Parallel and Distributed Software Systems : Lab Session 2 Tutorial: Simulation of data centre management systems using DCSim

December, 2nd 2014
pds@intec.ugent.be

Objectives of this tutorial:

- Learn how large-scale data centres are simulated using DCSim
- Learn how to set up a DCSim project
- Introduction to the DCSim_vis visualizer

Before you start:

- In this tutorial and lab session we use the DCSim data centre simulator[1] and the DCSim_vis visualizer. Both have been packaged as jar files and are available on Minerva.
- To make it easier to create, run and modify simulations, start files have been provided. These files are available on Minerva in the "resource allocation" folder in "lab sessions", and contain DCSim, DCSim_vis and a pds package containing start code.
- While DCSim is not restricted to one specific IDE, we use Netbeans during the tutorial and lab session.

1 Data centres

Data centres contain large numbers of server and network resources, and are used to power many Internet and cloud applications. As the use and complexity of these applications increases, the scale of the data centres in which they are executed must increase as well. This however results in many management challenges, including the cost-efficient use of resources, failure management, quality and security challenges, and energy efficiency.

Modern data centres are generally managed by virtualizing the servers, making it possible to host multiple Virtual Machines (VMs) on a single host. Using VMs has multiple advantages: the application contained in the VM is less dependent on the underlying hardware, multiple VMs can be co-located on a single host, the resources allocated to a VMs can change when the application needs change, and VMs can be migrated to other hosts when the need presents itself.

2 DCSim

In recent years, the scale of data centres has increased greatly. Developing and evaluating the algorithms that manage on these large-scale environments by running the algorithms on physical hardware is impossible, because the number of servers needed for a realistic evaluation quickly becomes cost-prohibitive. Therefore, the data centre is typically simulated. This makes it possible to quickly evaluate the performance of algorithms using fewer resources, at the cost of slightly less accurate results. DCSim is an event-based data centre

[1] https://github.com/digs-uwo/dcsim

simulator, which is capable of simulating events within a data centre. The simulator models a collection of hosts within a virtual data centre. VMs are executed on the hosts, and managed by a data centre management system. Various management algorithms, policies, and VM properties can be modified, making it possible to evaluate management algorithms in multiple scenarios.

By default, DCSim models a data centre where multiple applications are deployed. The applications are based on a multi-tier architecture, meaning they consist of multiple interacting VMs. Application workloads are modelled by using utilization traces, which can be selected by the user, and a queuing model is used to determine the duration of request handling. A host energy model is used to model the energy use of hosts, making it possible to determine the energy efficiency of data centre management policies.

2.1 DCSim architecture

While not all of its internal workings need to be understood to use DCSim, it is important to understand how hosts, VMs and applications are defined and how they interact.

2.1.1 The data centre

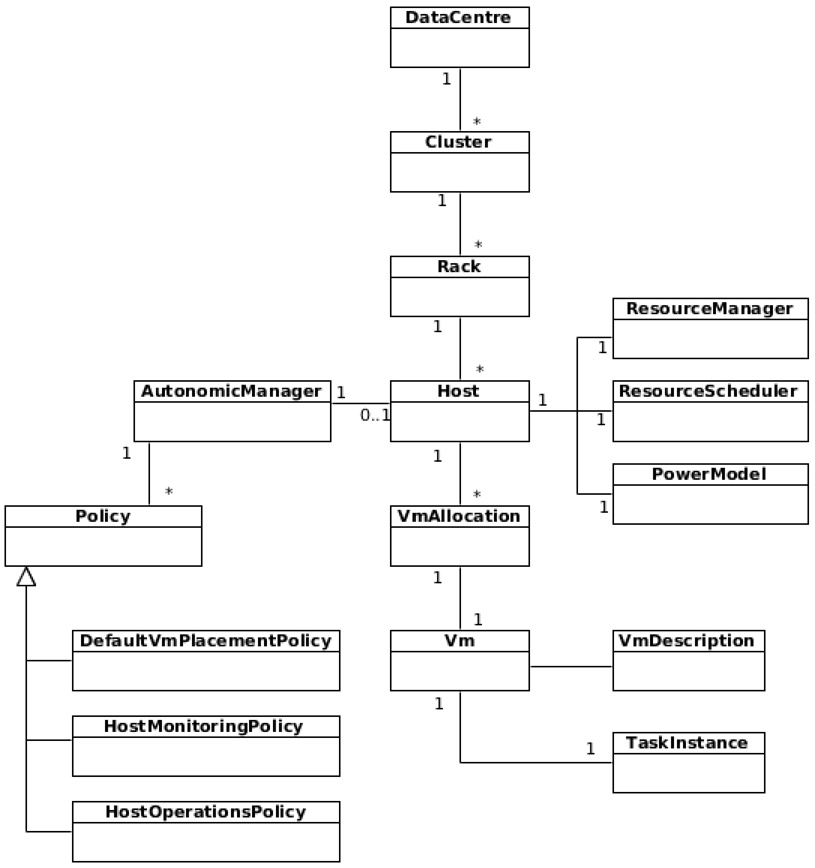


Figure 1: The classes representing the data centre and its state.

Figure 1 shows the various classes that are used to represent a data centre using DCSim. VMs represent a single task which is associated with an application, and are executed on hosts. Multiple VMs may be allocated on a single host. VMs are represented using the `Vm` class. To link a `Vm` with a `Host` on which it is deployed a `VmAllocation` object, representing a VM allocation is used.

The `DataCentre` class represents a single data centre. The data centre may consist of multiple `Cluster`s that each contain multiple `Rack`s. A rack contains multiple `Host`s. Host and data centre management is done using `AutonomicManager`s. An autonomic manager represents a single agent that makes management decisions, possibly by merely forwarding instructions to other management agents. The actions that a manager executes are determined by selected management policies, that implement a `Policy` interface. Every host has an associated autonomic manager that is responsible for managing it. Using policies, the management information associated with a host can also be transmitted to a different autonomic manager, e.g. a centralized management node. DCSim provides three pre-implemented policies:

- The HostMonitoringPolicy is responsible for monitoring the status of a host. The policy will then transmit the status to a chosen autonomic manager.
- The HostOperationsPolicy handles basic host operations such as executing migrations and shutting down the host when it is unused.
- The DefaultVmPlacementPolicy uses a simple VM placement algorithm to allocate VM requests on hosts.

The hosts also have associated ResourceManagers and ResourceSchedulers, which can be modified to change how resources are divided between VMs. Additionally, a PowerModel which models the energy use of servers based on their resource use is provided.

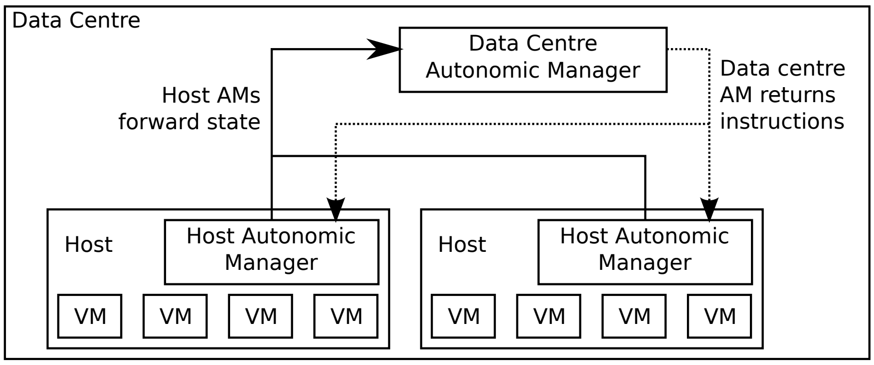


Figure 2: A simple data centre configuration illustrating how host autonomic managers and the data centre autonomic manager can work together to manage the data centre.

Figure 2 illustrates how a simple data centre with two hosts using a centralized management system can be configured. The hosts within the data centre execute VMs, and are managed by a local host autonomic manager. This host forwards management state to the a centralized data centre autonomic manager, which is done using the HostMonitoringPolicy. Based on the received information, the data centre autonomic manager can make management decisions, which it forwards back to the host autonomic managers. The host autonomic managers are then responsible for executing the changes, for which the HostOperationsPolicy is used. Once the host state changes, these changes will then be forwarded to back to the data centre autonomic manager by the HostMonitoringPolicy.

2.1.2 The applications

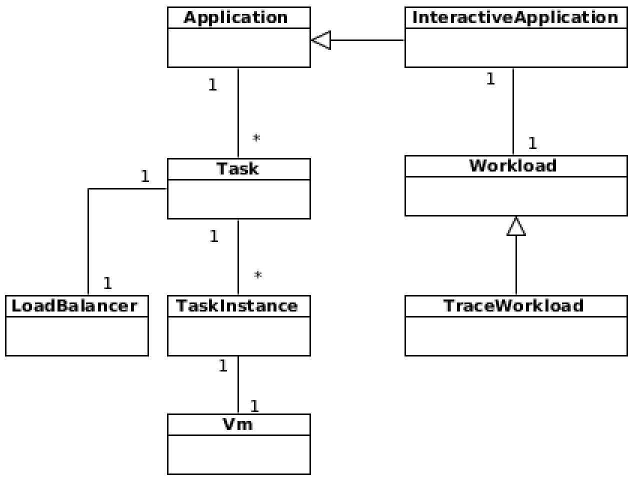


Figure 3: The classes representing Applications and VMs.

The way applications are specified in DCSim is depicted in Figure 3. An Application consists of multiple Tasks, that work together to provide the application. Conceptually, these tasks map to application components, such as for example the various tiers in a three tier web application.

Multiple instances of a task may be needed to service all client requests. In this case, a `LoadBalancer` is used to balance requests between multiple `TaskInstance`s. Every `TaskInstance` is associated with a `Vm`. This `Vm` can then be associated with a physical host as discussed in the previous section.

The `Application` is an abstract class which defines how applications within the data centre must be defined. DCSim provides a default application implementation which models a multi-tier interactive application. This `InteractiveApplication` defines multiple tiers, where requests must go from one tier to the next. The number of requests arriving at the first tier is defined using a `Workload`. A default `TraceWorkload` which determines the load based on an external trace file is provided.

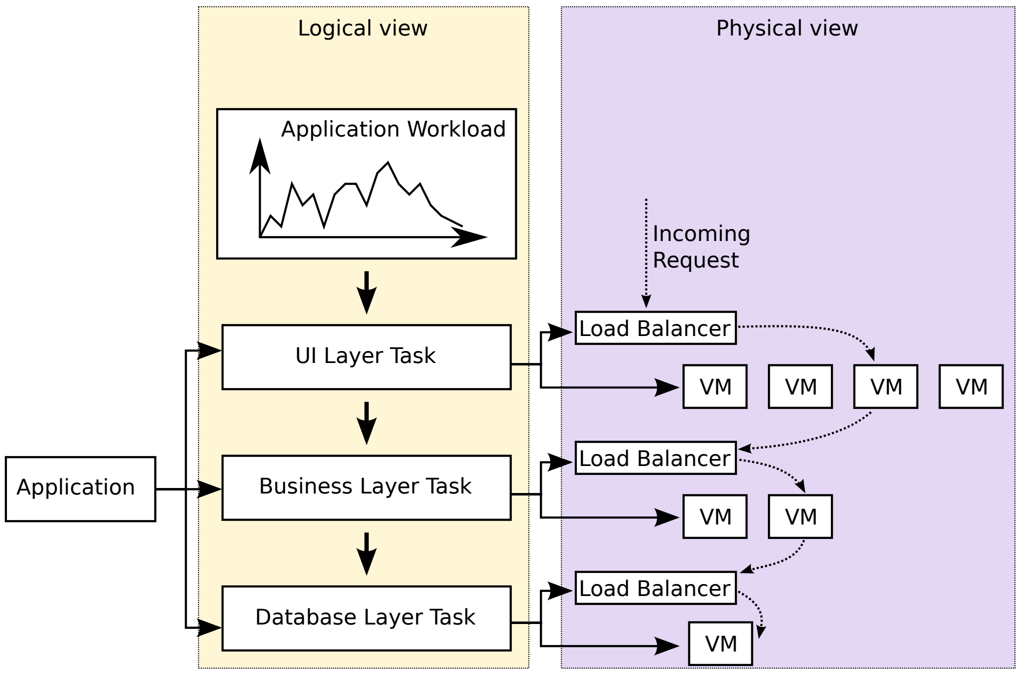


Figure 4: An illustration of how a multi-tier web application can be modeled in DCSim.

Figure 4 shows how a multi-tier web application is represented in DCSim. The application itself consists of three tasks, one for every application layer. Each of these tasks is provided using one or more VMs that are load balanced. The number of incoming requests is determined by an application workload, which may vary throughout time.

2.1.3 DCSim metrics

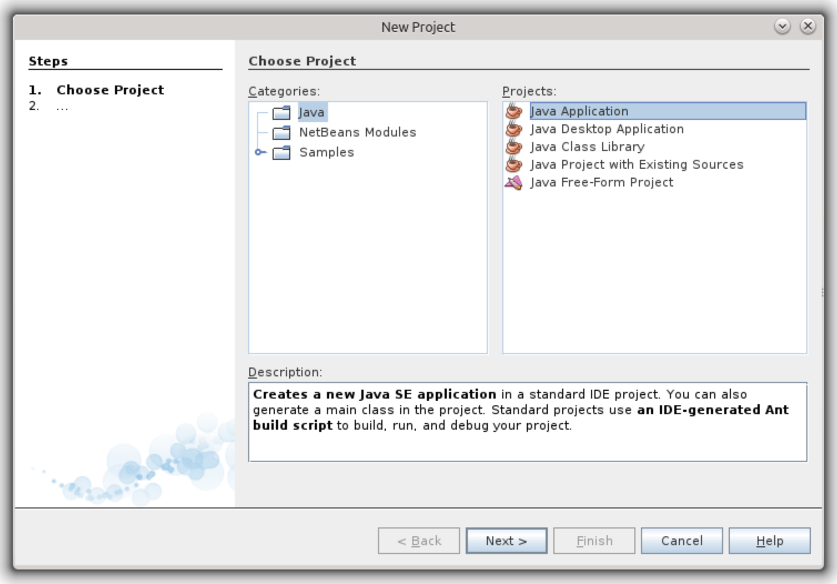
After finishing a simulation, the DCSim simulator shows a simulation summary containing the most important measured metrics. Within this tutorial and lab session, we will focus on a limited subset of these parameters:

- The **mean number of active hosts** represents the number of hosts that are, on average, active during the simulated scenario.
- The **total power consumption** represents the energy use of the hosts during the simulated scenario and is shown in kWh.
- The **mean power consumption** measures the average power consumption during the scenario, and is measured in Watt per second.
- The **CPU underprovision** metric returns the average percentage of CPU demand for which there is insufficient capacity. When this occurs, the applications that have fewer resources than needed will perform perform badly.
- The **maximum response time** metric is indicative of how long it takes for a VM to respond to a request during high load.
- The **failed placement** metric counts the number of application placement requests that failed because there was insufficient host capacity for adding them.

2.2 Creating a DCSim project

In this section, we will how a DCSim project can be created in Netbeans, and how a small demo scenario can be started. Correctly configuring and implementing DCSim simulation is relatively complex because many components must be included and correctly configured. Therefore, start and helper files are provided on Minerva (dcsim_startfiles.zip).

1. Download the DCSim start files from Minerva, and unzip them on the computer.
2. Create a new project in Netbeans by right clicking in the "Projects" panel and selecting "New Project". Select "Java Application" from the "Java" category.



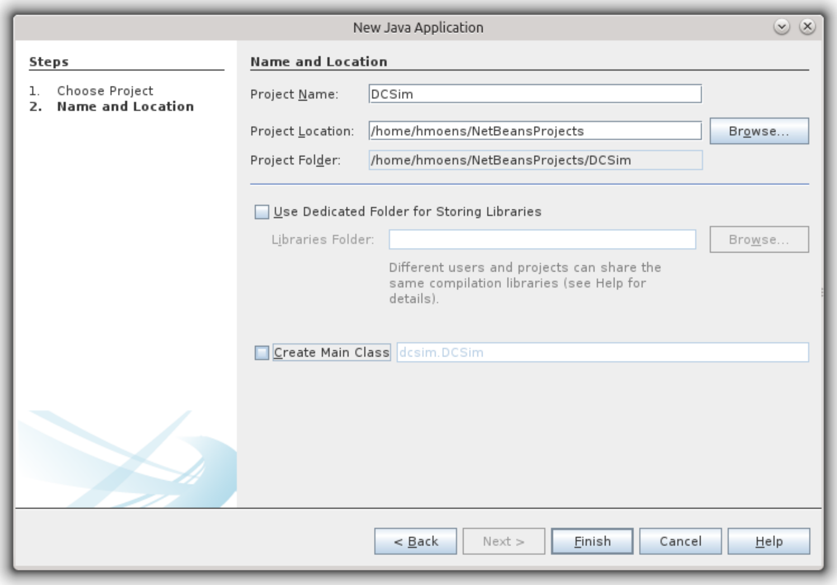
Categories:
- Java
- NetBeans Modules
- Samples

Projects:
- Java Application
- Java Desktop Application
- Java Class Library
- Java Project with Existing Sources
- Java Free-Form Project

Description:
Creates a new Java SE application in a standard IDE project. You can also generate a main class in the project. Standard projects use **an IDE-generated Ant build script** to build, run, and debug your project.

Figure 5: Create a new project.

3. Choose a project name and uncheck the "Create Main Class" checkbox. Click on "Finish" to create the new project.



Project Name: DCSim
Project Location: /home/hmoens/NetBeansProjects
Project Folder: /home/hmoens/NetBeansProjects/DCSim

☐ Use Dedicated Folder for Storing Libraries
Libraries Folder:
Different users and projects can share the same compilation libraries (see Help for details).

☐ Create Main Class dcsim.DCSim

Figure 6: Choose a project name and uncheck the "Create Main Class" checkbox.

4. Next, add commons-math3-3.0.jar, log4j-1.2.16.jar and dcsim2.jar, as dependencies for the project. These files are stored in the "dcsim" directory of the start files. This can be done by right-clicking on the "Libraries" drop-down in the "Projects" panel and selecting "Add JAR/Folder...", and then navigating to these start files on the hard disk.

5. Next, copy the "config" and "traces" directories from the "dcsim" folder in the start files to the project folder as illustrated in Figure 7. This can be done by selecting and copying the files in a file manager and pasting them in the Netbeans window. Note that the folders must end up next to the "src" directory of the project and not in it, meaning the "Files" panel which shows all project files must be used.

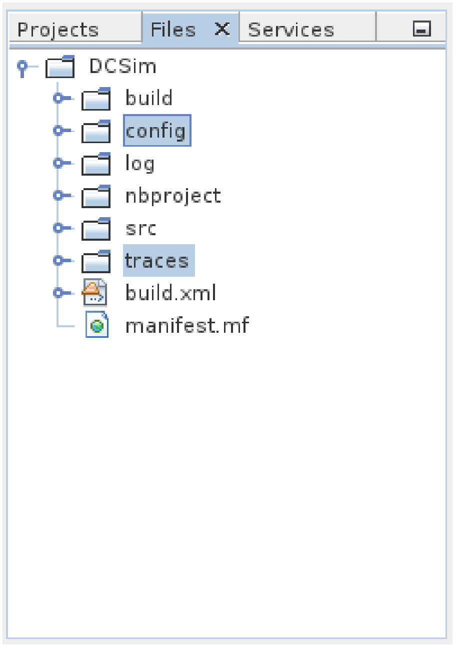


Figure 7: Copying the "config" and "traces" directory to the Netbeans project. This should be done using the "Files" panel in Netbeans.

6. Afterwards, copy the "pds" directory from the "dcsim" directory in the start files to the "src" directory in Netbeans.

7. After this, all of the DCSim dependencies have been added, and a simulation can be started. The project should look like the one shown in below.

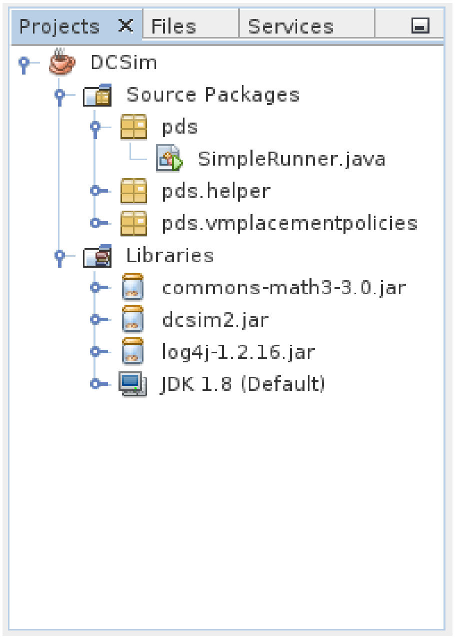


Figure 8: The project after adding the DCSim libraries and the config, traces and pds folders.

2.3 Running a simulation

2.3.1 Running a first simulation

Once the Netbeans project has been created and all the DCSim dependencies have been added, it is possible to start simulations. A simple simulation can be started by right-clicking the SimpleRunner.java class and selecting “Run File”. After a short duration, the simulation report containing various metrics is printed on the console:

```
run:
INFO  edu.uwo.csd.dcsim.core.Simulation            - Starting DCSim
INFO  edu.uwo.csd.dcsim.core.Simulation            - Random Seed: -5217230306070299805
INFO  edu.uwo.csd.dcsim.core.Simulation            -
INFO  edu.uwo.csd.dcsim.core.Simulation            - Completed simulation dynamic-service-spawn-simpleVmPlacementPolicy
INFO  pds.SimpleRunner                             - -- HOSTS --
INFO  pds.SimpleRunner                             - Active Hosts
INFO  pds.SimpleRunner                             -    max: 10.0
INFO  pds.SimpleRunner                             -    mean: 9.943
INFO  pds.SimpleRunner                             -    min: 0.0
INFO  pds.SimpleRunner                             -    util: 32.195%
INFO  pds.SimpleRunner                             -    total util: 32.011%
INFO  pds.SimpleRunner                             - Power
INFO  pds.SimpleRunner                             -    consumed: 212.317kWh
INFO  pds.SimpleRunner                             -    max: 2161.566Ws
INFO  pds.SimpleRunner                             -    mean: 1769.306Ws
INFO  pds.SimpleRunner                             -    min: 0.0Ws
INFO  pds.SimpleRunner                             -    efficiency: 0.0cpu/watt
INFO  pds.SimpleRunner                             -
INFO  pds.SimpleRunner                             - -- APPLICATIONS --
INFO  pds.SimpleRunner                             - CPU Underprovision
INFO  pds.SimpleRunner                             -    percentage: 0.0%
INFO  pds.SimpleRunner                             - SLA
INFO  pds.SimpleRunner                             -   aggregate penalty
INFO  pds.SimpleRunner                             -     total: 0
INFO  pds.SimpleRunner                             -     max: 0.0
INFO  pds.SimpleRunner                             -     mean: 0.0
INFO  pds.SimpleRunner                             -     min: 0.0
INFO  pds.SimpleRunner                             -   per application penalty
INFO  pds.SimpleRunner                             -     mean: 0.0
INFO  pds.SimpleRunner                             -     stdev: 0.0
INFO  pds.SimpleRunner                             -     max: 0.0
INFO  pds.SimpleRunner                             -     95th: 0.0
INFO  pds.SimpleRunner                             -     75th: 0.0
INFO  pds.SimpleRunner                             -     50th: 0.0
INFO  pds.SimpleRunner                             -     25th: 0.0
INFO  pds.SimpleRunner                             -     min: 0.0
INFO  pds.SimpleRunner                             - Response Time
INFO  pds.SimpleRunner                             -     max: 0.003
INFO  pds.SimpleRunner                             -     mean: 0.0
INFO  pds.SimpleRunner                             -     min: 0.0
INFO  pds.SimpleRunner                             - Throughput
INFO  pds.SimpleRunner                             -     max: 598.656
INFO  pds.SimpleRunner                             -     mean: 0.0
INFO  pds.SimpleRunner                             -     min: 141.625
INFO  pds.SimpleRunner                             - Spawning
INFO  pds.SimpleRunner                             -    spawned: 220
INFO  pds.SimpleRunner                             -    shutdown: 46
INFO  pds.SimpleRunner                             -    failed placement: 16
INFO  pds.SimpleRunner                             - Interactive Application Model Algorithm:
INFO  pds.SimpleRunner                             - Schweitzers MVA Approximation
INFO  pds.SimpleRunner                             -
INFO  pds.SimpleRunner                             - -- MANAGEMENT --
INFO  pds.SimpleRunner                             - Messages
INFO  pds.SimpleRunner                             -     edu.uwo.csd.dcsim.management.events.HostStatusEvent: 14325
INFO  pds.SimpleRunner                             - Message BW
INFO  pds.SimpleRunner                             -     edu.uwo.csd.dcsim.management.events.HostStatusEvent: 0.0
INFO  pds.SimpleRunner                             - Migrations
INFO  pds.SimpleRunner                             -
INFO  pds.SimpleRunner                             - -- SIMULATION --
INFO  pds.SimpleRunner                             -    execution time: 9.0s
INFO  pds.SimpleRunner                             -    simulated time: 5.0 days
INFO  pds.SimpleRunner                             -    metric recording start: 0ms
INFO  pds.SimpleRunner                             -    metric recording duration: 5.0 days
INFO  pds.SimpleRunner                             -    application scheduling timed out: 0
BUILD SUCCESSFUL (total time: 9 seconds)
```

2.3.2 The SimpleRunner class

The code of the `SimpleRunner` class is shown below. We will now briefly explain the how the simulation was set up and configured.

```
 1 public class SimpleRunner {
 2   private static Logger logger = Logger.getLogger(SimpleRunner.class);
 3
 4   public static void main(String args[]) {
 5     Simulation.initializeLogging();
 6
 7     SimulationTask task = new DynamicServiceSpawning(
 8         "dynamic-service-spawn-simpleVmPlacementPolicy",
 9         -5217230306702998051,
10         10,
11         "traces/clarknet",
12         new DynamicServiceSpawning.PolicyInstaller() {
13           public void installDatacentrePolicies(AutonomicManager dcAM){
14             dcAM.installPolicy(new VmPlacementPolicy(new FirstFitVmPlacementAlgorithm()));
15           }
16         }
17         );
18
19     task.run();
20
21     task.getMetrics().printDefault(logger);
22   }
23 }
```

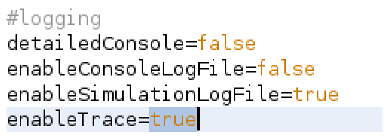
- Lines 2 and 5 initialize the logging system which is used for printing out simulation results.
- A `SimulationTask` of the type `DynamicServiceSpawning` is instantiated on Line 7. This is a helper class defined in the `pds.helper` package which creates and configures a data centre, and creates a simple scenario where a large collection of single-task applications are generated, deployed and removed over time. A host autonomic manager that forwards requests to a centralized data centre autonomic manager is installed on every host, making it possible to change data centre policies by modifying the configuration of the data centre autonomic manager. The arguments for instantiating the `DynamicServiceSpawning` object are the following:
  - A name for the simulation (Line 8), which is printed in the simulation output and which is used to name trace and log files associated with the simulation.
  - A seed for the random generator used in the simulator. This seed is used to create the random generator, and ensures it always results in the same sequence of random numbers. This makes it possible to run the simulation multiple times using the same seed, resulting in an identical scenario (same arrival time, start and stop time, and load for applications) for multiple executions (Line 9).
  - The number of hosts in the simulated data centre (Line 10).
  - The data centre trace used to model the workload of the simulated VMs (Line 11). In this lab session we use the ClarckNet trace, which records two weeks of HTTP logs from an Internet access provider. This trace is stored in the “traces” folder within the project.
  - A policy installer instance which installs policies on the data centre autonomic manager once it is created.
- A VM placement policy is installed in the data centre on Line 14. This policy is used to determine on which host new VMs are allocated. In this example, a `FirstFitVmPlacementAlgorithm` is used which allocates VMs on the first host it can find with sufficient resource capacity. The `FirstFitVmPlacementAlgorithm` must be wrapped in a `DefaultVmPlacementPolicy` helper object which does the heavy lifting when it comes to aggregating host load information and executing the changes which are computed by the placement algorithm. Both the `DefaultVmPlacementPolicy` and `FirstFitVmPlacementAlgorithm` are defined in the `pds.vmplacementpolicies` package.
- The task is started on Line 19, using the `run()` method.
- Finally, the results of the simulation are printed on Line 21, resulting in the console output shown above.

2.4 Visualizing a simulation

While having a simulation summary showing statistics for various metrics is useful, it is hard to observe what is happening within the data centre during the simulation. This makes it thard to interpret and debug unexpected results. The DCSim_vis tool can be used in conjunction with DCSim to visualize the execution results of a simulation based on a provided trace file. This trace file must be generated during the execution of a simulation, and can afterwards be inspected using DCSim_vis. We provide a custom build of the DCSim_vis tool, provided as dcsim_vis.jar in the start files, which provides both a textual and visual interface. The runner allows the user to select a trace file, and generates the necessary files. The visualizer itself is implemented as a locally executed web site, which the dcsim_vis.jar runner automatically opens in Firefox. Firefox must therefore be installed for the visualizer to run correctly.

2.4.1 Starting the visualizer

1. To be able to visualize a simulation, a detailed trace must be generated during the simulation. As these traces require much disk space they are disabled by default. To enable them, the "simulation.properties" file in the "config" directory should be edited[2]. The line "enableTrace=false" should be modified and replaced by the line "enableTrace=true".

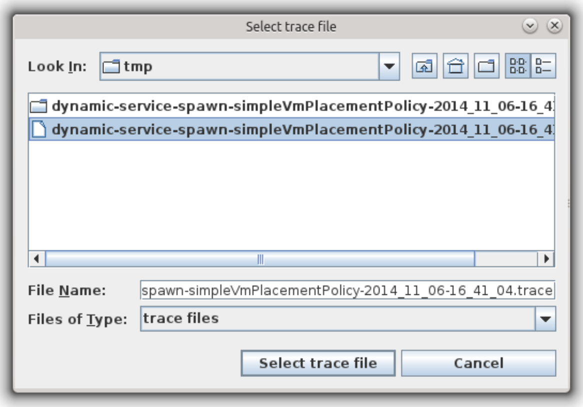


```
#logging
detailedConsole=false
enableConsoleLogFile=false
enableSimulationLogFile=true
enableTrace=true
```

Figure 9: The enableTrace parameter must be set to "true" in the simulation.properties file.

2. Run `SimpleRunner` again. Now that trace files have been enabled, a new file with the .trace extension will be created within the log folder in the project folder. Navigate to this file using the file explorer and note down the full path of the trace file. In this example, `/home/username/NetBeansProjects/DCSim/log/SimpleRunner-2014_11_06-16_41_04.trace` was created.

3. Next, run the dcsim_vis.jar file, and select the trace file in the shown file manager.



Select trace file

Look In: tmp

dynamic-service-spawn-simpleVmPlacementPolicy-2014_11_06-16_4
dynamic-service-spawn-simpleVmPlacementPolicy-2014_11_06-16_4

File Name: spawn-simpleVmPlacementPolicy-2014_11_06-16_41_04.trace
Files of Type: trace files

Figure 10: Select the trace file.

4. The trace file will now be processed automatically. Once the processing has finished, a Firefox window containing DCSim_vis should be shown. It may be necessary to resize the window to correctly see its contents.

5. If Firefox is not started automatically, navigate to the folder containing the trace file, and enter the newly created folder (it has the same name as the trace file, but the ".trace" extension is removed). From this folder navigate to the "html/Animation/" directory, and open "HostsMainAnimation.htm".

[2]The config directory is present in the project directory, and not in the src directory, and will therefore not show up in the "projects" panel in Netbeans. By switching to the "Files" panel these files can be accessed.

Alternatively, the command line interface of dcsim_vis.jar may be used:

1. Open a terminal window and navigate to the folder in the start files containing the dcsim_vis.jar file.
2. Execute the command "java -jar dcsim_vis.jar" with as argument the full path to the trace file.

```
$ java -jar dcsim_vis.jar /home/username/NetBeansProjects/DCSim/log/SimpleRunner-2014_11_06-16
    _41_04.trace
Parsing
Closed all Writers....
Wrote capacities....
Wrote VMs Details events....
Wrote VMs events per hosts events ....
Wrote VMs Stats per host....
VMs Iteration: 01
DONE!
Parsing DONE
Unzipping web files
html/
html/Hosts/
...
source/img/glyphicons-halflings-white.png
Start.html
Unzipping web files DONE
Starting Firefox
Starting Firefox DONE
```

3. The script will then generate the visualization data, and start a new window containing DCSim_vis. It may be necessary to resize the window to correctly see its contents.
4. If Firefox is not started automatically, navigate to the folder containing the trace file, and enter the newly created folder (it has the same name as the trace file, but the ".trace" extension is removed). From this folder navigate to the "html/Animation/" directory, and open "HostsMainAnimation.htm".

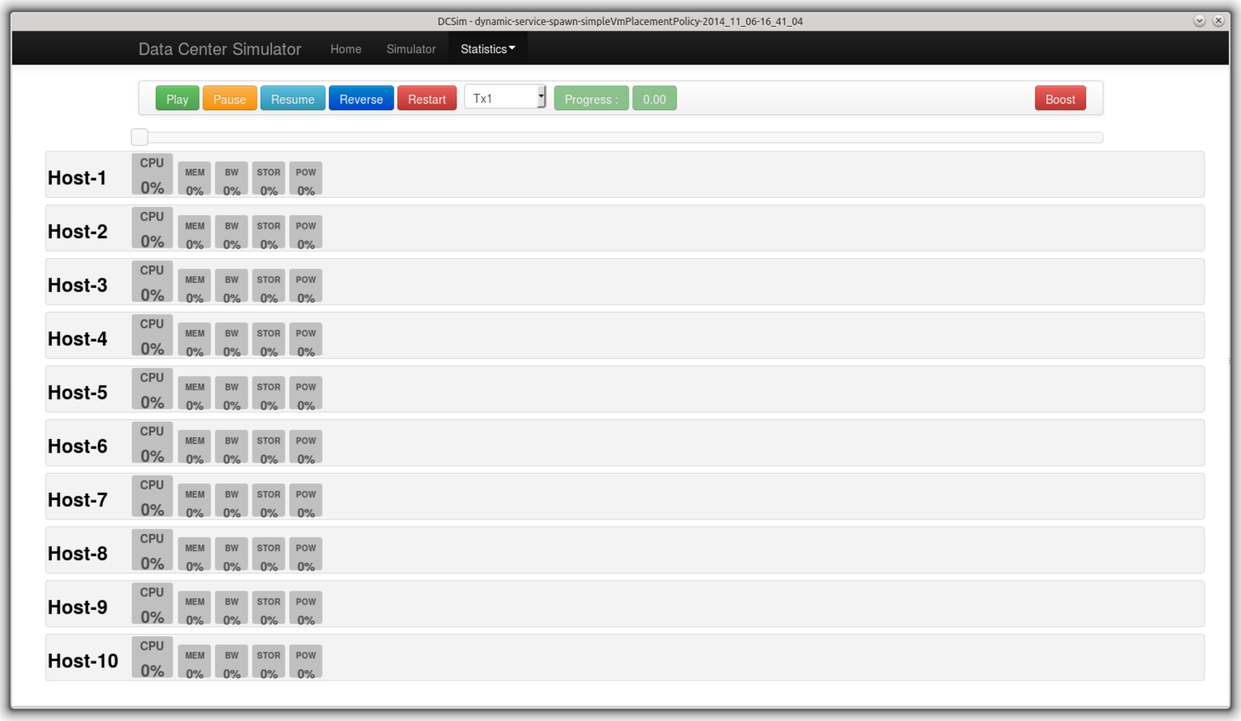


Figure 11: The main DCSim_vis window.

2.4.2 Using the visualizer

Once DCSim_vis starts, it is possible to view the state the simulation had throughout its execution. The playback can be controlled using the controls at the top of the window. The most important controls are listed here:

1. **Play** starts playing the simulation.
2. **Pause** pauses the simulation playback.
3. **Reverse** rewinds the simulation.
4. **Restart** stops the playback of the simulation and starts again from the beginning.
5. The dropdown box, initially showing **Tx1** shows the time multiplier. By default DCSim_vis shows the simulation in real-time: one second within the simulation corresponds to one second in the visualizer. As the `SimpleRunner` simulates 5 days, it is best to increase this multiplier to a much larger value such as **Tx1000**.

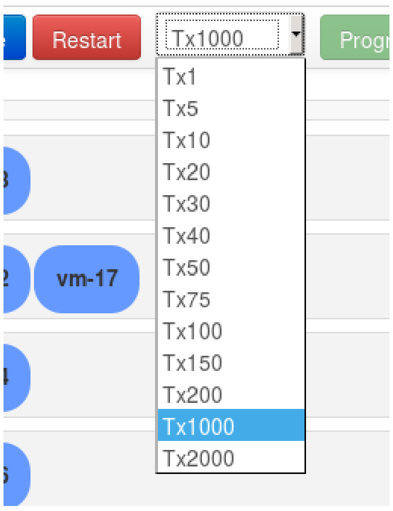
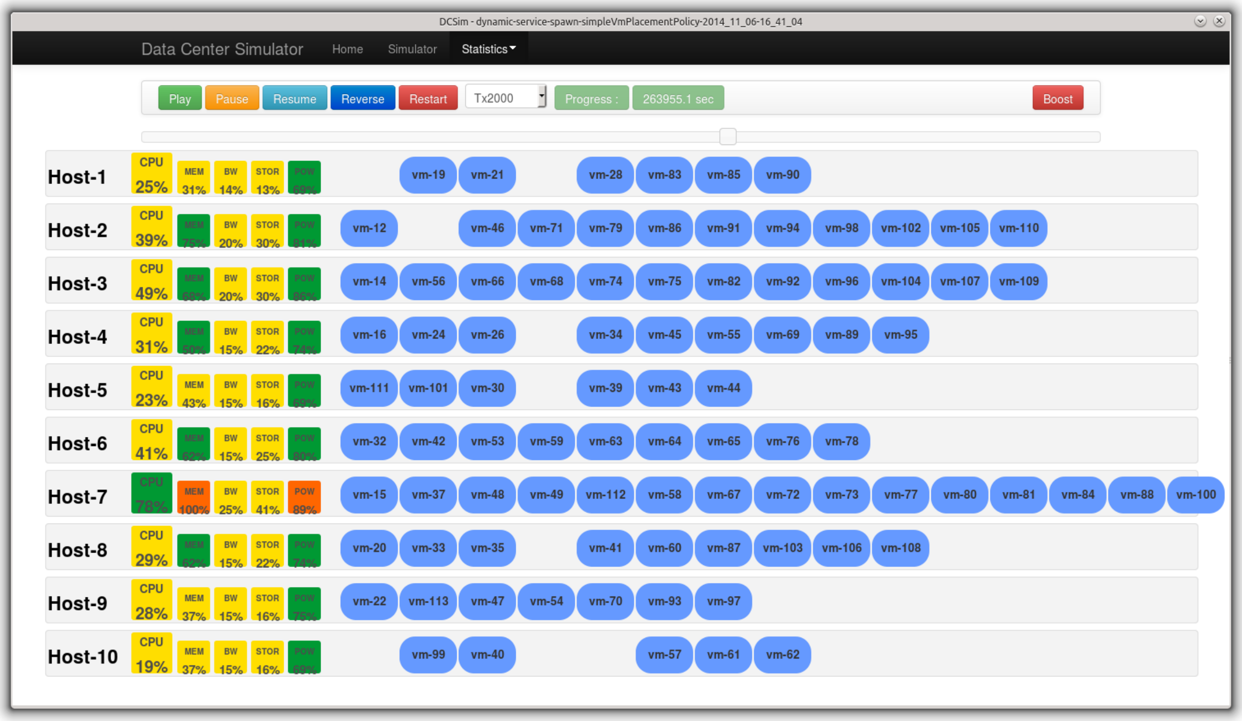


Figure 12: Fast-forwarding in DCSim_vis.

Playing back the simulation, DCSim_vis shows when and where VMs are allocated. Additionally, the load of the various resources of the hosts is shown in different colours.



	CPU	MEM	BW	STOR	POW	VMs
Host-1	25%	31%	14%	13%	69%	vm-19 vm-21 vm-28 vm-83 vm-85 vm-90
Host-2	39%	75%	20%	30%	81%	vm-12 vm-46 vm-71 vm-79 vm-86 vm-91 vm-94 vm-98 vm-102 vm-105 vm-110
Host-3	49%	68%	20%	30%	86%	vm-14 vm-56 vm-66 vm-68 vm-74 vm-75 vm-82 vm-92 vm-96 vm-104 vm-107 vm-109
Host-4	31%	50%	15%	22%	74%	vm-16 vm-24 vm-26 vm-34 vm-45 vm-55 vm-69 vm-89 vm-95
Host-5	23%	43%	15%	16%	69%	vm-111 vm-101 vm-30 vm-39 vm-43 vm-44
Host-6	41%	62%	15%	25%	80%	vm-32 vm-42 vm-53 vm-59 vm-63 vm-64 vm-65 vm-76 vm-78
Host-7	78%	100%	25%	41%	89%	vm-15 vm-37 vm-48 vm-49 vm-112 vm-58 vm-67 vm-72 vm-73 vm-77 vm-80 vm-81 vm-84 vm-88 vm-100
Host-8	29%	62%	15%	22%	74%	vm-20 vm-33 vm-35 vm-41 vm-60 vm-87 vm-103 vm-106 vm-108
Host-9	28%	37%	15%	16%	75%	vm-22 vm-113 vm-47 vm-54 vm-70 vm-93 vm-97
Host-10	19%	37%	15%	16%	69%	vm-99 vm-40 vm-57 vm-61 vm-62

Figure 13: DCSim_vis showing the allocation of VMs on a collection of hosts.

When the resources of a host are greyed out, this shows that the host has been turned off, meaning it is not using any power. If a host resource type is underutilized, it is shown in yellow. When a host resource is over-utilized, it is shown in red. Finally, when a host has high utilization, without being over-utilized, it is shown in green. The various considered resource types are, in the order shown in DCSim_vis: CPU load, memory usage, bandwidth usage, storage power and serve power utilization.