printTasklist ()

- virtual void addResource (DcResource ∗res)
- double getTotalCap (res_type type)
- virtual void updateEnergyAndConsumption ()=0
- int testSchedulingPossibility (CloudTask ∗tskobj)
- int trySchedulingTsk (CloudTask ∗tskobj)
- void sendTaskOutput (CloudTask ∗task)
- void scheduleNextExent (double nextDeadline)
- TskComAgent ∗ getTskComAgent ()

Public Attributes

- std::vector< std::vector< DcResource ∗ > > resource_list
- int id_
- int ntasks_
- double currentLoad_
- double currentLoadMem_
- double currentLoadStor_
- double currentLoadNet_
- int eDVFS_enabled_
- int tskFailed_
- TskComAgent ∗ tskComAgent

Static Public Attributes

- static double uplink_overhead =ResourceProvider::MTU/ResourceProvider::useful_bytes
- static double MTU =1500.0
- static double useful_bytes =1460.0

Protected Member Functions

- virtual void handle (Event ∗event)
- void updateEvent ()
- void nextEvent (double delay)
- double getCurrentLoad ()
- double updateResTypeUtil (res_type type)
- double getFreeCap (res_type type)
- double getFreeCapRecursive (res_type type)
- double getUsedNet (bool in, bool out)
- double getUsedNetRecursive (bool in, bool out)
- void setTskComAgent (TskComAgent ∗agnt)
- void setAgent (ProviderOutAgent ∗agent)
- TcpAgent ∗ getAgent ()
- void attachSink (VmMigrationSink ∗vm_mig_sink)
- void attachSource (ProviderOutAgent ∗tcp_agent)
- void detachSink (VmMigrationSink ∗vm_mig_sink)
- void detachSource (ProviderOutAgent ∗tcp_agent)

Protected Attributes

- double resource_utilization [LastResType+1]
- std::vector< ResourceConsumer ∗ > hosted_vms_
- std::vector< VmMigrationSink ∗ > vm_migration_sinks_
- std::vector< ProviderOutAgent ∗ > vm_migration_sources_
- TskComSink ∗ tskComSink_
- ProviderOutAgent ∗ poagent_
- ResourceProvider ∗ host
- bool started_
- int status_
- Event event_

Private Member Functions

- void _sched (double delay)
- void _cancel ()

4.45.1 Detailed Description

Definition at line 38 of file resourceprovider.h.

4.45.2 Member Enumeration Documentation

4.45.2.1 enum **ResourceProvider::EventStatus**

Enumerator

EVENT_IDLE

EVENT_PENDING

EVENT_HANDLING

Definition at line 91 of file resourceprovider.h.

```
{ EVENT_IDLE, EVENT_PENDING, EVENT_HANDLING };
```

4.45.3 Constructor & Destructor Documentation

4.45.3.1 ResourceProvider::ResourceProvider ( )

Definition at line 18 of file resourceprovider.cc.

```
                                    : id_(0), ntasks_(0),
                                 currentLoad_ (0.0), currentLoadMem_(0.0),
     currentLoadStor_(0.0), currentLoadNet_(0.0),
                                 eDVFS_enabled_(0.0), tskFailed_(0),
     tskComAgent(NULL),   host(NULL), started_(false)
{

                for(int i = 0; i <= LastResType; i++){
                                resource_list.push_back(std::vector <DcResource*>());
                }
                for(int i = 0; i <= LastResType; i++){
                                resource_utilization[i] = 0.0;
                }
                hosted_vms_.clear();
                status_ = EVENT_IDLE;
                poagent_ = NULL;
}
```

4.45.3.2 ResourceProvider::∼ResourceProvider ( ) [virtual]

Definition at line 34 of file resourceprovider.cc.

```
                                {
                std::vector <std::vector<DcResource*> >::iterator iter;
                for(iter = resource_list.begin(); iter!=
     resource_list.end() ;iter++){
                                std::vector<DcResource*>::iterator iter2;
                                                for(iter2 = iter->begin(); iter2!=iter->end() ;iter2++){
                                                                delete (*iter2);
                                                }
                }
                delete poagent_;

}
```

4.45.4 Member Function Documentation

4.45.4.1 void ResourceProvider::_cancel ( ) [inline], [private]

Definition at line 520 of file resourceprovider.cc.

```
                                    {
                (void)Scheduler::instance().cancel(&event_);
                // no need to free event_ since it's statically allocated
}
```

4.45.4.2 void ResourceProvider::_sched (double *delay*) [inline], [private]

Definition at line 517 of file resourceprovider.cc.

```
                                                {
                (void)Scheduler::instance().schedule(this, &event_, delay);
}
```

4.45.4.3 void ResourceProvider::addResource (DcResource ∗ *res*) [virtual]

Reimplemented in DcHost, and VM.

Definition at line 498 of file resourceprovider.cc.

```
                                              {

                resource_list[res->getType()].push_back(res);
                if(res->getType()==Computing){
                                CPU* cpu_res = static_cast<CPU*>(res);
                                cpu_res->setDVFS(eDVFS_enabled_);
                                cpu_res->setProvider(this);
                }
                if(res->getType()==Networking){
                                NIC* nic_res = static_cast<NIC*>(res);
                                nic_res->setRp(this);
                }

}
```

4.45.4.4 bool ResourceProvider::addVM (VM ∗ *newVm*)

Definition at line 199 of file resourceprovider.cc.

```
199                                     {
200 
201                 this->updateEnergyAndConsumption();
202 
203                 if(tryToAllocate(newVm)){
204                                 ((newVm))->setHost(this);
205                                 hosted_vms_.push_back(newVm);
206                                 return true;
207                 } else {
208                                 return false;
209                 }
210 }
```

4.45.4.5 void ResourceProvider::attachSink (VmMigrationSink ∗ *vm_mig_sink*) [protected]

Definition at line 565 of file resourceprovider.cc.

```
565                                                             {
566                 vm_migration_sinks_.push_back(vm_mig_sink);
567 }
```

4.45.4.6 void ResourceProvider::attachSource (ProviderOutAgent ∗ *tcp_agent*) [protected]

Definition at line 570 of file resourceprovider.cc.

```
570                                                        {
571                 vm_migration_sources_.push_back(poa);
572 }
```

4.45.4.7 int ResourceProvider::command (int *argc,* const char ∗const ∗ *argv*) [virtual]

Reimplemented in DcHost, and VM.

Definition at line 585 of file resourceprovider.cc.

```
586 {
587                 Tcl& tcl = Tcl::instance();
588 
589                 if (argc == 2) {
590                                 return (TCL_ERROR);
591 
592                 } else if (argc == 3) {
593                                 if (strcmp(argv[1], "attach-agent") == 0) {
594                                                 setAgent((
      ProviderOutAgent*) TclObject::lookup(argv[2]));
595                                                 if (getAgent() == 0) {
596                                                                 tcl.resultf("no such agent %s", argv[2]);
597                                                                 return(TCL_ERROR);
598                                                 }
599                                                 return(TCL_OK);
600                                 }
601                                 else if (strcmp(argv[1], "set-taskcomagent") == 0) {
602                                                 TskComAgent *agnt = dynamic_cast<
      TskComAgent*> (TclObject::lookup(argv[2]));
603                                                 if(agnt){
604                                                                 setTskComAgent(agnt);
605                                                                 return (TCL_OK);
606                                                 }
607                                                 return (TCL_ERROR);
608                                 }
609                                 else            if (strcmp(argv[1], "attach-vm-mig-sink") == 0) {
610                                                 VmMigrationSink* vm_mig_sink = ((
VmMigrationSink*) TclObject::lookup(argv[2]));
                                                attachSink(vm_mig_sink);
                                                if (getAgent() == 0) {
                                                                tcl.resultf("no such agent %s", argv[2]);
                                                                return(TCL_ERROR);
                                                }
                                                return(TCL_OK);
                                } else          if (strcmp(argv[1], "attach-vm-mig-source") == 0) {
                                                ProviderOutAgent* vm_migration_source =  ((
ProviderOutAgent*) TclObject::lookup(argv[2]));
                                                attachSource(vm_migration_source);
                                                if (getAgent() == 0) {
                                                                tcl.resultf("no such agent %s", argv[2]);
                                                                return(TCL_ERROR);
                                                }
                                                return(TCL_OK);
                                }  else         if (strcmp(argv[1], "detach-vm-mig-sink") == 0) {
                                                VmMigrationSink* vm_mig_sink = ((
VmMigrationSink*) TclObject::lookup(argv[2]));
                                                detachSink(vm_mig_sink);
                                                if (getAgent() == 0) {
                                                                tcl.resultf("no such agent %s", argv[2]);
                                                                return(TCL_ERROR);
                                                }
                                                return(TCL_OK);
                                } else          if (strcmp(argv[1], "detach-vm-mig-source") == 0) {
                                                ProviderOutAgent* vm_migration_source =  ((
ProviderOutAgent*) TclObject::lookup(argv[2]));
                                                detachSource(vm_migration_source);
                                                if (getAgent() == 0) {
                                                                tcl.resultf("no such agent %s", argv[2]);
                                                                return(TCL_ERROR);
                                                }
                                                return(TCL_OK);
                                } else if (strcmp(argv[1], "add-resource") == 0) {
                                                DcResource* res = (
DcResource*) TclObject::lookup(argv[2]);
                                                if (res == NULL) {
                                                                tcl.resultf("no such resource %s", argv[2])
;
                                                                return(TCL_ERROR);
                                                }
                                                addResource(res);
                                                return(TCL_OK);
                                } else if (strcmp(argv[1], "add-vm") == 0) {
                                                VM* vm = (VM*) TclObject::lookup(argv[2]);
                                                if (vm == NULL) {
                                                                tcl.resultf("no such vm %s", argv[2]);
                                                                return(TCL_ERROR);
                                                }
                                                if(addVM(vm)){
                                                                return(TCL_OK);
                                                } else {
                                                                /* It was impossible to allocate vm on the
 host.*/
                                                                std::cerr << "ERROR: A VM was allocated on
 a machine that has not enough resources. (Creation was called from Tcl)";
                                                                return(TCL_ERROR);
                                                }
                                }
                }
                return (TCL_ERROR);
}
```

4.45.4.8 void ResourceProvider::detachSink (VmMigrationSink * *vm_mig_sink*) [protected]

Definition at line 574 of file resourceprovider.cc.

```
                                                {
                vm_migration_sinks_.erase(remove(
vm_migration_sinks_.begin(),vm_migration_sinks_.end(),vm_mig_sink),
                                                vm_migration_sinks_.end()); /*
erase-remove idiom*/
}
```

4.45.4.9 void ResourceProvider::detachSource (ProviderOutAgent ∗ *tcp_agent*) [protected]

Definition at line 580 of file resourceprovider.cc.

```
                                                {
                vm_migration_sources_.erase(remove(
    vm_migration_sources_.begin(),vm_migration_sources_.end(),poa),
                                                vm_migration_sources_.end()); /*
    erase-remove idiom*/
}
```

4.45.4.10 TcpAgent ∗ ResourceProvider::getAgent ( ) [protected]

Definition at line 534 of file resourceprovider.cc.

```
                                {
                return poagent_;

}
```

4.45.4.11 double ResourceProvider::getCurrentLoad ( ) [protected]

Definition at line 414 of file resourceprovider.cc.

```
{
                double nominal_mips = 0;
                double current_mips = 0;
                std::vector <DcResource*>::iterator cpu_iter;
                for(cpu_iter=resource_list[Computing].begin(); cpu_iter !=
    resource_list[Computing].end(); cpu_iter++){
                                DcResource* res = *cpu_iter;
                                CPU* cpu  = (CPU*) res;
                                nominal_mips += cpu->getNominalMIPS();
                                current_mips += cpu->getCurrentMIPS();
                }
                currentLoad_ = current_mips/nominal_mips;

                return currentLoad_;
}
```

4.45.4.12 double ResourceProvider::getFreeCap (res_type *type*) [protected]

Definition at line 308 of file resourceprovider.cc.

```
                                            {
                double free_cap = 0;
                std::vector <DcResource*>::iterator dc_res;
                for(dc_res=resource_list[type].begin(); dc_res !=
    resource_list[type].end(); dc_res++){
                                std::vector <Capacity>::iterator free_cap_iter;
                                for(free_cap_iter = (*dc_res)->capacity.begin();
                                                                free_cap_iter != (*dc_res)->capacity.end();
                                                                free_cap_iter++){
                                                free_cap +=     *free_cap_iter;
                                }
                }

                return free_cap;
}
```

4.45.4.13 double ResourceProvider::getFreeCapRecursive (res_type type) [protected]

Definition at line 298 of file resourceprovider.cc.

```
                                                        {
                double free_cap = getFreeCap(type);

                std::vector <ResourceConsumer*>::iterator vm_iter;
                for(vm_iter=hosted_vms_.begin(); vm_iter !=
   hosted_vms_.end(); vm_iter++){
                                VM* vm = static_cast<VM*>(*vm_iter);
                                free_cap += vm->getFreeCapRecursive(type);
                }
                return free_cap;
}
```

4.45.4.14 ResourceProvider * ResourceProvider::getHost ( )

Definition at line 184 of file resourceprovider.cc.

```
                                          {
                return host;
}
```

4.45.4.15 double ResourceProvider::getResTypeUtil (res_type type)

Definition at line 379 of file resourceprovider.cc.

```
                                                   {
                if(type==Networking){
                                return resource_utilization[
   Networking];
                } else {
                                return updateResTypeUtil(type);
                }
}
```

4.45.4.16 DcHost * ResourceProvider::getRootHost ( )

Definition at line 189 of file resourceprovider.cc.

```
                                    {
                if(host == NULL){
                                DcHost* root = static_cast<DcHost*>(this);
                                return root;
                } else {
                                return host->getRootHost();
                }

}
```

4.45.4.17 double ResourceProvider::getTotalCap (res_type type)

Definition at line 363 of file resourceprovider.cc.

```
                                                {
                double total_cap = 0;
                std::vector <DcResource*>::iterator dc_res;
                for(dc_res=resource_list[type].begin(); dc_res !=
   resource_list[type].end(); dc_res++){
                                std::vector <Capacity>::iterator total_cap_iter;
                                for(total_cap_iter = (*dc_res)->specification->capacity.begin();
                                                                total_cap_iter != (*dc_res)->specification
   ->capacity.end();
                                                                total_cap_iter++){
                                                total_cap += *total_cap_iter;
                                }

                }
                return total_cap;

}
```

4.45.4.18 TskComAgent ∗ ResourceProvider::getTskComAgent ( )

Definition at line 543 of file resourceprovider.cc.

```
{
        return this->tskComAgent;
}
```

4.45.4.19 double ResourceProvider::getUsedNet (bool *in,* bool *out*) [protected]

Definition at line 323 of file resourceprovider.cc.

```
{
        double result = 0;
        if(in){
                double elapsed_time = Scheduler::instance().clock() - this->tskComSink_->getLastBytesSinceTime();
                if(elapsed_time>0){
                double down_link_util = this->tskComSink_->resetBytesSince();
                std::vector<VmMigrationSink*>::iterator vms;
                for(vms = vm_migration_sinks_.begin();vms!=vm_migration_sinks_.end();vms++){
                        double recent_bytes = (*vms)->resetBytesSince();
                        down_link_util += recent_bytes;
}
                result += (down_link_util/elapsed_time);
                }
        }
        if(out){
                double elapsed_time = Scheduler::instance().clock() - this->poagent_->updateTime();
                if(elapsed_time>0){
                double up_link_util = this->poagent_->updateAgentDataBytes();
                std::vector<ProviderOutAgent*>::iterator poa;
                for(poa = vm_migration_sources_.begin();poa!=vm_migration_sources_.end();poa++){
                        double recent_bytes = (*poa)->updateAgentDataBytes();
                        up_link_util += recent_bytes;
                }
                result += ((up_link_util*ResourceProvider::uplink_overhead)/elapsed_time);
                }
        }

        return result;
}
```

4.45.4.20 double ResourceProvider::getUsedNetRecursive (bool *in,* bool *out*) [protected]

Definition at line 352 of file resourceprovider.cc.

```
{
        double used_net = getUsedNet(in,out);
        std::vector <ResourceConsumer*>::iterator vm_iter;
        for(vm_iter=hosted_vms_.begin(); vm_iter != hosted_vms_.end(); vm_iter++){
                VM* vm = static_cast<VM*>(*vm_iter);
                used_net += vm->getUsedNetRecursive(in,out);
        }

        return used_net;
}
```

4.45.4.21 void ResourceProvider::handle (Event * *event*) [protected], [virtual]

Definition at line 284 of file resourceprovider.cc.

```
{
                std::vector <CoreScheduler*>::iterator core_s;
                std::vector <DcResource*>::iterator cpu_iter;
                for(cpu_iter=resource_list[Computing].begin(); cpu_iter !=
      resource_list[Computing].end(); cpu_iter++){
                                CPU* cpu = (CPU*) (*cpu_iter);
                                for(core_s=cpu->cores_schedulers_.begin(); core_s != cpu->
      cores_schedulers_.end(); core_s++){
                                                (*core_s)->updateTskList();
                                }
                }


}
```

4.45.4.22 void ResourceProvider::nextEvent (double *delay*) [protected]

Definition at line 270 of file resourceprovider.cc.

```
{
                if (status_ == EVENT_PENDING) {
                                _cancel();
                                status_ = EVENT_IDLE;
                }

                event_.handler_ = this;
                event_.time_ = Scheduler::instance().clock();

                _sched(delay);
                status_ = EVENT_PENDING;
}
```

4.45.4.23 virtual void ResourceProvider::print ( ) [pure virtual]

Implemented in DcHost, and VM.

4.45.4.24 void ResourceProvider::printTasklist ( ) [virtual]

Reimplemented in DcHost, and VM.

Definition at line 526 of file resourceprovider.cc.

```
                                {
                std::cout << "printTasklist Status: (FUNCTION UNDER CONSTRUCTION)\n";

}
```

4.45.4.25 void ResourceProvider::recv (ResourceConsumer ∗ *rcobj*)

Definition at line 228 of file resourceprovider.cc.

```
{
        this->updateEnergyAndConsumption();
        if(rcobj->isTask==true){
                vector<CloudTask*>::iterator iter;
                CloudTask* tskobj = (CloudTask*) rcobj;

                ntasks_ ++;                                             // update total number of the received tasks

                if(tskobj->scheduled_==false){
                        if(tryScheduling Tsk(tskobj)==false){
                                tskobj->fail(this);
//                                      std::cout << "Unscheduled task failed due to insufficient resources";
                                return;
                        }
                }
                std::vector <CoreScheduler*>::iterator core_s;
                std::vector <DcResource*>::iterator cpu_iter;

                /*If it is posible to allocate:*/
                if(tryToAllocate(tskobj)){
                        for(cpu_iter=resource_list[Computing].begin(); cpu_iter != resource_list[Computing].end(); cpu_iter++){
                                CPU* cpu = (CPU*) (*cpu_iter);
                                for(core_s=cpu->cores_schedulers_.begin(); core_s != cpu->cores_schedulers_.end(); core_s++){
                                        (*core_s)->startTaskExecution(tskobj);
                                }
                        }
                        /*Otherwise task fails!*/
                } else {
                        tskobj->fail(this);
                        std::cout << "Task failed due to insufficient resources";
                        return;
                }

        } else {
                std::cerr <<"It is not a task!";
                return;
        }
}
```

4.45.4.26 bool ResourceProvider::releaseAllocation (ResourceConsumer ∗ *rc*)

Definition at line 137 of file resourceprovider.cc.

```
                                                        {
        if((*rc).res_demands.empty()){
                std::cerr << "Nothing to release \n";
                return true;
        }       //      std::cerr << "Something to release \n";

        std::vector <ResDemand*>::iterator u_res;
        for (u_res = rc->res_demands.begin() ; u_res!=rc->res_demands.end(); u_res++)
        {

                if(((*u_res)->getType()!=Computing && (*u_res)->getType()!= Networking ) || rc->isTask == false){
                        std::vector <Capacity>::iterator consumption;
                        std::vector <Capacity *>::iterator location;
                        for(consumption=(*u_res)->capacity.begin(),
                                        location=(*u_res)->capacity_location.begin();
                                        consumption!=(*u_res)->capacity.end();
                                        consumption++,location++){
                                if((*location)==NULL){
                                } else {
                                        **location = (**location)+
(*consumption);
                                                *location = NULL;
                                        }
                                }
                        }
                        if(rc->isTask==false){
                                if((*u_res)->supported_virtual_resource != NULL){
                                        if(             (*u_res)->supported_virtual_resource->getType()==Computing){
                                                CPU* cpu = (CPU*)(*u_res)->supported_virtual_resource;
                                                std::vector<CoreScheduler*>::iterator cs;
                                                for(cs = cpu->cores_schedulers_.begin(); cs != cpu->cores_schedulers_.end(); cs++){
                                                        CoreScheduler* host_cs = (*cs)->getHostScheduler();
                                                        if(host_cs!=NULL){
                                                                host_cs->removeVcoreScheduler(*cs);
                                                        }
                                                }
                                        }
                                }
                        }

                }
                if(rc->isTask==false){
                        // non-task (VM  or migration) specific cleanup. Handled in the respective classes.
                }

                return true;
}
```

4.45.4.27 bool ResourceProvider::removeVM (VM ∗ *vm*)

Definition at line 212 of file resourceprovider.cc.

```
{
                this->updateEnergyAndConsumption();
                if(releaseAllocation(vm)){
                        (vm)->setHost(NULL);
                        hosted_vms_.erase(remove(hosted_vms_.begin(), hosted_vms_.end(),vm),
                                                        hosted_vms_.end()); /* erase-remove idiom*/
                        return true;
                } else {
                        return false;
                }
}
```

4.45.4.28 void ResourceProvider::scheduleNextExent (double *nextDeadline*)

Definition at line 558 of file resourceprovider.cc.

```
{
                /* reschedule next update */
                if (nextDeadline != DBL_MAX) nextEvent(nextDeadline);


}
```

4.45.4.29 void ResourceProvider::sendTaskOutput (CloudTask ∗ *task*)

Definition at line 547 of file resourceprovider.cc.

```
{
                if ((getAgent()) && (task->getOutput() != 0)) {
                        /*Record finish time of task on the server.*/
                        task->info_->setServerFinishTime(Scheduler::instance().clock());
                        task->info_->setResourceProvider(this);
                        /*Send task output.*/
                        poagent_->sendmsg(task->getOutput(),task);
                }
}
```

4.45.4.30 void ResourceProvider::setAgent (ProviderOutAgent * *agent*) [protected]

Definition at line 530 of file resourceprovider.cc.

```
                                            {
                poagent_ = agent;
}
```

4.45.4.31 void ResourceProvider::setTskComAgent (TskComAgent * *agnt*) [protected]

Definition at line 539 of file resourceprovider.cc.

```
                                            {
                this->tskComAgent = agnt;
}
```

4.45.4.32 void ResourceProvider::setTskComSink (TskComSink * *tcs*)

Definition at line 513 of file resourceprovider.cc.

```
                                        {
                this->tskComSink_ =tcs;
}
```

4.45.4.33 int ResourceProvider::testSchedulingPossibility (CloudTask * *tskobj*)

Definition at line 430 of file resourceprovider.cc.

```
                                                            {
                int result = trySchedulingTsk(tskobj);
                if(result){
                                releaseAllocation(tskobj);
                                tskobj->releaseAllTaskAllocs();
                                tskobj->scheduled_=false;
                }
                return result;
}
```

4.45.4.34 int ResourceProvider::trySchedulingTsk (CloudTask * *tskobj*)

Definition at line 440 of file resourceprovider.cc.

```
{

                /* get minimum processing rate required by the task */
                if(tryToAllocate(tskobj)){
                                releaseAllocation(tskobj);
                } else {
                                return false;
                }

                std::vector<TaskAlloc*> tmp_task_allocs;
                tmp_task_allocs.clear();

                std::vector<TaskAlloc*>::iterator iter;
                for(iter = tskobj->task_allocations_.begin() ; iter != tskobj->
   task_allocations_.end();iter++){
                                TaskAlloc* task_alloc = (*iter);
                                bool core_found = false;
                                tmp_task_allocs.push_back(task_alloc);

459                              double tskrate = (double)task_alloc->getMIPS()/(task_alloc->
      getDeadline() - Scheduler::instance().clock());
460
461                              std::vector <CoreScheduler*>::iterator core_s;
462                              std::vector <DcResource*>::iterator cpu_iter;
463                              for(cpu_iter=resource_list[
      Computing].begin(); cpu_iter != resource_list[Computing].end(); cpu_iter++){
464                                      CPU* cpu = (CPU*) (*cpu_iter);
465                                      for(core_s=cpu->
      cores_schedulers_.begin(); core_s != cpu->cores_schedulers_.end(); core_s
      ++){
466
467                                              double maxrate = (*core_s)->
      getMostUrgentTaskRate();
468                                              if (tskrate > maxrate){maxrate = tskrate;}
469                                              if (maxrate*((*core_s)->getAllTasksNumber()
       + 1) <= (*core_s)->getAvailableMIPS()){
470                                                      /* task can be scheduled,
       add to the in-fly list */
471                                                      (*core_s)->assignTask(
      task_alloc);
472                                                      core_found = true;
473                                                      break;
474                                              }
475                                      }
476                                      if(core_found){break;}
477                              }
478                              if(core_found==false){
479                                              tskobj->
      releaseAllTaskAllocs();
480                                      //Release all tmp_task_allocs
481 //                                   std::vector<TaskAlloc*>::iterator failed_alloc;
482 //                                   for(failed_alloc = tmp_task_allocs.begin();failed_alloc !=
       tmp_task_allocs.end(); failed_alloc++){
483 //                                           CoreScheduler* core_of_failed =
       (*failed_alloc)->getCoreScheduler();
484 //                                           if(core_of_failed!=NULL){
485 //
      core_of_failed->removeFromAssginedList((*failed_alloc));
486 //                                           }
487 //                                   }
488                                      return false;
489                              }
490                      }
491                      tskobj->scheduled_=true;
492                      return true;
493
494 }
```

4.45.4.35 int ResourceProvider::tryToAllocate (ResourceConsumer * *rc*)

Definition at line 47 of file resourceprovider.cc.

```
47                                                      {
48                   std::vector <ResDemand*>::iterator u_res;
49
50                   // TODO (possible) 1. Sort the provider resources according to the free capacity
       (descending)
51                   // TODO (possible) 2. Sort the consumer ... (the same).
52
53                   /*For each resource demand of consumer:*/
54                   for (u_res = rc->res_demands.begin() ; u_res!=rc->
      res_demands.end(); u_res++)
55                   {
56                           //                              /*For dynamic consumers (e.g. tasks) do not
       reserve computing and networking resource.
57                           //                               * For non-dynamic resources (e.g. VMs):
       reserve computing and networking. */
58                           if(((*u_res)->getType()!=Computing && (*u_res)->getType()!=
      Networking ) || rc->isTask == false){
59                                           bool possible = false;
60
61                                           std::vector <Capacity>::iterator req_cap_cons = (*u_res)->
      capacity.begin();
62                                           std::vector <Capacity *>::iterator loc_cap_cons = (*u_res)
      ->capacity_location.begin();
63                                           std::vector <CoreScheduler*>::iterator u_core;
64                                           if((*u_res)->getType()==
      Computing && (*u_res)->supported_virtual_resource){
65                                                   u_core = ((CPU*)((*u_res)->
      supported_virtual_resource))->cores_schedulers_.begin();
```

```
                                             }

                                             std::vector <DcResource*>::iterator p_res;
                                             std::vector <CoreScheduler*>::iterator p_core;
                                             /*For each DcResource of provider:*/
                                             for(p_res = resource_list[(*u_res)->getType()]
.begin(); p_res!=resource_list[(*u_res)->getType()].end() ;p_res++){

                                                             /*Check architecture*/
                                                             if((*u_res)->getArch() <= (*p_res)->getArch
()){

                                                                             if((*u_res)->capacity.empty
()==true){
                                                                                             possible =
true;
                                                                             }

                                                                             std::vector
 <Capacity>::iterator aval_cap_prov = (*p_res)->capacity.begin();
                                                                             if((*u_res)->getType()==
Computing && (*u_res)->supported_virtual_resource){
                                                                                             p_core = ((
CPU*)(*p_res))->cores_schedulers_.begin();
                                                                             }
                                                                             /*Case of empty capacity
 vector - check only arch constraint.*/

                                                                             for(; aval_cap_prov!=
(*p_res)->capacity.end() ; ){
                                                                                             if((*
aval_cap_prov)>=(*req_cap_cons)){

(*aval_cap_prov)-=(*req_cap_cons);

/*Demands capacities are linked with the supported resources capacities (1 to 1),

 *  so the intermediary ResDemand is neglected:*/

if(rc->isVM){

(*aval_cap_prov).virtual_capacities.push_back(req_cap_cons->virtual_capacities.at(0));

}

(*loc_cap_cons)=&(*aval_cap_prov);

if((*u_res)->getType()==Computing && (*u_res)->supported_virtual_resource){

(*p_core)->addVcoreScheduler((*u_core));

}

req_cap_cons++;

loc_cap_cons++;

if(req_cap_cons== (*u_res)->capacity.end()){

possible = true;

break;

}

if((*u_res)->getType()==Computing && (*u_res)->supported_virtual_resource){

u_core++;

}

                                                                                              } else {

aval_cap_prov++;

if((*u_res)->getType()==Computing && (*u_res)->supported_virtual_resource){

p_core++;

}
                                                                                              }
                                                                              }

                                                              }
                                                              /*Resource architecture rejected:*/
                                                              else {
                                                                              //std::cerr << "Arch,
Requested: " << (*u_res)->getArch()  << "\tProvided: " <<(*p_res)->getArch() << "\n";
                                        }
                                        if(possible == true){
                                                /*Break the main loop of scanning the provider resources.*/
                                                break;
                                        }
                                }
                                if(possible==false){
                                        //std::cerr << "Impossible to allocate here.\n";
                                        releaseAllocation(rc);
                                        return false;
                                }
                        }
                }
                //std::cerr << "Allocation success.\n";
                return true;
}
```

4.45.4.36 virtual void ResourceProvider::updateEnergyAndConsumption ( ) [pure virtual]

Implemented in DcHost, and VM.

4.45.4.37 void ResourceProvider::updateEvent ( ) [protected]

4.45.4.38 double ResourceProvider::updateResTypeUtil (res_type *type*) [protected]

Definition at line 387 of file resourceprovider.cc.

```
                                                {
                if(type==Computing){
                        resource_utilization[Computing] =  getCurrentLoad();
                        return resource_utilization[Computing];
                } else if(type == Networking){
                        double total_cap = getTotalCap(type) * 2; // Bidirectional links
                        double used_net_bytes = getUsedNetRecursive(true,true);
                        double result = used_net_bytes/total_cap;
                        resource_utilization[Networking] =currentLoadNet_ = result;
                        return resource_utilization[Networking];
                } else {
                        double total_cap = getTotalCap(type);
                        if(total_cap==0){
                                return 0; //There is no components of this resource type
                        }
                        double free_cap = getFreeCapRecursive(type);
                        double result =  1 - (free_cap/total_cap);
                        if(type == Memory){
                                currentLoadMem_ = result;
                        } else if(type==Storage){
                                currentLoadStor_ = result;
                        }
                        resource_utilization[type] = result;
                        return result;
                }
}
```

4.45.5 Member Data Documentation

4.45.5.1 double ResourceProvider::currentLoad_

Definition at line 76 of file resourceprovider.h.

4.45.5.2 double ResourceProvider::currentLoadMem_

Definition at line 77 of file resourceprovider.h.

4.45.5.3 double ResourceProvider::currentLoadNet_

Definition at line 79 of file resourceprovider.h.

4.45.5.4 double ResourceProvider::currentLoadStor_

Definition at line 78 of file resourceprovider.h.

4.45.5.5 int ResourceProvider::eDVFS_enabled_

Definition at line 88 of file resourceprovider.h.

4.45.5.6 Event ResourceProvider::event_ [protected]

Definition at line 139 of file resourceprovider.h.

4.45.5.7 ResourceProvider* ResourceProvider::host [protected]

Definition at line 110 of file resourceprovider.h.

4.45.5.8 std::vector<ResourceConsumer*> ResourceProvider::hosted_vms_ [protected]

hosted vm list

Definition at line 104 of file resourceprovider.h.

4.45.5.9 int ResourceProvider::id_

Definition at line 71 of file resourceprovider.h.

4.45.5.10 double ResourceProvider::MTU =1500.0 [static]

Definition at line 95 of file resourceprovider.h.

4.45.5.11 int ResourceProvider::ntasks_

Definition at line 75 of file resourceprovider.h.

4.45.5.12 ProviderOutAgent* ResourceProvider::poagent_ [protected]

Definition at line 108 of file resourceprovider.h.

4.45.5.13 std::vector<std::vector <DcResource*> > ResourceProvider::resource_list

Definition at line 42 of file resourceprovider.h.

4.45.5.14 double ResourceProvider::resource_utilization[LastResType+1] [protected]

These values are for reading

Definition at line 103 of file resourceprovider.h.

4.45.5.15 bool ResourceProvider::started_ [protected]

Definition at line 137 of file resourceprovider.h.

4.45.5.16 int ResourceProvider::status_ [protected]

Definition at line 138 of file resourceprovider.h.

4.45.5.17 TskComAgent∗ ResourceProvider::tskComAgent

Definition at line 98 of file resourceprovider.h.

4.45.5.18 TskComSink∗ ResourceProvider::tskComSink_ [protected]

Definition at line 107 of file resourceprovider.h.

4.45.5.19 int ResourceProvider::tskFailed_

Definition at line 93 of file resourceprovider.h.

4.45.5.20 double ResourceProvider::uplink_overhead =ResourceProvider::MTU/ResourceProvider::useful_bytes [static]

Definition at line 94 of file resourceprovider.h.

4.45.5.21 double ResourceProvider::useful_bytes =1460.0 [static]

Definition at line 96 of file resourceprovider.h.

4.45.5.22 std::vector<VmMigrationSink∗> ResourceProvider::vm_migration_sinks_ [protected]

Definition at line 105 of file resourceprovider.h.

4.45.5.23 std::vector<ProviderOutAgent ∗> ResourceProvider::vm_migration_sources_ [protected]

Definition at line 106 of file resourceprovider.h.

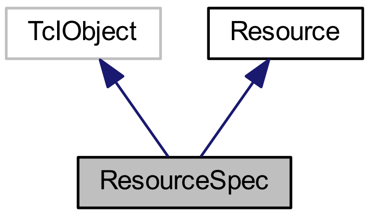
The documentation for this class was generated from the following files:

- resourceprovider.h
- resourceprovider.cc

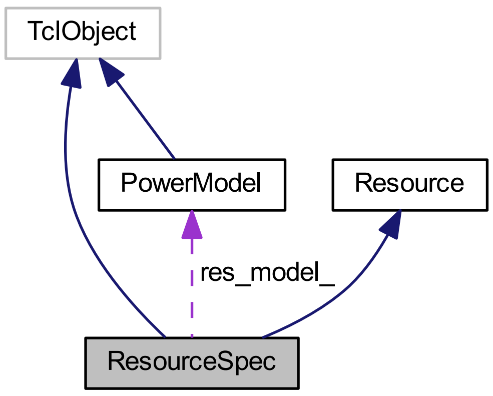
4.46 ResourceSpec Class Reference

```
#include <resourcespec.h>
```

Inheritance diagram for ResourceSpec:



Collaboration diagram for ResourceSpec:



Public Member Functions

- ResourceSpec ()
- virtual ∼ResourceSpec ()
- virtual int command (int argc, const char ∗const ∗argv)
- void print ()
- int addCapacity (double cap)
- int addPowerState (int ps)
- int setName (const char ∗name)
- int setArch (const char ∗name)
- PowerModel ∗ getPowerModel ()

Public Attributes

- std::string name_
- std::vector< int > power_states

Private Member Functions

- void setPowerModel (PowerModel *model)

Private Attributes

- PowerModel * res_model_

Friends

- class DcResource

Additional Inherited Members

4.46.1 Detailed Description

Definition at line 24 of file resourcespec.h.

4.46.2 Constructor & Destructor Documentation

4.46.2.1 ResourceSpec::ResourceSpec ( )

Definition at line 19 of file resourcespec.cc.

```
                            :  name_("NA"), res_model_(NULL){
                //              std::cerr << ("Resource Spec Constructor.\n");
                capacity.clear();
                power_states.clear();
}
```

4.46.2.2 ResourceSpec::∼ResourceSpec ( ) [virtual]

Definition at line 25 of file resourcespec.cc.

```
                             {
                name_.clear();
}
```

4.46.3 Member Function Documentation

4.46.3.1 int ResourceSpec::addCapacity (double *cap*)

Definition at line 29 of file resourcespec.cc.

```
                                    {
                capacity.push_back(cap);
                return 0;
}
```

4.46.3.2 int ResourceSpec::addPowerState (int *ps*)

Definition at line 33 of file resourcespec.cc.

```
                                  {
                power_states.push_back(ps);
                return 0;
}
```

4.46.3.3 int ResourceSpec::command (int *argc,* const char ∗const ∗ *argv*) [virtual]

Definition at line 74 of file resourcespec.cc.

```
                                                     {


                if (argc == 2) {
                                if (strcmp(argv[1], "print") == 0) {
                                                print();
                                                return (TCL_OK);
                                }
                } else if (argc == 3) {
                                if (strcmp(argv[1], "add-capacity") == 0) {
                                                addCapacity(atof(argv[2]));
                                                return (TCL_OK);
                                } else if(strcmp(argv[1], "add-power-state") == 0){
                                                addPowerState(atoi(argv[2]));
                                                return (TCL_OK);
                                } else if(strcmp(argv[1], "set-type") == 0){
                                                if(Resource::setType(argv[2])==0){
                                                                return (TCL_OK);
                                                } else {
                                                                return (TCL_ERROR);
                                                }
                                } else if(strcmp(argv[1], "set-name") == 0){
                                                setName(argv[2]);
                                                return (TCL_OK);
                                } else if(strcmp(argv[1], "set-arch") == 0){
                                                setArch(argv[2]);
                                                 return (TCL_OK);
                                 } else if (strcmp(argv[1], "set-power-model") == 0) {
                                                 PowerModel* pm = (
      PowerModel*) TclObject::lookup(argv[2]);
                                                 this->setPowerModel(pm);
                                                 return (TCL_OK);
                                 }
                 }
                 return (ResourceSpec::command(argc, argv));
}
```

4.46.3.4 PowerModel ∗ ResourceSpec::getPowerModel ( )

Definition at line 53 of file resourcespec.cc.

```
                                    {
                return res_model_;
}
```

4.46.3.5 void ResourceSpec::print ( )

Definition at line 57 of file resourcespec.cc.

```
                        {
                std::cerr << "ResourceSpec:\t";
                std::cerr << name_;
                std::cerr << "\n";
                Resource::print();
                std::vector <int>::iterator iter;
                std::cerr << "Power states:\t";
                for (iter = power_states.begin(); iter!=power_states.end(); iter++)
                {
                                std::cerr << (*iter) << ",";
                }
                std::cerr << "\n";

}
```

4.46.3.6 int ResourceSpec::setArch (const char * *name*)

Definition at line 44 of file resourcespec.cc.

```
                                        {
                arch = atof(name);
                return 0;
}
```

4.46.3.7 int ResourceSpec::setName (const char * *name*)

Definition at line 39 of file resourcespec.cc.

```
                                        {
                name_=name;
                return 0;
}
```

4.46.3.8 void ResourceSpec::setPowerModel (PowerModel * *model*) [private]

Definition at line 49 of file resourcespec.cc.

```
                                                {
                res_model_  = model;
}
```

4.46.4 Friends And Related Function Documentation

4.46.4.1 friend class DcResource [friend]

Definition at line 26 of file resourcespec.h.

4.46.5 Member Data Documentation

4.46.5.1 std::string ResourceSpec::name_

Definition at line 34 of file resourcespec.h.

4.46.5.2 std::vector<int> ResourceSpec::power_states

Definition at line 36 of file resourcespec.h.

4.46.5.3 PowerModel∗ ResourceSpec::res_model_ [private]

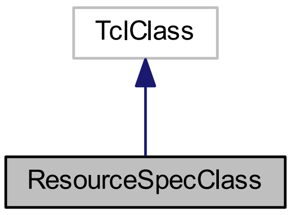
Definition at line 47 of file resourcespec.h.

The documentation for this class was generated from the following files:

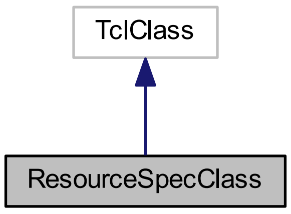
- resourcespec.h
- resourcespec.cc

4.47 ResourceSpecClass Class Reference

Inheritance diagram for ResourceSpecClass:



Collaboration diagram for ResourceSpecClass:



Public Member Functions

- ResourceSpecClass ()
- TclObject ∗ create (int argc, const char ∗const ∗argv)

4.47.1 Detailed Description

Definition at line 11 of file resourcespec.cc.

4.47.2 Constructor & Destructor Documentation

4.47.2.1 ResourceSpecClass::ResourceSpecClass ( ) [inline]

Definition at line 13 of file resourcespec.cc.

```
13 : TclClass("ResourceSpec") {}
```

4.47.3 Member Function Documentation

4.47.3.1 TclObject∗ ResourceSpecClass::create (int *argc*, const char ∗const ∗ *argv*) [inline]

Definition at line 14 of file resourcespec.cc.

```
14                                               {
15                          return (new ResourceSpec());
16            }
```

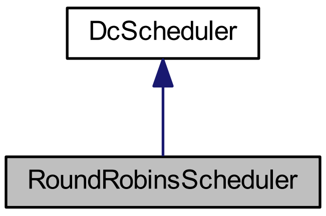
The documentation for this class was generated from the following file:

- resourcespec.cc

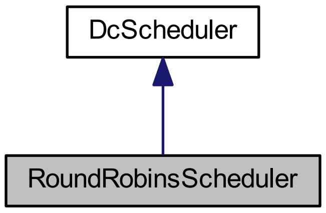
4.48 RoundRobinsScheduler Class Reference

```
#include <roundrobinscheduler.h>
```

Inheritance diagram for RoundRobinsScheduler:



Collaboration diagram for RoundRobinsScheduler:



Public Member Functions

- RoundRobinsScheduler ()
- virtual ∼RoundRobinsScheduler ()
- virtual TskComAgent $*$ scheduleTask (CloudTask $*$task, std::vector$<$ ResourceProvider $*$ $>$ providers)

4.48.1 Detailed Description

Definition at line 13 of file roundrobinscheduler.h.

4.48.2 Constructor & Destructor Documentation

4.48.2.1 RoundRobinsScheduler::RoundRobinsScheduler ( )

Definition at line 10 of file roundrobinscheduler.cc.

```
                                          {


}
```

4.48.2.2 RoundRobinsScheduler::∼RoundRobinsScheduler ( ) [virtual]

Definition at line 15 of file roundrobinscheduler.cc.

```
                                           {

}
```

4.48.3 Member Function Documentation

4.48.3.1 TskComAgent ∗ RoundRobinsScheduler::scheduleTask (CloudTask ∗ *task*, std::vector< ResourceProvider ∗ > *providers*) [virtual]

Implements DcScheduler.

Definition at line 19 of file roundrobinscheduler.cc.

```
                                                                                              {
                int j = task->id_ % providers.size();

                return (providers.at(j)->getTskComAgent());
}
```

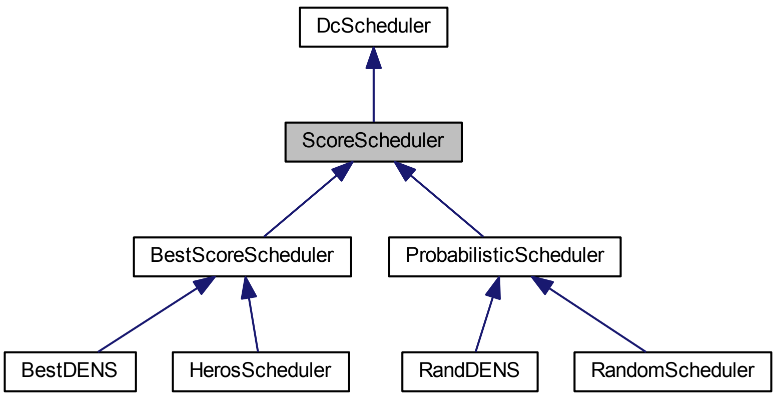
The documentation for this class was generated from the following files:

- roundrobinscheduler.h
- roundrobinscheduler.cc

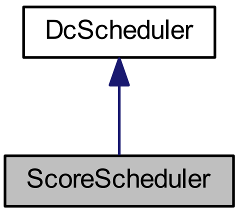
4.49 ScoreScheduler Class Reference

`#include <scorescheduler.h>`

Inheritance diagram for ScoreScheduler:



Collaboration diagram for ScoreScheduler:



Public Member Functions

- ScoreScheduler ()
- virtual ∼ScoreScheduler ()

Private Member Functions

- virtual double calculateScore (ResourceProvider ∗rp)=0

4.49.1 Detailed Description

Definition at line 13 of file scorescheduler.h.

4.49.2 Constructor & Destructor Documentation

4.49.2.1 ScoreScheduler::ScoreScheduler ( )

Definition at line 10 of file scorescheduler.cc.

```
10                                   {
11
12
13 }
```

4.49.2.2 ScoreScheduler::∼ScoreScheduler ( ) [virtual]

Definition at line 15 of file scorescheduler.cc.

```
15                                    {
16
17 }
```

4.49.3 Member Function Documentation

4.49.3.1 virtual double ScoreScheduler::calculateScore (ResourceProvider ∗ *rp*) [private], [pure virtual]

Implemented in HerosScheduler, BestDENS, RandDENS, ProbabilisticScheduler, RandomScheduler, and Best←ScoreScheduler.

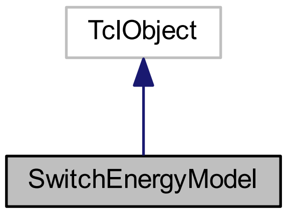
The documentation for this class was generated from the following files:

- scorescheduler.h
- scorescheduler.cc

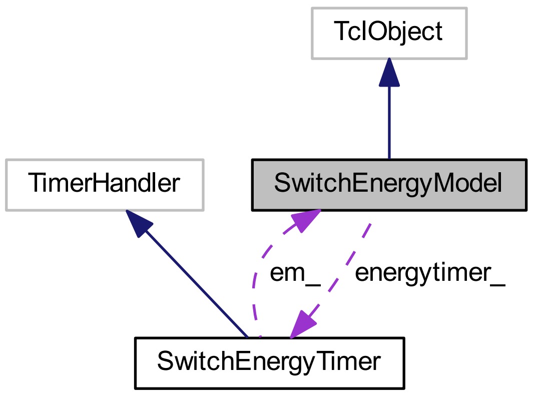
4.50 SwitchEnergyModel Class Reference

```
#include <switchenergymodel.h>
```

Inheritance diagram for SwitchEnergyModel:



Collaboration diagram for SwitchEnergyModel:



Public Member Functions

- SwitchEnergyModel ()
- virtual ∼SwitchEnergyModel ()
- virtual int command (int argc, const char ∗const ∗argv)
- virtual void timeout ()
- void setClassifier (Classifier ∗classifier)
- void updateEnergy (int curSlot, int nports)
- void start ()
- void stop ()

Public Attributes

- double eConsumed_
- double eChassis_
- double eLineCard_
- double ePort_
- double eSimEnd_
- int eDVFS_enabled_
- int eDNS_enabled_
- double eDNS_delay_

Protected Member Functions

- double computeCurrentRate ()

Protected Attributes

- int eEnabled_
- double eCurrentRate_
- double eLastSample_
- int eActivePorts_
- double eSimDuration_
- Classifier ∗ classifier_
- SwitchEnergyTimer energytimer_

4.50.1 Detailed Description

Definition at line 25 of file switchenergymodel.h.

4.50.2 Constructor & Destructor Documentation

4.50.2.1 SwitchEnergyModel::SwitchEnergyModel ( )

Definition at line 16 of file switchenergymodel.cc.

```
                                      : eConsumed_(0.0), eChassis_(0.0),
      eLineCard_(0.0), ePort_(0.0), eSimEnd_(0.0),
      eDVFS_enabled_(0), eDNS_enabled_(0), eDNS_delay_(0.0),
      eEnabled_(0), eCurrentRate_(0.0), eActivePorts_(0),
      eSimDuration_(0.0), classifier_(NULL), energytimer_(this)
{
                bind("eConsumed_", &eConsumed_);
                bind("eChassis_", &eChassis_);
                bind("eLineCard_", &eLineCard_);
                bind("ePort_", &ePort_);
                bind("eSimEnd_", &eSimEnd_);
                bind("eDVFS_enabled_", &eDVFS_enabled_);
      /* ON when DVFS is enabled */
                bind("eDNS_enabled_", &eDNS_enabled_);
      /* ON when DNS is enabled */
                bind("eDNS_delay_", &eDNS_delay_);
}
```

4.50.2.2 SwitchEnergyModel::∼SwitchEnergyModel ( ) [virtual]

Definition at line 28 of file switchenergymodel.cc.

```
{
}
```

4.50.3 Member Function Documentation

4.50.3.1 int SwitchEnergyModel::command (int *argc,* const char ∗const ∗ *argv*) [virtual]

Definition at line 72 of file switchenergymodel.cc.

```
{
                if (argc == 2) {
                                if (strcmp(argv[1], "start") == 0) {
                                                start();
                                                return (TCL_OK);
                                }
                                if (strcmp(argv[1], "stop") == 0) {
                                                stop();
                                                return (TCL_OK);
                                }
                }
                return (SwitchEnergyModel::command(argc, argv));
}
```

4.50.3.2 double SwitchEnergyModel::computeCurrentRate ( ) [protected]

Definition at line 49 of file switchenergymodel.cc.

```
{
                eCurrentRate_ = eChassis_ + eLineCard_ +
     eActivePorts_*ePort_;

                return eCurrentRate_;
}
```

4.50.3.3 void SwitchEnergyModel::setClassifier (Classifier ∗ *classifier*) [inline]

Definition at line 32 of file switchenergymodel.h.

```
{classifier_ = classifier;};
```

4.50.3.4 void SwitchEnergyModel::start ( )

Definition at line 32 of file switchenergymodel.cc.

```
{
                eEnabled_ = 1;
                eLastSample_ = Scheduler::instance().clock();
                eSimDuration_ = eSimEnd_ - eLastSample_;

                if (classifier_) eActivePorts_ =
     classifier_->maxslot();

                if (eDNS_enabled_) eCurrentRate_ = 0.0;
                else computeCurrentRate();
}
```

4.50.3.5 void SwitchEnergyModel::stop ( )

Definition at line 44 of file switchenergymodel.cc.

```
45 {
46                 updateEnergy(0, 0);
47 }
```

4.50.3.6 void SwitchEnergyModel::timeout ( ) [virtual]

Definition at line 87 of file switchenergymodel.cc.

```
87                                {
88
89                 eConsumed_ += eCurrentRate_*(Scheduler::instance().clock() -
      eLastSample_)/3600; // update energy
90                 eCurrentRate_ = 0.0;
91                 eLastSample_ = Scheduler::instance().clock();
92 }
```

4.50.3.7 void SwitchEnergyModel::updateEnergy (int *curSlot,* int *nports*)

Definition at line 56 of file switchenergymodel.cc.

```
57 {
58                 if (eEnabled_ == 0) return;
59
60                 /* Compute energy spent since last call */
61                 if (nports != eActivePorts_) {
62                                 eConsumed_ += eCurrentRate_*(Scheduler::instance().
      clock() - eLastSample_)/3600;         // update energy
63                                 eActivePorts_ = nports;
      // update number of active ports
64                                 computeCurrentRate();
65                                 eLastSample_ = Scheduler::instance().clock();
66
67                                 /* if DNS is enabled start sleep-mode timer */
68                                 if ((eDNS_enabled_)&& (eDNS_delay_))
      energytimer_.resched(eDNS_delay_);
69                 }
70 }
```

4.50.4 Member Data Documentation

4.50.4.1 Classifier* SwitchEnergyModel::classifier_ [protected]

Definition at line 61 of file switchenergymodel.h.

4.50.4.2 int SwitchEnergyModel::eActivePorts_ [protected]

Definition at line 57 of file switchenergymodel.h.

4.50.4.3 double SwitchEnergyModel::eChassis_

Definition at line 40 of file switchenergymodel.h.

4.50.4.4 double SwitchEnergyModel::eConsumed_

Definition at line 38 of file switchenergymodel.h.

4.50.4.5 double SwitchEnergyModel::eCurrentRate_ [protected]

Definition at line 55 of file switchenergymodel.h.

4.50.4.6 double SwitchEnergyModel::eDNS_delay_

Definition at line 48 of file switchenergymodel.h.

4.50.4.7 int SwitchEnergyModel::eDNS_enabled_

Definition at line 47 of file switchenergymodel.h.

4.50.4.8 int SwitchEnergyModel::eDVFS_enabled_

Definition at line 46 of file switchenergymodel.h.

4.50.4.9 int SwitchEnergyModel::eEnabled_ [protected]

Definition at line 53 of file switchenergymodel.h.

4.50.4.10 double SwitchEnergyModel::eLastSample_ [protected]

Definition at line 56 of file switchenergymodel.h.

4.50.4.11 double SwitchEnergyModel::eLineCard_

Definition at line 41 of file switchenergymodel.h.

4.50.4.12 SwitchEnergyTimer SwitchEnergyModel::energytimer_ [protected]

Definition at line 62 of file switchenergymodel.h.

4.50.4.13 double SwitchEnergyModel::ePort_

Definition at line 42 of file switchenergymodel.h.

4.50.4.14 double SwitchEnergyModel::eSimDuration_ [protected]

Definition at line 59 of file switchenergymodel.h.

4.50.4.15 double SwitchEnergyModel::eSimEnd_

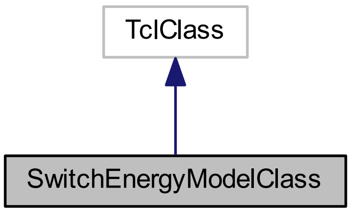
Definition at line 44 of file switchenergymodel.h.

The documentation for this class was generated from the following files:

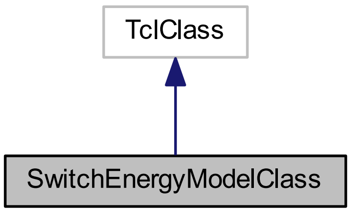
- switchenergymodel.h
- switchenergymodel.cc

4.51 SwitchEnergyModelClass Class Reference

Inheritance diagram for SwitchEnergyModelClass:



Collaboration diagram for SwitchEnergyModelClass:



Public Member Functions

- SwitchEnergyModelClass ()
- TclObject ∗ create (int argc, const char ∗const ∗argv)

4.51.1 Detailed Description

Definition at line 8 of file switchenergymodel.cc.

4.51.2 Constructor & Destructor Documentation

4.51.2.1 SwitchEnergyModelClass::SwitchEnergyModelClass ( ) [inline]

Definition at line 10 of file switchenergymodel.cc.

```
10 : TclClass("SwitchEnergyModel") {}
```

4.51.3 Member Function Documentation

4.51.3.1 TclObject∗ SwitchEnergyModelClass::create (int *argc*, const char ∗const ∗ *argv*) [inline]

Definition at line 11 of file switchenergymodel.cc.

```
                                                   {
                                return (new SwitchEnergyModel());
                }
```

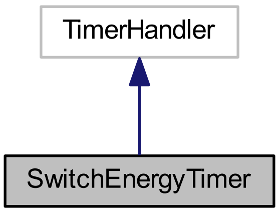
The documentation for this class was generated from the following file:

- switchenergymodel.cc

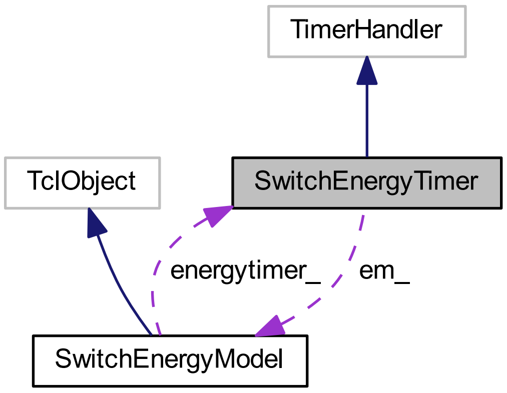
4.52 SwitchEnergyTimer Class Reference

`#include <switchenergymodel.h>`

Inheritance diagram for SwitchEnergyTimer:



Collaboration diagram for SwitchEnergyTimer:



Public Member Functions

- SwitchEnergyTimer (SwitchEnergyModel ∗em)

Protected Member Functions

- void expire (Event ∗)

Protected Attributes

- SwitchEnergyModel ∗ em_

4.52.1 Detailed Description

Definition at line 17 of file switchenergymodel.h.

4.52.2 Constructor & Destructor Documentation

4.52.2.1 SwitchEnergyTimer::SwitchEnergyTimer (**SwitchEnergyModel** ∗ *em*) [inline]

Definition at line 19 of file switchenergymodel.h.

```
: em_(em) {}
```

4.52.3 Member Function Documentation

4.52.3.1 void SwitchEnergyTimer::expire (Event ∗) [protected]

Definition at line 94 of file switchenergymodel.cc.

```
{
                em_->timeout();
}
```

4.52.4 Member Data Documentation

4.52.4.1 **SwitchEnergyModel**∗ SwitchEnergyTimer::em_ [protected]

Definition at line 22 of file switchenergymodel.h.

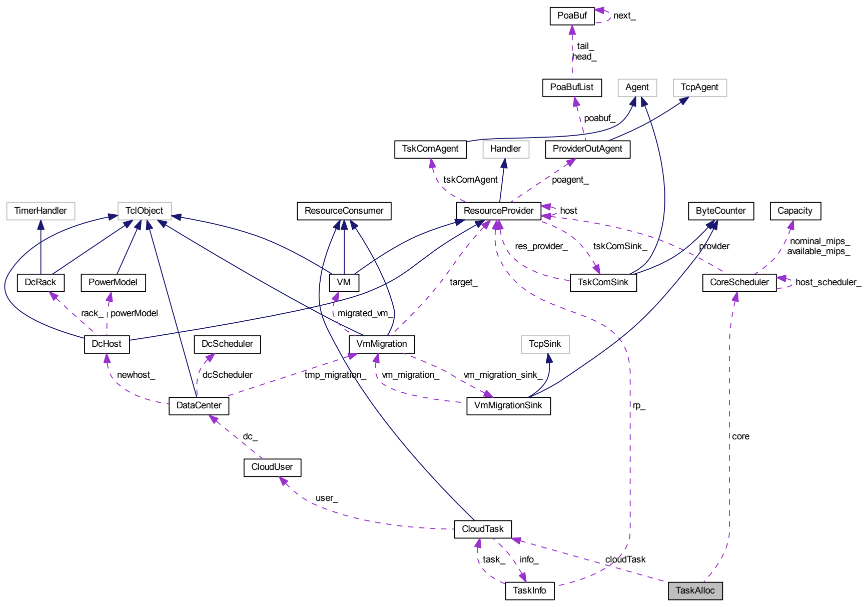
The documentation for this class was generated from the following files:

- switchenergymodel.h
- switchenergymodel.cc

4.53 TaskAlloc Class Reference

```
#include <taskalloc.h>
```

Collaboration diagram for TaskAlloc:



Public Member Functions

- TaskAlloc (CloudTask ∗ct, int rd, int cap)
- virtual ∼TaskAlloc ()
- double getMIPS ()
- double getDeadline ()
- void setExecTime (double execTime)
- void setCoreScheduler (CoreScheduler ∗cs)
- CoreScheduler ∗ getCoreScheduler ()
- bool operator== (const TaskAlloc &other) const
- void setComputingRate (double rate)
- double execTime ()
- void updateMIPS ()
- void removeAfterFailure ()
- void print ()

Public Attributes

- CloudTask ∗ cloudTask
- CoreScheduler ∗ core
- double executedSince_

Private Attributes

- int rd
- int cap

4.53.1 Detailed Description

Definition at line 14 of file taskalloc.h.

4.53.2 Constructor & Destructor Documentation

4.53.2.1 TaskAlloc::TaskAlloc (CloudTask ∗ *ct,* int *rd,* int *cap*)

Definition at line 16 of file taskalloc.cc.

```
16                                                        :  core(NULL),
     executedSince_(0.0){
17                  this->cloudTask = ct;
18                  this->rd = rd;
19                  this->cap = cap;
20 }
```

4.53.2.2 TaskAlloc::∼TaskAlloc ( ) [virtual]

Definition at line 12 of file taskalloc.cc.

```
12                        {
13
14 }
```

4.53.3 Member Function Documentation

4.53.3.1 double TaskAlloc::execTime ( )

Definition at line 73 of file taskalloc.cc.

```
74 {
75                  if (cloudTask->res_demands.at(rd)->current_performance.at(
     cap)){
76                                  return ((double)(cloudTask->res_demands.at(
     rd)->capacity.at(cap))/cloudTask->res_demands.at(rd)->current_performance.at(
     cap));
77                  }
78                  else{
79                                  return DBL_MAX;
80                  }
81 }
```

4.53.3.2 CoreScheduler ∗ TaskAlloc::getCoreScheduler ( )

Definition at line 31 of file taskalloc.cc.

```
31                                          {
32                  return this->core;
33 }
```

4.53.3.3 double TaskAlloc::getDeadline ( )

Definition at line 24 of file taskalloc.cc.

```
                        {
        return cloudTask->getDeadline();
}
```

4.53.3.4 double TaskAlloc::getMIPS ( )

Definition at line 21 of file taskalloc.cc.

```
                    {
        return cloudTask->getMIPS(rd,cap);
}
```

4.53.3.5 bool TaskAlloc::operator== (const TaskAlloc & *other*) const

Definition at line 35 of file taskalloc.cc.

```
                                                {
        if(cloudTask==other.cloudTask && rd == other.
    rd && cap == other.cap){
                return true;
        } else {
                return false;
        }
}
```

4.53.3.6 void TaskAlloc::print ( )

Definition at line 50 of file taskalloc.cc.

```
                {
        std::cerr << "id "<< cloudTask->id_ << " "<< rd << " ,c: " <<
    cap << " mips left: " << getMIPS() << "\texec since: " <<
    executedSince_;
}
```

4.53.3.7 void TaskAlloc::removeAfterFailure ( )

4.53.3.8 void TaskAlloc::setComputingRate (double *rate*)

Definition at line 43 of file taskalloc.cc.

```
{
        /* update what has already been computed */
        updateMIPS();
        cloudTask->res_demands.at(rd)->current_performance.at(
    cap)= rate;
}
```

4.53.3.9 void TaskAlloc::setCoreScheduler (CoreScheduler * *cs*)

Definition at line 28 of file taskalloc.cc.

```
                                                {
                this->core=cs;
}
```

4.53.3.10 void TaskAlloc::setExecTime (double *execTime*) [inline]

Definition at line 28 of file taskalloc.h.

```
{executedSince_ = execTime;};
```

4.53.3.11 void TaskAlloc::updateMIPS ( )

Definition at line 54 of file taskalloc.cc.

```
{
                ResDemand* res_dem = cloudTask->res_demands.at(
    rd);

                double operationsComputed = (res_dem)->current_performance.at(
    cap)*(Scheduler::instance().clock() - executedSince_);

                if((res_dem)->capacity.at(cap) > operationsComputed){
                                (res_dem)->capacity.at(cap) -= operationsComputed;
                } else {
                                (res_dem)->capacity.at(cap) = 0;
                }
//              std::cout << "--\t Task: " << this->cloudTask->id_ << " MIPS: " << getMIPS() << "\n";


                executedSince_ = Scheduler::instance().clock();
}
```

4.53.4 Member Data Documentation

4.53.4.1 int TaskAlloc::cap [private]

capacity

Definition at line 40 of file taskalloc.h.

4.53.4.2 CloudTask* TaskAlloc::cloudTask

Definition at line 17 of file taskalloc.h.

4.53.4.3 CoreScheduler* TaskAlloc::core

Definition at line 19 of file taskalloc.h.

4.53.4.4 double TaskAlloc::executedSince_

last time instance of task execution

Definition at line 20 of file taskalloc.h.

4.53.4.5 int TaskAlloc::rd [private]

resource demand

Definition at line 39 of file taskalloc.h.

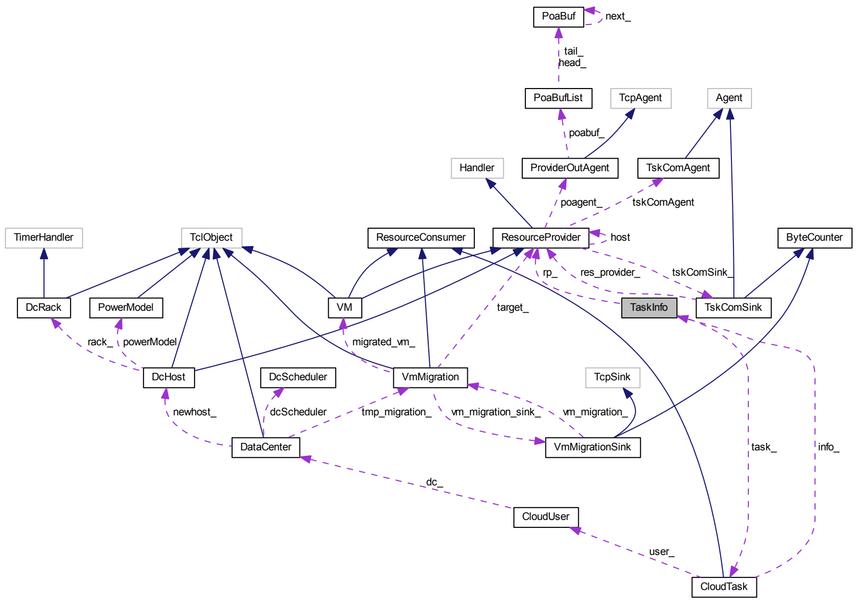
The documentation for this class was generated from the following files:

- taskalloc.h
- taskalloc.cc

4.54 TaskInfo Class Reference

```
#include <taskinfo.h>
```

Collaboration diagram for TaskInfo:



Public Member Functions

- TaskInfo (CloudTask ∗ct, double release_time, double due_time)
- virtual ∼TaskInfo ()
- CloudTask ∗ getTask ()
- void deleteTask ()
- double getReleaseTime ()
- double getDueTime ()
- double getServerFinishTime ()
- double getDcExitTime ()
- ResourceProvider ∗ getResourceProvider ()
- void setResourceProvider (ResourceProvider ∗rp)
- int getTaskId ()
- void setServerFinishTime (double time)
- void finalizeDcExitTime (double time)

Protected Attributes

- CloudTask ∗ task_
- int task_id_
- double release_time_
- double due_time_
- double server_finish_time_
- double dc_exit_time_
- ResourceProvider ∗ rp_

4.54.1 Detailed Description

Definition at line 13 of file taskinfo.h.

4.54.2 Constructor & Destructor Documentation

4.54.2.1 TaskInfo::TaskInfo (CloudTask ∗ *ct*, double *release_time*, double *due_time*)

Definition at line 10 of file taskinfo.cc.

```
10                                                                             :
11 task_(ct),task_id_(ct->id_), release_time_(release_time),
      due_time_(due_time), server_finish_time_(-1),
      dc_exit_time_(-1) {
12
13
14 }
```

4.54.2.2 TaskInfo::∼TaskInfo ( ) [virtual]

Definition at line 16 of file taskinfo.cc.

```
16                       {
17
18 }
```

4.54.3 Member Function Documentation

4.54.3.1 void TaskInfo::deleteTask ( )

Definition at line 23 of file taskinfo.cc.

```
23                            {
24                   delete task_;
25                   task_ = NULL;
26 }
```

4.54.3.2 void TaskInfo::finalizeDcExitTime (double *time*)

Definition at line 45 of file taskinfo.cc.

```
                                            {
                dc_exit_time_ = time;
}
```

4.54.3.3 double TaskInfo::getDcExitTime ( )

Definition at line 36 of file taskinfo.cc.

```
                              {
                return dc_exit_time_;
}
```

4.54.3.4 double TaskInfo::getDueTime ( )

Definition at line 30 of file taskinfo.cc.

```
                           {
                return due_time_;
}
```

4.54.3.5 double TaskInfo::getReleaseTime ( )

Definition at line 27 of file taskinfo.cc.

```
                               {
                return release_time_;
}
```

4.54.3.6 ResourceProvider ∗ TaskInfo::getResourceProvider ( )

Definition at line 49 of file taskinfo.cc.

```
                                               {
                return rp_;
}
```

4.54.3.7 double TaskInfo::getServerFinishTime ( )

Definition at line 33 of file taskinfo.cc.

```
                                    {
                return server_finish_time_;
}
```

4.54.3.8 CloudTask ∗ TaskInfo::getTask ( )

Definition at line 20 of file taskinfo.cc.

```
                        {
                return task_;
}
```

4.54.3.9 int TaskInfo::getTaskId ( )

Definition at line 39 of file taskinfo.cc.

```
                    {
                return task_id_;
}
```

4.54.3.10 void TaskInfo::setResourceProvider (ResourceProvider ∗ *rp*)

Definition at line 53 of file taskinfo.cc.

```
                                                {
                rp_ = rp;
}
```

4.54.3.11 void TaskInfo::setServerFinishTime (double *time*)

Definition at line 42 of file taskinfo.cc.

```
                                        {
                server_finish_time_ = time;
}
```

4.54.4 Member Data Documentation

4.54.4.1 double TaskInfo::dc_exit_time_ [protected]

Definition at line 35 of file taskinfo.h.

4.54.4.2 double TaskInfo::due_time_ [protected]

Definition at line 33 of file taskinfo.h.

4.54.4.3 double TaskInfo::release_time_ [protected]

Definition at line 32 of file taskinfo.h.

4.54.4.4 ResourceProvider∗ TaskInfo::rp_ [protected]

Definition at line 36 of file taskinfo.h.

4.54.4.5 double TaskInfo::server_finish_time_ [protected]

Definition at line 34 of file taskinfo.h.

4.54.4.6 CloudTask* TaskInfo::task_ [protected]

Definition at line 30 of file taskinfo.h.

4.54.4.7 int TaskInfo::task_id_ [protected]

Definition at line 31 of file taskinfo.h.

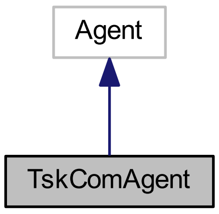
The documentation for this class was generated from the following files:

- taskinfo.h
- taskinfo.cc

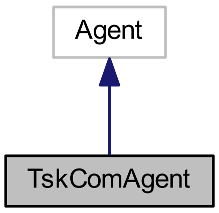
4.55 TskComAgent Class Reference

`#include <tskagent.h>`

Inheritance diagram for TskComAgent:



Collaboration diagram for TskComAgent:



Public Member Functions

- TskComAgent ()
- TskComAgent (packet_t)
- virtual void sendmsg (int nbytes, void ∗pTaskObj, const char ∗flags=0)
- virtual void sendmsg (int nbytes, AppData ∗data, void ∗pTaskObj, const char ∗flags=0)
- virtual void recv (Packet ∗pkt, Handler ∗)
- virtual int command (int argc, const char ∗const ∗argv)

Protected Attributes

- int seqno_

4.55.1 Detailed Description

Definition at line 17 of file tskagent.h.

4.55.2 Constructor & Destructor Documentation

4.55.2.1 TskComAgent::TskComAgent ( )

Definition at line 29 of file tskagent.cc.

```
                              : Agent(PT_UDP), seqno_(-1)
{
                bind("packetSize_", &size_);
}
```

4.55.2.2 TskComAgent::TskComAgent (packet_t *type*)

Definition at line 34 of file tskagent.cc.

```
                                          : Agent(type)
{
                bind("packetSize_", &size_);
}
```

4.55.3 Member Function Documentation

4.55.3.1 int TskComAgent::command (int *argc,* const char ∗const ∗ *argv*) [virtual]

Definition at line 126 of file tskagent.cc.

```
{
                //              Tcl& tcl = Tcl::instance();
                //              if (argc == 4) {
                //                              if (strcmp(argv[1], "send") == 0) {
                //                                              PacketData* data = new PacketData(1 +
      strlen(argv[3]));
                //                                              strcpy((char*)data->data(), argv[3]);
                //                                              sendmsg(atoi(argv[2]), data, 0);
                //                                              return (TCL_OK);
                //                              }
                //              } else if (argc == 5) {
                //                              if (strcmp(argv[1], "sendmsg") == 0) {
                //                                              PacketData* data = new PacketData(1 +
      strlen(argv[3]));
                //                                              strcpy((char*)data->data(), argv[3]);
                //                                              sendmsg(atoi(argv[2]), data, 0, argv[4]);
                //                                              return (TCL_OK);
                //                              }
                //              }

                return (Agent::command(argc, argv));
}
```

4.55.3.2 void TskComAgent::recv (Packet * *pkt*, Handler *) [virtual]

Definition at line 100 of file tskagent.cc.

```
{
                if (app_ ) {
                                // If an application is attached, pass the data to the app
                                hdr_cmn* h = hdr_cmn::access(pkt);
                                app_->process_data(h->size(), pkt->userdata());
                } else if (pkt->userdata() && pkt->userdata()->type() == PACKET_DATA) {
                                // otherwise if it's just PacketData, pass it to Tcl
                                //
                                // Note that a Tcl procedure Agent/Udp recv {from data}
                                // needs to be defined.  For example,
                                //
                                // Agent/Udp instproc recv {from data} {puts data}

                                PacketData* data = (PacketData*)pkt->userdata();

                                hdr_ip* iph = hdr_ip::access(pkt);
                                Tcl& tcl = Tcl::instance();
                                tcl.evalf("%s process_data %d {%s}", name(),
                                                                iph->src_.addr_ >> Address::instance().
   NodeShift_[1],
                                                                data->data());
                }
                Packet::free(pkt);
}
```

4.55.3.3 virtual void TskComAgent::sendmsg (int *nbytes*, void * *pTaskObj*, const char * *flags* = 0) [inline], [virtual]

Definition at line 21 of file tskagent.h.

```
                {
                                sendmsg(nbytes, NULL, pTaskObj, flags);
                }
```

4.55.3.4 void TskComAgent::sendmsg (int *nbytes*, AppData * *data*, void * *pTaskObj*, const char * *flags* = 0) [virtual]

Definition at line 39 of file tskagent.cc.

```
{
                Packet *p;
                int n;

                assert (size_ > 0);

                n = nbytes / size_;
                int initialseqno = seqno_;

                if (nbytes == -1) {
                                printf("Error: sendmsg() for Tsk should not be -1\n");
                                return;
                }

                // If they are sending data, then it must fit within a single packet.
                if (data && nbytes > size_) {
                                printf("Error: data greater than maximum Tsk packet size\n");
                                return;
                }

                double          local_time = Scheduler::instance().clock();
                while (n-- > 0) {
                                p = allocpkt();
                                hdr_cmn::access(p)->size() = size_;
                                hdr_cmn::access(p)->pt_obj_addr() = 0;
                                if (initialseqno == seqno_){
                                                /* Add pointer to TaskObj for the first packet in the bulk
   */
                                                hdr_cmn::access(p)->pt_obj_addr() = pTaskObj;
```

```
                        }
                        hdr_rtp* rh = hdr_rtp::access(p);
                        rh->flags() = 0;
                        rh->seqno() = ++seqno_;
                        hdr_cmn::access(p)->timestamp() =
                                                (u_int32_t)(
    SAMPLERATE*local_time);
                        if (flags && (0 ==strcmp(flags, "NEW_BURST")))
                                rh->flags() |= RTP_M;
                        p->setdata(data);
                        target_->recv(p);
                }
                n = nbytes % size_;
                if (n > 0) {
                        p = allocpkt();
                        hdr_cmn::access(p)->size() = n;
                        hdr_cmn::access(p)->pt_obj_addr() = 0;
                        if (initialseqno == seqno_){
                                /* Add pointer to TaskObj for the first packet in the bulk
    */
                                hdr_cmn::access(p)->pt_obj_addr() = pTaskObj;
                        }
                        hdr_rtp* rh = hdr_rtp::access(p);
                        rh->flags() = 0;
                        rh->seqno() = ++seqno_;
                        hdr_cmn::access(p)->timestamp() =
                                                (u_int32_t)(
    SAMPLERATE*local_time);
                        if (flags && (0 == strcmp(flags, "NEW_BURST")))
                                rh->flags() |= RTP_M;
                        p->setdata(data);
                        target_->recv(p);
                }
                idle();
}
```

4.55.4 Member Data Documentation

4.55.4.1 int TskComAgent::seqno_ [protected]

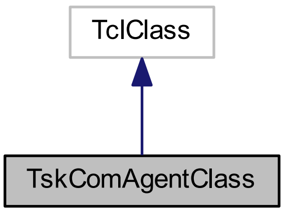
Definition at line 29 of file tskagent.h.

The documentation for this class was generated from the following files:

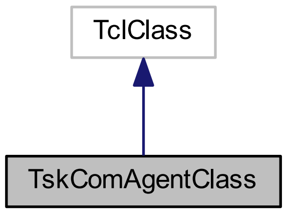
- tskagent.h
- tskagent.cc

4.56 TskComAgentClass Class Reference

Inheritance diagram for TskComAgentClass:



Collaboration diagram for TskComAgentClass:



Public Member Functions

- TskComAgentClass ()
- TclObject ∗ create (int, const char ∗const ∗)

4.56.1 Detailed Description

Definition at line 21 of file tskagent.cc.

4.56.2 Constructor & Destructor Documentation

4.56.2.1 TskComAgentClass::TskComAgentClass ( ) [inline]

Definition at line 23 of file tskagent.cc.

```
: TclClass("Agent/TskComAgent") {}
```

4.56.3 Member Function Documentation

4.56.3.1 TclObject∗ TskComAgentClass::create (int , const char ∗const ∗) [inline]

Definition at line 24 of file tskagent.cc.

```
                                                   {
                                return (new TskComAgent());
                }
```

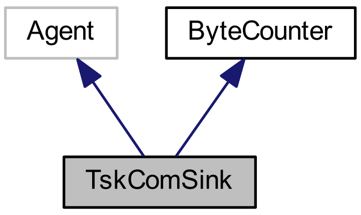
The documentation for this class was generated from the following file:

- tskagent.cc

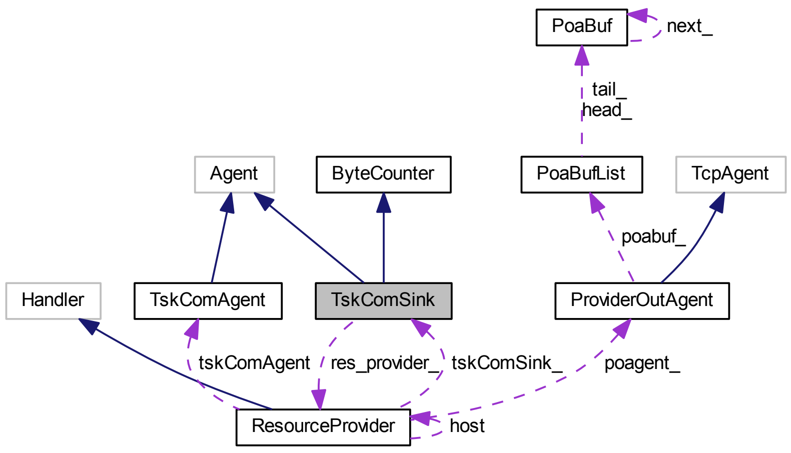
4.57 TskComSink Class Reference

`#include <tskcomsink.h>`

Inheritance diagram for TskComSink:



Collaboration diagram for TskComSink:



Public Member Functions

- TskComSink ()
- virtual ∼TskComSink ()
- void addResourceProvider (ResourceProvider ∗newrp)
- virtual int command (int argc, const char ∗const ∗argv)
- virtual void recv (Packet ∗pkt, Handler ∗)

Protected Attributes

- int nlost_
- int npkts_
- int expected_
- int bytes_
- int seqno_
- double last_packet_time_
- ResourceProvider ∗ res_provider_

4.57.1 Detailed Description

Definition at line 19 of file tskcomsink.h.

4.57.2 Constructor & Destructor Documentation

4.57.2.1 TskComSink::TskComSink ( )

Definition at line 28 of file tskcomsink.cc.

```
                        : Agent(PT_NTYPE)
{
                bytes_ = 0;
                bytes_since_ = 0;
                nlost_ = 0;
                npkts_ = 0;
                expected_ = -1;
                last_packet_time_ = 0.;
                last_bytes_since_ = 0.;
                seqno_ = 0;
                bind("nlost_", &nlost_);
                bind("npkts_", &npkts_);
                bind("bytes_", &bytes_);
                bind("lastPktTime_", &last_packet_time_);
                bind("expected_", &expected_);
}
```

4.57.2.2 TskComSink::∼TskComSink ( ) [virtual]

Definition at line 45 of file tskcomsink.cc.

```
{
                res_provider_ = NULL;
}
```

4.57.3 Member Function Documentation

4.57.3.1 void TskComSink::addResourceProvider (ResourceProvider ∗ *newrp*)

Definition at line 70 of file tskcomsink.cc.

```
{
                res_provider_  = newrp;
                res_provider_->setTskComSink(this);
}
```

4.57.3.2 int TskComSink::command (int *argc*, const char *const * *argv*) [virtual]

Definition at line 80 of file tskcomsink.cc.

```
{
        if (argc == 2) {
                if (strcmp(argv[1], "clear") == 0) {
                        expected_ = -1;
                        return (TCL_OK);
                }
        }
        if (argc == 3) {
                if (strcmp(argv[1], "connect-resprovider") == 0) {
                        ResourceProvider *hst = dynamic_cast<ResourceProvider*> (TclObject::lookup(argv[2]));
                        if(hst){
                                addResourceProvider(hst);
                                return (TCL_OK);
                        }
                        return (TCL_ERROR);
                }
        }

        return (Agent::command(argc, argv));
}
```

4.57.3.3 void TskComSink::recv (Packet * *pkt*, Handler *) [virtual]

Definition at line 50 of file tskcomsink.cc.

```
{
        /* Get TskObject and start its execution */
        ResourceConsumer *recvTskObj = (ResourceConsumer*) hdr_cmn::access(pkt)->pt_obj_addr();
//      std::cerr << "Pointer recieved:" << recvTskObj << "\n";

        if (recvTskObj) { /* Valid pointer and can be executed */
//              std::cerr << "Task id:" << ((CloudTask*)recvTskObj)->id_ << "\n";
                if (res_provider_) res_provider_->recv(recvTskObj);
                else printf("Error: task is received but no ResourceProvider is attached\n");
        }

        bytes_ += hdr_cmn::access(pkt)->size();
        bytes_since_ += hdr_cmn::access(pkt)->size();
        ++npkts_;

        last_packet_time_ = Scheduler::instance().clock();
        Packet::free(pkt);
}
```

4.57.4 Member Data Documentation

4.57.4.1 int TskComSink::bytes_ [protected]

Definition at line 31 of file tskcomsink.h.

4.57.4.2 int TskComSink::expected_ [protected]

Definition at line 30 of file tskcomsink.h.

4.57.4.3 double TskComSink::last_packet_time_ [protected]

Definition at line 33 of file tskcomsink.h.

4.57.4.4 int TskComSink::nlost_ [protected]

Definition at line 28 of file tskcomsink.h.

4.57.4.5 int TskComSink::npkts_ [protected]

Definition at line 29 of file tskcomsink.h.

4.57.4.6 ResourceProvider* TskComSink::res_provider_ [protected]

Definition at line 36 of file tskcomsink.h.

4.57.4.7 int TskComSink::seqno_ [protected]

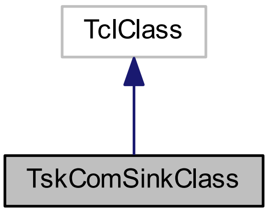
Definition at line 32 of file tskcomsink.h.

The documentation for this class was generated from the following files:

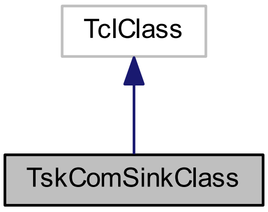
- tskcomsink.h
- tskcomsink.cc

4.58 TskComSinkClass Class Reference

Inheritance diagram for TskComSinkClass:



Collaboration diagram for TskComSinkClass:



Public Member Functions

- TskComSinkClass ()
- TclObject ∗ create (int, const char ∗const ∗)

4.58.1 Detailed Description

Definition at line 20 of file tskcomsink.cc.

4.58.2 Constructor & Destructor Documentation

4.58.2.1 TskComSinkClass::TskComSinkClass ( ) [inline]

Definition at line 22 of file tskcomsink.cc.

```
: TclClass("Agent/TskComSink") {}
```

4.58.3 Member Function Documentation

4.58.3.1 TclObject∗ TskComSinkClass::create (int , const char ∗const ∗) [inline]

Definition at line 23 of file tskcomsink.cc.

```
                                                  {
                                   return (new TskComSink());
                  }
```

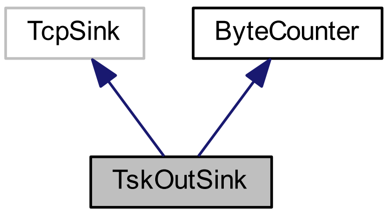
The documentation for this class was generated from the following file:

- tskcomsink.cc

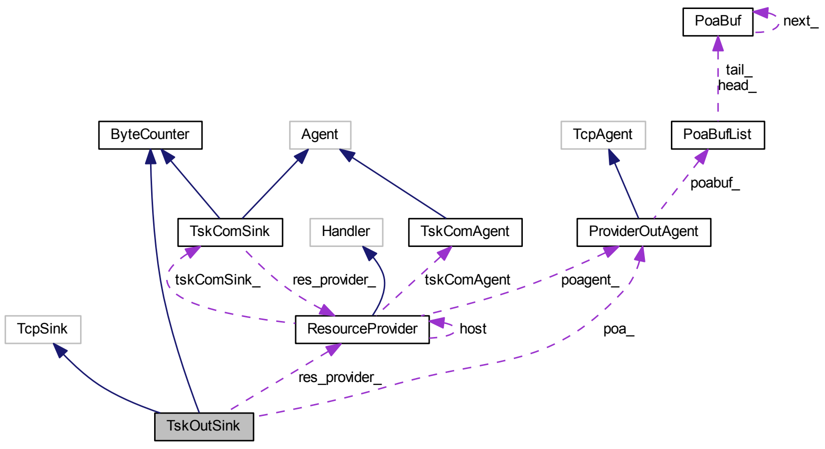
4.59 TskOutSink Class Reference

`#include <tskoutsink.h>`

Inheritance diagram for TskOutSink:



Collaboration diagram for TskOutSink:



Public Member Functions

- TskOutSink ()
- virtual ∼TskOutSink ()
- void addResourceProvider (ResourceProvider ∗newrp)
- virtual int command (int argc, const char ∗const ∗argv)
- virtual void recv (Packet ∗pkt, Handler ∗)

Protected Attributes

- int nlost_
- int npkts_
- int expected_
- int bytes_
- int seqno_
- double last_packet_time_
- ProviderOutAgent ∗ poa_
- ResourceProvider ∗ res_provider_

4.59.1 Detailed Description

Definition at line 20 of file tskoutsink.h.

4.59.2 Constructor & Destructor Documentation

4.59.2.1 TskOutSink::TskOutSink ( )

Definition at line 30 of file tskoutsink.cc.

```
30                          : TcpSink(new Acker()), poa_(NULL)
31 {
32 
33 }
```

4.59.2.2 TskOutSink::∼TskOutSink ( ) [virtual]

Definition at line 35 of file tskoutsink.cc.

```
{
                res_provider_ = NULL;
}
```

4.59.3 Member Function Documentation

4.59.3.1 void TskOutSink::addResourceProvider (ResourceProvider ∗ *newrp*)

4.59.3.2 int TskOutSink::command (int *argc,* const char ∗const ∗ *argv*) [virtual]

Definition at line 54 of file tskoutsink.cc.

```
{
                if (argc == 2) {
                                if (strcmp(argv[1], "clear") == 0) {
                                                expected_ = -1;
                                                return (TCL_OK);
                                }
                }
                if (argc == 3) {
                                if (strcmp(argv[1], "connect-tskoutagent") == 0) {
                                                ProviderOutAgent *poa =(
    ProviderOutAgent*)(TclObject::lookup(argv[2]));
                                                if(poa){

    poa_ = poa;
                                                                return (TCL_OK);
                                                }
                                                return (TCL_ERROR);
                                }
                }

                return (Agent::command(argc, argv));
}
```

4.59.3.3 void TskOutSink::recv (Packet ∗ *pkt,* Handler ∗ *h*) [virtual]

Definition at line 40 of file tskoutsink.cc.

```
{
                /* Get TskObject and start its execution */
                CloudTask *recvTskObj = (CloudTask*)hdr_cmn::access(pkt)->pt_obj_addr();
                if (recvTskObj) { /* Valid pointer and can be executed */
//                              std::cout << "Task id:" << recvTskObj->id_ << "exits the DC at: "<<
     Scheduler::instance().clock() <<"\n";
                                recvTskObj->info_->finalizeDcExitTime(
    Scheduler::instance().clock());
                                poa_->tryToSend();
                                recvTskObj->info_->getResourceProvider()->
    getRootHost()->updateEnergyAndConsumption();
                }
                TcpSink::recv(pkt,h);
}
```

4.59.4 Member Data Documentation

4.59.4.1 int TskOutSink::bytes_ [protected]

Definition at line 32 of file tskoutsink.h.

4.59.4.2 int TskOutSink::expected_ `[protected]`

Definition at line 31 of file tskoutsink.h.

4.59.4.3 double TskOutSink::last_packet_time_ `[protected]`

Definition at line 34 of file tskoutsink.h.

4.59.4.4 int TskOutSink::nlost_ `[protected]`

Definition at line 29 of file tskoutsink.h.

4.59.4.5 int TskOutSink::npkts_ `[protected]`

Definition at line 30 of file tskoutsink.h.

4.59.4.6 ProviderOutAgent* TskOutSink::poa_ `[protected]`

Definition at line 35 of file tskoutsink.h.

4.59.4.7 ResourceProvider* TskOutSink::res_provider_ `[protected]`

Definition at line 37 of file tskoutsink.h.

4.59.4.8 int TskOutSink::seqno_ `[protected]`

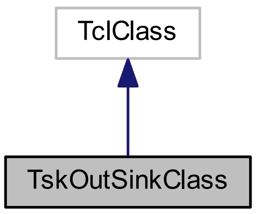
Definition at line 33 of file tskoutsink.h.

The documentation for this class was generated from the following files:

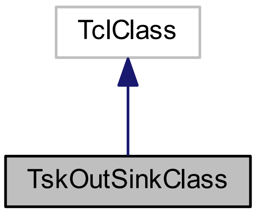
- tskoutsink.h
- tskoutsink.cc

4.60 TskOutSinkClass Class Reference

Inheritance diagram for TskOutSinkClass:



Collaboration diagram for TskOutSinkClass:



Public Member Functions

- TskOutSinkClass ()
- TclObject ∗ create (int, const char ∗const ∗)

4.60.1 Detailed Description

Definition at line 22 of file tskoutsink.cc.

4.60.2 Constructor & Destructor Documentation

4.60.2.1 TskOutSinkClass::TskOutSinkClass ( ) [inline]

Definition at line 24 of file tskoutsink.cc.

```
24 : TclClass("Agent/TCPSink/TskOutSink") {}
```

4.60.3 Member Function Documentation

4.60.3.1 TclObject∗ TskOutSinkClass::create (int , const char ∗const ∗) [inline]

Definition at line 25 of file tskoutsink.cc.

```
                                                  {
                                return (new TskOutSink());
                }
```

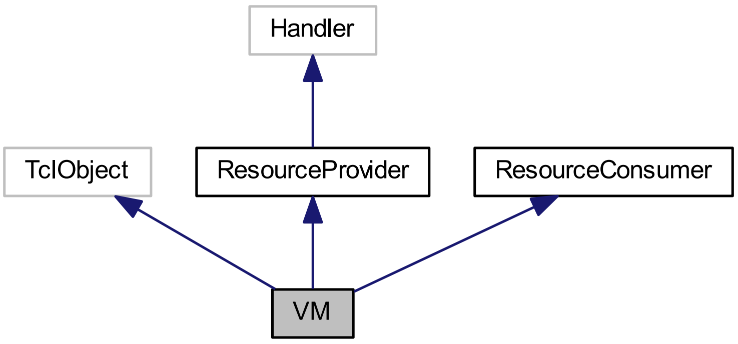
The documentation for this class was generated from the following file:

- tskoutsink.cc

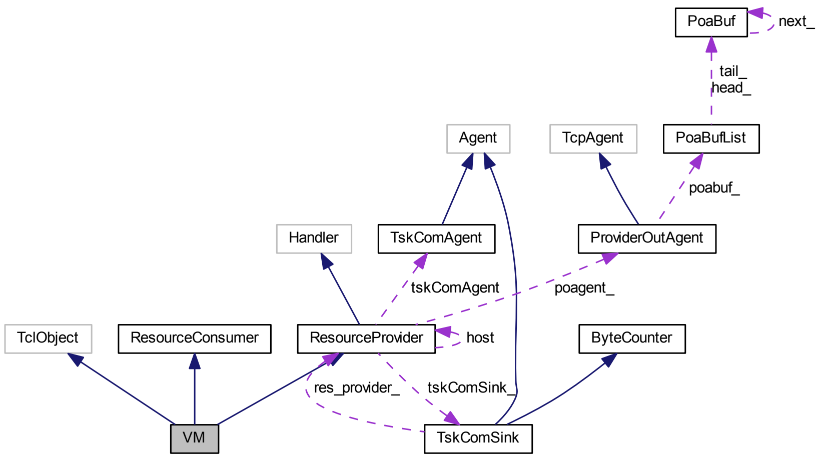
4.61 VM Class Reference

```
#include <vm.h>
```

Inheritance diagram for VM:



Collaboration diagram for VM:



Public Member Functions

- VM ()
- virtual ∼VM ()
- virtual void print ()
- virtual void printTasklist ()
- virtual int command (int argc, const char ∗const ∗argv)
- virtual void updateMIPS ()
- virtual void addResource (DcResource ∗res)
- vm_state getVmState ()
- void setHost (ResourceProvider ∗newHost)

Protected Member Functions

- virtual void updateEnergyAndConsumption ()

Protected Attributes

- vm_state state

Additional Inherited Members

4.61.1 Detailed Description

Definition at line 29 of file vm.h.

4.61.2 Constructor & Destructor Documentation

4.61.2.1 VM::VM ( )

Definition at line 18 of file vm.cc.

```
         {

                /* It should be always false for VMs */
                isTask = false;
                /* It should be always true for VMs */
                isVM = true;

                bind("id_", &id_);
                bind("ntasks_", &ntasks_);
                bind("currentLoad_", &currentLoad_);
                bind("currentLoadMem_", &currentLoadMem_);
                bind("currentLoadStor_", &currentLoadStor_);
                bind("tskFailed_", &tskFailed_);
                bind("eDVFS_enabled_", &eDVFS_enabled_);
    /* ON when DVFS is enabled */
                state = Ready;

}
```

4.61.2.2 VM::∼VM ( ) [virtual]

Definition at line 36 of file vm.cc.

```
          {

}
```

4.61.3 Member Function Documentation

4.61.3.1 void VM::addResource (DcResource ∗ *res*) [virtual]

Reimplemented from ResourceProvider.

Definition at line 49 of file vm.cc.

```
                                     {

                ResourceProvider::addResource(res);
                res_demands.push_back(new ResDemand(*res,res));

}
```

4.61.3.2 int VM::command (int *argc*, const char ∗const ∗ *argv*) [virtual]

Reimplemented from ResourceProvider.

Definition at line 56 of file vm.cc.

```
{

                if (argc == 2) {
                                if (strcmp(argv[1], "start") == 0) {

                                                state = Running;
                                                return (TCL_OK);
                                } else if (strcmp(argv[1], "stop") == 0) {

                                                state = Stopped;
                                                return (TCL_OK);
                                } else if (strcmp(argv[1], "print") == 0) {
                                                /* print general info */
                                                print();
                                                return (TCL_OK);
                                }
                }
                return (ResourceProvider::command(argc, argv));
}
```

4.61.3.3 vm_state VM::getVmState ( )

Definition at line 41 of file vm.cc.

```
{return state;};
```

4.61.3.4 void VM::print ( ) [virtual]

Implements ResourceProvider.

Definition at line 90 of file vm.cc.

```
               {
                std::cout << "VM:\t";
                std::cout << id_;
                std::cout << "\n";
                if(host!=NULL){
                                std::cout << "Hosted on" << host->id_;
                } else {
                                std::cout << "Not hosted";
                }
                std::cout << "\n";
                 std::cout << "Resources provisions:\n";
                 std::vector <std::vector<DcResource*> >::iterator iter_out;
                 for(iter_out = resource_list.begin(); iter_out!=
resource_list.end() ;iter_out++){
                                 std::vector <DcResource*>::iterator iter;
                                 for (iter = iter_out->begin(); iter!=iter_out->end(); iter++)
                                 {
                                                 (*iter)->print();
                                 }
                 }
                 std::cout << "Resources demands:\n";
                 std::vector<ResDemand*>::iterator iter_dem;
                 for(iter_dem = res_demands.begin(); iter_dem!=
res_demands.end() ;iter_dem++){
                                 (*iter_dem)->print();
                 }
                 std::cout << "\n";

}
```

4.61.3.5 void VM::printTasklist ( ) [virtual]

Reimplemented from ResourceProvider.

Definition at line 118 of file vm.cc.

```
                        {
                std::vector<CloudTask *>::iterator iter;
                std::cout <<"VM " <<this->id_ << "\n";

                ResourceProvider::printTasklist();
}
```

4.61.3.6 void VM::setHost (ResourceProvider * newHost)

Definition at line 43 of file vm.cc.

```
{host = newHost;};
```

4.61.3.7 void VM::updateEnergyAndConsumption ( ) [protected], [virtual]

Implements ResourceProvider.

Definition at line 81 of file vm.cc.

```
                                {
                if(host!= NULL){
                                host->updateEnergyAndConsumption();
                } else {
                                std::cerr << "ERROR: Task is allocated on an unallocated VM!\n";
                }
                return;
}
```

4.61.3.8 void VM::updateMIPS ( ) [virtual]

Definition at line 77 of file vm.cc.

```
                  {
                return;
}
```

4.61.4 Member Data Documentation

4.61.4.1 vm_state VM::state [protected]

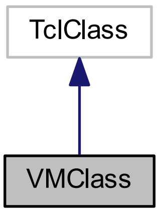
Definition at line 48 of file vm.h.

The documentation for this class was generated from the following files:

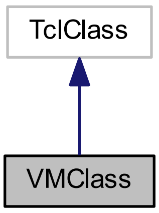
- vm.h
- vm.cc

4.62 VMClass Class Reference

Inheritance diagram for VMClass:



Collaboration diagram for VMClass:



Public Member Functions

- VMClass ()
- TclObject $*$ create (int argc, const char $*$const $*$argv)

4.62.1 Detailed Description

Definition at line 10 of file vm.cc.

4.62.2 Constructor & Destructor Documentation

4.62.2.1 VMClass::VMClass ( ) [inline]

Definition at line 12 of file vm.cc.

```
12 : TclClass("VM") {}
```

4.62.3 Member Function Documentation

4.62.3.1 TclObject∗ VMClass::create (int *argc*, const char ∗const ∗ *argv*) [inline]

Definition at line 13 of file vm.cc.

```
{
        return (new VM());
}
```

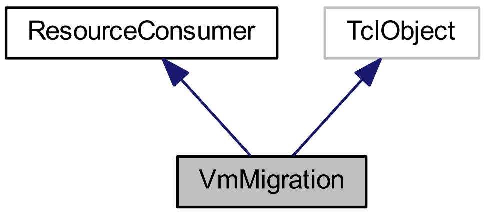
The documentation for this class was generated from the following file:

- vm.cc

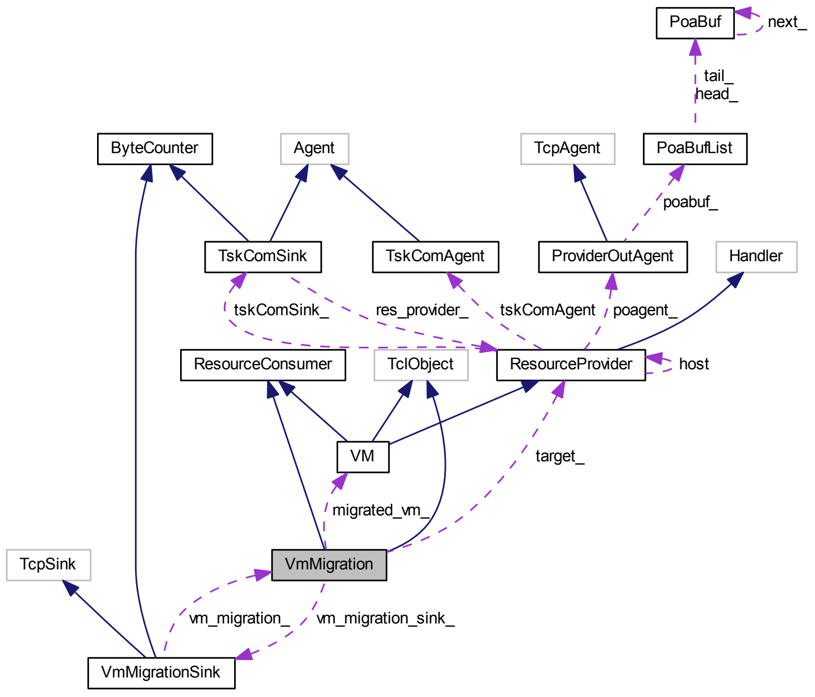
4.63 VmMigration Class Reference

`#include <vmmigration.h>`

Inheritance diagram for VmMigration:



Collaboration diagram for VmMigration:



Public Member Functions

- VmMigration ()
- void initalizeMigration (VM ∗vm, ResourceProvider ∗target)
- virtual ∼VmMigration ()
- virtual int command (int argc, const char ∗const ∗argv)
- void finalizeMigration ()
- void startMigration ()

Private Attributes

- VM ∗ migrated_vm_
- ResourceProvider ∗ target_
- VmMigrationSink ∗ vm_migration_sink_
- TcpAgent ∗ vm_migration_sender_
- int id_

Additional Inherited Members

4.63.1 Detailed Description

Definition at line 17 of file vmmigration.h.

4.63.2 Constructor & Destructor Documentation

4.63.2.1 VmMigration::VmMigration ( )

Definition at line 18 of file vmmigration.cc.

```
18                            {
19
20 }
```

4.63.2.2 VmMigration::∼VmMigration ( ) [virtual]

Definition at line 70 of file vmmigration.cc.

```
70                             {
71                  //  TODO: How about memory management:
72                  //              delete vm_migration_sender_;
73                  //              delete vm_migration_sink_;
74
75 }
```

4.63.3 Member Function Documentation

4.63.3.1 int VmMigration::command (int *argc,* const char ∗const ∗ *argv*) [virtual]

Definition at line 147 of file vmmigration.cc.

```
147                                                         {
148                 if (argc == 3) {
149                                 if (strcmp(argv[1], "set-sink") == 0) {
150                                                 VmMigrationSink *vms = dynamic_cast<
      VmMigrationSink*> (TclObject::lookup(argv[2]));
151                                                 if(vms){
152
      vm_migration_sink_ = vms;
153
      vm_migration_sink_->setVmMigration(this);
154                                                                 return (TCL_OK);
155                                                 }
156                                                 return (TCL_ERROR);
157                                 }               else if (strcmp(argv[1], "set-source") == 0) {
158                                                 TcpAgent *vma = dynamic_cast<TcpAgent*> (TcpAgent::lookup(
      argv[2]));
159                                                 if(vma){
160
      vm_migration_sender_=vma;
161                                                                 return (TCL_OK);
162                                                 }
163                                                 return (TCL_ERROR);
164                                 } else if (strcmp(argv[1], "set-id") == 0) {
165                                                 char* stat;
166                                                 int num = strtol(argv[2], &stat, 10);
167                                                 if (!*stat){
168                                                                 id_=num;
169                                                                 return (TCL_OK);
170                                                 }
171                                                 else {
172                                                                 return (TCL_ERROR);
173                                                 }
174                                 }
175                 }
176                 return TCL_ERROR;
177 }
```

4.63.3.2 void VmMigration::finalizeMigration ( )

Definition at line 96 of file vmmigration.cc.

```
96                               {
97              migrated_vm_->getHost()->updateEnergyAndConsumption();
98              target_->updateEnergyAndConsumption();
99
100              Tcl& tcl = Tcl::instance();
101
102              tcl.evalf("$hosts_(%d) detach-vm-mig-source $vmmigrationsource_(%d)", migrated_vm_->getHost()->id_,id_);
103              tcl.evalf("$hosts_(%d) detach-vm-mig-sink   $vmmigrationsink_(%d)", target_->id_,id_);
104
105              tcl.evalf("$ns detach-agent $servers_(%d) $vmmigrationsource_(%d)", migrated_vm_->getHost()->id_,id_);
106              tcl.evalf("$ns detach-agent $servers_(%d) $vmmigrationsink_(%d) ", target_->id_,id_);
107
108
109              target_->releaseAllocation(this);
110
111              ResourceProvider* source = migrated_vm_->getHost();
112              source->removeVM(migrated_vm_);
113              //              Re-attach tasks sinks
114              tcl.evalf("$ns detach-agent $servers_(%d) $vmtsksink_(%d) ",source->id_,migrated_vm_->id_);
115              tcl.evalf("$ns attach-agent $servers_(%d) $vmtsksink_(%d) ", target_->id_,migrated_vm_->id_);
116
117              //              Re-attach tasks com agents:
118              tcl.evalf("$ns detach-agent $switch_C1_([expr %d/($top(NServers)/$NTSwitches)]) $vmtskcomagnt_C_(%d)",source->id_,migrated_vm_->id_);
119              tcl.evalf("$ns attach-agent $switch_C1_([expr %d/($top(NServers)/$NTSwitches)]) $vmtskcomagnt_C_(%d)",target_->id_,migrated_vm_->id_);
120              //  Connect com agents with sinks:
121              tcl.evalf("$ns connect $vmtskcomagnt_C_(%d) $vmtsksink_(%d)", migrated_vm_->id_,migrated_vm_->id_);
122
123              //  Re-attach tasks output agent:
124              tcl.evalf("$ns detach-agent $servers_(%d) $vmtskoutputagent_(%d) ",source->id_,migrated_vm_->id_);
125              tcl.evalf("$ns attach-agent $servers_(%d) $vmtskoutputagent_(%d) ", target_->id_,migrated_vm_->id_);
126              //  Re-attach tasks output sink:
127              tcl.evalf("$ns detach-agent $switch_C1_([expr %d/($top(NServers)/$NTSwitches)]) $vmtskoutputsink_(%d)",source->id_,migrated_vm_->id_);
128              tcl.evalf("$ns attach-agent $switch_C1_([expr %d/($top(NServers)/$NTSwitches)]) $vmtskoutputsink_(%d)",target_->id_,migrated_vm_->id_);
129              //Connect:
130              tcl.evalf("$ns connect $vmtskoutputagent_(%d) $vmtskoutputsink_(%d)", migrated_vm_->id_,migrated_vm_->id_);
131
132
133              tcl.evalf("$vms_(%d) attach-agent $vmtskoutputagent_(%d)", migrated_vm_->id_,migrated_vm_->id_);
134
135              if(target_->addVM(migrated_vm_)){
136 //                                                      std::cerr << "Migrated VM successfully allocated on target.\n";
137              } else {
138                              std::cerr << "Error: Migration object NOT successfully allocated on target.\n";
139                              return;
140              }
141
142              //              Migration finalization finished.
143              //              IFF the migration object is no longer needed:
144              delete this; // Nothing related to migration object after this point!
145 }
```

4.63.3.3 void VmMigration::initalizeMigration (VM ∗ *vm,* ResourceProvider ∗ *target*)

Definition at line 22 of file vmmigration.cc.

```
                                                                        {

                migrated_vm_ = vm;
                target_ =  target;

                this->size_ = vm->size_;
                this->isTask = false;
                this->isVM = false;
                res_demands.clear();
                res_demands = std::vector<ResDemand *>(vm->
      res_demands.size(),NULL);

                std::vector <ResDemand*>::iterator iter;
                std::vector <ResDemand*>::iterator iter2;
                for (iter = vm->res_demands.begin(), iter2=
      res_demands.begin(); iter!=vm->res_demands.end(); iter++,iter2++)
                {
                                (*iter2)=new ResDemand(*(*iter),NULL);
                }


                // Create new migration sink and source;

                Tcl& tcl = Tcl::instance();
                // Create communication source and sink:
                tcl.evalf("set vmmigrationsource_($next_migration_id) [new Agent/TCP/ProvOutAgent]");
                tcl.evalf("set vmmigrationsink_($next_migration_id) [new Agent/TCPSink/VmMigrationSink]");

                //Attach source and sink in network topology
                tcl.evalf("$ns attach-agent $servers_(%d) $vmmigrationsource_($next_migration_id)",
      migrated_vm_->getHost()->id_);
                tcl.evalf("$ns attach-agent $servers_(%d) $vmmigrationsink_($next_migration_id)",
      target_->id_);
                tcl.evalf("$ns connect $vmmigrationsource_($next_migration_id)
       $vmmigrationsink_($next_migration_id)");


                //Attach source and sink to host to track network interface utilization
                tcl.evalf("$hosts_(%d) attach-vm-mig-source $vmmigrationsource_($next_migration_id)",
      migrated_vm_->getHost()->id_);
                tcl.evalf("$hosts_(%d) attach-vm-mig-sink   $vmmigrationsink_($next_migration_id)",
      target_->id_);



                //Set source and destination in migration object itself:
                tcl.evalf("$vmmigration_($next_migration_id) set-source
       $vmmigrationsource_($next_migration_id)");
                tcl.evalf("$vmmigration_($next_migration_id) set-sink
       $vmmigrationsink_($next_migration_id)");



                tcl.evalf("incr next_migration_id");

 }
```

4.63.3.4 void VmMigration::startMigration ( )

Definition at line 77 of file vmmigration.cc.

```
                                {
                migrated_vm_->getHost()->
      updateEnergyAndConsumption();
                target_->updateEnergyAndConsumption();

                if(target_->tryToAllocate(this)){
                                //                              std::cerr << "Migration object successfully
       allocated on target.\n";
                } else {
                                std::cerr << "Error: Migration object NOT successfully allocated on target.
      \n";
                                return;
                }

                // Create, attach and allocate migration agents: source and sink
                //TODO: change mig_s to USED capacity instead of total... (but right now there is no
       overhead of VM, so it could be 0 bytes for idle machine.)
                int mig_s = migrated_vm_->getTotalCap(
      Memory); // migration size
                int p_s = vm_migration_sender_->getPacketSize(); // packet size
                vm_migration_sink_->seq_expected_ = mig_s % p_s ? ( mig_s /
      p_s ) + 1: mig_s / p_s; // Ceiling of the number of the last packet
                vm_migration_sender_->sendmsg(mig_s, (void*) 1);
 }
```

4.63.4 Member Data Documentation

4.63.4.1 int VmMigration::id_ [private]

Definition at line 31 of file vmmigration.h.

4.63.4.2 VM∗ VmMigration::migrated_vm_ [private]

Definition at line 27 of file vmmigration.h.

4.63.4.3 ResourceProvider∗ VmMigration::target_ [private]

Definition at line 28 of file vmmigration.h.

4.63.4.4 TcpAgent∗ VmMigration::vm_migration_sender_ [private]

Definition at line 30 of file vmmigration.h.

4.63.4.5 VmMigrationSink∗ VmMigration::vm_migration_sink_ [private]

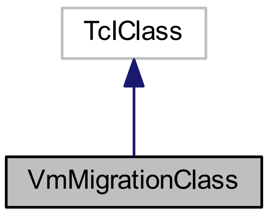
Definition at line 29 of file vmmigration.h.

The documentation for this class was generated from the following files:

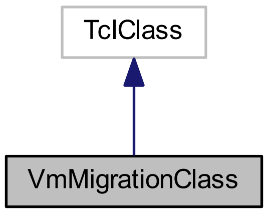
- vmmigration.h
- vmmigration.cc

4.64 VmMigrationClass Class Reference

Inheritance diagram for VmMigrationClass:



Collaboration diagram for VmMigrationClass:



Public Member Functions

- VmMigrationClass ()
- TclObject ∗ create (int argc, const char ∗const ∗argv)

4.64.1 Detailed Description

Definition at line 10 of file vmmigration.cc.

4.64.2 Constructor & Destructor Documentation

4.64.2.1 VmMigrationClass::VmMigrationClass ( ) [inline]

Definition at line 12 of file vmmigration.cc.

```
: TclClass("VmMigration") {}
```

4.64.3 Member Function Documentation

4.64.3.1 TclObject∗ VmMigrationClass::create (int *argc,* const char ∗const ∗ *argv*) [inline]

Definition at line 13 of file vmmigration.cc.

```
                                                        {
                                 return (new VmMigration());
                 }
```

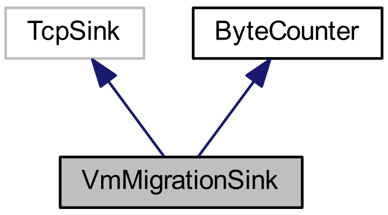
The documentation for this class was generated from the following file:

- vmmigration.cc

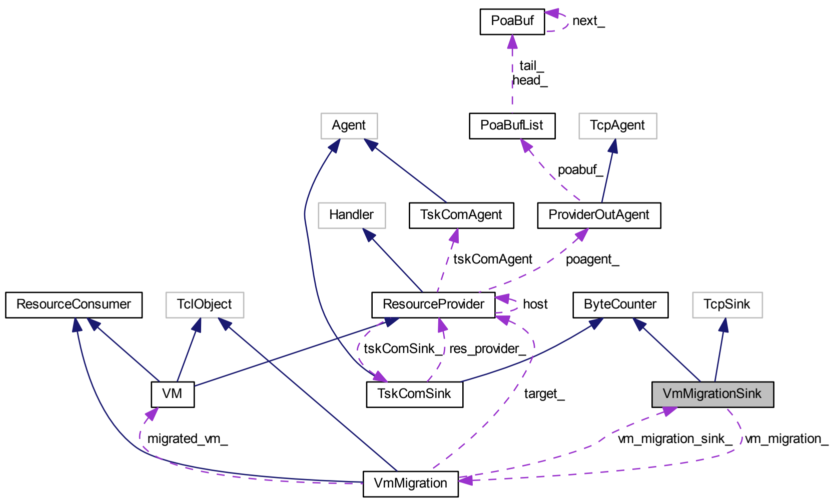
4.65 VmMigrationSink Class Reference

```
#include <vmmigrationsink.h>
```

Inheritance diagram for VmMigrationSink:



Collaboration diagram for VmMigrationSink:



Public Member Functions

- VmMigrationSink (Acker ∗)
- virtual ∼VmMigrationSink ()
- virtual void recv (Packet ∗, Handler ∗)
- void setVmMigration (VmMigration ∗vm_migration)

Public Attributes

- int seq_expected_

Private Attributes

- VmMigration ∗ vm_migration_
- bool migration_finished_

Additional Inherited Members

4.65.1 Detailed Description

Definition at line 16 of file vmmigrationsink.h.

4.65.2 Constructor & Destructor Documentation

4.65.2.1 VmMigrationSink::VmMigrationSink (Acker ∗ *a*)

Definition at line 19 of file vmmigrationsink.cc.

```
                                        : TcpSink(a), seq_expected_(-1),
      migration_finished_(false){

}
```

4.65.2.2 VmMigrationSink::∼VmMigrationSink ( ) [virtual]

Definition at line 23 of file vmmigrationsink.cc.

```
                                     {

}
```

4.65.3 Member Function Documentation

4.65.3.1 void VmMigrationSink::recv (Packet ∗ *pkt,* Handler ∗) [virtual]

Definition at line 31 of file vmmigrationsink.cc.

```
{
                bytes_since_ += hdr_cmn::access(pkt)->size();

                if(seq_expected_ == hdr_tcp::access(pkt)->seqno()){
                                //                              Migration complete.
                                migration_finished_ = true;
                                vm_migration_->finalizeMigration();
                }
                else if(migration_finished_){
                                std::cerr << "ERROR! Something went wrong. Packets received after migration
       is finished!\n";
                }
                TcpSink::recv(pkt,this);
}
```

4.65.3.2 void VmMigrationSink::setVmMigration (VmMigration ∗ *vm_migration*)

Definition at line 27 of file vmmigrationsink.cc.

```
                                                            {
                vm_migration_ = vm_migration;
}
```

4.65.4 Member Data Documentation

4.65.4.1 bool VmMigrationSink::migration_finished_ [private]

Definition at line 25 of file vmmigrationsink.h.

4.65.4.2 int VmMigrationSink::seq_expected_

Definition at line 21 of file vmmigrationsink.h.

4.65.4.3 VmMigration∗ VmMigrationSink::vm_migration_ [private]

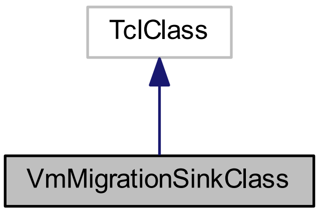
Definition at line 24 of file vmmigrationsink.h.

The documentation for this class was generated from the following files:

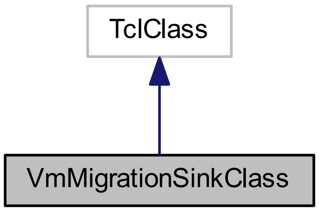
- vmmigrationsink.h
- vmmigrationsink.cc

4.66 VmMigrationSinkClass Class Reference

Inheritance diagram for VmMigrationSinkClass:



Collaboration diagram for VmMigrationSinkClass:



Public Member Functions

- VmMigrationSinkClass ()
- TclObject ∗ create (int, const char ∗const ∗)

4.66.1 Detailed Description

Definition at line 11 of file vmmigrationsink.cc.

4.66.2 Constructor & Destructor Documentation

4.66.2.1 VmMigrationSinkClass::VmMigrationSinkClass ( ) [inline]

Definition at line 13 of file vmmigrationsink.cc.

```
: TclClass("Agent/TCPSink/VmMigrationSink") {}
```

4.66.3 Member Function Documentation

4.66.3.1 TclObject∗ VmMigrationSinkClass::create (int , const char ∗const ∗) [inline]

Definition at line 14 of file vmmigrationsink.cc.

```
                                                         {
                                   return (new VmMigrationSink(new Acker()));
                  }
```

The documentation for this class was generated from the following file:

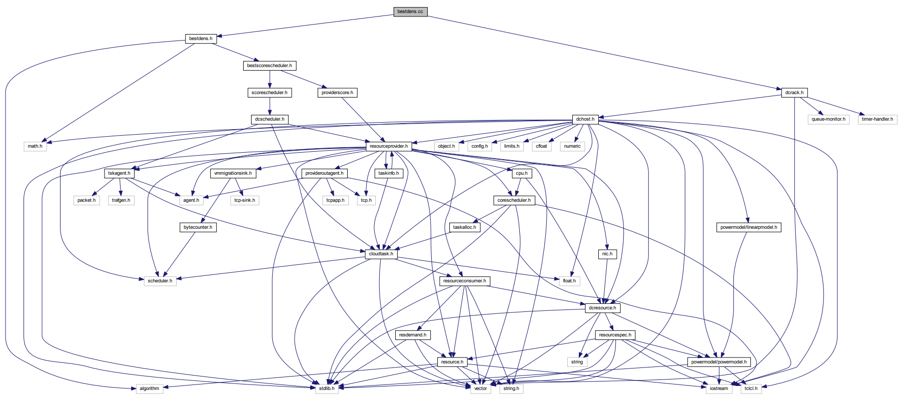
- vmmigrationsink.cc

5 File Documentation

5.1 bestdens.cc File Reference

```
#include "bestdens.h"
#include "dcrack.h"
```

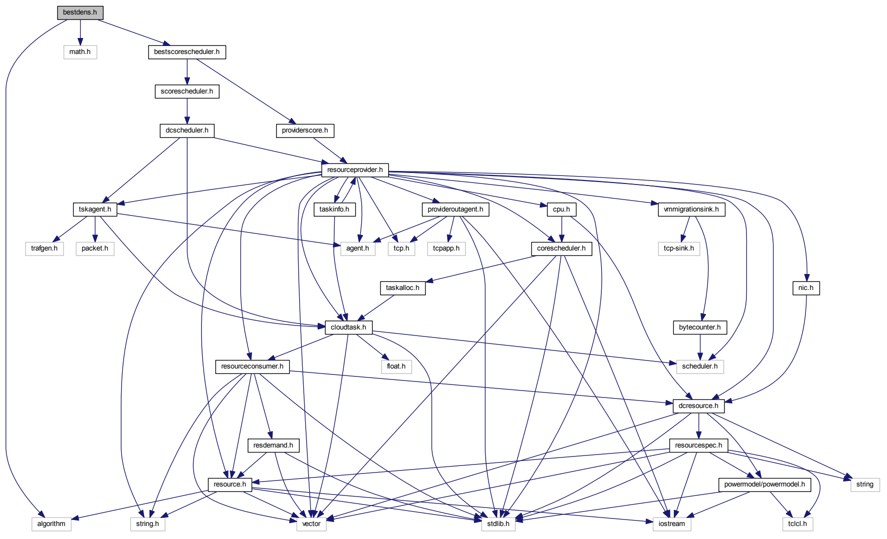
Include dependency graph for bestdens.cc:



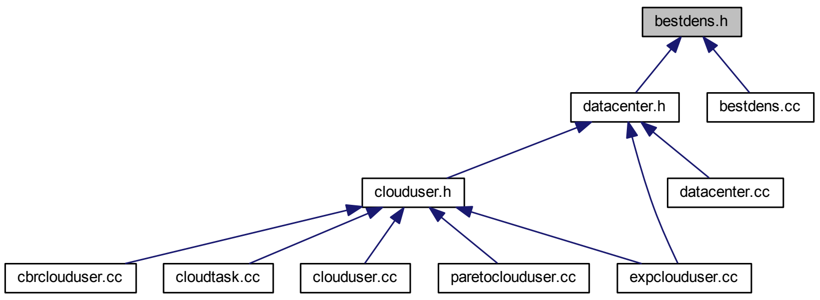
5.2 bestdens.h File Reference

```
#include <algorithm>
#include <math.h>
#include "bestscorescheduler.h"
```

Include dependency graph for bestdens.h:



This graph shows which files directly or indirectly include this file:



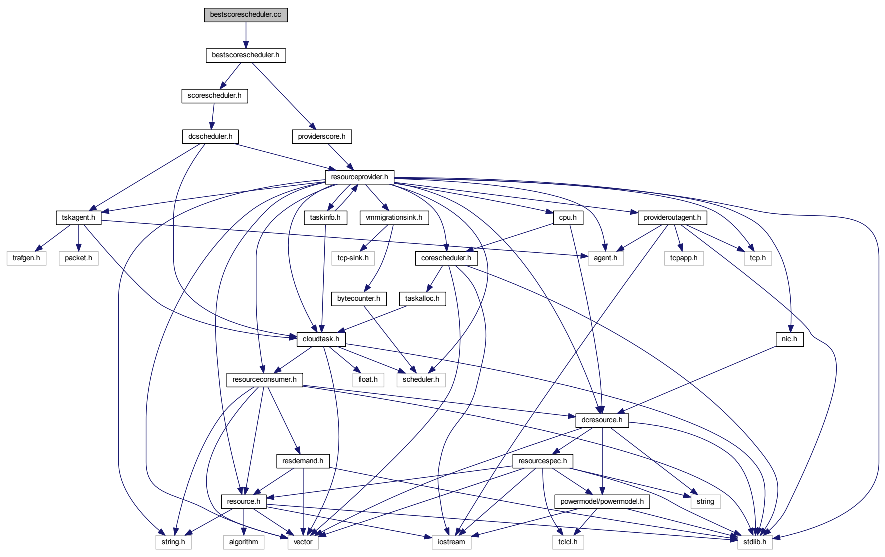
Classes

- class BestDENS

5.3 bestscorescheduler.cc File Reference

```
#include "bestscorescheduler.h"
```

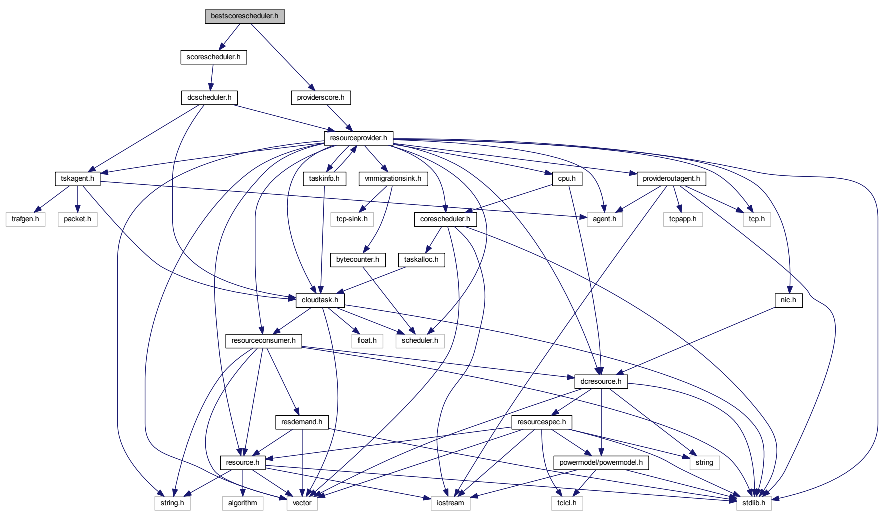
Include dependency graph for bestscorescheduler.cc:



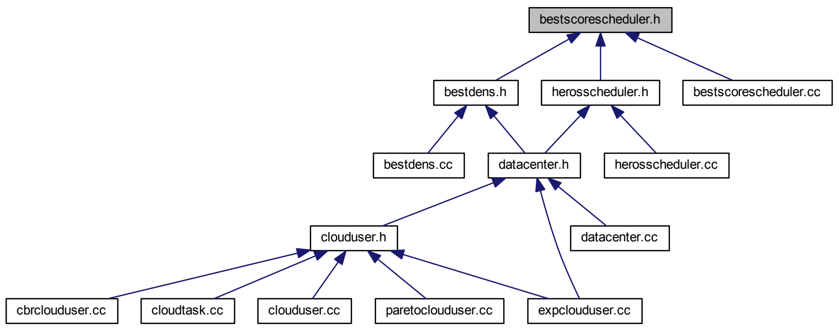
5.4 bestscorescheduler.h File Reference

```
#include "scorescheduler.h"
#include "providerscore.h"
```

Include dependency graph for bestscorescheduler.h:



This graph shows which files directly or indirectly include this file:



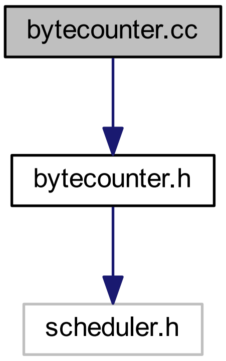
Classes

- class BestScoreScheduler

5.5 bytecounter.cc File Reference

```
#include "bytecounter.h"
```

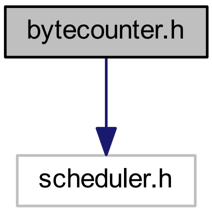
Include dependency graph for bytecounter.cc:



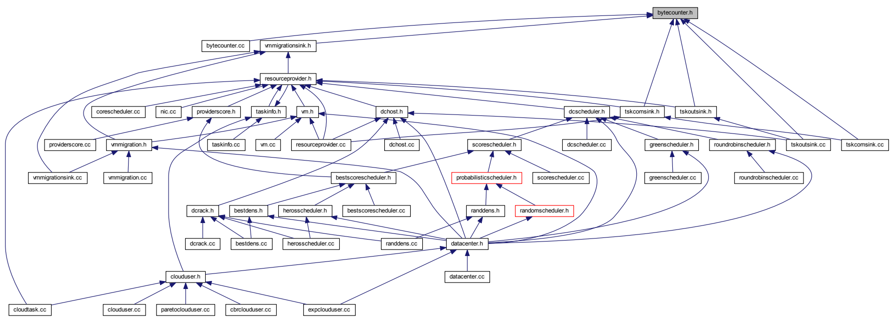
5.6 bytecounter.h File Reference

```
#include "scheduler.h"
```

Include dependency graph for bytecounter.h:



This graph shows which files directly or indirectly include this file:



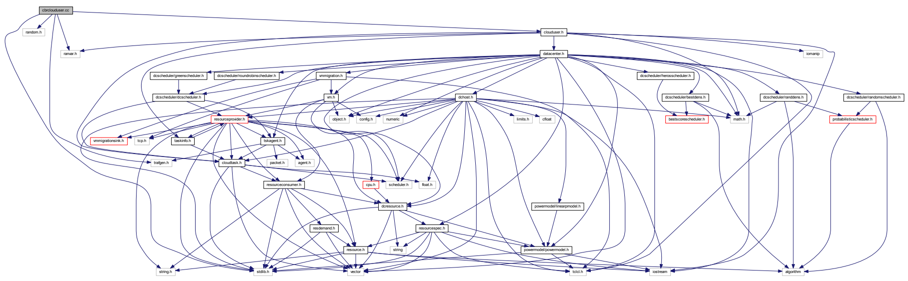
Classes

- class ByteCounter

5.7 cbrclouduser.cc File Reference

```
#include <stdlib.h>
#include "random.h"
#include "trafgen.h"
#include "ranvar.h"
#include "clouduser.h"
```

Include dependency graph for cbrclouduser.cc:



Classes

- class CBRCloudUser
- class CBRCloudUserClass

Variables

- CBRCloudUserClass class_cbr_clouduser

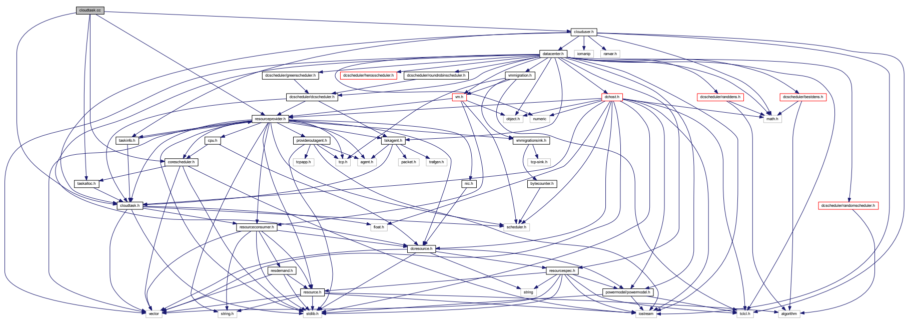
5.7.1 Variable Documentation

5.7.1.1 CBRCloudUserClass class_cbr_clouduser [static]

5.8 cloudtask.cc File Reference

```
#include "cloudtask.h"
#include "taskalloc.h"
#include "corescheduler.h"
#include "resourceprovider.h"
#include "clouduser.h"
```

Include dependency graph for cloudtask.cc:



Variables

- static const char rcsid [ ]

5.8.1 Variable Documentation

5.8.1.1 const char rcsid[ ] [static]

Initial value:

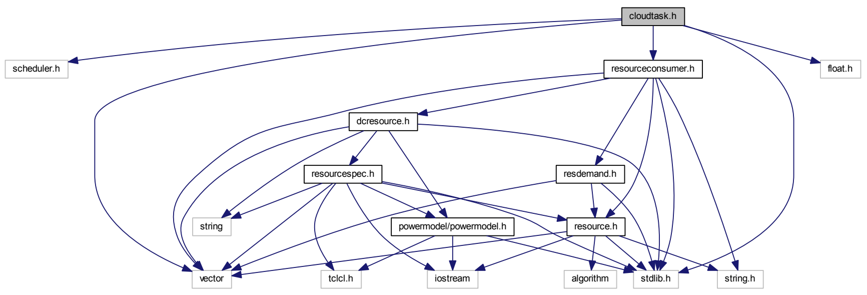
```
=
                                     "@(#) $Header: /cvsroot/nsnam/ns-2/common/taskobject.cc,v 1.43 $"
```

Definition at line 6 of file cloudtask.cc.

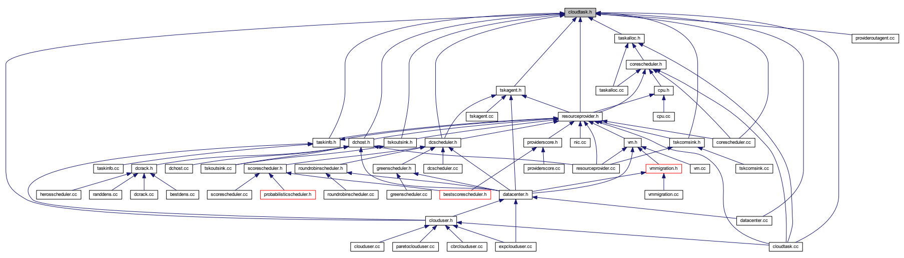
5.9 cloudtask.h File Reference

```
#include "scheduler.h"
#include "resourceconsumer.h"
#include <stdlib.h>
#include <float.h>
#include <vector>
```

Include dependency graph for cloudtask.h:



This graph shows which files directly or indirectly include this file:



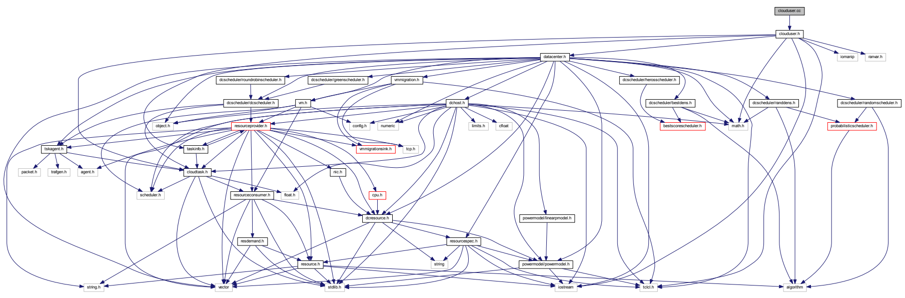
Classes

- class CloudTask

5.10 clouduser.cc File Reference

```
#include "clouduser.h"
```

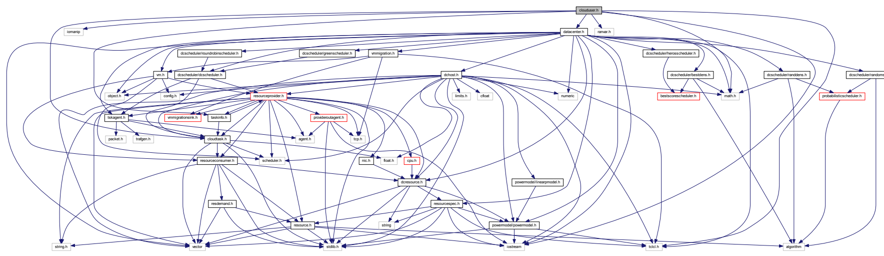
Include dependency graph for clouduser.cc:



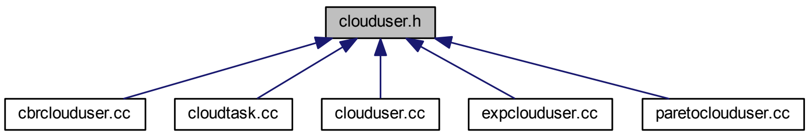
5.11 clouduser.h File Reference

```
#include <iostream>
#include <iomanip>
#include <math.h>
#include "tclcl.h"
#include "ranvar.h"
#include "datacenter.h"
#include "cloudtask.h"
#include "taskinfo.h"
```

Include dependency graph for clouduser.h:



This graph shows which files directly or indirectly include this file:



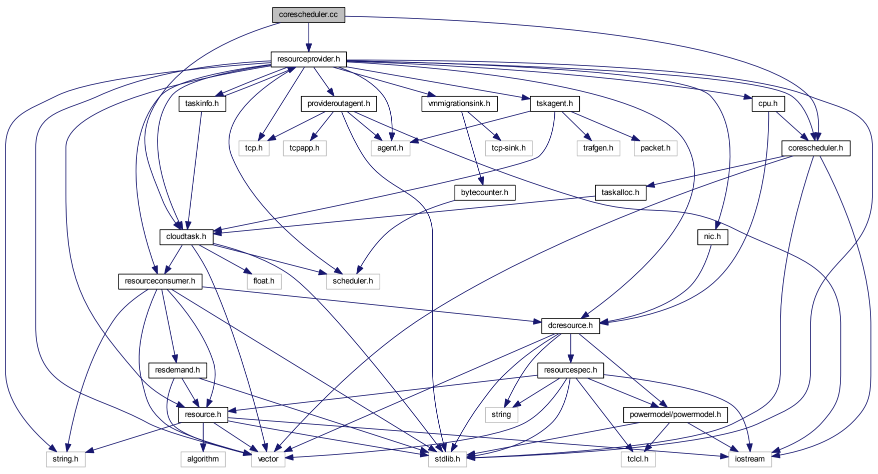
Classes

- class CloudUser

5.12 corescheduler.cc File Reference

```
#include "corescheduler.h"
#include "cloudtask.h"
#include "resourceprovider.h"
```

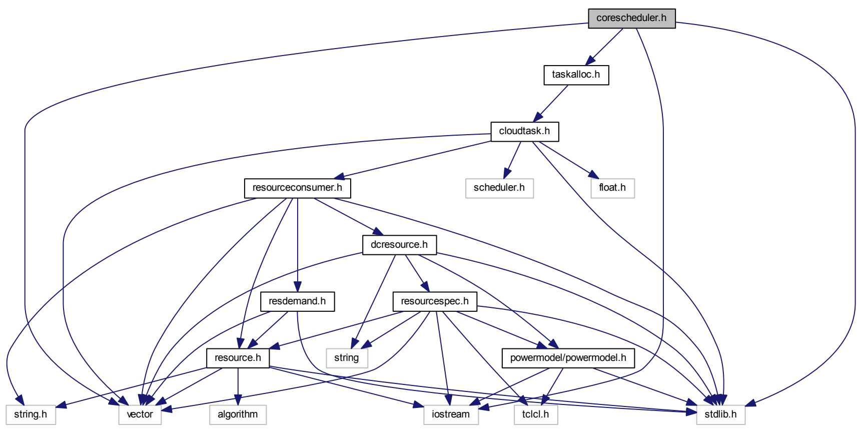
Include dependency graph for corescheduler.cc:



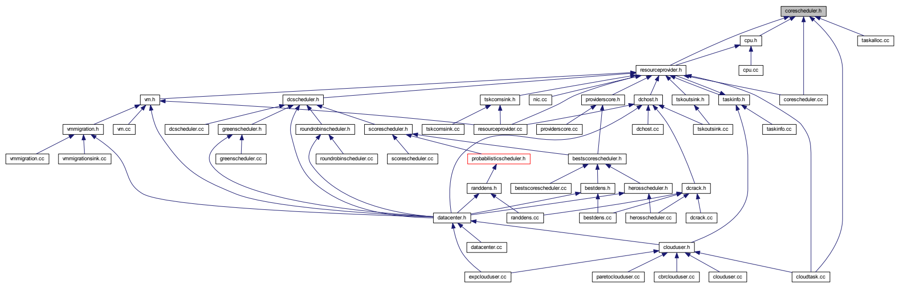
5.13 corescheduler.h File Reference

```
#include <stdlib.h>
#include <vector>
#include <iostream>
#include "taskalloc.h"
```

Include dependency graph for corescheduler.h:



This graph shows which files directly or indirectly include this file:



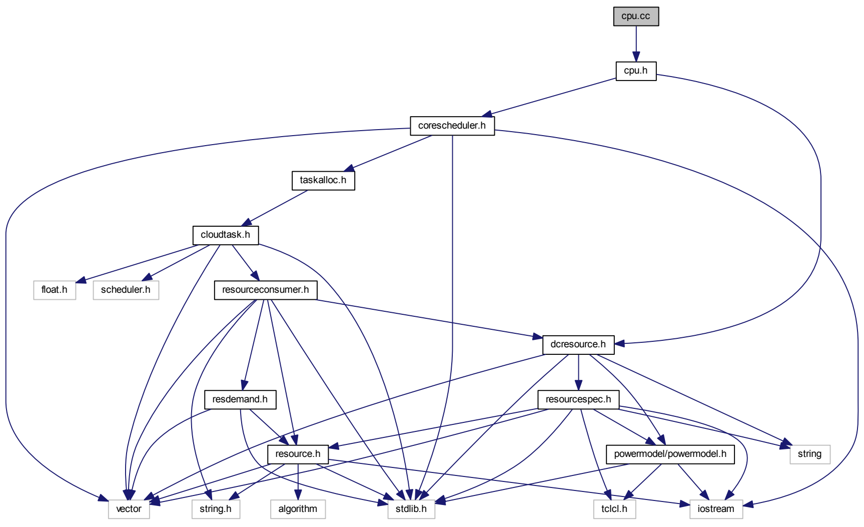
Classes

- class CoreScheduler

5.14 cpu.cc File Reference

```
#include "cpu.h"
```

Include dependency graph for cpu.cc:



Classes

- class CpuClass

Variables

- CpuClass class_cpu

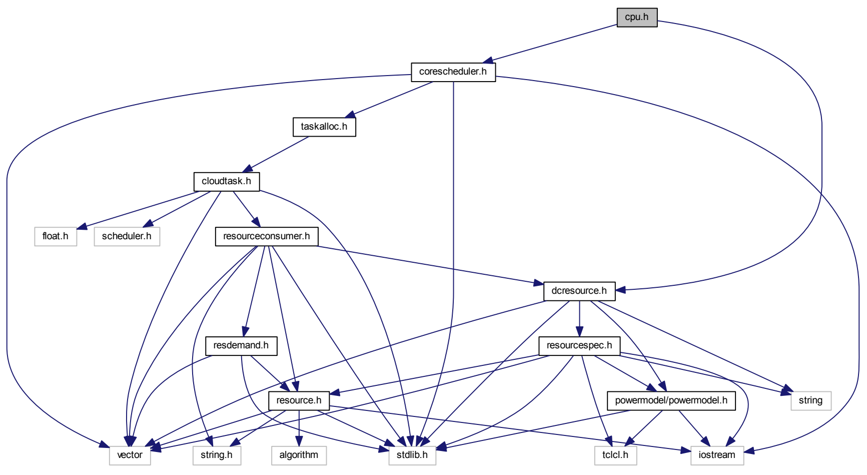
5.14.1 Variable Documentation

5.14.1.1 CpuClass class_cpu [static]

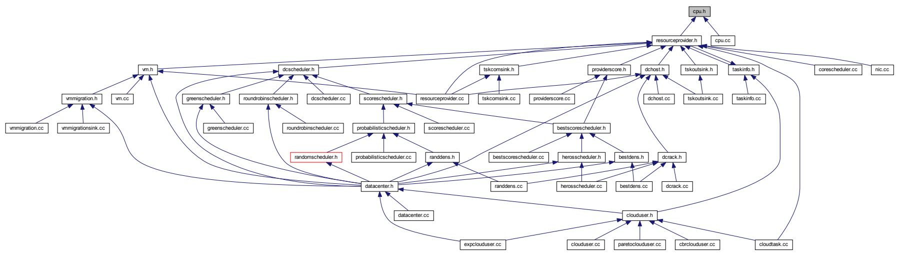
5.15 cpu.h File Reference

```
#include "corescheduler.h"
#include "dcresource.h"
```

Include dependency graph for cpu.h:



This graph shows which files directly or indirectly include this file:



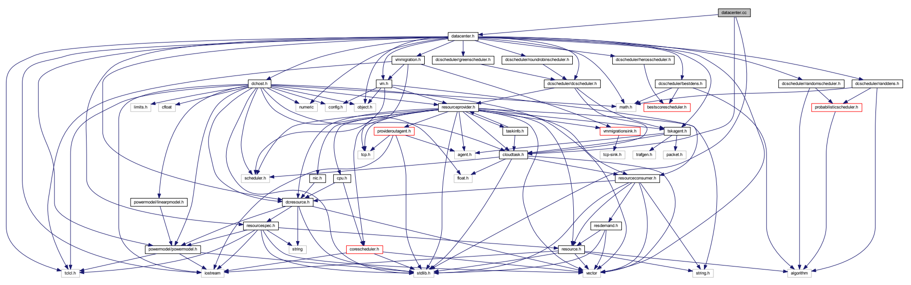
Classes

- class CPU

5.16 datacenter.cc File Reference

```
#include <stdlib.h>
#include "datacenter.h"
#include "cloudtask.h"
```

Include dependency graph for datacenter.cc:



Classes

- class DataCenterClass

Variables

- static const char rcsid [ ]
- DataCenterClass class_datacenter

5.16.1 Variable Documentation

5.16.1.1 DataCenterClass class_datacenter [static]

5.16.1.2 const char rcsid[ ] [static]

Initial value:

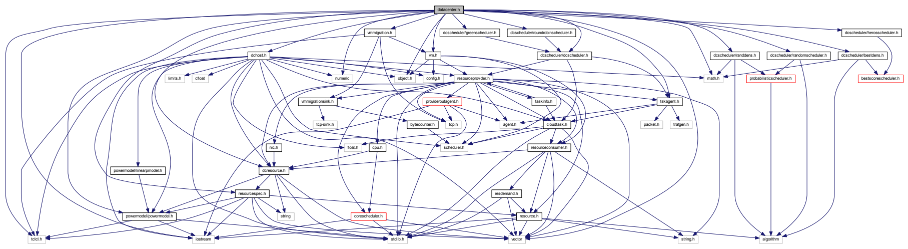
```
=
                                        "@(#) $Header: /cvsroot/nsnam/ns-2/common/datacenter.cc,v 1.43 $"
```

Definition at line 6 of file datacenter.cc.

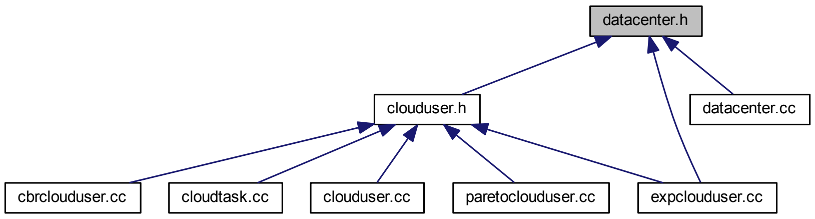
5.17 datacenter.h File Reference

```
#include "object.h"
#include "dchost.h"
#include "vm.h"
#include "tskagent.h"
#include "resourcespec.h"
#include "dcresource.h"
#include "powermodel/powermodel.h"
#include "vmmigration.h"
#include "dcscheduler/dcscheduler.h"
#include "dcscheduler/greenscheduler.h"
#include "dcscheduler/roundrobinscheduler.h"
#include "dcscheduler/randdens.h"
#include "dcscheduler/bestdens.h"
#include "dcscheduler/randomscheduler.h"
#include "dcscheduler/herosscheduler.h"
#include <tclcl.h>
#include <vector>
#include <math.h>
#include <iostream>
#include <numeric>
```

Include dependency graph for datacenter.h:



This graph shows which files directly or indirectly include this file:



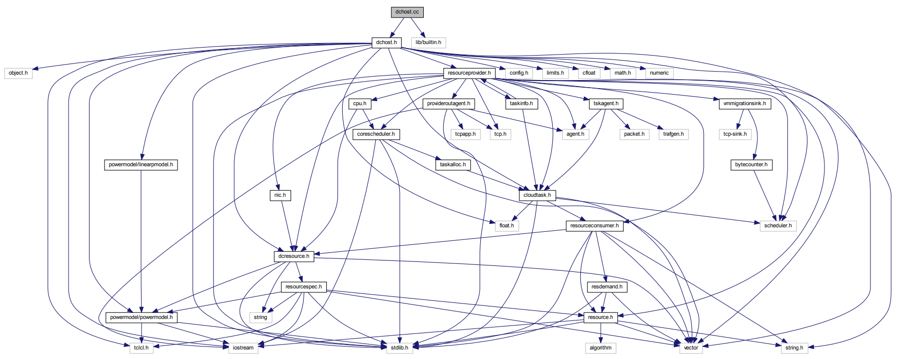
Classes

- class DataCenter

5.18 dchost.cc File Reference

```
#include "dchost.h"
#include <lib/builtin.h>
```

Include dependency graph for dchost.cc:



Classes

- class DcHostClass

Variables

- DcHostClass class_dchost

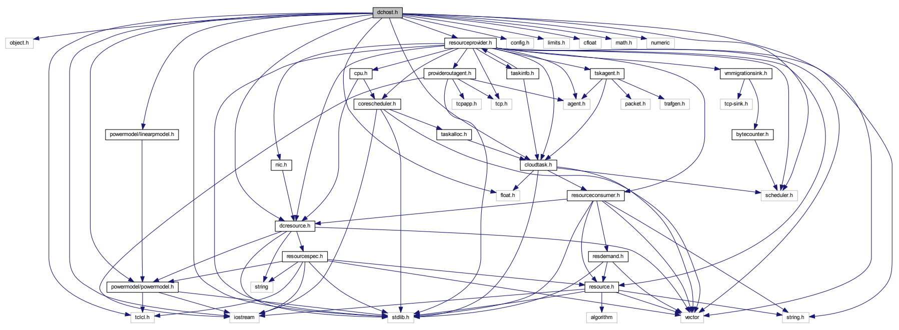
5.18.1 Variable Documentation

5.18.1.1 DcHostClass class_dchost `[static]`

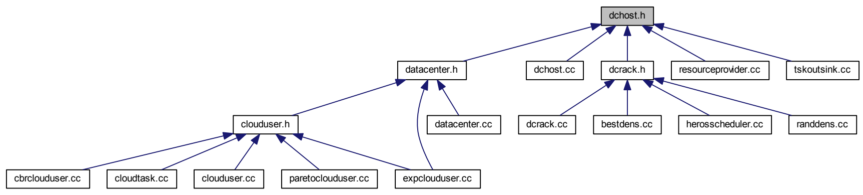
5.19 dchost.h File Reference

```
#include "object.h"
#include "cloudtask.h"
#include "config.h"
#include "scheduler.h"
#include <stdlib.h>
#include <limits.h>
#include <tclcl.h>
#include <vector>
#include <cfloat>
#include <float.h>
#include <math.h>
#include <iostream>
#include <numeric>
#include "dcresource.h"
#include "resourceprovider.h"
#include "powermodel/linearpmodel.h"
#include "powermodel/powermodel.h"
```

Include dependency graph for dchost.h:



This graph shows which files directly or indirectly include this file:



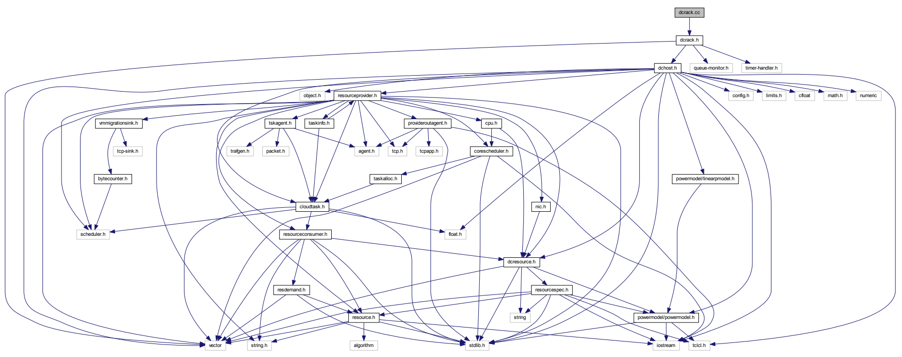
Classes

- class DcHost

5.20 dcrack.cc File Reference

```
#include "dcrack.h"
```

Include dependency graph for dcrack.cc:



Classes

- class DcRackClass

Variables

- DcRackClass class_dcrack

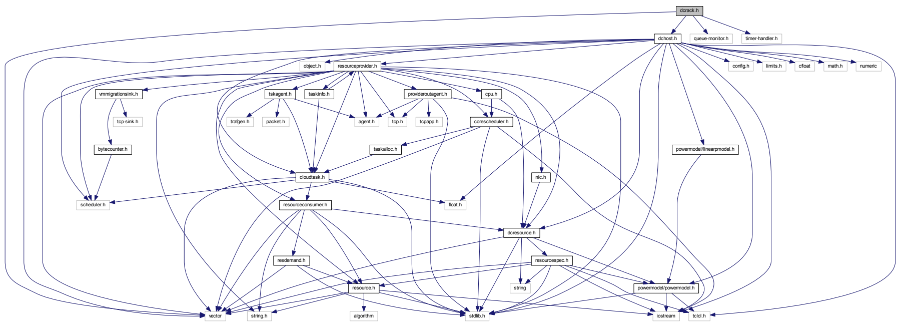
5.20.1 Variable Documentation

5.20.1.1 DcRackClass class_dcrack [static]

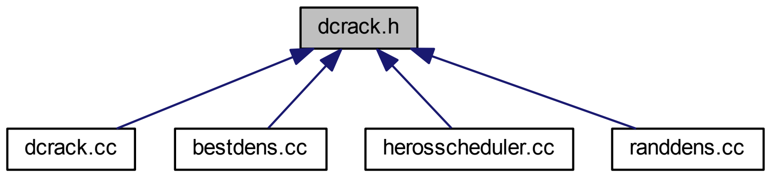
5.21 dcrack.h File Reference

```
#include <vector>
#include "dchost.h"
#include "queue-monitor.h"
#include "timer-handler.h"
```

Include dependency graph for dcrack.h:



This graph shows which files directly or indirectly include this file:



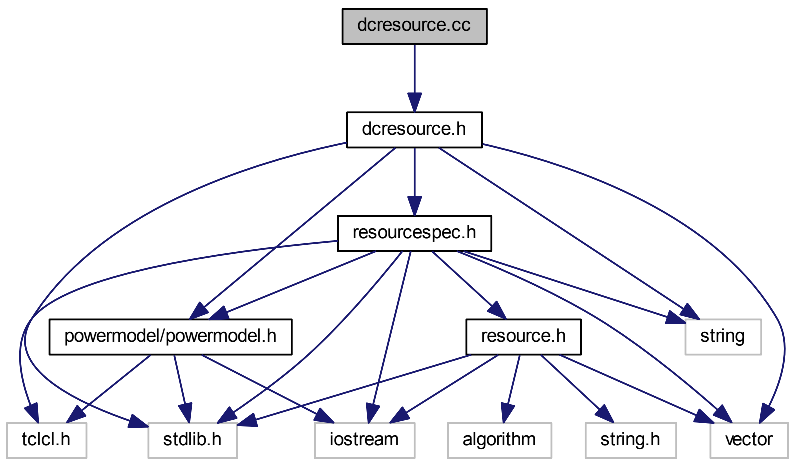
Classes

- class DcRack

5.22 dcresource.cc File Reference

```
#include "dcresource.h"
```

Include dependency graph for dcresource.cc:



Classes

- class DcResourceClass

Variables

- DcResourceClass class_dcresource

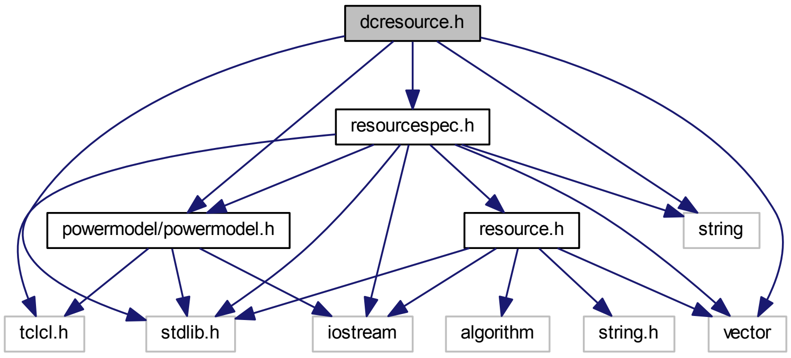
5.22.1 Variable Documentation

5.22.1.1 DcResourceClass class_dcresource [static]

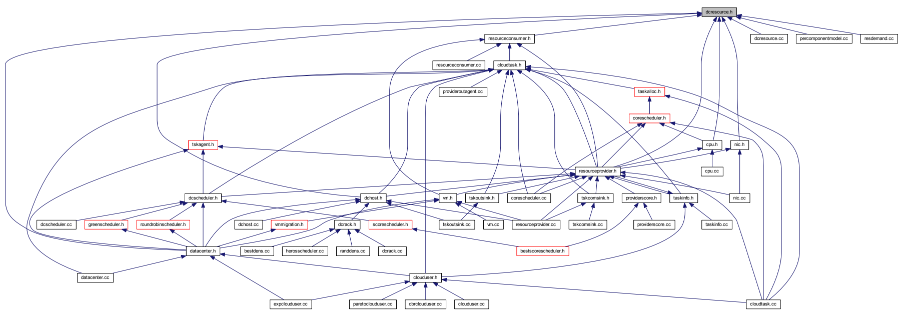
5.23 dcresource.h File Reference

```
#include <stdlib.h>
#include <vector>
#include <string>
#include "resourcespec.h"
#include "powermodel/powermodel.h"
```

Include dependency graph for dcresource.h:



This graph shows which files directly or indirectly include this file:

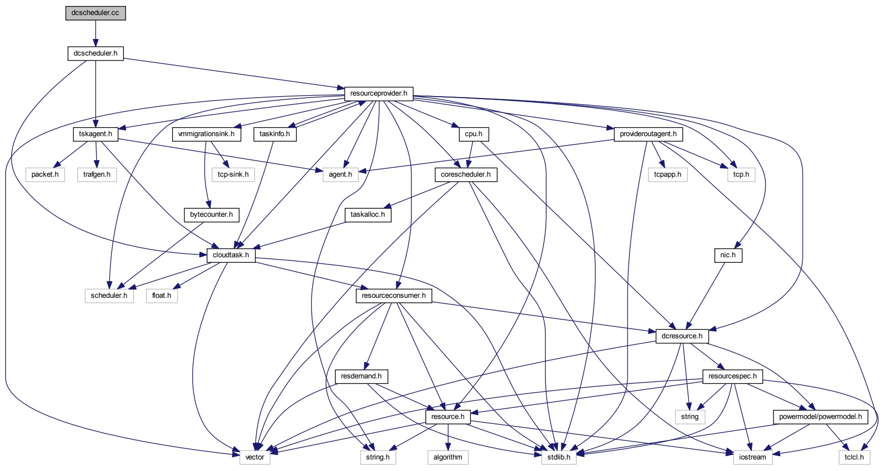


Classes

- class DcResource

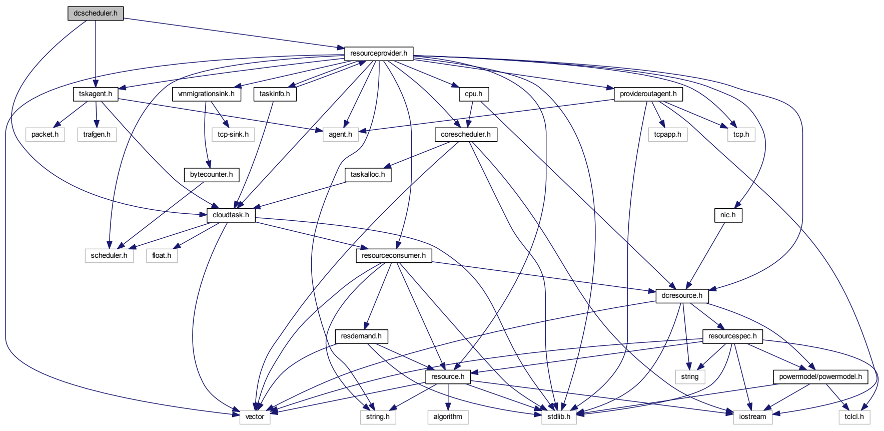
5.24 dcscheduler.cc File Reference

```
#include "dcscheduler.h"
```
Include dependency graph for dcscheduler.cc:

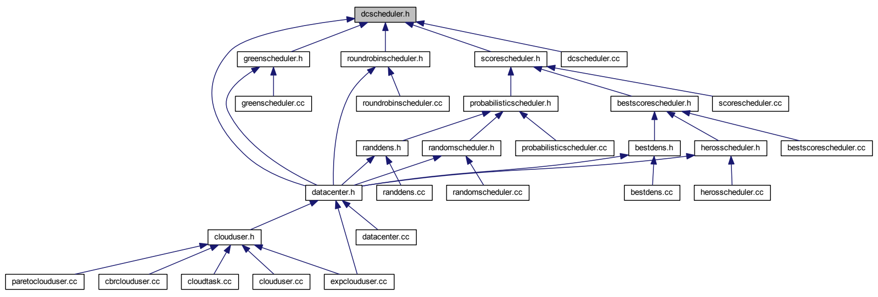


5.25 dcscheduler.h File Reference

```
#include "tskagent.h"
#include "cloudtask.h"
#include "resourceprovider.h"
```
Include dependency graph for dcscheduler.h:



This graph shows which files directly or indirectly include this file:



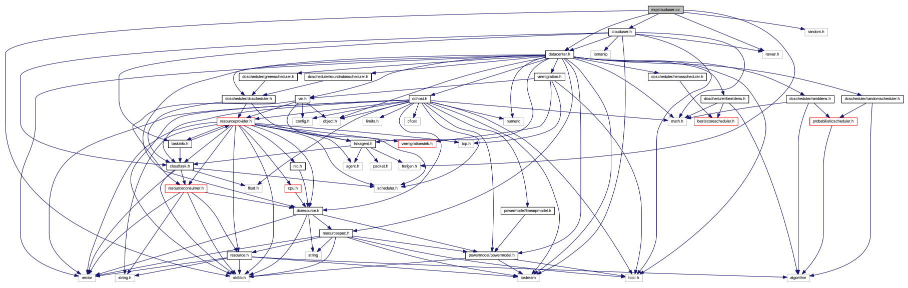
Classes

- class DcScheduler

5.26 expclouduser.cc File Reference

```
#include <stdlib.h>
#include "random.h"
#include "datacenter.h"
#include "trafgen.h"
#include "ranvar.h"
#include "clouduser.h"
```

Include dependency graph for expclouduser.cc:



Classes

- class ExpCloudUser
- class ExpCloudUserClass

Variables

- static const char rcsid [ ]
- ExpCloudUserClass class_exp_cloud_user

5.26.1 Variable Documentation

5.26.1.1 ExpCloudUserClass class_exp_cloud_user [static]

5.26.1.2 const char rcsid[ ] [static]

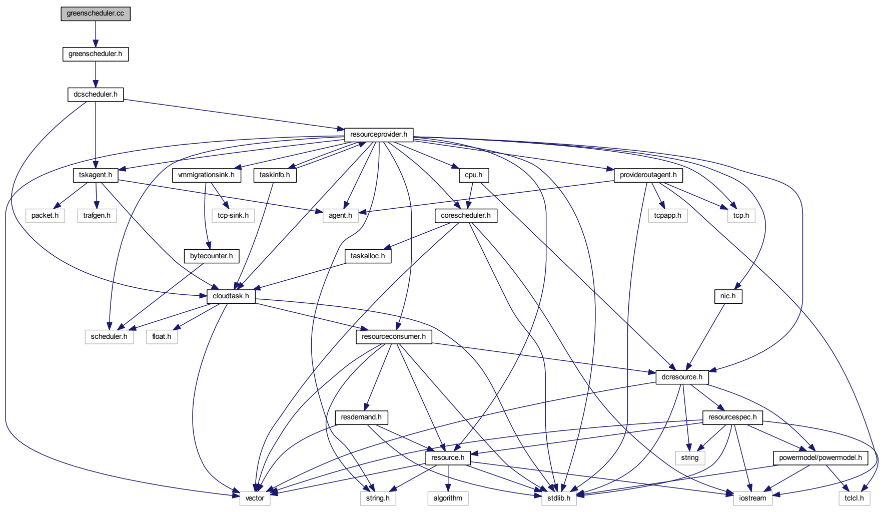
Initial value:

```
=
                                   "@(#) $Header: /cvsroot/nsnam/ns-2/tools/clouduser.cc,v 1.15  Exp $"
```

Definition at line 2 of file expclouduser.cc.

5.27 greenscheduler.cc File Reference

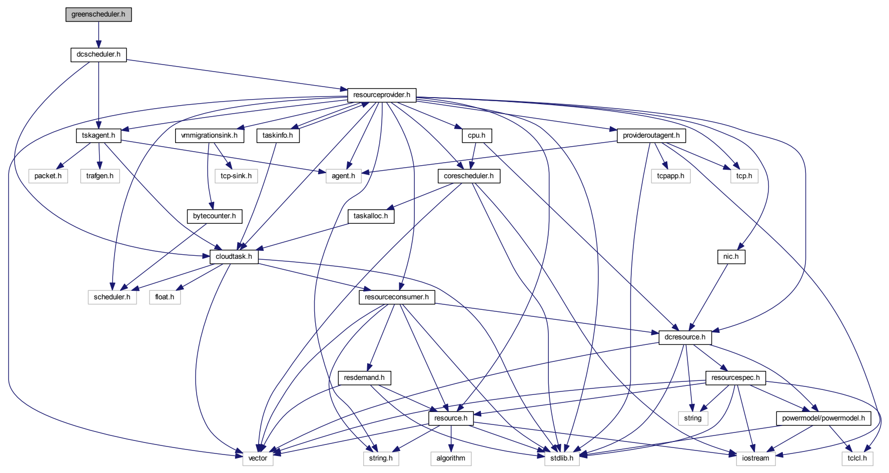
`#include "greenscheduler.h"`
Include dependency graph for greenscheduler.cc:



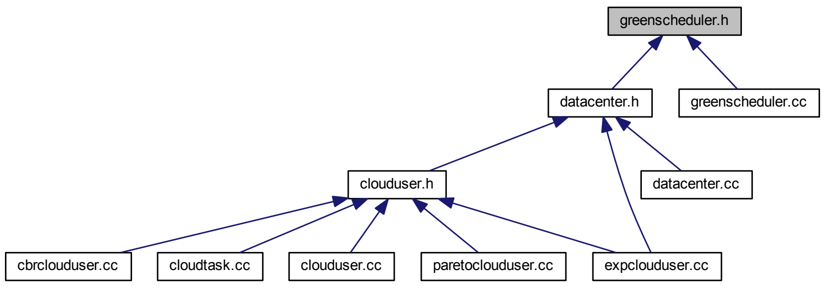
5.28 greenscheduler.h File Reference

`#include "dcscheduler.h"`

Include dependency graph for greenscheduler.h:



This graph shows which files directly or indirectly include this file:



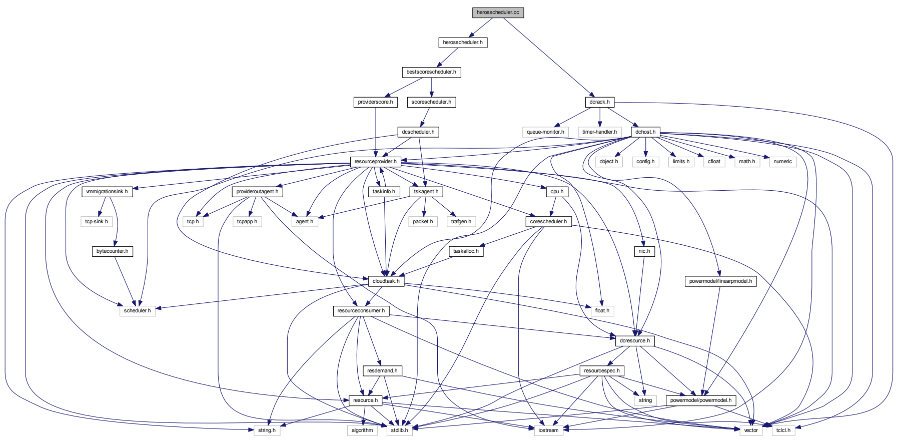
Classes

- class GreenScheduler

5.29 herosscheduler.cc File Reference

```
#include "herosscheduler.h"
#include "dcrack.h"
```

Include dependency graph for herosscheduler.cc:



Functions

- bool herosComparator (const ProviderScore &first, const ProviderScore &second)

5.29.1 Function Documentation

5.29.1.1 bool herosComparator (const ProviderScore & *first*, const ProviderScore & *second*)

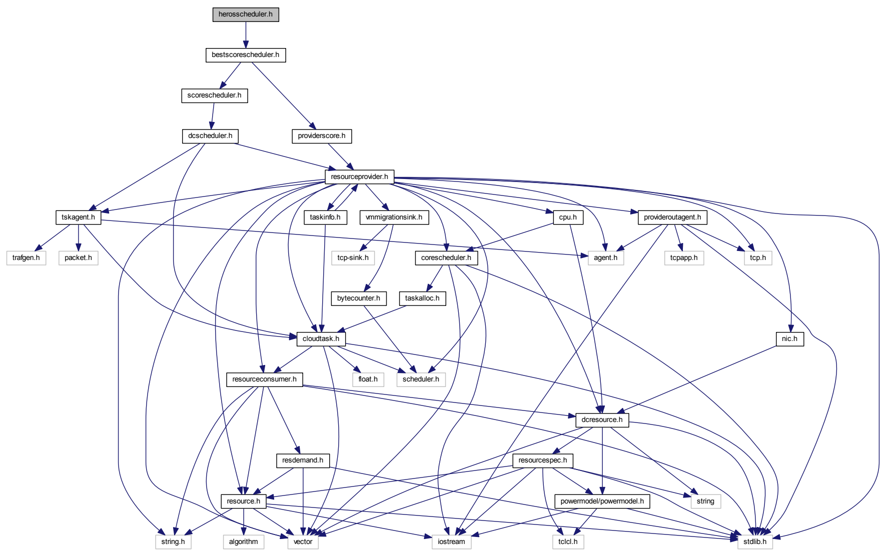
Definition at line 113 of file herosscheduler.cc.

```
                                                                              {
                if(first.score_ != second.score_){
                                return first.score_ < second.score_;
                } else {
                               return HerosScheduler::performancePerWattMax
   (first.provider_)*first.comm_potential_ <
   HerosScheduler::performancePerWattMax(second.
   provider_)*second.comm_potential_;
                }
}
```

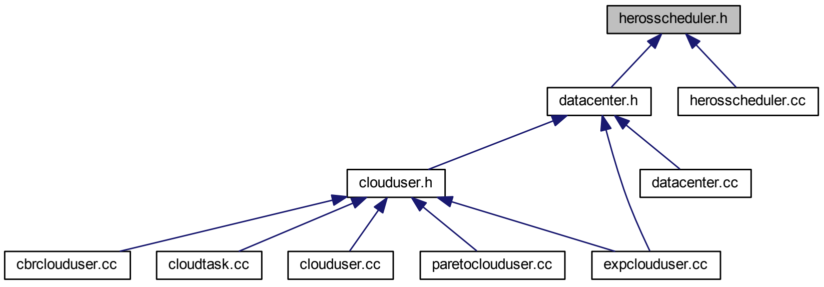
5.30 herosscheduler.h File Reference

`#include "bestscorescheduler.h"`

Include dependency graph for herosscheduler.h:



This graph shows which files directly or indirectly include this file:



Classes

- class HerosScheduler

Functions

- bool herosComparator (const ProviderScore &first, const ProviderScore &second)

5.30.1 Function Documentation

5.30.1.1 bool herosComparator (const **ProviderScore** & *first*, const **ProviderScore** & *second*)

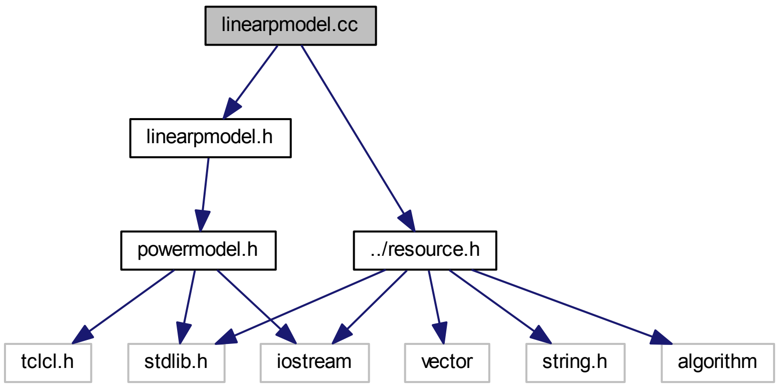
Definition at line 113 of file herosscheduler.cc.

```
                                                                        {
                if(first.score_ != second.score_){
                                return first.score_ < second.score_;
                } else {
                               return HerosScheduler::performancePerWattMax
      (first.provider_)*first.comm_potential_ <
      HerosScheduler::performancePerWattMax(second.
      provider_)*second.comm_potential_;
                }
 }
```

5.31 linearpmodel.cc File Reference

```
#include "linearpmodel.h"
#include "../resource.h"
```

Include dependency graph for linearpmodel.cc:



Classes

- class LinearPModelClass

Variables

- LinearPModelClass class_powermodel

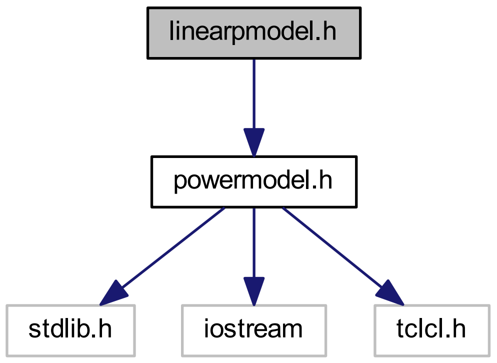
5.31.1 Variable Documentation

5.31.1.1 LinearPModelClass class_powermodel [static]

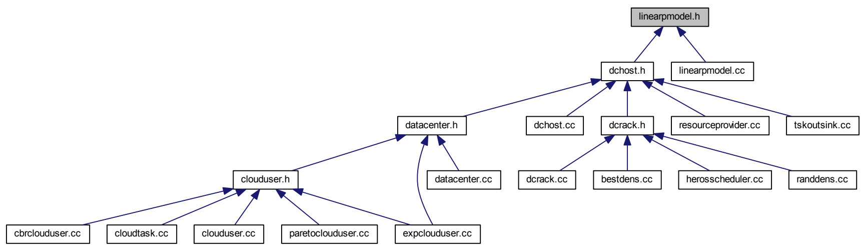
5.32 linearpmodel.h File Reference

```
#include "powermodel.h"
```

Include dependency graph for linearpmodel.h:



This graph shows which files directly or indirectly include this file:



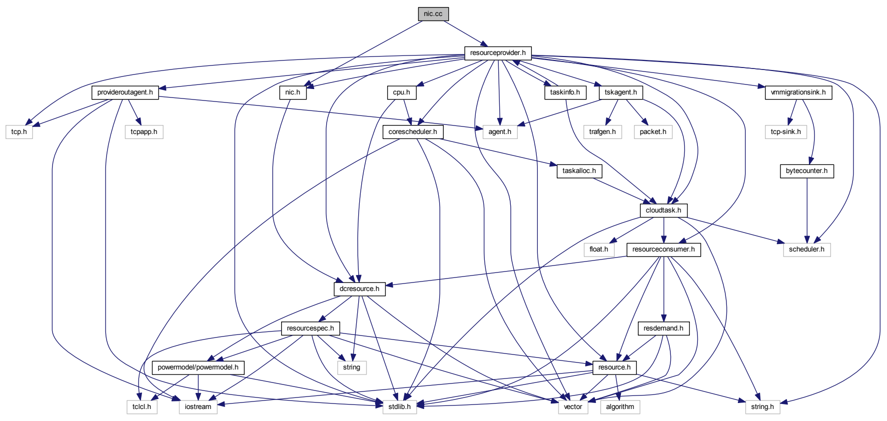
Classes

- class LinearPModel

5.33 nic.cc File Reference

```
#include "nic.h"
#include "resourceprovider.h"
```

Include dependency graph for nic.cc:



Classes

- class NicClass

Variables

- NicClass class_nic

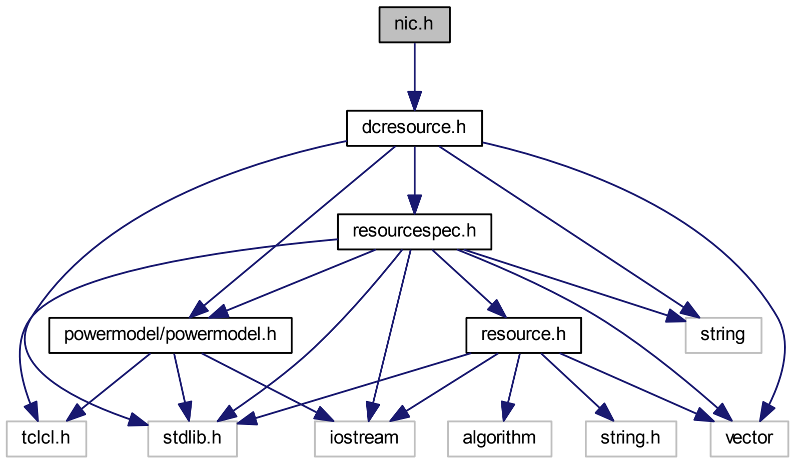
5.33.1 Variable Documentation

5.33.1.1 NicClass class_nic [static]

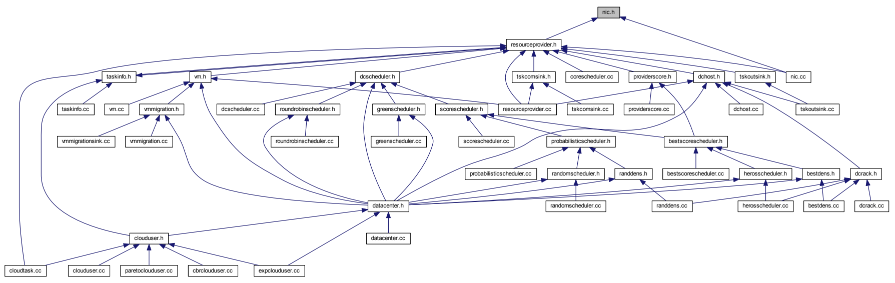
5.34 nic.h File Reference

`#include "dcresource.h"`

Include dependency graph for nic.h:



This graph shows which files directly or indirectly include this file:



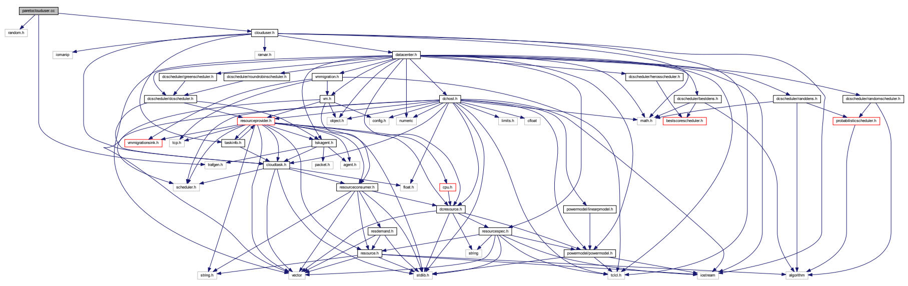
Classes

- class NIC

5.35 paretoclouduser.cc File Reference

```
#include "random.h"
#include "trafgen.h"
#include "clouduser.h"
```

Include dependency graph for paretoclouduser.cc:



Classes

- class ParetoCloudUser
- class POOTrafficClass

Variables

- static const char rcsid [ ]
- POOTrafficClass class_pareto_clouduser

5.35.1 Variable Documentation

5.35.1.1 POOTrafficClass class_pareto_clouduser [static]

5.35.1.2 const char rcsid[ ] [static]

Initial value:

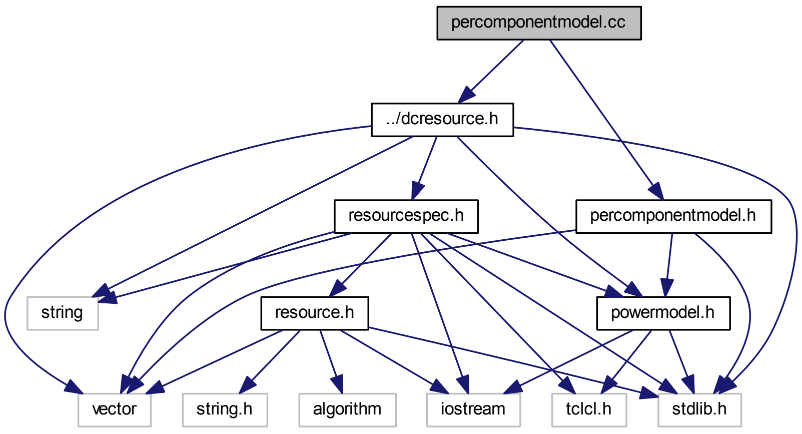
```
=
                                "@(#) $Header: /cvsroot/nsnam/ns-2/tools/pareto.cc,v 1.9 2005/08/26
      05:05:31 tomh Exp $"
```

Definition at line 6 of file paretoclouduser.cc.

5.36 percomponentmodel.cc File Reference

```
#include "percomponentmodel.h"
#include "../dcresource.h"
```

Include dependency graph for percomponentmodel.cc:



Classes

- class PerComponentModelClass

Variables

- PerComponentModelClass class_powermodel

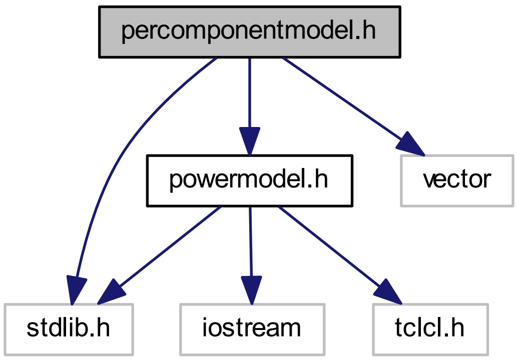
5.36.1 Variable Documentation

5.36.1.1 PerComponentModelClass class_powermodel [static]

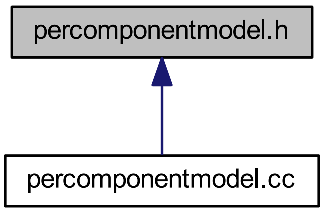
5.37 percomponentmodel.h File Reference

```
#include "powermodel.h"
#include <stdlib.h>
#include <vector>
```

Include dependency graph for percomponentmodel.h:



This graph shows which files directly or indirectly include this file:



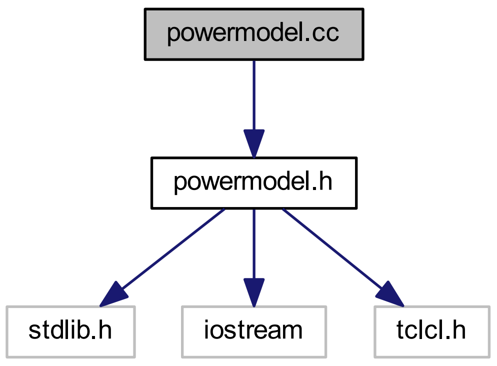
Classes

- class PerComponentModel

5.38 powermodel.cc File Reference

```
#include "powermodel.h"
```

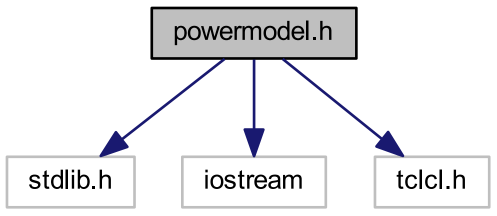
Include dependency graph for powermodel.cc:



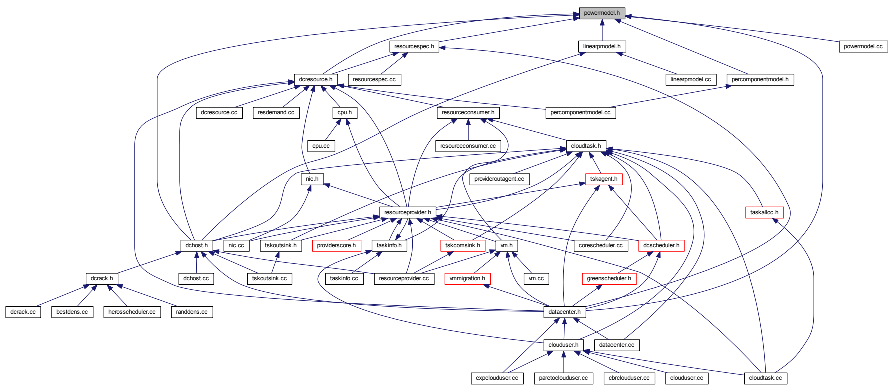
5.39 powermodel.h File Reference

```
#include <stdlib.h>
#include <iostream>
#include <tclcl.h>
```

Include dependency graph for powermodel.h:



This graph shows which files directly or indirectly include this file:



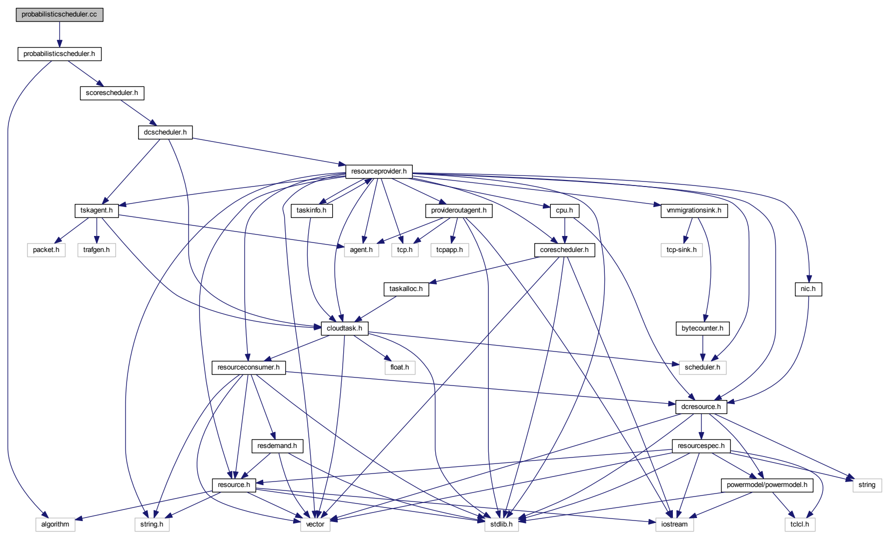
Classes

- class PowerModel

5.40 probabilisticscheduler.cc File Reference

```
#include "probabilisticscheduler.h"
```

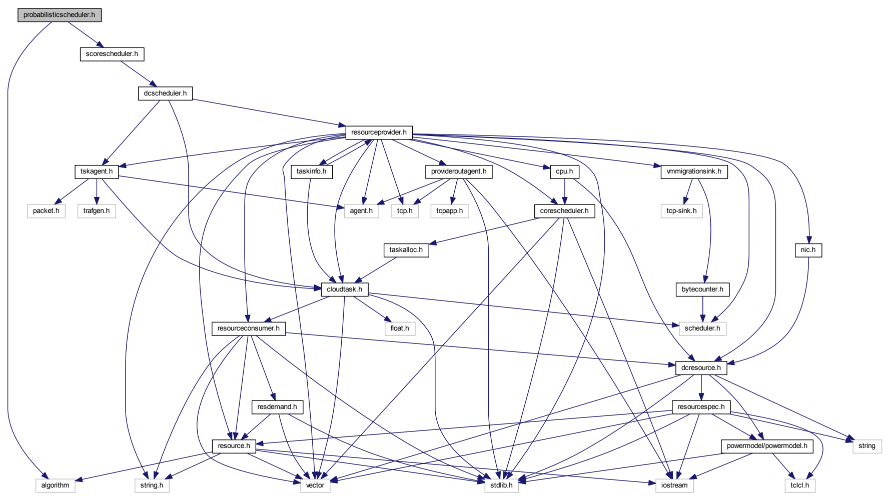
Include dependency graph for probabilisticscheduler.cc:



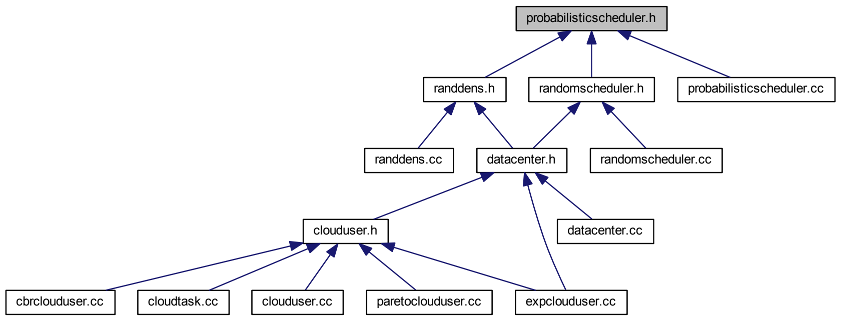
5.41 probabilisticscheduler.h File Reference

```
#include <algorithm>
#include "scorescheduler.h"
```

Include dependency graph for probabilisticscheduler.h:



This graph shows which files directly or indirectly include this file:



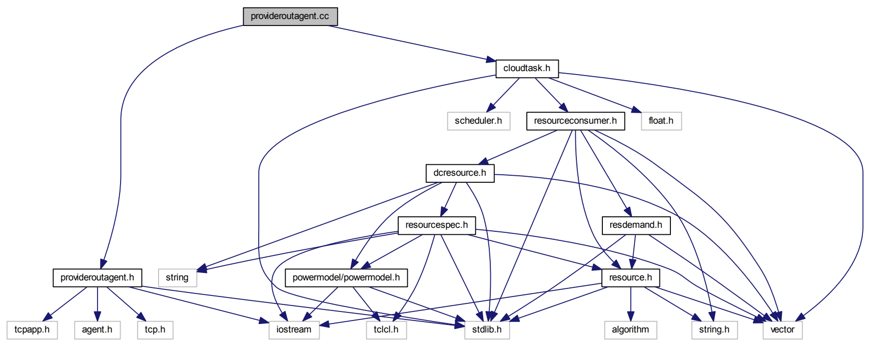
Classes

- class ProbabilisticScheduler

5.42 provideroutagent.cc File Reference

```
#include "provideroutagent.h"
#include "cloudtask.h"
```

Include dependency graph for provideroutagent.cc:



Classes

- class ProvOutAgentClass

Variables

- ProvOutAgentClass class_tsk_provoutagent

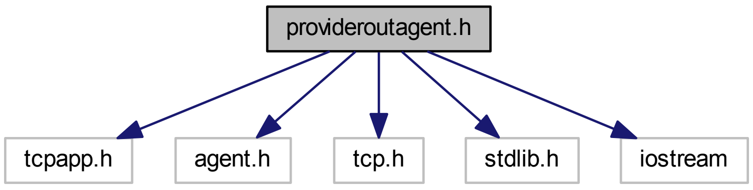
5.42.1 Variable Documentation

5.42.1.1 ProvOutAgentClass class_tsk_provoutagent `[static]`

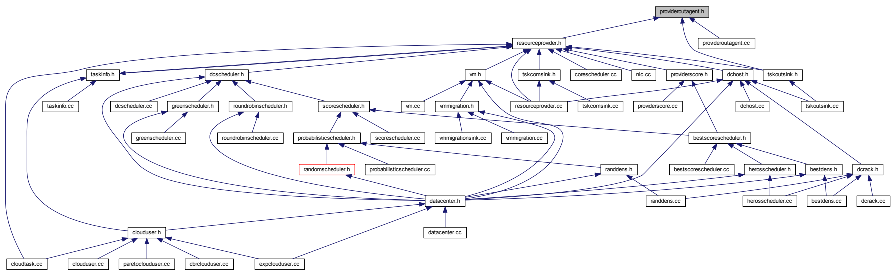
5.43 provideroutagent.h File Reference

```
#include <tcpapp.h>
#include "agent.h"
#include "tcp.h"
#include <stdlib.h>
#include <iostream>
```

Include dependency graph for provideroutagent.h:



This graph shows which files directly or indirectly include this file:



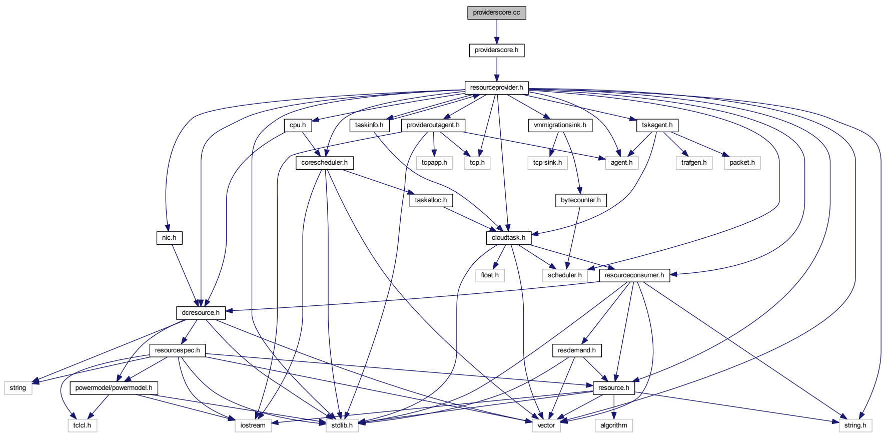
Classes

- class PoaBuf
- class PoaBufList
- class ProviderOutAgent

5.44 providerscore.cc File Reference

```
#include "providerscore.h"
```

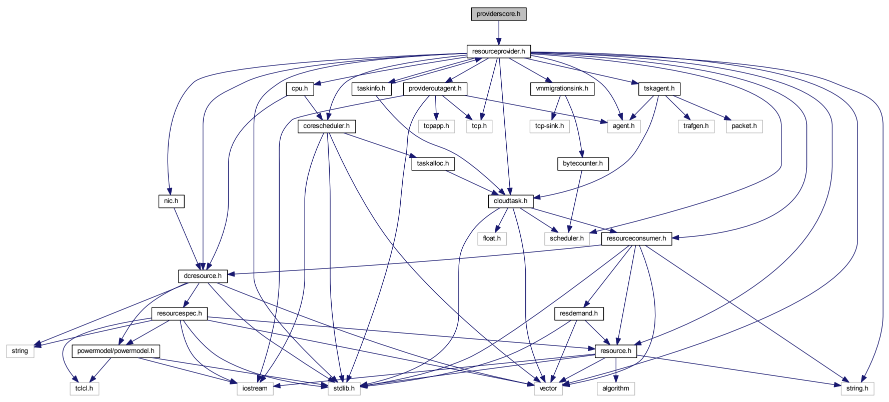
Include dependency graph for providerscore.cc:



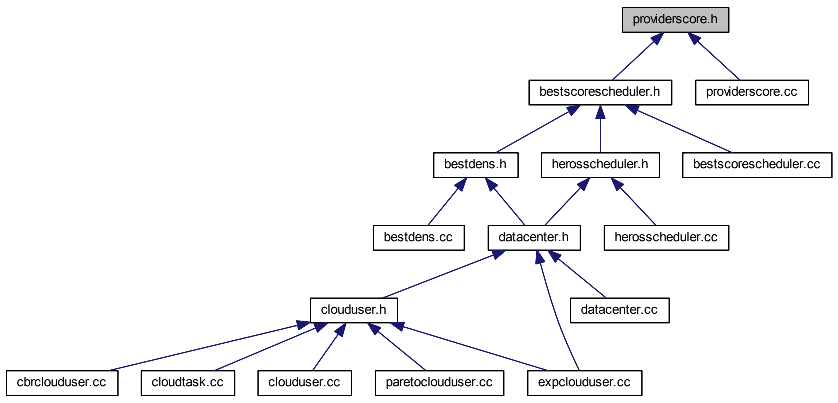
5.45 providerscore.h File Reference

```
#include "resourceprovider.h"
```

Include dependency graph for providerscore.h:



This graph shows which files directly or indirectly include this file:



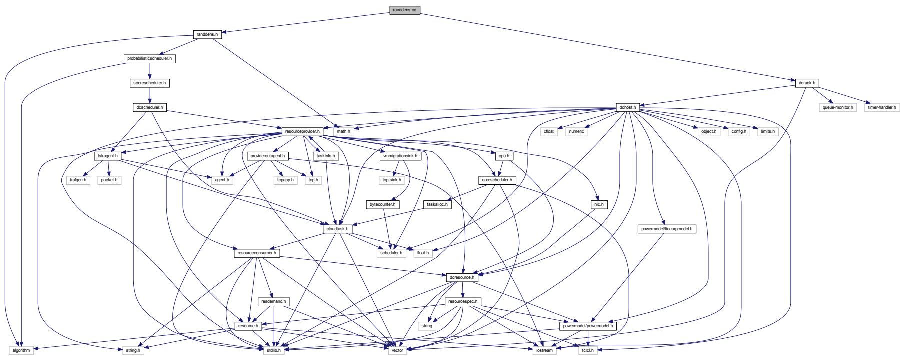
Classes

- class ProviderScore

5.46 randdens.cc File Reference

```
#include "randdens.h"
#include "dcrack.h"
```

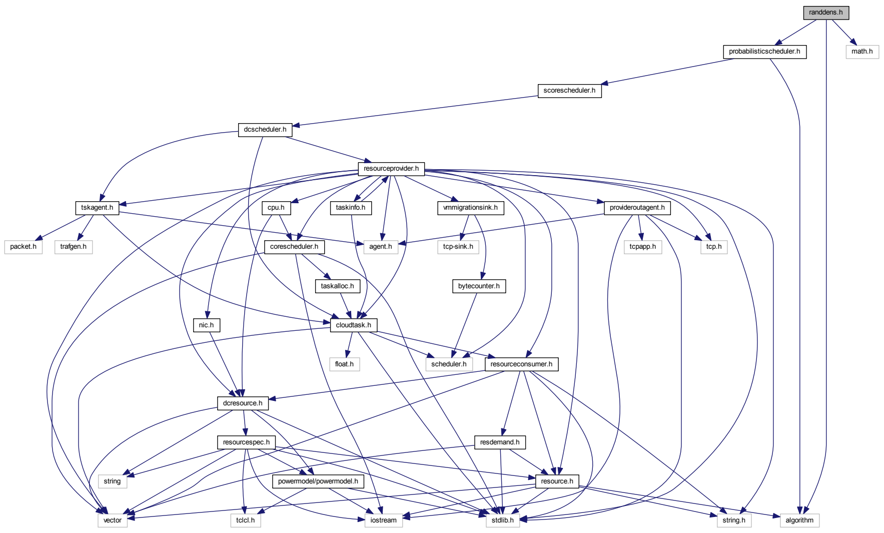
Include dependency graph for randdens.cc:



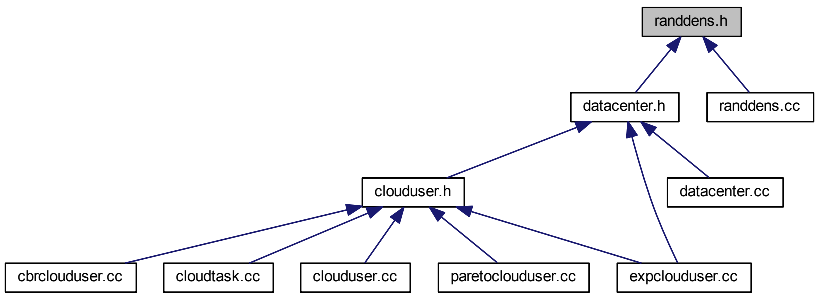
5.47 randdens.h File Reference

```
#include <algorithm>
#include <math.h>
#include "probabilisticscheduler.h"
```

Include dependency graph for randdens.h:



This graph shows which files directly or indirectly include this file:



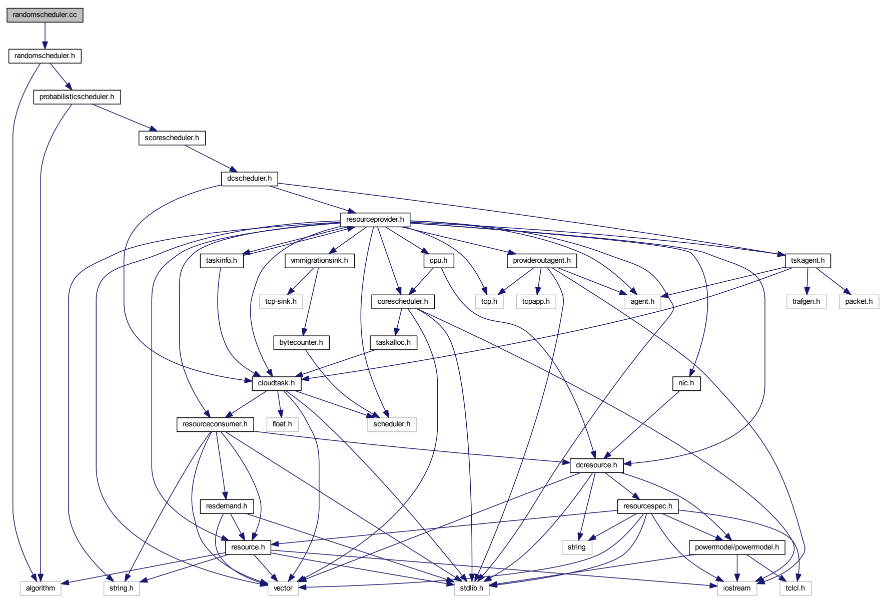
Classes

- class RandDENS

5.48 randomscheduler.cc File Reference

```
#include "randomscheduler.h"
```

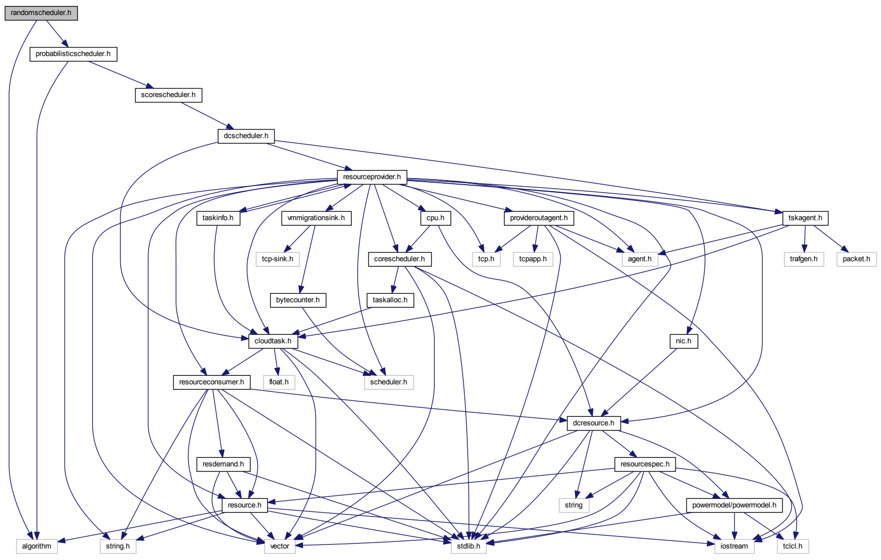
Include dependency graph for randomscheduler.cc:



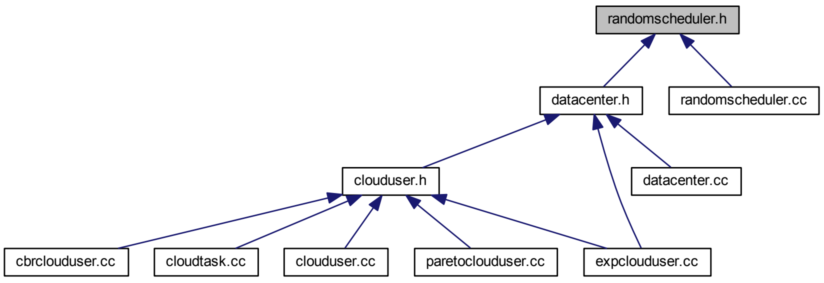
5.49 randomscheduler.h File Reference

```
#include <algorithm>
#include "probabilisticscheduler.h"
```

Include dependency graph for randomscheduler.h:



This graph shows which files directly or indirectly include this file:



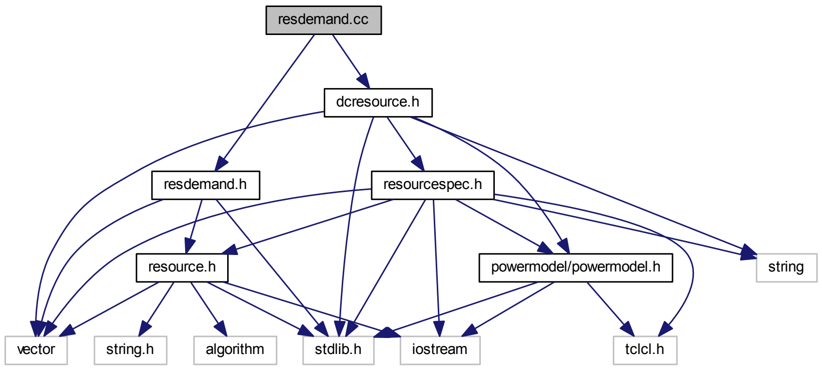
Classes

- class RandomScheduler

5.50 resdemand.cc File Reference

```
#include "resdemand.h"
#include "dcresource.h"
```

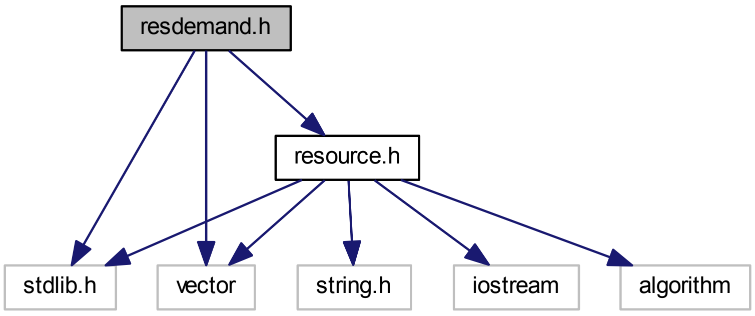
Include dependency graph for resdemand.cc:



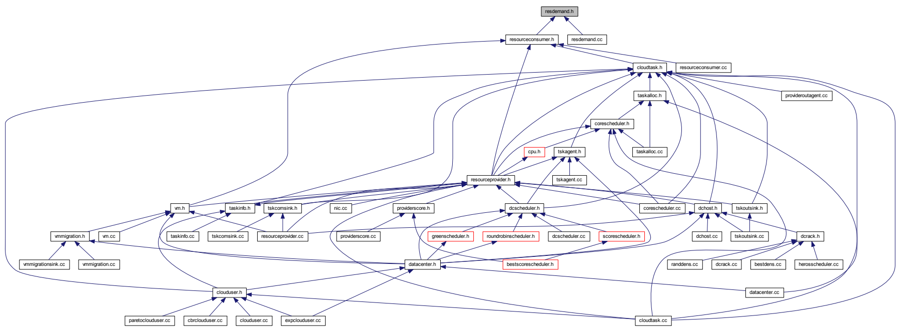
5.51 resdemand.h File Reference

```
#include <stdlib.h>
#include <vector>
#include "resource.h"
```

Include dependency graph for resdemand.h:



This graph shows which files directly or indirectly include this file:



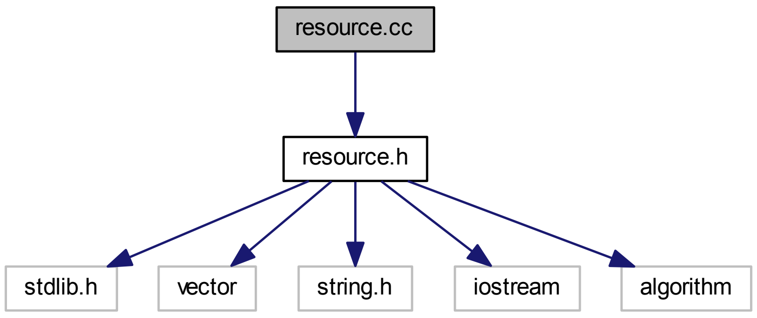
Classes

- class ResDemand

5.52 resource.cc File Reference

```
#include "resource.h"
```

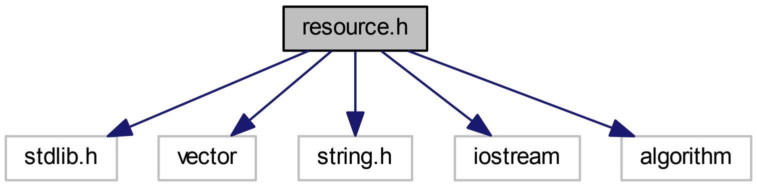
Include dependency graph for resource.cc:



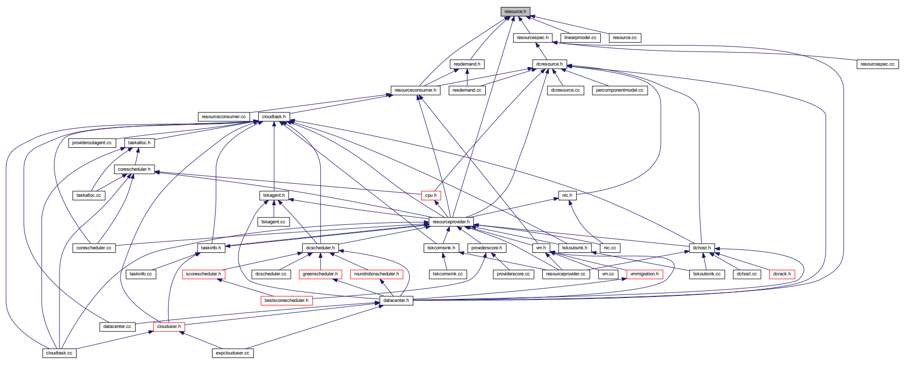
5.53 resource.h File Reference

```
#include <stdlib.h>
#include <vector>
#include <string.h>
#include <iostream>
#include <algorithm>
```

Include dependency graph for resource.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Capacity
- class Resource

Enumerations

- enum res_type

Functions

- bool operator== (const Capacity &lhs, const Capacity &rhs)
- bool operator!= (const Capacity &lhs, const Capacity &rhs)
- bool operator$<$ (const Capacity &lhs, const Capacity &rhs)
- bool operator$>$ (const Capacity &lhs, const Capacity &rhs)
- bool operator$<=$ (const Capacity &lhs, const Capacity &rhs)
- bool operator$>=$ (const Capacity &lhs, const Capacity &rhs)
- Capacity operator+ (Capacity lhs, const Capacity &rhs)
- Capacity operator- (Capacity lhs, const Capacity &rhs)
- bool operator== (const Capacity &lhs, const double &rhs)
- bool operator!= (const Capacity &lhs, const double &rhs)
- bool operator$<$ (const Capacity &lhs, const double &rhs)
- bool operator$>$ (const Capacity &lhs, const double &rhs)
- bool operator$<=$ (const Capacity &lhs, const double &rhs)
- bool operator$>=$ (const Capacity &lhs, const double &rhs)
- Capacity operator+ (Capacity lhs, const double &rhs)
- Capacity operator- (Capacity lhs, const double &rhs)

5.53.1 Enumeration Type Documentation

5.53.1.1 enum res_type

Enumerator

Computing

Memory

Storage

Networking

FirstResType

LastResType

Definition at line 19 of file resource.h.

```
                 {
                   Computing,
                   Memory,
                   Storage,
                   Networking,
                   FirstResType = Computing,
                   LastResType = Networking
};
```

5.53.2 Function Documentation

5.53.2.1 bool operator!= (const Capacity & *lhs*, const Capacity & *rhs*) [inline]

Definition at line 48 of file resource.h.

```
{return !operator==(lhs,rhs);}
```

5.53.2.2 bool operator!= (const Capacity & *lhs*, const double & *rhs*) [inline]

Definition at line 65 of file resource.h.

```
{return !operator==(lhs,rhs);}
```

5.53.2.3 Capacity operator+ (Capacity *lhs*, const Capacity & *rhs*) [inline]

Definition at line 54 of file resource.h.

```
{
  lhs += rhs;
  return lhs;
}
```

5.53.2.4 Capacity operator+ (Capacity *lhs*, const double & *rhs*) [inline]

Definition at line 71 of file resource.h.

```
{
  lhs += rhs;
  return lhs;
}
```

5.53.2.5 Capacity operator- (Capacity *lhs*, const Capacity & *rhs*) [inline]

Definition at line 59 of file resource.h.

```
{
  lhs -= rhs;
  return lhs;
}
```

5.53.2.6 Capacity operator- (Capacity *lhs*, const double & *rhs*) [inline]

Definition at line 76 of file resource.h.

```
{
  lhs -= rhs;
  return lhs;
}
```

5.53.2.7 bool operator< (const Capacity & *lhs*, const Capacity & *rhs*) [inline]

Definition at line 49 of file resource.h.

```
{ return (lhs.value < rhs.value);}
```

5.53.2.8 bool operator< (const Capacity & *lhs*, const double & *rhs*) [inline]

Definition at line 66 of file resource.h.

```
{ return (lhs.value < rhs);}
```

5.53.2.9 bool operator<= (const Capacity & *lhs*, const Capacity & *rhs*) [inline]

Definition at line 51 of file resource.h.

```
{return !operator> (lhs,rhs);}
```

5.53.2.10 bool operator<= (const Capacity & *lhs*, const double & *rhs*) [inline]

Definition at line 68 of file resource.h.

```
{return !operator> (lhs,rhs);}
```

5.53.2.11 bool operator== (const Capacity & *lhs*, const Capacity & *rhs*) [inline]

Definition at line 47 of file resource.h.

```
{ return (lhs.value ==rhs.value);}
```

5.53.2.12 bool operator== (const Capacity & *lhs*, const double & *rhs*) [inline]

Definition at line 64 of file resource.h.

```
{ return (lhs.value ==rhs);}
```

5.53.2.13 bool operator> (const Capacity & *lhs*, const Capacity & *rhs*) [inline]

Definition at line 50 of file resource.h.

```
{return  operator< (rhs,lhs);}
```

5.53.2.14 bool operator> (const Capacity & *lhs*, const double & *rhs*) [inline]

Definition at line 67 of file resource.h.

```
{return  operator< (rhs,lhs);}
```

5.53.2.15 bool operator>= (const Capacity & *lhs*, const Capacity & *rhs*) [inline]

Definition at line 52 of file resource.h.

```
{return !operator< (lhs,rhs);}
```

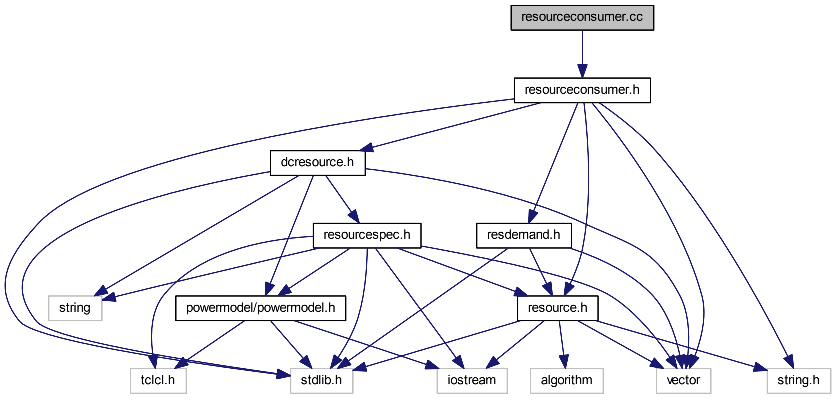
5.53.2.16 bool operator>= (const Capacity & *lhs*, const double & *rhs*) [inline]

Definition at line 69 of file resource.h.

```
{return !operator< (lhs,rhs);}
```

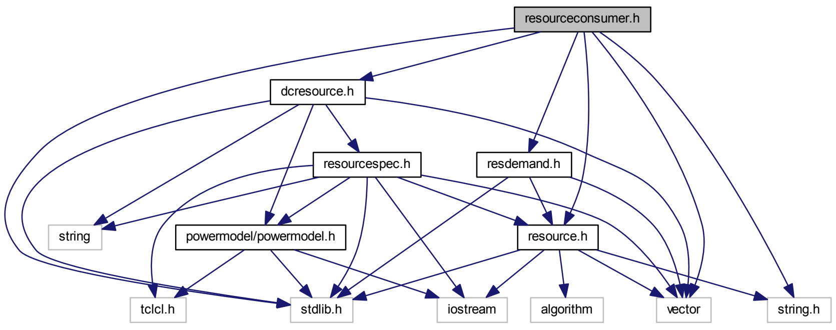
5.54 resourceconsumer.cc File Reference

```
#include "resourceconsumer.h"
```
Include dependency graph for resourceconsumer.cc:

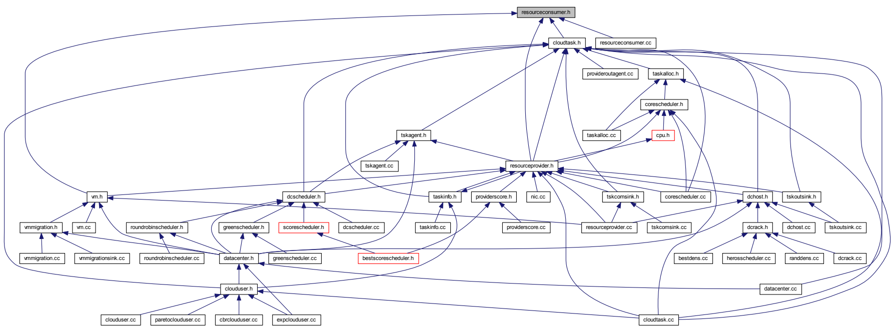


5.55 resourceconsumer.h File Reference

```
#include "resdemand.h"
#include "resource.h"
#include <stdlib.h>
#include <vector>
#include <string.h>
#include "dcresource.h"
```
Include dependency graph for resourceconsumer.h:



This graph shows which files directly or indirectly include this file:



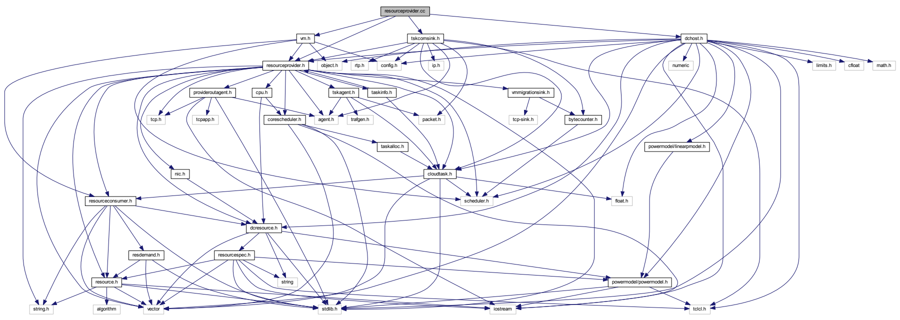
Classes

- class ResourceConsumer

5.56 resourceprovider.cc File Reference

```
#include "resourceprovider.h"
#include "vm.h"
#include "tskcomsink.h"
#include "dchost.h"
```

Include dependency graph for resourceprovider.cc:

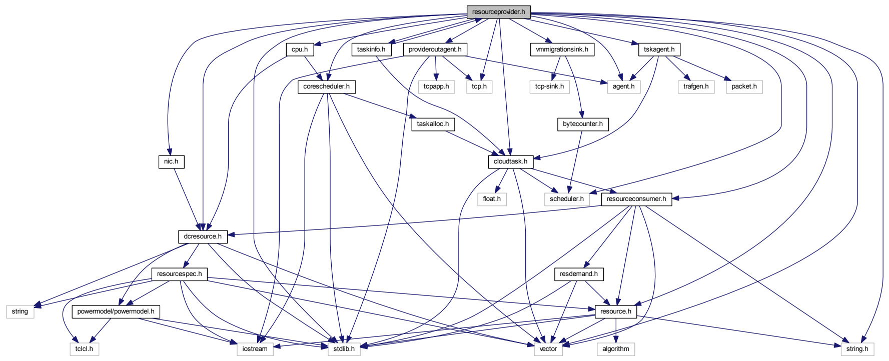


5.57 resourceprovider.h File Reference

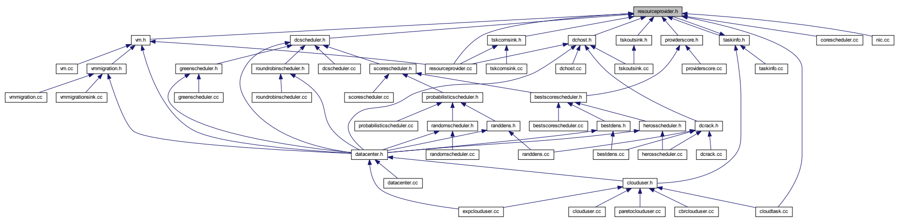
```
#include <stdlib.h>
```

```
#include <vector>
#include <string.h>
#include "resource.h"
#include "dcresource.h"
#include "cpu.h"
#include "nic.h"
#include "resourceconsumer.h"
#include "cloudtask.h"
#include "scheduler.h"
#include "agent.h"
#include "tcp.h"
#include "provideroutagent.h"
#include "vmmigrationsink.h"
#include "corescheduler.h"
#include "tskagent.h"
#include "taskinfo.h"
```

Include dependency graph for resourceprovider.h:



This graph shows which files directly or indirectly include this file:



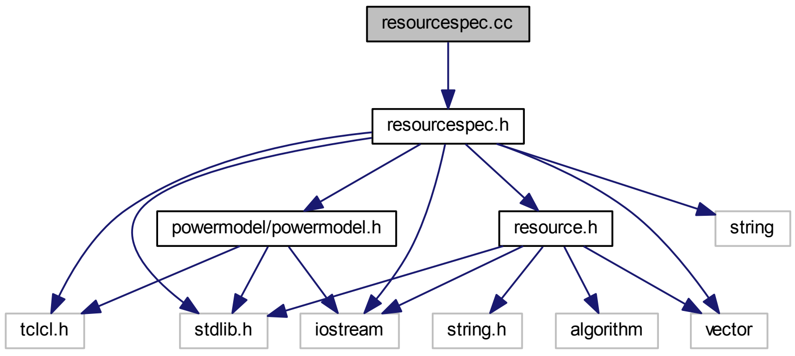
Classes

- class ResourceProvider

5.58 resourcespec.cc File Reference

```
#include "resourcespec.h"
```

Include dependency graph for resourcespec.cc:



Classes

- class ResourceSpecClass

Variables

- ResourceSpecClass class_resourcespec

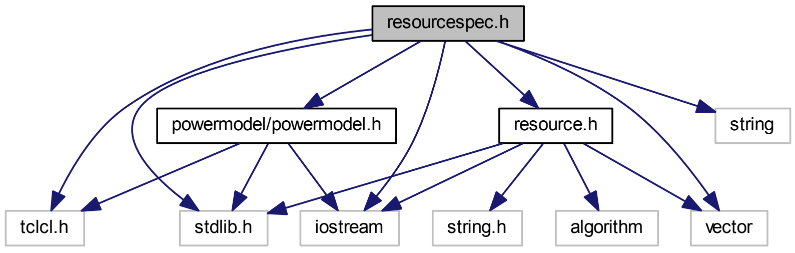
5.58.1 Variable Documentation

5.58.1.1 ResourceSpecClass class_resourcespec `[static]`

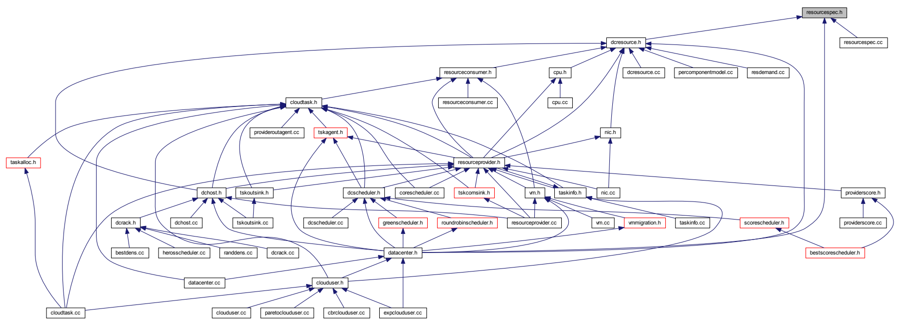
5.59 resourcespec.h File Reference

```
#include "resource.h"
#include "powermodel/powermodel.h"
#include <stdlib.h>
#include <vector>
#include <tclcl.h>
#include <iostream>
#include <string>
```

Include dependency graph for resourcespec.h:



This graph shows which files directly or indirectly include this file:



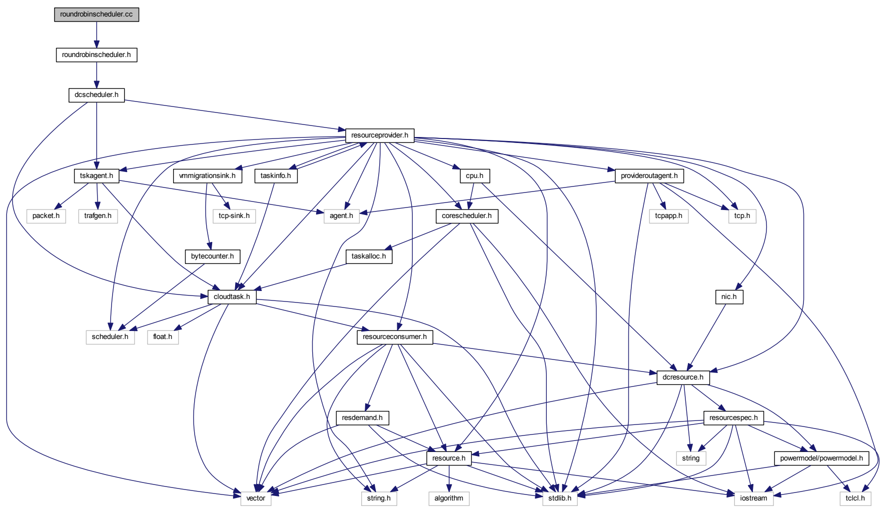
Classes

- class ResourceSpec

5.60 roundrobinscheduler.cc File Reference

```
#include "roundrobinscheduler.h"
```

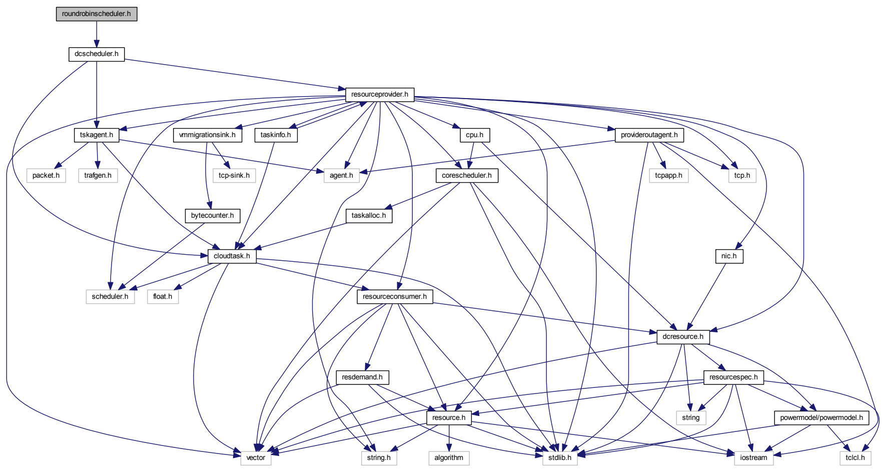
Include dependency graph for roundrobinscheduler.cc:



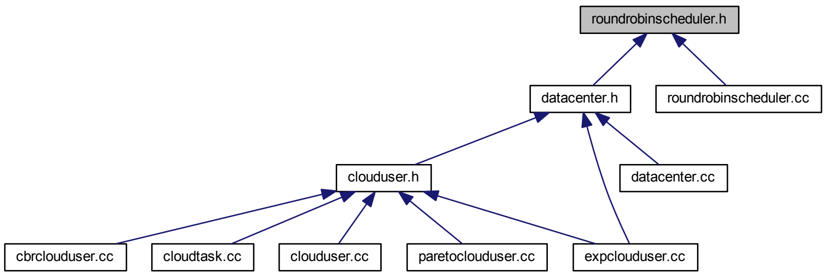
5.61 roundrobinscheduler.h File Reference

```
#include "dcscheduler.h"
```

Include dependency graph for roundrobinscheduler.h:



This graph shows which files directly or indirectly include this file:



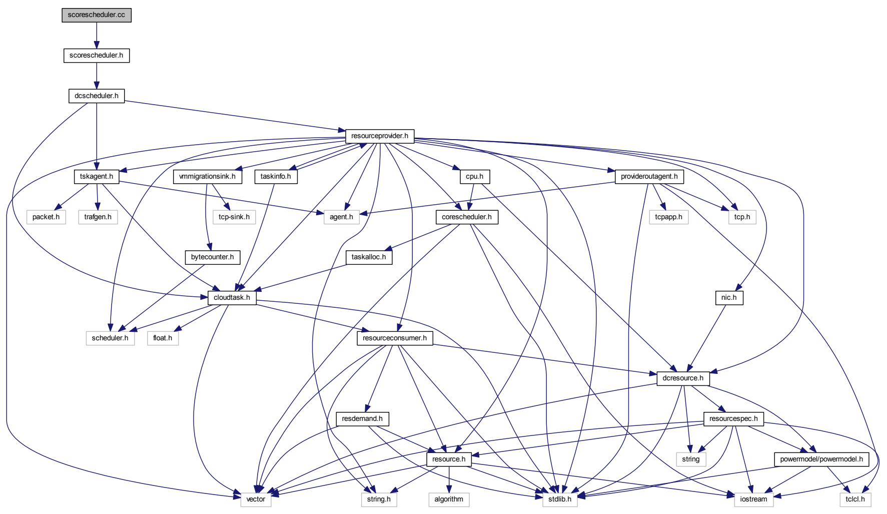
Classes

- class RoundRobinsScheduler

5.62 scorescheduler.cc File Reference

```
#include "scorescheduler.h"
```

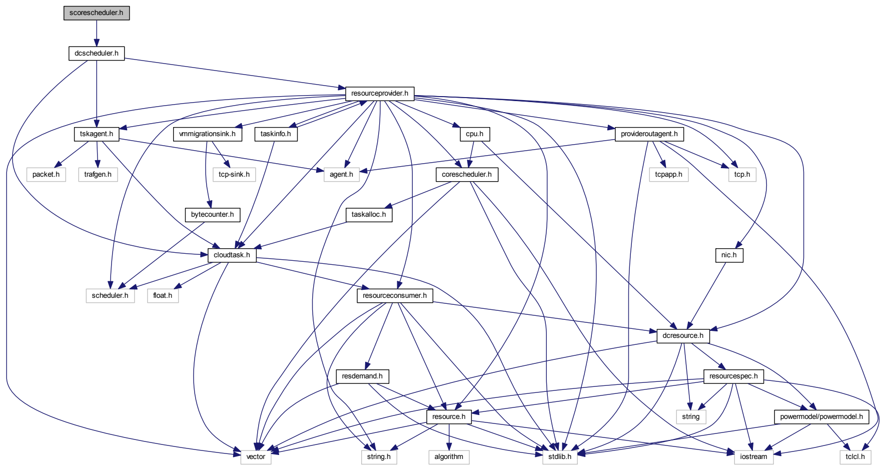
Include dependency graph for scorescheduler.cc:



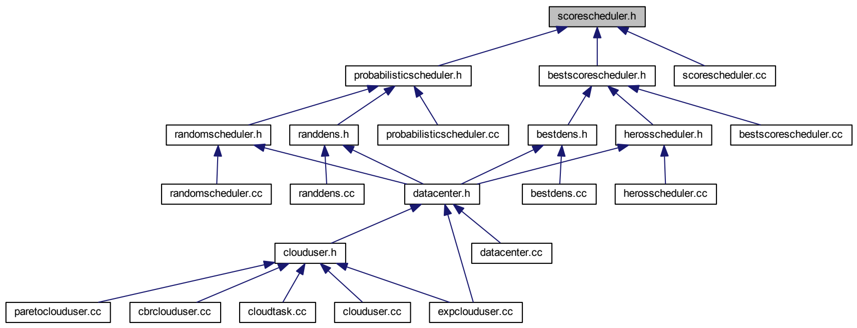
5.63 scorescheduler.h File Reference

```
#include "dcscheduler.h"
```

Include dependency graph for scorescheduler.h:



This graph shows which files directly or indirectly include this file:



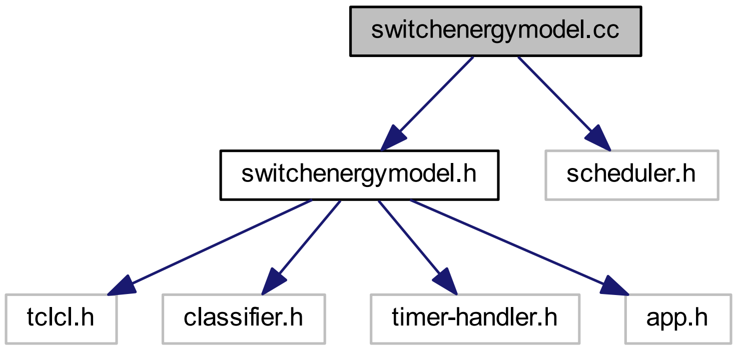
Classes

- class ScoreScheduler

5.64 switchenergymodel.cc File Reference

```
#include "switchenergymodel.h"
#include "scheduler.h"
```

Include dependency graph for switchenergymodel.cc:



Classes

- class SwitchEnergyModelClass

Variables

- SwitchEnergyModelClass class_switchenergymodel

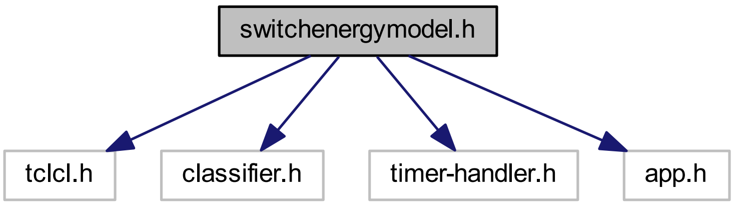
5.64.1 Variable Documentation

5.64.1.1 SwitchEnergyModelClass class_switchenergymodel [static]

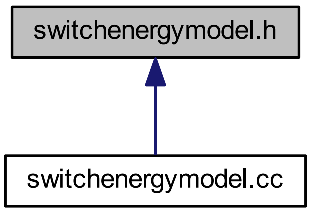
5.65 switchenergymodel.h File Reference

```
#include "tclcl.h"
#include "classifier.h"
#include "timer-handler.h"
#include "app.h"
```

Include dependency graph for switchenergymodel.h:



This graph shows which files directly or indirectly include this file:

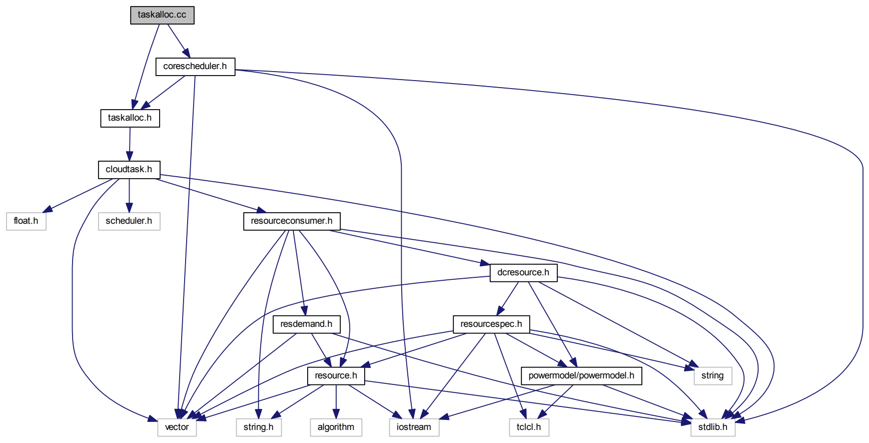


Classes

- class SwitchEnergyTimer
- class SwitchEnergyModel

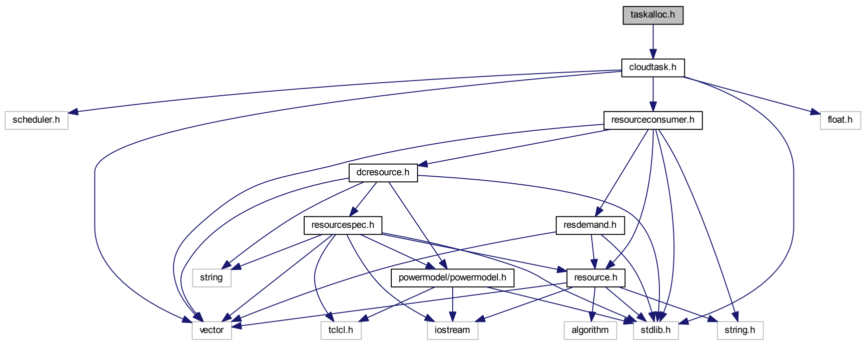
5.66 taskalloc.cc File Reference

```
#include "taskalloc.h"
#include "corescheduler.h"
```
Include dependency graph for taskalloc.cc:

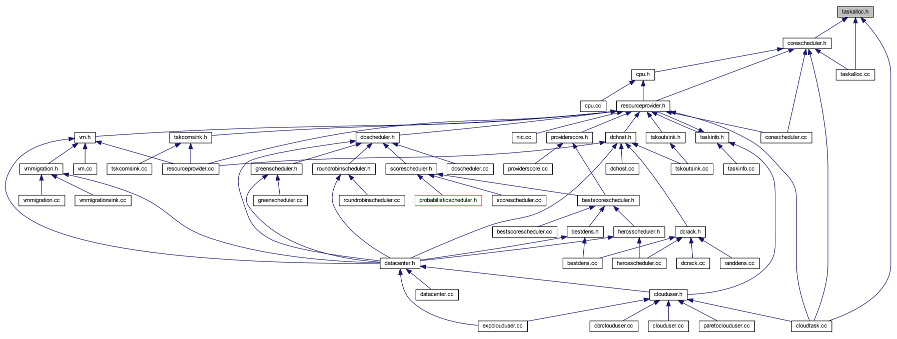


5.67 taskalloc.h File Reference

```
#include "cloudtask.h"
```
Include dependency graph for taskalloc.h:



This graph shows which files directly or indirectly include this file:



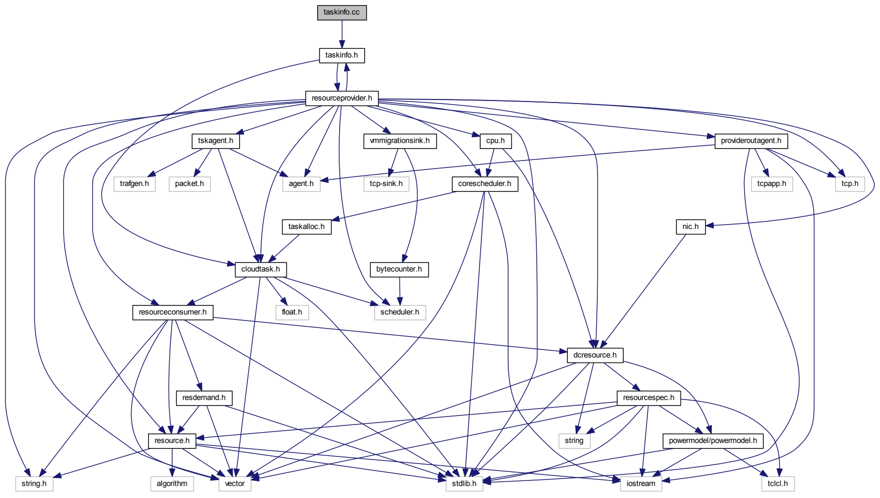
Classes

- class TaskAlloc

5.68 taskinfo.cc File Reference

```
#include "taskinfo.h"
```

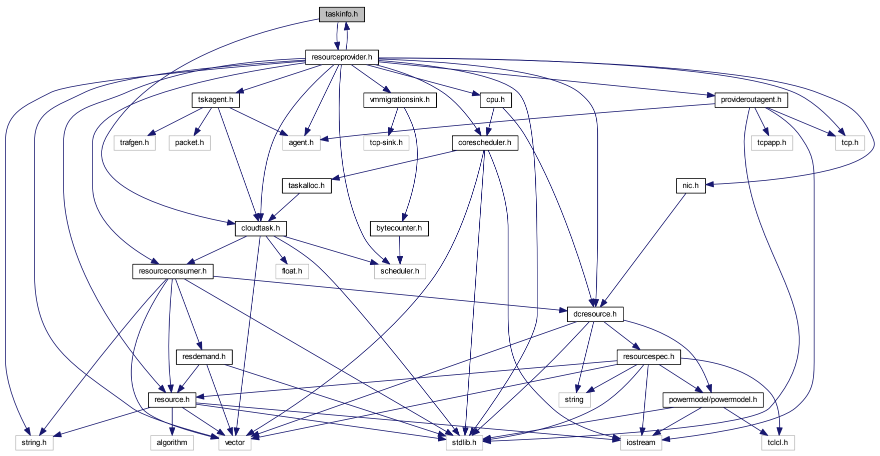
Include dependency graph for taskinfo.cc:



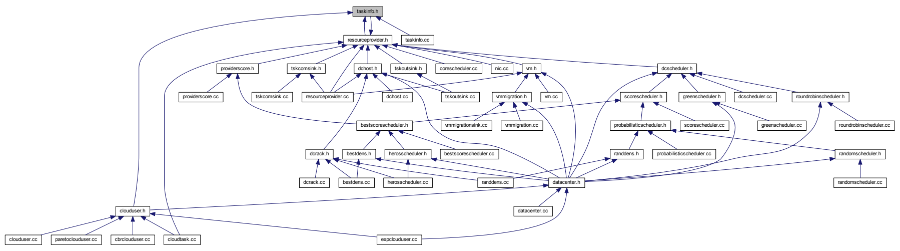
5.69 taskinfo.h File Reference

```
#include "cloudtask.h"
#include "resourceprovider.h"
```

Include dependency graph for taskinfo.h:



This graph shows which files directly or indirectly include this file:



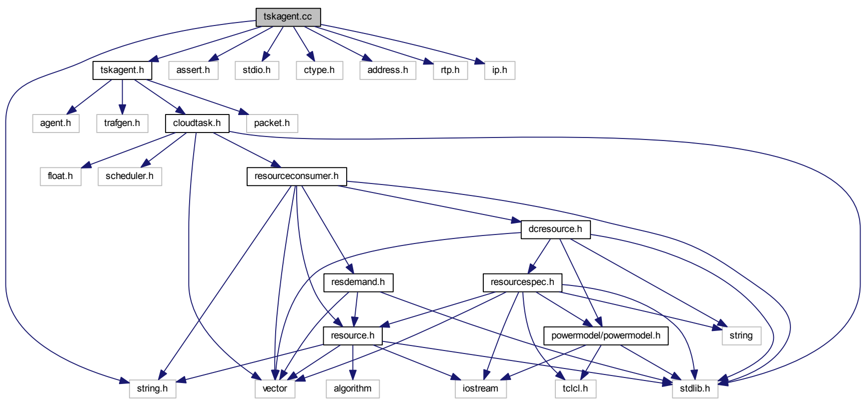
Classes

- class TaskInfo

5.70 tskagent.cc File Reference

```
#include "tskagent.h"
#include <string.h>
#include <assert.h>
#include <stdio.h>
#include <ctype.h>
#include "address.h"
#include "rtp.h"
#include "ip.h"
```

Include dependency graph for tskagent.cc:



Classes

- class TskComAgentClass

Variables

- static const char rcsid [ ]
- TskComAgentClass class_tsk_comagent

5.70.1 Variable Documentation

5.70.1.1 TskComAgentClass class_tsk_comagent [static]

5.70.1.2 const char rcsid[ ] [static]

Initial value:

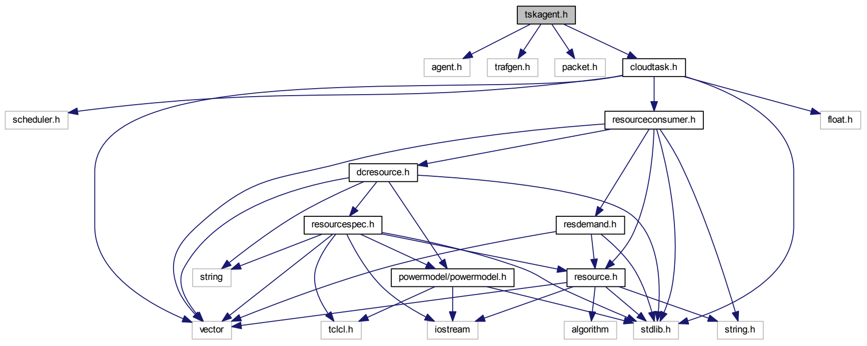
```
=
                                "@(#) $Header: /cvsroot/nsnam/ns-2/apps/tskagent.cc,v 1.21 2005/08/26
        05:05:28 tomh Exp $ (Xerox)"
```

Definition at line 6 of file tskagent.cc.

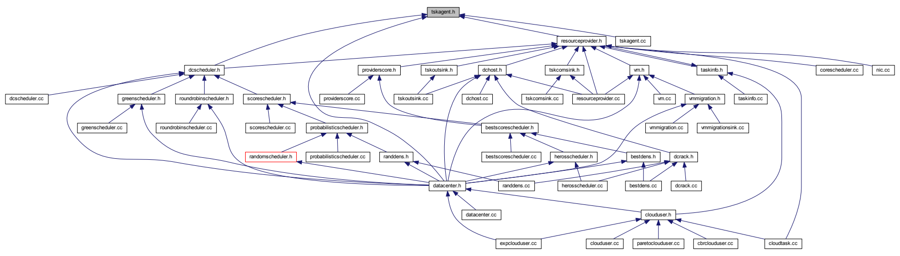
5.71 tskagent.h File Reference

```
#include "agent.h"
#include "trafgen.h"
#include "packet.h"
#include "cloudtask.h"
```

Include dependency graph for tskagent.h:



This graph shows which files directly or indirectly include this file:



Classes

- class TskComAgent

Macros

- #define SAMPLERATE 8000
- #define RTP_M 0x0080

5.71.1 Macro Definition Documentation

5.71.1.1 #define RTP_M 0x0080

Definition at line 15 of file tskagent.h.

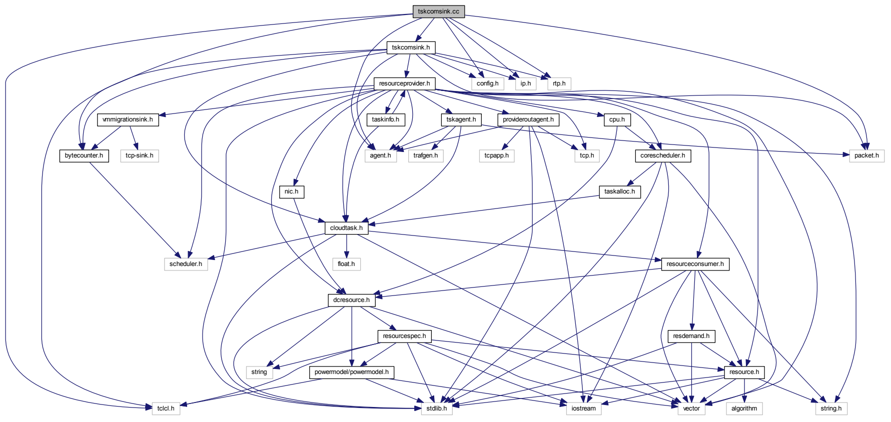
5.71.1.2 #define SAMPLERATE 8000

Definition at line 14 of file tskagent.h.

5.72 tskcomsink.cc File Reference

```
#include <tclcl.h>
#include "agent.h"
#include "config.h"
#include "packet.h"
#include "ip.h"
#include "rtp.h"
#include "bytecounter.h"
#include "tskcomsink.h"
```

Include dependency graph for tskcomsink.cc:



Classes

- class TskComSinkClass

Variables

- static const char rcsid [ ]
- TskComSinkClass class_tsk_comsink

5.72.1 Variable Documentation

5.72.1.1 TskComSinkClass class_tsk_comsink [static]

5.72.1.2 const char rcsid[ ] [static]

Initial value:

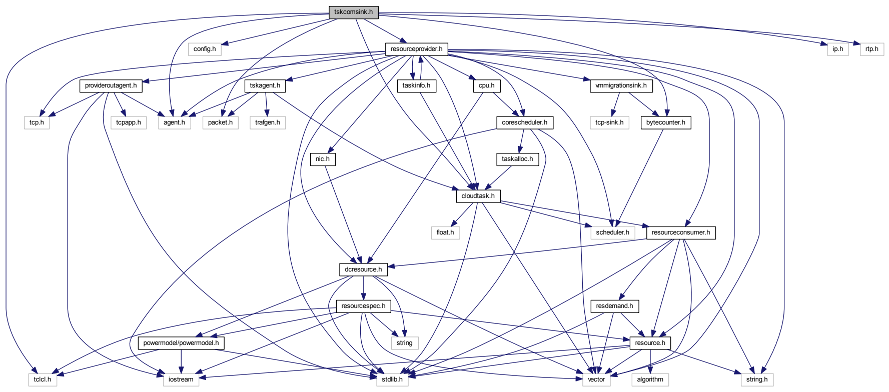
```
=
                              "@(#) $Header: /cvsroot/nsnam/ns-2/tools/tskcomsink.cc,v 1.18 2000/09/01
    03:04:06 haoboy Exp $ (LBL)"
```

Definition at line 6 of file tskcomsink.cc.

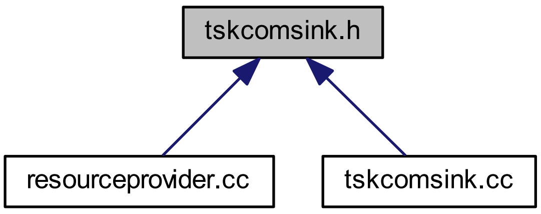
5.73 tskcomsink.h File Reference

```
#include <tclcl.h>
#include "agent.h"
#include "config.h"
#include "packet.h"
#include "resourceprovider.h"
#include "ip.h"
#include "rtp.h"
#include "cloudtask.h"
#include "bytecounter.h"
```

Include dependency graph for tskcomsink.h:



This graph shows which files directly or indirectly include this file:



Classes

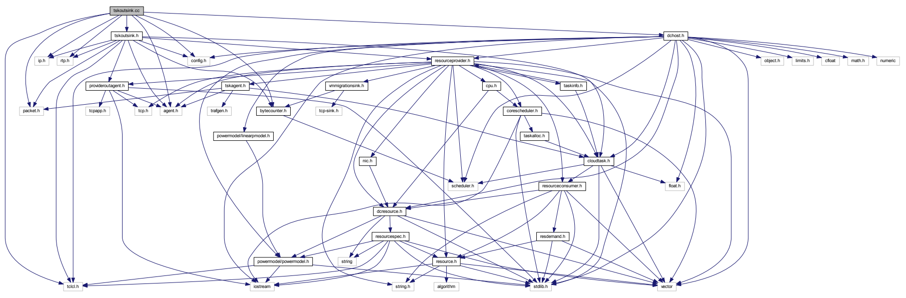
- class TskComSink

5.74 tskoutsink.cc File Reference

```
#include <tclcl.h>
```

```
#include "agent.h"
#include "config.h"
#include "packet.h"
#include "ip.h"
#include "rtp.h"
#include "bytecounter.h"
#include "tskoutsink.h"
#include "dchost.h"
```

Include dependency graph for tskoutsink.cc:



Classes

- class TskOutSinkClass

Variables

- static const char rcsid [ ]
- TskOutSinkClass class_tsk_outsink

5.74.1 Variable Documentation

5.74.1.1 TskOutSinkClass class_tsk_outsink [static]

5.74.1.2 const char rcsid[ ] [static]

Initial value:

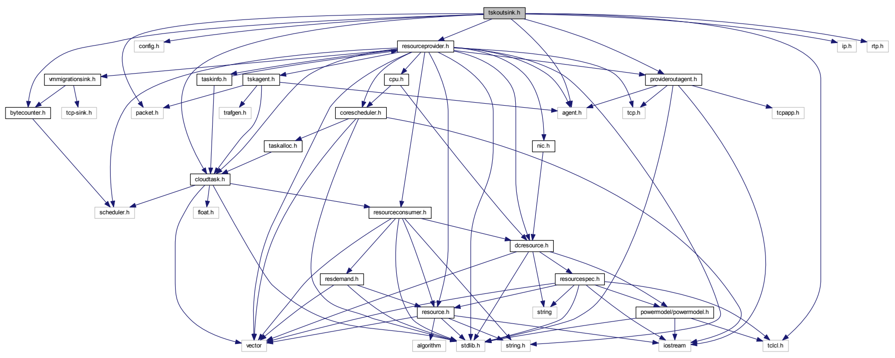
```
=
                             "@(#) $Header: /cvsroot/nsnam/ns-2/tools/TskOutSink.cc,v 1.18 2000/09/01
      03:04:06 haoboy Exp $ (LBL)"
```

Definition at line 6 of file tskoutsink.cc.

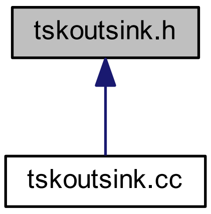
5.75 tskoutsink.h File Reference

```
#include <tclcl.h>
#include "agent.h"
#include "config.h"
#include "packet.h"
#include "resourceprovider.h"
#include "ip.h"
#include "rtp.h"
#include "cloudtask.h"
#include "bytecounter.h"
#include "provideroutagent.h"
```

Include dependency graph for tskoutsink.h:



This graph shows which files directly or indirectly include this file:



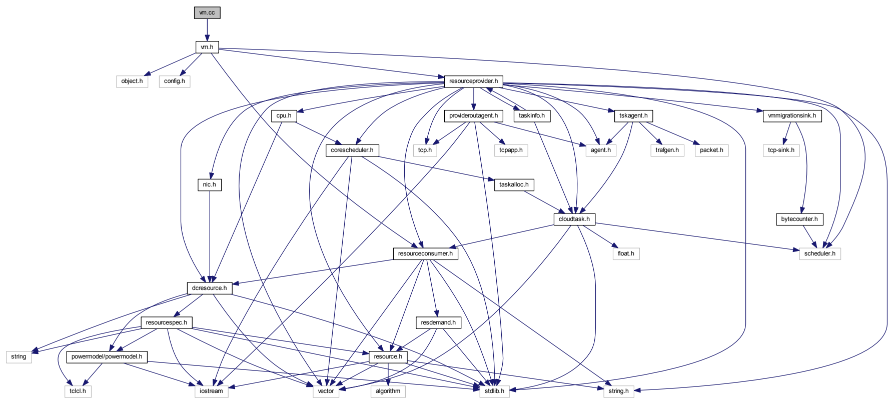
Classes

- class TskOutSink

5.76 vm.cc File Reference

```
#include "vm.h"
```

Include dependency graph for vm.cc:



Classes

- class VMClass

Variables

- VMClass class_vm

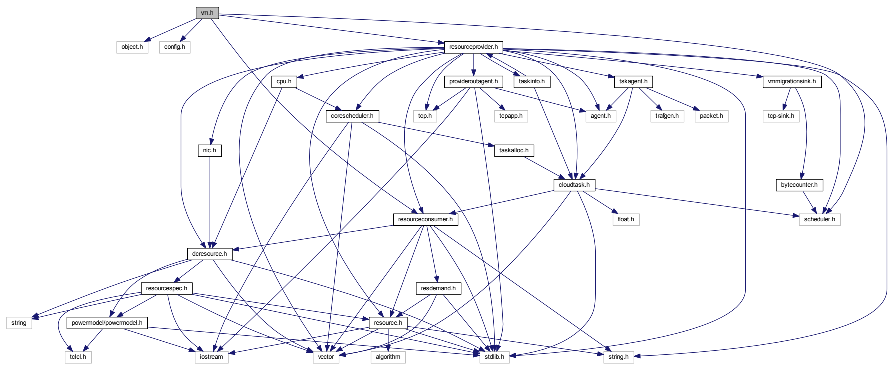
5.76.1 Variable Documentation

5.76.1.1 VMClass class_vm [static]

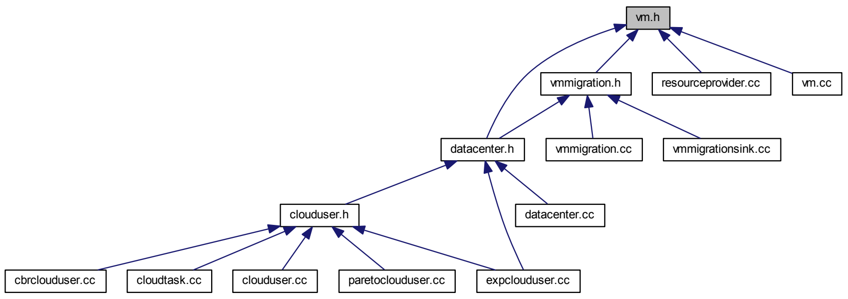
5.77 vm.h File Reference

```
#include "object.h"
#include "config.h"
#include "scheduler.h"
#include "resourceprovider.h"
#include "resourceconsumer.h"
```

Include dependency graph for vm.h:



This graph shows which files directly or indirectly include this file:



Classes

- class VM

Enumerations

- enum vm_state

5.77.1 Enumeration Type Documentation

5.77.1.1 enum vm_state

Enumerator

Ready

Running

Suspended

Stopped

Dead

FirstVmState

LastVmState

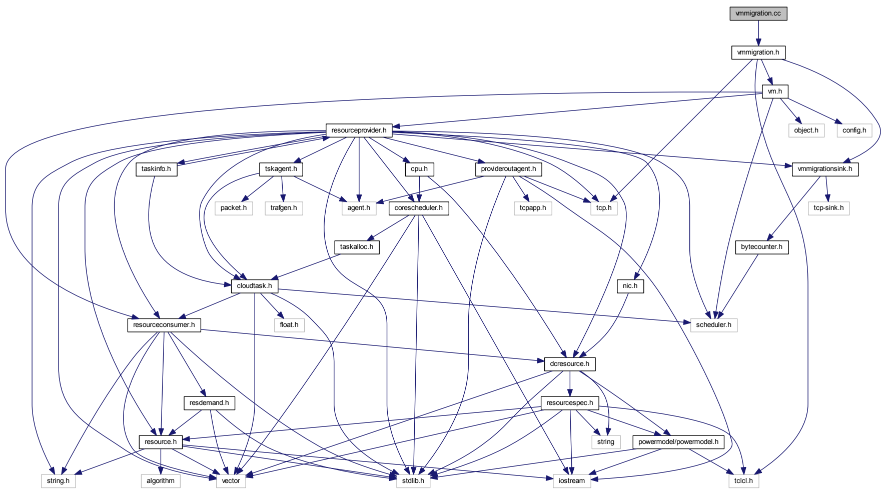
Definition at line 19 of file vm.h.

```
               {
                 Ready,
                 Running,
                 Suspended,
                 Stopped,
                 Dead,
                 FirstVmState = Ready,
                 LastVmState = Dead
 };
```

5.78 vmmigration.cc File Reference

```
#include "vmmigration.h"
```

Include dependency graph for vmmigration.cc:



Classes

- class VmMigrationClass

Variables

- VmMigrationClass class_vmmigration

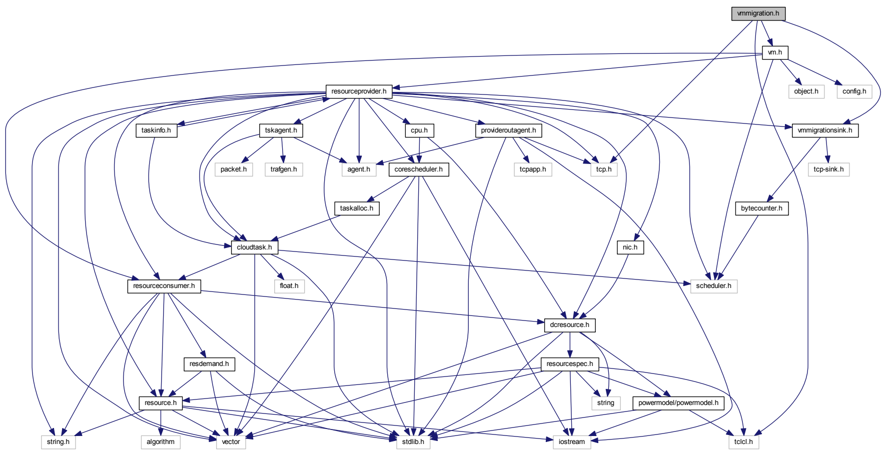
5.78.1 Variable Documentation

5.78.1.1 VmMigrationClass class_vmmigration [static]

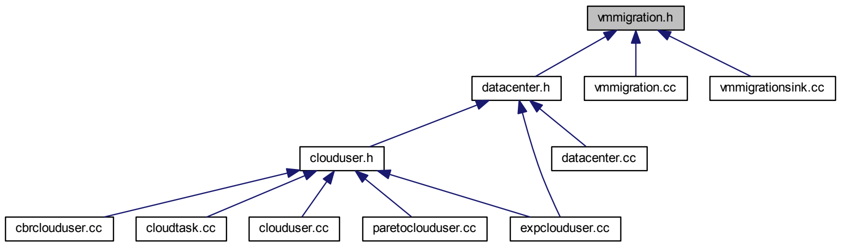
5.79 vmmigration.h File Reference

```
#include <tclcl.h>
#include "tcp.h"
#include "vmmigrationsink.h"
#include "vm.h"
```

Include dependency graph for vmmigration.h:



This graph shows which files directly or indirectly include this file:



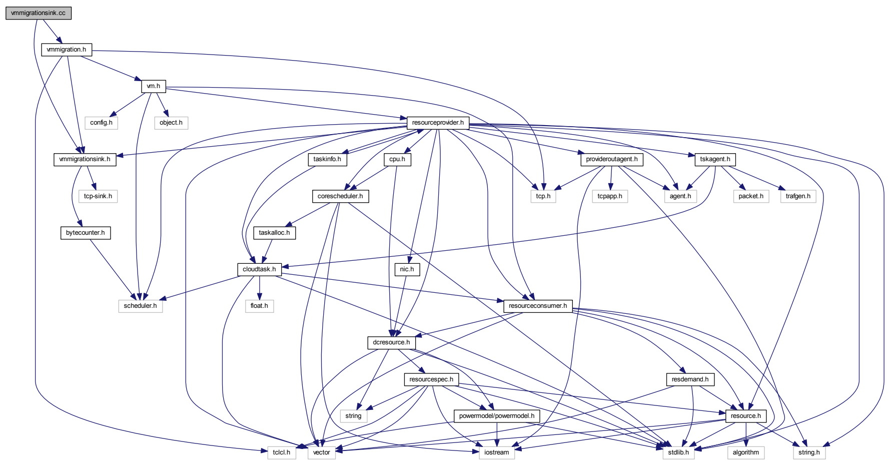
Classes

- class VmMigration

5.80 vmmigrationsink.cc File Reference

```
#include "vmmigrationsink.h"
#include "vmmigration.h"
```

Include dependency graph for vmmigrationsink.cc:



Classes

- class VmMigrationSinkClass

Variables

- VmMigrationSinkClass class_vm_migrationsink

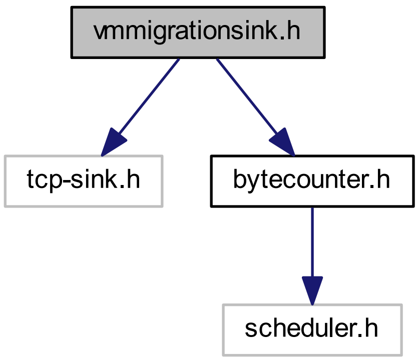
5.80.1 Variable Documentation

5.80.1.1 VmMigrationSinkClass class_vm_migrationsink [static]

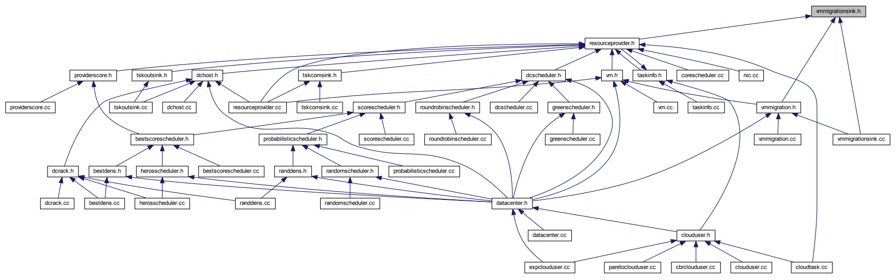
5.81 vmmigrationsink.h File Reference

```
#include "tcp-sink.h"
#include "bytecounter.h"
```

Include dependency graph for vmmigrationsink.h:



This graph shows which files directly or indirectly include this file:



Classes

- class VmMigrationSink

Index