

## GreenCloud Simulator

Generated by Doxygen 1.8.11

## Contents

<b>1</b>	<b>Hierarchical Index</b>	<b>2</b>
1.1	Class Hierarchy . . . . .	2
<b>2</b>	<b>Class Index</b>	<b>4</b>
2.1	Class List . . . . .	4
<b>3</b>	<b>File Index</b>	<b>7</b>
3.1	File List . . . . .	7
<b>4</b>	<b>Class Documentation</b>	<b>9</b>
4.1	BestDENS Class Reference . . . . .	9
4.1.1	Detailed Description . . . . .	11
4.1.2	Constructor & Destructor Documentation . . . . .	11
4.1.3	Member Function Documentation . . . . .	11
4.1.4	Member Data Documentation . . . . .	12
4.2	BestScoreScheduler Class Reference . . . . .	13
4.2.1	Detailed Description . . . . .	14
4.2.2	Constructor & Destructor Documentation . . . . .	14
4.2.3	Member Function Documentation . . . . .	14
4.3	ByteCounter Class Reference . . . . .	15
4.3.1	Detailed Description . . . . .	15
4.3.2	Constructor & Destructor Documentation . . . . .	15
4.3.3	Member Function Documentation . . . . .	16
4.3.4	Member Data Documentation . . . . .	16
4.4	Capacity Class Reference . . . . .	16
4.4.1	Detailed Description . . . . .	17
4.4.2	Constructor & Destructor Documentation . . . . .	17
4.4.3	Member Function Documentation . . . . .	18
4.4.4	Member Data Documentation . . . . .	19
4.5	CBRCloudUser Class Reference . . . . .	20

4.5.1	Detailed Description . . . . .	21
4.5.2	Constructor & Destructor Documentation . . . . .	21
4.5.3	Member Function Documentation . . . . .	21
4.5.4	Member Data Documentation . . . . .	23
4.6	CBRCloudUserClass Class Reference . . . . .	23
4.6.1	Detailed Description . . . . .	24
4.6.2	Constructor & Destructor Documentation . . . . .	24
4.6.3	Member Function Documentation . . . . .	24
4.7	CloudTask Class Reference . . . . .	24
4.7.1	Detailed Description . . . . .	26
4.7.2	Constructor & Destructor Documentation . . . . .	26
4.7.3	Member Function Documentation . . . . .	27
4.7.4	Member Data Documentation . . . . .	30
4.8	CloudUser Class Reference . . . . .	31
4.8.1	Detailed Description . . . . .	32
4.8.2	Constructor & Destructor Documentation . . . . .	32
4.8.3	Member Function Documentation . . . . .	33
4.8.4	Member Data Documentation . . . . .	35
4.9	CoreScheduler Class Reference . . . . .	37
4.9.1	Detailed Description . . . . .	38
4.9.2	Constructor & Destructor Documentation . . . . .	39
4.9.3	Member Function Documentation . . . . .	39
4.9.4	Member Data Documentation . . . . .	45
4.10	CPU Class Reference . . . . .	47
4.10.1	Detailed Description . . . . .	48
4.10.2	Constructor & Destructor Documentation . . . . .	48
4.10.3	Member Function Documentation . . . . .	49
4.10.4	Member Data Documentation . . . . .	51
4.11	CpuClass Class Reference . . . . .	52
4.11.1	Detailed Description . . . . .	52

4.11.2	Constructor & Destructor Documentation . . . . .	52
4.11.3	Member Function Documentation . . . . .	53
4.12	DataCenter Class Reference . . . . .	53
4.12.1	Detailed Description . . . . .	55
4.12.2	Constructor & Destructor Documentation . . . . .	55
4.12.3	Member Function Documentation . . . . .	55
4.12.4	Member Data Documentation . . . . .	63
4.13	DataCenterClass Class Reference . . . . .	64
4.13.1	Detailed Description . . . . .	65
4.13.2	Constructor & Destructor Documentation . . . . .	65
4.13.3	Member Function Documentation . . . . .	65
4.14	DcHost Class Reference . . . . .	66
4.14.1	Detailed Description . . . . .	67
4.14.2	Constructor & Destructor Documentation . . . . .	67
4.14.3	Member Function Documentation . . . . .	68
4.14.4	Member Data Documentation . . . . .	70
4.15	DcHostClass Class Reference . . . . .	71
4.15.1	Detailed Description . . . . .	71
4.15.2	Constructor & Destructor Documentation . . . . .	72
4.15.3	Member Function Documentation . . . . .	72
4.16	DcRack Class Reference . . . . .	72
4.16.1	Detailed Description . . . . .	73
4.16.2	Constructor & Destructor Documentation . . . . .	73
4.16.3	Member Function Documentation . . . . .	74
4.16.4	Member Data Documentation . . . . .	75
4.17	DcRackClass Class Reference . . . . .	76
4.17.1	Detailed Description . . . . .	77
4.17.2	Constructor & Destructor Documentation . . . . .	77
4.17.3	Member Function Documentation . . . . .	77
4.18	DcResource Class Reference . . . . .	77

4.18.1 Detailed Description . . . . .	78
4.18.2 Constructor & Destructor Documentation . . . . .	79
4.18.3 Member Function Documentation . . . . .	79
4.18.4 Member Data Documentation . . . . .	81
4.19 DcResourceClass Class Reference . . . . .	81
4.19.1 Detailed Description . . . . .	82
4.19.2 Constructor & Destructor Documentation . . . . .	82
4.19.3 Member Function Documentation . . . . .	82
4.20 DcScheduler Class Reference . . . . .	82
4.20.1 Detailed Description . . . . .	83
4.20.2 Constructor & Destructor Documentation . . . . .	83
4.20.3 Member Function Documentation . . . . .	83
4.21 ExpCloudUser Class Reference . . . . .	84
4.21.1 Detailed Description . . . . .	85
4.21.2 Constructor & Destructor Documentation . . . . .	85
4.21.3 Member Function Documentation . . . . .	85
4.21.4 Member Data Documentation . . . . .	87
4.22 ExpCloudUserClass Class Reference . . . . .	88
4.22.1 Detailed Description . . . . .	89
4.22.2 Constructor & Destructor Documentation . . . . .	89
4.22.3 Member Function Documentation . . . . .	89
4.23 GreenScheduler Class Reference . . . . .	89
4.23.1 Detailed Description . . . . .	90
4.23.2 Constructor & Destructor Documentation . . . . .	90
4.23.3 Member Function Documentation . . . . .	91
4.24 HerosScheduler Class Reference . . . . .	91
4.24.1 Detailed Description . . . . .	93
4.24.2 Constructor & Destructor Documentation . . . . .	93
4.24.3 Member Function Documentation . . . . .	93
4.24.4 Member Data Documentation . . . . .	95

4.25	LinearPModel Class Reference . . . . .	96
4.25.1	Detailed Description . . . . .	97
4.25.2	Constructor & Destructor Documentation . . . . .	97
4.25.3	Member Function Documentation . . . . .	98
4.25.4	Member Data Documentation . . . . .	100
4.26	LinearPModelClass Class Reference . . . . .	101
4.26.1	Detailed Description . . . . .	101
4.26.2	Constructor & Destructor Documentation . . . . .	101
4.26.3	Member Function Documentation . . . . .	102
4.27	NIC Class Reference . . . . .	102
4.27.1	Detailed Description . . . . .	103
4.27.2	Constructor & Destructor Documentation . . . . .	103
4.27.3	Member Function Documentation . . . . .	103
4.27.4	Member Data Documentation . . . . .	104
4.28	NicClass Class Reference . . . . .	104
4.28.1	Detailed Description . . . . .	105
4.28.2	Constructor & Destructor Documentation . . . . .	105
4.28.3	Member Function Documentation . . . . .	105
4.29	ParetoCloudUser Class Reference . . . . .	105
4.29.1	Detailed Description . . . . .	107
4.29.2	Constructor & Destructor Documentation . . . . .	107
4.29.3	Member Function Documentation . . . . .	107
4.29.4	Member Data Documentation . . . . .	109
4.30	PerComponentModel Class Reference . . . . .	110
4.30.1	Detailed Description . . . . .	111
4.30.2	Constructor & Destructor Documentation . . . . .	111
4.30.3	Member Function Documentation . . . . .	111
4.30.4	Member Data Documentation . . . . .	112
4.31	PerComponentModelClass Class Reference . . . . .	113
4.31.1	Detailed Description . . . . .	113

4.31.2	Constructor & Destructor Documentation . . . . .	113
4.31.3	Member Function Documentation . . . . .	114
4.32	PoaBuf Class Reference . . . . .	114
4.32.1	Detailed Description . . . . .	114
4.32.2	Constructor & Destructor Documentation . . . . .	115
4.32.3	Member Function Documentation . . . . .	115
4.32.4	Friends And Related Function Documentation . . . . .	115
4.32.5	Member Data Documentation . . . . .	115
4.33	PoaBufList Class Reference . . . . .	116
4.33.1	Detailed Description . . . . .	116
4.33.2	Constructor & Destructor Documentation . . . . .	117
4.33.3	Member Function Documentation . . . . .	117
4.33.4	Member Data Documentation . . . . .	117
4.34	POOTrafficClass Class Reference . . . . .	118
4.34.1	Detailed Description . . . . .	118
4.34.2	Constructor & Destructor Documentation . . . . .	119
4.34.3	Member Function Documentation . . . . .	119
4.35	PowerModel Class Reference . . . . .	119
4.35.1	Detailed Description . . . . .	120
4.35.2	Constructor & Destructor Documentation . . . . .	120
4.35.3	Member Function Documentation . . . . .	121
4.35.4	Member Data Documentation . . . . .	122
4.36	ProbabilisticScheduler Class Reference . . . . .	122
4.36.1	Detailed Description . . . . .	123
4.36.2	Constructor & Destructor Documentation . . . . .	123
4.36.3	Member Function Documentation . . . . .	124
4.37	ProviderOutAgent Class Reference . . . . .	125
4.37.1	Detailed Description . . . . .	126
4.37.2	Constructor & Destructor Documentation . . . . .	126
4.37.3	Member Function Documentation . . . . .	126

4.37.4	Member Data Documentation . . . . .	127
4.38	ProviderScore Class Reference . . . . .	128
4.38.1	Detailed Description . . . . .	128
4.38.2	Constructor & Destructor Documentation . . . . .	129
4.38.3	Member Function Documentation . . . . .	129
4.38.4	Member Data Documentation . . . . .	129
4.39	ProvOutAgentClass Class Reference . . . . .	130
4.39.1	Detailed Description . . . . .	130
4.39.2	Constructor & Destructor Documentation . . . . .	131
4.39.3	Member Function Documentation . . . . .	131
4.40	RandDENS Class Reference . . . . .	131
4.40.1	Detailed Description . . . . .	132
4.40.2	Constructor & Destructor Documentation . . . . .	133
4.40.3	Member Function Documentation . . . . .	133
4.40.4	Member Data Documentation . . . . .	134
4.41	RandomScheduler Class Reference . . . . .	134
4.41.1	Detailed Description . . . . .	135
4.41.2	Constructor & Destructor Documentation . . . . .	135
4.41.3	Member Function Documentation . . . . .	136
4.42	ResDemand Class Reference . . . . .	137
4.42.1	Detailed Description . . . . .	138
4.42.2	Constructor & Destructor Documentation . . . . .	138
4.42.3	Member Data Documentation . . . . .	138
4.43	Resource Class Reference . . . . .	139
4.43.1	Detailed Description . . . . .	140
4.43.2	Constructor & Destructor Documentation . . . . .	140
4.43.3	Member Function Documentation . . . . .	141
4.43.4	Member Data Documentation . . . . .	142
4.44	ResourceConsumer Class Reference . . . . .	143
4.44.1	Detailed Description . . . . .	143

4.44.2	Constructor & Destructor Documentation . . . . .	143
4.44.3	Member Function Documentation . . . . .	144
4.44.4	Member Data Documentation . . . . .	145
4.45	ResourceProvider Class Reference . . . . .	145
4.45.1	Detailed Description . . . . .	148
4.45.2	Member Enumeration Documentation . . . . .	148
4.45.3	Constructor & Destructor Documentation . . . . .	148
4.45.4	Member Function Documentation . . . . .	149
4.45.5	Member Data Documentation . . . . .	161
4.46	ResourceSpec Class Reference . . . . .	164
4.46.1	Detailed Description . . . . .	165
4.46.2	Constructor & Destructor Documentation . . . . .	165
4.46.3	Member Function Documentation . . . . .	166
4.46.4	Friends And Related Function Documentation . . . . .	167
4.46.5	Member Data Documentation . . . . .	167
4.47	ResourceSpecClass Class Reference . . . . .	168
4.47.1	Detailed Description . . . . .	169
4.47.2	Constructor & Destructor Documentation . . . . .	169
4.47.3	Member Function Documentation . . . . .	169
4.48	RoundRobinsScheduler Class Reference . . . . .	169
4.48.1	Detailed Description . . . . .	170
4.48.2	Constructor & Destructor Documentation . . . . .	170
4.48.3	Member Function Documentation . . . . .	171
4.49	ScoreScheduler Class Reference . . . . .	171
4.49.1	Detailed Description . . . . .	172
4.49.2	Constructor & Destructor Documentation . . . . .	172
4.49.3	Member Function Documentation . . . . .	172
4.50	SwitchEnergyModel Class Reference . . . . .	173
4.50.1	Detailed Description . . . . .	174
4.50.2	Constructor & Destructor Documentation . . . . .	174

4.50.3	Member Function Documentation	175
4.50.4	Member Data Documentation	176
4.51	SwitchEnergyModelClass Class Reference	178
4.51.1	Detailed Description	178
4.51.2	Constructor & Destructor Documentation	178
4.51.3	Member Function Documentation	179
4.52	SwitchEnergyTimer Class Reference	179
4.52.1	Detailed Description	180
4.52.2	Constructor & Destructor Documentation	180
4.52.3	Member Function Documentation	180
4.52.4	Member Data Documentation	180
4.53	TaskAlloc Class Reference	181
4.53.1	Detailed Description	182
4.53.2	Constructor & Destructor Documentation	182
4.53.3	Member Function Documentation	182
4.53.4	Member Data Documentation	184
4.54	TaskInfo Class Reference	185
4.54.1	Detailed Description	186
4.54.2	Constructor & Destructor Documentation	186
4.54.3	Member Function Documentation	186
4.54.4	Member Data Documentation	188
4.55	TskComAgent Class Reference	189
4.55.1	Detailed Description	190
4.55.2	Constructor & Destructor Documentation	190
4.55.3	Member Function Documentation	190
4.55.4	Member Data Documentation	192
4.56	TskComAgentClass Class Reference	192
4.56.1	Detailed Description	193
4.56.2	Constructor & Destructor Documentation	193
4.56.3	Member Function Documentation	193

4.57	TskComSink Class Reference	194
4.57.1	Detailed Description	195
4.57.2	Constructor & Destructor Documentation	195
4.57.3	Member Function Documentation	195
4.57.4	Member Data Documentation	196
4.58	TskComSinkClass Class Reference	197
4.58.1	Detailed Description	198
4.58.2	Constructor & Destructor Documentation	198
4.58.3	Member Function Documentation	198
4.59	TskOutSink Class Reference	198
4.59.1	Detailed Description	199
4.59.2	Constructor & Destructor Documentation	199
4.59.3	Member Function Documentation	200
4.59.4	Member Data Documentation	200
4.60	TskOutSinkClass Class Reference	201
4.60.1	Detailed Description	202
4.60.2	Constructor & Destructor Documentation	202
4.60.3	Member Function Documentation	202
4.61	VM Class Reference	203
4.61.1	Detailed Description	204
4.61.2	Constructor & Destructor Documentation	204
4.61.3	Member Function Documentation	204
4.61.4	Member Data Documentation	206
4.62	VMClass Class Reference	207
4.62.1	Detailed Description	207
4.62.2	Constructor & Destructor Documentation	207
4.62.3	Member Function Documentation	208
4.63	VmMigration Class Reference	208
4.63.1	Detailed Description	209
4.63.2	Constructor & Destructor Documentation	210

4.63.3	Member Function Documentation	210
4.63.4	Member Data Documentation	213
4.64	VmMigrationClass Class Reference	213
4.64.1	Detailed Description	214
4.64.2	Constructor & Destructor Documentation	214
4.64.3	Member Function Documentation	214
4.65	VmMigrationSink Class Reference	215
4.65.1	Detailed Description	216
4.65.2	Constructor & Destructor Documentation	216
4.65.3	Member Function Documentation	216
4.65.4	Member Data Documentation	217
4.66	VmMigrationSinkClass Class Reference	217
4.66.1	Detailed Description	218
4.66.2	Constructor & Destructor Documentation	218
4.66.3	Member Function Documentation	218
<b>5</b>	<b>File Documentation</b>	<b>219</b>
5.1	bestdens.cc File Reference	219
5.2	bestdens.h File Reference	219
5.3	bestscorescheduler.cc File Reference	220
5.4	bestscorescheduler.h File Reference	220
5.5	bytecounter.cc File Reference	221
5.6	bytecounter.h File Reference	222
5.7	cbrclouduser.cc File Reference	223
5.7.1	Variable Documentation	223
5.8	cloudtask.cc File Reference	223
5.8.1	Variable Documentation	224
5.9	cloudtask.h File Reference	224
5.10	clouduser.cc File Reference	225
5.11	clouduser.h File Reference	225
5.12	corescheduler.cc File Reference	226

5.13	corescheduler.h File Reference . . . . .	226
5.14	cpu.cc File Reference . . . . .	227
5.14.1	Variable Documentation . . . . .	228
5.15	cpu.h File Reference . . . . .	228
5.16	datacenter.cc File Reference . . . . .	228
5.16.1	Variable Documentation . . . . .	229
5.17	datacenter.h File Reference . . . . .	229
5.18	dchost.cc File Reference . . . . .	230
5.18.1	Variable Documentation . . . . .	231
5.19	dchost.h File Reference . . . . .	231
5.20	dcrack.cc File Reference . . . . .	232
5.20.1	Variable Documentation . . . . .	232
5.21	dcrack.h File Reference . . . . .	232
5.22	dcresource.cc File Reference . . . . .	233
5.22.1	Variable Documentation . . . . .	234
5.23	dcresource.h File Reference . . . . .	234
5.24	dcscheduler.cc File Reference . . . . .	235
5.25	dcscheduler.h File Reference . . . . .	235
5.26	expclouduser.cc File Reference . . . . .	236
5.26.1	Variable Documentation . . . . .	237
5.27	greenscheduler.cc File Reference . . . . .	237
5.28	greenscheduler.h File Reference . . . . .	237
5.29	herosscheduler.cc File Reference . . . . .	238
5.29.1	Function Documentation . . . . .	239
5.30	herosscheduler.h File Reference . . . . .	239
5.30.1	Function Documentation . . . . .	240
5.31	linearpmodel.cc File Reference . . . . .	241
5.31.1	Variable Documentation . . . . .	241
5.32	linearpmodel.h File Reference . . . . .	242
5.33	nic.cc File Reference . . . . .	242

5.33.1 Variable Documentation . . . . .	243
5.34 nic.h File Reference . . . . .	243
5.35 paretoclouduser.cc File Reference . . . . .	244
5.35.1 Variable Documentation . . . . .	245
5.36 percomponentmodel.cc File Reference . . . . .	245
5.36.1 Variable Documentation . . . . .	246
5.37 percomponentmodel.h File Reference . . . . .	246
5.38 powermodel.cc File Reference . . . . .	247
5.39 powermodel.h File Reference . . . . .	247
5.40 probabilisticscheduler.cc File Reference . . . . .	248
5.41 probabilisticscheduler.h File Reference . . . . .	248
5.42 provideroutagent.cc File Reference . . . . .	249
5.42.1 Variable Documentation . . . . .	250
5.43 provideroutagent.h File Reference . . . . .	250
5.44 providerscore.cc File Reference . . . . .	251
5.45 providerscore.h File Reference . . . . .	251
5.46 randdens.cc File Reference . . . . .	252
5.47 randdens.h File Reference . . . . .	253
5.48 randomscheduler.cc File Reference . . . . .	254
5.49 randomscheduler.h File Reference . . . . .	254
5.50 resdemand.cc File Reference . . . . .	255
5.51 resdemand.h File Reference . . . . .	256
5.52 resource.cc File Reference . . . . .	257
5.53 resource.h File Reference . . . . .	257
5.53.1 Enumeration Type Documentation . . . . .	259
5.53.2 Function Documentation . . . . .	259
5.54 resourceconsumer.cc File Reference . . . . .	262
5.55 resourceconsumer.h File Reference . . . . .	262
5.56 resourceprovider.cc File Reference . . . . .	263
5.57 resourceprovider.h File Reference . . . . .	263

5.58	resourcespec.cc File Reference	264
5.58.1	Variable Documentation	265
5.59	resourcespec.h File Reference	265
5.60	roundrobinscheduler.cc File Reference	266
5.61	roundrobinscheduler.h File Reference	266
5.62	scorescheduler.cc File Reference	267
5.63	scorescheduler.h File Reference	268
5.64	switchenergymodel.cc File Reference	269
5.64.1	Variable Documentation	270
5.65	switchenergymodel.h File Reference	270
5.66	taskalloc.cc File Reference	271
5.67	taskalloc.h File Reference	271
5.68	taskinfo.cc File Reference	272
5.69	taskinfo.h File Reference	272
5.70	tskagent.cc File Reference	273
5.70.1	Variable Documentation	274
5.71	tskagent.h File Reference	275
5.71.1	Macro Definition Documentation	275
5.72	tskcomsink.cc File Reference	276
5.72.1	Variable Documentation	276
5.73	tskcomsink.h File Reference	277
5.74	tskoutsink.cc File Reference	277
5.74.1	Variable Documentation	278
5.75	tskoutsink.h File Reference	279
5.76	vm.cc File Reference	280
5.76.1	Variable Documentation	280
5.77	vm.h File Reference	280
5.77.1	Enumeration Type Documentation	281
5.78	vmmigration.cc File Reference	282
5.78.1	Variable Documentation	282
5.79	vmmigration.h File Reference	282
5.80	vmmigrationsink.cc File Reference	283
5.80.1	Variable Documentation	284
5.81	vmmigrationsink.h File Reference	284

<a href="#">Index</a>	287
-----------------------	-----

## 1 Hierarchical Index

### 1.1 Class Hierarchy

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

Agent

**TskComAgent** 189

**TskComSink** 194

**ByteCounter** 15

**TskComSink** 194

**TskOutSink** 198

**VmMigrationSink** 215

**Capacity** 16

**CloudUser** 31

**CBRCloudUser** 20

**ExpCloudUser** 84

**ParetoCloudUser** 105

**CoreScheduler** 37

**DcScheduler** 82

**GreenScheduler** 89

**RoundRobinsScheduler** 169

**ScoreScheduler** 171

**BestScoreScheduler** 13

**BestDENS** 9

**HerosScheduler** 91

**ProbabilisticScheduler** 122

**RandDENS** 131

**RandomScheduler** 134

Handler

**ResourceProvider** 145

**DcHost** 66

VM	203
PoaBuf	114
PoaBufList	116
ProviderScore	128
Resource	139
DcResource	77
CPU	47
NIC	102
ResDemand	137
ResourceSpec	164
ResourceConsumer	143
CloudTask	24
VM	203
VmMigration	208
TaskAlloc	181
TaskInfo	185
TclClass	
CBRCloudUserClass	23
CpuClass	52
DataCenterClass	64
DcHostClass	71
DcRackClass	76
DcResourceClass	81
ExpCloudUserClass	88
LinearPModelClass	101
NicClass	104
PerComponentModelClass	113
POOTrafficClass	118
ProvOutAgentClass	130
ResourceSpecClass	168
SwitchEnergyModelClass	178
TskComAgentClass	192

<b>TskComSinkClass</b>	<b>197</b>
<b>TskOutSinkClass</b>	<b>201</b>
<b>VMClass</b>	<b>207</b>
<b>VmMigrationClass</b>	<b>213</b>
<b>VmMigrationSinkClass</b>	<b>217</b>
TclObject	
<b>DataCenter</b>	<b>53</b>
<b>DcHost</b>	<b>66</b>
<b>DcRack</b>	<b>72</b>
<b>DcResource</b>	<b>77</b>
<b>PowerModel</b>	<b>119</b>
<b>LinearPModel</b>	<b>96</b>
<b>PerComponentModel</b>	<b>110</b>
<b>ResourceSpec</b>	<b>164</b>
<b>SwitchEnergyModel</b>	<b>173</b>
<b>VM</b>	<b>203</b>
<b>VmMigration</b>	<b>208</b>
TcpAgent	
<b>ProviderOutAgent</b>	<b>125</b>
TcpSink	
<b>TskOutSink</b>	<b>198</b>
<b>VmMigrationSink</b>	<b>215</b>
TimerHandler	
<b>DcRack</b>	<b>72</b>
<b>SwitchEnergyTimer</b>	<b>179</b>
TrafficGenerator	
<b>CBRCloudUser</b>	<b>20</b>
<b>ExpCloudUser</b>	<b>84</b>
<b>ParetoCloudUser</b>	<b>105</b>

## 2 Class Index

### 2.1 Class List

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

<b>BestDENS</b>	<b>9</b>
<b>BestScoreScheduler</b>	<b>13</b>
<b>ByteCounter</b>	<b>15</b>
<b>Capacity</b>	<b>16</b>
<b>CBRCloudUser</b>	<b>20</b>
<b>CBRCloudUserClass</b>	<b>23</b>
<b>CloudTask</b>	<b>24</b>
<b>CloudUser</b>	<b>31</b>
<b>CoreScheduler</b>	<b>37</b>
<b>CPU</b>	<b>47</b>
<b>CpuClass</b>	<b>52</b>
<b>DataCenter</b>	<b>53</b>
<b>DataCenterClass</b>	<b>64</b>
<b>DcHost</b>	<b>66</b>
<b>DcHostClass</b>	<b>71</b>
<b>DcRack</b>	<b>72</b>
<b>DcRackClass</b>	<b>76</b>
<b>DcResource</b>	<b>77</b>
<b>DcResourceClass</b>	<b>81</b>
<b>DcScheduler</b>	<b>82</b>
<b>ExpCloudUser</b>	<b>84</b>
<b>ExpCloudUserClass</b>	<b>88</b>
<b>GreenScheduler</b>	<b>89</b>
<b>HerosScheduler</b>	<b>91</b>
<b>LinearPModel</b>	<b>96</b>
<b>LinearPModelClass</b>	<b>101</b>
<b>NIC</b>	<b>102</b>
<b>NicClass</b>	<b>104</b>
<b>ParetoCloudUser</b>	<b>105</b>
<b>PerComponentModel</b>	<b>110</b>
<b>PerComponentModelClass</b>	<b>113</b>
<b>PoaBuf</b>	<b>114</b>

PoaBufList	116
POOTrafficClass	118
PowerModel	119
ProbabilisticScheduler	122
ProviderOutAgent	125
ProviderScore	128
ProvOutAgentClass	130
RandDENS	131
RandomScheduler	134
ResDemand	137
Resource	139
ResourceConsumer	143
ResourceProvider	145
ResourceSpec	164
ResourceSpecClass	168
RoundRobinsScheduler	169
ScoreScheduler	171
SwitchEnergyModel	173
SwitchEnergyModelClass	178
SwitchEnergyTimer	179
TaskAlloc	181
TaskInfo	185
TskComAgent	189
TskComAgentClass	192
TskComSink	194
TskComSinkClass	197
TskOutSink	198
TskOutSinkClass	201
VM	203
VMClass	207
VmMigration	208
VmMigrationClass	213

<a href="#">VmMigrationSink</a>	215
<a href="#">VmMigrationSinkClass</a>	217

## 3 File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

<a href="#">bestdens.cc</a>	219
<a href="#">bestdens.h</a>	219
<a href="#">bestscorescheduler.cc</a>	220
<a href="#">bestscorescheduler.h</a>	220
<a href="#">bytecounter.cc</a>	221
<a href="#">bytecounter.h</a>	222
<a href="#">cbrclouduser.cc</a>	223
<a href="#">cloudtask.cc</a>	223
<a href="#">cloudtask.h</a>	224
<a href="#">clouduser.cc</a>	225
<a href="#">clouduser.h</a>	225
<a href="#">corescheduler.cc</a>	226
<a href="#">corescheduler.h</a>	226
<a href="#">cpu.cc</a>	227
<a href="#">cpu.h</a>	228
<a href="#">datacenter.cc</a>	228
<a href="#">datacenter.h</a>	229
<a href="#">dchost.cc</a>	230
<a href="#">dchost.h</a>	231
<a href="#">dcrack.cc</a>	232
<a href="#">dcrack.h</a>	232
<a href="#">dcresource.cc</a>	233
<a href="#">dcresource.h</a>	234
<a href="#">dcscheduler.cc</a>	235
<a href="#">dcscheduler.h</a>	235

<a href="#">expclouduser.cc</a>	236
<a href="#">greenscheduler.cc</a>	237
<a href="#">greenscheduler.h</a>	237
<a href="#">herosscheduler.cc</a>	238
<a href="#">herosscheduler.h</a>	239
<a href="#">linearpmodel.cc</a>	241
<a href="#">linearpmodel.h</a>	242
<a href="#">nic.cc</a>	242
<a href="#">nic.h</a>	243
<a href="#">paretoclouduser.cc</a>	244
<a href="#">percomponentmodel.cc</a>	245
<a href="#">percomponentmodel.h</a>	246
<a href="#">powermodel.cc</a>	247
<a href="#">powermodel.h</a>	247
<a href="#">probabilisticscheduler.cc</a>	248
<a href="#">probabilisticscheduler.h</a>	248
<a href="#">provideroutagent.cc</a>	249
<a href="#">provideroutagent.h</a>	250
<a href="#">providerscore.cc</a>	251
<a href="#">providerscore.h</a>	251
<a href="#">randdens.cc</a>	252
<a href="#">randdens.h</a>	253
<a href="#">randomscheduler.cc</a>	254
<a href="#">randomscheduler.h</a>	254
<a href="#">resdemand.cc</a>	255
<a href="#">resdemand.h</a>	256
<a href="#">resource.cc</a>	257
<a href="#">resource.h</a>	257
<a href="#">resourceconsumer.cc</a>	262
<a href="#">resourceconsumer.h</a>	262
<a href="#">resourceprovider.cc</a>	263
<a href="#">resourceprovider.h</a>	263

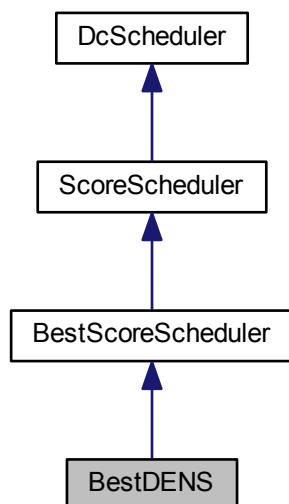
<a href="#">resourcespec.cc</a>	264
<a href="#">resourcespec.h</a>	265
<a href="#">roundrobinscheduler.cc</a>	266
<a href="#">roundrobinscheduler.h</a>	266
<a href="#">scorescheduler.cc</a>	267
<a href="#">scorescheduler.h</a>	268
<a href="#">switchenergymodel.cc</a>	269
<a href="#">switchenergymodel.h</a>	270
<a href="#">taskalloc.cc</a>	271
<a href="#">taskalloc.h</a>	271
<a href="#">taskinfo.cc</a>	272
<a href="#">taskinfo.h</a>	272
<a href="#">tskagent.cc</a>	273
<a href="#">tskagent.h</a>	275
<a href="#">tskcomsink.cc</a>	276
<a href="#">tskcomsink.h</a>	277
<a href="#">tskoutsink.cc</a>	277
<a href="#">tskoutsink.h</a>	279
<a href="#">vm.cc</a>	280
<a href="#">vm.h</a>	280
<a href="#">vmmigration.cc</a>	282
<a href="#">vmmigration.h</a>	282
<a href="#">vmmigrationsink.cc</a>	283
<a href="#">vmmigrationsink.h</a>	284

## 4 Class Documentation

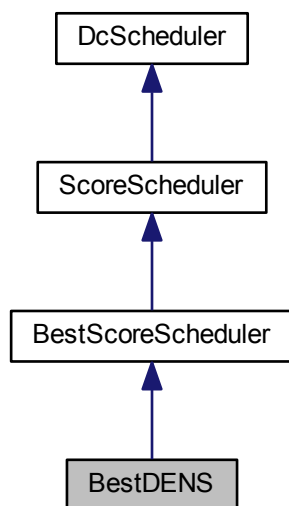
### 4.1 BestDENS Class Reference

```
#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 &amp; Destructor Documentation

## 4.1.2.1 BestDENS::BestDENS ( )

Definition at line 11 of file bestdens.cc.

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

## 4.1.2.2 BestDENS::~BestDENS ( ) [virtual]

Definition at line 15 of file bestdens.cc.

```
15
16
17 }
```

## 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.

```
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,
29             epsilon);
29             }
30             result=result/(LastResType+1); // normalize according to the number of
31             dimensions
32             std::cerr << "Res_prov " << rp->id_ << "\n";
33             DcHost* host = rp->getRootHost();
34             DcRack* rack = host->rack_;
35             std::cerr << "Rack " << rack->rack_id_ << "\n";
36             double ll = rack->link_load;
37             std::cerr << "ll " << ll << "\n";
38             result *= pow(linkLoadFactor(rp->getRootHost()->
39             rack\_->link\_load),2);
40             result += linkLoadFactor(ll);
41             double load = rp->getResTypeUtil(Computing);
42             result = densLoadFactor(load,0.1);
43             std::cerr << "Host " << rp->id_ /*<< "CPU load: " << load*/ << "\tDENS score: " << result
44             << "\n";
45             return result;
46 }
```

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

Definition at line 48 of file bestdens.cc.

```

48                                     {
49         return 1/(1+exp(-10*(load-0.5))) - 1/(1+exp((-10/epsilon)*(load-(1- (
50     epsilon/2)))));

```

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

Definition at line 52 of file bestdens.cc.

```

52                                     {
53         return exp(-(load*load));
54     }

```

#### 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.

```

19                                     {
20         return BestScoreScheduler::scheduleTask(task,providers);
21     }

```

### 4.1.4 Member Data Documentation

#### 4.1.4.1 double BestDENS::epsilon [private]

Definition at line 27 of file bestdens.h.

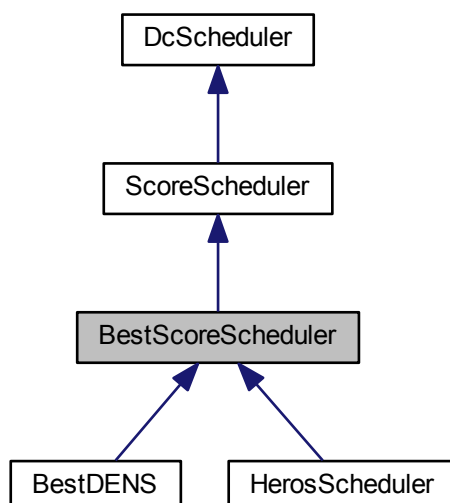
The documentation for this class was generated from the following files:

- [bestdens.h](#)
- [bestdens.cc](#)

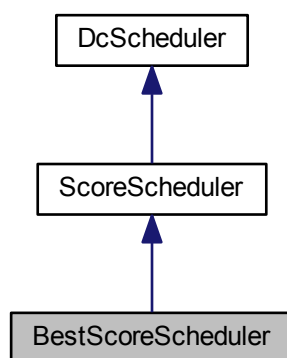
## 4.2 BestScoreScheduler Class Reference

```
#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](#).

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

#### 4.2.2.2 [BestScoreScheduler::~~BestScoreScheduler](#) ( ) [virtual]

Definition at line 15 of file [bestscorescheduler.cc](#).

```
15                                     {
16
17 }
```

### 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](#).

```
19                                     {
20         vector<ProviderScore> scored_providers_;
21         vector<ResourceProvider*>::iterator iter;
22         for (iter = providers.begin(); iter!=providers.end(); iter++)
23         {
24                 if ((*iter)->testSchedulingPossibility(task)){
25                         scored_providers_.push_back(
26                 ProviderScore((*iter),calculateScore((*iter))));
27         } else {
28                 std::cerr << "Provider full!\n";
29         }
30         if(scored_providers_.empty()){
31                 return NULL;
32         } else {
33                 ProviderScore best = (*max_element(scored_providers_.begin(),
34                 scored_providers_.end()));
35                 scored_providers_.clear();
36                 return best.provider_->getTskComAgent();
37 }
```

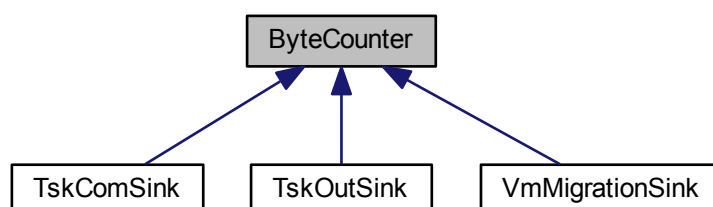
The documentation for this class was generated from the following files:

- [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`.

```
10         {  
11             resetBytesSince ();  
12         }  
13 }
```

#### 4.3.2.2 ByteCounter::~ByteCounter ( ) [virtual]

Definition at line 15 of file bytecounter.cc.

```
15  
16  
17 }
```

### 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.

```
26  
27         return last_bytes_since_;  
28 }
```

#### 4.3.3.2 int ByteCounter::resetBytesSince ( )

Returns the value of bytes since last check.

Definition at line 19 of file bytecounter.cc.

```
19  
20         {  
21             int result = this->bytes_since_;  
22             this->bytes_since_ = 0;  
23             this->last_bytes_since_ = Scheduler::instance().clock();  
24             return result;  
25 }
```

### 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 &amp; Destructor Documentation

## 4.4.2.1 Capacity::Capacity ( )

Definition at line 11 of file resource.cc.

```

11                                     : value(0.0) {
12
13                                     }
```

## 4.4.2.2 Capacity::Capacity ( int i )

Definition at line 15 of file resource.cc.

```

15                                     : value(i) {
16
17                                     }
```

## 4.4.2.3 Capacity::Capacity ( double d )

Definition at line 19 of file resource.cc.

```

19                                     : value(d) {
20
21                                     }
```

### 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.

```
54                                     {
55         this->value -= rhs;
56         return *this;
57 }
```

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

Definition at line 23 of file resource.cc.

```
23                                     {
24         this->value = val;
25         return *this; // Return ref for multiple assignment
26 }
```

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

Definition at line 28 of file resource.cc.

```
28                                     {
29         if (this != &c) { // make sure it is not the same object
30             this->value = c.value;
31             this->virtual_capacities.clear();
32             this->virtual_capacities = c.
virtual_capacities;
33         }
34         return *this; // Return ref for multiple assignment
35 }
```

### 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

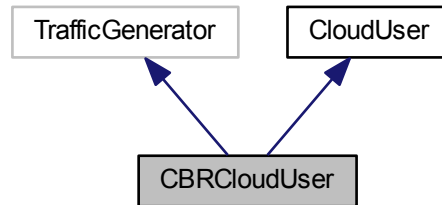
Definition at line 34 of file resource.h.

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

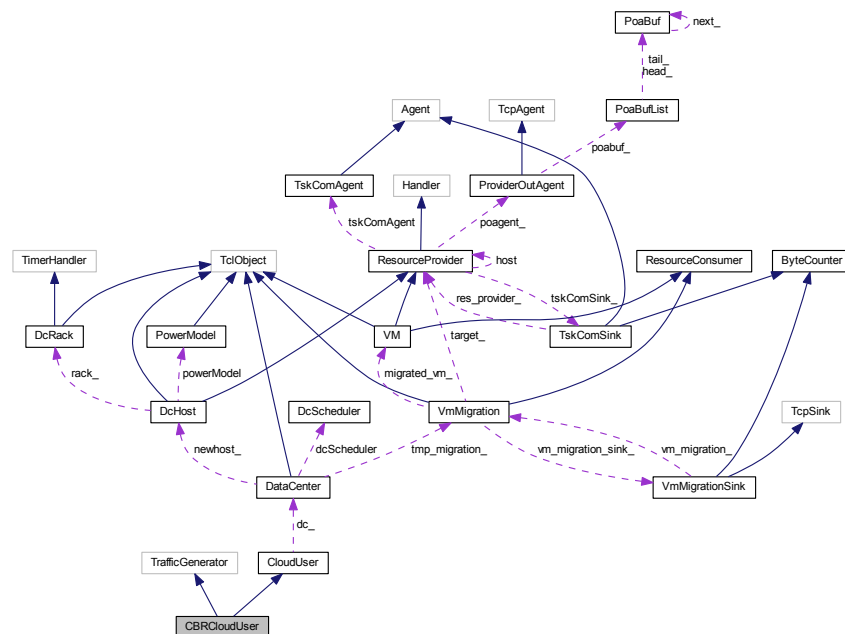
- [resource.h](#)
- [resource.cc](#)

## 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 &amp; Destructor Documentation

## 4.5.2.1 CBRCLOUDUSER::CBRCLOUDUSER ( )

Definition at line 62 of file `cbrclouduser.cc`.

```

62             : seqno_(0)
63 {
64     bind_time("random_tskmips_", &random_tskmips_.avgp());
65     bind_bw("rate_", &rate_);
66     bind("random_", &random_);
67     bind("packetSize_", &size_);
68     bind("maxpkts_", &maxpkts_);
69
70     // Bind CloudUser variables
71     bind("id_", &id_);
72     bind("tskmips_", &tskmips_);
73     bind("tsksize_", &tsksize_);
74     bind("tskmaxduration_", &tskmaxduration_);
75     bind("mean_response_time_", &mean_response_time_);
76     bind("sd_response_time_", &sd_response_time_);
77     bind("unfinished_tasks_", &unfinished_tasks_);
78 }
```

## 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`.

```

47                                     {
48
49         if(argc==3) {
50             if (strcmp(argv[1], "join-datacenter") == 0) {
51                 DataCenter *dc = dynamic_cast<
DataCenter*> (TclObject::lookup(argv[2]));
52                 if (dc) {
53                     dc_ = dc;
54                     return (TCL_OK);
55                 }
56                 return (TCL_ERROR);
57             }
58         }
59         return Application::command(argc, argv);
60 }
```

#### 4.5.3.3 void CBRCLOUDUser::init ( ) [protected]

Definition at line 80 of file cbrclouduser.cc.

```

81 {
82     // compute inter-packet interval
83     interval_ = (double)(size_ << 3)/(double)rate_;
84     if (agent_)
85         if (agent_>get_pkttype() != PT_TCP &&
86             agent_>get_pkttype() != PT_TFRC)
87             agent_>set_pkttype(PT_CBR);
88 }
```

#### 4.5.3.4 double CBRCLOUDUser::interval ( ) [inline]

Definition at line 23 of file cbrclouduser.cc.

```

23 { return (interval_); }
```

#### 4.5.3.5 double CBRCLOUDUser::next\_interval ( int & size ) [virtual]

Definition at line 97 of file cbrclouduser.cc.

```

98 {
99     // Recompute interval in case rate_ or size_ has changes
100     interval_ = (double)(size_ << 3)/(double)rate_;
101     double t = interval_;
102     if (random_)
103         t += interval_ * Random::uniform(-0.5, 0.5);
104     size = size_;
105     if (++seqno_ < maxpkts_)
106         return(t);
107     else
108         return(-1);
109 }
```

#### 4.5.3.6 void CBRCLOUDUser::start ( ) [protected],[virtual]

Definition at line 90 of file cbrclouduser.cc.

```

91 {
92     init();
93     running_ = 1;
94     timeout();
95 }
```

#### 4.5.3.7 void CBRCLOUDUser::timeout ( ) [virtual]

Definition at line 111 of file cbrclouduser.cc.

```

112 {
113     if (! running_)
114         return;
115
116     /* send a packet */
117     dc_>receivedTsk(size_, createTask());
118     /* figure out when to send the next one */
119     nextPkttime_ = next_interval(size_);
120     /* schedule it */
121     if (nextPkttime_ > 0)
122         timer_.resched(nextPkttime_);
123     else
124         running_ = 0;
125 }
```

#### 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]

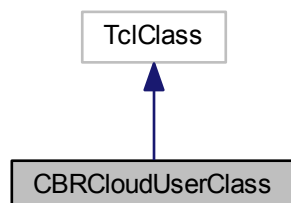
Definition at line 34 of file cbrclouduser.cc.

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

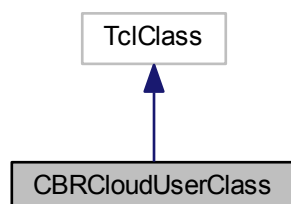
- [cbrclouduser.cc](#)

## 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`.

```
41 : 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`.

```
42                                     {
43         return (new CBRCloudUser());
44     }
```

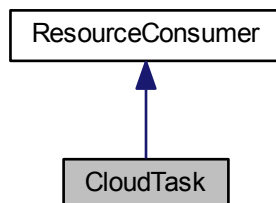
The documentation for this class was generated from the following file:

- [cbrclouduser.cc](#)

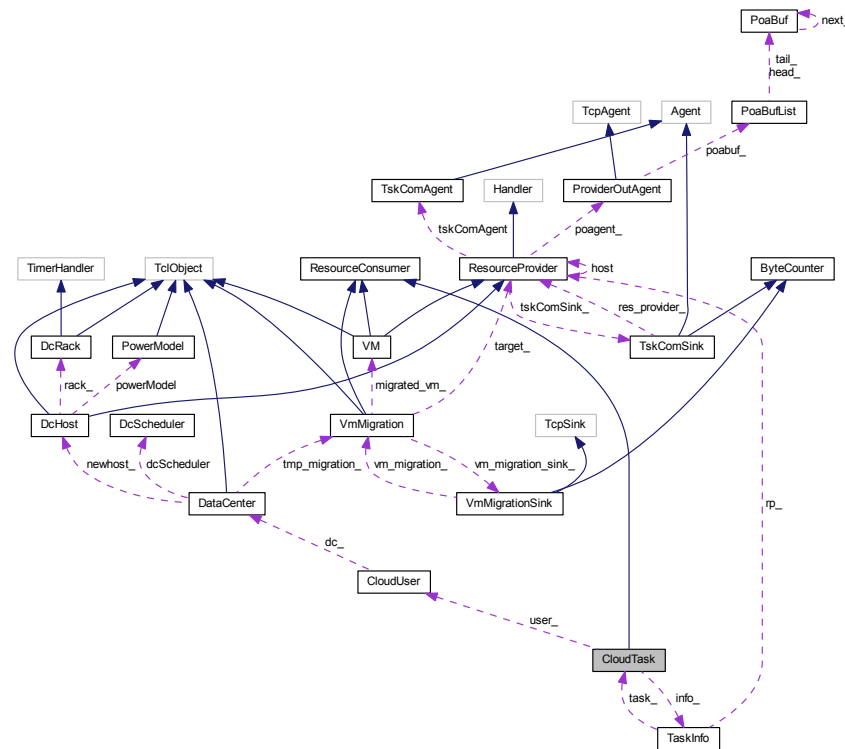
## 4.7 CloudTask Class Reference

```
#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 clouddtask.cc.

```

40 {
41     std::vector<ResDemand*>::iterator iter;
42     for (iter = res_demands.begin(); iter!=res_demands.end(); iter++)
43     {
44         delete (*iter);
45     }
46 }
```

## 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 clouddtask.cc.

```

59 {
60     // std::cerr << "Task\t" << id_ << "\tfailed on provider:\t" << provider->id_ << "\n";
61     failed_ = true;
62     if (this->started_==true) {
63         provider->releaseAllocation(this);
64     }
65     provider->tskFailed_++;
66     releaseAllTaskAllocs();
67     task_allocations_.clear();
68 }
```

## 4.7.3.2 double CloudTask::getDeadline ( ) [inline]

Definition at line 23 of file clouddtask.h.

```

23 {return deadline_;;}
```

## 4.7.3.3 int CloudTask::getID ( ) [inline]

Definition at line 21 of file clouddtask.h.

```

21 {return id_;;}
```

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

Definition at line 52 of file clouddtask.cc.

```

52 {
53     if (res_demands.at(rd)->getType() != Computing) {
54         std::cerr << "MIPS requested for non-Computing resource";
55         abort();
56     }
57     return res_demands.at(rd)->capacity.at(cap);}
```

#### 4.7.3.5 int CloudTask::getOutput ( ) [inline]

Definition at line 24 of file clouddtask.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 clouddtask.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 clouddtask.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.
103                         end(); cap++) {
104                         std::cerr << (*cap) << " ";
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 clouddtask.cc.

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

#### 4.7.3.10 void CloudTask::removeTaskAlloc ( TaskAlloc \* ta )

Removes task allocation.

Definition at line 78 of file clouddtask.cc.

```
78                                     {
79         task_allocations_.erase(remove(
task_allocations_.begin(), task_allocations_.end(), ta),
80                                     task_allocations_.end()); /*erase-remove
81         idiom*/
81 }
```

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

Definition at line 28 of file clouddtask.h.

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

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

Definition at line 27 of file clouddtask.h.

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

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

Definition at line 30 of file clouddtask.h.

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

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

Definition at line 48 of file clouddtask.cc.

```
48                                     {
49         res_demands.at(rd)->capacity.at(cap) = mips;
50 }
```

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

Definition at line 29 of file clouddtask.h.

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

#### 4.7.4 Member Data Documentation

##### 4.7.4.1 `double CloudTask::deadline_`

task deadline

Definition at line 54 of file clouddtask.h.

##### 4.7.4.2 `bool CloudTask::failed_`

Definition at line 55 of file clouddtask.h.

##### 4.7.4.3 `int CloudTask::id_`

Definition at line 51 of file clouddtask.h.

##### 4.7.4.4 `TaskInfo* CloudTask::info_`

Definition at line 61 of file clouddtask.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 clouddtask.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 clouddtask.h.

##### 4.7.4.7 `bool CloudTask::scheduled_`

true if task has been scheduled

Definition at line 52 of file clouddtask.h.

##### 4.7.4.8 `bool CloudTask::started_`

true if task has started its execution

Definition at line 53 of file clouddtask.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 clouddtask.h.

## 4.7.4.10 CloudUser\* CloudTask::user\_ [protected]

The cloud user that created the task.

Definition at line 65 of file clouddtask.h.

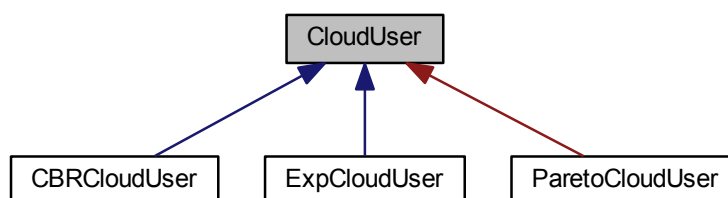
The documentation for this class was generated from the following files:

- [clouddtask.h](#)
- [clouddtask.cc](#)

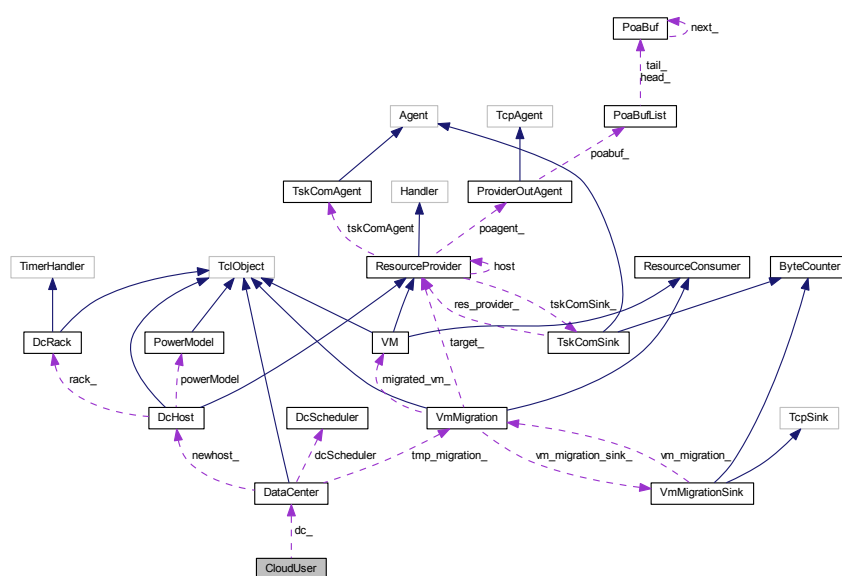
## 4.8 CloudUser Class Reference

```
#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.

```
16 {
17 }
```

## 4.8.3 Member Function Documentation

## 4.8.3.1 void CloudUser::calculateStatistics( ) [protected]

Definition at line 141 of file clouduser.cc.

```
141 {
142     std::vector<TaskInfo*>::iterator i;
143     double sum = 0;
144     int counter = 0;
145     unfinished_tasks_ = 0;
146
147     //mean calculation
148     if(!tasks_info_.empty()){
149         for(i = tasks_info_.begin(); i < tasks_info_.end(); i++){
150             if((*i)->getDcExitTime() != -1){
151                 sum+= (*i)->getDcExitTime() - (*i)->getReleaseTime();
152                 counter++;
153             } else {
154                 unfinished_tasks_++;
155             }
156         }
157
158         mean_response_time_ = sum / counter;
159
160         // sd calculation
161         sum = 0;
162         double tmp;
163         for(i = tasks_info_.begin(); i < tasks_info_.end(); i++){
164             if((*i)->getDcExitTime() != -1){
165                 tmp = pow( ((*i)->getDcExitTime() - (*i)->getReleaseTime()
166 ) - mean_response_time_, 2.0f);
167                 sum += tmp;
168             }
169         }
170         sd_response_time_ = sqrt(sum/counter);
171     } else {
172         std::cerr << "WARNING: No tasks generated by the cloud user: " <<
173         id_ << " (normally it should not happen).\n";
174     }
```

## 4.8.3.2 CloudTask \* CloudUser::createTask( )

Definition at line 19 of file clouduser.cc.

```
20 {
21     std::vector<Resource*> task_demand;
22
23     std::vector<Capacity> task_proc_cap;
24     double mips;
25     int processes_number = 1;
26     for(int i = 0 ; i < processes_number; i++){
27         if(!randomized_){
28             mips = tskmips_/processes_number;
29         } else {
30             do{
31                 mips = random_tskmips_.value()/
32                 processes_number;
33             }while(mips > (tskmips_/processes_number)*
34             tskmaxduration_*0.98);
35         task_proc_cap.push_back(mips);
36     }
```

```

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(toutputsizes_);
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" <<
126                             id_ << "\t";
127                         std::cout << fixed << setprecision(2) << "Task unfinished, id: " << (*i)->
128                             getTaskId() <<
129                             " Rel: "<< (*i)->getReleaseTime() <<
130                             " Ser: "<< (*i)->getServerFinishTime() <<
131                             " Ext: "<< (*i)->getDcExitTime() <<
132                             " Due: " << (*i)->getDueTime() <<
133                             "\n";
134                     }
135                     if(ok){
136                         std::cout << "Cloud User:\t" << id_ << "\t: all tasks finished sucesfully.
137                             \n";
138                     } else {
139                         std::cout << "Cloud User:\t" << id_ << "\t:\t"<<
140                             unfinished_tasks_ << "\ttasks did NOT exit datacenter.\n";
141                     }
142                 }
143             }
144         }

```

#### 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() <<

```

```

108                                     " Rel: "<< (*i)->getReleaseTime() <<
109                                     " Ser: "<< (*i)->getServerFinishTime() <<
110                                     " Ext: "<< (*i)->getDcExitTime() <<
111                                     " Due: " << (*i)->getDueTime() <<
112                                     "\n";
113                                     }
114 }

```

#### 4.8.3.5 int CloudUser::process\_command ( int argc, const char \*const \* argv )

Definition at line 73 of file clouduser.cc.

```

73                                     {
74                                     if (argc==2) {
75                                         if (strcmp(argv[1], "print-tasks-status") == 0) {
76                                             printTasksStatus();
77                                             return (TCL_OK);
78                                         } else if (strcmp(argv[1], "post-simulation-test-tasks") == 0) {
79                                             postSimulationTestTasks();
80                                             return (TCL_OK);
81                                         } else if (strcmp(argv[1], "calculate-statistics") == 0) {
82                                             calculateStatistics();
83                                             return (TCL_OK);
84                                         }
85                                     }
86                                     } else if (argc==3) {
87                                         if (strcmp(argv[1], "join-datacenter") == 0) {
88                                             DataCenter *dc = dynamic_cast<
89 DataCenter*> (TclObject::lookup(argv[2]));
90                                             if (dc) {
91                                                 dc_ = dc;
92                                                 return (TCL_OK);
93                                             }
94                                             return (TCL_ERROR);
95                                         } else if (strcmp(argv[1], "set-randomized") == 0) {
96                                             setRandomized(atoi(argv[2]));
97                                             return (TCL_OK);
98                                         }
99                                     }
100                                     return -1;
101 }

```

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

Definition at line 66 of file clouduser.cc.

```

66                                     {
67                                     randomized_ =i;
68                                     if (i!=0) {
69                                         random_tskmips_.setavg(tskmips_);
70                                     }
71 }

```

### 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.

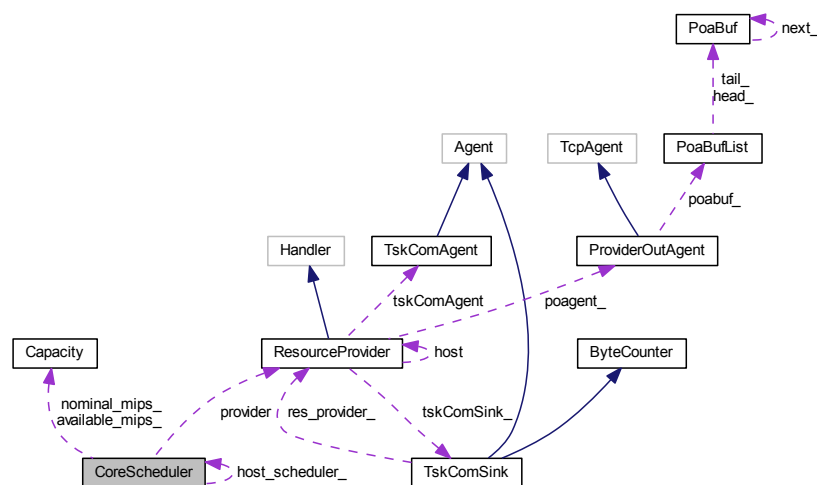
The documentation for this class was generated from the following files:

- [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 &amp; Destructor Documentation

## 4.9.2.1 CoreScheduler::CoreScheduler ( Capacity \* nominal\_mips\_, Capacity \* available\_mips\_ )

Definition at line 12 of file corescheduler.cc.

```

12                                     :
    current_mips_(0.0), host_scheduler_(NULL),
    taskAllocFailed_(0) {
13         this->nominal_mips_=nominal_mips_;
14         this->available_mips_=available_mips_;
15         tasks_alloc_list_.clear();
16         tasks_alloc_assigned_.clear();
17         hosted_vcores_schedulers.clear();
18     }
19 }
```

## 4.9.2.2 CoreScheduler::~CoreScheduler ( ) [virtual]

Definition at line 23 of file corescheduler.cc.

```

23         {
24             std::vector<TaskAlloc*>::iterator task;
25             for(task = tasks_alloc_list_.begin(); task!=
tasks_alloc_list_.end(); task++){
26                 delete (*task);
27             }
28             tasks_alloc_list_.~vector();
29             for(task = tasks_alloc_assigned_.begin(); task!=
tasks_alloc_assigned_.end(); task++){
30                 delete (*task);
31             }
32             tasks_alloc_assigned_.~vector();
33 }
```

## 4.9.3 Member Function Documentation

## 4.9.3.1 void CoreScheduler::addVcoreScheduler ( CoreScheduler \* cs )

Definition at line 308 of file corescheduler.cc.

```

308                                     {
309         hosted_vcores_schedulers.push_back(cs);
310         cs->host_scheduler_ = this;
311 }
```

## 4.9.3.2 void CoreScheduler::assignTask ( TaskAlloc \* tskobj )

Definition at line 71 of file corescheduler.cc.

```

71                                     {
72         tasks_alloc_assigned_.push_back(tskobj);
73         tskobj->setCoreScheduler(this);
74 }
```

#### 4.9.3.3 void CoreScheduler::executeTask ( TaskAlloc \* *tskobj* )

Definition at line 76 of file corescheduler.cc.

```

76                                     {
77     tasks_alloc_list_.push_back(tskobj);           // add to the active tasks
78     links    tskobj->setCoreScheduler (this);
79 }
```

#### 4.9.3.4 int CoreScheduler::getAllTasksNumber ( )

Definition at line 67 of file corescheduler.cc.

```

67                                     {
68     return tasks_alloc_list_.size() +
69     tasks_alloc_assigned_.size();
69 }
```

#### 4.9.3.5 double CoreScheduler::getAvailableMIPS ( )

Definition at line 59 of file corescheduler.cc.

```

59                                     {
60     return *available_mips_;
61 }
```

#### 4.9.3.6 double CoreScheduler::getCurrentMIPS ( )

Definition at line 43 of file corescheduler.cc.

```

43                                     {
44     return current_mips_;
45 }
```

#### 4.9.3.7 double CoreScheduler::getCurrentMIPSRecursive ( )

Definition at line 46 of file corescheduler.cc.

```

46                                     {
47     double result = getCurrentMIPS();
48     std::vector<CoreScheduler*>::iterator iter;
49     int i = 1;
50     for(iter = hosted_vcores_schedulers.begin(); iter !=
51     hosted_vcores_schedulers.end(); iter++){
52         result += (*iter)->getCurrentMIPSRecursive();
53         i++;
54     }
55     return result;
55 }
```

#### 4.9.3.8 CoreScheduler \* CoreScheduler::getHostScheduler ( )

Definition at line 319 of file corescheduler.cc.

```

319                                     {
320     return host_scheduler_;
321 }
```

## 4.9.3.9 double CoreScheduler::getMostUrgentTaskRate ( )

Definition at line 201 of file corescheduler.cc.

```

202 {
203     std::vector<TaskAlloc*>::iterator iter;
204
205     /* update what is already computing */
206
207     for (iter = tasks_alloc_list_.begin(); iter !=
tasks_alloc_list_.end(); iter++)
208     {
209         (*iter)->updateMIPS();           // update what is already computing
210     }
211     /* remove completed */
212     removeCompleted();
213
214     /* Compute highest MIPS/deadline ratio */
215     double maxrate = 0.0;
216
217     /* get most urgent task rate from the execution list */
218     for (iter = tasks_alloc_list_.begin(); iter !=
tasks_alloc_list_.end(); )
219     {
220         if((double)((*iter)->getDeadline() - Scheduler::instance().clock())>0 &&
(*iter)->cloudTask->failed_==false){
221             double rate = (double)(*iter)->getMIPS()/ (double)((*iter)->
getDeadline() - Scheduler::instance().clock());
222             if(rate>maxrate){
223                 maxrate = rate;
224             }
225             iter++;
226         }
227         else {
228             removeFailedTaskAlloc(iter,true);
229         }
230     }
231
232     /* get most urgent task rate from the in-fly list */
233     for (iter = tasks_alloc_assigned_.begin(); iter !=
tasks_alloc_assigned_.end(); )
234     {
235         if((double)((*iter)->getDeadline() - Scheduler::instance().clock())>0 &&
(*iter)->cloudTask->failed_==false){
236             double rate = (double)(*iter)->getMIPS()/ (double)((*iter)->
getDeadline() - Scheduler::instance().clock());
237             if(rate>maxrate){
238                 maxrate = rate;
239             }
240             iter++;
241         } else {
242             removeFailedTaskAlloc(iter,false);
243         }
244     }
245     if (maxrate > getAvailableMIPS()) {return
getAvailableMIPS();}
246     else{return maxrate;}
247 }

```

## 4.9.3.10 double CoreScheduler::getNominalMIPS ( )

Definition at line 56 of file corescheduler.cc.

```

56     {
57         return *nominal_mips_;
58     }

```

## 4.9.3.11 ResourceProvider \* CoreScheduler::getProvider ( )

Definition at line 63 of file corescheduler.cc.

```

63     {
64         return provider;
65     }

```

#### 4.9.3.12 bool CoreScheduler::removeAllocationsFromAssignedList ( 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        /*Task run over its deadline, remove it and mark as failed. */
189        else if (Scheduler::instance().clock() >= (*iter)->cloudTask->getDeadline())
190        {
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::removeFromAssignedList ( 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.

```

39         {
40             this->eDVFS_enabled_ = eDVFS_enabled_;
41         }

```

#### 4.9.3.21 void CoreScheduler::setProvider ( ResourceProvider \* *provider* )

Definition at line 35 of file corescheduler.cc.

```

35         {
36             this->provider = provider;
37         }

```

#### 4.9.3.22 void CoreScheduler::startTaskExecution ( CloudTask \* *tskobj* )

Definition at line 110 of file corescheduler.cc.

```

110         {
111             vector<TaskAlloc*>::iterator iter;
112             for (iter = tasks_alloc_assigned_.begin(); iter !=
tasks_alloc_assigned_.end(); )
113             {
114                 /* task received remove from in-fly list */
115                 if ((*iter)->cloudTask->id_ == tskobj->id_) {
116                     executeTask((*iter));
117                     (*iter)->setExecTime(Scheduler::instance().clock());
118                     iter = tasks_alloc_assigned_.erase(
iter);
119                 } else {
120                     iter++;
121                 }
122             }
123             if(!tskobj->started_) {
124                 tskobj->started_ = true;
125             }
126             this->updateTskList();
127         }
128     }

```

#### 4.9.3.23 void CoreScheduler::updateTskComputingRates ( double *c\_mips* )

Definition at line 251 of file corescheduler.cc.

```

252     {
253         vector<TaskAlloc*>::iterator iter;
254
255         for (iter = tasks_alloc_list_.begin(); iter !=
tasks_alloc_list_.end(); iter++)
256         {
257             /* each task with then update mips left */
258             (*iter)->setComputingRate((double)c_mips/
tasks_alloc_list_.size());
259         }
260     }
261 }

```

**4.9.3.24 void CoreScheduler::updateTskList ( double *c\_mips* )**

Definition at line 263 of file corescheduler.cc.

```

264 {
265
266     vector<TaskAlloc*>::iterator iter;
267     if (tasks_alloc_list_.size()==0) return;
268     // std::cout << "Prov: " << provider->id_ << "\n";
269     /* update task computing rates to see which tasks are completed */
270     updateTskComputingRates(c_mips);
271     removeCompleted();
272
273     /* set server computing rate */
274     setComputingRate();
275
276     /* compute next deadline */
277     double nextDeadline = DBL_MAX;
278     for (iter = tasks_alloc_list_.begin(); iter !=
tasks_alloc_list_.end(); iter++)
279     {
280         if (nextDeadline > (*iter)->execTime()){
281             nextDeadline = (*iter)->execTime();
282         }
283     }
284
285     provider->scheduleNextExent(nextDeadline);
286
287
288 }
```

**4.9.3.25 void CoreScheduler::updateTskList ( )**

Definition at line 290 of file corescheduler.cc.

```

290     {
291         updateTskList(this->current_mips_);
292     }
```

**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 resource 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.

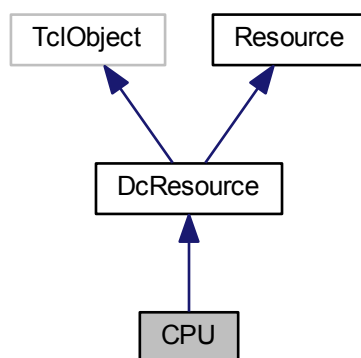
The documentation for this class was generated from the following files:

- [corescheduler.h](#)
- [corescheduler.cc](#)

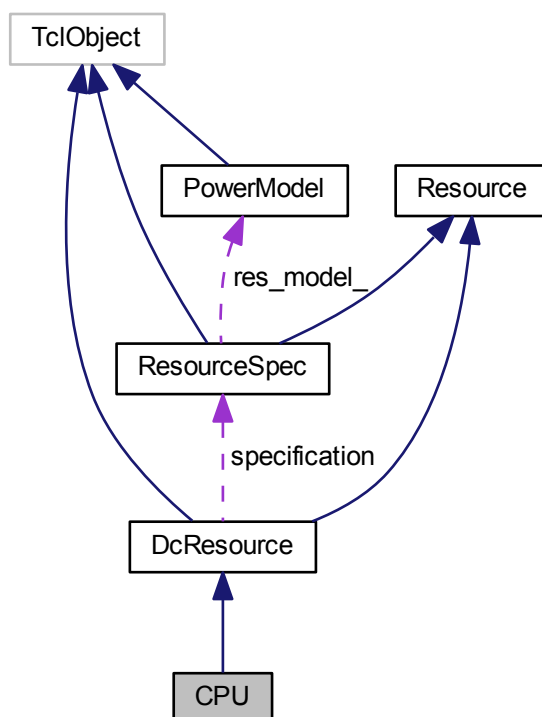
## 4.10 CPU Class Reference

```
#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.

```

45                                     {
46
47         if (argc == 2) {
48             if (strcmp(argv[1], "print") == 0) {
49                 this->print();
50                 return (TCL_OK);
51             } else if (strcmp(argv[1], "getMIPS") == 0) {
52                 this->getMIPS();
53                 return (TCL_OK);
54             }
55         }
56         return (CPU::command(argc, argv));
57     }

```

## 4.10.3.2 double CPU::getCurrentMIPS ( )

Definition at line 110 of file cpu.cc.

```

110                                     {
111         std::vector<CoreScheduler*>::iterator iter;
112         double result = 0;
113         for(iter= cores_schedulers_.begin();
114             iter != cores_schedulers_.end();
115             iter++){
116
117             result += (*iter)->getCurrentMIPSRecursive();
118         }
119         return result;
120     }

```

## 4.10.3.3 void CPU::getMIPS ( ) [private]

Definition at line 31 of file cpu.cc.

```

31         {
32             double result = 0;
33             std::vector<Capacity>::iterator iter;
34             for (iter = this->specification->capacity.begin(); iter!=this->
specification->capacity.end(); iter++)
35             {
36                 result += (*iter);
37             }
38             char out[100];
39             sprintf(out, "set tmp_cpu_mips %.0f", result);
40             Tcl& tcl = Tcl::instance();
41             tcl.eval(out);
42
43     }

```

## 4.10.3.4 double CPU::getNominalMIPS ( )

Definition at line 121 of file cpu.cc.

```

121                                     {
122         return nominal_mips_;
123     }

```

#### 4.10.3.5 double CPU::getUtilization ( ) [virtual]

Reimplemented from [DcResource](#).

Definition at line 125 of file cpu.cc.

```
125         {
126             return getCurrentMIPS()/getNominalMIPS();
127     }
```

#### 4.10.3.6 void CPU::print ( ) [virtual]

Reimplemented from [DcResource](#).

Definition at line 26 of file cpu.cc.

```
26     {
27         std::cerr << "CPU: ";
28         DcResource::print();
29     }
```

#### 4.10.3.7 void CPU::setDVFS ( int eDVFS\_enabled\_ )

Definition at line 98 of file cpu.cc.

```
98         {
99             if(cores_schedulers_.empty()){
100                 std::cerr << "No core schedulers!\n";
101             }
102             std::vector<CoreScheduler*>::iterator iter;
103             for(iter= cores_schedulers_.begin();
104                 iter != cores_schedulers_.end();
105                 iter++){
106                 (*iter)->setDVFS(eDVFS_enabled_);
107             }
108     }
```

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

Sets the resource provider of the resource.

Definition at line 85 of file cpu.cc.

```
85         {
86             if(cores_schedulers_.empty()){
87                 std::cerr << "No core schedulers!\n";
88             }
89             std::vector<CoreScheduler*>::iterator iter;
90             for(iter= cores_schedulers_.begin();
91                 iter != cores_schedulers_.end();
92                 iter++){
93                 (*iter)->setProvider(provider);
94             }
95         }
96     }
```

## 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.

```

61                                     {
62         if(resspec==NULL){
63             std::cerr << "ERROR: Null pointer passed as ResourceSpec.";
64             return 1;
65         }
66         this->DcResource::setSpecification(resspec);
67         std::vector<Capacity>::iterator iter_nominal;
68         std::vector<Capacity>::iterator iter_reserved;
69         for(iter_nominal= resspec->capacity.begin(),iter_reserved=this->
capacity.begin();
70             iter_nominal != resspec->
capacity.end();
71             iter_nominal++,iter_reserved++){
72             cores_schedulers_.push_back(new
CoreScheduler (&(*iter_nominal), &(*iter_reserved)));
73         }
74         std::vector<CoreScheduler*>::iterator iter;
75         nominal_mips_ = 0;
76         for(iter= cores_schedulers_.begin();
77             iter != cores_schedulers_.end();
78             iter++){
79             nominal_mips_ += (*iter)->getNominalMIPS();
80         }
81         return 0;
82     }
83 }
```

## 4.10.4 Member Data Documentation

## 4.10.4.1 std::vector&lt;CoreScheduler\*&gt; 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]

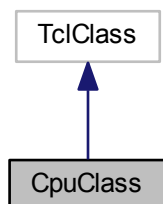
Definition at line 41 of file cpu.h.

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

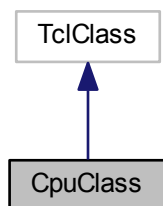
- [cpu.h](#)
- [cpu.cc](#)

## 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 Tclobj\* CpuClass::create ( int argc, const char \*const \* argv ) [inline]

Definition at line 13 of file cpu.cc.

```

13                                     {
14         return (new CPU());
15     }

```

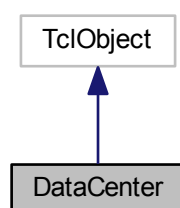
The documentation for this class was generated from the following file:

- [cpu.cc](#)

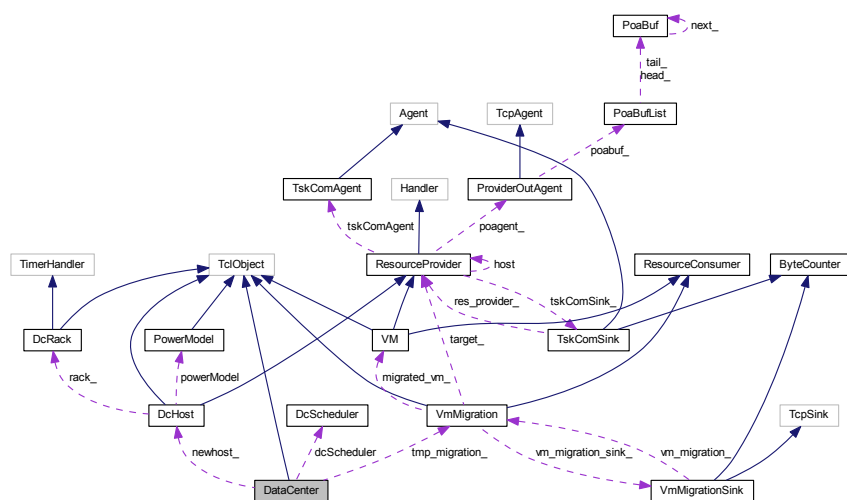
## 4.12 DataCenter Class Reference

```
#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.

```

22             : tskSubmitted_ (0),tskFailed_ (0),
  avgLoad_(0.0), avgLoadMem_(0.0),avgLoadStor_(0.0),
  avgPower_(0.0), dcScheduler (NULL), numHostTskAgents_(0),
  numVmTskAgents_(0),scheduleOnVms_(false)
23
24 {
25     bind("tskSubmitted_", &tskSubmitted_);
26     bind("avgLoad_", &avgLoad_);
27     bind("avgLoadMem_", &avgLoadMem_);
28     bind("avgLoadStor_", &avgLoadStor_);
29     bind("avgPower_", &avgPower_);
30     bind("tskFailed_", &tskFailed_);
31     //     dcScheduler = new GreenScheduler();
32 }
```

##### 4.12.2.2 DataCenter::~DataCenter ( ) [virtual]

Definition at line 35 of file datacenter.cc.

```

36 {
37     clear();
38 }
```

#### 4.12.3 Member Function Documentation

##### 4.12.3.1 void DataCenter::addHostPointer ( DcHost \* newhst )

Definition at line 65 of file datacenter.cc.

```

66 {
67     host_list.push_back(newhst);
68 }
```

##### 4.12.3.2 void DataCenter::addHostTaskAgentPointer ( TskComAgent \* newagnt )

Definition at line 74 of file datacenter.cc.

```

75 {
76     newagnt->set_pkttype (PT_CLOUD_USER);
77     host_agent_list.push_back(newagnt);
78 }
```

#### 4.12.3.3 void DataCenter::addPModelPointer ( PowerModel \* newPModel )

Register power model in Data Center.

Definition at line 91 of file datacenter.cc.

```

91                                     {
92         power_model_list.push_back(newPModel);
93     }
```

#### 4.12.3.4 void DataCenter::addResourceSpecificationPointer ( ResourceSpec \* newresspec )

Registers resource used in Data Center.

Definition at line 86 of file datacenter.cc.

```

87 {
88         resource_specification_list.push_back(newresspec);
89 }
```

#### 4.12.3.5 void DataCenter::addVirtualResourceSpecificationPointer ( ResourceSpec \* newresspec )

Registers virtual resource used in Data Center.

Definition at line 96 of file datacenter.cc.

```

97 {
98         virt_resource_specification_list.push_back(newresspec);
99 }
```

#### 4.12.3.6 void DataCenter::addVmPointer ( VM \* newvm )

Definition at line 70 of file datacenter.cc.

```

70                                     {
71         vm_list.push_back(newvm);
72     }
```

#### 4.12.3.7 void DataCenter::addVmTaskAgentPointer ( TskComAgent \* newagnt )

Definition at line 80 of file datacenter.cc.

```

81 {
82         newagnt->set_pkttype (PT_CLOUD_USER);
83         vm_agent_list.push_back(newagnt);
84 }
```

## 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
43             object.
44             host_list.clear();
45             host_agent_list.clear();
46             vector <ResourceSpec*>::iterator rs;
47             for(rs = resource_specification_list.begin(); rs!=
48                 resource_specification_list.end(); rs++){
49                 delete (* rs);
50             }
51             resource_specification_list.clear();
52             vector <ResourceProvider*>::iterator iter;
53             for(iter = host_list.begin(); iter!= host_list.end(); iter++){
54                 delete (* iter);
55             }
56             for(iter = vm_list.begin(); iter!= vm_list.end(); iter++){
57                 delete (* iter);
58             }
59             host_list.clear();
60             vm_list.clear();
61         }

```

## 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<
246             DcHost*> (TclObject::lookup(argv[2]));
247             if(hst){
248                 addHostPointer(hst);
249                 return (TCL_OK);
250             }
251             return (TCL_ERROR);
252         } else if (strcmp(argv[1], "add-vm") == 0) {
253             VM *vm = dynamic_cast<VM*> (TclObject::lookup(argv[2]));
254             if(vm){
255                 addVmPointer(vm);
256                 return (TCL_OK);
257             }
258             return (TCL_ERROR);
259         } else if (strcmp(argv[1], "add-hosttaskagent") == 0) {
260             TskComAgent *agnt = dynamic_cast<
261             TskComAgent*> (TclObject::lookup(argv[2]));
262             if(agnt){
263                 addHostTaskAgentPointer(agnt);
264                 numHostTskAgents_ = (int) host_agent_list.size();

```

```

264                                     return (TCL_OK);
265                                     }
266                                     return (TCL_ERROR);
267             } else if (strcmp(argv[1], "add-vmtaskagent") == 0) {
268                 TskComAgent *agnt = dynamic_cast<
TskComAgent*> (TclObject::lookup(argv[2]));
269                 if(agnt){
270
271                     addVmTaskAgentPointer(agnt);
272                                     numVmTskAgents_ = (int)
vm_agent_list.size();
273                                     return (TCL_OK);
274                                     }
275                                     return (TCL_ERROR);
276             } else if (strcmp(argv[1], "add-resspec") == 0) {
277                 ResourceSpec *resspec = dynamic_cast<
ResourceSpec*> (TclObject::lookup(argv[2]));
278                 if(resspec){
279
280                     addResourceSpecificationPointer(resspec);
281                                     return (TCL_OK);
282                                     }
283                                     return (TCL_ERROR);
284             } else if (strcmp(argv[1], "add-vresspec") == 0) {
ResourceSpec *resspec = dynamic_cast<
285                 ResourceSpec*> (TclObject::lookup(argv[2]));
286                 if(resspec){
287
288                     addVirtualResourceSpecificationPointer(resspec);
289                                     return (TCL_OK);
290                                     }
291                                     return (TCL_ERROR);
292             } else if (strcmp(argv[1], "add-pmodel") == 0) {
PowerModel *pmodel = dynamic_cast<
293                 PowerModel*> (TclObject::lookup(argv[2]));
294                 if(pmodel){
295
296                     addPModelPointer(pmodel);
297                                     return (TCL_OK);
298                                     }
299                                     return (TCL_ERROR);
300             } else if (strcmp(argv[1], "get-newest-migration") == 0) {
VmMigration *mig = dynamic_cast<
301                 VmMigration*> (TclObject::lookup(argv[2]));
302                 if(mig){
303
304                     tmp_migration_=mig;
305                                     return (TCL_OK);
306                                     }
307                                     return (TCL_ERROR);
308             } else if (strcmp(argv[1], "set-scheduler") == 0) {
309                 if(setScheduler(argv[2])){
310                                     return (TCL_OK);
311                                     }
312                                     return (TCL_ERROR);
313             } else if (argc == 4) {
314                 if (strcmp(argv[1], "configure-resource") == 0) {
DcResource *res = dynamic_cast<
315                 DcResource*> (TclObject::lookup(argv[2]));
316                 if(res){
317
318                     if(
319                                     return (TCL_OK);
320                                     } else {
321                                     return (TCL_ERROR);
322                                     }
323                                     }
324                                     return (TCL_ERROR);
325             } else if (strcmp(argv[1], "configure-vresource") == 0) {
DcResource *res = dynamic_cast<
326                 DcResource*> (TclObject::lookup(argv[2]));
327                 if(res){
328
329                     if(
330                                     return (TCL_OK);
331                                     } else {
332                                     return (TCL_ERROR);
333                                     }
334                                     }
335                                     return (TCL_ERROR);
336             } else if (strcmp(argv[1], "migrate-vm") == 0) {
VM *vm = dynamic_cast<VM*> (TclObject::lookup(argv[2]));
337                 ResourceProvider *target = dynamic_cast<
ResourceProvider*> (TclObject::lookup(argv[3]));
338                 if(vm && target){
339                     migrateVm( vm, target);

```

```

336                                     return (TCL_OK);
337                                     } else {
338                                     return (TCL_ERROR);
339                                     }
340                                     }
341                                     return (TCL_ERROR);
342
343
344                                     }
345                                     return (DataCenter::command(argc, argv));
346 }

```

#### 4.12.3.10 void DataCenter::computeLoad ( ) [protected]

Definition at line 348 of file datacenter.cc.

```

349 {
350     /* Traverse servers and compute their average load */
351     vector <ResourceProvider*>::iterator iter;
352
353     double avgLoad = 0;
354     double avgLoadMem = 0;
355     double avgLoadStor = 0;
356     double avgPower = 0;
357     for (iter = host_list.begin(); iter!=host_list.end(); iter++)
358     {
359
360         Computing);
361         Memory);
362         Storage);
363         avgPower += ((DcHost*) (*iter))>eCurrentConsumption_;
364
365     }
366
367     for (iter = vm_list.begin(); iter!=vm_list.end(); iter++)
368     {
369         Computing);
370         Memory);
371         Storage);
372     }
373
374
375     avgLoad_ = avgLoad / host_list.size();
376     avgLoadMem_ = avgLoadMem / host_list.size();
377     avgLoadStor_ = avgLoadStor / host_list.size();
378     avgPower_ = avgPower / host_list.size();
379 }
380 }

```

#### 4.12.3.11 int DataCenter::configureResource ( DcResource \* confRes, const char \* spec\_name )

Definition at line 391 of file datacenter.cc.

```

391                                     {
392     vector <ResourceSpec*>::iterator iter;
393     int result = 1;
394     std::string test = spec_name;
395     for (iter = resource_specification_list.begin(); iter!=
resource_specification_list.end(); iter++)
396     {
397         if ((*iter)>name_==test){
398             result = 0;
399             break;
400         }
401     }
402     if(result==0){
403         confRes->setSpecification(*iter);
404     } else {
405         std::cerr << "ERROR: The requested resource specification is not registered
in the data center!\n";
406     }
407     return result;
408 }

```

#### 4.12.3.12 int DataCenter::configureVirtualResource ( DcResource \* *confRes*, const char \* *spec\_name* )

Definition at line 410 of file datacenter.cc.

```

410                                     {
411         vector <ResourceSpec*>::iterator iter;
412         int result = 1;
413         std::string test = spec_name;
414         for (iter = virt_resource_specification_list.begin(); iter!
= virt_resource_specification_list.end(); iter++)
415         {
416                 if ((*iter)->name_==test) {
417                         result = 0;
418                         break;
419                 }
420         }
421         if(result==0){
422                 confRes->setSpecification(*iter);
423         } else {
424                 std::cerr << "ERROR: The requested virtual resource specification is not
registered in the data center!\n";
425         }
426         return result;
427 }

```

#### 4.12.3.13 void DataCenter::createNewMigration ( ) [private]

Definition at line 142 of file datacenter.cc.

```

142                                     {
143         // Create and allocate VmMigration object
144         Tcl& tcl = Tcl::instance();
145         tcl.evalf("set vmmigration_($next_migration_id) [new VmMigration]");
146         tcl.evalf("$vmmigration_($next_migration_id) set-id $next_migration_id");
147         tcl.evalf(
148                 "$DCenter get-newest-migration
149         $vmmigration_($next_migration_id)");
150 }

```

#### 4.12.3.14 int DataCenter::initiallyConfigureVms ( )

TODO: Dynamically configure the initial state of VMs.

Definition at line 101 of file datacenter.cc.

```

102 {
103         char output[100];
104         // Create the output for the Tcl interpreter.
105         sprintf(output, "puts \"Configuring VMs...\"");
106         Tcl& tcl = Tcl::instance();
107         tcl.eval(output);
108         return 0;
109 }

```

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

Definition at line 151 of file datacenter.cc.

```

151                                     {
152         createNewMigration(); // Creates and sets new migrations as
tmp_migration_
153         tmp_migration_->inititalizeMigration(vm,target);
154
155         tmp_migration_->startMigration();
156         // The finalization of migration will be done by the migration object.
157         // The data center (probably) does not need to be informed about that, as the vm agent will
be re-attached to the new location.
158         tmp_migration_ = NULL;
159 }

```

## 4.12.3.16 void DataCenter::printResourceSpecs ( )

Definition at line 382 of file datacenter.cc.

```

382         {
383             vector <ResourceSpec*>::iterator iter;
384             for (iter = resource_specification_list.begin(); iter!=
resource_specification_list.end(); iter++)
385             {
386                 (*iter)->print();
387             }
388         }
389     }

```

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

Definition at line 161 of file datacenter.cc.

```

162 {
163     /* Update Stats */
164     tskSubmitted_ ++;
165
166     /* Schedule it */
167     TskComAgent *tagent_ ;
168     if(scheduleOnVms_)// You can switch to another scheduler or providers list
using this variable.
169     {
170         tagent_ = dcScheduler->scheduleTask((
CloudTask *) pTask,vm_list); }
171     else {
172         tagent_ = dcScheduler->scheduleTask((
CloudTask *) pTask,host_list);
173     }
174     if(tagent_==NULL){
175         tskFailed_++;
176     } else {
177         tagent_->sendmsg(tsksize, pTask, flags);
178     }
179 }

```

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

Definition at line 188 of file datacenter.cc.

```

189 {
190     vector <ResourceProvider*>::iterator iter;
191
192     for (iter = host_list.begin(); iter!=host_list.end(); iter++)
193     {
194         if ((*iter)->trySchedulingTsk(tsk))
195             return (*iter)->getTskComAgent();
196         // return (host_agent_list.at ((*iter)->id_));
197     }
198     return NULL;
199 }

```

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

Definition at line 208 of file datacenter.cc.

```

209 {
210     vector <ResourceProvider*>::iterator iter;
211
212     for (iter = vm_list.begin(); iter!=vm_list.end(); iter++)
213     {
214         if ((*iter)->trySchedulingTsk(tsk))
215             return (*iter)->getTskComAgent();
216         // return (vm_agent_list.at ((*iter)->id_));
217     }
218     return NULL;
219 }

```

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

Definition at line 181 of file datacenter.cc.

```
182 {
183     int j = tskSubmitted_% numHostTskAgents_;
184
185     return (host_agent_list.at(j));
186 }
```

#### 4.12.3.21 TskComAgent \* DataCenter::scheduleRoundRobin ( CloudTask \* *tsk*, std::vector< TskComAgent \* > *agent\_list* ) [protected]

Definition at line 201 of file datacenter.cc.

```
202 {
203     int j = tskSubmitted_% agent_list.size();
204
205     return (agent_list.at(j));
206 }
```

#### 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.

```
111                                     {
112     if(dcScheduler!=NULL) {
113         delete dcScheduler;
114     }
115     std::cout << "Selected DC scheduler: " << scheduler_name << "\n";
116     if(strcmp(scheduler_name, "Green") == 0){
117         dcScheduler = new GreenScheduler();
118         return 1;
119     } else if(strcmp(scheduler_name, "RoundRobin") == 0){
120         dcScheduler = new
RoundRobinsScheduler();
121         return 1;
122     } else if(strcmp(scheduler_name, "Random") == 0){
123         dcScheduler = new RandomScheduler();
124         return 1;
125     } else if(strcmp(scheduler_name, "RandDENS") == 0){
126         dcScheduler = new RandDENS();
127         return 1;
128     } else if(strcmp(scheduler_name, "BestDENS") == 0){
129         dcScheduler = new BestDENS();
130         return 1;
131     } else if(strcmp(scheduler_name, "HEROS") == 0){
132         dcScheduler = new HerosScheduler();
133         return 1;
134     }
135
136
137     std::cerr << "Unknown scheduler type: " << scheduler_name;
138     abort();
139
140 }
```

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

Definition at line 61 of file datacenter.cc.

```
61                                     {
62     this->scheduleOnVms_ = scheduleOnVms;
63 }
```

#### 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]

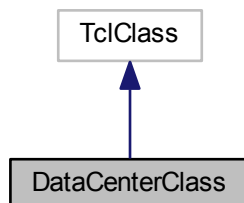
Definition at line 62 of file datacenter.h.

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

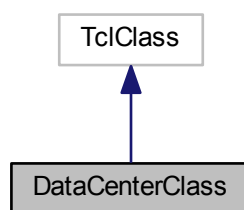
- [datacenter.h](#)
- [datacenter.cc](#)

## 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`.

```
16 : 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`.

```
17                                     {  
18         return (new DataCenter());  
19     }
```

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

- [datacenter.cc](#)



## 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 &amp; Destructor Documentation

## 4.14.2.1 DcHost::DcHost ( )

Definition at line 16 of file dchost.cc.

```

16         : eConsumed_(0.0), eNominalrate_(0.0),
          eCurrentConsumption_(0.0), eDNS_enabled_(0.0) ,
          eLastUpdateTime_(0.0)
17 {
18     bind("id_", &id_);
19     bind("ntasks_", &ntasks_);
20     bind("currentLoad_", &currentLoad_);
21     bind("currentLoadMem_", &currentLoadMem_);
22     bind("currentLoadStor_", &currentLoadStor_);
23     bind("tskFailed_", &tskFailed_);
24
25     bind("eConsumed_", &eConsumed_);
26     /* total W of energy consumed */
27     bind("eNominalrate_", &eNominalrate_);
28     bind("eCurrentConsumption_", &eCurrentConsumption_);
29     /* current consumption rate */
30
31     bind("eDVFS_enabled_", &eDVFS_enabled_);
32     /* ON when DVFS is enabled */
33     bind("eDNS_enabled_", &eDNS_enabled_);
34     /* ON when DNS is enabled */
35 }

```

#### 4.14.2.2 DcHost::~DcHost ( ) [virtual]

Definition at line 34 of file dchost.cc.

```
35 {
36     delete powerModel;
37 }
```

### 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.

```
130     {
131         ResourceProvider::addResource(res);
132         if(res->specification->getPowerModel() != NULL) {
133             powerModel->addComponent(res);
134         }
135 }
```

#### 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.

```
45 {
46     Tcl& tcl = Tcl::instance();
47
48     if (argc == 2) {
49         if (strcmp(argv[1], "start") == 0) {
50
51             /* start counting energy consumed */
52             setCurrentConsumption();
53             eLastUpdateTime_ = Scheduler::instance().
54                 clock();
55             started_ = true;
56             return (TCL_OK);
57         } else if (strcmp(argv[1], "stop") == 0) {
58             /* update total energy consumed */
59             updateEnergyAndConsumption();
60             return (TCL_OK);
61         } else if (strcmp(argv[1], "print") == 0) {
62             /* print general info */
63             print();
64             return (TCL_OK);
65         }
66     } else if (argc == 3) {
67         if (strcmp(argv[1], "set-power-model") == 0) {
68             PowerModel* pModel = (
69                 PowerModel*) TclObject::lookup(argv[2]);
70             if (pModel == NULL) {
71                 tcl.resultf("no such power model %s", argv[
72                     2]);
73                 return (TCL_ERROR);
74             }
75             setPowerModel(pModel);
76             return (TCL_OK);
77         }
78     }
79     return (ResourceProvider::command(argc, argv));
80 }
```

## 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;
124     /* time in hours */
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.

```

39                                     {
40         powerModel = pModel;
41     }
```

#### 4.14.3.8 void DcHost::updateEnergyAndConsumption ( ) [virtual]

Implements [ResourceProvider](#).

Definition at line 138 of file dchost.cc.

```

138                                     {
139         setCurrentConsumption();
140         eUpdate();
141     }
```

### 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\_

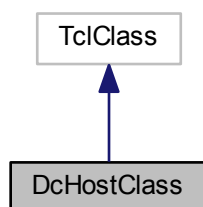
Definition at line 33 of file dchost.h.

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

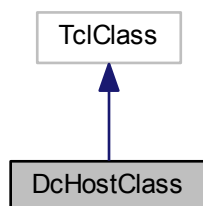
- [dchost.h](#)
- [dchost.cc](#)

## 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.

```
10 : 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.

```
11                                     {
12                                     return (new DcHost());
13                                     }
```

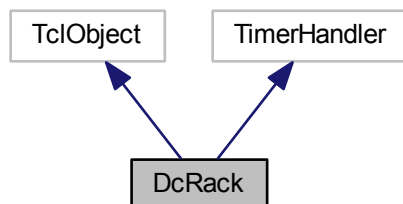
The documentation for this class was generated from the following file:

- [dchost.cc](#)

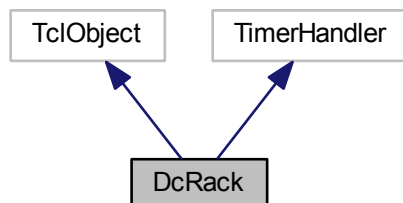
## 4.16 DcRack Class Reference

```
#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](#).

```

15         : rack\_id\_(0), stat\_interval (0.0),
      breceived\_(0), breceived\_old\_(0)
16 {
17     bind("rack_id_", &rack\_id\_);
18     bind("breceived_", &breceived\_);
19     bind("stat_interval", &stat\_interval);
20     bind("uplink_B", &uplink\_B);
21 }
```

**4.16.2.2 DcRack::~~DcRack ( ) [virtual]**

Definition at line 23 of file [dcrack.cc](#).

```

24 {
25     hosts\_list\_.~vector();
26     qmon\_uplink\_list.~vector();
27 }
```

### 4.16.3 Member Function Documentation

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

Definition at line 60 of file dcrack.cc.

```

60         {
61             hosts_list_.push_back(hst);
62             hst->rack_=this;
63     }
```

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

Definition at line 65 of file dcrack.cc.

```

66 {
67     if (argc == 2) {
68         if (strcmp(argv[1], "update-stats") == 0) {
69             updatestats();
70             return (TCL_OK);
71         } else if (strcmp(argv[1], "start") == 0) {
72             expire(new Event());
73             return (TCL_OK);
74         }
75     } else if (argc == 3) {
76         if (strcmp(argv[1], "add-dchost") == 0) {
77             DcHost *hst = dynamic_cast<
DcHost*> (TclObject::lookup(argv[2]));
78             if(hst) {
79                 addHost(hst);
80                 return (TCL_OK);
81             }
82             return (TCL_ERROR);
83         }
84         else if (strcmp(argv[1], "add-uplink-qmon") == 0) {
85             QueueMonitor *uplinkqmon = dynamic_cast<QueueMonitor*> (
TclObject::lookup(argv[2]));
86             if(uplinkqmon) {
87                 qmon_uplink_list.push_back(
uplinkqmon);
88
89                 return (TCL_OK);
90             }
91             return (TCL_ERROR);
92         }
93     }
94     return (DcRack::command(argc, argv));
95 }
```

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

Definition at line 54 of file dcrack.cc.

```

54         {
55
56             updatestats();
57             this->resched(stat_interval);
58     }
```

**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

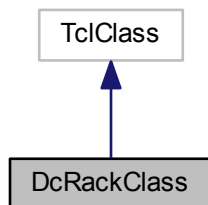
Definition at line 18 of file dcrack.h.

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

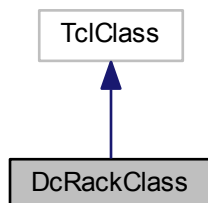
- [dcrack.h](#)
- [dcrack.cc](#)

### 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 &amp; 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.

```
10
11                                     return (new DcRack());
12 }
```

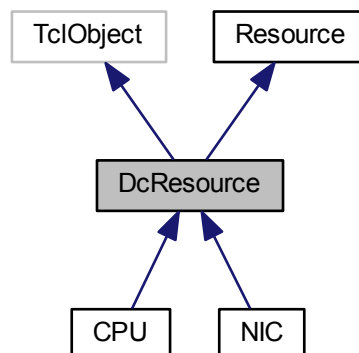
The documentation for this class was generated from the following file:

- [dcrack.cc](#)

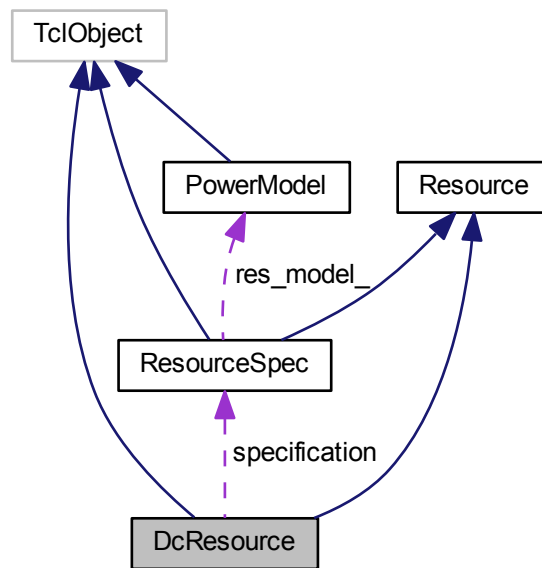
## 4.18 DcResource Class Reference

```
#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 &amp; 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.

```

58         {
59             int n = 1;
60             double* utilization = new double[n];
61             utilization[0] = getUtilization();
62             double result = specification->getPowerModel()->
estimate(n,utilization);
63             delete utilization;
64             return result;
65     }
```

#### 4.18.3.4 double DcResource::getUtilization ( ) [virtual]

Reimplemented in [CPU](#), and [NIC](#).

Definition at line 46 of file dcresource.cc.

```

46         {
47             double free = 0;
48
49             std::vector<Capacity>::iterator iter;
50             for(iter=capacity.begin(); iter!= capacity.end();iter++){
51                 free += iter->getValueRecursive();
52             }
53             return 1-(free/total_cap);
54
55     }
```

#### 4.18.3.5 void DcResource::print ( ) [virtual]

Reimplemented in [CPU](#).

Definition at line 81 of file dcresource.cc.

```

81         {
82             std::cout << "DcResource";
83             std::cout << "\n";
84             specification->print();
85             std::cout << "Available capacities:\t";
86             std::vector<Capacity>::iterator iter;
87             for (iter = capacity.begin(); iter!=capacity.end(); iter++)
88             {
89                 std::cout << (*iter) << ",";
90             }
91             std::cout << "\n";
92             std::cout << "Used power state:\t" << used_power_state_;
93
94             std::cout << "\n";
95     }
```

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

Reimplemented in [CPU](#).

Definition at line 28 of file dcresource.cc.

```

28         {
29             if(resspec==NULL){
30                 std::cerr << "ERROR: Null pointer passed as ResourceSpec.";
31                 return 1;
32             }
33             specification=resspec;
34             capacity = resspec->capacity;
35             used_power_state_ = specification->
power_states.at(0);
36             type=resspec->type;
37             arch=resspec->arch;
38             total_cap = 0;
39             std::vector<Capacity>::iterator iter;
40             for(iter = resspec->capacity.begin();iter!= resspec->
capacity.end();iter++){
41                 total_cap += iter->value;
42             }
43             return 0;
44     }
```

#### 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

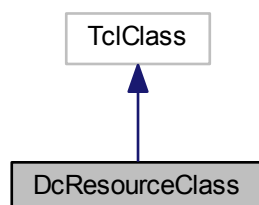
Definition at line 36 of file dcresource.h.

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

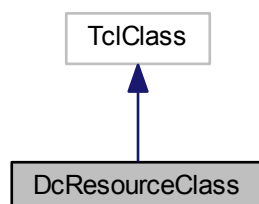
- [dcresource.h](#)
- [dcresource.cc](#)

## 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.

```
12 : 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.

```
13                                     {
14                                     return (new DcResource());
15                                     }
```

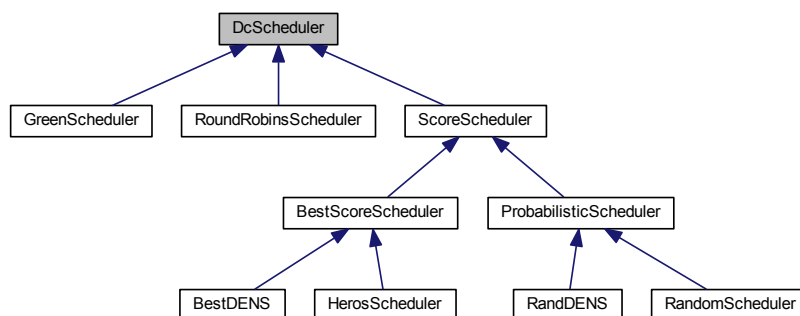
The documentation for this class was generated from the following file:

- [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 &amp; 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]

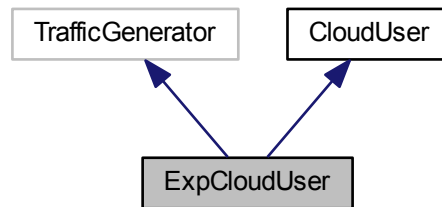
Implemented in [HerosScheduler](#), [BestDENS](#), [RandDENS](#), [ProbabilisticScheduler](#), [RandomScheduler](#), [BestScoreScheduler](#), [GreenScheduler](#), and [RoundRobinsScheduler](#).

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

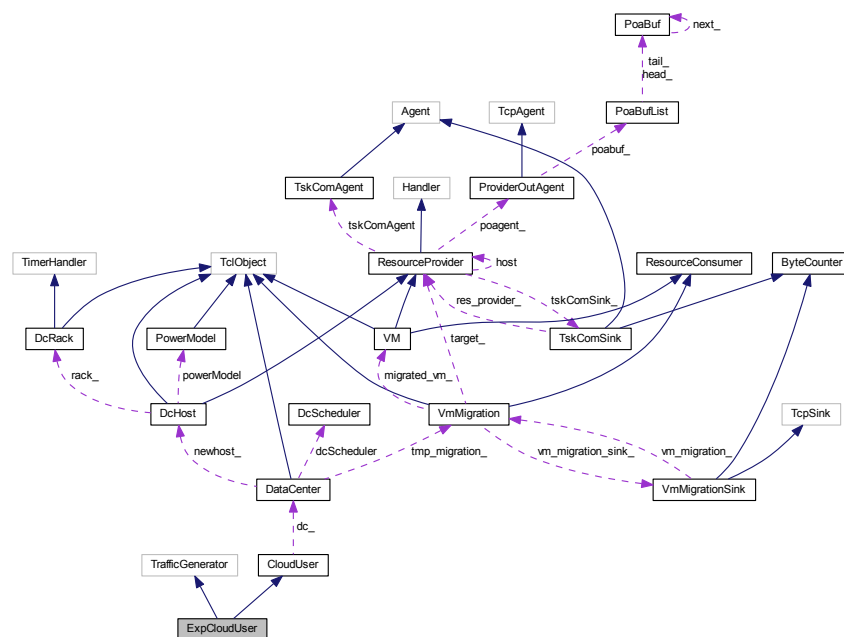
- [dcscheduler.h](#)
- [dcscheduler.cc](#)

## 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 &amp; 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.

```

57                                     {
58     int result = CloudUser::process_command(argc,argv);
59     if(result!=-1){
60     if(argc==3){
61         if (strcmp(argv[1], "use-rng") == 0) {
62             burstlen_.seed((char *)argv[2]);
63             Offtime_.seed((char *)argv[2]);
64             return (TCL_OK);
65         }
66         //ADDED CODE
67         else if (strcmp(argv[1], "set-rate") == 0) {
68             int new_rate = atoi(argv[2]);
69             if(1){
70                 rate_=new_rate;
71                 interval_ = (double)(size_ << 3)/(
double)rate_;
72                                     }
73                                     return (TCL_OK);
74                                     return (TCL_ERROR);
75                                     }
76                                     //ADDED CODE
77     }
78     return Application::command(argc,argv);
79     } else {
80         return result;
81     }
82 }

```

#### 4.21.3.3 void ExpCloudUser::init ( ) [protected]

Definition at line 107 of file expclouduser.cc.

```

108 {
109     /* compute inter-packet interval during bursts based on
110     * packet size and burst rate. then compute average number
111     * of packets in a burst.
112     */
113     interval_ = (double)(size_ << 3)/(double)rate_;
114     burstlen_.setavg(ontime_/interval_);
115     rem_ = 0;
116 }

```

#### 4.21.3.4 double ExpCloudUser::next\_interval ( int & size ) [virtual]

Definition at line 118 of file expclouduser.cc.

```

119 {
120     double t = interval_;
121
122     if (rem_ == 0) {
123         /* compute number of packets in next burst */
124         rem_ = int(burstlen_.value() + .5);
125         /* make sure we got at least 1 */
126         if (rem_ == 0)
127             rem_ = 1;
128         /* start of an idle period, compute idle time */
129         t += Offtime_.value();
130     }
131     rem_--;
132
133     size = size_;
134     //TODO: add change factor (for decay the change should be >1):interval_ = interval_ *
change_ ;
135     // OR: use change by constant term as below:
136     //         interval_ = interval_ + 0.000005;
137     // if(interval_<=0) interval_ = 0.000005;
138     return(t);
139 }

```

**4.21.3.5 void ExpCloudUser::timeout ( ) [virtual]**

Definition at line 141 of file expclouduser.cc.

```

142 {
143     if (! running_)
144         return;
145
146     if (nextPkttime_ != interval_ || nextPkttime_ == -1){
147         dc->receivedTsk(size_, createTask(), "NEW_BURST");
148     }
149     else {
150         dc->receivedTsk(size_,
151             createTask());
152     }
153
154     /* figure out when to send the next one */
155     nextPkttime_ = next_interval(size_);
156     /* schedule it */
157     if (nextPkttime_ > 0)
158         timer_.resched(nextPkttime_);
159 }
```

**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

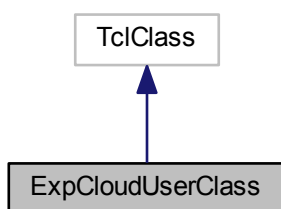
Definition at line 36 of file expclouduser.cc.

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

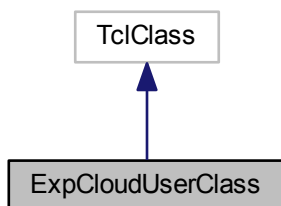
- [expclouduser.cc](#)

## 4.22 ExpCloudUserClass Class Reference

Inheritance diagram for ExpCloudUserClass:



Collaboration diagram for ExpCloudUserClass:



### Public Member Functions

- [ExpCloudUserClass](#) ()
- TcIObject \* [create](#) (int, const char \*const \*)

## 4.22.1 Detailed Description

Definition at line 44 of file expclouduser.cc.

## 4.22.2 Constructor &amp; Destructor Documentation

## 4.22.2.1 ExpCloudUserClass::ExpCloudUserClass ( ) [inline]

Definition at line 46 of file expclouduser.cc.

```
46 : 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.

```
47                                     {
48                                     return (new ExpCloudUser());
49                                     }
```

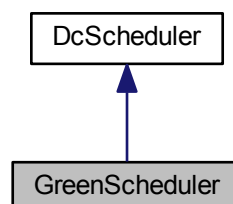
The documentation for this class was generated from the following file:

- [expclouduser.cc](#)

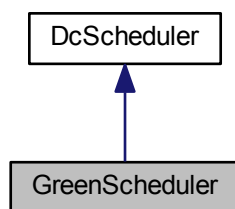
## 4.23 GreenScheduler Class Reference

```
#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.

```

11         {
12
13
14     }
```

##### 4.23.2.2 `GreenScheduler::~~GreenScheduler ( )` [virtual]

Definition at line 16 of file greenscheduler.cc.

```

16         {
17
18     }
```

## 4.23.3 Member Function Documentation

## 4.23.3.1 TskComAgent \* GreenScheduler::scheduleTask ( CloudTask \* task, std::vector&lt; ResourceProvider \* &gt; providers ) [virtual]

Implements [DcScheduler](#).

Definition at line 20 of file greenscheduler.cc.

```

20                                     {
21         vector <ResourceProvider*>::iterator iter;
22
23         for (iter = providers.begin(); iter!=providers.end(); iter++)
24         {
25             if ((*iter)->trySchedulingTsk(task))
26                 return (*iter)->getTskComAgent();
27         }
28         return NULL;
29     }
```

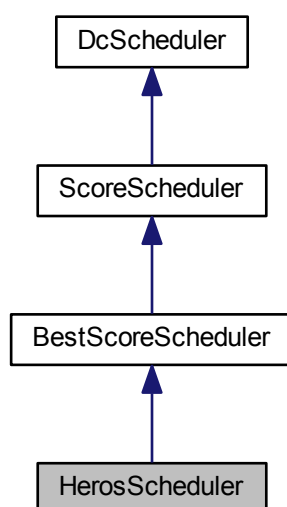
The documentation for this class was generated from the following files:

- [greenscheduler.h](#)
- [greenscheduler.cc](#)

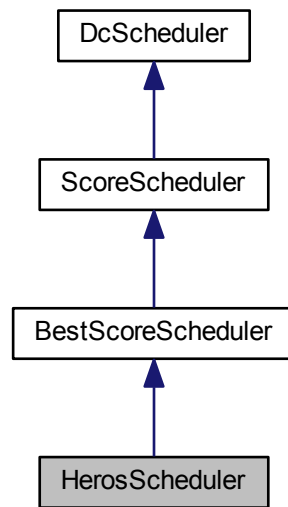
## 4.24 HerosScheduler Class Reference

```
#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 &amp; Destructor Documentation

## 4.24.2.1 HerosScheduler::HerosScheduler ( )

Definition at line 11 of file herosscheduler.cc.

```

11             : epsilon(0.1) {
12
13
14 }
```

## 4.24.2.2 HerosScheduler::~HerosScheduler ( ) [virtual]

Definition at line 16 of file herosscheduler.cc.

```

16             {
17
18 }
```

## 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.

```

21             {
22         double result = 0;
23
24         result = herosTransformation(rp,110,0.90,1.2);
25         // std::cerr << "Heros transformation result" << result << "\n";
26         result *= pow(linkLoadFactor(rp->getRootHost()->
27         rack_->link_load),2);
27         // std::cerr << "Final result" << result << "\n";
28         return result;
29 }
```

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

Definition at line 32 of file herosscheduler.cc.

```

32             {
33         return 1/(1+exp(-10*(load-0.5))) - 1/(1+exp((-10/epsilon)*(load-(1-
34         epsilon/2)))));
34 }
```

#### 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.

```

76                                                                                                     {
77         //                                                                                         std::cerr<< "HEROS is making decision:\n";
78         vector<ProviderScore> scored_providers_;
79         vector<ResourceProvider*>::iterator iter;
80         for (iter = providers.begin(); iter!=providers.end(); iter++)
81         {
82             if ((*iter)->testSchedulingPossibility(task)){
83                 scored_providers_.push_back(
ProviderScore((*iter),calculateScore((*iter)),
linkLoadFactor((*iter)->getRootHost()->rack_->link_load));
84             }
85         }
86         if(scored_providers_.empty()){
87             return NULL;
88         } else {
89             vector<ProviderScore>::iterator sp;
90             sort(scored_providers_.begin(),scored_providers_.end(),
herosComparator);
91             vector<ProviderScore>::reverse_iterator rsp = scored_providers_.rbegin();
92             ProviderScore best = *rsp;
93             int max_n = 0;
94             for (; rsp != scored_providers_.rend(); rsp++) {
95                 if(!herosComparator((*rsp),best)){
96                     ++max_n;
97                 } else {
98                     break;
99                 }
100             }
101             if(max_n!=1){
102                 int selected = (double)rand() / (double)RAND_MAX * max_n +1;
103                 best = scored_providers_.at(scored_providers_.size()-
selected);
104             }
105             scored_providers_.clear();
106             std::cerr<< "Selected prov: " << best.provider_->id_ << "\tScore:" <<
best.score_ << "\tSelected task:" << task->id_ << "\n";
107 //
108             return best.provider_->getTskComAgent();
109         }
110     }
111 }
```

#### 4.24.4 Member Data Documentation

##### 4.24.4.1 double HerosScheduler::epsilon [private]

Definition at line 38 of file herosscheduler.h.

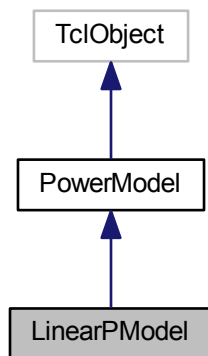
The documentation for this class was generated from the following files:

- [herosscheduler.h](#)
- [herosscheduler.cc](#)

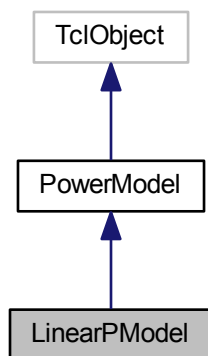
## 4.25 LinearPModel Class Reference

```
#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 &amp; Destructor Documentation

## 4.25.2.1 LinearPModel::LinearPModel ( )

Definition at line 20 of file `linearpmodel.cc`.

```

20         {
21
22             /*LastResType+2 = +1 for elements number, +1 for the intercept:*/
23             coefficients = new double[LastResType+2];
24             initialized = new bool[LastResType+2];
25             setCoefNumber(LastResType+2);
26             for(int i = 0; i < LastResType+2; i++){
27                 coefficients[i]= 0;
28                 initialized[i]=false;
29             }
30
31     }
```

## 4.25.2.2 LinearPModel::~LinearPModel ( ) [virtual]

Definition at line 33 of file `linearpmodel.cc`.

```

33         {
34
35             delete[] coefficients;
36             delete[] initialized;
37             name_.clear();
38     }
```

### 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.

```

52                                     {
53     /* Linear power model does not accept components*/
54     return;
55 }
```

#### 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.

```

129 {
130
131     if (argc == 2) {
132         if (strcmp(argv[1], "print") == 0) {
133             /* print general info */
134             print();
135             return (TCL_OK);
136         }
137     } else if (argc==3){
138         if (strcmp(argv[1], "set-name") == 0) {
139             setName(argv[2]);
140             return(TCL_OK);
141         } else if (strcmp(argv[1], "set-coef-number") == 0) {
142             setCoefficientNumber(atoi(argv[2]));
143             return(TCL_OK);
144         } else {
145             return(TCL_ERROR);
146         }
147     }
148     else if (argc == 4) {
149         if (strcmp(argv[1], "set-coefficient") == 0) {
150             setCoefficient(argv[2],atof(argv[3]));
151             return(TCL_OK);
152         } else if (strcmp(argv[1], "set-coefficient-numeric") == 0) {
153             setCoefficientNumeric(argv[2],atof(
154                 argv[3]));
155             return(TCL_OK);
156         } else {
157             return(TCL_ERROR);
158         }
159     }
160     return (TCL_ERROR);
161 }
```

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

Implements [PowerModel](#).

Definition at line 57 of file linearpmodel.cc.

```

57                                     {
58     if(ready){
59         if(size!= coef_number - 1){
60             std::cerr <<"Incorrect size of predictors array!\n";
61         }
62         double result = coefficients[size];
63         for(int i = 0; i < size; i++){
64             result += predictors[i] *
65                 coefficients[i];
66         }
67         return result;
68     } else {
69         std::cerr << "The model is not correctly initalized.\n" ;
70         print();
71         std::cerr << "Aborting simulation";
72         abort();
73     }
74 }
```

## 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 initialized 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.

```

40                                     {
41
42         coef_number = number;
43         if(coefficients != NULL){
44             delete[] coefficients;
45             delete[] initialized;
46         }
47         coefficients = new double[coef_number];
48         initialized = new bool[coef_number];
49     }
```

#### 4.25.3.9 void LinearPModel::updateInit ( ) [private]

Definition at line 120 of file linearpmodel.cc.

```

120                                     {
121         bool result = true;
122         for(int i = 0; i < coef_number-1; i++){
123             result = result && initialized[i];
124         }
125         ready = result;
126     }
```

### 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

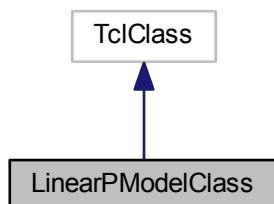
Definition at line 25 of file linearpmodel.h.

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

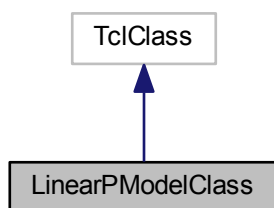
- [linearpmodel.h](#)
- [linearpmodel.cc](#)

## 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`.

```

14                                     {
15                                     return (new LinearPModel());
16                                     }

```

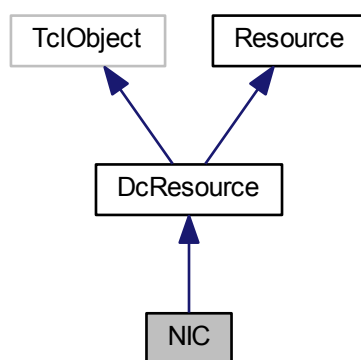
The documentation for this class was generated from the following file:

- [linearpmodel.cc](#)

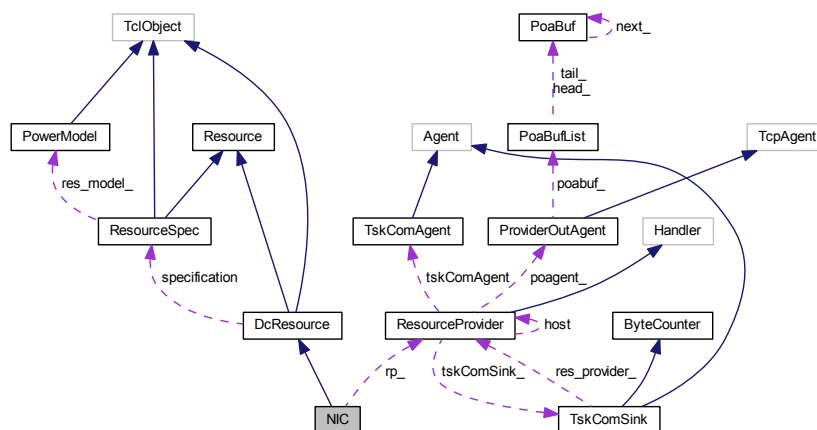
## 4.27 NIC Class Reference

```
#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](#).

```
19         {  
20  
21  
22     }
```

##### 4.27.2.2 [NIC::~~NIC](#) ( ) [virtual]

Definition at line 24 of file [nic.cc](#).

```
24         {  
25  
26     }
```

#### 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](#).

```
32         {  
33             return rp\_->getResTypeUtil(Networking);  
34     }
```

#### 4.27.3.2 void NIC::setRp ( ResourceProvider \* rp )

Definition at line 28 of file nic.cc.

```
28                                     {  
29         rp_ = rp;  
30 }
```

#### 4.27.4 Member Data Documentation

##### 4.27.4.1 ResourceProvider\* NIC::rp\_ [private]

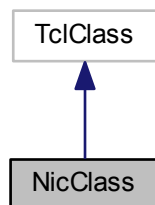
Definition at line 21 of file nic.h.

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

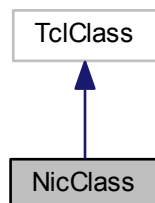
- [nic.h](#)
- [nic.cc](#)

#### 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 &amp; Destructor Documentation

4.28.2.1 `NicClass::NicClass ( )` `[inline]`

Definition at line 13 of file `nic.cc`.

```
13 : 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`.

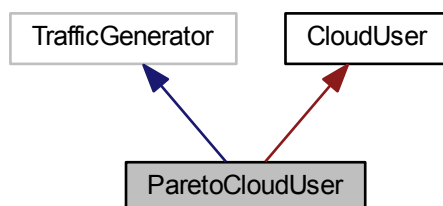
```
14                                     {
15                                     return (new NIC());
16                                     }
```

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

- [nic.cc](#)

## 4.29 ParetoCloudUser Class Reference

Inheritance diagram for `ParetoCloudUser`:





## Additional Inherited Members

## 4.29.1 Detailed Description

Definition at line 14 of file `paretoclouduser.cc`.

## 4.29.2 Constructor &amp; 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.

```

103 {
104
105     double t = interval_;
106
107     on_ = 1;
108     if (rem_ == 0) {
109         /* compute number of packets in next burst */
110         if (rng_ == 0) {
111             rem_ = int(Random::pareto(
112                 pl_, shape_) + .5);
113         }
114         else {
115             // Added by Debojyoti Dutta 13th October 2000
116             rem_ = int(rng_>pareto(
117                 pl_, shape_) + .5);
118         }
119         /* make sure we got at least 1 */
120         if (rem_ == 0)
121             rem_ = 1;
122         /* start of an idle period, compute idle time */
123         if (rng_ == 0) {
124             t += Random::pareto(p2_,
125                 shape_);
126         }
127         else {
128             // Added by Debojyoti Dutta 13th October 2000
129             t += rng_>pareto(p2_,
130                 shape_);
131         }
132         on_ = 0;
133     }
134     rem_--;
135     size = size_;
136     return(t);
137 }

```

#### 4.29.3.4 int ParetoCloudUser::on ( ) [inline]

Definition at line 19 of file paretoclouduser.cc.

```

19 { return on_ ; }

```

#### 4.29.3.5 void ParetoCloudUser::timeout ( ) [virtual]

Definition at line 137 of file paretoclouduser.cc.

```

138 {
139     if (! running_)
140         return;
141
142     /* send a packet */
143     dc->receivedTsk(size_, createTask());
144     /* figure out when to send the next one */
145     nextPkttime_ = next_interval(size_);
146     /* schedule it */
147     if (nextPkttime_ > 0)
148         timer_.resched(nextPkttime_);
149     else
150         running_ = 0;
151 }

```

#### 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`.

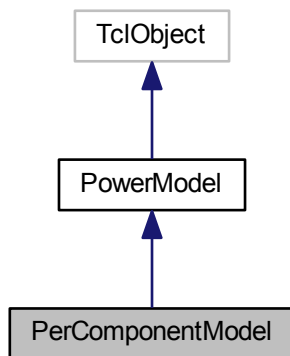
The documentation for this class was generated from the following file:

- [paretoclouduser.cc](#)

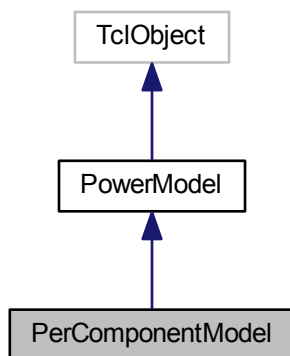
#### 4.30 PerComponentModel Class Reference

```
#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 &amp; Destructor Documentation

## 4.30.2.1 PerComponentModel::PerComponentModel ( )

Definition at line 19 of file `percomponentmodel.cc`.

```

19                                     {
20
21
22 }
```

## 4.30.2.2 PerComponentModel::~PerComponentModel ( ) [virtual]

Definition at line 24 of file `percomponentmodel.cc`.

```

24                                     {
25
26 }
```

## 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`.

```

28                                     {
29     modeled_components_.push_back(component);
30 }
```

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`.

```

32                                     {
33     if (argc == 2) {
34         if (strcmp(argv[1], "print") == 0) {
35             /* print general info */
36             print();
37             return (TCL_OK);
38         }
39     } else if (argc==3){
40         if (strcmp(argv[1], "set-name") == 0) {
41             this->setName(argv[2]);
42             return(TCL_OK);
43         } else {
44             return(TCL_ERROR);
45         }
46     }
47     return(TCL_ERROR);
48 }
49 }
```

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

Implements [PowerModel](#).

Definition at line 50 of file percomponentmodel.cc.

```

50                                     {
51         std::vector<DcResource*>::iterator iter;
52         double result = 0;
53         for(iter = modeled_components_.begin(); iter !=
modeled_components_.end(); iter++){
54             result+=(*iter)->getPower();
55         }
56         return result;
57     }
```

#### 4.30.3.4 double PerComponentModel::getMaxPower ( ) [virtual]

Implements [PowerModel](#).

Definition at line 59 of file percomponentmodel.cc.

```

59                                     {
60         std::vector<DcResource*>::iterator iter;
61         double result = 0;
62         for(iter = modeled_components_.begin(); iter !=
modeled_components_.end(); iter++){
63             result+=(*iter)->getMaxPower();
64         }
65         return result;
66     }
```

#### 4.30.3.5 void PerComponentModel::print ( ) [virtual]

Reimplemented from [PowerModel](#).

Definition at line 69 of file percomponentmodel.cc.

```

69                                     {
70         std::cout << "Per component power model.\nModeled components:\n";
71         std::vector<DcResource*>::iterator iter;
72         for(iter = modeled_components_.begin(); iter !=
modeled_components_.end(); iter++){
73             std::cout << (*iter)->specification->getPowerModel()->name_ << "\n";
74         }
75     }
76 }
```

### 4.30.4 Member Data Documentation

#### 4.30.4.1 std::vector<DcResource\*> PerComponentModel::modeled\_components\_

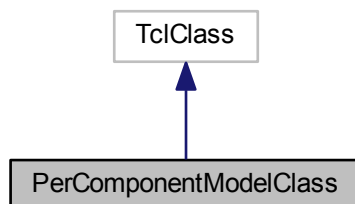
Definition at line 26 of file percomponentmodel.h.

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

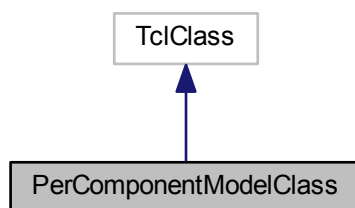
- [percomponentmodel.h](#)
- [percomponentmodel.cc](#)

## 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.

```

14
15
16
                                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.

```
64 {  
65     nbytes_ = nbytes;  
66     pointer_=c;  
67     next_ = NULL;  
68 }
```

##### 4.32.2.2 PoaBuf::~PoaBuf ( ) [inline]

Definition at line 20 of file provideroutagent.h.

```
20     {  
21 }
```

#### 4.32.3 Member Function Documentation

##### 4.32.3.1 int PoaBuf::bytes ( ) [inline]

Definition at line 23 of file provideroutagent.h.

```
23 { return nbytes_; }
```

##### 4.32.3.2 void\* PoaBuf::pointer ( ) [inline]

Definition at line 22 of file provideroutagent.h.

```
22 { 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`.

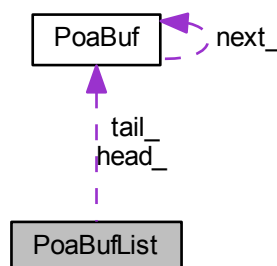
The documentation for this class was generated from the following files:

- [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 &amp; Destructor Documentation

## 4.33.2.1 PoaBufList::PoaBufList ( ) [inline]

Definition at line 38 of file provideroutagent.h.

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

## 4.33.2.2 PoaBufList::~PoaBufList ( )

Definition at line 70 of file provideroutagent.cc.

```
71 {
72     while (head_ != NULL) {
73         tail_ = head_;
74         head_ = head_>next_;
75         delete tail_;
76     }
77 }
```

## 4.33.3 Member Function Documentation

## 4.33.3.1 PoaBuf \* PoaBufList::detach ( )

Definition at line 89 of file provideroutagent.cc.

```
90 {
91     if (head_ == NULL)
92         return NULL;
93     PoaBuf *p = head_;
94     if ((head_ = head_>next_) == NULL)
95         tail_ = NULL;
96     return p;
97 }
```

## 4.33.3.2 void PoaBufList::insert ( PoaBuf \* poabuf )

Definition at line 79 of file provideroutagent.cc.

```
80 {
81     if (tail_ == NULL)
82         head_ = tail_ = poabuf;
83     else {
84         tail_>next_ = poabuf;
85         tail_ = poabuf;
86     }
87 }
```

## 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]

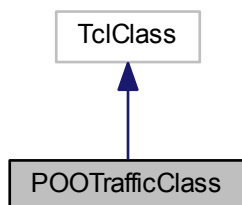
Definition at line 46 of file provideroutagent.h.

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

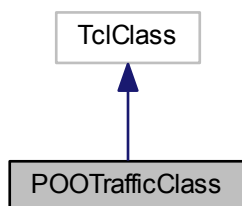
- [provideroutagent.h](#)
- [provideroutagent.cc](#)

### 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`.

```
46 : 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`.

```
47                                     {  
48                                     return (new ParetoCloudUser());  
49                                     }
```

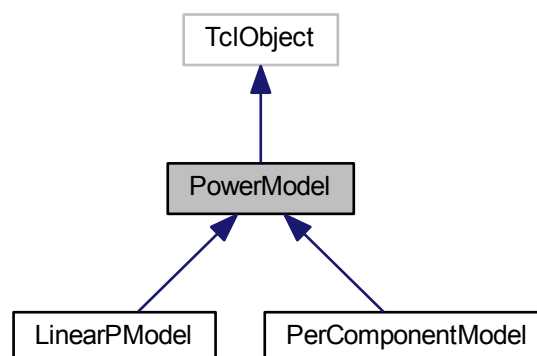
The documentation for this class was generated from the following file:

- [paretoclouduser.cc](#)

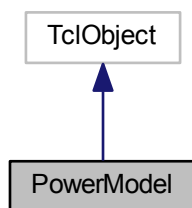
## 4.35 PowerModel Class Reference

```
#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`.

```

11         {
12
13     }
```

##### 4.35.2.2 PowerModel::~~PowerModel ( ) [virtual]

Definition at line 15 of file `powermodel.cc`.

```

15         {
16
17     }
```

## 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.

```

27 {
28
29     if (argc == 2) {
30         if (strcmp(argv[1], "print") == 0) {
31
32             /* print general info */
33             print();
34             return (TCL_OK);
35         }
36     }
37     return (PowerModel::command(argc, argv));
38 }
```

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.

```

22     {
23         std::cout << "Abstract power model";
24     }
```

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

Definition at line 19 of file powermodel.cc.

```

19     {
20         name_ = name;
21     }
```

#### 4.35.4 Member Data Documentation

##### 4.35.4.1 `std::string PowerModel::name_`

Definition at line 27 of file `powermodel.h`.

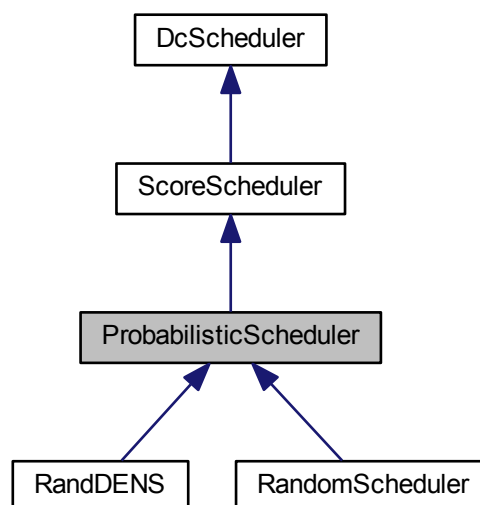
The documentation for this class was generated from the following files:

- [powermodel.h](#)
- [powermodel.cc](#)

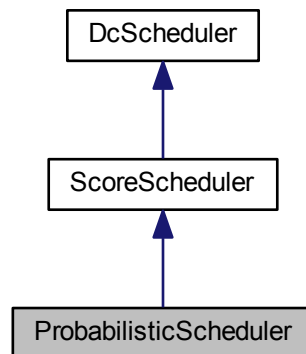
#### 4.36 ProbabilisticScheduler Class Reference

```
#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.

```

10                                     {
11
12
13 }
```

##### 4.36.2.2 ProbabilisticScheduler::~~ProbabilisticScheduler ( ) [virtual]

Definition at line 15 of file probabilisticscheduler.cc.

```

15                                     {
16
17 }
```

### 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.

```

19                                                                                                     {
20         //1. calculate mDENS score array
21         //2. generate random number in [0:summed mDENS score]
22         //3. binary search on the array
23         //4. return the selected host
24         TskComAgent* selected = NULL;
25         std::vector<double> mdens_score(providers.size());
26         std::vector<double> mdens_score_cumulative(providers.size());
27         vector<ResourceProvider*>::iterator res_p;
28         vector<double>::iterator score;
29         vector<double>::iterator cumul;
30         for (res_p = providers.begin(), score = mdens_score.begin(), cumul = mdens_score_cumulative
31             .begin();
32             res_p!=providers.end();
33             res_p++, score++, cumul++)
34         {
35             (*score) = calculateScore((*res_p));
36             if(cumul==mdens_score_cumulative.begin()){
37                 (*cumul) = (*score);
38             } else {
39                 (*cumul) = (*score) + (*(cumul-1));
40             }
41             std::cout << "score:" << (*score)<< " cumul: " << (*cumul) << "\n";
42             // srand(time(0));
43             double r = ( (double)rand() / (double)RAND_MAX ) * mdens_score_cumulative.at(
44                 mdens_score_cumulative.size()-1);
45             // std::cerr << "Random: " << r << "\n";
46             // vector<double>::iterator lb = lower_bound(mdens_score_cumulative.begin(),
47                 mdens_score_cumulative.end(),r);
48             int sel_ind = lb - mdens_score_cumulative.begin();
49             // std::cerr << "Lb: "<< (*lb) << " ind: " << sel_ind << "\n";
50             // abort();
51             selected = providers.at(sel_ind)->getTskComAgent();
52             return selected;
53         }

```

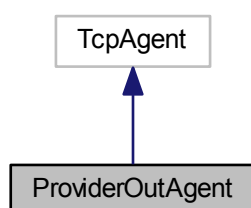
The documentation for this class was generated from the following files:

- [probabilisticscheduler.h](#)
- [probabilisticscheduler.cc](#)

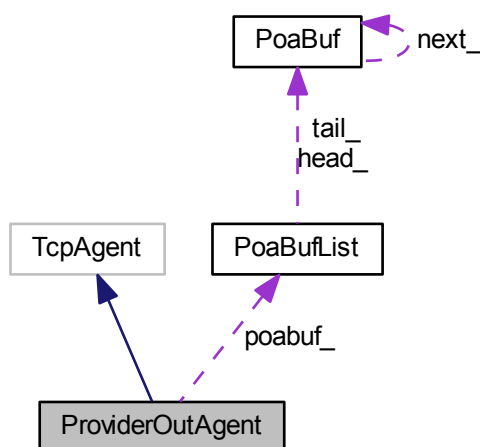
## 4.37 ProviderOutAgent Class Reference

```
#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.

```

19         : lastTrackedBytes_(0),
      lastTrackedTime_(0.0) {
20
21         lastTrackedTime_ = Scheduler::instance().clock();
22         lastTrackedBytes = this->getNdatabytes();
23     }
```

#### 4.37.2.2 ProviderOutAgent::~ProviderOutAgent ( ) [virtual]

Definition at line 25 of file provideroutagent.cc.

```

25         {
26         PoaBuf* tmp;
27         while(true) {
28             tmp = poabuf_.detach();
29             if(tmp!=NULL) {
30                 delete tmp;
31             } else {
32                 break;
33             }
34     }
```

### 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.

```

48         {
49         poabuf_.insert(new PoaBuf( pointer,nbytes));
50         tryToSend();
51     }
```

## 4.37.3.2 void ProviderOutAgent::tryToSend ( )

Definition at line 53 of file provideroutagent.cc.

```

53         {
54             if (current_pointer_==NULL) {
55                 PoaBuf* cb = poabuf_.detach();
56                 if (cb!=NULL) {
57                     TcpAgent::sendmsg (cb->bytes(), (void*) cb->
58                     pointer());
59                     delete cb;
60                 }
61     }
```

## 4.37.3.3 int ProviderOutAgent::updateAgentDataBytes ( )

Definition at line 36 of file provideroutagent.cc.

```

36         {
37             int result = getNdatabytes() - lastTrackedBytes_;
38             lastTrackedBytes_ = getNdatabytes();
39             return result;
40     }
```

## 4.37.3.4 double ProviderOutAgent::updateTime ( )

Definition at line 42 of file provideroutagent.cc.

```

42         {
43             double result = lastTrackedTime_;
44             lastTrackedTime_ = Scheduler::instance().clock();
45             return result;
46     }
```

## 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.

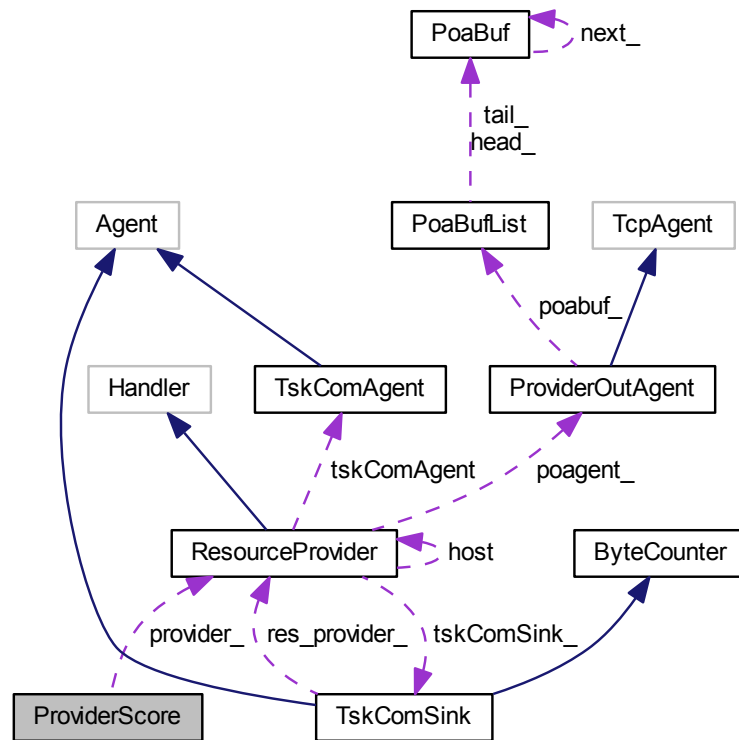
The documentation for this class was generated from the following files:

- [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.

```

10                                     : provider_(provider),
11     score_(score) {
12
13 }
```

#### 4.38.2.2 ProviderScore::ProviderScore ( ResourceProvider \* *provider*, double *score*, double *comm\_potential\_* )

Definition at line 15 of file providerscore.cc.

```

15                                     :
16     provider_(provider), score_(score), comm_potential_(
17     comm_potential_){
18 }
```

#### 4.38.2.3 ProviderScore::~ProviderScore ( ) [virtual]

Definition at line 22 of file providerscore.cc.

```

22                                     {
23
24 }
```

### 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.

```

26                                     {
27     return this->score_ < other.score_;
28 }
```

### 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\_

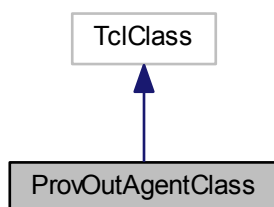
Definition at line 17 of file providerscore.h.

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

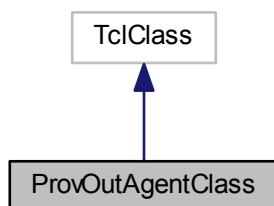
- [providerscore.h](#)
- [providerscore.cc](#)

### 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.

```
13 : 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.

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

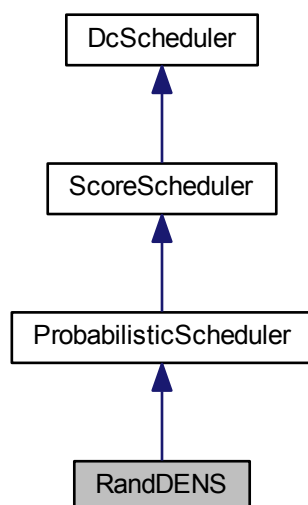
The documentation for this class was generated from the following file:

- [provideroutagent.cc](#)

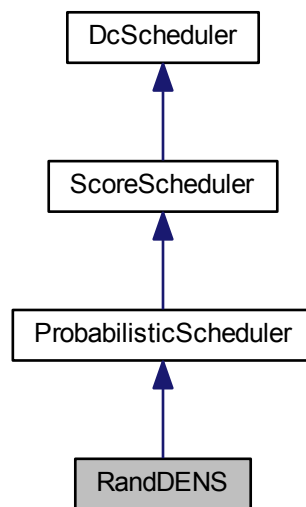
## 4.40 RandDENS Class Reference

```
#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 &amp; 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         {
25             double result = 0;
26             double load;
27             for(int i = FirstResType; i <= LastResType ; i++){
28                 load = rp->getResTypeUtil(static_cast<res_type>(i));
29                 result+= densLoadFactor(load,
30                 epsilon);
31             }
32             result=result/(LastResType+1); // normalize according to the number of
33             dimensions
34             result += linkLoadFactor(rp->getRootHost()->
35             rack_->link_load);
36
37             return result;
38         }
```

## 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-
39             epsilon/2)))));
40 }
```

## 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.

```
19                                     {
20     return ProbabilisticScheduler::scheduleTask(task,
21     providers);
21 }
```

#### 4.40.4 Member Data Documentation

4.40.4.1 double RandDENS::epsilon [private]

Definition at line 27 of file randdens.h.

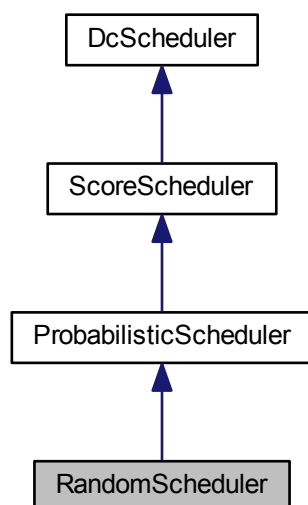
The documentation for this class was generated from the following files:

- [randdens.h](#)
- [randdens.cc](#)

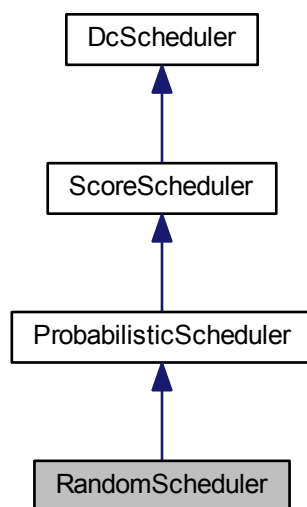
#### 4.41 RandomScheduler Class Reference

```
#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`.

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

#### 4.41.2.2 RandomScheduler::~RandomScheduler ( ) [virtual]

Definition at line 15 of file randomscheduler.cc.

```
15             {
16
17 }
```

#### 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.

```
23             {
24                 double result = 1;
25                 return result;
26 }
```

##### 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.

```
19             {
20                 return ProbabilisticScheduler::scheduleTask(task,
21                     providers);
21 }
```

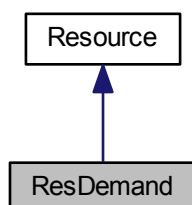
The documentation for this class was generated from the following files:

- [randomscheduler.h](#)
- [randomscheduler.cc](#)

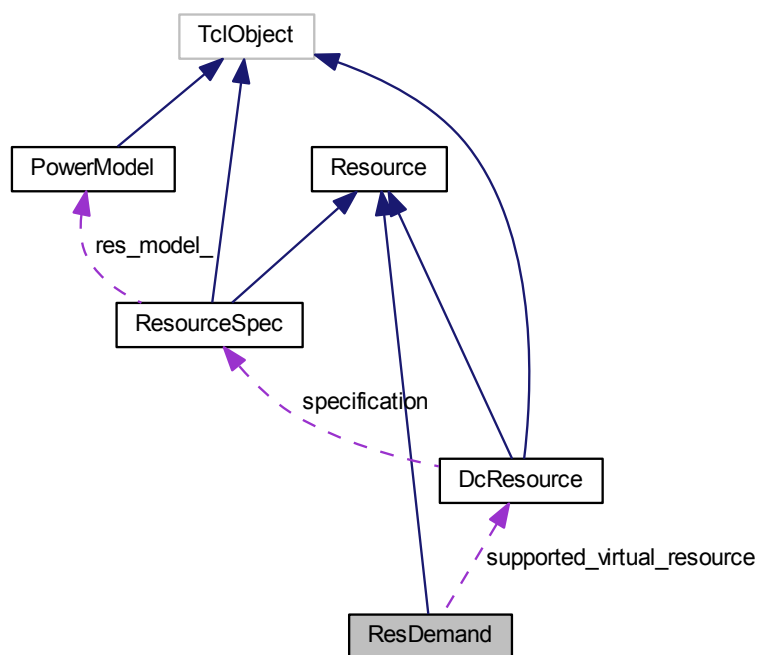
## 4.42 ResDemand Class Reference

```
#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`.

```

11             : Resource( res){
12     capacity_location = std::vector <Capacity *>(res.
    capacity.size(),NULL) ;
13     current_performance = std::vector <double>(res.
    capacity.size(),0);
14     supported_virtual_resource = NULL;
15 }
16 }
```

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

Definition at line 18 of file `resdemand.cc`.

```

18             : Resource( res){
19     capacity_location = std::vector <Capacity *>(res.
    capacity.size(),NULL) ;
20     current_performance = std::vector <double>(res.
    capacity.size(),0);
21     if(svr != NULL){
22         std::vector<Capacity>::iterator i_res;
23         std::vector<Capacity>::iterator i_dem;
24         for(i_res = svr->capacity.begin(), i_dem = this->
    capacity.begin();
25             i_res != svr->
    capacity.end();
26             i_res++, i_dem++){
27             i_dem->virtual_capacities.push_back(&(*i_res));
28         }
29     }
30     supported_virtual_resource = svr;
31 }
```

### 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`.

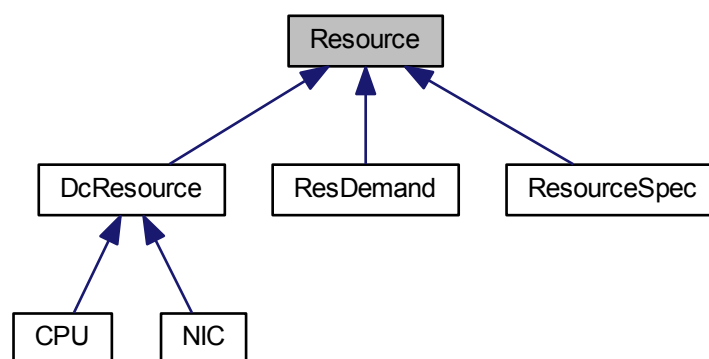
The documentation for this class was generated from the following files:

- [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.

```

74                                     {
75         type = t;
76         arch = a;
77         this->setCapacity (cap);
78 }
```

#### 4.43.2.2 Resource::~~Resource ( ) [virtual]

Definition at line 80 of file resource.cc.

```

80                                     {
81
82 }
```

#### 4.43.2.3 Resource::Resource ( ) [protected]

Definition at line 70 of file resource.cc.

```

70                                     {
71
72 }
```

## 4.43.3 Member Function Documentation

## 4.43.3.1 double Resource::getArch ( ) [inline]

Definition at line 90 of file resource.h.

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

## 4.43.3.2 res\_type Resource::getType ( ) [inline]

Definition at line 89 of file resource.h.

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

## 4.43.3.3 Resource &amp; Resource::operator= ( const Resource &amp; r ) [protected]

Definition at line 119 of file resource.cc.

```
119                                     {
120         if (this != &r) { // make sure it is not the same object
121             capacity.clear();
122             capacity = r.capacity;
123             arch = r.arch;
124             type = r.type;
125         }
126         return *this; // Return ref for multiple assignment
127 }
```

## 4.43.3.4 void Resource::print ( )

Definition at line 129 of file resource.cc.

```
129         {
130             std::cerr << "Type:\t"<< type;
131             std::cerr << "\n";
132             std::cerr << "Architecture:\t"<< arch;
133             std::cerr << "\n";
134             std::cerr << "Capacities:\t";
135             std::vector <Capacity>::iterator iter;
136             for (iter = capacity.begin(); iter!=capacity.end(); iter++)
137             {
138                 std::cerr << (*iter) << ",";
139             }
140
141 }
```

## 4.43.3.5 int Resource::setCapacity ( std::vector&lt; Capacity &gt; cap ) [protected]

Definition at line 110 of file resource.cc.

```
110                                     {
111         capacity = cap;
112         return 0;
113 }
```

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

Definition at line 103 of file `resource.cc`.

```
103             {
104         type = translateType(t);
105         return 0;
106 }
```

#### 4.43.3.7 `void Resource::sortCapacity ( )` [protected]

Definition at line 115 of file `resource.cc`.

```
115             {
116         std::sort(capacity.begin(), capacity.end());
117 }
```

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

Definition at line 85 of file `resource.cc`.

```
85             {
86         res_type type;
87         if(strcmp(t, "Computing") == 0){
88             type=Computing;
89         } else if(strcmp(t, "Memory") == 0){
90             type=Memory;
91         } else if(strcmp(t, "Storage") == 0){
92             type=Storage;
93         } else if(strcmp(t, "Networking") == 0){
94             type=Networking;
95         } else {
96             std::cerr << "Unknown resource type" << t;
97             abort();
98         }
99         return type;
100     }
101 }
```

### 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`.

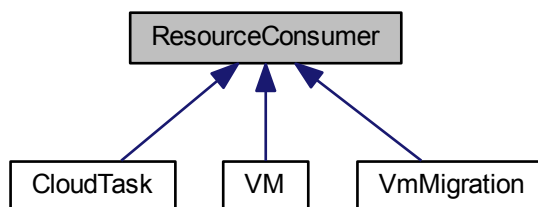
The documentation for this class was generated from the following files:

- [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 &amp; Destructor Documentation

## 4.44.2.1 ResourceConsumer::ResourceConsumer ( )

Definition at line 10 of file resourceconsumer.cc.

```

10         {
11             res\_demands.clear();
12     }
```

#### 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.

```

14      isTask(isTask), isVM(isVM), size_(size) {
15
16          res_demands.clear();
17          res_demands = std::vector<ResDemand *>(demand.size(),NULL);
18          std::vector<Resource*>::iterator iter;
19          std::vector<ResDemand*>::iterator iter2;
20          for (iter = demand.begin(), iter2=res_demands.begin(); iter!=demand.end(); iter+
21              +,iter2++)
22              {
23                  (*iter2)=new ResDemand(*(*iter));
24              }
25          for (iter = demand.begin() ; iter!=demand.end(); iter++)
26              {
27                  delete (*iter);
28              }
29      }
```

#### 4.44.2.3 ResourceConsumer::~~ResourceConsumer ( ) [virtual]

Definition at line 31 of file resourceconsumer.cc.

```

31      {
32          std::vector<ResDemand*>::iterator iter2;
33          for (iter2 = res_demands.begin() ; iter2!=res_demands.end(); iter2++)
34              {
35                  delete (*iter2);
36              }
37      }
```

### 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.

```

40 {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.

```

43 {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`.

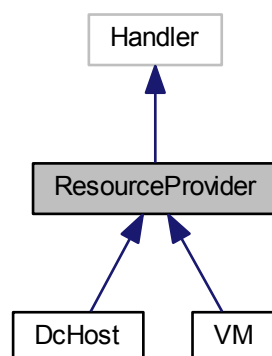
The documentation for this class was generated from the following files:

- [resourceconsumer.h](#)
- [resourceconsumer.cc](#)

## 4.45 ResourceProvider Class Reference

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

Inheritance diagram for ResourceProvider:





- 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`.

```
91 { 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`.

```
18                                     : id_(0), ntasks_(0),
19                                     currentLoad_(0.0), currentLoadMem_(0.0),
   currentLoadStor_(0.0), currentLoadNet_(0.0),
20                                     eDvfs_enabled_(0.0), tskFailed_(0),
   tskComAgent(NULL), host(NULL), started_(false)
21 {
22
23     for(int i = 0; i <= LastResType; i++){
24         resource_list.push_back(std::vector<DcResource*>());
25     }
26     for(int i = 0; i <= LastResType; i++){
27         resource_utilization[i] = 0.0;
28     }
29     hosted_vms_.clear();
30     status_ = EVENT_IDLE;
31     poagent_ = NULL;
32 }
```

## 4.45.3.2 ResourceProvider::~ResourceProvider ( ) [virtual]

Definition at line 34 of file resourceprovider.cc.

```

34         {
35             std::vector<std::vector<DcResource*> >::iterator iter;
36             for(iter = resource_list.begin(); iter!=
resource_list.end() ;iter++){
37                 std::vector<DcResource*>::iterator iter2;
38                 for(iter2 = iter->begin(); iter2!=iter->end() ;iter2++){
39                     delete (*iter2);
40                 }
41             }
42             delete poagent_;
43         }
44     }
```

## 4.45.4 Member Function Documentation

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

Definition at line 520 of file resourceprovider.cc.

```

520         {
521             (void)Scheduler::instance().cancel(&event_);
522             // no need to free event_ since it's statically allocated
523     }
```

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

Definition at line 517 of file resourceprovider.cc.

```

517         {
518             (void)Scheduler::instance().schedule(this, &event_, delay);
519     }
```

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

Reimplemented in [DcHost](#), and [VM](#).

Definition at line 498 of file resourceprovider.cc.

```

498         {
499
500             resource_list[res->getType()].push_back(res);
501             if(res->getType()==Computing){
502                 CPU* cpu_res = static_cast<CPU*>(res);
503                 cpu_res->setDVFS(eDVFS_enabled_);
504                 cpu_res->setProvider(this);
505             }
506             if(res->getType()==Networking){
507                 NIC* nic_res = static_cast<NIC*>(res);
508                 nic_res->setRp(this);
509             }
510
511     }
```

#### 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     } else if (argc == 3) {
592         if (strcmp(argv[1], "attach-agent") == 0) {
593             setAgent((
594 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<
603 TskComAgent*> (TclObject::lookup(argv[2]));
604             if (agnt) {
605                 setTskComAgent (agnt);
606                 return (TCL_OK);
607             }
608             return (TCL_ERROR);
609         }
610         else if (strcmp(argv[1], "attach-vm-mig-sink") == 0) {
611             VmMigrationSink* vm_mig_sink = (

```

```

VmMigrationSink*) TclObject::lookup(argv[2]));
611         attachSink(vm_mig_sink);
612         if (getAgent() == 0) {
613             tcl.resultf("no such agent %s", argv[2]);
614             return(TCL_ERROR);
615         }
616         return(TCL_OK);
617     } else
618     if (strcmp(argv[1], "attach-vm-mig-source") == 0) {
        ProviderOutAgent* vm_migration_source = (
VmMigrationSink*) TclObject::lookup(argv[2]);
619         attachSource(vm_migration_source);
620         if (getAgent() == 0) {
621             tcl.resultf("no such agent %s", argv[2]);
622             return(TCL_ERROR);
623         }
624         return(TCL_OK);
625     } else
626     if (strcmp(argv[1], "detach-vm-mig-sink") == 0) {
        VmMigrationSink* vm_mig_sink = (
VmMigrationSink*) TclObject::lookup(argv[2]);
627         detachSink(vm_mig_sink);
628         if (getAgent() == 0) {
629             tcl.resultf("no such agent %s", argv[2]);
630             return(TCL_ERROR);
631         }
632         return(TCL_OK);
633     } else
634     if (strcmp(argv[1], "detach-vm-mig-source") == 0) {
        ProviderOutAgent* vm_migration_source = (
ProviderOutAgent*) TclObject::lookup(argv[2]);
635         detachSource(vm_migration_source);
636         if (getAgent() == 0) {
637             tcl.resultf("no such agent %s", argv[2]);
638             return(TCL_ERROR);
639         }
640         return(TCL_OK);
641     } else if (strcmp(argv[1], "add-resource") == 0) {
        DcResource* res = (
DcResource*) TclObject::lookup(argv[2]);
643         if (res == NULL) {
644             tcl.resultf("no such resource %s", argv[2]);
645             return(TCL_ERROR);
646         }
647         addResource(res);
648         return(TCL_OK);
649     } else if (strcmp(argv[1], "add-vm") == 0) {
        VM* vm = (VM*) TclObject::lookup(argv[2]);
650         if (vm == NULL) {
651             tcl.resultf("no such vm %s", argv[2]);
652             return(TCL_ERROR);
653         }
654         if(addVM(vm)) {
655             return(TCL_OK);
656         } else {
657             /* It was impossible to allocate vm on the
658             host.*/
659             std::cerr << "ERROR: A VM was allocated on
a machine that has not enough resources. (Creation was called from Tcl)";
660             return(TCL_ERROR);
661         }
662     }
663 }
664 return (TCL_ERROR);
665 }

```

#### 4.45.4.8 void ResourceProvider::detachSink ( VmMigrationSink \* vm\_mig\_sink ) [protected]

Definition at line 574 of file resourceprovider.cc.

```

574         {
575             vm_migration_sinks_.erase(remove(
vm_migration_sinks_.begin(), vm_migration_sinks_.end(), vm_mig_sink),
vm_migration_sinks_.end()); /*
576             erase-remove idiom*/
577 }

```

#### 4.45.4.9 void ResourceProvider::detachSource ( ProviderOutAgent \* tcp\_agent ) [protected]

Definition at line 580 of file resourceprovider.cc.

```

580                                     {
581         vm_migration_sources_.erase(remove(
vm_migration_sources_.begin(), vm_migration_sources_.end(), poa),
582                                     vm_migration_sources_.end()); /*
erase-remove idiom*/
583 }
```

#### 4.45.4.10 TcpAgent \* ResourceProvider::getAgent ( ) [protected]

Definition at line 534 of file resourceprovider.cc.

```

534                                     {
535         return poagent_;
536
537 }
```

#### 4.45.4.11 double ResourceProvider::getCurrentLoad ( ) [protected]

Definition at line 414 of file resourceprovider.cc.

```

415 {
416         double nominal_mips = 0;
417         double current_mips = 0;
418         std::vector<DcResource*>::iterator cpu_iter;
419         for(cpu_iter=resource_list[Computing].begin(); cpu_iter !=
resource_list[Computing].end(); cpu_iter++){
420                 DcResource* res = *cpu_iter;
421                 CPU* cpu = (CPU*) res;
422                 nominal_mips += cpu->getNominalMIPS();
423                 current_mips += cpu->getCurrentMIPS();
424         }
425         currentLoad_ = current_mips/nominal_mips;
426
427         return currentLoad_;
428 }
```

#### 4.45.4.12 double ResourceProvider::getFreeCap ( res\_type type ) [protected]

Definition at line 308 of file resourceprovider.cc.

```

308                                     {
309         double free_cap = 0;
310         std::vector<DcResource*>::iterator dc_res;
311         for(dc_res=resource_list[type].begin(); dc_res !=
resource_list[type].end(); dc_res++){
312                 std::vector<Capacity*>::iterator free_cap_iter;
313                 for(free_cap_iter = (*dc_res)->capacity.begin();
314                 free_cap_iter != (*dc_res)->capacity.end();
315                 free_cap_iter++){
316                         free_cap += *free_cap_iter;
317                 }
318         }
319
320         return free_cap;
321 }
```

## 4.45.4.13 double ResourceProvider::getFreeCapRecursive ( res\_type type ) [protected]

Definition at line 298 of file resourceprovider.cc.

```

298                                     {
299         double free_cap = getFreeCap(type);
300
301         std::vector<ResourceConsumer*>::iterator vm_iter;
302         for(vm_iter=hosted_vms_.begin(); vm_iter !=
hosted_vms_.end(); vm_iter++){
303             VM* vm = static_cast<VM*>(*vm_iter);
304             free_cap += vm->getFreeCapRecursive(type);
305         }
306         return free_cap;
307     }

```

## 4.45.4.14 ResourceProvider \* ResourceProvider::getHost ( )

Definition at line 184 of file resourceprovider.cc.

```

184                                     {
185         return host;
186     }

```

## 4.45.4.15 double ResourceProvider::getResTypeUtil ( res\_type type )

Definition at line 379 of file resourceprovider.cc.

```

379                                     {
380         if(type==Networking){
381             return resource_utilization[
Networking];
382         } else {
383             return updateResTypeUtil(type);
384         }
385     }

```

## 4.45.4.16 DcHost \* ResourceProvider::getRootHost ( )

Definition at line 189 of file resourceprovider.cc.

```

189                                     {
190         if(host == NULL){
191             DcHost* root = static_cast<DcHost*>(this);
192             return root;
193         } else {
194             return host->getRootHost();
195         }
196
197     }

```

## 4.45.4.17 double ResourceProvider::getTotalCap ( res\_type type )

Definition at line 363 of file resourceprovider.cc.

```

363                                     {
364         double total_cap = 0;
365         std::vector<DcResource*>::iterator dc_res;
366         for(dc_res=resource_list[type].begin(); dc_res !=
resource_list[type].end(); dc_res++){
367             std::vector<Capacity*>::iterator total_cap_iter;
368             for(total_cap_iter = (*dc_res)->specification->capacity.begin();
total_cap_iter != (*dc_res)->specification
->capacity.end();
369                                     total_cap_iter++){
370                 total_cap += *total_cap_iter;
371             }
372         }
373
374         return total_cap;
375     }
376
377 }

```

#### 4.45.4.18 TskComAgent \* ResourceProvider::getTskComAgent ( )

Definition at line 543 of file resourceprovider.cc.

```

543                                     {
544         return this->tskComAgent;
545     }

```

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

Definition at line 323 of file resourceprovider.cc.

```

323                                     {
324         double result = 0;
325         if(in){
326             double elapsed_time = Scheduler::instance().clock() - this->
tskComSink_->getLastBytesSinceTime();
327             if(elapsed_time>0){
328                 double down_link_util = this->tskComSink_->
resetBytesSince();
329                 std::vector<VmMigrationSink*>::iterator vms;
330                 for(vms = vm_migration_sinks_.begin();vms!=
vm_migration_sinks_.end();vms++){
331                     double recent_bytes = (*vms)->resetBytesSince();
332                     down_link_util += recent_bytes;
333                 }
334                 result += (down_link_util/elapsed_time);
335             }
336             if(out){
337                 double elapsed_time = Scheduler::instance().clock() - this->
poagent_->updateTime();
338                 if(elapsed_time>0){
339                     double up_link_util = this->poagent_->
updateAgentDataBytes();
340                     std::vector<ProviderOutAgent*>::iterator poa;
341                     for(poa = vm_migration_sources_.begin();poa!=
vm_migration_sources_.end();poa++){
342                         double recent_bytes = (*poa)->updateAgentDataBytes();
343                         up_link_util += recent_bytes;
344                     }
345                     result += ((up_link_util*
ResourceProvider::uplink_overhead)/elapsed_time);
346                 }
347             }
348             return result;
349         }
350     }

```

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

Definition at line 352 of file resourceprovider.cc.

```

352                                     {
353         double used_net = getUsedNet(in,out);
354         std::vector<ResourceConsumer*>::iterator vm_iter;
355         for(vm_iter=hosted_vms_.begin(); vm_iter !=
hosted_vms_.end(); vm_iter++){
356             VM* vm = static_cast<VM*>(*vm_iter);
357             used_net += vm->getUsedNetRecursive(in,out);
358         }
359         return used_net;
360     }
361 }

```

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

Definition at line 284 of file resourceprovider.cc.

```

285 {
286     std::vector <CoreScheduler*>::iterator core_s;
287     std::vector <DcResource*>::iterator cpu_iter;
288     for(cpu_iter=resource_list[Computing].begin(); cpu_iter !=
resource_list[Computing].end(); cpu_iter++){
289         CPU* cpu = (CPU*) (*cpu_iter);
290         for(core_s=cpu->cores_schedulers_.begin(); core_s != cpu->
cores_schedulers_.end(); core_s++){
291             (*core_s)->updateTskList();
292         }
293     }
294 }
295 }
296 }
```

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

Definition at line 270 of file resourceprovider.cc.

```

271 {
272     if (status_ == EVENT_PENDING) {
273         _cancel();
274         status_ = EVENT_IDLE;
275     }
276
277     event_.handler_ = this;
278     event_.time_ = Scheduler::instance().clock();
279
280     _sched(delay);
281     status_ = EVENT_PENDING;
282 }
```

**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.

```

526     {
527         std::cout << "printTasklist Status: (FUNCTION UNDER CONSTRUCTION)\n";
528     }
529 }
```

#### 4.45.4.25 void ResourceProvider::recv ( ResourceConsumer \* rcobj )

Definition at line 228 of file resourceprovider.cc.

```

229 {
230     this->updateEnergyAndConsumption();
231     if(rcobj->isTask==true){
232         vector<CloudTask*>::iterator iter;
233         CloudTask* tskobj = (CloudTask*) rcobj;
234
235         ntasks_ ++;                                     // update
236         total number of the received tasks
237
238         if(tskobj->scheduled_==false){
239             if(trySchedulingTsk(tskobj)==false){
240                 tskobj->fail(this);
241             }
242             std::cout << "Unscheduled task failed due to insufficient resources";
243             return;
244         }
245         std::vector <CoreScheduler*>::iterator core_s;
246         std::vector <DcResource*>::iterator cpu_iter;
247
248         /*If it is possible to allocate:*/
249         if(tryToAllocate(tskobj)){
250             for(cpu_iter=resource_list[
251                 Computing].begin(); cpu_iter != resource_list[Computing].end(); cpu_iter++){
252                 CPU* cpu = (
253                     CPU*) (*cpu_iter);
254                 for(core_s=cpu->
255                     cores_schedulers_.begin(); core_s != cpu->cores_schedulers_.end(); core_s
256                     ++){
257                     (*core_s)->
258                     startTaskExecution(tskobj);
259                 }
260             }
261             } else {
262                 /*Otherwise task fails!*/
263                 tskobj->fail(this);
264                 std::cout << "Task failed due to insufficient resources";
265                 return;
266             }
267         } else {
268             std::cerr <<"It is not a task!";
269             return;
270         }
271     }
272 }
```

#### 4.45.4.26 bool ResourceProvider::releaseAllocation ( ResourceConsumer \* rc )

Definition at line 137 of file resourceprovider.cc.

```

137 {
138     if((*rc).res_demands.empty()){
139         std::cerr << "Nothing to release \n";
140         return true;
141     }
142     // std::cerr << "Something to release \n";
143     std::vector <ResDemand*>::iterator u_res;
144     for (u_res = rc->res_demands.begin() ; u_res!=rc->
145         res_demands.end(); u_res++)
146     {
147         if(((*u_res)->getType() !=Computing && (*u_res)->getType() !=
148         Networking ) || rc->isTask == false){
149             std::vector <Capacity*>::iterator consumption;
150             std::vector <Capacity *>::iterator location;
151             for(consumption=(*u_res)->capacity.begin(),
152                 location=(*u_res)->
153                 capacity_location.begin();
154                 consumption!=(*u_res)->
155                 capacity.end();
156                 consumption++,location++){
157                 if ((*location)==NULL){
158                     } else {
159                         **location = (**location)+

```

```

    (*consumption);
157
158                                     }
159                                     }
160                                     }
161                                     if(rc->isTask==false){
162                                     if((*u_res)->supported_virtual_resource != NULL){
163                                     if(                                     (*u_res)->
supported_virtual_resource->getType() == Computing) {
164
CPU* cpu = (CPU*) (*u_res)->supported_virtual_resource;
165                                     std::vector<CoreScheduler*
>::iterator cs;
166                                     for(cs = cpu->
cores_schedulers_.begin(); cs != cpu->cores_schedulers_.end(); cs++){
167
CoreScheduler* host_cs = (*cs)->getHostScheduler();
168                                     if(host_cs
!=NULL) {
169
host_cs->removeVcoreScheduler(*cs);
170                                     }
171                                     }
172                                     }
173                                     }
174                                     }
175                                     }
176                                     }
177                                     if(rc->isTask==false){
178                                     // non-task (VM or migration) specific cleanup. Handled in the respective
classes.
179                                     }
180
return true;
181
182 }

```

#### 4.45.4.27 bool ResourceProvider::removeVM ( VM \* vm )

Definition at line 212 of file resourceprovider.cc.

```

212                                     {
213                                     this->updateEnergyAndConsumption();
214                                     if(releaseAllocation(vm)){
215                                     (vm)->setHost(NULL);
216                                     hosted_vms_.erase(remove(hosted_vms_.begin(),
hosted_vms_.end(), vm),
217                                     hosted_vms_.end()); /*
erase-remove idiom*/
218                                     return true;
219                                     } else {
220                                     return false;
221                                     }
222 }

```

#### 4.45.4.28 void ResourceProvider::scheduleNextExnt ( double nextDeadline )

Definition at line 558 of file resourceprovider.cc.

```

558                                     {
559                                     /* reschedule next update */
560                                     if (nextDeadline != DBL_MAX) nextEvent(nextDeadline);
561
562
563 }

```

#### 4.45.4.29 void ResourceProvider::sendTaskOutput ( CloudTask \* task )

Definition at line 547 of file resourceprovider.cc.

```

547                                     {
548                                     if ((getAgent()) && (task->getOutput() != 0)) {
549                                     /*Record finish time of task on the server.*/
550                                     task->info_->setServerFinishTime(
Scheduler::instance().clock());
551                                     task->info_->setResourceProvider(this);
552                                     /*Send task output.*/
553                                     poagent_->sendmsg(task->getOutput(), task);
554                                     }
555 }

```

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

Definition at line 530 of file resourceprovider.cc.

```
530                                     {
531         poagent_ = agent;
532 }
```

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

Definition at line 539 of file resourceprovider.cc.

```
539                                     {
540         this->tskComAgent = agnt;
541 }
```

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

Definition at line 513 of file resourceprovider.cc.

```
513                                     {
514         this->tskComSink_ = tcs;
515 }
```

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

Definition at line 430 of file resourceprovider.cc.

```
430                                     {
431         int result = trySchedulingTsk(tskobj);
432         if(result){
433                 releaseAllocation(tskobj);
434                 tskobj->releaseAllTaskAllocs();
435                 tskobj->scheduled_=false;
436         }
437         return result;
438 }
```

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

Definition at line 440 of file resourceprovider.cc.

```
441 {
442
443         /* get minimum processing rate required by the task */
444         if(tryToAllocate(tskobj)){
445                 releaseAllocation(tskobj);
446         } else {
447                 return false;
448         }
449
450         std::vector<TaskAlloc> tmp_task_allocs;
451         tmp_task_allocs.clear();
452
453         std::vector<TaskAlloc>::iterator iter;
454         for(iter = tskobj->task_allocations_.begin() ; iter != tskobj->
task_allocations_.end(); iter++){
455                 TaskAlloc* task_alloc = (*iter);
456                 bool core_found = false;
457                 tmp_task_allocs.push_back(task_alloc);
458 }
```

```

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 =
484             //     if(core_of_failed!=NULL){
485             //         core_of_failed->removeFromAssginedList((*failed_alloc));
486             //     }
487             //     return false;
488         }
489     }
490     tskobj->scheduled_=true;
491     return true;
492 }
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();

```

```

66         }
67
68         std::vector <DcResource*>::iterator p_res;
69         std::vector <CoreScheduler*>::iterator p_core;
70         /*For each DcResource of provider:*/
71         for(p_res = resource_list[(*u_res)->getType()]
.begin(); p_res!=resource_list[(*u_res)->getType()].end() ;p_res++){
72
73
74         /*Check architecture*/
75         if((*u_res)->getArch() <= (*p_res)->getArch
76
77         if((*u_res)->capacity.empty
78
79         possible =
80
81         std::vector
82         if((*u_res)->getType()==
83         p_core = ((
84         /*Case of empty capacity
85
86         for(; aval_cap_prov!=
87         if((*
88
89         Computing && (*u_res)->supported_virtual_resource){
90
91         CPU*)(*p_res))->cores_schedulers_.begin();
92
93         }
94         /*Demands capacities are linked with the supported resources capacities (1 to 1),
95         * so the intermediary ResDemand is neglected:*/
96         if(rc->isVM){
97         (*aval_cap_prov).virtual_capacities.push_back(req_cap_cons->virtual_capacities.at(0));
98         }
99         (*loc_cap_cons)=&(*aval_cap_prov);
100         if((*u_res)->getType()==Computing && (*u_res)->supported_virtual_resource){
101         (*p_core)->addVcoreScheduler((*u_core));
102         }
103         req_cap_cons++;
104         loc_cap_cons++;
105         if(req_cap_cons== (*u_res)->capacity.end()){
106         possible = true;
107         break;
108         }
109         if((*u_res)->getType()==Computing && (*u_res)->supported_virtual_resource){
110         u_core++;
111         }
112         } else {
113
114         }
115
116         }
117         /*Resource architecture rejected:*/
118         else {
119         //std::cerr << "Arch,

```

```

    Requested: " << (*u_res)->getArch() << "\tProvided: " <<(*p_res)->getArch() << "\n";
120     }
121     if(possible == true){
122         /*Break the main loop of
    scanning the provider resources.*/
123         break;
124     }
125 }
126 if(possible==false){
127     //std::cerr << "Impossible to allocate
    here.\n";
128 }
129 releaseAllocation(rc);
130     return false;
131 }
132 }
133 //std::cerr << "Allocation success.\n";
134 return true;
135 }

```

#### 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.

```

387     {
388         if(type==Computing){
389             resource_utilization[
    Computing] = getCurrentLoad();
390             return resource_utilization[
    Computing];
391         } else if(type == Networking){
392             double total_cap = getTotalCap(type) * 2; // Bidirectional links
393             double used_net_bytes = getUsedNetRecursive(true,true);
394             double result = used_net_bytes/total_cap;
395             resource_utilization[
    Networking] =currentLoadNet_ = result;
396             return resource_utilization[
    Networking];
397         } else {
398             double total_cap = getTotalCap(type);
399             if(total_cap==0){
400                 return 0; //There is no components of this resource type
401             }
402             double free_cap = getFreeCapRecursive(type);
403             double result = 1 - (free_cap/total_cap);
404             if(type == Memory){
405                 currentLoadMem_ = result;
406             } else if(type==Storage){
407                 currentLoadStor_ = result;
408             }
409             resource_utilization[type] = result;
410             return result;
411         }
412     }

```

### 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.

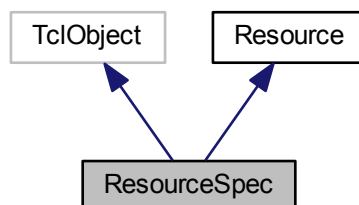
The documentation for this class was generated from the following files:

- [resourceprovider.h](#)
- [resourceprovider.cc](#)

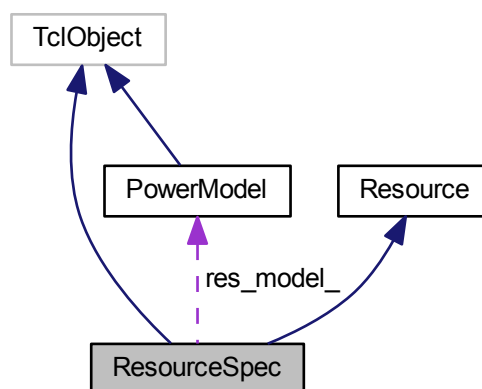
#### 4.46 ResourceSpec Class Reference

```
#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`.

```
19             :   name_("NA"), res_model_(NULL) {
20                 //          std::cerr << ("Resource Spec Constructor.\n");
21                 capacity.clear();
22                 power_states.clear();
23 }
```

##### 4.46.2.2 ResourceSpec::~ResourceSpec ( ) [virtual]

Definition at line 25 of file `resourcespec.cc`.

```
25             {
26                 name_.clear();
27 }
```

### 4.46.3 Member Function Documentation

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

Definition at line 29 of file resourcespec.cc.

```

29                                     {
30         capacity.push_back(cap);
31         return 0;
32     }
```

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

Definition at line 33 of file resourcespec.cc.

```

33                                     {
34         power_states.push_back(ps);
35         return 0;
36     }
```

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

Definition at line 74 of file resourcespec.cc.

```

74                                     {
75
76
77         if (argc == 2) {
78             if (strcmp(argv[1], "print") == 0) {
79                 print();
80                 return (TCL_OK);
81             }
82         } else if (argc == 3) {
83             if (strcmp(argv[1], "add-capacity") == 0) {
84                 addCapacity(atof(argv[2]));
85                 return (TCL_OK);
86             } else if (strcmp(argv[1], "add-power-state") == 0) {
87                 addPowerState(atoi(argv[2]));
88                 return (TCL_OK);
89             } else if (strcmp(argv[1], "set-type") == 0) {
90                 if (Resource::setType(argv[2]) == 0) {
91                     return (TCL_OK);
92                 } else {
93                     return (TCL_ERROR);
94                 }
95             } else if (strcmp(argv[1], "set-name") == 0) {
96                 setName(argv[2]);
97                 return (TCL_OK);
98             } else if (strcmp(argv[1], "set-arch") == 0) {
99                 setArch(argv[2]);
100                 return (TCL_OK);
101             } else if (strcmp(argv[1], "set-power-model") == 0) {
102                 PowerModel* pm = (
103                     PowerModel*) TclObject::lookup(argv[2]);
104                 this->setPowerModel(pm);
105                 return (TCL_OK);
106             }
107         }
108         return (ResourceSpec::command(argc, argv));
109     }
```

#### 4.46.3.4 PowerModel \* ResourceSpec::getPowerModel ( )

Definition at line 53 of file resourcespec.cc.

```

53                                     {
54         return res_model_;
55     }
```

## 4.46.3.5 void ResourceSpec::print ( )

Definition at line 57 of file resourcespec.cc.

```

57         {
58             std::cerr << "ResourceSpec:\t";
59             std::cerr << name_;
60             std::cerr << "\n";
61             Resource::print();
62             std::vector<int>::iterator iter;
63             std::cerr << "Power states:\t";
64             for (iter = power_states.begin(); iter!=power_states.end(); iter++)
65             {
66                 std::cerr << (*iter) << ", ";
67             }
68             std::cerr << "\n";
69         }
70     }
```

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

Definition at line 44 of file resourcespec.cc.

```

44                                     {
45             arch = atof(name);
46             return 0;
47     }
```

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

Definition at line 39 of file resourcespec.cc.

```

39                                     {
40             name_=name;
41             return 0;
42     }
```

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

Definition at line 49 of file resourcespec.cc.

```

49                                     {
50             res_model_ = model;
51     }
```

## 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]`

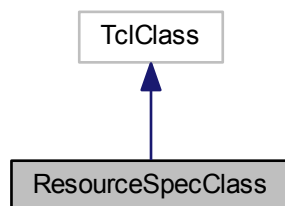
Definition at line 47 of file `resourcespec.h`.

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

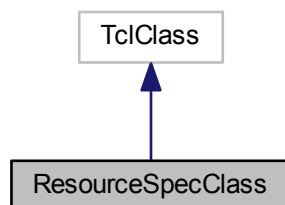
- [resourcespec.h](#)
- [resourcespec.cc](#)

### 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 &amp; 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                                     }
```

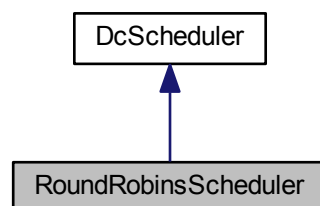
The documentation for this class was generated from the following file:

- [resourcespec.cc](#)

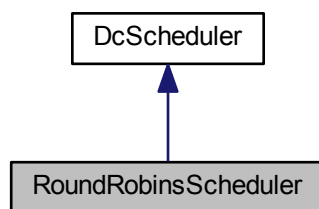
## 4.48 RoundRobinsScheduler Class Reference

```
#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.

```

10         {
11
12
13     }
```

##### 4.48.2.2 RoundRobinsScheduler::~RoundRobinsScheduler ( ) [virtual]

Definition at line 15 of file roundrobinscheduler.cc.

```

15         {
16
17     }
```

## 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.

```

19                                     {
20         int j = task->id_ % providers.size();
21
22         return (providers.at(j)->getTskComAgent());
23     }
```

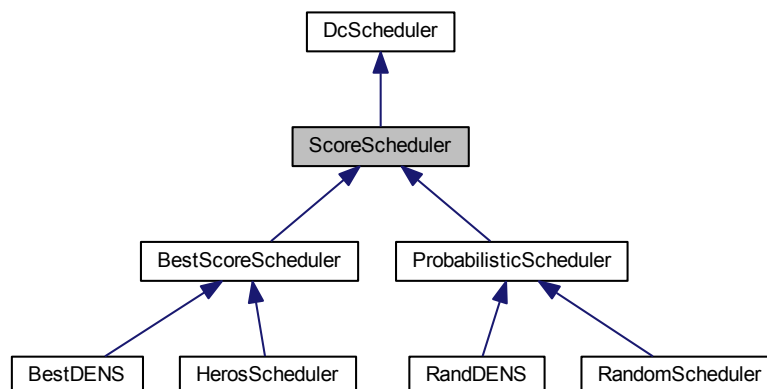
The documentation for this class was generated from the following files:

- [roundrobinscheduler.h](#)
- [roundrobinscheduler.cc](#)

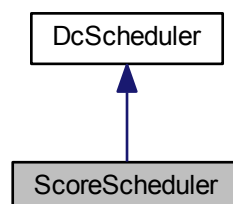
## 4.49 ScoreScheduler Class Reference

```
#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 [BestScoreScheduler](#).

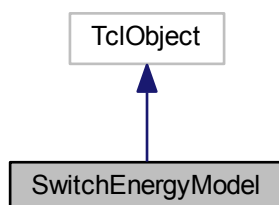
The documentation for this class was generated from the following files:

- [scorescheduler.h](#)
- [scorescheduler.cc](#)

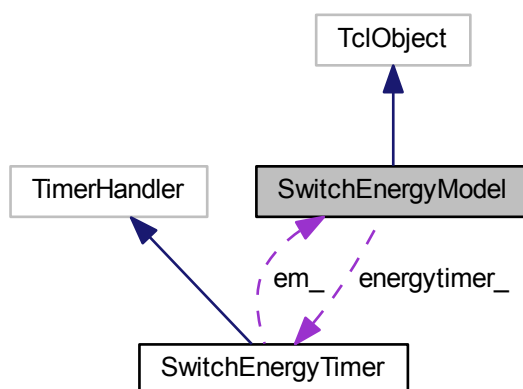
## 4.50 SwitchEnergyModel Class Reference

```
#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`.

```

16                                     : 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)
17 {
18     bind("eConsumed_", &eConsumed_);
19     bind("eChassis_", &eChassis_);
20     bind("eLineCard_", &eLineCard_);
21     bind("ePort_", &ePort_);
22     bind("eSimEnd_", &eSimEnd_);
23     bind("eDVFS_enabled_", &eDVFS_enabled_);
/* ON when DVFS is enabled */
24     bind("eDNS_enabled_", &eDNS_enabled_);
/* ON when DNS is enabled */
25     bind("eDNS_delay_", &eDNS_delay_);
26 }
```

## 4.50.2.2 SwitchEnergyModel::~SwitchEnergyModel ( ) [virtual]

Definition at line 28 of file switchenergymodel.cc.

```
29 {
30 }
```

## 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.

```
73 {
74     if (argc == 2) {
75         if (strcmp(argv[1], "start") == 0) {
76             start();
77             return (TCL_OK);
78         }
79         if (strcmp(argv[1], "stop") == 0) {
80             stop();
81             return (TCL_OK);
82         }
83     }
84     return (SwitchEnergyModel::command(argc, argv));
85 }
```

## 4.50.3.2 double SwitchEnergyModel::computeCurrentRate ( ) [protected]

Definition at line 49 of file switchenergymodel.cc.

```
50 {
51     eCurrentRate_ = eChassis_ + eLineCard_ +
    eActivePorts_*ePort_;
52     return eCurrentRate_;
53 }
54 }
```

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

Definition at line 32 of file switchenergymodel.h.

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

## 4.50.3.4 void SwitchEnergyModel::start ( )

Definition at line 32 of file switchenergymodel.cc.

```
33 {
34     eEnabled_ = 1;
35     eLastSample_ = Scheduler::instance().clock();
36     eSimDuration_ = eSimEnd_ - eLastSample_;
37     if (classifier_) eActivePorts_ =
    classifier_>maxslot();
38     if (eDNS_enabled_) eCurrentRate_ = 0.0;
39     else computeCurrentRate();
40 }
41 }
```

#### 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;
64         // update number of active ports
65         computeCurrentRate();
66         eLastSample_ = Scheduler::instance().clock();
67
68         /* if DNS is enabled start sleep-mode timer */
69         if ((eDNS_enabled_) && (eDNS_delay_))
energytimer_.resched(eDNS_delay_);
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\_**

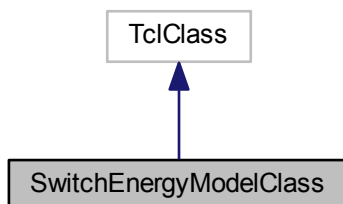
Definition at line 44 of file switchenergymodel.h.

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

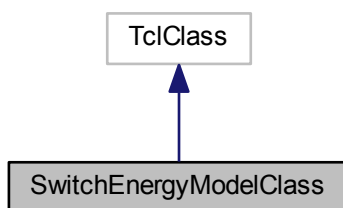
- [switchenergymodel.h](#)
- [switchenergymodel.cc](#)

## 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.

```
11  
12  
13      }  
      return (new SwitchEnergyModel());  
      {
```

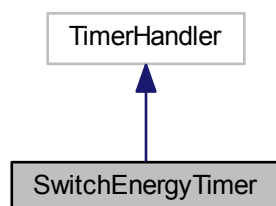
The documentation for this class was generated from the following file:

- [switchenergymodel.cc](#)

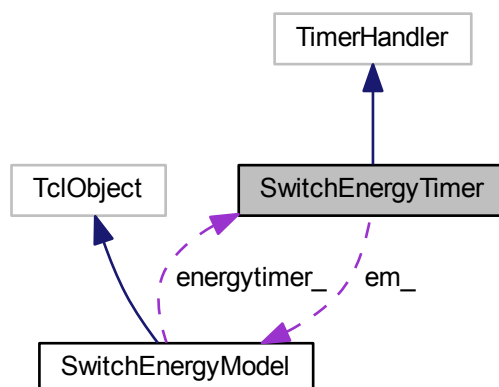
## 4.52 SwitchEnergyTimer Class Reference

```
#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`.

```
19 : 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`.

```
95 {  
96     em\_->timeout();  
97 }
```

### 4.52.4 Member Data Documentation

#### 4.52.4.1 `SwitchEnergyModel* SwitchEnergyTimer::em_` `[protected]`

Definition at line 22 of file `switchenergymodel.h`.

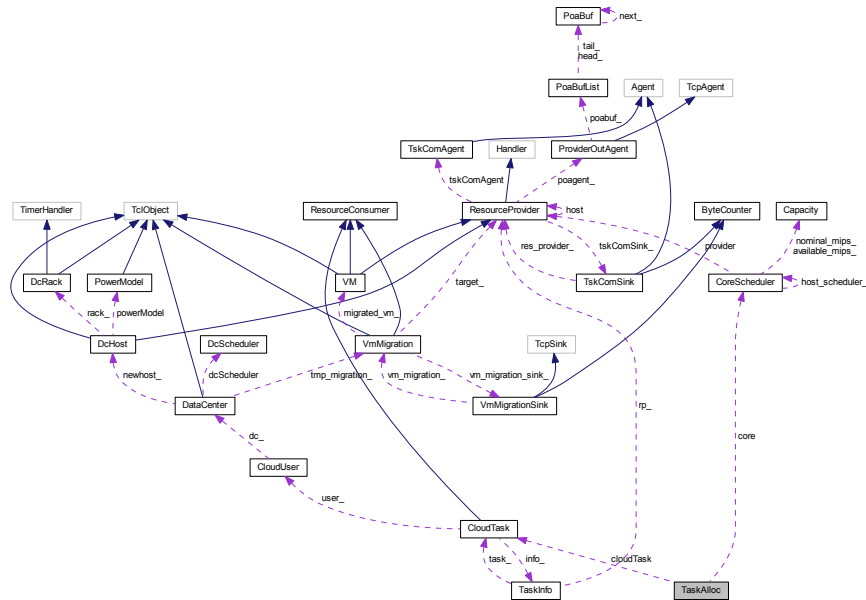
The documentation for this class was generated from the following files:

- [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.

```

24         {
25         return cloudTask->getDeadline();
26     }
```

## 4.53.3.4 double TaskAlloc::getMIPS ( )

Definition at line 21 of file taskalloc.cc.

```

21         {
22         return cloudTask->getMIPS(rd, cap);
23     }
```

## 4.53.3.5 bool TaskAlloc::operator== ( const TaskAlloc &amp; other ) const

Definition at line 35 of file taskalloc.cc.

```

35         {
36         if(cloudTask==other.cloudTask && rd == other.
rd && cap == other.cap){
37             return true;
38         } else {
39             return false;
40         }
41     }
```

## 4.53.3.6 void TaskAlloc::print ( )

Definition at line 50 of file taskalloc.cc.

```

50         {
51         std::cerr << "id "<< cloudTask->id_ << " "<< rd << " ,c: " <<
cap << " mips left: " << getMIPS() << "\texec since: " <<
executedSince_;
52     }
```

## 4.53.3.7 void TaskAlloc::removeAfterFailure ( )

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

Definition at line 43 of file taskalloc.cc.

```

44 {
45     /* update what has already been computed */
46     updateMIPS();
47     cloudTask->res_demands.at(rd)->current_performance.at(
cap)= rate;
48 }
```

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

Definition at line 28 of file taskalloc.cc.

```
28                                     {
29         this->core=cs;
30 }
```

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

Definition at line 28 of file taskalloc.h.

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

#### 4.53.3.11 void TaskAlloc::updateMIPS ( )

Definition at line 54 of file taskalloc.cc.

```
55 {
56     ResDemand* res_dem = cloudTask->res_demands.at(
57         rd);
58     double operationsComputed = (res_dem->current_performance.at(
59         cap)*(Scheduler::instance().clock() - executedSince_);
60     if((res_dem->capacity.at(cap) > operationsComputed){
61         (res_dem->capacity.at(cap) -= operationsComputed;
62     } else {
63         (res_dem->capacity.at(cap) = 0;
64     }
65     // std::cout << "--\t Task: " << this->cloudTask->id_ << " MIPS: " << getMIPS() << "\n";
66
67     executedSince_ = Scheduler::instance().clock();
68 }
69 }
```

### 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`.

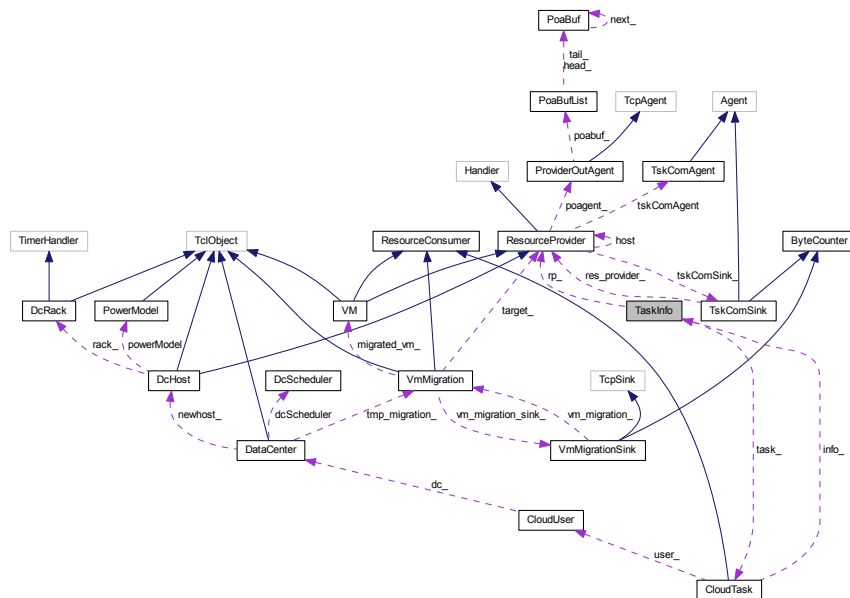
The documentation for this class was generated from the following files:

- [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.

```
45                                     {  
46         dc_exit_time_ = time;  
47     }
```

#### 4.54.3.3 double TaskInfo::getDcExitTime ( )

Definition at line 36 of file taskinfo.cc.

```
36         {  
37         return dc_exit_time_;  
38     }
```

#### 4.54.3.4 double TaskInfo::getDueTime ( )

Definition at line 30 of file taskinfo.cc.

```
30         {  
31         return due_time_;  
32     }
```

#### 4.54.3.5 double TaskInfo::getReleaseTime ( )

Definition at line 27 of file taskinfo.cc.

```
27         {  
28         return release_time_;  
29     }
```

#### 4.54.3.6 ResourceProvider \* TaskInfo::getResourceProvider ( )

Definition at line 49 of file taskinfo.cc.

```
49                                     {  
50         return rp_;  
51     }
```

#### 4.54.3.7 double TaskInfo::getServerFinishTime ( )

Definition at line 33 of file taskinfo.cc.

```
33         {  
34         return server_finish_time_;  
35     }
```

#### 4.54.3.8 CloudTask \* TaskInfo::getTask ( )

Definition at line 20 of file taskinfo.cc.

```
20         {
21             return task_;
22     }
```

#### 4.54.3.9 int TaskInfo::getTaskId ( )

Definition at line 39 of file taskinfo.cc.

```
39         {
40             return task_id_;
41     }
```

#### 4.54.3.10 void TaskInfo::setResourceProvider ( ResourceProvider \* rp )

Definition at line 53 of file taskinfo.cc.

```
53         {
54             rp_ = rp;
55     }
```

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

Definition at line 42 of file taskinfo.cc.

```
42         {
43             server_finish_time_ = time;
44     }
```

### 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.

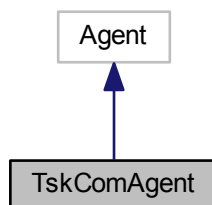
The documentation for this class was generated from the following files:

- [taskinfo.h](#)
- [taskinfo.cc](#)

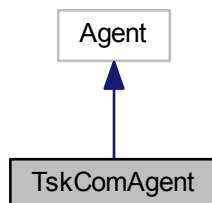
## 4.55 TskComAgent Class Reference

```
#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.

```

29             : Agent (PT_UDP), seqno\_(-1)
30 {
31     bind("packetSize_", &size_);
32 }
```

#### 4.55.2.2 TskComAgent::TskComAgent ( packet\_t type )

Definition at line 34 of file tskagent.cc.

```

34             : Agent (type)
35 {
36     bind("packetSize_", &size_);
37 }
```

### 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.

```

127 {
128     //          Tcl& tcl = Tcl::instance();
129     //          if (argc == 4) {
130     //              if (strcmp(argv[1], "send") == 0) {
131     //                  PacketData* data = new PacketData(1 +
132     //                      strlen(argv[3]));
133     //                  strcpy((char*)data->data(), argv[3]);
134     //                  sendmsg(atoi(argv[2]), data, 0);
135     //                  return (TCL_OK);
136     //              } else if (argc == 5) {
137     //                  if (strcmp(argv[1], "sendmsg") == 0) {
138     //                      PacketData* data = new PacketData(1 +
139     //                          strlen(argv[3]));
140     //                      strcpy((char*)data->data(), argv[3]);
141     //                      sendmsg(atoi(argv[2]), data, 0, argv[4]);
142     //                      return (TCL_OK);
143     //                  }
144     //              }
145     //          }
146     return (Agent::command(argc, argv));
147 }
```

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

Definition at line 100 of file tskagent.cc.

```

101 {
102     if (app_ ) {
103         // If an application is attached, pass the data to the app
104         hdr_cmh* h = hdr_cmh::access(pkt);
105         app_>process_data(h->size(), pkt->userdata());
106     } else if (pkt->userdata() && pkt->userdata()->type() == PACKET_DATA) {
107         // otherwise if it's just PacketData, pass it to Tcl
108         //
109         // Note that a Tcl procedure Agent/Udp recv {from data}
110         // needs to be defined. For example,
111         //
112         // Agent/Udp instproc recv {from data} {puts data}
113
114         PacketData* data = (PacketData*)pkt->userdata();
115
116         hdr_ip* iph = hdr_ip::access(pkt);
117         Tcl& tcl = Tcl::instance();
118         tcl.evalf("%s process_data %d {%s}", name(),
119                 iph->src_.addr_ >> Address::instance().
120         NodeShift_[1],
121                 data->data());
122     }
123     Packet::free(pkt);
124 }
```

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.

```

22     {
23         sendmsg(nbytes, NULL, pTaskObj, flags);
24     }
```

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.

```

40 {
41     Packet *p;
42     int n;
43
44     assert (size_ > 0);
45
46     n = nbytes / size_;
47     int initialseqno = seqno_;
48
49     if (nbytes == -1) {
50         printf("Error: sendmsg() for Tsk should not be -1\n");
51         return;
52     }
53
54     // If they are sending data, then it must fit within a single packet.
55     if (data && nbytes > size_) {
56         printf("Error: data greater than maximum Tsk packet size\n");
57         return;
58     }
59
60     double local_time = Scheduler::instance().clock();
61     while (n-- > 0) {
62         p = allocpkt();
63         hdr_cmh::access(p)->size() = size_;
64         hdr_cmh::access(p)->pt_obj_addr() = 0;
65         if (initialseqno == seqno_) {
66             /* Add pointer to TaskObj for the first packet in the bulk
67             */
68             hdr_cmh::access(p)->pt_obj_addr() = pTaskObj;
69         }
70     }
```

```

68             }
69             hdr_rtp* rh = hdr_rtp::access(p);
70             rh->flags() = 0;
71             rh->seqno() = ++seqno_;
72             hdr_cmh::access(p)->timestamp() =
73             (u_int32_t) (
SAMPLERATE*local_time);
74             if (flags && (0 == strcmp(flags, "NEW_BURST")))
75                 rh->flags() |= RTP_M;
76             p->setdata(data);
77             target_->recv(p);
78         }
79         n = nbytes % size_;
80         if (n > 0) {
81             p = allocpkt();
82             hdr_cmh::access(p)->size() = n;
83             hdr_cmh::access(p)->pt_obj_addr() = 0;
84             if (initialseqno == seqno_) {
85                 /* Add pointer to TaskObj for the first packet in the bulk
86
87                 hdr_cmh::access(p)->pt_obj_addr() = pTaskObj;
88             }
89             hdr_rtp* rh = hdr_rtp::access(p);
90             rh->flags() = 0;
91             rh->seqno() = ++seqno_;
92             hdr_cmh::access(p)->timestamp() =
93             (u_int32_t) (
SAMPLERATE*local_time);
94             if (flags && (0 == strcmp(flags, "NEW_BURST")))
95                 rh->flags() |= RTP_M;
96             p->setdata(data);
97             target_->recv(p);
98         }
99     }

```

#### 4.55.4 Member Data Documentation

##### 4.55.4.1 int TskComAgent::seqno\_ [protected]

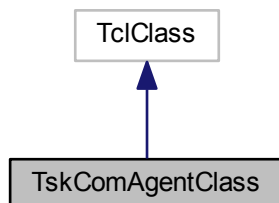
Definition at line 29 of file tskagent.h.

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

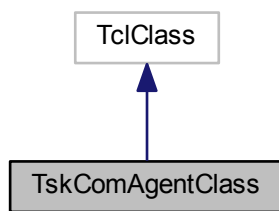
- [tskagent.h](#)
- [tskagent.cc](#)

## 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`.

```
23 : 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`.

```
24                                     {  
25                                     return (new TskComAgent ());  
26                                     }
```

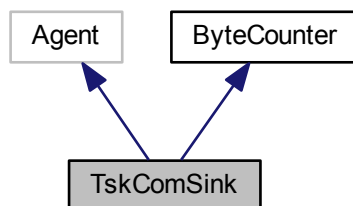
The documentation for this class was generated from the following file:

- [tskagent.cc](#)

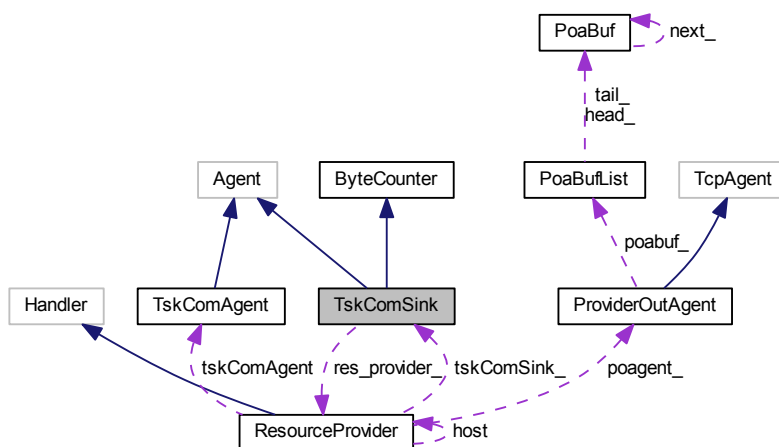
#### 4.57 TskComSink Class Reference

```
#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 &amp; Destructor Documentation

## 4.57.2.1 TskComSink::TskComSink ( )

Definition at line 28 of file `tskcomsink.cc`.

```

28             : Agent(PT_NTTYPE)
29 {
30     bytes_ = 0;
31     bytes_since_ = 0;
32     nlost_ = 0;
33     npkts_ = 0;
34     expected_ = -1;
35     last_packet_time_ = 0.;
36     last_bytes_since_ = 0.;
37     seqno_ = 0;
38     bind("nlost_", &nlost_);
39     bind("npkts_", &npkts_);
40     bind("bytes_", &bytes_);
41     bind("lastPktTime_", &last_packet_time_);
42     bind("expected_", &expected_);
43 }
```

## 4.57.2.2 TskComSink::~TskComSink ( ) [virtual]

Definition at line 45 of file `tskcomsink.cc`.

```

46 {
47     res_provider_ = NULL;
48 }
```

## 4.57.3 Member Function Documentation

## 4.57.3.1 void TskComSink::addResourceProvider ( ResourceProvider \* newrp )

Definition at line 70 of file `tskcomsink.cc`.

```

71 {
72     res_provider_ = newrp;
73     res_provider_->setTskComSink(this);
74 }
```

#### 4.57.3.2 int TskComSink::command ( int argc, const char \*const \* argv ) [virtual]

Definition at line 80 of file tskcomsink.cc.

```

81 {
82     if (argc == 2) {
83         if (strcmp(argv[1], "clear") == 0) {
84             expected_ = -1;
85             return (TCL_OK);
86         }
87     }
88     if (argc == 3) {
89         if (strcmp(argv[1], "connect-resprovider") == 0) {
90             ResourceProvider *hst = dynamic_cast<
ResourceProvider*> (TclObject::lookup(argv[2]));
91             if(hst) {
92                 addResourceProvider(hst);
93                 return (TCL_OK);
94             }
95             return (TCL_ERROR);
96         }
97     }
98     return (Agent::command(argc, argv));
99 }
100 }
```

#### 4.57.3.3 void TskComSink::recv ( Packet \* pkt, Handler \* ) [virtual]

Definition at line 50 of file tskcomsink.cc.

```

51 {
52     /* Get TskObject and start its execution */
53     ResourceConsumer *recvTskObj = (ResourceConsumer*)
hdr_cm::access(pkt)->pt_obj_addr();
54 //     std::cerr << "Pointer recieved:" << recvTskObj << "\n";
55
56     if (recvTskObj) { /* Valid pointer and can be executed */
57 //         std::cerr << "Task id:" << ((CloudTask*)recvTskObj)->id_ << "\n";
58         if (res_provider_ res_provider_->
recv(recvTskObj);
59         else printf("Error: task is received but no ResourceProvider is attached\n"
);
60     }
61
62     bytes_ += hdr_cm::access(pkt)->size();
63     bytes_since_ += hdr_cm::access(pkt)->size();
64     ++npkts_;
65
66     last_packet_time_ = Scheduler::instance().clock();
67     Packet::free(pkt);
68 }
```

### 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]`

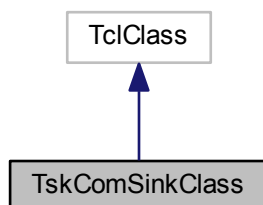
Definition at line 32 of file `tskcomsink.h`.

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

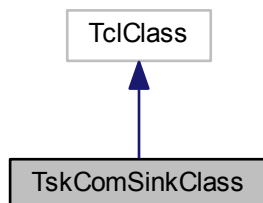
- [tskcomsink.h](#)
- [tskcomsink.cc](#)

## 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`.

```
22 : 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`.

```
23                                     {
24         return (new TskComSink());
25     }
```

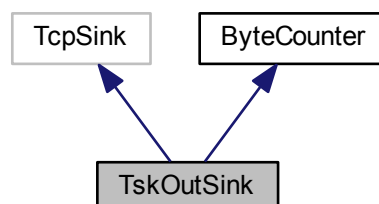
The documentation for this class was generated from the following file:

- [tskcomsink.cc](#)

## 4.59 TskOutSink Class Reference

```
#include <tskoutsink.h>
```

Inheritance diagram for TskOutSink:





#### 4.59.2.2 TskOutSink::~TskOutSink ( ) [virtual]

Definition at line 35 of file tskoutsink.cc.

```
36 {
37     res_provider_ = NULL;
38 }
```

### 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.

```
55 {
56     if (argc == 2) {
57         if (strcmp(argv[1], "clear") == 0) {
58             expected_ = -1;
59             return (TCL_OK);
60         }
61     }
62     if (argc == 3) {
63         if (strcmp(argv[1], "connect-tskoutagent") == 0) {
64             ProviderOutAgent *poa = (
ProviderOutAgent*) (TclObject::lookup(argv[2]));
65             if (poa) {
66                 poa_ = poa;
67                 return (TCL_OK);
68             }
69             return (TCL_ERROR);
70         }
71     }
72     return (Agent::command(argc, argv));
73 }
74 }
```

#### 4.59.3.3 void TskOutSink::recv ( Packet \* pkt, Handler \* h ) [virtual]

Definition at line 40 of file tskoutsink.cc.

```
41 {
42     /* Get TskObject and start its execution */
43     CloudTask *recvTskObj = (CloudTask*)hdr_cmh::access(pkt)->pt_obj_addr();
44     if (recvTskObj) { /* Valid pointer and can be executed */
45         // Scheduler::instance().clock() << "\n";
46         Scheduler::instance().clock();
47         poa->tryToSend();
48         recvTskObj->info->getResourceProvider()->
getRootHost()->updateEnergyAndConsumption();
49     }
50     TcpSink::recv(pkt, h);
51 }
```

### 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]

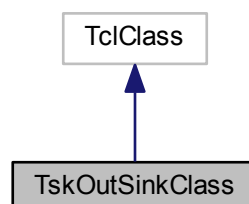
Definition at line 33 of file tskoutsink.h.

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

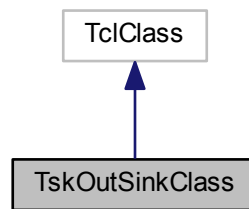
- [tskoutsink.h](#)
- [tskoutsink.cc](#)

## 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.

```
25                                     {
26                                     return (new TskOutSink());
27                                     }
```

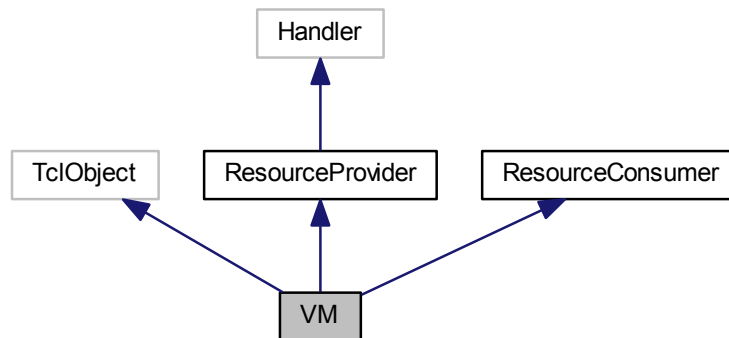
The documentation for this class was generated from the following file:

- [tskoutsink.cc](#)

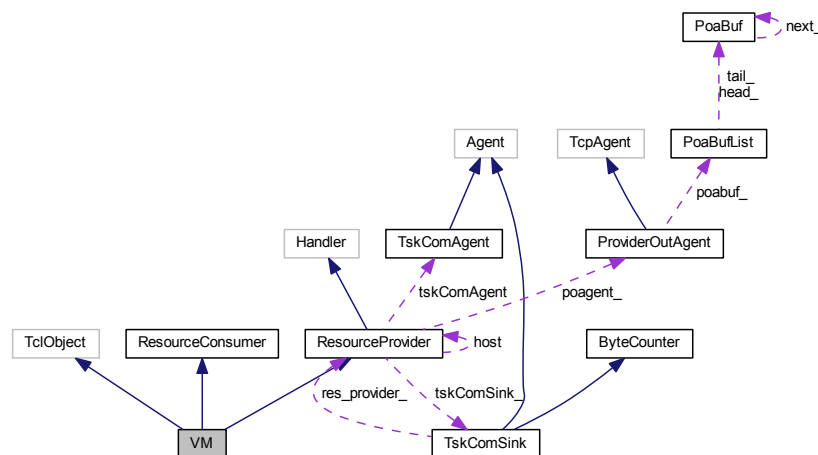
## 4.61 VM Class Reference

```
#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.

```

18     {
19
20         /* It should be always false for VMs */
21         isTask = false;
22         /* It should be always true for VMs */
23         isVM = true;
24
25         bind("id_", &id_);
26         bind("ntasks_", &ntasks_);
27         bind("currentLoad_", &currentLoad_);
28         bind("currentLoadMem_", &currentLoadMem_);
29         bind("currentLoadStor_", &currentLoadStor_);
30         bind("tskFailed_", &tskFailed_);
31         bind("eDVFS_enabled_", &eDVFS_enabled_);
32         /* ON when DVFS is enabled */
33         state = Ready;
34     }
```

#### 4.61.2.2 VM::~VM ( ) [virtual]

Definition at line 36 of file vm.cc.

```

36     {
37
38     }
```

### 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.

```

49     {
50
51         ResourceProvider::addResource(res);
52         res_demands.push_back(new ResDemand(*res, res));
53
54     }
```

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.

```

57 {
58
59     if (argc == 2) {
60         if (strcmp(argv[1], "start") == 0) {
61
62             state = Running;
63             return (TCL_OK);
64         } else if (strcmp(argv[1], "stop") == 0) {
65
66             state = Stopped;
67             return (TCL_OK);
68         } else if (strcmp(argv[1], "print") == 0) {
69             /* print general info */
70             print();
71             return (TCL_OK);
72         }
73     }
74     return (ResourceProvider::command(argc, argv));
75 }

```

4.61.3.3 `vm_state VM::getVmState ( )`

Definition at line 41 of file vm.cc.

```

41 {return state;};

```

4.61.3.4 `void VM::print ( )` [virtual]

Implements [ResourceProvider](#).

Definition at line 90 of file vm.cc.

```

90 {
91     std::cout << "VM:\t";
92     std::cout << id_;
93     std::cout << "\n";
94     if (host != NULL) {
95         std::cout << "Hosted on" << host->id_;
96     } else {
97         std::cout << "Not hosted";
98     }
99     std::cout << "\n";
100     std::cout << "Resources provisions:\n";
101     std::vector<std::vector<DcResource*>>::iterator iter_out;
102     for (iter_out = resource_list.begin(); iter_out !=
resource_list.end(); iter_out++) {
103         std::vector<DcResource*>::iterator iter;
104         for (iter = iter_out->begin(); iter != iter_out->end(); iter++)
105             {
106                 (*iter)->print();
107             }
108     }
109     std::cout << "Resources demands:\n";
110     std::vector<ResDemand*>::iterator iter_dem;
111     for (iter_dem = res_demands.begin(); iter_dem !=
res_demands.end(); iter_dem++) {
112         (*iter_dem)->print();
113     }
114     std::cout << "\n";
115 }
116 }

```

**4.61.3.5 void VM::printTasklist ( ) [virtual]**

Reimplemented from [ResourceProvider](#).

Definition at line 118 of file vm.cc.

```

118         {
119             std::vector<CloudTask *>::iterator iter;
120             std::cout <<"VM " <<this->id_ << "\n";
121
122             ResourceProvider::printTasklist();
123     }
```

**4.61.3.6 void VM::setHost ( ResourceProvider \* newHost )**

Definition at line 43 of file vm.cc.

```

43 {host = newHost;;}
```

**4.61.3.7 void VM::updateEnergyAndConsumption ( ) [protected],[virtual]**

Implements [ResourceProvider](#).

Definition at line 81 of file vm.cc.

```

81         {
82             if(host!= NULL) {
83                 host->updateEnergyAndConsumption();
84             } else {
85                 std::cerr << "ERROR: Task is allocated on an unallocated VM!\n";
86             }
87             return;
88     }
```

**4.61.3.8 void VM::updateMIPS ( ) [virtual]**

Definition at line 77 of file vm.cc.

```

77         {
78             return;
79     }
```

**4.61.4 Member Data Documentation****4.61.4.1 vm\_state VM::state [protected]**

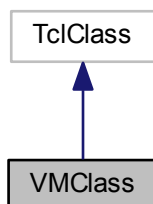
Definition at line 48 of file vm.h.

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

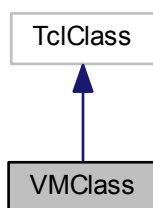
- [vm.h](#)
- [vm.cc](#)

## 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`.

```
13                                     {  
14                                     return (new VM());  
15                                     }
```

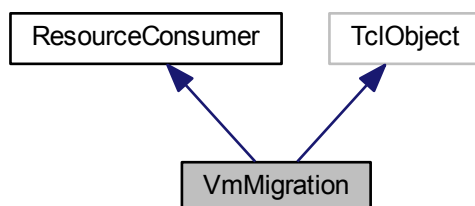
The documentation for this class was generated from the following file:

- [vm.cc](#)

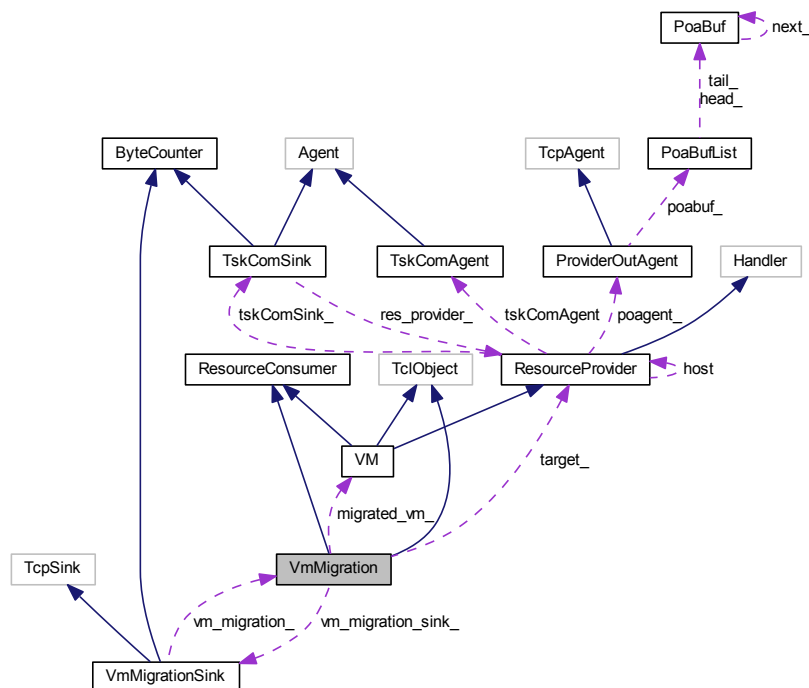
### 4.63 VmMigration Class Reference

```
#include <vmmigration.h>
```

Inheritance diagram for VmMigration:



Collaboration diagram for VmMigration:



#### Public Member Functions

- [VmMigration](#) ()
- void [italizeMigration](#) ([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
149         if (argc == 3) {
150             if (strcmp(argv[1], "set-sink") == 0) {
151                 VmMigrationSink *vms = dynamic_cast<
VmMigrationSink*> (TclObject::lookup(argv[2]));
152                 if(vms){
153                     vm_migration_sink_ = vms;
154                     vm_migration_sink_>setVmMigration(this);
155                     return (TCL_OK);
156                 }
157                 return (TCL_ERROR);
158             } else if (strcmp(argv[1], "set-source") == 0) {
159                 TcpAgent *vma = dynamic_cast<TcpAgent*> (TcpAgent::lookup(
argv[2]));
160                 if(vma){
161                     vm_migration_sender_=vma;
162                     return (TCL_OK);
163                 }
164                 return (TCL_ERROR);
165             } else if (strcmp(argv[1], "set-id") == 0) {
166                 char* stat;
167                 int num = strtol(argv[2], &stat, 10);
168                 if (!*stat){
169                     id_=num;
170                     return (TCL_OK);
171                 }
172                 else {
173                     return (TCL_ERROR);
174                 }
175             }
176         }
177         return TCL_ERROR;
178 }
```

## 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) $vmmigrationsink_(%d) ",
target_>id_, id_);
107
108             target_>releaseAllocation(this);
109
110             ResourceProvider* source = migrated_vm_>
getHost();
112             source->removeVM(migrated_vm_);
113             // Re-attach tasks sinks
114             tcl.evalf("$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) $vmtskoutputsink_(%d)",
migrated_vm_>id_, migrated_vm_>id_);
131
132             tcl.evalf("$vms_(%d) attach-agent $vmtskoutputagent_(%d)",
migrated_vm_>id_, migrated_vm_>id_);
133
134             if(target_>addVM(migrated_vm_)){
135                 //
136                 // 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::initilizeMigration ( VM \* vm, ResourceProvider \* target )

Definition at line 22 of file vmmigration.cc.

```

22                                     {
23
24         migrated_vm_ = vm;
25         target_ = target;
26
27         this->size_ = vm->size_;
28         this->isTask = false;
29         this->isVM = false;
30         res_demands.clear();
31         res_demands = std::vector<ResDemand *>(vm->
res_demands.size(), NULL);
32
33         std::vector<ResDemand*>::iterator iter;
34         std::vector<ResDemand*>::iterator iter2;
35         for (iter = vm->res_demands.begin(), iter2=
res_demands.begin(); iter!=vm->res_demands.end(); iter++, iter2++)
36         {
37                 (*iter2)=new ResDemand(*(*iter), NULL);
38         }
39
40
41         // Create new migration sink and source;
42
43         Tcl& tcl = Tcl::instance();
44         // Create communication source and sink:
45         tcl.evalf("set vmmigrationsource_($next_migration_id) [new Agent/TCP/ProvOutAgent]");
46         tcl.evalf("set vmmigrationsink_($next_migration_id) [new Agent/TCPSink/VmMigrationSink]");
47
48         //Attach source and sink in network topology
49         tcl.evalf("$ns attach-agent $servers_($d) $vmmigrationsource_($next_migration_id)",
migrated_vm_>getHost()->id_);
50         tcl.evalf("$ns attach-agent $servers_($d) $vmmigrationsink_($next_migration_id)",
target_>id_);
51         tcl.evalf("$ns connect $vmmigrationsource_($next_migration_id)
$vmmigrationsink_($next_migration_id)");
52
53
54         //Attach source and sink to host to track network interface utilization
55         tcl.evalf("$hosts_($d) attach-vm-mig-source $vmmigrationsource_($next_migration_id)",
migrated_vm_>getHost()->id_);
56         tcl.evalf("$hosts_($d) attach-vm-mig-sink $vmmigrationsink_($next_migration_id)",
target_>id_);
57
58
59
60         //Set source and destination in migration object itself:
61         tcl.evalf("$vmmigration_($next_migration_id) set-source
$vmmigrationsource_($next_migration_id)");
62         tcl.evalf("$vmmigration_($next_migration_id) set-sink
$vmmigrationsink_($next_migration_id)");
63
64
65
66         tcl.evalf("incr next_migration_id");
67
68 }

```

#### 4.63.3.4 void VmMigration::startMigration ( )

Definition at line 77 of file vmmigration.cc.

```

77         {
78         migrated_vm_>getHost()->
updateEnergyAndConsumption();
79         target_>updateEnergyAndConsumption();
80
81         if(target_>tryToAllocate(this)){
82                 //
std::cerr << "Migration object successfully
allocated on target.\n";
83         } else {
84                 std::cerr << "Error: Migration object NOT successfully allocated on target.
\n";
85                 return;
86         }
87
88         // Create, attach and allocate migration agents: source and sink
89         //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.)
90         int mig_s = migrated_vm_>getTotalCap(
Memory); // migration size
91         int p_s = vm_migration_sender->getPacketSize(); // packet size
92         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
93         vm_migration_sender->sendmsg(mig_s, (void*) 1);
94 }

```

#### 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]`

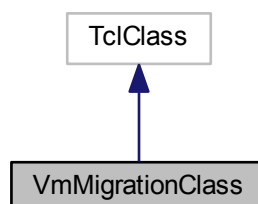
Definition at line 29 of file vmmigration.h.

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

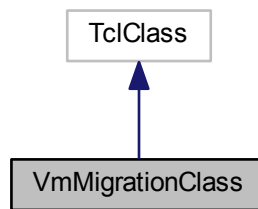
- [vmmigration.h](#)
- [vmmigration.cc](#)

#### 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`.

```
12 : 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`.

```
13                                     {
14         return (new VmMigration());
15     }
```

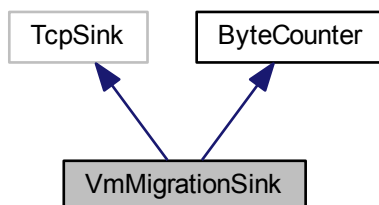
The documentation for this class was generated from the following file:

- [vmmigration.cc](#)

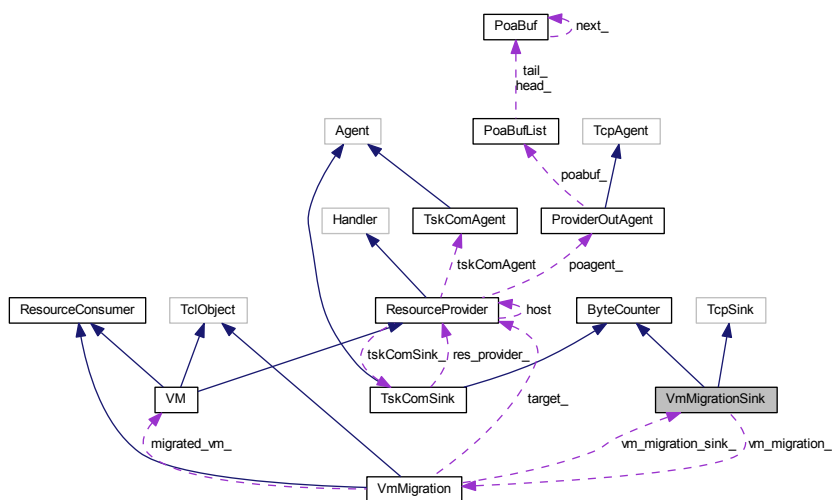
## 4.65 VmMigrationSink Class Reference

```
#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 `vmigrationsink.h`.

### 4.65.2 Constructor & Destructor Documentation

#### 4.65.2.1 VmMigrationSink::VmMigrationSink ( Acker \* a )

Definition at line 19 of file `vmigrationsink.cc`.

```

19                                     : TcpSink(a), seq_expected_(-1),
   migration_finished_(false) {
20
21 }
```

#### 4.65.2.2 VmMigrationSink::~VmMigrationSink ( ) [virtual]

Definition at line 23 of file `vmigrationsink.cc`.

```

23                                     {
24
25 }
```

### 4.65.3 Member Function Documentation

#### 4.65.3.1 void VmMigrationSink::recv ( Packet \* pkt, Handler \* ) [virtual]

Definition at line 31 of file `vmigrationsink.cc`.

```

32 {
33     bytes_since_ += hdr_cmn::access(pkt)->size();
34
35     if(seq_expected_ == hdr_tcp::access(pkt)->seqno()) {
36         // Migration complete.
37         migration_finished_ = true;
38         vm_migration_->finalizeMigration();
39     }
40     else if(migration_finished_) {
41         std::cerr << "ERROR! Something went wrong. Packets received after migration
   is finished!\n";
42     }
43     TcpSink::recv(pkt, this);
44 }
```

#### 4.65.3.2 void VmMigrationSink::setVmMigration ( VmMigration \* vm\_migration )

Definition at line 27 of file vmmigrationsink.cc.

```
27                                     {  
28         vm_migration_ = vm_migration;  
29     }
```

#### 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]

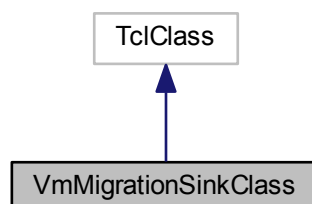
Definition at line 24 of file vmmigrationsink.h.

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

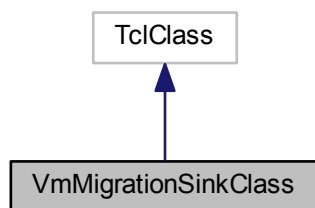
- [vmmigrationsink.h](#)
- [vmmigrationsink.cc](#)

## 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`.

```
13 : 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`.

```
14                                     {
15         return (new VmMigrationSink(new Acker()));
16     }
```

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

- [vmmigrationsink.cc](#)

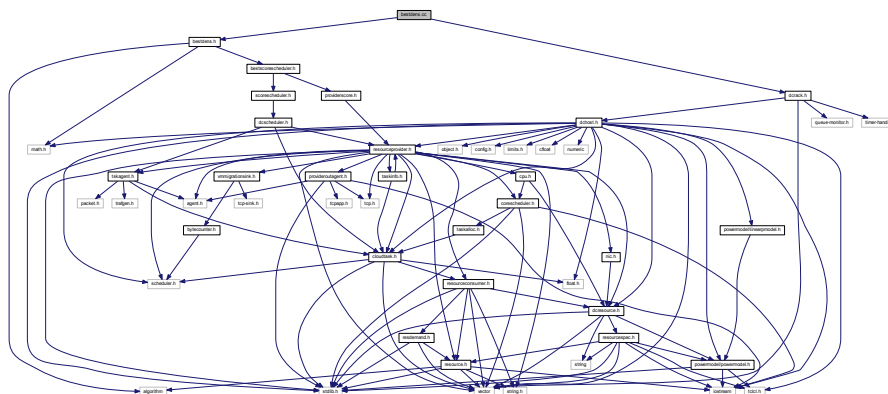
## 5 File Documentation

## 5.1 bestdens.cc File Reference

```
#include "bestdens.h"
```

```
#include "dcrack.h"
```

Include dependency graph for bestdens.cc:



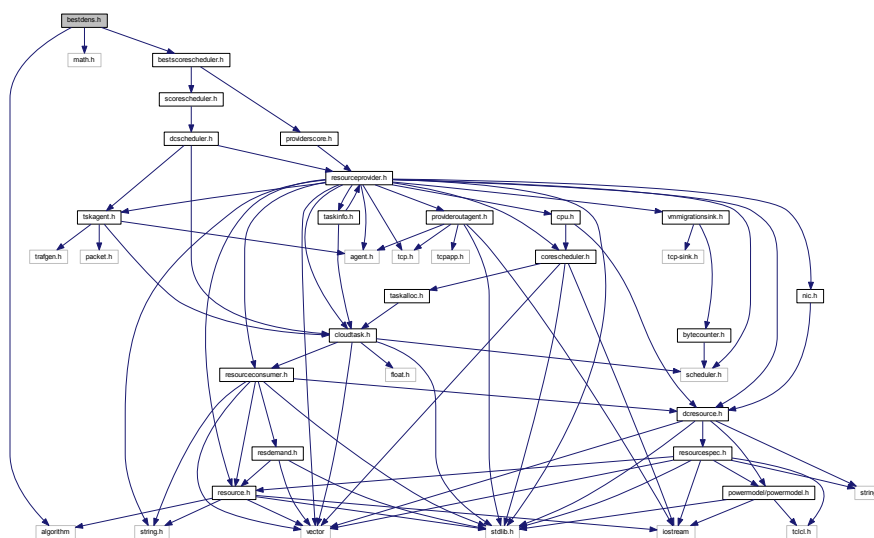
## 5.2 bestdens.h File Reference

```
#include <algorithm>
```

```
#include <math.h>
```

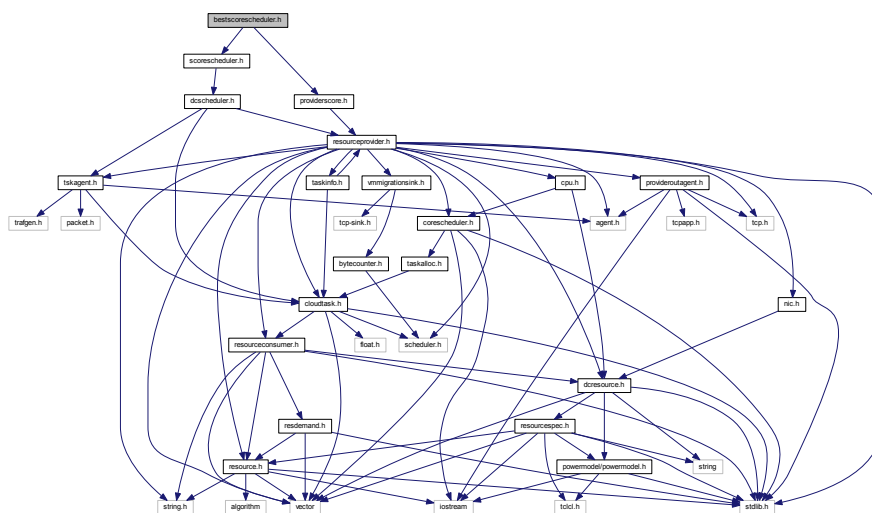
```
#include "bestscorescheduler.h"
```

Include dependency graph for bestdens.h:

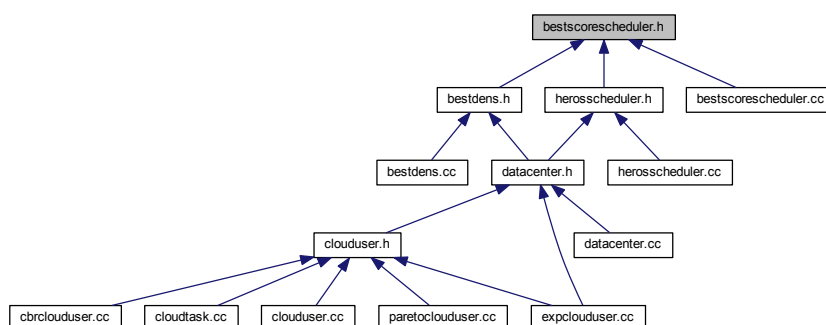




Include dependency graph for bestscorescheduler.h:



This graph shows which files directly or indirectly include this file:



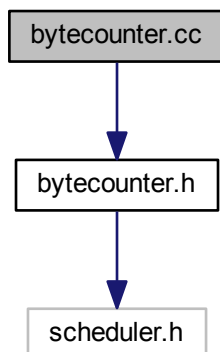
## Classes

- class **BestScoreScheduler**

## 5.5 bytearray.cc File Reference

```
#include "bytecounter.h"
```

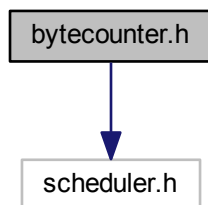
Include dependency graph for bytecounter.cc:



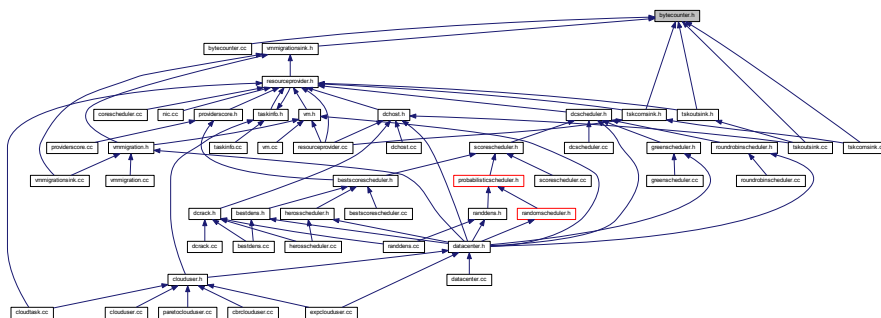
## 5.6 bytearray.h File Reference

```
#include "scheduler.h"
```

Include dependency graph for bytecounter.h:



This graph shows which files directly or indirectly include this file:



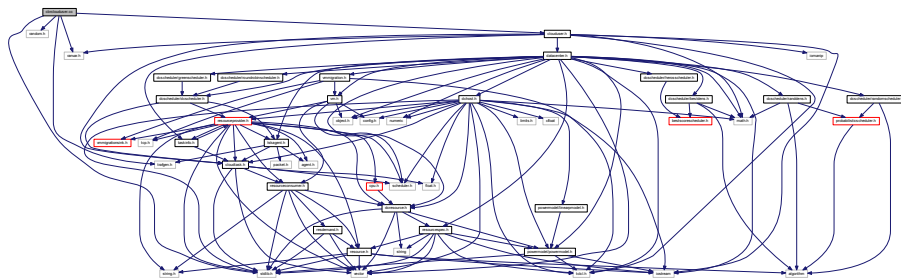
## Classes

- 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`

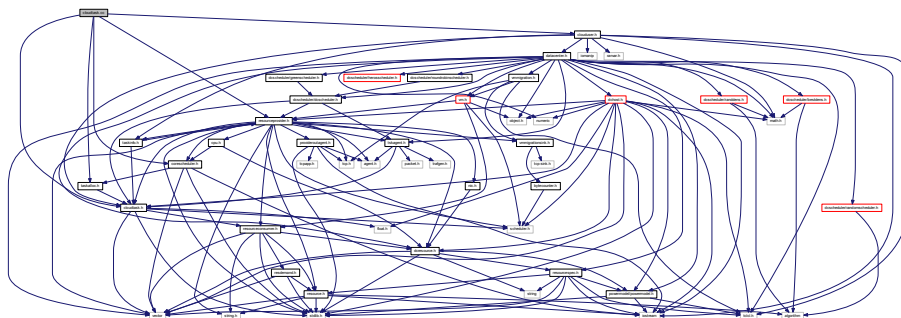
## 5.7.1 Variable Documentation

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:

=

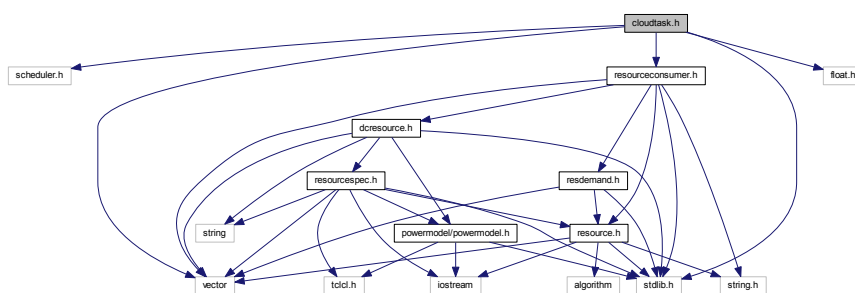
```
"@(#) $Header: /cvsroot/nsnam/ns-2/common/taskobject.cc,v 1.43 $"
```

Definition at line 6 of file cloudtask.cc.

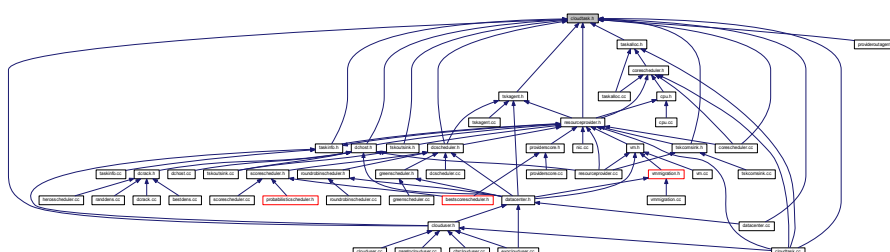
## 5.9 cloudtask.h File Reference

```
#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:



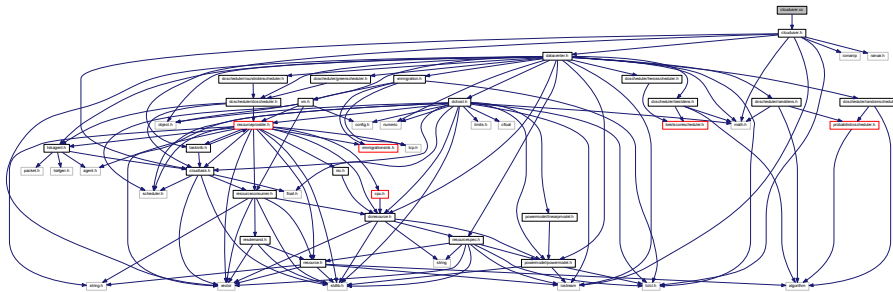
## Classes

- class [CloudTask](#)

## 5.10 clouduser.cc File Reference

```
#include "clouduser.h"
```

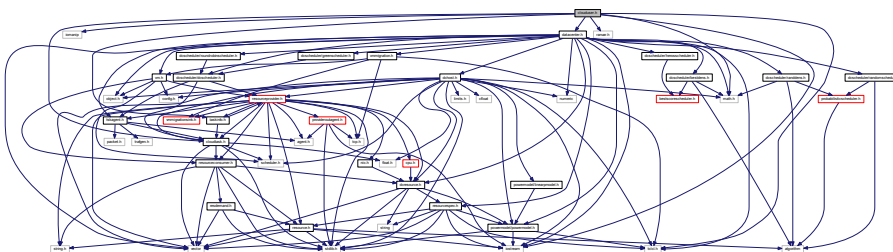
Include dependency graph for clouduser.cc:



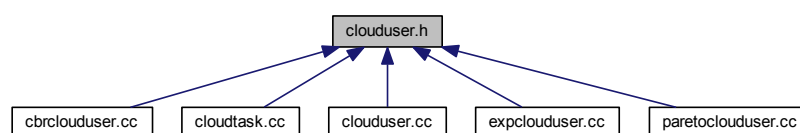
## 5.11 clouduser.h File Reference

```
#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:

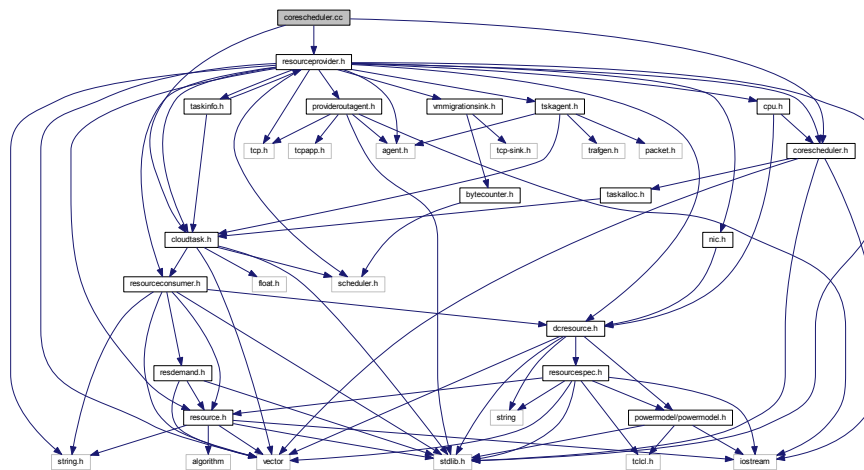


## Classes

- class [CloudUser](#)

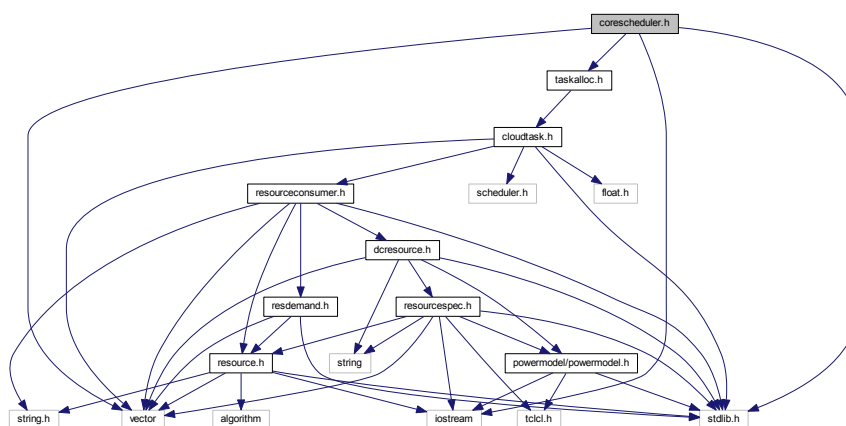
## 5.12 corescheduler.cc File Reference

```
#include "corescheduler.h"
#include "cloudtask.h"
#include "resourceprovider.h"
Include dependency graph for corescheduler.cc:
```

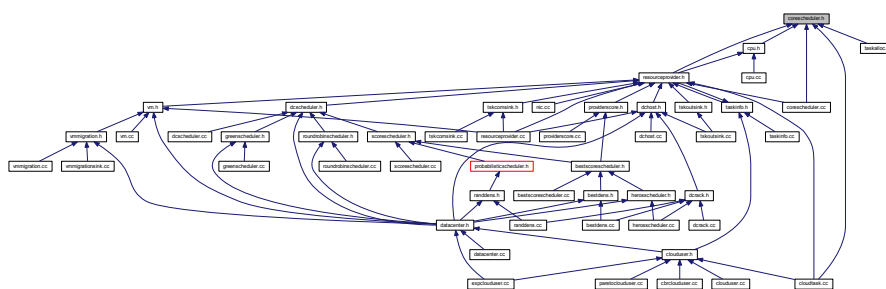


### 5.13 corescheduler.h File Reference

```
#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:



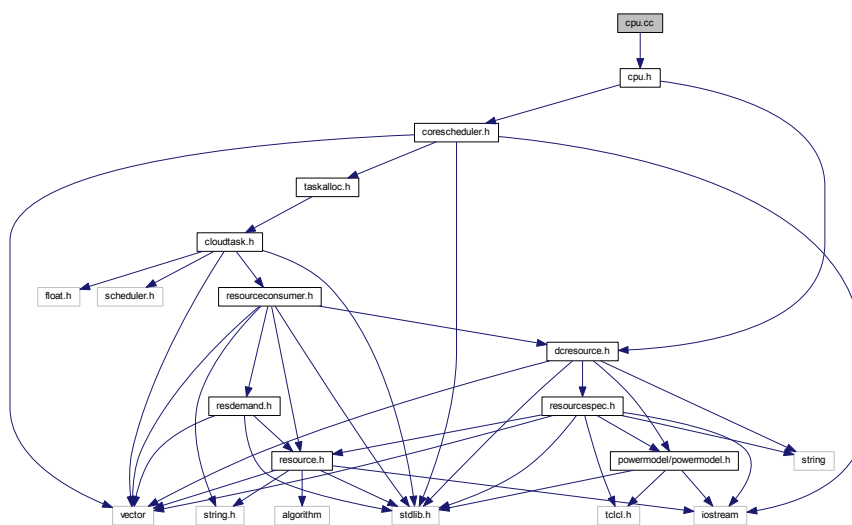
## Classes

- 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

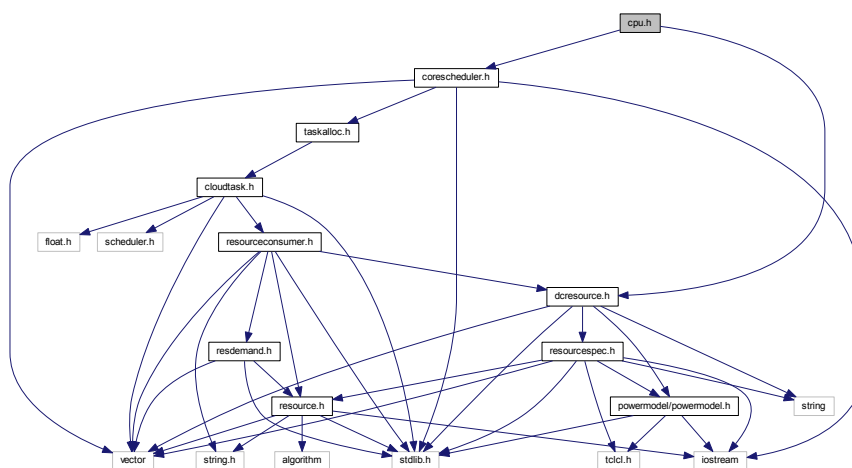
#### 5.14.1.1 CpuClass class\_cpu [static]

## 5.15 cpu.h File Reference

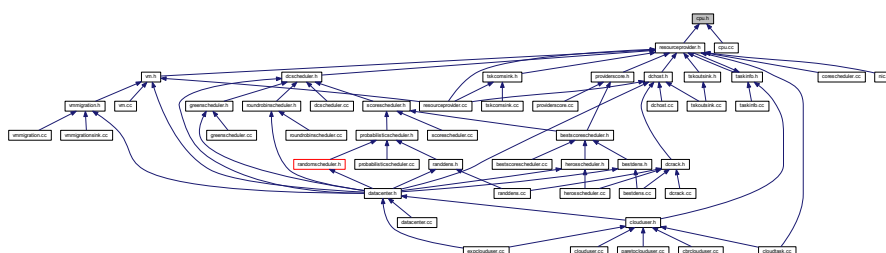
```
#include "corescheduler.h"
```

```
#include "dcresource.h"
```

Include dependency graph for cpu.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [CPU](#)

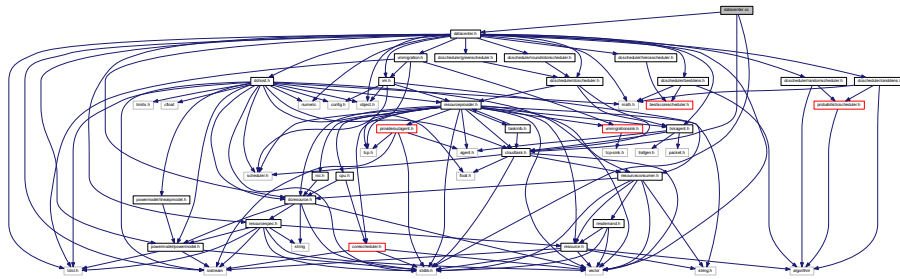
## 5.16 datacenter.cc File Reference

```
#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 [class\\_datacenter](#)

### 5.16.1 Variable Documentation

5.16.1.1 [DataCenterClass](#) class [class\\_datacenter](#) [static]

5.16.1.2 const char [rcsid](#)[] [static]

#### Initial value:

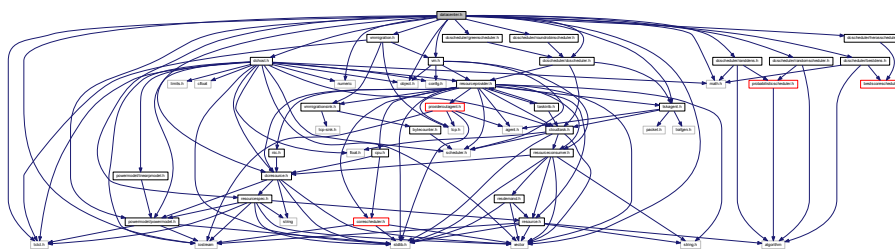
```
=
                                     "@(#) $Header: /cvsrcroot/nsnam/ns-2/common/datacenter.cc,v 1.43 $"
```

Definition at line 6 of file datacenter.cc.

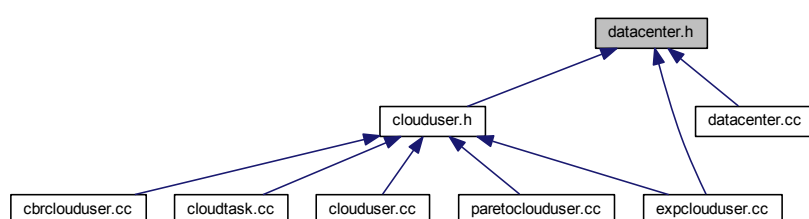
## 5.17 datacenter.h File Reference

```
#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 <ttlcl.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:

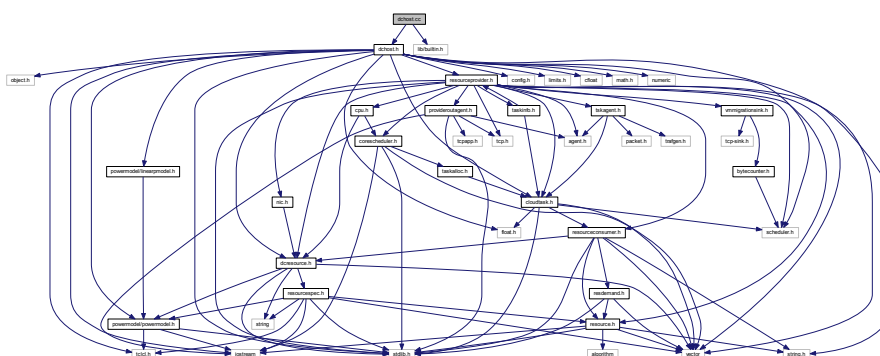


## Classes

- 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](#)

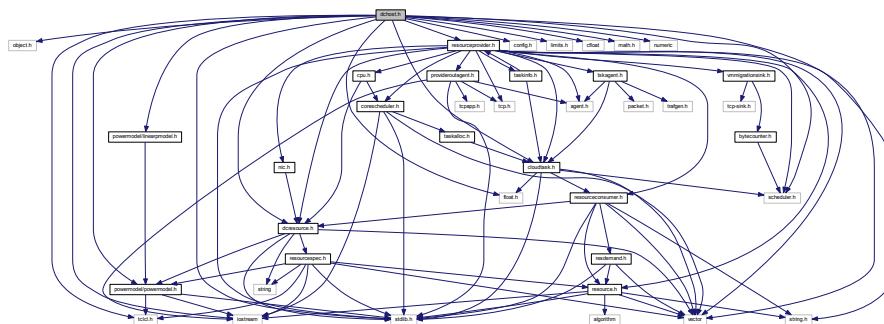
## 5.18.1 Variable Documentation

## 5.18.1.1 DcHostClass class\_dchost [static]

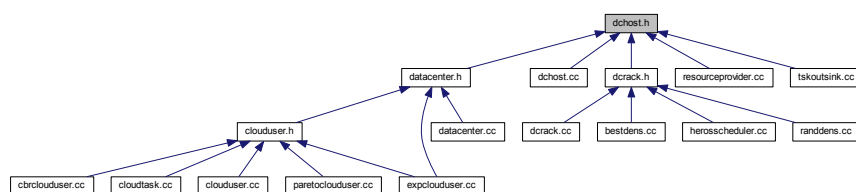
## 5.19 dchost.h File Reference

```
#include "object.h"
#include "cloudtask.h"
#include "config.h"
#include "scheduler.h"
#include <stdlib.h>
#include <limits.h>
#include <time.h>
#include <vector>
#include <cmath>
#include <float.h>
#include <math.h>
#include <iostream>
#include <numeric>
#include "dcresource.h"
#include "resourceprovider.h"
#include "powermodel/linearpowermodel.h"
#include "powermodel/powermodel.h"
```

Include dependency graph for dchost.h:



This graph shows which files directly or indirectly include this file:



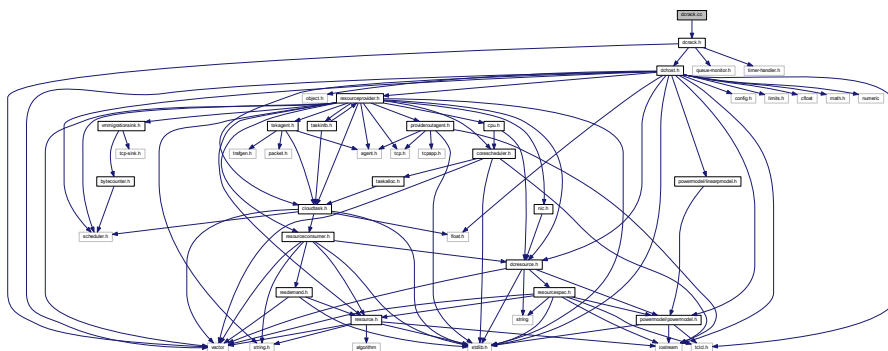
## Classes

- class [DcHost](#)

## 5.20 dcrack.cc File Reference

```
#include "dcrack.h"
```

Include dependency graph for dcrack.cc:



## Classes

- class [DcRackClass](#)

## Variables

- [DcRackClass](#) `class_dcrack`

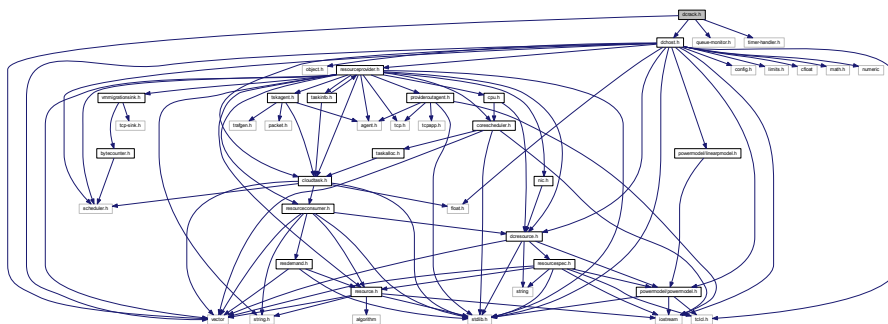
## 5.20.1 Variable Documentation

5.20.1.1 [DcRackClass](#) `class_dcrack` [static]

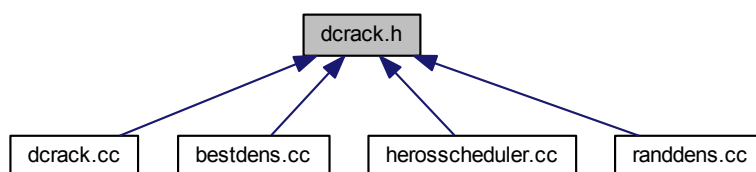
## 5.21 dcrack.h File Reference

```
#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:



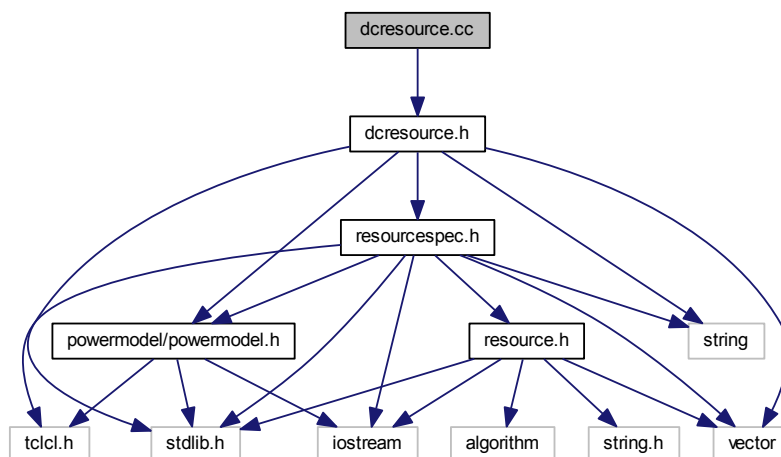
#### Classes

- class [DcRack](#)

#### 5.22 dcresource.cc File Reference

```
#include "dcresource.h"
```

Include dependency graph for dcresource.cc:



#### Classes

- class [DcResourceClass](#)

#### Variables

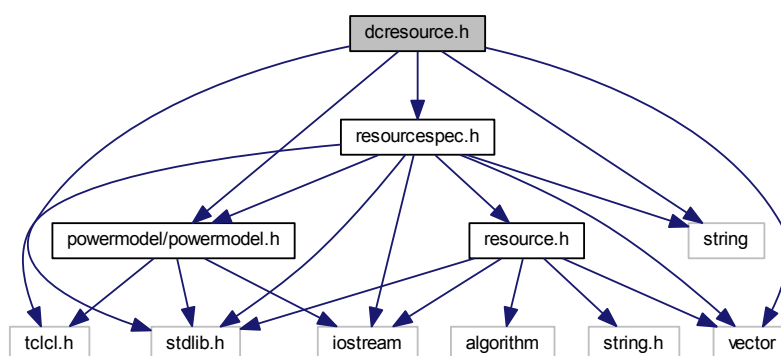
- [DcResourceClass](#) `class_dcresource`

### 5.22.1 Variable Documentation

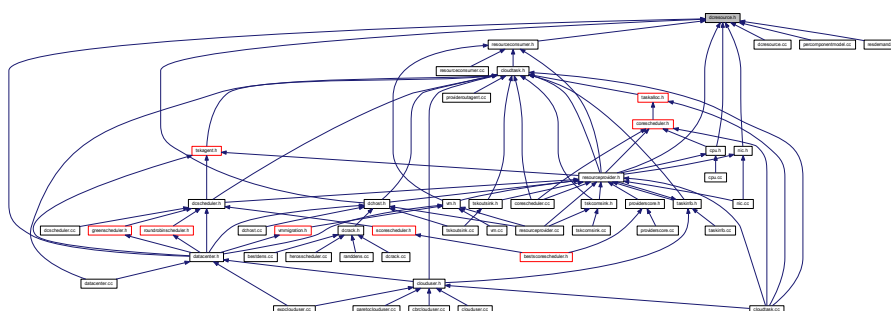
#### 5.22.1.1 DcResourceClass class\_dcresource [static]

### 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:



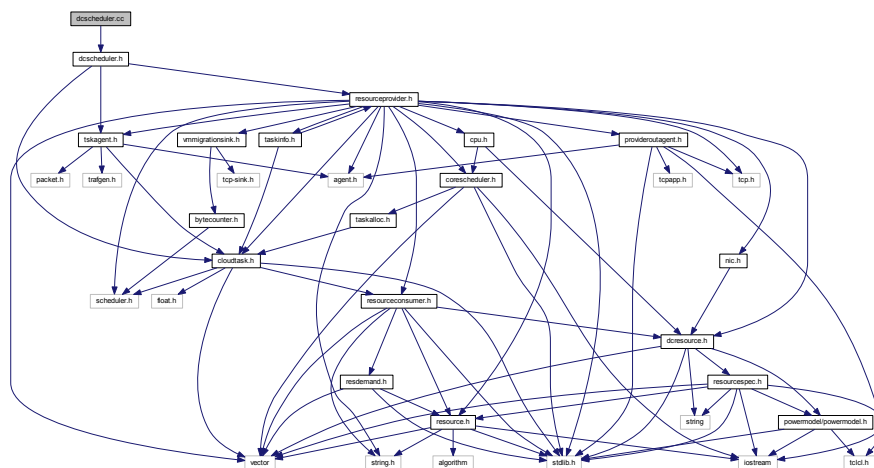
### Classes

- class [DcResource](#)

## 5.24 dcscheduler.cc File Reference

```
#include "dcscheduler.h"
```

Include dependency graph for dcscheduler.cc:



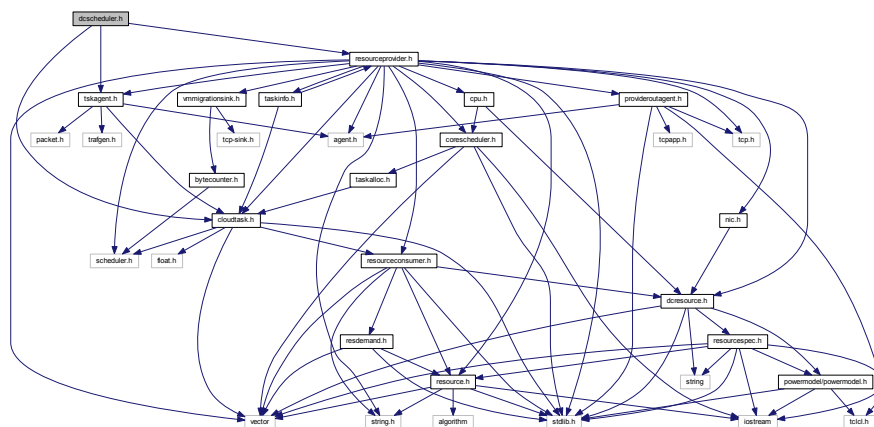
## 5.25 dcscheduler.h File Reference

```
#include "tskagent.h"
```

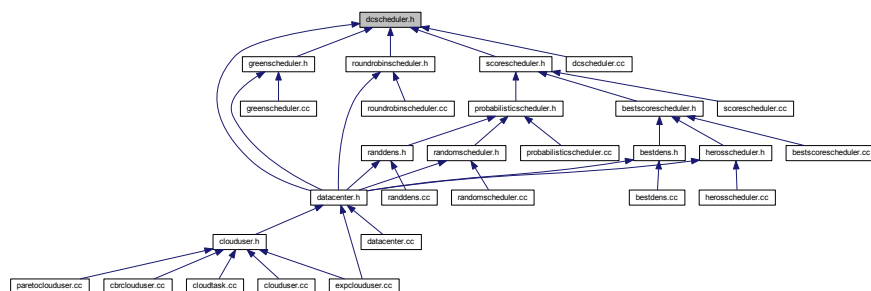
```
#include "cloudtask.h"
```

```
#include "resourceprovider.h"
```

Include dependency graph for dcscheduler.h:



This graph shows which files directly or indirectly include this file:



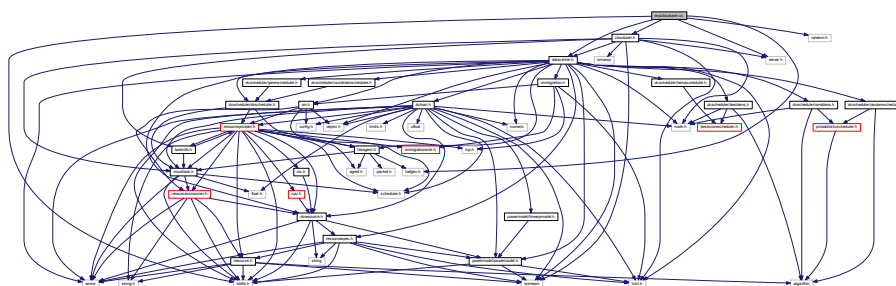
## Classes

- 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 $"

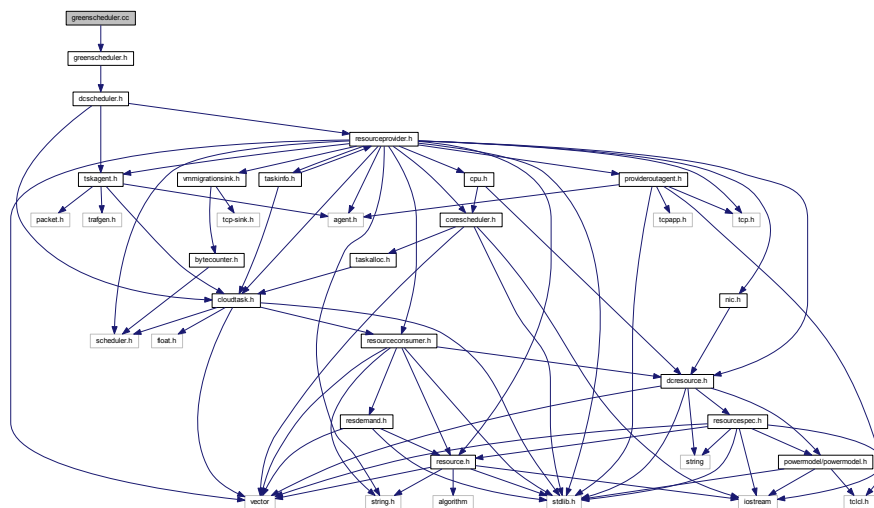
```

Definition at line 2 of file expclouduser.cc.

## 5.27 greenscheduler.cc File Reference

```
#include "greenscheduler.h"
```

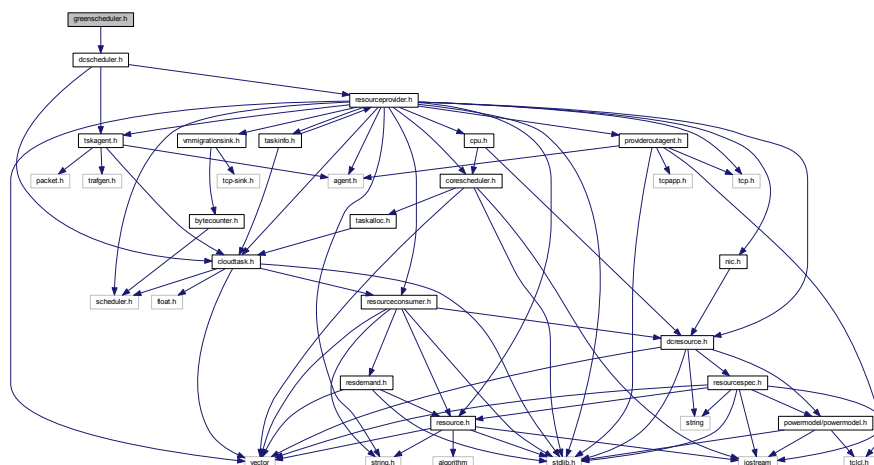
Include dependency graph for greenscheduler.cc:



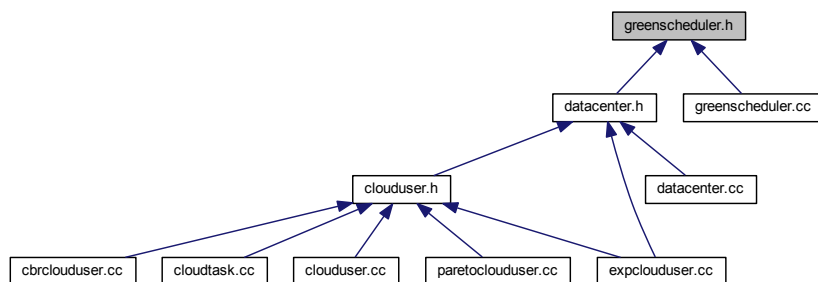
## 5.28 greenscheduler.h File Reference

```
#include "dcscheduler.h"
```

Include dependency graph for greenscheduler.h:



This graph shows which files directly or indirectly include this file:



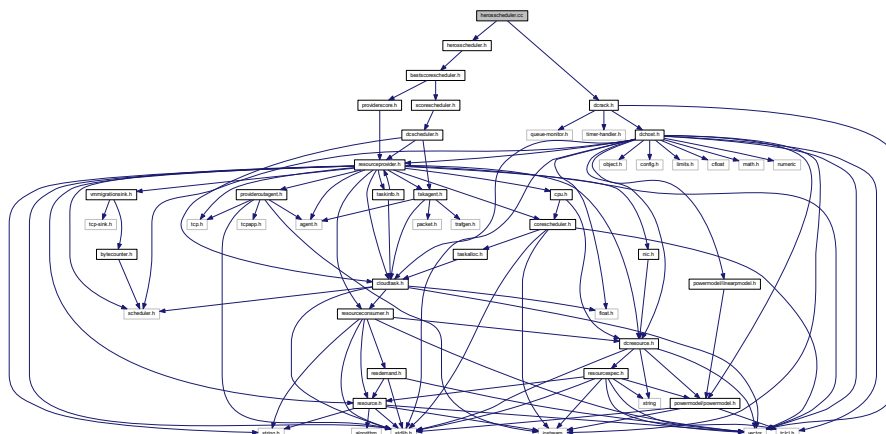
## Classes

- 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.

```

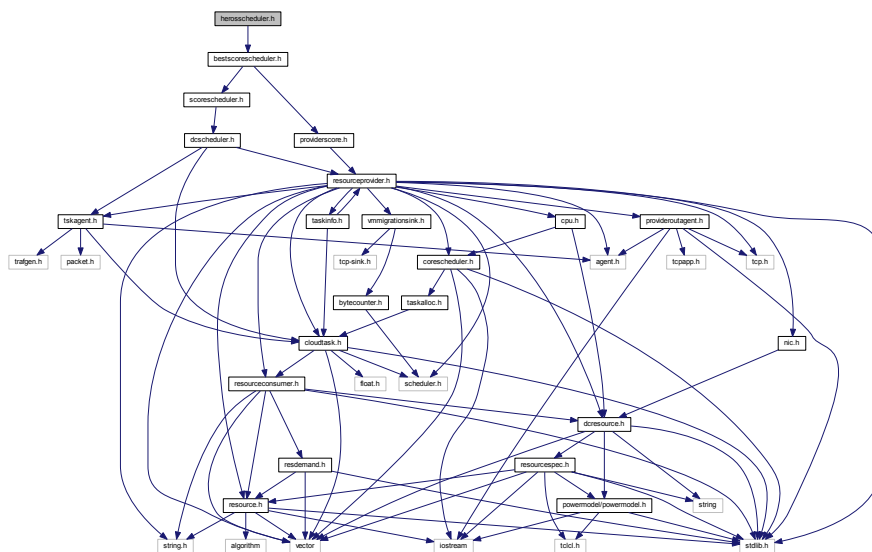
113
114         if(first.score_ != second.score_){
115             return first.score_ < second.score_;
116         } else {
117             return HerosScheduler::performancePerWattMax
118                 (first.provider_)*first.comm_potential_ <
119                 HerosScheduler::performancePerWattMax(second.
120                 provider_)*second.comm_potential_;
121         }
122     }
123 }

```

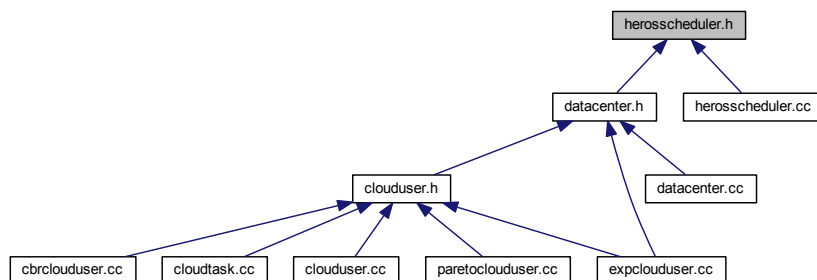
### 5.30 herosscheduler.h File Reference

```
#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.

```

113                                     {
114         if(first.score_ != second.score_){
115             return first.score_ < second.score_;
116         } else {
117             return HerosScheduler::performancePerWattMax
118 (first.provider_)*first.comm_potential_ <
119 HerosScheduler::performancePerWattMax(second.
120 provider_)*second.comm_potential_;
121         }
122     }
123 }

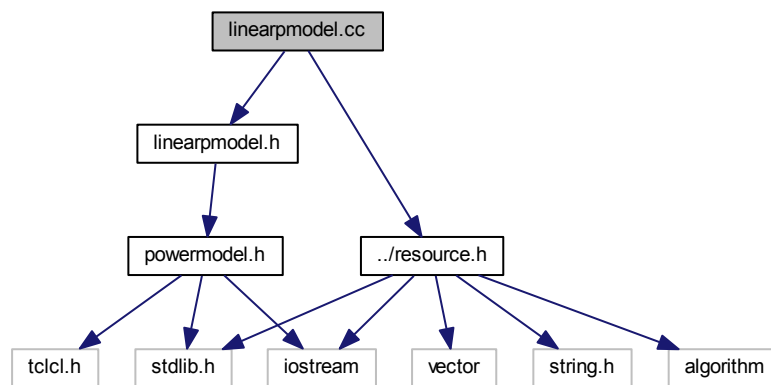
```

## 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

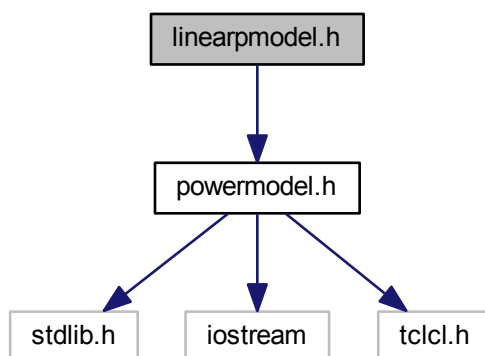
#### 5.31.1 Variable Documentation

##### 5.31.1.1 LinearPModelClass class\_powermodel [static]

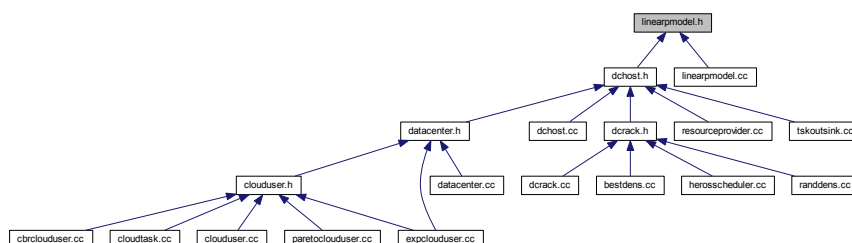
### 5.32 linearpmode.h File Reference

```
#include "powermodel.h"
```

Include dependency graph for linearpmode.h:



This graph shows which files directly or indirectly include this file:



#### Classes

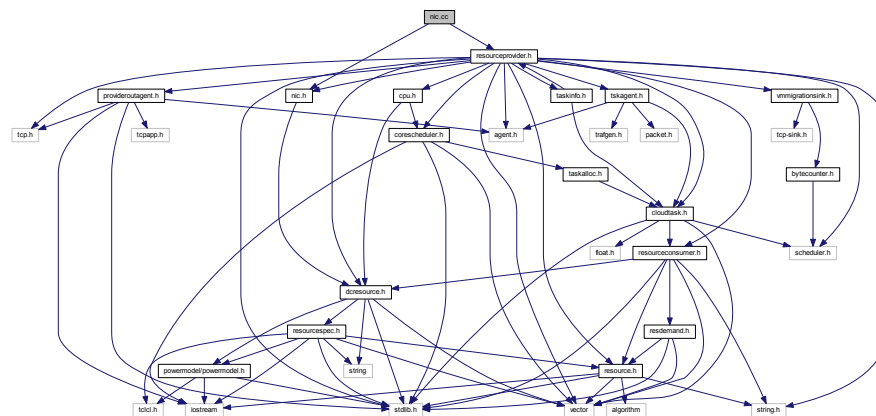
- class [LinearPModel](#)

### 5.33 nic.cc File Reference

```
#include "nic.h"
```

```
#include "resourceprovider.h"
```

Include dependency graph for nic.cc:



## Classes

- class [NicClass](#)

## Variables

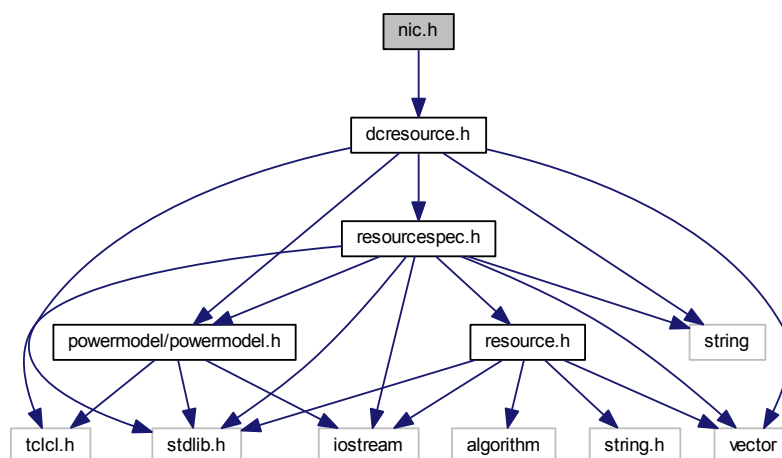
- [NicClass](#) `class_nic`

### 5.33.1 Variable Documentation

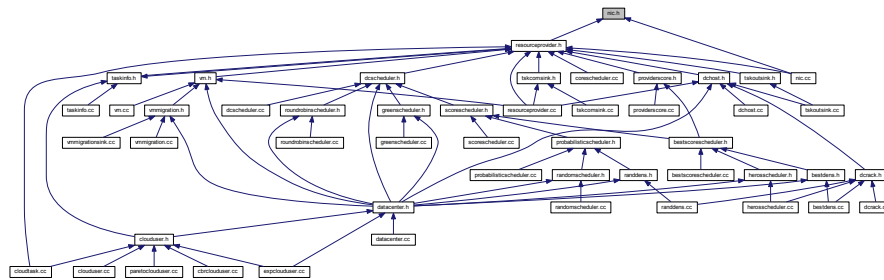
#### 5.33.1.1 NicClass `class_nic` [static]

## 5.34 nic.h File Reference

```
#include "dresource.h"
Include dependency graph for nic.h:
```



This graph shows which files directly or indirectly include this file:



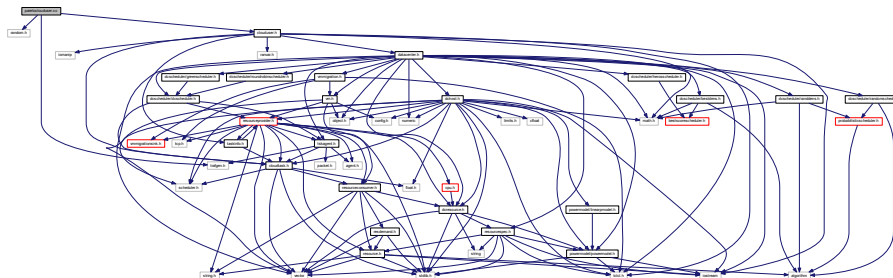
## Classes

- 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:

```
=
05:05:31 tomh Exp $"      "@{#) $Header: /cvsroot/nsnam/ns-2/tools/pareto.cc,v 1.9 2005/08/26
```

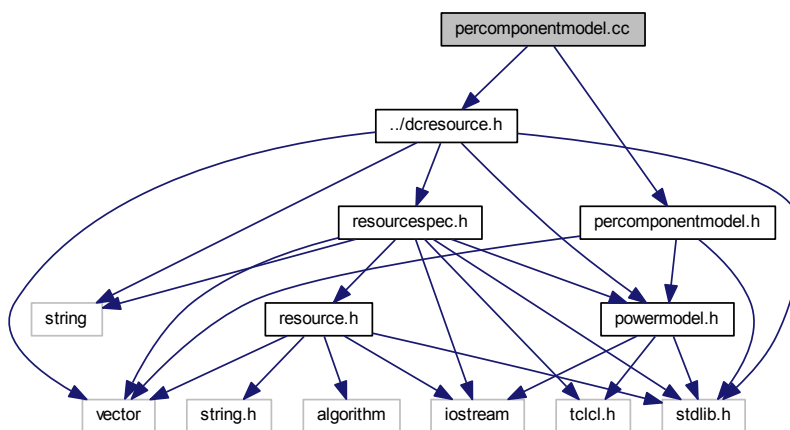
Definition at line 6 of file paretoclouduser.cc.

## 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](#)

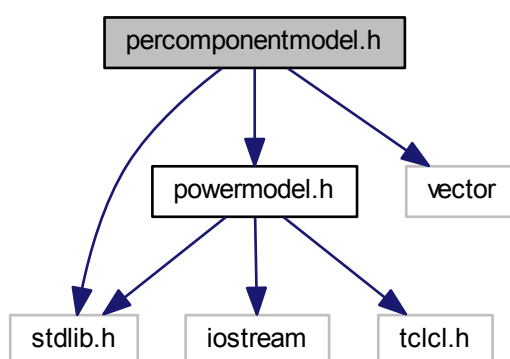
### 5.36.1 Variable Documentation

#### 5.36.1.1 PerComponentModelClass class\_powermodel [static]

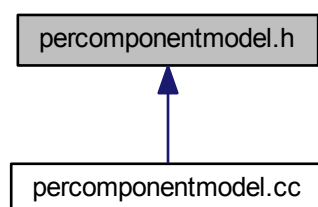
## 5.37 percomponentmodel.h File Reference

```
#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:



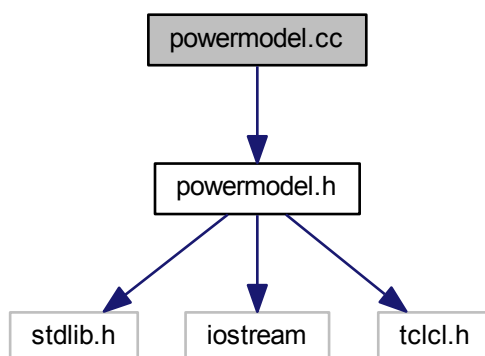
### Classes

- class [PerComponentModel](#)

## 5.38 powermodel.cc File Reference

```
#include "powermodel.h"
```

Include dependency graph for powermodel.cc:



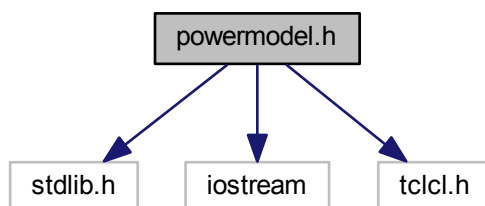
## 5.39 powermodel.h File Reference

```
#include <stdlib.h>
```

```
#include <iostream>
```

```
#include <tclcl.h>
```

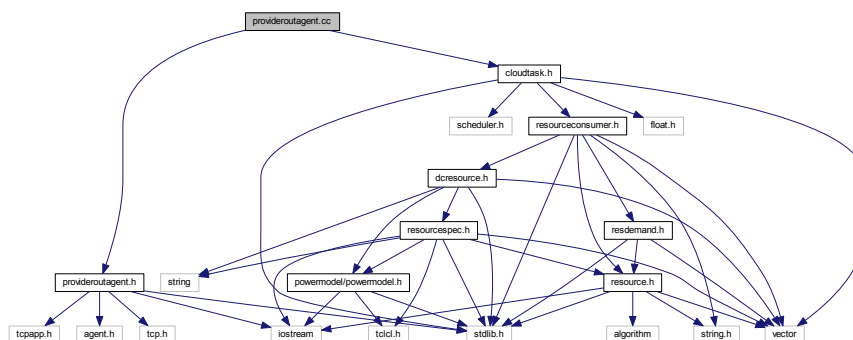
Include dependency graph for powermodel.h:







Include dependency graph for provideroutagent.cc:



## Classes

- class [ProvOutAgentClass](#)

## Variables

- [ProvOutAgentClass class\\_tsk\\_provoutagent](#)

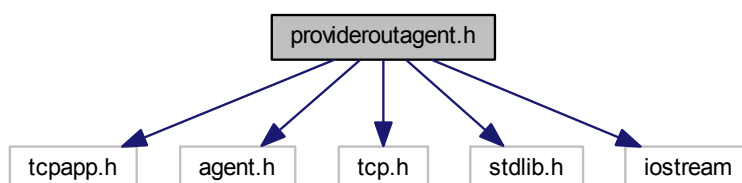
### 5.42.1 Variable Documentation

#### 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>
```

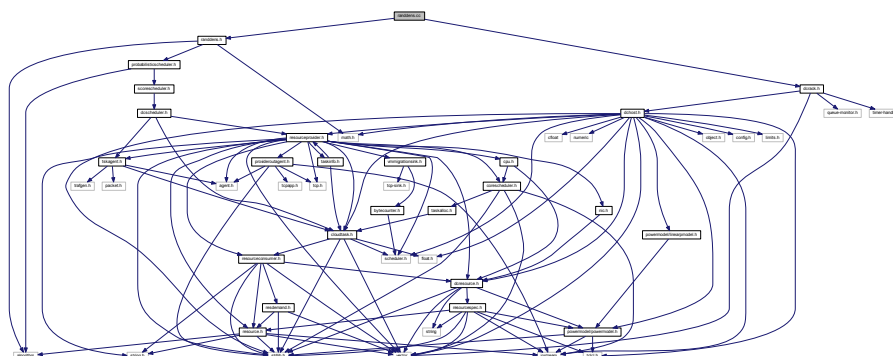
Include dependency graph for provideroutagent.h:





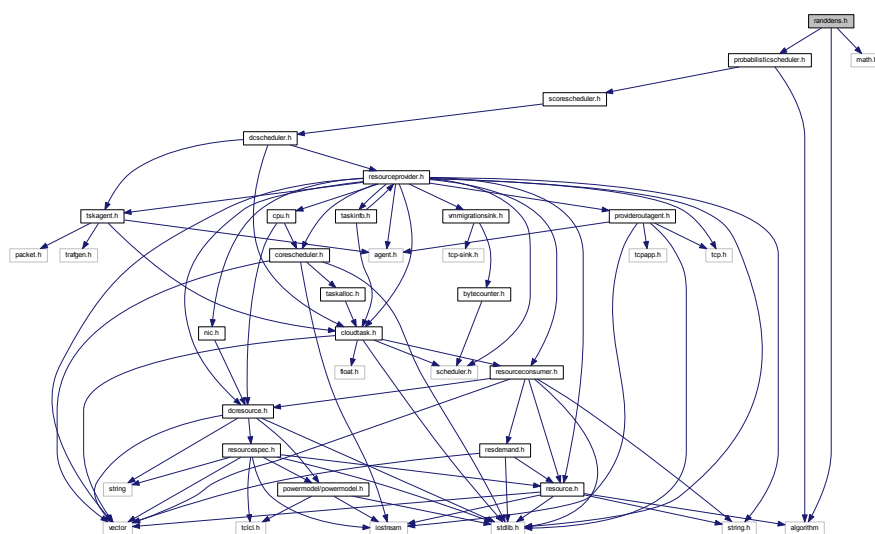


Include dependency graph for randdens.cc:



## 5.47 randdens.h File Reference

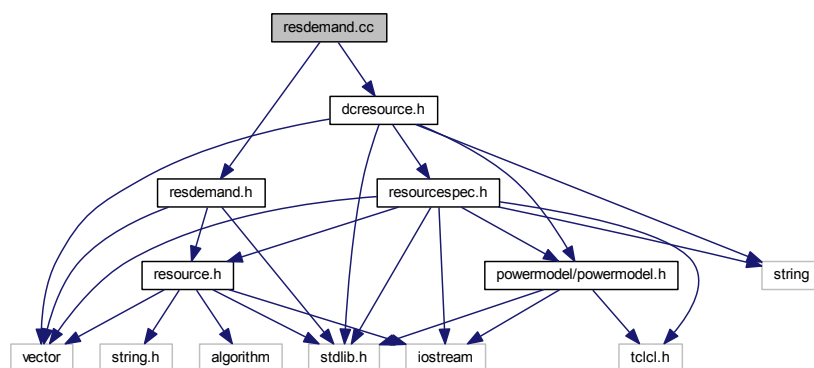
```
#include <algorithm>
#include <math.h>
#include "probabilisticscheduler.h"
Include dependency graph for randdens.h:
```







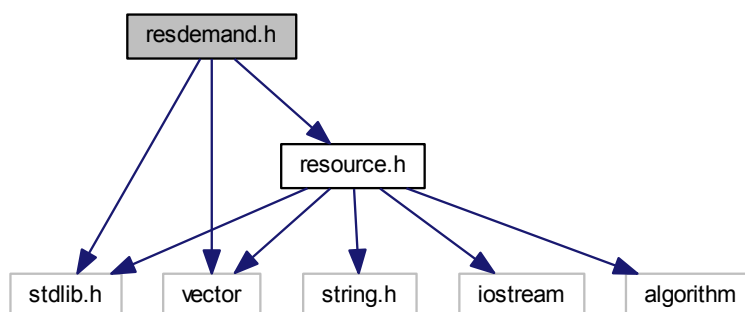
Include dependency graph for resdemand.cc:



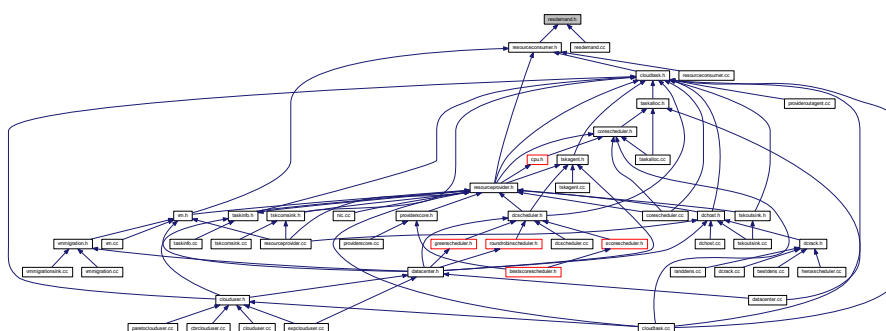
### 5.51 resdemand.h File Reference

```
#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:



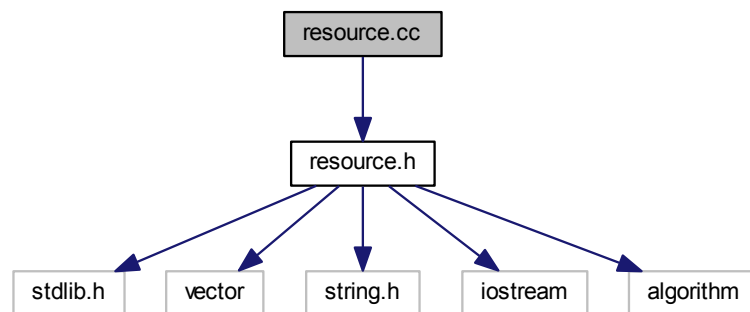
## Classes

- 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>
```

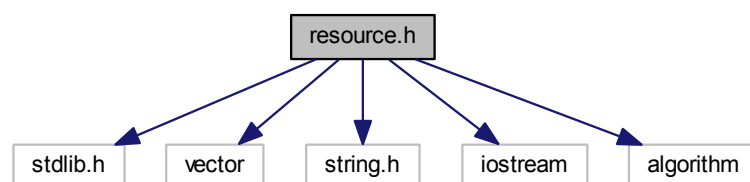
```
#include <vector>
```

```
#include <string.h>
```

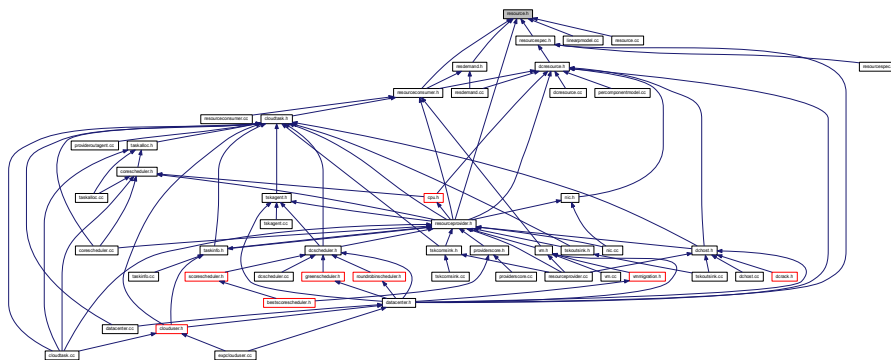
```
#include <iostream>
```

```
#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.

```
19         {  
20             Computing,  
21             Memory,  
22             Storage,  
23             Networking,  
24             FirstResType = Computing,  
25             LastResType = Networking  
26     };
```

### 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.

```
48 {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.

```
65 {return !operator==(lhs, rhs);}
```

#### 5.53.2.3 Capacity operator+( Capacity lhs, const Capacity & rhs ) [inline]

Definition at line 54 of file resource.h.

```
55 {  
56     lhs += rhs;  
57     return lhs;  
58 }
```

#### 5.53.2.4 Capacity operator+ ( Capacity lhs, const double & rhs ) [inline]

Definition at line 71 of file resource.h.

```
72 {  
73     lhs += rhs;  
74     return lhs;  
75 }
```

#### 5.53.2.5 Capacity operator- ( Capacity lhs, const Capacity & rhs ) [inline]

Definition at line 59 of file resource.h.

```
60 {  
61     lhs -= rhs;  
62     return lhs;  
63 }
```

#### 5.53.2.6 Capacity operator- ( Capacity lhs, const double & rhs ) [inline]

Definition at line 76 of file resource.h.

```
77 {  
78     lhs -= rhs;  
79     return lhs;  
80 }
```

#### 5.53.2.7 bool operator< ( const Capacity & lhs, const Capacity & rhs ) [inline]

Definition at line 49 of file resource.h.

```
49 { 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.

```
66 { 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.

```
51 { 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.

```
68 { 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.

```
47 { 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.

```
64 { 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.

```
50 {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.

```
67 {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.

```
52 {return !operator< (lhs, rhs); }
```

**5.53.2.16** `bool operator>=( const Capacity & lhs, const double & rhs )` `[inline]`

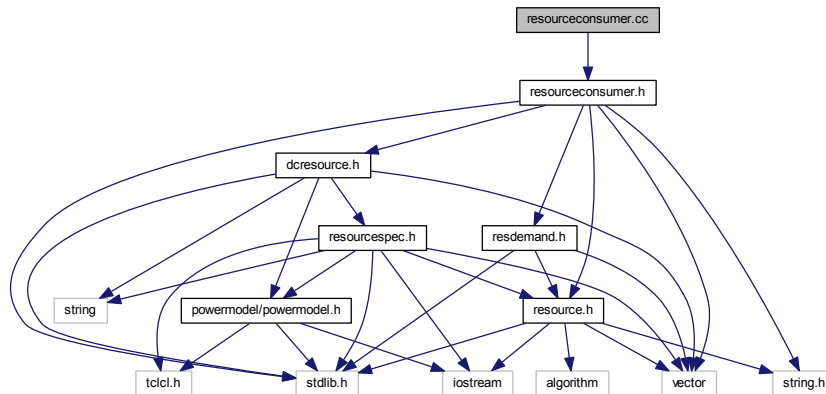
Definition at line 69 of file resource.h.

```
69 {return !operator< (lhs, rhs); }
```

### 5.54 resourceconsumer.cc File Reference

```
#include "resourceconsumer.h"
```

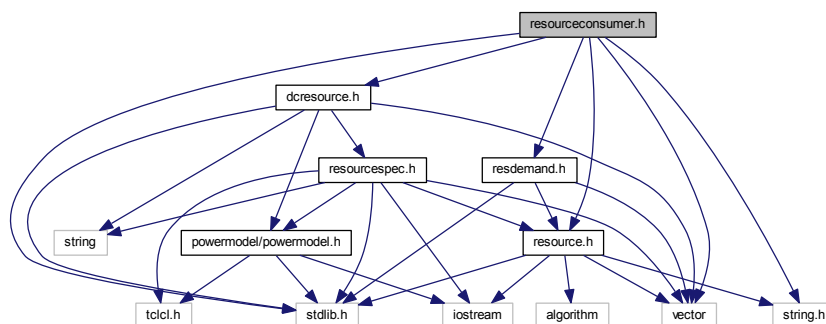
Include dependency graph for resourceconsumer.cc:



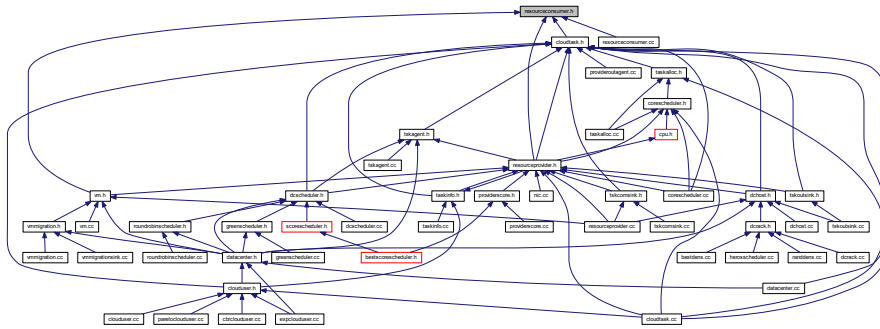
### 5.55 resourceconsumer.h File Reference

```
#include "resdemand.h"
#include "resource.h"
#include <stdlib.h>
#include <vector>
#include <string.h>
#include "dresource.h"
```

Include dependency graph for resourceconsumer.h:



This graph shows which files directly or indirectly include this file:



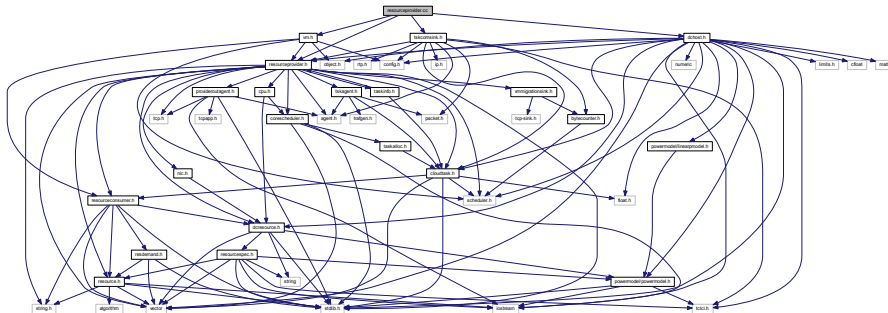
## Classes

- 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:

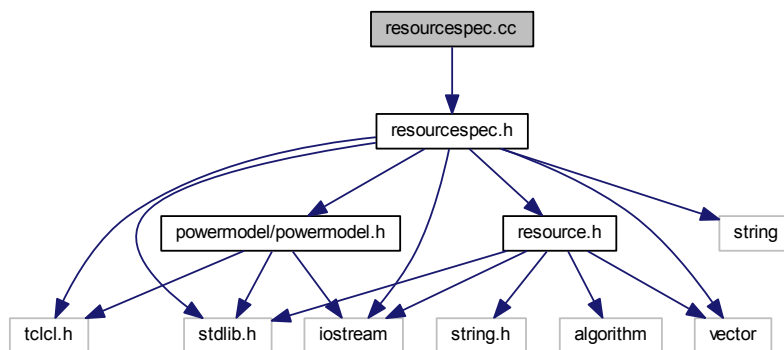


## 5.57 resourceprovider.h File Reference

```
#include <stdlib.h>
```



Include 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]

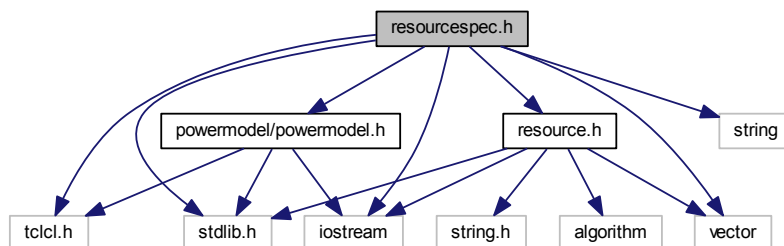
## 5.59 resourcespec.h File Reference

```

#include "resource.h"
#include "powermodel/powermodel.h"
#include <stdlib.h>
#include <vector>
#include <tcclcl.h>
#include <iostream>
#include <string>

```

Include dependency graph for resourcespec.h:

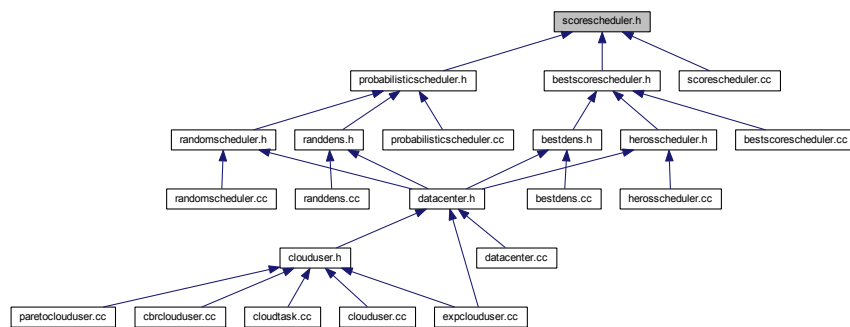








This graph shows which files directly or indirectly include this file:

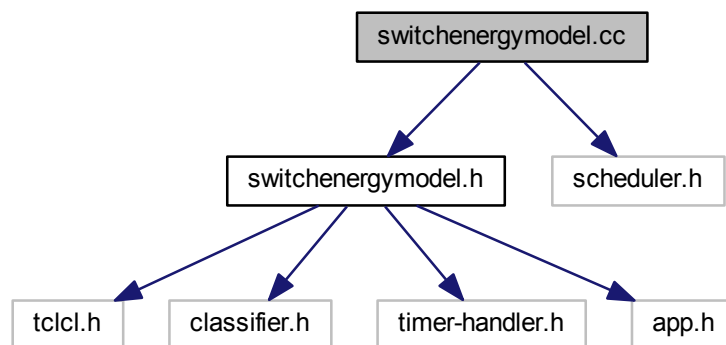


## Classes

- 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

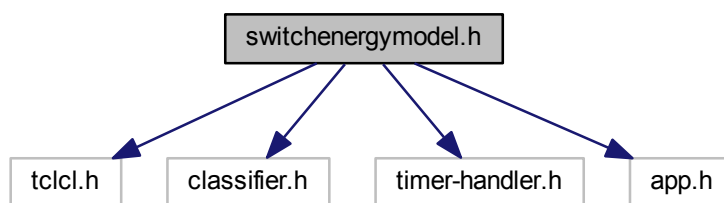
### 5.64.1 Variable Documentation

#### 5.64.1.1 SwitchEnergyModelClass class\_switchenergymodel [static]

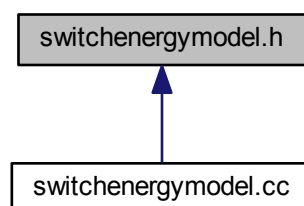
### 5.65 switchenergymodel.h File Reference

```
#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

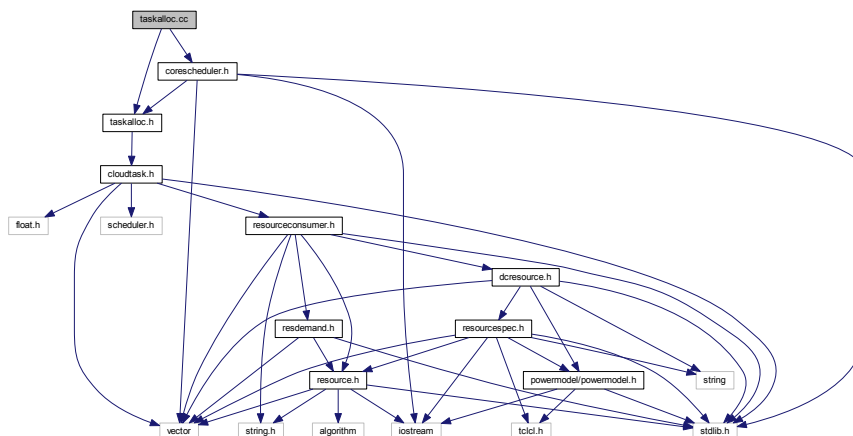
- class [SwitchEnergyTimer](#)
- class [SwitchEnergyModel](#)

## 5.66 taskalloc.cc File Reference

```
#include "taskalloc.h"
```

```
#include "corescheduler.h"
```

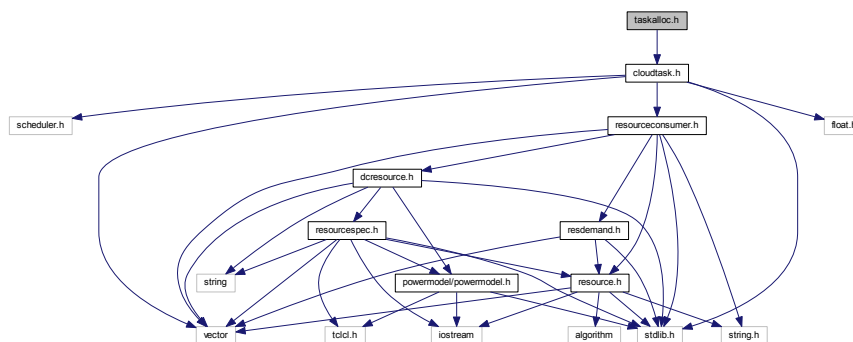
Include dependency graph for taskalloc.cc:



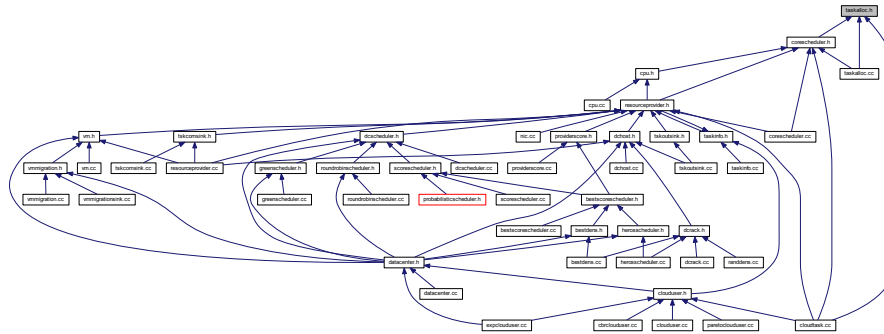
## 5.67 taskalloc.h File Reference

```
#include "cloudtask.h"
```

Include dependency graph for taskalloc.h:



This graph shows which files directly or indirectly include this file:

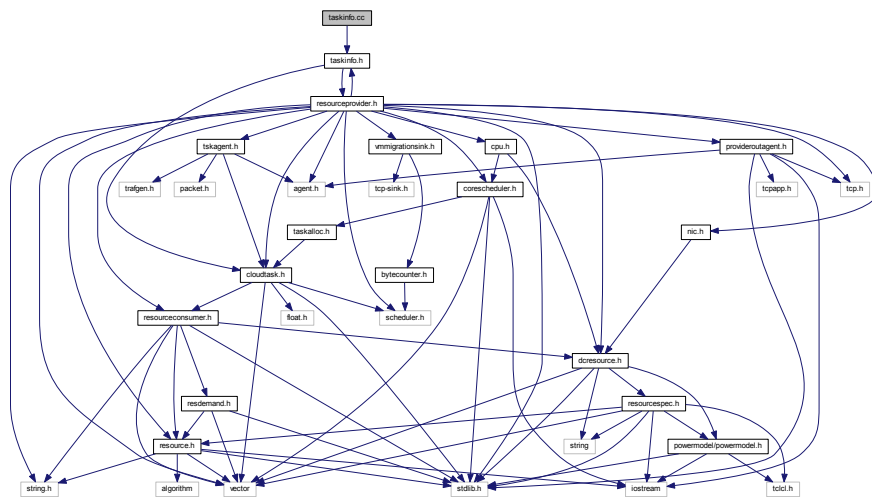


## Classes

- class [TaskAlloc](#)

## 5.68 taskinfo.cc File Reference

```
#include "taskinfo.h"
Include dependency graph for taskinfo.cc:
```

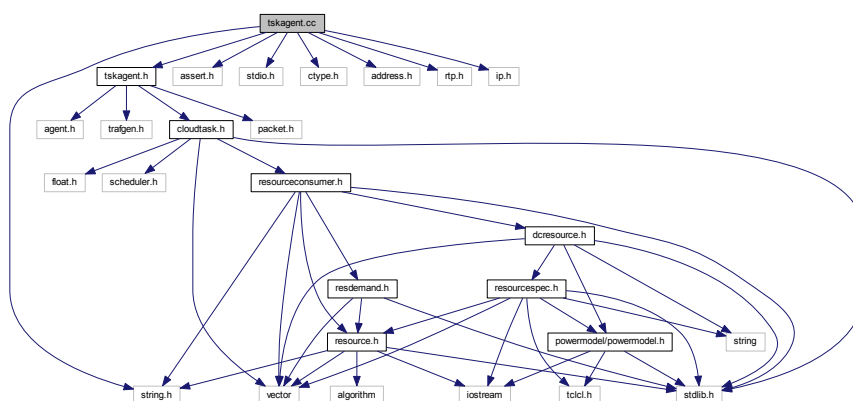


## 5.69 taskinfo.h File Reference

```
#include "cloudtask.h"
#include "resourceprovider.h"
```



Include dependency graph for 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]

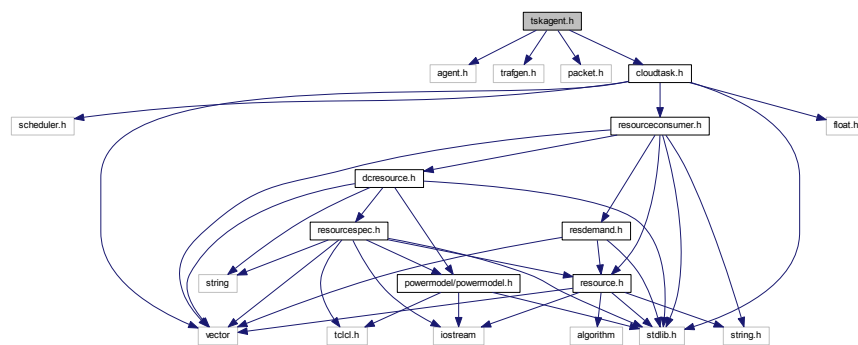
### Initial value:

```
=
                                "@(#) $Header: /cvsroot/nsnam/ns-2/apps/tskagent.cc,v 1.21 2005/08/26
05:05:28 tomh Exp $ (Xerox)"
```

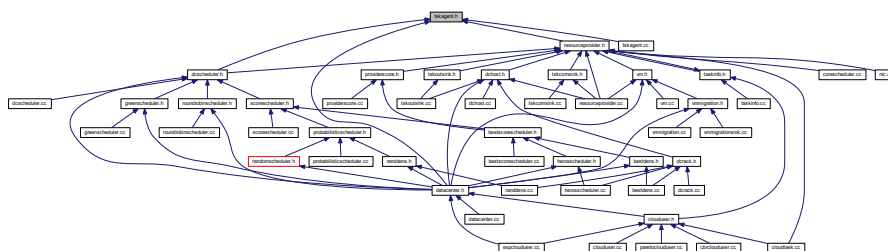
Definition at line 6 of file tskagent.cc.

## 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

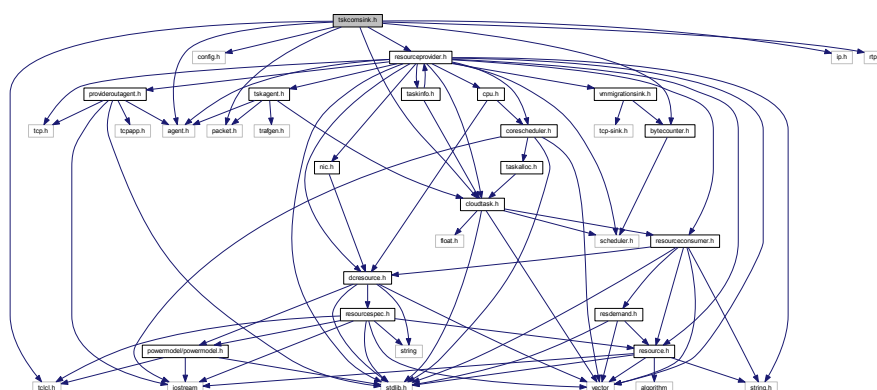
## 5.71.1.1 #define RTP\_M 0x0080

Definition at line 15 of file `tskagent.h`.

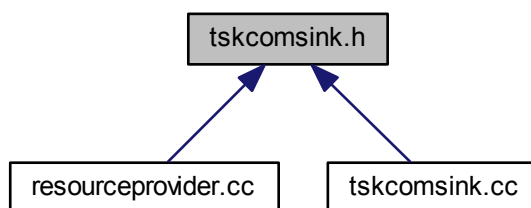


### 5.73 tskcomsink.h File Reference

```
#include <tcclcl.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 tskscomsink.h:
```



This graph shows which files directly or indirectly include this file:



## Classes

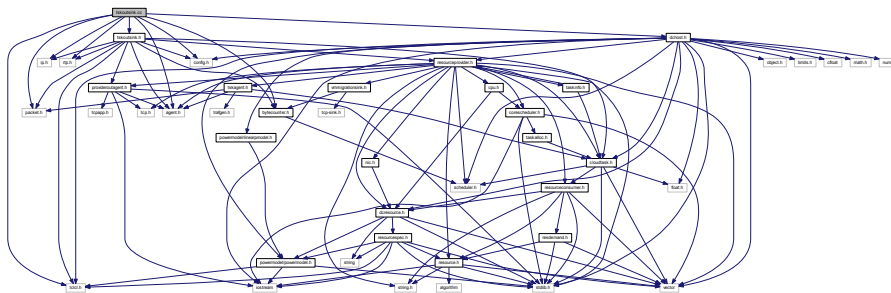
- class TskComSink

## 5.74 tskoutsink.cc File Reference

```
#include <tccl.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]

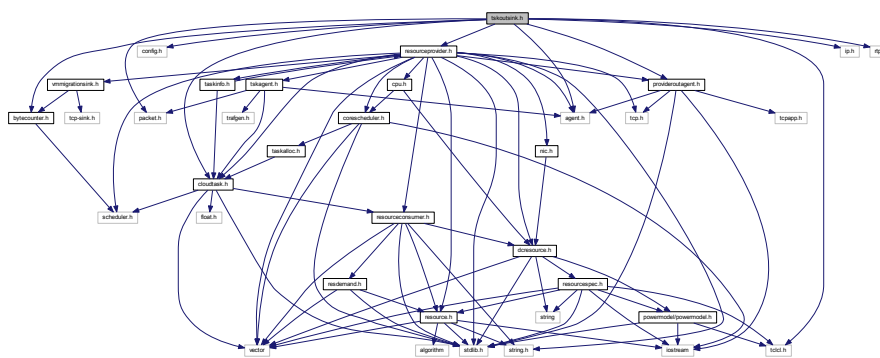
#### Initial value:

```
=
                                "@(#) $Header: /cvsroot/nsnam/ns-2/tools/TskOutSink.cc,v 1.18 2000/09/01
03:04:06 haoboy Exp $ (LBL)"
```

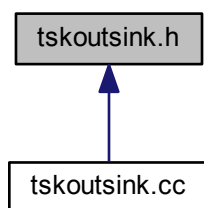
Definition at line 6 of file tskoutsink.cc.

### 5.75 tskoutsink.h File Reference

```
#include <tcclcl.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:



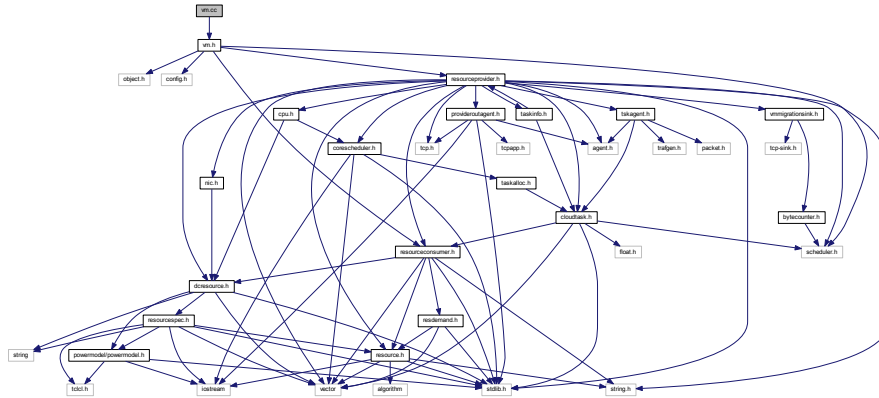
## Classes

- 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"
```

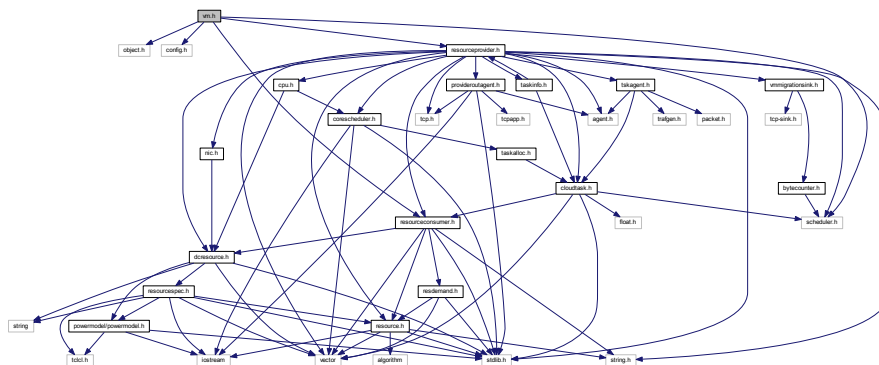
```
#include "config.h"
```

```
#include "scheduler.h"
```

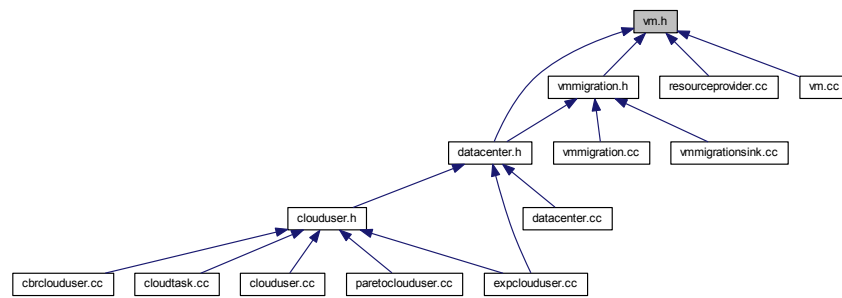
```
#include "resourceprovider.h"
```

```
#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.

```

19      {
20          Ready,
21          Running,
22          Suspended,
23          Stopped,
24          Dead,
25          FirstVmState = Ready,
26          LastVmState = Dead
27  };

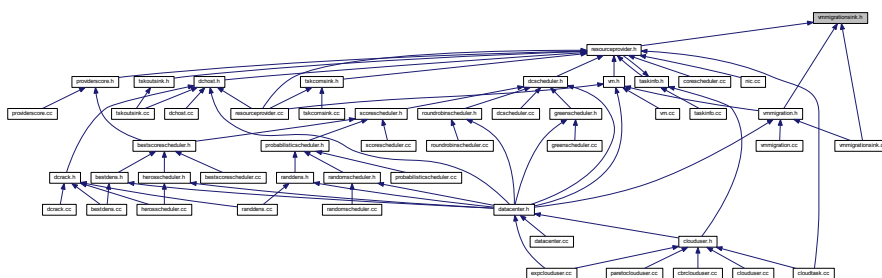
```







This graph shows which files directly or indirectly include this file:



## Classes

- class `VmMigrationSink`



## Index

- \_cancel
    - ResourceProvider, [149](#)
  - \_sched
    - ResourceProvider, [149](#)
- ~BestDENS
  - BestDENS, [11](#)
- ~BestScoreScheduler
  - BestScoreScheduler, [14](#)
- ~ByteCounter
  - ByteCounter, [15](#)
- ~CPU
  - CPU, [48](#)
- ~CloudTask
  - CloudTask, [26](#)
- ~CloudUser
  - CloudUser, [32](#)
- ~CoreScheduler
  - CoreScheduler, [39](#)
- ~DataCenter
  - DataCenter, [55](#)
- ~DcHost
  - DcHost, [67](#)
- ~DcRack
  - DcRack, [73](#)
- ~DcResource
  - DcResource, [79](#)
- ~DcScheduler
  - DcScheduler, [83](#)
- ~GreenScheduler
  - GreenScheduler, [90](#)
- ~HerosScheduler
  - HerosScheduler, [93](#)
- ~LinearPModel
  - LinearPModel, [97](#)
- ~NIC
  - NIC, [103](#)
- ~PerComponentModel
  - PerComponentModel, [111](#)
- ~PoaBuf
  - PoaBuf, [115](#)
- ~PoaBufList
  - PoaBufList, [117](#)
- ~PowerModel
  - PowerModel, [120](#)
- ~ProbabilisticScheduler
  - ProbabilisticScheduler, [123](#)
- ~ProviderOutAgent
  - ProviderOutAgent, [126](#)
- ~ProviderScore
  - ProviderScore, [129](#)
- ~RandDENS
  - RandDENS, [133](#)
- ~RandomScheduler
  - RandomScheduler, [135](#)
- ~Resource
  - Resource, [140](#)
  - ~ResourceConsumer
    - ResourceConsumer, [144](#)
  - ~ResourceProvider
    - ResourceProvider, [148](#)
  - ~ResourceSpec
    - ResourceSpec, [165](#)
  - ~RoundRobinsScheduler
    - RoundRobinsScheduler, [170](#)
  - ~ScoreScheduler
    - ScoreScheduler, [172](#)
  - ~SwitchEnergyModel
    - SwitchEnergyModel, [174](#)
  - ~TaskAlloc
    - TaskAlloc, [182](#)
  - ~TaskInfo
    - TaskInfo, [186](#)
  - ~TskComSink
    - TskComSink, [195](#)
  - ~TskOutSink
    - TskOutSink, [199](#)
  - ~VM
    - VM, [204](#)
  - ~VmMigration
    - VmMigration, [210](#)
  - ~VmMigrationSink
    - VmMigrationSink, [216](#)
- addCapacity
  - ResourceSpec, [166](#)
- addComponent
  - LinearPModel, [98](#)
  - PerComponentModel, [111](#)
  - PowerModel, [121](#)
- addDataCenterPointer
  - CBRCloudUser, [21](#)
  - ExpCloudUser, [85](#)
- addHost
  - DcRack, [74](#)
- addHostPointer
  - DataCenter, [55](#)
- addHostTaskAgentPointer
  - DataCenter, [55](#)
- addPModelPointer
  - DataCenter, [55](#)
- addPowerState
  - ResourceSpec, [166](#)
- addResource
  - DcHost, [68](#)
  - ResourceProvider, [149](#)
  - VM, [204](#)
- addResourceProvider
  - TskComSink, [195](#)
  - TskOutSink, [200](#)
- addResourceSpecificationPointer

- DataCenter, 56
- addUsedCapacity
  - ResourceConsumer, 144
- addVcoreScheduler
  - CoreScheduler, 39
- addVirtualResourceSpecificationPointer
  - DataCenter, 56
- addVM
  - ResourceProvider, 149
- addVmPointer
  - DataCenter, 56
- addVmTaskAgentPointer
  - DataCenter, 56
- arch
  - Resource, 142
- assignTask
  - CoreScheduler, 39
- attachSink
  - ResourceProvider, 150
- attachSource
  - ResourceProvider, 150
- available\_mips\_
  - CoreScheduler, 45
- avgLoad\_
  - DataCenter, 63
- avgLoadMem\_
  - DataCenter, 63
- avgLoadStor\_
  - DataCenter, 63
- avgPower\_
  - DataCenter, 63
- BestDENS, 9
  - ~BestDENS, 11
  - BestDENS, 11
  - calculateScore, 11
  - densLoadFactor, 11
  - epsilon, 12
  - linkLoadFactor, 12
  - scheduleTask, 12
- BestScoreScheduler, 13
  - ~BestScoreScheduler, 14
  - BestScoreScheduler, 14
  - calculateScore, 14
  - scheduleTask, 14
- bestdens.cc, 219
- bestdens.h, 219
- bestscorescheduler.cc, 220
- bestscorescheduler.h, 220
- breceived\_
  - DcRack, 75
- breceived\_old\_
  - DcRack, 75
- burstlen\_
  - ExpCloudUser, 87
  - ParetoCloudUser, 109
- ByteCounter, 15
  - ~ByteCounter, 15
  - ByteCounter, 15
- bytes\_since\_, 16
- getLastBytesSinceTime, 16
- last\_bytes\_since\_, 16
- resetBytesSince, 16
- bytecounter.cc, 221
- bytecounter.h, 222
- bytes
  - PoaBuf, 115
- bytes\_
  - TskComSink, 196
  - TskOutSink, 200
- bytes\_since\_
  - ByteCounter, 16
- CBRCloudUser, 20
  - addDataCenterPointer, 21
  - CBRCloudUser, 21
  - command, 21
  - init, 21
  - interval, 22
  - interval\_, 23
  - maxpkts\_, 23
  - next\_interval, 22
  - random\_, 23
  - rate\_, 23
  - seqno\_, 23
  - start, 22
  - timeout, 22
- CBRCloudUserClass, 23
  - CBRCloudUserClass, 24
  - create, 24
- CPU, 47
  - ~CPU, 48
  - CPU, 48
  - command, 49
  - cores\_schedulers\_, 51
  - getCurrentMIPS, 49
  - getMIPS, 49
  - getNominalMIPS, 49
  - getUtilization, 49
  - nominal\_mips\_, 51
  - print, 50
  - setDVFS, 50
  - setProvider, 50
  - setSpecification, 50
- calculateScore
  - BestDENS, 11
  - BestScoreScheduler, 14
  - HerosScheduler, 93
  - ProbabilisticScheduler, 124
  - RandDENS, 133
  - RandomScheduler, 136
  - ScoreScheduler, 172
- calculateStatistics
  - CloudUser, 33
- cap
  - TaskAlloc, 184
- Capacity, 16
  - Capacity, 17

- getValueRecursive, 18
  - operator double, 18
  - operator+=, 18
  - operator-=, 18
  - operator=, 19
  - value, 19
  - virtual\_capacities, 19
- capacity
  - Resource, 142
- capacity\_location
  - ResDemand, 138
- cbrclouduser.cc, 223
  - class\_cbr\_clouduser, 223
- class\_cbr\_clouduser
  - cbrclouduser.cc, 223
- class\_cpu
  - cpu.cc, 228
- class\_datacenter
  - datacenter.cc, 229
- class\_dchost
  - dchost.cc, 231
- class\_dcrack
  - dcrack.cc, 232
- class\_dcresource
  - dcresource.cc, 234
- class\_exp\_cloud\_user
  - expclouduser.cc, 237
- class\_nic
  - nic.cc, 243
- class\_pareto\_clouduser
  - paretoclouduser.cc, 245
- class\_powermodel
  - linearpmodel.cc, 241
  - percomponentmodel.cc, 246
- class\_resourcespec
  - resourcespec.cc, 265
- class\_switchenergymodel
  - switchenergymodel.cc, 270
- class\_tsk\_comagent
  - tskagent.cc, 274
- class\_tsk\_comsink
  - tskcomsink.cc, 276
- class\_tsk\_outsink
  - tskoutsink.cc, 278
- class\_tsk\_provoutagent
  - provideroutagent.cc, 250
- class\_vm
  - vm.cc, 280
- class\_vm\_migrationsink
  - vmmigrationsink.cc, 284
- class\_vmmigration
  - vmmigration.cc, 282
- classifier\_
  - SwitchEnergyModel, 176
- clear
  - DataCenter, 56
- CloudTask, 24
  - ~CloudTask, 26
- CloudTask, 26
  - deadline\_, 30
  - fail, 27
  - failed\_, 30
  - getDeadline, 27
  - getID, 27
  - getMIPS, 27
  - getOutput, 27
  - handler, 28
  - id\_, 30
  - info\_, 30
  - intercom\_, 30
  - isFinished, 28
  - output\_, 30
  - printCompCapacites, 28
  - releaseAllTaskAllocs, 28
  - removeTaskAlloc, 28
  - scheduled\_, 30
  - setDeadline, 29
  - setID, 29
  - setIntercom, 29
  - setMIPS, 29
  - setOutput, 29
  - started\_, 30
  - task\_allocations\_, 30
  - user\_, 30
- cloudTask
  - TaskAlloc, 184
- CloudUser, 31
  - ~CloudUser, 32
  - calculateStatistics, 33
  - CloudUser, 32
  - createTask, 33
  - dc\_, 35
  - id\_, 35
  - mean\_response\_time\_, 35
  - memory\_, 36
  - postSimulationTestTasks, 34
  - printTasksStatus, 34
  - process\_command, 35
  - random\_tskmips\_, 36
  - randomized\_, 36
  - sd\_response\_time\_, 36
  - setRandomized, 35
  - storage\_, 36
  - taskcounter\_, 36
  - tasks\_info\_, 36
  - tintercom\_, 36
  - toutputsizes\_, 36
  - tskmaxduration\_, 36
  - tskmips\_, 37
  - tsksize\_, 37
  - unfinished\_tasks\_, 37
- cloudtask.cc, 223
  - rcsid, 224
- cloudtask.h, 224
- clouduser.cc, 225
- clouduser.h, 225

- coef\_number
  - LinearPModel, 100
- coefficients
  - LinearPModel, 100
- comm\_potential\_
  - ProviderScore, 129
- command
  - CBRCloudUser, 21
  - CPU, 49
  - DataCenter, 57
  - DcHost, 68
  - DcRack, 74
  - DcResource, 79
  - ExpCloudUser, 85
  - LinearPModel, 98
  - ParetoCloudUser, 107
  - PerComponentModel, 111
  - PowerModel, 121
  - ResourceProvider, 150
  - ResourceSpec, 166
  - SwitchEnergyModel, 175
  - TskComAgent, 190
  - TskComSink, 195
  - TskOutSink, 200
  - VM, 204
  - VmMigration, 210
- computeCurrentRate
  - SwitchEnergyModel, 175
- computeLoad
  - DataCenter, 59
- Computing
  - resource.h, 259
- configureResource
  - DataCenter, 59
- configureVirtualResource
  - DataCenter, 59
- core
  - TaskAlloc, 184
- CoreScheduler, 37
  - ~CoreScheduler, 39
  - addVcoreScheduler, 39
  - assignTask, 39
  - available\_mips\_, 45
  - CoreScheduler, 39
  - current\_mips\_, 45
  - eDVFS\_enabled\_, 45
  - executeTask, 39
  - getAllTasksNumber, 40
  - getAvailableMIPS, 40
  - getCurrentMIPSRecursive, 40
  - getCurrentMIPS, 40
  - getHostScheduler, 40
  - getMostUrgentTaskRate, 40
  - getNominalMIPS, 41
  - getProvider, 41
  - host\_scheduler\_, 45
  - hosted\_vcores\_schedulers, 46
  - nominal\_mips\_, 46
  - provider, 46
  - removeAllocationsFromAssignedList, 41
  - removeCompleted, 42
  - removeFailedTaskAlloc, 42
  - removeFromAssignedList, 42
  - removeTaskAlloc, 43
  - removeVcoreScheduler, 43
  - setComputingRate, 43
  - setDVFS, 43
  - setProvider, 44
  - startTaskExecution, 44
  - tasks\_alloc\_assigned\_, 46
  - tasks\_alloc\_list\_, 46
  - tskAllocFailed\_, 46
  - updateTskComputingRates, 44
  - updateTskList, 44, 45
- cores\_schedulers\_
  - CPU, 51
- corescheduler.cc, 226
- corescheduler.h, 226
- cpu.cc, 227
  - class\_cpu, 228
- cpu.h, 228
- CpuClass, 52
  - CpuClass, 52
  - create, 53
- create
  - CBRCloudUserClass, 24
  - CpuClass, 53
  - DataCenterClass, 65
  - DcHostClass, 72
  - DcRackClass, 77
  - DcResourceClass, 82
  - ExpCloudUserClass, 89
  - LinearPModelClass, 102
  - NicClass, 105
  - POOTrafficClass, 119
  - PerComponentModelClass, 114
  - ProvOutAgentClass, 131
  - ResourceSpecClass, 169
  - SwitchEnergyModelClass, 179
  - TskComAgentClass, 193
  - TskComSinkClass, 198
  - TskOutSinkClass, 202
  - VMClass, 208
  - VmMigrationClass, 214
  - VmMigrationSinkClass, 218
- createNewMigration
  - DataCenter, 60
- createTask
  - CloudUser, 33
- currProcRate\_
  - ResourceConsumer, 145
- current\_mips\_
  - CoreScheduler, 45
- current\_performance
  - ResDemand, 138
- currentLoad\_

- ResourceProvider, 161
- currentLoadMem\_
  - ResourceProvider, 161
- currentLoadNet\_
  - ResourceProvider, 162
- currentLoadStor\_
  - ResourceProvider, 162
- DataCenter, 53
  - ~DataCenter, 55
  - addHostPointer, 55
  - addHostTaskAgentPointer, 55
  - addPModelPointer, 55
  - addResourceSpecificationPointer, 56
  - addVirtualResourceSpecificationPointer, 56
  - addVmPointer, 56
  - addVmTaskAgentPointer, 56
  - avgLoad\_, 63
  - avgLoadMem\_, 63
  - avgLoadStor\_, 63
  - avgPower\_, 63
  - clear, 56
  - command, 57
  - computeLoad, 59
  - configureResource, 59
  - configureVirtualResource, 59
  - createNewMigration, 60
  - DataCenter, 55
  - dcScheduler, 63
  - host\_agent\_list, 63
  - host\_list, 63
  - initiallyConfigureVms, 60
  - migrateVm, 60
  - newhost\_, 63
  - numHostTskAgents\_, 63
  - numVmTskAgents\_, 63
  - power\_model\_list, 63
  - printResourceSpecs, 60
  - receivedTsk, 61
  - resource\_specification\_list, 63
  - scheduleGreen, 61
  - scheduleGreenVmOnly, 61
  - scheduleOnVms\_, 63
  - scheduleRoundRobin, 61, 62
  - setScheduler, 62
  - setVmScheduling, 62
  - tmp\_migration\_, 64
  - tskFailed\_, 64
  - tskSubmitted\_, 64
  - virt\_resource\_specification\_list, 64
  - vm\_agent\_list, 64
  - vm\_list, 64
- DataCenterClass, 64
  - create, 65
  - DataCenterClass, 65
- datacenter.cc, 228
  - class\_datacenter, 229
  - rcsid, 229
- datacenter.h, 229
- dc\_
  - CloudUser, 35
- dc\_exit\_time\_
  - TaskInfo, 188
- DcHost, 66
  - ~DcHost, 67
  - addResource, 68
  - command, 68
  - DcHost, 67
  - eConsumed\_, 70
  - eCurrentConsumption\_, 70
  - eDNS\_enabled\_, 70
  - eLastUpdateTime\_, 70
  - eNominalrate\_, 70
  - eUpdate, 68
  - powerModel, 70
  - print, 69
  - printTasklist, 69
  - rack\_, 70
  - setCurrentConsumption, 69
  - setPowerModel, 69
  - updateEnergyAndConsumption, 70
- DcHostClass, 71
  - create, 72
  - DcHostClass, 72
- DcRack, 72
  - ~DcRack, 73
  - addHost, 74
  - breceived\_, 75
  - breceived\_old\_, 75
  - command, 74
  - DcRack, 73
  - expire, 74
  - hosts\_list\_, 75
  - link\_load, 75
  - qmon\_uplink\_list, 75
  - rack\_id\_, 75
  - stat\_interval, 76
  - updatestats, 74
  - uplink\_B, 76
- DcRackClass, 76
  - create, 77
  - DcRackClass, 77
- DcResource, 77
  - ~DcResource, 79
  - command, 79
  - DcResource, 79
  - getMaxPower, 79
  - getPower, 79
  - getUtilization, 80
  - print, 80
  - ResourceSpec, 167
  - setSpecification, 80
  - specification, 81
  - total\_cap, 81
  - used\_power\_state\_, 81
- DcResourceClass, 81
  - create, 82

- DcResourceClass, 82
- DcScheduler, 82
  - ~DcScheduler, 83
  - DcScheduler, 83
  - scheduleTask, 83
- dcScheduler
  - DataCenter, 63
- dchost.cc, 230
  - class\_dchost, 231
- dchost.h, 231
- dcrack.cc, 232
  - class\_dcrack, 232
- dcrack.h, 232
- dcresource.cc, 233
  - class\_dcresource, 234
- dcresource.h, 234
- dcscheduler.cc, 235
- dcscheduler.h, 235
- Dead
  - vm.h, 281
- deadline\_
  - CloudTask, 30
- deleteTask
  - TaskInfo, 186
- densLoadFactor
  - BestDENS, 11
  - HerosScheduler, 93
  - RandDENS, 133
- detach
  - PoaBufList, 117
- detachSink
  - ResourceProvider, 151
- detachSource
  - ResourceProvider, 151
- due\_time\_
  - TaskInfo, 188
- eActivePorts\_
  - SwitchEnergyModel, 176
- eChassis\_
  - SwitchEnergyModel, 176
- eConsumed\_
  - DcHost, 70
  - SwitchEnergyModel, 176
- eCurrentConsumption\_
  - DcHost, 70
- eCurrentRate\_
  - SwitchEnergyModel, 177
- eDNS\_delay\_
  - SwitchEnergyModel, 177
- eDNS\_enabled\_
  - DcHost, 70
  - SwitchEnergyModel, 177
- eDVFS\_enabled\_
  - CoreScheduler, 45
  - ResourceProvider, 162
  - SwitchEnergyModel, 177
- eEnabled\_
  - SwitchEnergyModel, 177
- eLastSample\_
  - SwitchEnergyModel, 177
- eLastUpdateTime\_
  - DcHost, 70
- eLineCard\_
  - SwitchEnergyModel, 177
- eNominalrate\_
  - DcHost, 70
- ePort\_
  - SwitchEnergyModel, 177
- eSimDuration\_
  - SwitchEnergyModel, 177
- eSimEnd\_
  - SwitchEnergyModel, 177
- eUpdate
  - DcHost, 68
- EVENT\_HANDLING
  - ResourceProvider, 148
- EVENT\_IDLE
  - ResourceProvider, 148
- EVENT\_PENDING
  - ResourceProvider, 148
- em\_
  - SwitchEnergyTimer, 180
- energytimer\_
  - SwitchEnergyModel, 177
- epsilon
  - BestDENS, 12
  - HerosScheduler, 95
  - RandDENS, 134
- estimate
  - LinearPModel, 98
  - PerComponentModel, 111
  - PowerModel, 121
- event\_
  - ResourceProvider, 162
- EventStatus
  - ResourceProvider, 148
- execTime
  - TaskAlloc, 182
- executeTask
  - CoreScheduler, 39
- executedSince\_
  - TaskAlloc, 184
- ExpCloudUser, 84
  - addDataCenterPointer, 85
  - burstlen\_, 87
  - command, 85
  - ExpCloudUser, 85
  - init, 86
  - interval\_, 87
  - next\_interval, 86
  - Offtime\_, 87
  - offtime\_, 87
  - ontime\_, 87
  - rate\_, 87
  - rem\_, 87
  - timeout, 86

- ExpCloudUserClass, 88
  - create, 89
  - ExpCloudUserClass, 89
- expclouduser.cc, 236
  - class\_exp\_cloud\_user, 237
  - rcsid, 237
- expected\_
  - TskComSink, 196
  - TskOutSink, 200
- expire
  - DcRack, 74
  - SwitchEnergyTimer, 180
- fail
  - CloudTask, 27
- failed\_
  - CloudTask, 30
- finalizeDcExitTime
  - TaskInfo, 186
- finalizeMigration
  - VmMigration, 210
- FirstResType
  - resource.h, 259
- FirstVmState
  - vm.h, 281
- getAgent
  - ResourceProvider, 152
- getAllTasksNumber
  - CoreScheduler, 40
- getArch
  - Resource, 141
- getAvailableMIPS
  - CoreScheduler, 40
- getCoreScheduler
  - TaskAlloc, 182
- getCurrentLoad
  - ResourceProvider, 152
- getCurrentMIPSRecursive
  - CoreScheduler, 40
- getCurrentMIPS
  - CPU, 49
  - CoreScheduler, 40
- getDcExitTime
  - TaskInfo, 187
- getDeadline
  - CloudTask, 27
  - TaskAlloc, 182
- getDueTime
  - TaskInfo, 187
- getFreeCap
  - ResourceProvider, 152
- getFreeCapRecursive
  - ResourceProvider, 152
- getHost
  - ResourceProvider, 153
- getHostScheduler
  - CoreScheduler, 40
- getID
  - CloudTask, 27
- getLastBytesSinceTime
  - ByteCounter, 16
- getMIPS
  - CPU, 49
  - CloudTask, 27
  - TaskAlloc, 183
- getMaxPower
  - DcResource, 79
  - LinearPModel, 98
  - PerComponentModel, 112
  - PowerModel, 121
- getMostUrgentTaskRate
  - CoreScheduler, 40
- getNominalMIPS
  - CPU, 49
  - CoreScheduler, 41
- getOutput
  - CloudTask, 27
- getPower
  - DcResource, 79
- getPowerModel
  - ResourceSpec, 166
- getProvider
  - CoreScheduler, 41
- getReleaseTime
  - TaskInfo, 187
- getResTypeUtil
  - ResourceProvider, 153
- getResourceProvider
  - TaskInfo, 187
- getRootHost
  - ResourceProvider, 153
- getServerFinishTime
  - TaskInfo, 187
- getSize
  - ResourceConsumer, 144
- getTask
  - TaskInfo, 187
- getTaskId
  - TaskInfo, 188
- getTotalCap
  - ResourceProvider, 153
- getTskComAgent
  - ResourceProvider, 153
- getType
  - Resource, 141
- getUsedNet
  - ResourceProvider, 154
- getUsedNetRecursive
  - ResourceProvider, 154
- getUtilization
  - CPU, 49
  - DcResource, 80
  - NIC, 103
- getValueRecursive
  - Capacity, 18
- getVmState

- VM, 205
- GreenScheduler, 89
  - ~GreenScheduler, 90
  - GreenScheduler, 90
  - scheduleTask, 91
- greenscheduler.cc, 237
- greenscheduler.h, 237
- handle
  - ResourceProvider, 154
- handler
  - CloudTask, 28
- head\_
  - PoaBufList, 117
- herosComparator
  - herosscheduler.cc, 239
  - herosscheduler.h, 240
- HerosScheduler, 91
  - ~HerosScheduler, 93
  - calculateScore, 93
  - densLoadFactor, 93
  - epsilon, 95
  - HerosScheduler, 93
  - herosTransformation, 93
  - linkLoadFactor, 94
  - performancePerWatt, 94
  - performancePerWattMax, 94
  - scheduleTask, 94
- herosTransformation
  - HerosScheduler, 93
- herosscheduler.cc, 238
  - herosComparator, 239
- herosscheduler.h, 239
  - herosComparator, 240
- host
  - ResourceProvider, 162
- host\_agent\_list
  - DataCenter, 63
- host\_list
  - DataCenter, 63
- host\_scheduler\_
  - CoreScheduler, 45
- hosted\_vcores\_schedulers
  - CoreScheduler, 46
- hosted\_vms\_
  - ResourceProvider, 162
- hosts\_list\_
  - DcRack, 75
- id\_
  - CloudTask, 30
  - CloudUser, 35
  - ResourceProvider, 162
  - VmMigration, 213
- info\_
  - CloudTask, 30
- init
  - CBRCloudUser, 21
  - ExpCloudUser, 86
  - ParetoCloudUser, 107
- initializeMigration
  - VmMigration, 211
- initialized
  - LinearPModel, 100
- initiallyConfigureVms
  - DataCenter, 60
- insert
  - PoaBufList, 117
- intercom\_
  - CloudTask, 30
- interval
  - CBRCloudUser, 22
- interval\_
  - CBRCloudUser, 23
  - ExpCloudUser, 87
  - ParetoCloudUser, 109
- isFinished
  - CloudTask, 28
- isTask
  - ResourceConsumer, 145
- isVM
  - ResourceConsumer, 145
- last\_bytes\_since\_
  - ByteCounter, 16
- last\_packet\_time\_
  - TskComSink, 196
  - TskOutSink, 201
- LastResType
  - resource.h, 259
- lastTrackedBytes\_
  - ProviderOutAgent, 127
- lastTrackedTime\_
  - ProviderOutAgent, 127
- LastVmState
  - vm.h, 281
- LinearPModel, 96
  - ~LinearPModel, 97
  - addComponent, 98
  - coef\_number, 100
  - coefficients, 100
  - command, 98
  - estimate, 98
  - getMaxPower, 98
  - initialized, 100
  - LinearPModel, 97
  - print, 99
  - ready, 100
  - setCoefNumber, 99
  - setCoefficient, 99
  - setCoefficientNumeric, 99
  - updateInit, 100
- LinearPModelClass, 101
  - create, 102
  - LinearPModelClass, 101
- linearpmodel.cc, 241
  - class\_powermodel, 241
- linearpmodel.h, 242

- link\_load
  - DcRack, [75](#)
- linkLoadFactor
  - BestDENS, [12](#)
  - HerosScheduler, [94](#)
  - RandDENS, [133](#)
- MTU
  - ResourceProvider, [162](#)
- maxpkts\_
  - CBRCloudUser, [23](#)
- mean\_response\_time\_
  - CloudUser, [35](#)
- Memory
  - resource.h, [259](#)
- memory\_
  - CloudUser, [36](#)
- migrateVm
  - DataCenter, [60](#)
- migrated\_vm\_
  - VmMigration, [213](#)
- migration\_finished\_
  - VmMigrationSink, [217](#)
- modeled\_components\_
  - PerComponentModel, [112](#)
- NIC, [102](#)
  - ~NIC, [103](#)
  - getUtilization, [103](#)
  - NIC, [103](#)
  - rp\_, [104](#)
  - setRp, [103](#)
- name\_
  - PowerModel, [122](#)
  - ResourceSpec, [167](#)
- nbytes\_
  - PoaBuf, [115](#)
- Networking
  - resource.h, [259](#)
- newhost\_
  - DataCenter, [63](#)
- next\_
  - PoaBuf, [115](#)
- next\_interval
  - CBRCloudUser, [22](#)
  - ExpCloudUser, [86](#)
  - ParetoCloudUser, [107](#)
- nextEvent
  - ResourceProvider, [155](#)
- nic.cc, [242](#)
  - class\_nic, [243](#)
- nic.h, [243](#)
- NicClass, [104](#)
  - create, [105](#)
  - NicClass, [105](#)
- nlost\_
  - TskComSink, [196](#)
  - TskOutSink, [201](#)
- nominal\_mips\_
  - CPU, [51](#)
  - CoreScheduler, [46](#)
- npkts\_
  - TskComSink, [197](#)
  - TskOutSink, [201](#)
- ntasks\_
  - ResourceProvider, [162](#)
- numHostTskAgents\_
  - DataCenter, [63](#)
- numVmTskAgents\_
  - DataCenter, [63](#)
- Offtime\_
  - ExpCloudUser, [87](#)
- offtime\_
  - ExpCloudUser, [87](#)
  - ParetoCloudUser, [109](#)
- on
  - ParetoCloudUser, [108](#)
- on\_
  - ParetoCloudUser, [109](#)
- ontime\_
  - ExpCloudUser, [87](#)
  - ParetoCloudUser, [109](#)
- operator double
  - Capacity, [18](#)
- operator!=
  - resource.h, [259](#)
- operator<
  - ProviderScore, [129](#)
  - resource.h, [260](#)
- operator<=
  - resource.h, [260](#)
- operator>
  - resource.h, [261](#)
- operator>=
  - resource.h, [261](#)
- operator+
  - resource.h, [259](#)
- operator+=
  - Capacity, [18](#)
- operator-
  - resource.h, [260](#)
- operator-=
  - Capacity, [18](#)
- operator=
  - Capacity, [19](#)
  - Resource, [141](#)
- operator==
  - resource.h, [260](#), [261](#)
  - TaskAlloc, [183](#)
- output\_
  - CloudTask, [30](#)
- p1\_
  - ParetoCloudUser, [109](#)
- p2\_
  - ParetoCloudUser, [109](#)
- POOTrafficClass, [118](#)

- create, 119
- POOTrafficClass, 119
- ParetoCloudUser, 105
  - burstlen\_, 109
  - command, 107
  - init, 107
  - interval\_, 109
  - next\_interval, 107
  - offtime\_, 109
  - on, 108
  - on\_, 109
  - ontime\_, 109
  - p1\_, 109
  - p2\_, 109
  - ParetoCloudUser, 107
  - rate\_, 109
  - rem\_, 109
  - rng\_, 109
  - shape\_, 109
  - timeout, 108
- paretoclouduser.cc, 244
  - class\_pareto\_clouduser, 245
  - rcsid, 245
- PerComponentModel, 110
  - ~PerComponentModel, 111
  - addComponent, 111
  - command, 111
  - estimate, 111
  - getMaxPower, 112
  - modeled\_components\_, 112
  - PerComponentModel, 111
  - print, 112
- PerComponentModelClass, 113
  - create, 114
  - PerComponentModelClass, 113
- percomponentmodel.cc, 245
  - class\_powermodel, 246
- percomponentmodel.h, 246
- performancePerWatt
  - HerosScheduler, 94
- performancePerWattMax
  - HerosScheduler, 94
- poa\_
  - TskOutSink, 201
- PoaBuf, 114
  - ~PoaBuf, 115
  - bytes, 115
  - nbytes\_, 115
  - next\_, 115
  - PoaBuf, 115
  - PoaBufList, 115
  - pointer, 115
  - pointer\_, 116
- PoaBufList, 116
  - ~PoaBufList, 117
  - detach, 117
  - head\_, 117
  - insert, 117
  - PoaBuf, 115
  - PoaBufList, 117
  - tail\_, 117
- poabuf\_
  - ProviderOutAgent, 127
- poagent\_
  - ResourceProvider, 162
- pointer
  - PoaBuf, 115
- pointer\_
  - PoaBuf, 116
- postSimulationTestTasks
  - CloudUser, 34
- power\_model\_list
  - DataCenter, 63
- power\_states
  - ResourceSpec, 167
- PowerModel, 119
  - ~PowerModel, 120
  - addComponent, 121
  - command, 121
  - estimate, 121
  - getMaxPower, 121
  - name\_, 122
  - PowerModel, 120
  - print, 121
  - setName, 121
- powerModel
  - DcHost, 70
- powermodel.cc, 247
- powermodel.h, 247
- print
  - CPU, 50
  - DcHost, 69
  - DcResource, 80
  - LinearPModel, 99
  - PerComponentModel, 112
  - PowerModel, 121
  - Resource, 141
  - ResourceProvider, 155
  - ResourceSpec, 166
  - TaskAlloc, 183
  - VM, 205
- printCompCapacites
  - CloudTask, 28
- printResourceSpecs
  - DataCenter, 60
- printTasklist
  - DcHost, 69
  - ResourceProvider, 155
  - VM, 205
- printTasksStatus
  - CloudUser, 34
- ProbabilisticScheduler, 122
  - ~ProbabilisticScheduler, 123
  - calculateScore, 124
  - ProbabilisticScheduler, 123
  - scheduleTask, 124

- probabilisticscheduler.cc, 248
- probabilisticscheduler.h, 248
- process\_command
  - CloudUser, 35
- ProvOutAgentClass, 130
  - create, 131
  - ProvOutAgentClass, 131
- provider
  - CoreScheduler, 46
- provider\_
  - ProviderScore, 129
- ProviderOutAgent, 125
  - ~ProviderOutAgent, 126
  - lastTrackedBytes\_, 127
  - lastTrackedTime\_, 127
  - poabuf\_, 127
  - ProviderOutAgent, 126
  - sendmsg, 126
  - tryToSend, 126
  - updateAgentDataBytes, 127
  - updateTime, 127
- ProviderScore, 128
  - ~ProviderScore, 129
  - comm\_potential\_, 129
  - operator<, 129
  - provider\_, 129
  - ProviderScore, 129
  - score\_, 129
- provideroutagent.cc, 249
  - class\_tsk\_provoutagent, 250
- provideroutagent.h, 250
- providerscore.cc, 251
- providerscore.h, 251
- qmon\_uplink\_list
  - DcRack, 75
- RTP\_M
  - tskagent.h, 275
- rack\_
  - DcHost, 70
- rack\_id\_
  - DcRack, 75
- RandDENS, 131
  - ~RandDENS, 133
  - calculateScore, 133
  - densLoadFactor, 133
  - epsilon, 134
  - linkLoadFactor, 133
  - RandDENS, 133
  - scheduleTask, 133
- randdens.cc, 252
- randdens.h, 253
- random\_
  - CBRCloudUser, 23
- random\_tskmips\_
  - CloudUser, 36
- RandomScheduler, 134
  - ~RandomScheduler, 135
  - calculateScore, 136
  - RandomScheduler, 135
  - scheduleTask, 136
- randomized\_
  - CloudUser, 36
- randomscheduler.cc, 254
- randomscheduler.h, 254
- rate\_
  - CBRCloudUser, 23
  - ExpCloudUser, 87
  - ParetoCloudUser, 109
- rcsid
  - cloudtask.cc, 224
  - datacenter.cc, 229
  - expclouduser.cc, 237
  - paretoclouduser.cc, 245
  - tskagent.cc, 274
  - tskcomsink.cc, 276
  - tskoutsink.cc, 278
- rd
  - TaskAlloc, 184
- Ready
  - vm.h, 281
- ready
  - LinearPModel, 100
- receivedTsk
  - DataCenter, 61
- recv
  - ResourceProvider, 155
  - TskComAgent, 190
  - TskComSink, 196
  - TskOutSink, 200
  - VmMigrationSink, 216
- release\_time\_
  - TaskInfo, 188
- releaseAllTaskAllocs
  - CloudTask, 28
- releaseAllocation
  - ResourceProvider, 156
- rem\_
  - ExpCloudUser, 87
  - ParetoCloudUser, 109
- removeAfterFailure
  - TaskAlloc, 183
- removeAllocationsFromAssginedList
  - CoreScheduler, 41
- removeCompleted
  - CoreScheduler, 42
- removeFailedTaskAlloc
  - CoreScheduler, 42
- removeFromAssginedList
  - CoreScheduler, 42
- removeTaskAlloc
  - CloudTask, 28
  - CoreScheduler, 43
- removeVcoreScheduler
  - CoreScheduler, 43
- removeVM

- ResourceProvider, 157
- res\_demands
  - ResourceConsumer, 145
- res\_model\_
  - ResourceSpec, 168
- res\_provider\_
  - TskComSink, 197
  - TskOutSink, 201
- res\_type
  - resource.h, 259
- ResDemand, 137
  - capacity\_location, 138
  - current\_performance, 138
  - ResDemand, 138
  - supported\_virtual\_resource, 139
- resdemand.cc, 255
- resdemand.h, 256
- resetBytesSince
  - ByteCounter, 16
- Resource, 139
  - ~Resource, 140
  - arch, 142
  - capacity, 142
  - getArch, 141
  - getType, 141
  - operator=, 141
  - print, 141
  - Resource, 140
  - setCapacity, 141
  - setType, 141
  - sortCapacity, 142
  - translateType, 142
  - type, 142
- resource.cc, 257
- resource.h, 257
  - Computing, 259
  - FirstResType, 259
  - LastResType, 259
  - Memory, 259
  - Networking, 259
  - operator!=, 259
  - operator<, 260
  - operator<=, 260
  - operator>, 261
  - operator>=, 261
  - operator+, 259
  - operator-, 260
  - operator==, 260, 261
  - res\_type, 259
  - Storage, 259
- resource\_list
  - ResourceProvider, 162
- resource\_specification\_list
  - DataCenter, 63
- resource\_utilization
  - ResourceProvider, 162
- ResourceConsumer, 143
  - ~ResourceConsumer, 144
  - addUsedCapacity, 144
  - currProcRate\_, 145
  - getSize, 144
  - isTask, 145
  - isVM, 145
  - res\_demands, 145
  - ResourceConsumer, 143
  - setCurrentPerformance, 144
  - setSize, 144
  - size\_, 145
- ResourceProvider, 145
  - \_cancel, 149
  - \_sched, 149
  - ~ResourceProvider, 148
  - addResource, 149
  - addVM, 149
  - attachSink, 150
  - attachSource, 150
  - command, 150
  - currentLoad\_, 161
  - currentLoadMem\_, 161
  - currentLoadNet\_, 162
  - currentLoadStor\_, 162
  - detachSink, 151
  - detachSource, 151
  - eDVFS\_enabled\_, 162
  - EVENT\_HANDLING, 148
  - EVENT\_IDLE, 148
  - EVENT\_PENDING, 148
  - event\_, 162
  - EventStatus, 148
  - getAgent, 152
  - getCurrentLoad, 152
  - getFreeCap, 152
  - getFreeCapRecursive, 152
  - getHost, 153
  - getResTypeUtil, 153
  - getRootHost, 153
  - getTotalCap, 153
  - getTskComAgent, 153
  - getUsedNet, 154
  - getUsedNetRecursive, 154
  - handle, 154
  - host, 162
  - hosted\_vms\_, 162
  - id\_, 162
  - MTU, 162
  - nextEvent, 155
  - ntasks\_, 162
  - poagent\_, 162
  - print, 155
  - printTasklist, 155
  - recv, 155
  - releaseAllocation, 156
  - removeVM, 157
  - resource\_list, 162
  - resource\_utilization, 162
  - ResourceProvider, 148

- scheduleNextExent, 157
- sendTaskOutput, 157
- setAgent, 157
- setTskComAgent, 158
- setTskComSink, 158
- started\_, 163
- status\_, 163
- testSchedulingPossibility, 158
- trySchedulingTsk, 158
- tryToAllocate, 159
- tskComAgent, 163
- tskComSink\_, 163
- tskFailed\_, 163
- updateEnergyAndConsumption, 161
- updateEvent, 161
- updateResTypeUtil, 161
- uplink\_overhead, 163
- useful\_bytes, 163
- vm\_migration\_sinks\_, 163
- vm\_migration\_sources\_, 163
- ResourceSpec, 164
  - ~ResourceSpec, 165
  - addCapacity, 166
  - addPowerState, 166
  - command, 166
  - DcResource, 167
  - getPowerModel, 166
  - name\_, 167
  - power\_states, 167
  - print, 166
  - res\_model\_, 168
  - ResourceSpec, 165
  - setArch, 167
  - setName, 167
  - setPowerModel, 167
- ResourceSpecClass, 168
  - create, 169
  - ResourceSpecClass, 169
- resourceconsumer.cc, 262
- resourceconsumer.h, 262
- resourceprovider.cc, 263
- resourceprovider.h, 263
- resourcespec.cc, 264
  - class\_resourcespec, 265
- resourcespec.h, 265
- rng\_
  - ParetoCloudUser, 109
- RoundRobinsScheduler, 169
  - ~RoundRobinsScheduler, 170
  - RoundRobinsScheduler, 170
  - scheduleTask, 171
- roundrobinscheduler.cc, 266
- roundrobinscheduler.h, 266
- rp\_
  - NIC, 104
  - TaskInfo, 188
- Running
  - vm.h, 281
- SAMPLERATE
  - tskagent.h, 275
- scheduleGreen
  - DataCenter, 61
- scheduleGreenVmOnly
  - DataCenter, 61
- scheduleNextExent
  - ResourceProvider, 157
- scheduleOnVms\_
  - DataCenter, 63
- scheduleRoundRobin
  - DataCenter, 61, 62
- scheduleTask
  - BestDENS, 12
  - BestScoreScheduler, 14
  - DcScheduler, 83
  - GreenScheduler, 91
  - HerosScheduler, 94
  - ProbabilisticScheduler, 124
  - RandDENS, 133
  - RandomScheduler, 136
  - RoundRobinsScheduler, 171
- scheduled\_
  - CloudTask, 30
- score\_
  - ProviderScore, 129
- ScoreScheduler, 171
  - ~ScoreScheduler, 172
  - calculateScore, 172
  - ScoreScheduler, 172
- scorescheduler.cc, 267
- scorescheduler.h, 268
- sd\_response\_time\_
  - CloudUser, 36
- sendTaskOutput
  - ResourceProvider, 157
- sendmsg
  - ProviderOutAgent, 126
  - TskComAgent, 191
- seq\_expected\_
  - VmMigrationSink, 217
- seqno\_
  - CBRCloudUser, 23
  - TskComAgent, 192
  - TskComSink, 197
  - TskOutSink, 201
- server\_finish\_time\_
  - TaskInfo, 188
- setAgent
  - ResourceProvider, 157
- setArch
  - ResourceSpec, 167
- setCapacity
  - Resource, 141
- setClassifier
  - SwitchEnergyModel, 175
- setCoefNumber
  - LinearPModel, 99

- setCoefficient
  - LinearPModel, 99
- setCoefficientNumeric
  - LinearPModel, 99
- setComputingRate
  - CoreScheduler, 43
  - TaskAlloc, 183
- setCoreScheduler
  - TaskAlloc, 183
- setCurrentConsumption
  - DcHost, 69
- setCurrentPerformance
  - ResourceConsumer, 144
- setDVFS
  - CPU, 50
  - CoreScheduler, 43
- setDeadline
  - CloudTask, 29
- setExecTime
  - TaskAlloc, 184
- setHost
  - VM, 206
- setID
  - CloudTask, 29
- setIntercom
  - CloudTask, 29
- setMIPS
  - CloudTask, 29
- setName
  - PowerModel, 121
  - ResourceSpec, 167
- setOutput
  - CloudTask, 29
- setPowerModel
  - DcHost, 69
  - ResourceSpec, 167
- setProvider
  - CPU, 50
  - CoreScheduler, 44
- setRandomized
  - CloudUser, 35
- setResourceProvider
  - TaskInfo, 188
- setRp
  - NIC, 103
- setScheduler
  - DataCenter, 62
- setServerFinishTime
  - TaskInfo, 188
- setSize
  - ResourceConsumer, 144
- setSpecification
  - CPU, 50
  - DcResource, 80
- setTskComAgent
  - ResourceProvider, 158
- setTskComSink
  - ResourceProvider, 158
- setType
  - Resource, 141
- setVmMigration
  - VmMigrationSink, 216
- setVmScheduling
  - DataCenter, 62
- shape\_
  - ParetoCloudUser, 109
- size\_
  - ResourceConsumer, 145
- sortCapacity
  - Resource, 142
- specification
  - DcResource, 81
- start
  - CBRCloudUser, 22
  - SwitchEnergyModel, 175
- startMigration
  - VmMigration, 212
- startTaskExecution
  - CoreScheduler, 44
- started\_
  - CloudTask, 30
  - ResourceProvider, 163
- stat\_interval
  - DcRack, 76
- state
  - VM, 206
- status\_
  - ResourceProvider, 163
- stop
  - SwitchEnergyModel, 175
- Stopped
  - vm.h, 281
- Storage
  - resource.h, 259
- storage\_
  - CloudUser, 36
- supported\_virtual\_resource
  - ResDemand, 139
- Suspended
  - vm.h, 281
- SwitchEnergyModel, 173
  - ~SwitchEnergyModel, 174
  - classifier\_, 176
  - command, 175
  - computeCurrentRate, 175
  - eActivePorts\_, 176
  - eChassis\_, 176
  - eConsumed\_, 176
  - eCurrentRate\_, 177
  - eDNS\_delay\_, 177
  - eDNS\_enabled\_, 177
  - eDVFS\_enabled\_, 177
  - eEnabled\_, 177
  - eLastSample\_, 177
  - eLineCard\_, 177
  - ePort\_, 177

- eSimDuration\_, 177
- eSimEnd\_, 177
- energytimer\_, 177
- setClassifier, 175
- start, 175
- stop, 175
- SwitchEnergyModel, 174
- timeout, 176
- updateEnergy, 176
- SwitchEnergyModelClass, 178
  - create, 179
  - SwitchEnergyModelClass, 178
- SwitchEnergyTimer, 179
  - em\_, 180
  - expire, 180
  - SwitchEnergyTimer, 180
- switchenergymodel.cc, 269
  - class\_switchenergymodel, 270
- switchenergymodel.h, 270
- tail\_
  - PoaBufList, 117
- target\_
  - VmMigration, 213
- task\_
  - TaskInfo, 189
- task\_allocations\_
  - CloudTask, 30
- task\_id\_
  - TaskInfo, 189
- TaskAlloc, 181
  - ~TaskAlloc, 182
  - cap, 184
  - cloudTask, 184
  - core, 184
  - execTime, 182
  - executedSince\_, 184
  - getCoreScheduler, 182
  - getDeadline, 182
  - getMIPS, 183
  - operator==, 183
  - print, 183
  - rd, 184
  - removeAfterFailure, 183
  - setComputingRate, 183
  - setCoreScheduler, 183
  - setExecTime, 184
  - TaskAlloc, 182
  - updateMIPS, 184
- TaskInfo, 185
  - ~TaskInfo, 186
  - dc\_exit\_time\_, 188
  - deleteTask, 186
  - due\_time\_, 188
  - finalizeDcExitTime, 186
  - getDcExitTime, 187
  - getDueTime, 187
  - getReleaseTime, 187
  - getResourceProvider, 187
  - getServerFinishTime, 187
  - getTask, 187
  - getTaskId, 188
  - release\_time\_, 188
  - rp\_, 188
  - server\_finish\_time\_, 188
  - setResourceProvider, 188
  - setServerFinishTime, 188
  - task\_, 189
  - task\_id\_, 189
  - TaskInfo, 186
- taskalloc.cc, 271
- taskalloc.h, 271
- taskcounter\_
  - CloudUser, 36
- taskinfo.cc, 272
- taskinfo.h, 272
- tasks\_alloc\_assigned\_
  - CoreScheduler, 46
- tasks\_alloc\_list\_
  - CoreScheduler, 46
- tasks\_info\_
  - CloudUser, 36
- testSchedulingPossibility
  - ResourceProvider, 158
- timeout
  - CBRCloudUser, 22
  - ExpCloudUser, 86
  - ParetoCloudUser, 108
  - SwitchEnergyModel, 176
- tintercom\_
  - CloudUser, 36
- tmp\_migration\_
  - DataCenter, 64
- total\_cap
  - DcResource, 81
- toutputsiz\_
  - CloudUser, 36
- translateType
  - Resource, 142
- trySchedulingTsk
  - ResourceProvider, 158
- tryToAllocate
  - ResourceProvider, 159
- tryToSend
  - ProviderOutAgent, 126
- tskAllocFailed\_
  - CoreScheduler, 46
- TskComAgent, 189
  - command, 190
  - recv, 190
  - sendmsg, 191
  - seqno\_, 192
  - TskComAgent, 190
- tskComAgent
  - ResourceProvider, 163
- TskComAgentClass, 192
  - create, 193

- TskComAgentClass, 193
- TskComSink, 194
  - ~TskComSink, 195
  - addResourceProvider, 195
  - bytes\_, 196
  - command, 195
  - expected\_, 196
  - last\_packet\_time\_, 196
  - nlost\_, 196
  - npkts\_, 197
  - recv, 196
  - res\_provider\_, 197
  - seqno\_, 197
  - TskComSink, 195
- tskComSink\_
  - ResourceProvider, 163
- TskComSinkClass, 197
  - create, 198
  - TskComSinkClass, 198
- tskFailed\_
  - DataCenter, 64
  - ResourceProvider, 163
- TskOutSink, 198
  - ~TskOutSink, 199
  - addResourceProvider, 200
  - bytes\_, 200
  - command, 200
  - expected\_, 200
  - last\_packet\_time\_, 201
  - nlost\_, 201
  - npkts\_, 201
  - poa\_, 201
  - recv, 200
  - res\_provider\_, 201
  - seqno\_, 201
  - TskOutSink, 199
- TskOutSinkClass, 201
  - create, 202
  - TskOutSinkClass, 202
- tskSubmitted\_
  - DataCenter, 64
- tskagent.cc, 273
  - class\_tsk\_comagent, 274
  - rcsid, 274
- tskagent.h, 275
  - RTP\_M, 275
  - SAMPLERATE, 275
- tskcomsink.cc, 276
  - class\_tsk\_comsink, 276
  - rcsid, 276
- tskcomsink.h, 277
- tskmaxduration\_
  - CloudUser, 36
- tskmips\_
  - CloudUser, 37
- tskoutsink.cc, 277
  - class\_tsk\_outsink, 278
  - rcsid, 278
- tskoutsink.h, 279
- tsksize\_
  - CloudUser, 37
- type
  - Resource, 142
- unfinished\_tasks\_
  - CloudUser, 37
- updateAgentDataBytes
  - ProviderOutAgent, 127
- updateEnergy
  - SwitchEnergyModel, 176
- updateEnergyAndConsumption
  - DcHost, 70
  - ResourceProvider, 161
  - VM, 206
- updateEvent
  - ResourceProvider, 161
- updateInit
  - LinearPModel, 100
- updateMIPS
  - TaskAlloc, 184
  - VM, 206
- updateResTypeUtil
  - ResourceProvider, 161
- updateTime
  - ProviderOutAgent, 127
- updateTskComputingRates
  - CoreScheduler, 44
- updateTskList
  - CoreScheduler, 44, 45
- updatestats
  - DcRack, 74
- uplink\_B
  - DcRack, 76
- uplink\_overhead
  - ResourceProvider, 163
- used\_power\_state\_
  - DcResource, 81
- useful\_bytes
  - ResourceProvider, 163
- user\_
  - CloudTask, 30
- VMClass, 207
  - create, 208
  - VMClass, 207
- value
  - Capacity, 19
- virt\_resource\_specification\_list
  - DataCenter, 64
- virtual\_capacities
  - Capacity, 19
- VM, 203
  - ~VM, 204
  - addResource, 204
  - command, 204
  - getVmState, 205
  - print, 205

- printTasklist, 205
- setHost, 206
- state, 206
- updateEnergyAndConsumption, 206
- updateMIPS, 206
- VM, 204
- vm.cc, 280
  - class\_vm, 280
- vm.h, 280
  - Dead, 281
  - FirstVmState, 281
  - LastVmState, 281
  - Ready, 281
  - Running, 281
  - Stopped, 281
  - Suspended, 281
  - vm\_state, 281
- vm\_agent\_list
  - DataCenter, 64
- vm\_list
  - DataCenter, 64
- vm\_migration\_
  - VmMigrationSink, 217
- vm\_migration\_sender\_
  - VmMigration, 213
- vm\_migration\_sink\_
  - VmMigration, 213
- vm\_migration\_sinks\_
  - ResourceProvider, 163
- vm\_migration\_sources\_
  - ResourceProvider, 163
- vm\_state
  - vm.h, 281
- VmMigration, 208
  - ~VmMigration, 210
  - command, 210
  - finalizeMigration, 210
  - id\_, 213
  - inititalizeMigration, 211
  - migrated\_vm\_, 213
  - startMigration, 212
  - target\_, 213
  - vm\_migration\_sender\_, 213
  - vm\_migration\_sink\_, 213
  - VmMigration, 210
- VmMigrationClass, 213
  - create, 214
  - VmMigrationClass, 214
- VmMigrationSink, 215
  - ~VmMigrationSink, 216
  - migration\_finished\_, 217
  - recv, 216
  - seq\_expected\_, 217
  - setVmMigration, 216
  - vm\_migration\_, 217
  - VmMigrationSink, 216
- VmMigrationSinkClass, 217
  - create, 218
  - VmMigrationSinkClass, 218
- vmmigration.cc, 282
  - class\_vmmigration, 282
- vmmigration.h, 282
- vmmigrationsink.cc, 283
  - class\_vm\_migrationsink, 284
- vmmigrationsink.h, 284