COMP Superscalar

User Manual

Application execution guide

Version: 1.3

August 19, 2015

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

This manual only provides information about how to execute COMPSs aplications. Specifically, it details the options, results and logs of an execution and it provides an overview of the COMPSs tools usage. It is highly recommended to follow this manual with a working COMPSs installation. For this purpose we provide a *COMPSs Virtual Machine* available at `http://compss.bsc.es/` .

For information about the installation process please refer to the *COMPSs Installation Guide* available at `http://compss.bsc.es/` .

For further information about the application development please refer to the *COMPSs User Manual: Application development guide* available at `http://compss.bsc.es/` .

For full COMPSs application examples (codes, execution commands, results, logs, etc.) please refer to the *COMPSs Sample Applications* available at `http://compss.bsc.es/` .

Contents

1 COMP Superscalar (COMPSs) **1**

2 Executing COMPSs applications **2**
2.1 Prerequisites . . . 2
2.2 Runcompss command . . . 2
2.3 Running a COMPSs application . . . 3
2.3.1 Running Java applications . . . 3
2.3.2 Running Python applications . . . 4
2.3.3 Running C/C++ applications . . . 4
2.4 Additional configurations . . . 5
2.4.1 Resources file . . . 6
2.4.2 Project file . . . 7
2.5 Configuration examples . . . 7
2.5.1 Services configuration example . . . 8
2.5.2 Cluster configuration (static resources) example . . . 8
2.5.3 Shared Disks configuration example . . . 9
2.5.4 Cloud configuration (dynamic resources) example . . . 10
2.5.4.1 Cloud connectors: Amazon EC2 . . . 13
2.5.4.2 Cloud connectors: rOCCI . . . 13

3 Results and logs **15**
3.1 Results . . . 15
3.2 Logs . . . 16

4 COMPSs Tools **19**
4.1 Application graph . . . 19
4.2 COMPSs Monitor . . . 19
4.2.1 Service configuration . . . 20
4.2.2 Usage . . . 20
4.2.3 Graphical Interface features . . . 21
4.3 Application tracing . . . 22
4.3.1 Trace Command . . . 22
4.3.2 Application Instrumentation . . . 23
4.3.3 Trace Visualization . . . 24
4.3.3.1 Trace Loading . . . 24
4.3.3.2 Configuration File . . . 24
4.3.3.3 View Adjustment . . . 24
4.3.4 Trace Interpretation . . . 27
4.3.5 Trace Analysis . . . 28
4.3.5.1 Graphical Analysis . . . 28
4.3.5.2 Numerical Analysis . . . 30
4.3.6 Other Trace examples . . . 33
4.4 IDE . . . 34

5 Common Issues 35
5.1 How to debug 35
5.2 Tasks are not executed 35
5.3 Jobs sistematically fail 35

List of Figures

1 Execution of a Java COMPSs application. 4
2 Execution of a Python COMPSs application. 5
3 Execution of a C++ COMPSs application. 5
4 Ouput generated by the execution of the *Simple* java application with COMPSs 15
5 Result comparison between a sequential and a COMPSs execution of the *Hello* java application 16
6 runtime.log generated by the execution of the *Simple* java application . . . 17
7 resources.log generated by the execution of the *Simple* java application . . 18
8 The dependency graph of the SparseLU application 19
9 COMPSs monitoring interface 21
10 Paraver menu 25
11 Trace file 25
12 Paraver view adjustment: Fit window 26
13 Paraver view adjustment: View Event Flags 26
14 Paraver view adjustment: Show info panel 27
15 Paraver view adjustment: Zoom configuration 27
16 Paraver view adjustment: Zoom configuration 27
17 Trace interpretation 28
18 Caption. 29
19 Caption. 29
20 Caption. 30
21 Caption. 30
22 Paraver Menu - New Histogram 30
23 Hmmpfam histogram 31
24 Paraver histogram options menu 32
25 Hmmpfam histogram with the number of bursts 33
26 Examples of complex traces 34

List of Tables

1 Configuration of resources.xml file, tag $< CloudProvider >$ 11
2 Configuration of project.xml file, tag $< Cloud >$ 12
3 Configuration of project.xml file, tag $< Provider >$ 12
4 Properties of the Amazon EC2 connector. 13
5 rOCCI extensions in the project.xml file. 14
6 Configuration of the $< provider > .xml$ templates file. 14

1 COMP Superscalar (COMPSs)

COMP Superscalar (COMPSs) is a programming model which aims to ease the development of applications for distributed infrastructures, such as Clusters, Grids and Clouds. COMP Superscalar also features a runtime system that exploits the inherent parallelism of applications at execution time.

For the sake of programming productivity, the COMPSs model has four key characteristics:

- **Sequential programming:** COMPSs programmers do not need to deal with the typical duties of parallelization and distribution, such as thread creation and synchronization, data distribution, messaging or fault tolerance. Instead, the model is based on sequential programming, which makes it appealing to users that either lack parallel programming expertise or are looking for better programmability.
- **Infrastructure unaware:** COMPSs offers a model that abstracts the application from the underlying distributed infrastructure. Hence, COMPSs programs do not include any detail that could tie them to a particular platform, like deployment or resource management. This makes applications portable between infrastructures with diverse characteristics.
- **Standard programming languages:** COMPSs is based on the popular programming language Java, but also offers language bindings for Python and C/C++ applications. This facilitates the learning of the model, since programmers can reuse most of their previous knowledge.
- **No APIs:** In the case of COMPSs applications in Java, the model does not require to use any special API call, pragma or construct in the application; everything is pure standard Java syntax and libraries. With regard the Python and C/C++ bindings, a small set of API calls should be used on the COMPSs applications.

2 Executing COMPSs applications

2.1 Prerequisites

Prerequisites vary depending on the application's code language: for Java applications the users need to have a **jar archive** containing all the application classes, for Python applications there are no requirements and for C/C++ applications the code must have been previously compiled by using the *buildapp* command.

For further information about how to develop applications for COMPSs please refer to the *COMPSs User Manual: Application development guide* available at the `http://compss.bsc.es/` webpage.

2.2 Runcompss command

All COMPSs applications are executed using the **runcompss** command which is invoked as follows:

```
compss@bsc:~$ runcompss [options] application_name [application_arguments]
```

The application name stands for the fully qualified name of the application in Java, for the path to the *.py* file containing the main program in Python and for the path to the master binary in C/C++.

The application arguments are the arguments that the users' main application recieves. If needed, they can be empty.

The runcompss command allows the users to customize each COMPSs execution by specifying different options. For clarity purposes, parameters are grouped in *Runtime configuration*, *Tools enablers* and *Advanced options*. Users can add any of them to the runcompss call by following the next usage description.

```
compss@bsc:~$ runcompss -h
  Runtime configuration options:
    - -project=<path>                          Path to the project XML file
                                               Default: /opt/COMPSs/Runtime/configuration/
                                               xml/projects/project.xml

    - -resources=<path>                        Path to the resources XML file
                                               Default: /opt/COMPSs/Runtime/configuration/
                                               xml/resources/resources.xml

    - -lang=<name>                             Language of the application (java/c/python)
                                               Default: java

    - -log_level=<level>, - -debug, -d         Set the debug level: off | info | debug
                                               Default: off

  Tools enablers:
    - -graph=<bool>, - -graph, -g              Generation of the complete graph (true/false)
                                               When no value is provided it is set to true
                                               Default: false

    - -tracing=<bool>, - -tracing, -t          Generation of traces (true/false)
                                               When no value is provided it is set to true
                                               Default: false
```

```
  --monitoring=<int>, --monitoring, -m  Period between monitoring samples
                                        (milliseconds)
                                        When no value is provided it is set to 2000
                                        Default: 0

 Advanced options:
  --comm=<path>                         Class that implements the adaptor for
                                        communications
                                        Default: integratedtoolkit.nio.master.
                                        NIOAdaptor

  --library_path=<path>                 Non-standard directories to search for
                                        libraries (e.g. Java JVM library, Python
                                        library, C binding library)
                                        Default: .

  --classpath=<path>                    Path for the application classes / modules
                                        Default: .

  --task_count=<int>                    Only for C/Python Bindings.
                                        Maximum number of different functions/methods
                                        invoked from the application that have been
                                        selected as tasks
                                        Default: 50

  --uuid=<int>                          Preset an application UUID
                                        Default: Automatic random generation
```

2.3 Running a COMPSs application

Before running COMPSs applications it is important to notice that the application files **must** be in the **CLASSPATH**. Thus, when launching a COMPSs application, users can manually pre-set the **CLASSPATH** environment variable or can add the - - *classpath* option to the runcompss command.

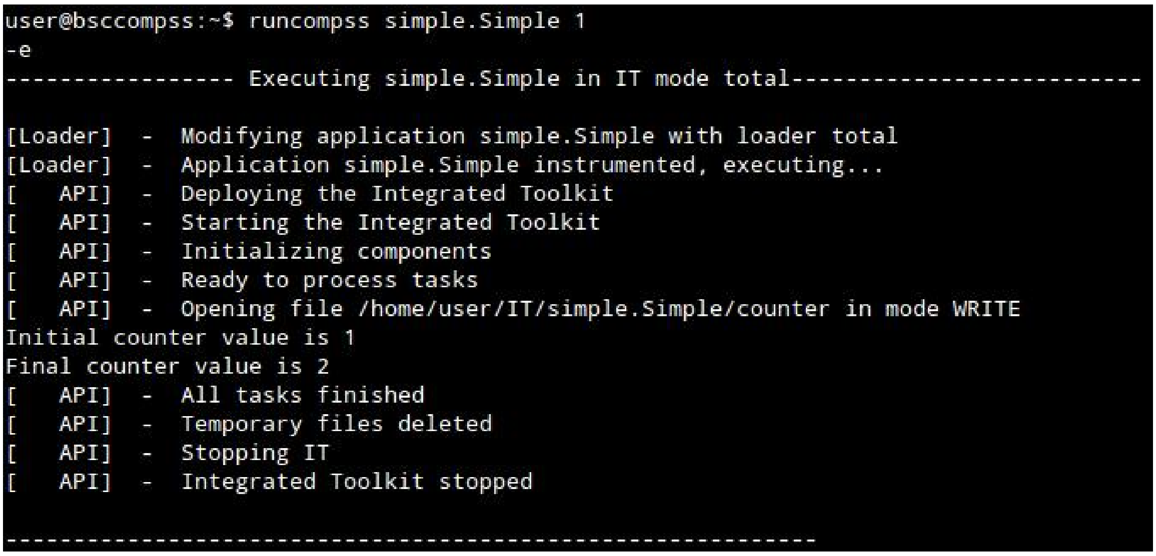
The next three sections provide specific information for launching COMPSs applications in different code languages (Java, Python and C/C++). For clarity purposes we will use the *Simple* application (developed in Java, Python and C++) available at our Virtual Machine or at `https://compss.bsc.es/projects/bar` webpage. This application takes an integer as input parameter and increases it by one unit using a task. For further details about the codes please refer to the *Sample Applications* document available at `http://compss.bsc.es` .

2.3.1 Running Java applications

A Java COMPSs application can be launched through the following command:

```
compss@bsc:~$ cd workspace_java/simple/jar/
compss@bsc:~/workspace_java/simple/jar$ runcompss simple.Simple <initial_number>
```

In this first execution we are taking advantage of the default value of the - -classpath option to automatically add the jar file to the classpath (by moving to the directory which contains the jar file before launching the COMPSs application). However, we can explicitly do this by exporting the **CLASSPATH** variable or by providing the - -classpath value. Next, we provide another two ways to perform the same execution:



```
user@bsccompss:~$ runcompss simple.Simple 1
-e
----------------- Executing simple.Simple in IT mode total--------------------------

[Loader]  -  Modifying application simple.Simple with loader total
[Loader]  -  Application simple.Simple instrumented, executing...
[   API]  -  Deploying the Integrated Toolkit
[   API]  -  Starting the Integrated Toolkit
[   API]  -  Initializing components
[   API]  -  Ready to process tasks
[   API]  -  Opening file /home/user/IT/simple.Simple/counter in mode WRITE
Initial counter value is 1
Final counter value is 2
[   API]  -  All tasks finished
[   API]  -  Temporary files deleted
[   API]  -  Stopping IT
[   API]  -  Integrated Toolkit stopped

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

Figure 1: Execution of a Java COMPSs application.

```
compss@bsc:~$ export CLASSPATH=$CLASSPATH:
                              /home/compss/workspace_java/simple/jar/simple.jar
compss@bsc:~$ runcompss simple.Simple <initial_number>
```

```
compss@bsc:~$ runcompss --classpath=/home/compss/workspace_java/simple/jar/simple.jar
                        simple.Simple <initial_number>
```

2.3.2 Running Python applications

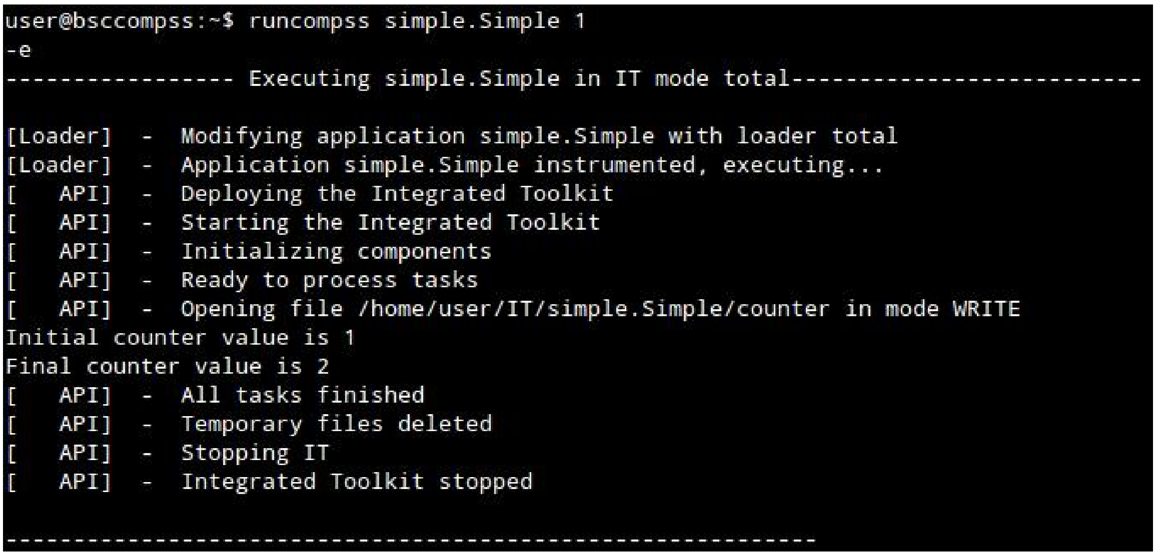
To lauch a Python application with COMPSs users only need to provide the - -*lang=python* option to the runcompss command. Next, we provide an example of a simple python application.

```
compss@bsc:~$ cd workspace_python/simple/
compss@bsc:~/workspace_python/simple$ runcompss --lang=python simple.py <initial_number>
```

2.3.3 Running C/C++ applications

To lauch a C/C++ application with COMPSs users need to compile the C/C++ application by using the *buildapp* command. For further information please refer to the *COMPSs User Manual: Application development guide* document available at our webpage `http://compss.bsc.es` . Once done, to run the C/C++ application with COMPSs, the users only need to provide the *–lang=c* option to the runcompss command. Next, we provide an example of a simple C++ application:

```
compss@bsc:~$ cd workspace_c/simple/
```



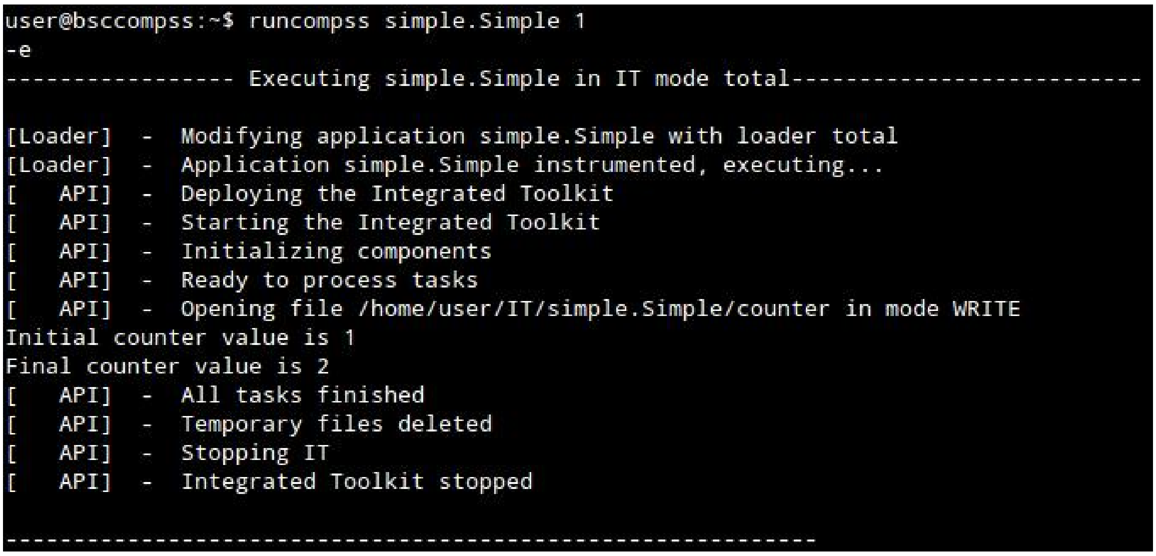
```
user@bsccompss:~$ runcompss simple.Simple 1
-e
---------------- Executing simple.Simple in IT mode total--------------------------

[Loader]  -  Modifying application simple.Simple with loader total
[Loader]  -  Application simple.Simple instrumented, executing...
[   API]  -  Deploying the Integrated Toolkit
[   API]  -  Starting the Integrated Toolkit
[   API]  -  Initializing components
[   API]  -  Ready to process tasks
[   API]  -  Opening file /home/user/IT/simple.Simple/counter in mode WRITE
Initial counter value is 1
Final counter value is 2
[   API]  -  All tasks finished
[   API]  -  Temporary files deleted
[   API]  -  Stopping IT
[   API]  -  Integrated Toolkit stopped

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

Figure 2: Execution of a Python COMPSs application.

```
compss@bsc:~/workspace_c/simple$ runcompss --lang=c simple <initial_number>
```



```
user@bsccompss:~$ runcompss simple.Simple 1
-e
---------------- Executing simple.Simple in IT mode total--------------------------

[Loader]  -  Modifying application simple.Simple with loader total
[Loader]  -  Application simple.Simple instrumented, executing...
[   API]  -  Deploying the Integrated Toolkit
[   API]  -  Starting the Integrated Toolkit
[   API]  -  Initializing components
[   API]  -  Ready to process tasks
[   API]  -  Opening file /home/user/IT/simple.Simple/counter in mode WRITE
Initial counter value is 1
Final counter value is 2
[   API]  -  All tasks finished
[   API]  -  Temporary files deleted
[   API]  -  Stopping IT
[   API]  -  Integrated Toolkit stopped

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

Figure 3: Execution of a C++ COMPSs application.

2.4 Additional configurations

COMPSs has **two** configuration files: *resources.xml* and *project.xml* . These files are meant to provide information about the execution environment and are completely independent from the application.

For each execution users can load the default configuration files or specify their custom configurations by using, respectively, the $--resources =< absolute\_path\_to\_resources.xml >$ and the $--project =< absolute\_path\_to\_project.xml >$ inside the *runcompss* command.

The default files are located under the */opt/COMPSs/Runtime/configuration/xml/* path. Users can edit these two files manually or can take advantage of the *Eclipse IDE* tool developed for COMPSs. For further information about the *Eclipse IDE* please refer to Section 4.4.

Next sections provide more in-depth information about the *resources.xml* and the *project.xml* files, listing and explaining the available tags.

2.4.1 Resources file

The resources file provides information about **all** the available resources. This file should normally be managed by the system administrators. Its full definition schema can be found at */opt/COMPSs/Runtime/configuration/xml/resources/resource*$_s$*chema.xsd*.

The typical strcuture of this file is one entry per available resource defining its name, its capabilities and its requirements. Administrators can define several resource capabilities (see example in next listing) but we would like to underline the importance of **Processor CoreCount**. This capability represents the number of available cores in the resource and it is used to schedule the correct number of tasks into a resource. Thus, it becomes essential to define it accordingly to the number of cores in the physical resource.

```
compss@bsc:~$ cat /opt/COMPSs/Runtime/configuration/xml/resources/resources.xml
<?xml version="1.0" encoding="UTF-8"?>
<ResourceList>
        <Resource Name="localhost">
                <Capabilities>
                        <Host>
                                <TaskCount>0</TaskCount>
                                <Queue>short</Queue>
                                <Queue/>
                        </Host>
                        <Processor>
                                <Architecture>IA32</Architecture>
                                <Speed>3.0</Speed>
                                < CoreCount > 4 < /CoreCount >
                        </Processor>
                        <OS>
                                <OSType>Linux</OSType>
                                <MaxProcessesPerUser>32</MaxProcessesPerUser>
                        </OS>
                        <StorageElement>
                                <Size>8</Size>
                        </StorageElement>
                        <Memory>
                                <PhysicalSize>4</PhysicalSize>
                                <VirtualSize>8</VirtualSize>
                        </Memory>
                        <ApplicationSoftware>
                                <Software>Java</Software>
                        </ApplicationSoftware>
                        <Service/>
                        <VO/>
                        <Cluster/>
                        <FileSystem/>
                        <NetworkAdaptor/>
                        <JobPolicy/>
                        <AccessControlPolicy/>
                </Capabilities>
                <Requirements/>
        </Resource>
</ResourceList>
```

2.4.2 Project file

The project file provides specific execution information. Particularly, it selects and configures in which resources is the application going to be executed. Consequently, the resources that appear in this file are a **subset** of the resources described in the *resources.xml* file. This file should normally be managed by the application users and will usally change from execution to execution. Its full definition schema can be found at */opt/COMPSs/Runtime/configuration/xml/projects/project*$_s$*chema.xsd*.

The typical strcuture of this file is one entry per worker, indicating the the resource name that it uses and some configuration properties (see example in next listing). We emphasize the importance of correctly defining the following entries:

installDir Indicates the path of the COMPSs installation **inside the resource** (not necessarily the same than in the local machine).

User Indicates the username used to connect via ssh to the resource. This user **must** have passwordless access to the resource (for more information check the *COMPSs Installation Manual* available at our website `http://compss.bsc.es`). If left empty COMPSs will automatically try to access the resource with the **same username than the one that lauches the COMPSs main application**.

LimitOfTasks The maximum number of tasks that can be simultaneously scheduled to a resource. Considering that a task can use more than one core but not least, this value must be lower or equal to the number of available cores in the resource.

```
compss@bsc:~$ cat /opt/COMPSs/Runtime/configuration/xml/projects/project.xml
<?xml version="1.0" encoding="UTF-8"?>
<Project>
  <!--Description for any physical node-->
  <Worker Name="localhost">
    < InstallDir > /opt/COMPSs/Runtime/scripts/system/ < /InstallDir >
    <WorkingDir>/tmp/</WorkingDir>
    <! − − < User > user < /User > −− >
    < LimitOfTasks > 4 < /LimitOfTasks >
  </Worker>
</Project>
```

2.5 Configuration examples

In the next subsections we provide specific information about the services, shared disks, cluster and cloud configurations. Moreover there are *project.xml* and *resources.xml* examples for each case that can be used as templates.

2.5.1 Services configuration example

To allow COMPSs applications use WebServices the *resources.xml* can include a special type of resource called *Service*. For each WebService it is necessary to specify its wsdl, its name, its namespace and its port as shown in the following example.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ResourceList>
        <Resource Name="localhost">
                ...
        </Resource>

        <Service wsdl="http://bscgrid05.bsc.es:20390/hmmerobj/hmmerobj?wsdl">
                <Name>HmmerObjects</Name>
                <Namespace>http://hmmerobj.worker</Namespace>
                <Port>HmmerObjectsPort</Port>
        </Service>
</ResourceList>
```

When configuring the *project.xml* file it is necessary to include the service as a worker by adding an special entry indicating only the name and the limit of tasks as shown in the following example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Project>
        <!--Description for any physical node-->
        <Worker Name="localhost">
               ...
        </Worker>

        <Worker Name="http://bscgrid05.bsc.es:20390/hmmerobj/hmmerobj?wsdl">
               <LimitOfTasks>3</LimitOfTasks>
        </Worker>
</Project>
```

2.5.2 Cluster configuration (static resources) example

In order to use external resources to execute the applications, the following steps have to be followed:

1. Install the *COMPSs Worker* package (or the full *COMPSs Framework* package) on all the new resources following the *Installation manual* available at `http://compss.bsc.es` .

2. Set SSH passwordless access to the rest of the remote resources.

3. Create the *WorkingDir* directory in the resource (remember this path because it is needed for the *project.xml* configuration).

4. Deploy the application manually on each resource.

Once these steps are done, users need to configure the *resources.xml* and the *project.xml* files accordingly. On the following lines, we provide examples about configuration files for Grid and Cluster environments, which can serve as a reference

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResourceList>
    <Resource Name="hostname1.domain.es">
        <Capabilities>
            <Host>
                <TaskCount>0</TaskCount>
                <Queue>Short</Queue>
            </Host>
            <Processor>
                <Architecture>x86_64</Architecture>
                <Speed>2.5</Speed>
                <CoreCount>4</CoreCount>
            </Processor>
            <OS>
                <OSType>Linux</OSType>
            </OS>
            <StorageElement>
                <Size>250.0</Size>
            </StorageElement>
            <Memory>
                <PhysicalSize>4.0</PhysicalSize>
            </Memory>
            <ApplicationSoftware>
                <Software>BLAST</Software>
            </ApplicationSoftware>
        </Capabilities>
        <Disks/>
    </Resource>
    <Resource Name="hostname2.domain.es">
  ...
    </Resource>
</ResourceList>
```

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Project>
    <Worker Name="hostname1.domain.es">
        <InstallDir>/opt/COMPSs/Runtime/scripts/system/</InstallDir>
        <WorkingDir>/home/user/</WorkingDir>
        <User>user</User>
        <LimitOfTasks>2</LimitOfTasks>
    </Worker>
    <Worker Name="hostname2.domain.es">
        ...
    </Worker>
</Project>
```

2.5.3 Shared Disks configuration example

Configuring shared disks might reduce the amount of data transfers improving the application performance. To configure a shared disk the users must edit the *resources.xml* indicating how the shared disk is hosted in the master node and how the shared disk is mounted in each worker.

To indicate a shared disk hosted in the master node the *resources.xml* file must include a *Disk* tag describing the disk and the mount point. The following example states that in the master node there is a shared disk labelled *sharedDisk0* mounted on the */sharedDisk* directory.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ResourceList>
    <Disk Name="sharedDisk0">
        <MountPoint>/sharedDisk</MountPoint>
    </Disk>
<ResourceList>
```

On the other side, to declare that a worker has a shared disk mounted the *resources.xml* file must include a *Disk* tag inside the specific worker indicating its name (defined in the master *Disk* tag) and its mount point inside the worker. The following example states that the *sharedDisk0* is mounted on resource *hostname1.domain.es* under the path */home/user/mySharedDisk/* .

```xml
<Resource Name="hostname1.domain.es">
    <Capabilities>
    ...
    </Capabilities>
    <Requirements/>
    <Disks>
        <Disk Name="sharedDisk0">
            <MountPoint>/home/user/mySharedDisk</MountPoint>
        </Disk>
    </Disks>
</Resource>
```

Although, the example only contains the definition of a single shared disk, the *Disks* tag can have multiple disk child nodes.

2.5.4 Cloud configuration (dynamic resources) example

In order to use cloud resources to execute the applications, the following steps have to be followed:

1. Prepare cloud images with the *COMPSs Worker* package or the full *COMPSs Framework* package installed.

2. The application will be deployed automatically during execution but the users need to set up the configuration files to specify the application files that must be deployed.

The COMPSs runtime communicates with the Cloud by means of Cloud connectors. Each connector implements the interaction of the runtime with a given Cloud provider, more precisely by supporting four basic operations: ask for the price of a certain VM in the provider, get the time needed to create a VM, create a new VM and terminate a VM. This design allows connectors to abstract the runtime from the particular API of each provider and facilitates the addition of new connectors for other providers.

In order to add cloud resources the *resources.xml* file can contain one or more $<CloudProvider>$ tags that encompass the information about a particular Cloud provider, associated to a given connector. The tag **must** have an attribute **name** to uniquely identify the provider. Table 1 summarizes the information to be specified by the user inside this tag.

Server	Endpoint of the provider's server

Connector	Class that implements the connector
ImageList • **Image** – **Architecture** – **OSType** – **ApplicationSoftware** ∗ **Software** – **SharedDisks** ∗ **Disk**	Multiple entries of VM templates • VM image – Architeture of the VM image – Operative System installed in the VM image – Multiple entries of software installed in the VM image ∗ Software installed in the VM image – Multiple entries of shared disks mounted in the VM image ∗ Disk description
InstanceTypes • **Resource** – **Capabilities** ∗ **Processor** ∗ **StorageElement** ∗ **Memory**	Multiple entries of resource templates • Instance type offered by the provider – Hardware details of instance type ∗ Architecture and number of available cores ∗ Size in GB of the storage ∗ PhysicalSize, in GB of the available RAM

Table 1: Configuration of resources.xml file, tag $< CloudProvider >$

The *project.xml* complements the information about Cloud providers specified in the *resources.xml* file. This file can contain a $< Cloud >$ tag where to specify a list of providers, each with a $< Provider >$ tag, whose **name** attribute must match one of the providers in the *resources.xml* file. Thus, the *project.xml* file **must** contain a subset of the providers specified in the *resources.xml* file. Table 3 summarizes the information that users need to specify inside the $< Cloud >$ tag and Table 2 summarizes the information that users need to specify inside the $< Provider >$ tag of the *project.xml* file.

InitialVMs	Number of VM to be created at the beginning of the application
minVMCount	Minimum number of VMs available in the computation
maxVMCount	Maximum number of VMs available in the computation
Provider	Multiple entries of Cloud providers

Table 2: Configuration of project.xml file, tag $< Cloud >$

LimitOfVMs	Maximum number of VMs allowed by the provider
Property • Name • Value	Multiple entries of provider-specific properties • Name of the property • Value of the property
ImageList • Image – InstallDir – WorkingDir – User – Package * Source * Target * InstalledSoftware	Multiple entries of VM images available at the provider • VM image – Path of the COMPSs worker scripts in the image – COMPSs working directory in the deployed instances – Account username – Multiple entries of local packages that have to be deployed in new instances * Local path of the package * Path where to deploy the package in the new instance * List of software included in the package
InstanceTypes • Resource	List of resource types that are available in the provider • Resource description

Table 3: Configuration of project.xml file, tag $< Provider >$

The next sections provide a description of each of the currently available connector.

2.5.4.1 Cloud connectors: Amazon EC2

The COMPSs runtime features a connector to interact with the Amazon Elastic Compute Cloud (EC2).

Amazon EC2 offers a well-defined pricing system for VM rental. A total of 8 pricing zones are established, corresponding to 8 different locations of Amazon datacenters around the globe. Besides, inside each zone, several per-hour prices exist for VM instances with different capabilities. The EC2 connector stores the prices of standard on-demand VM instance types (t1.micro, m1.small, m1.medium, m1.large and m1.xlarge) for each zone. Spot instances are not currently supported by the connector.

When the COMPSs runtime chooses to create a VM in the Amazon public Cloud, the EC2 connector receives the information about the requested characteristics of the new VM, namely the number of cores, memory, disk and architecture (32/64 bits). According to that information, the connector tries to find the VM instance type in Amazon that better matches those characteristics and then requests the creation of a new VM instance of that type.

Once an EC2 VM is created, a whole hour slot is paid in advance; for that reason, the connector keeps the VM alive at least during such period, saving it for later use if necessary. When the task load decreases and a VM is no longer used, the connector puts it aside if the hour slot has not expired yet, instead of terminating it. After that, if the task load increases again and the EC2 connector requests a VM, first the set of saved VMs is examined in order to find a VM that is compatible with the requested characteristics. If one is found, the VM is reused and becomes eligible again for the execution of tasks; hence, the cost and time to create a new VM are not paid. A VM is only destroyed when the end of its hour slot is approaching and it is still in saved state.

Table 3 summarizes the provider-specific properties that must be defined in the project.xml file for the Amazon EC2 connector.

Placement	Location of the amazon datacentre to use
Access Key Id	Identifier of the access key of the Amazon EC2 account
Secret Key Id	Identifier of the secret key of the Amazon EC2 account
Key host location	Path to the SSH key in the local host, used to connect to the VMs
KeyPair name	Name of the key pair to use
SecurityGroup name	Name of the security group to use

Table 4: Properties of the Amazon EC2 connector.

2.5.4.2 Cloud connectors: rOCCI

In order to execute a COMPSs application in the cloud, the rOCCI connector has to be configured properly. The connector uses the rOCCI binary client[1] (version newer or equal than 4.2.5) which has to be installed in the node where the COMPSs main application is executed.

This connector needs additional files providing details about the resource templates available on each provider. This file is located under $< COMPSs\_INSTALL\_DIR >$

[1] `https://appdb.egi.eu/store/software/rocci.cli`

$/configuration/xml/templates$ path. Additionally, the user must define the virtual images flavour and instance types offered by each provider; thus, when the runtime asks for the creation of a VM, the connector selects the appropriate image and resource template according to the requirements (in terms of CPU, memory, disk, etc) by invoking the rOCCI client through Mixins (heritable classes that override and extend the base templates).

Table 5 contains the rOCCI specific properties that must be defined in the project.xml file.

Provider	
ca-path	Path to CA certificates directory
user-cred	Path of the VOMS proxy
auth	Authentication method, x509 only supported
owner	Optional. Used by the VENUS-C Job Manager (PMES)
jobname	

Table 5: rOCCI extensions in the project.xml file.

Instance	Multiple entries of resource templates.
Type	Name of the resource template. It has to be the same name than in the previous files
CPU	Number of cores
Memory	Size in GB of the available RAM
Disk	Size in GB of the storage
Price	Cost per hour of the instance

Table 6: Configuration of the $< provider >$ *.xml* templates file.

3 Results and logs

3.1 Results

When executing a user application we consider different type of results:

- **Application Output:** Output generated by the application.
- **Application Files:** Files used or generated by the application.
- **Tasks Output:** Output generated by the tasks invoked from the application.

Regarding the application output, COMPSs will preserve the application output but will add some pre and post output to indicate the COMPSs Runtime state. Figure 3.1 shows the standard output generated by the execution of the simple java application. The green box highlights the application *stdout* while the rest of the output is produced by COMPSs.

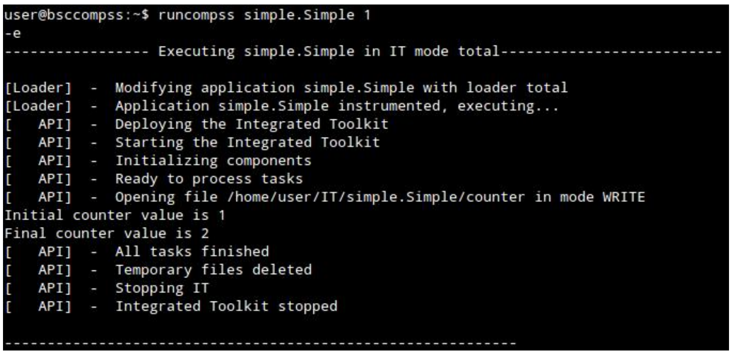


Figure 4: Ouput generated by the execution of the *Simple* java application with COMPSs

Regarding the application files, COMPSs **does not modify** any of them and thus, the results obtained by executing the application with COMPSs are the same than the ones generated by the sequential execution of the application.

Regarding the task output, COMPSs does introduce some modifications due to the fact that tasks can be executed in remote machines. Considering that we call a job the execution of a task in a given resource, after the execution, COMPSs stores the *stdout* and the *stderr* of each job inside the */home/$USER/.COMPSs/$APPNAME/$EXEC_NUMBER/jobs/* directory.

Figure 5 shows an example of the results obtained from the execution of the *Hello* java application. At rigth we provide the sequential execution of the application and, at left, its equivalent COMPSs execution. Notice that, the sequential execution produces the Ḧello World!m̈essage in the *stdout* while the COMPSs execution stores the message inside the *job1_NEW.out* file.

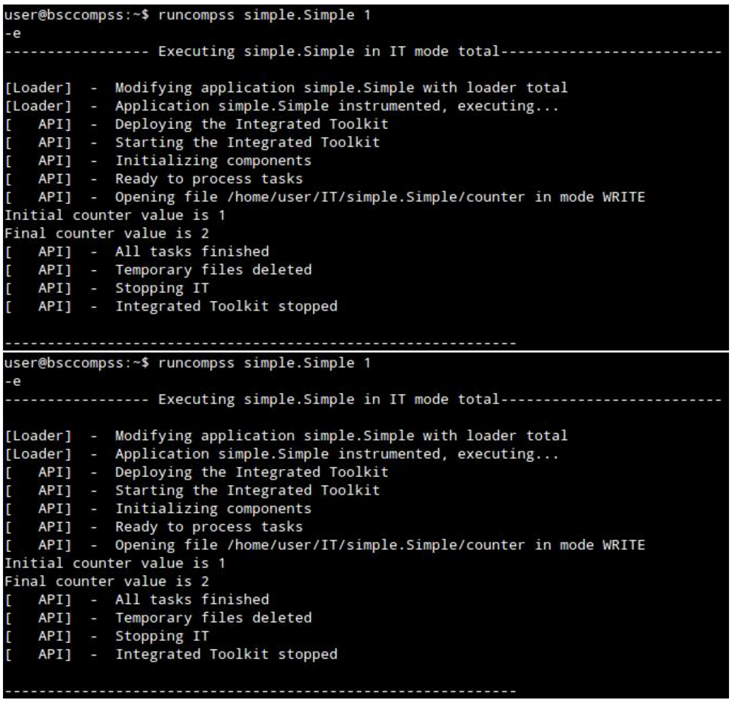


Figure 5: Result comparison between a sequential and a COMPSs execution of the *Hello* java application

3.2 Logs

COMPSs includes three log levels for running applications but users can modify them or add more levels by editing the logger files under the */opt/COMPSs/Runtime/configuration/log/* folder. Any of these log levels can be selected by adding the $--log\_level=<debug|info|off>$ flag to the runcompss command. The default value is *off*.

The logs generated by the execution number NUM_EXEC of the application APP by the user USER are stored under $/home/\$USER/.COMPSs/\$APP/\$NUM\_EXEC/$ folder (from this point on: **base log folder**). The execution number is automatically tracked to prevent the loss of data from previous executions, but users do not need to take care of this value.

When running COMPSs with **log level off** only the errors are reported. This means that the *base log folder* will be empty if no error has occured. If somehow the application failed, a *jobs* folder will appear containing the *stdout* and the *stderr* of each failed job. Figure 3.2 shows the logs generated by the execution of the simple java application (without errors) in **off** mode.

```
compss@bsc:~$ ls -l ~/.COMPSs/simple.Simple_01/
```

When running COMPSs with **log level info** the *base log folder* will contain two files (**runtime.log** and **resources.log**) and one folder (*jobs*). The **runtime.log** file contains the execution information retrieved from the master resource; including the file transfers and the job submission details. The **resources.log** file contains information about the available resources such as the number of processors of each resource (slots), the information about running or pending tasks in the resource queue and the created and destroyed resources. The *jobs* folder contains two files per submitted job; one for the *stdout* and another for the *stderr*. As an example, Figure 3.2 shows the logs generated by the same execution than the previous case but with **info** mode.

```
compss@bsc:~$ tree ~/.COMPSs/simple.Simple_01/
```

The runtime.log and resources.log are quite large files and, thus, should be only checked by advanced users. For an easier interpretation of these files the COMPSs Framework includes a monitor tool. For further information about the COMPSs Monitor please check Section 4.2.

Figures 3.2 and 3.2 provide the content of these two files generated by the execution of the *Simple* java application.

./Sections/3_Running_Apps/Figures/simple_runtimelog.jpeg

Figure 6: runtime.log generated by the execution of the *Simple* java application

./Sections/3_Running_Apps/Figures/simple_resourceslog.jpeg

Figure 7: resources.log generated by the execution of the *Simple* java application

Running COMPSs with **log level debug** generates the same files as the info log level but with more detailed information. Moreover, the COMPSs Runtime state is printed out on the *stdout*. The runtime.log and the resources.log files generated in this mode can be **extremely large**. Consequently, the users should take care of their quota and manually erase these files if needed.

Furthermore, when running other runcompss flags (such as monitoring or tracing) additional folders will appear inside the *base log folder*. The meaning of the files inside these folders is explained in Section 4.

4 COMPSs Tools

4.1 Application graph

At the end of the application execution a dependency graph can be generated representing the order of execution of each type of task and their dependencies. To allow the final graph generation the runcompss command must be run with the *-g* flag and the final graph will appear at the $base\_log\_folder/monitor/complete\_graph$.dot at the end of the execution.

Figure 4.1 shows a dependency graph example of a *SparseLU* java application. The graph can be visualized by running the following command:

```
compss@bsc:~$ gengraph ~/.COMPSs/sparseLU.arrays.SparseLU_01/monitor/complete_graph.dot
```

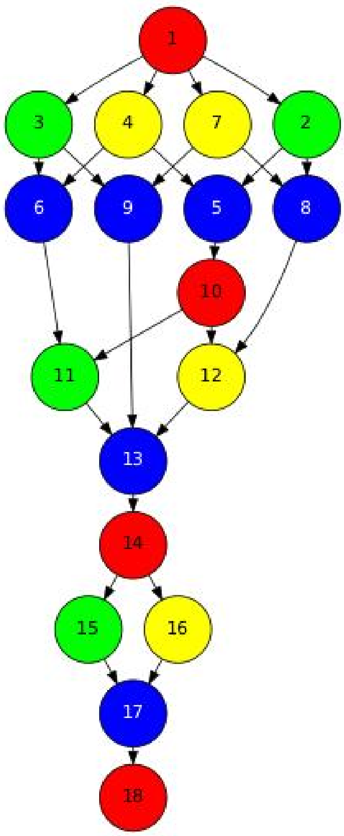


Figure 8: The dependency graph of the SparseLU application

4.2 COMPSs Monitor

The COMPSs Framework exposes a Web Service with a graphical interface that can be used to monitor either finished and running applications. COMPSs Monitor is installed as a service and can be easily managed by running any of the following commands:

```
compss@bsc:~$ sudo service compss-monitor usage
Usage: /usr/sbin/service compss-monitor
          {start | stop | reload | restart | try-restart | force-reload | status}
```

4.2.1 Service configuration

The COMPSs Monitor service can be configured by editing the */opt/COMPSs/Tools/monitor/apache-tomcat/conf/compss-monitor.conf* file which contains one line per property:

- *IT_MONITOR* Default directory to retrieve monitored applications.
- *COMPSs_MONITOR_PORT* Port where to run the compss-monitor web service.
- *COMPSs_MONITOR_TIMEOUT* Web page timeout between browser and server.

By default, the *IT_MONITOR* points to the *.COMPSs* folder inside the *root* user, opens the web service in the port 8080 and has a 20s timeout.

4.2.2 Usage

In order to use the COMPSs Monitor users need to start the service

```
compss@bsc:~$ sudo service compss-monitor start
```

And use a web browser to open the specific URL:

```
compss@bsc:~$ firefox http://localhost:8080/compss-monitor &
```

The COMPSs Monitor allows to monitor applications from different users and thus, users need to login first to grant access to their applications. Once done, as shown in Figure 4.2.2, the users can select any of their executed or on-going COMPSs applications and display it.

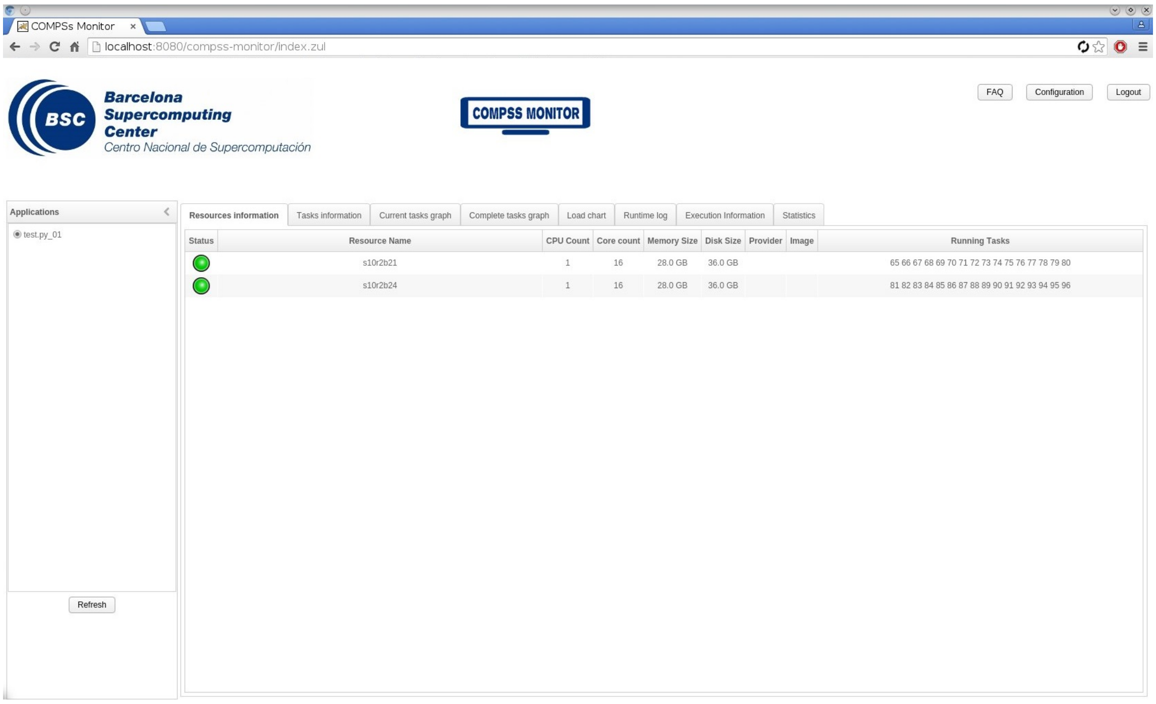


Figure 9: COMPSs monitoring interface

To enable **all** the COMPSs Monitor features, applications must run the runcompss command with the *-m* flag. This flag allows the COMPSs Runtime to store special information inside inside the *log base folder* under the *monitor* folder. Only advanced users should modify or delete any of these files. If the application that a user is trying to monitor has not been executed with this flag, some of the COMPSs Monitor features will be disabled.

4.2.3 Graphical Interface features

Next we provide a summary of the COMPSs Monitor supported features available through the graphical interface:

- **Resources information**
  Provides information about the resources used by the application

- **Tasks information**
  Provides information about the tasks definition used by the application

- **Current tasks graph**
  Shows the tasks dependency graph currently stored into the COMPSs Runtime

- **Complete tasks graph**
  Shows the complete tasks dependecy graph of the application

- **Load chart**
  Shows different dynamic charts representing the evolution over time of the resources load and the tasks load

- **Runtime log**
  Shows the runtime log
- **Execution Information**
  Shows specific job information allowing users to easily select failed or uncompleted jobs
- **Statistics**
  Shows application statistics such as the accumulated cloud cost.

Attention: To enable all the COMPSs Monitor features applications must run with the $-m$ flag.

The webpage also allows users to configure some performance parameters of the monitoring service by accessing the *Configuration* button at the top-right corner of the web page.

For specific COMPSs Monitor feature configuration please check our *FAQ* section at the top-right corner of the web page.

4.3 Application tracing

COMPSs Runtime can generate a post-execution trace of the distributed execution of the application. This trace is useful for performance analysis and diagnosis.

A trace file may contain different events to determine the COMPSs master state, the task execution state or the file-transfers. Despite the fact that in the current release we do not support file-transfers, we intend to support them in a near future release.

During the execution of the application, an XML file is created at worker nodes to keep track of these events. At the end of the execution, all the XML files are merged to get a final trace file.

In the following sections we explain the command used for tracing, how the events are registered, in a process called instrumentation, how to visualize the trace file and make a good analysis of performance based on the data shown in the trace.

4.3.1 Trace Command

In order to obtain a post-execution trace file the option **-t** must be added to the runcompss command. Next we provide an example of the command execution with the tracing option enabled for the Hmmer java application.

```
compss@bsc:~$ runcompss -t --classpath=/home/compss/workspace_java/hmmer/jar/hmmer.jar
                        hmmerobj.HMMPfam
                        /sharedDisk/Hmmer/smart.HMMs.bin /sharedDisk/Hmmer/256seq
                        /home/compss/out.txt 2 8 -A 222
```

4.3.2 Application Instrumentation

The instrumentation is the process that intercepts different events of the application execution and keeps log of them. This will cause an overhead in the execution time of the application that the user should take into account, but the collected data will be extremely useful for performance analysis and diagnosis.

COMPSs Runtime uses the *Extrae* tool to dynamically instrument the application and the *Paraver* tool to visualize the obtained tracefiles. Both tools are developped at *BSC* and are available in its webpage `http://bsc.es` .

At the worker nodes, in background, *Extrae* keeps track of the events in an intermediate format file (with *.mpit* extension). Inside the master node, at the end of the execution, *Extrae* merges the intermediate files to get the final trace file, a *Paraver* format file (.prv). See the visualization section 4.3.3 in this manual for further information about the *Paraver* tool.

When instrumenting the application *Extrae* will output several messages. At the master node, *Extrae* will show up its initialization at the begining of the execution and the merging process and the paraver generation at the end of the execution. At the worker nodes *Extrae* will inform about the intermediate files generation every time a task is executed. Next we provide a summary of the *stdout* generated by Hmmer java application execution with the trace flag enabled.

```
------------- Executing hmmerobj.HMMPfam ------------------

[   API]  -  Deploying the Integrated Toolkit
[   API]  -  Starting the Integrated Toolkit
[   API]  -  Initializing components

Welcome to Extrae 2.4.3rc4 (revision 311 based on framework/trunk/files/extrae)
Extrae: Generating intermediate files for Paraver traces.
Extrae: Intermediate files will be stored in /home/user/IT/hmmerobj.HMMPfam
Extrae: Tracing buffer can hold 500000 events
Extrae: Tracing mode is set to: Detail.
Extrae: Successfully initiated with 1 tasks

[   API]  -  Ready to process tasks

...
...
...
[   API]  -  No more tasks for app 1
[   API]  -  Stopping IT
[   API]  -  Cleaning

Extrae: Application has ended. Tracing has been terminated.
...

merger: Output trace format is: Paraver
merger: Extrae 2.4.3rc4 (revision 311 based on framework/trunk/files/extrae)
...

[   API]  -  Integrated Toolkit stopped
...

mpi2prv: Selected output trace format is Paraver

mpi2prv: Parsing intermediate files

mpi2prv: Generating tracefile (intermediate buffers of 1342156 events)

mpi2prv: Congratulations! hmmerobj.HMMPfam_compss_trace_1392736225.prv has been generated
```

```
    .
--------------------------------------------------------
```

For further information about *Extrae* please visit the following site:

`http://www.bsc.es/computer-science/extrae`

4.3.3 Trace Visualization

Paraver is the *BSC* tool for trace visualization. Trace events are encoded in *Paraver* format (*.prv*) by the *Extrae* tool (see previous section). *Paraver* is a powerful tool that allows users to show many views of the trace data by means of different configuration files. Users can manually load, edit or create configuration files to obtain different trace data views.

In the following subsections we will explain how to load a trace file into *Paraver*, open the task events view by means of an already predefined configuration file, and how to adjust the view to properly display the data.

For further information about *Paraver* please visit the following site:

`http://www.bsc.es/computer-sciences/performance-tools/paraver`

4.3.3.1 Trace Loading

The final trace file in *Paraver* format (*.prv*) can be found at the *base log folder* of the application execution inside the trace folder. The fastest way to open it is calling directly the *Paraver* binary using the trace-file name as arugment.

```
compss@bsc:~$ wxparaver /home/compss/.COMPSs/hmmerobj.HMMPfam_01/trace/*.prv
```

4.3.3.2 Configuration File

In order to open a view with the task events of the application, an already predefined configuration file is provided. To open it, just go in the main window to the "Load Configuration" option in the menu "File". The configuration file is under the following path */opt/COMPSs/Dependencies/paraver/cfgs/tasks.cfg*. After accepting the load of the configuration file, another window will appear to show the view. Figures 4.3.3.2 and 4.3.3.2 show an example of this process.

4.3.3.3 View Adjustment

In a *Paraver* view, a red exclamation sign may appear on the bottom-left corner (see last Figure 4.3.3.2 in previous section). This means that some little adjustments must be done to view the trace correctly:

- Fit window: this will give a better color scale to identify events.

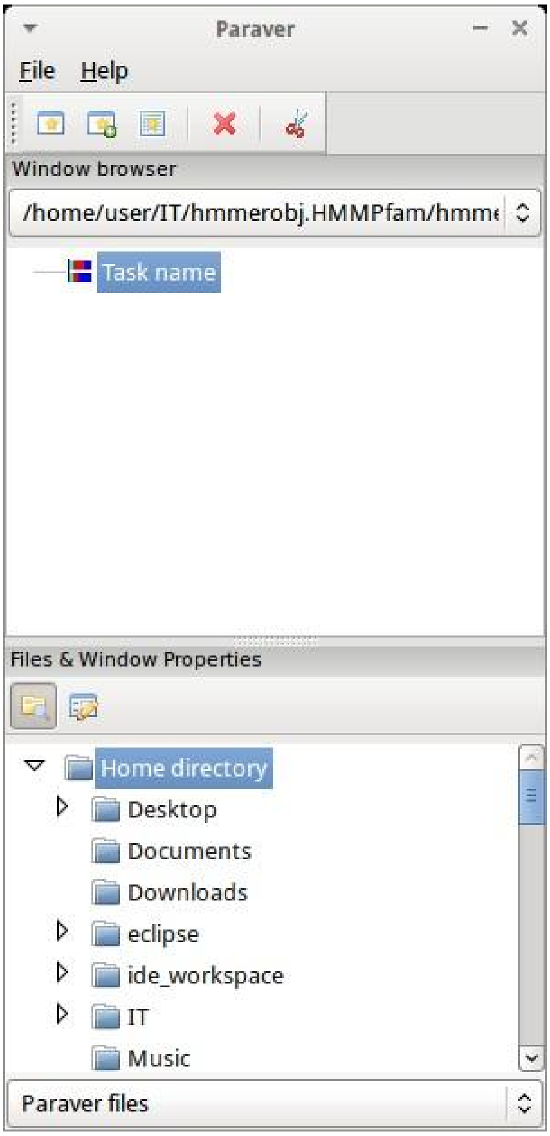


Figure 10: Paraver menu

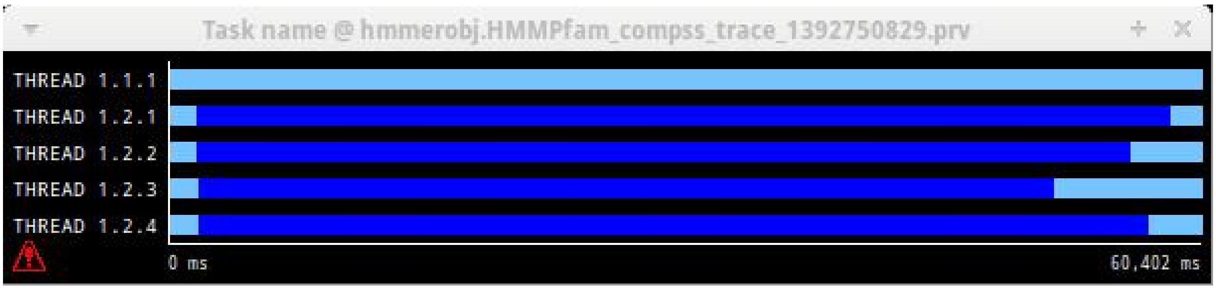


Figure 11: Trace file

- Right click on the trace window
- Chose the option Fit Semantic Scale / Fit Both

- View Event Flags: This will put a flag whenever an event starts/ends.

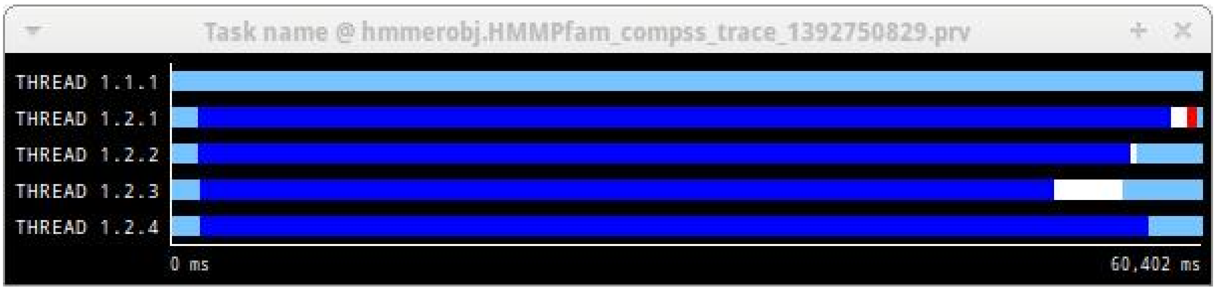


Figure 12: Paraver view adjustment: Fit window

  - Right click on the trace window
  - Chose the option View / Event Flags

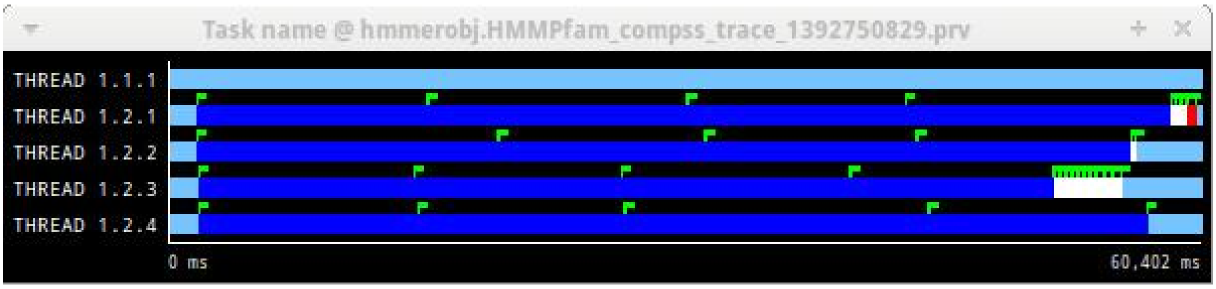


Figure 13: Paraver view adjustment: View Event Flags

- Show Info Panel: This will show an information panel. In the tab "Colors" we can see the legend of the colors shown in the view.
  - Right click on the trace window
  - Check the Info Panel option
  - Select the Colors tab in the panel

- Zoom: In order to understand a trace view better, sometimes it's a worth thing to zoom into it a little.
  - Select a region in the trace window to see that region in detail
  - And repeat the previous step as many times as needed
  - The undo-zoom option is in the right click panel

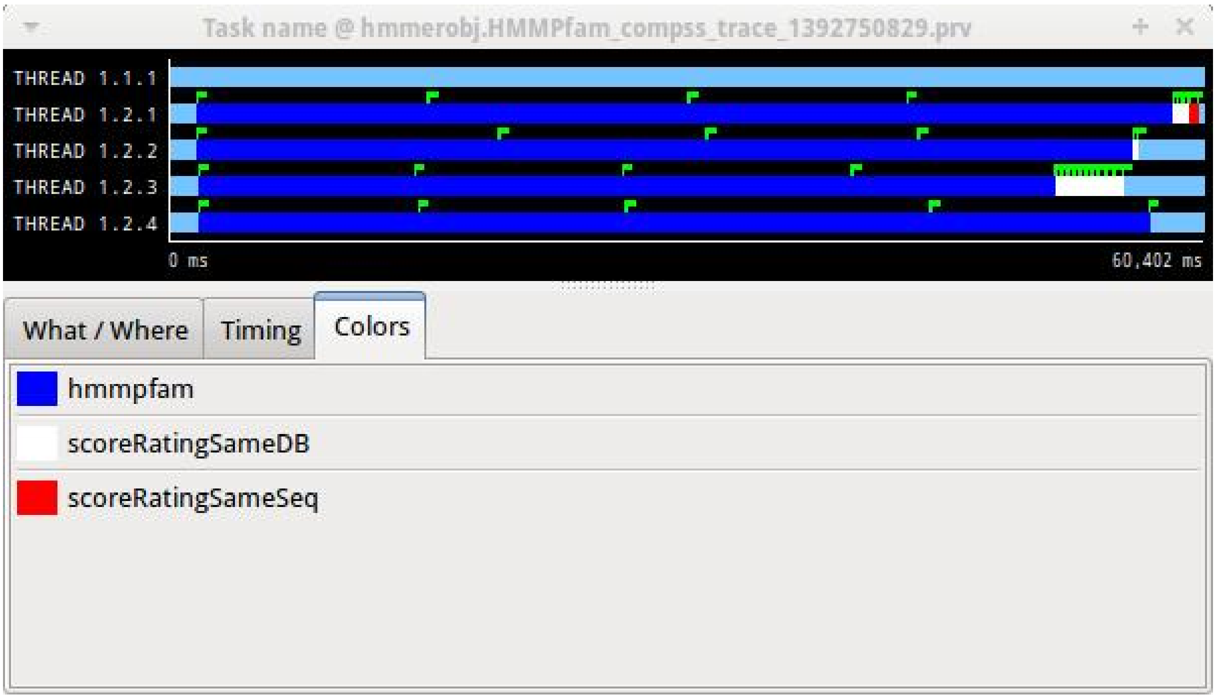


Figure 14: Paraver view adjustment: Show info panel

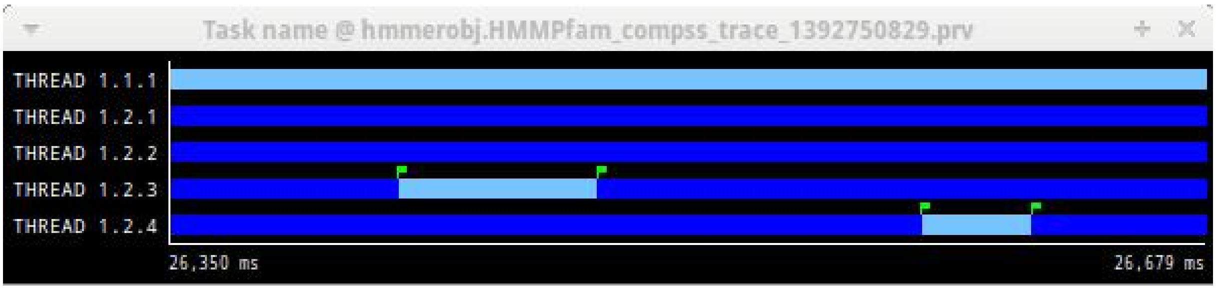


Figure 15: Paraver view adjustment: Zoom configuration

Figure 16: Paraver view adjustment: Zoom configuration

4.3.4 Trace Interpretation

In this section we will explain how to interpret a trace view once it has been adjusted as described in the previous section.

- The trace view has in its horizontal axis the execution time and in the vertical axis one line for the master at the top, and below it, one line for each of the workers.

- In a line, the light blue color means idle state, in the sense that there is no event at that time.
- Whenever an event starts or ends a flag is shown.
- In the middle of an event, the line shows a different color. Colors are assigned depending on the event type.
- In the info panel the legend of assigned color to event type is provided.

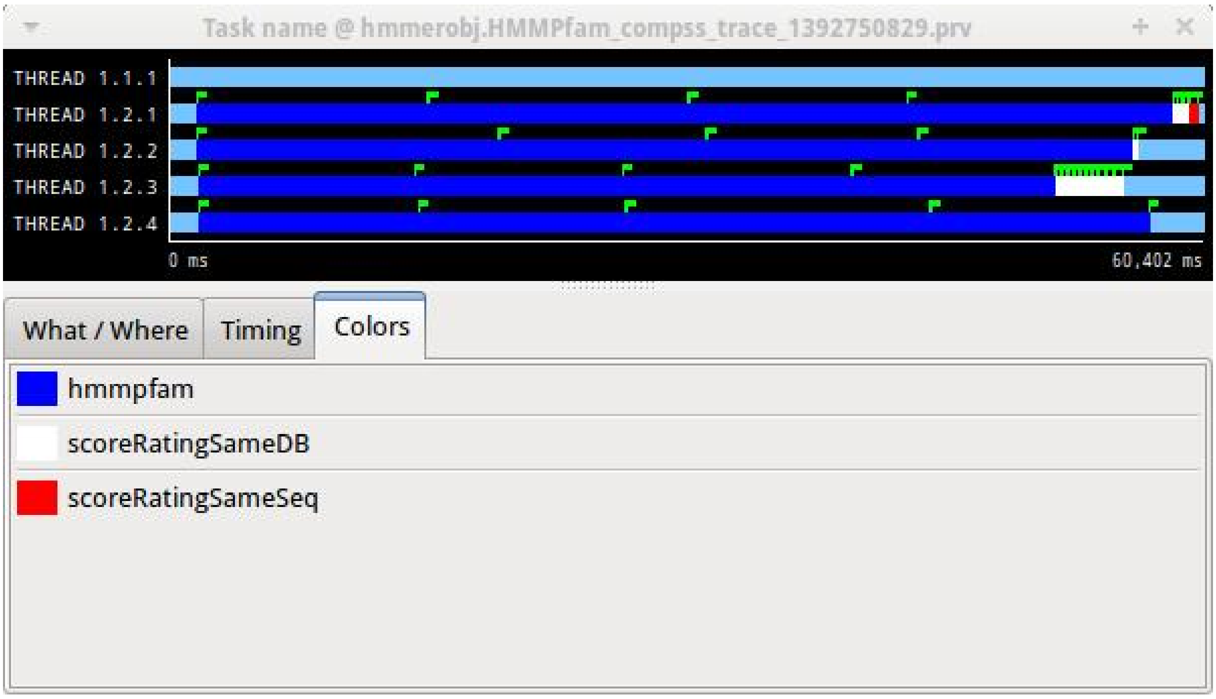


Figure 17: Trace interpretation

4.3.5 Trace Analysis

In this section, we will give some tips to analyse a COMPSs trace from two different points of view: graphically and numerically.

4.3.5.1 Graphical Analysis

The main concept is that computational events, the task events in this case, must be well distributed among all workers to have a good parallelism, and the duration of task events should be also balanced, this means, the duration of computational bursts.

In the previous trace view, all the tasks of type "hmmpfam" in dark blue appear to be well distributed among the four workers, each worker executes four "hmmpfam" tasks.

But some workers finish earlier than the others, worker 1.2.3 finish the first and worker 1.2.1 the last. So there is an imbalance in the duration of "hmmpfam" tasks. The

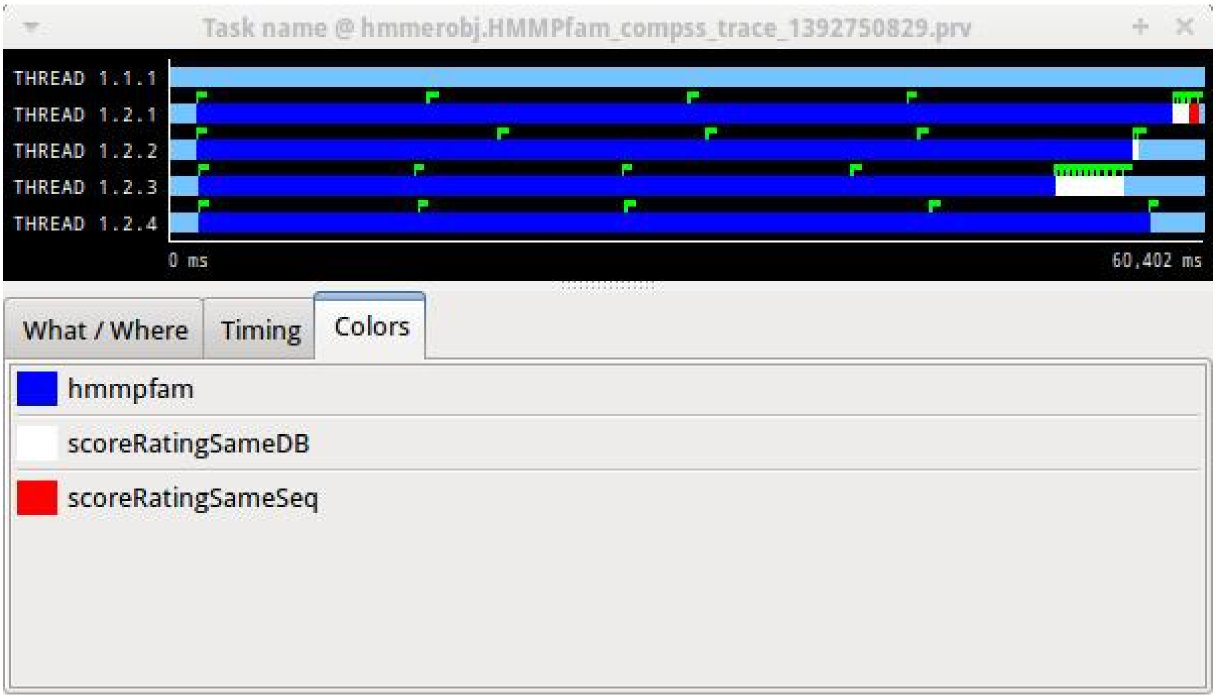


Figure 18: Caption.

programmer should analyse then whether all the tasks process the same amount of input data and do the same thing in order to find out the reason of such imbalance.

Another thing to highlight is that tasks of type "scoreRatingSameDB" are not equal distributed among all the workers. There are workers that execute more tasks of this type than the others. To understand better what happens here, let's take a look to the execution graph and also zoom in the last part of the trace.

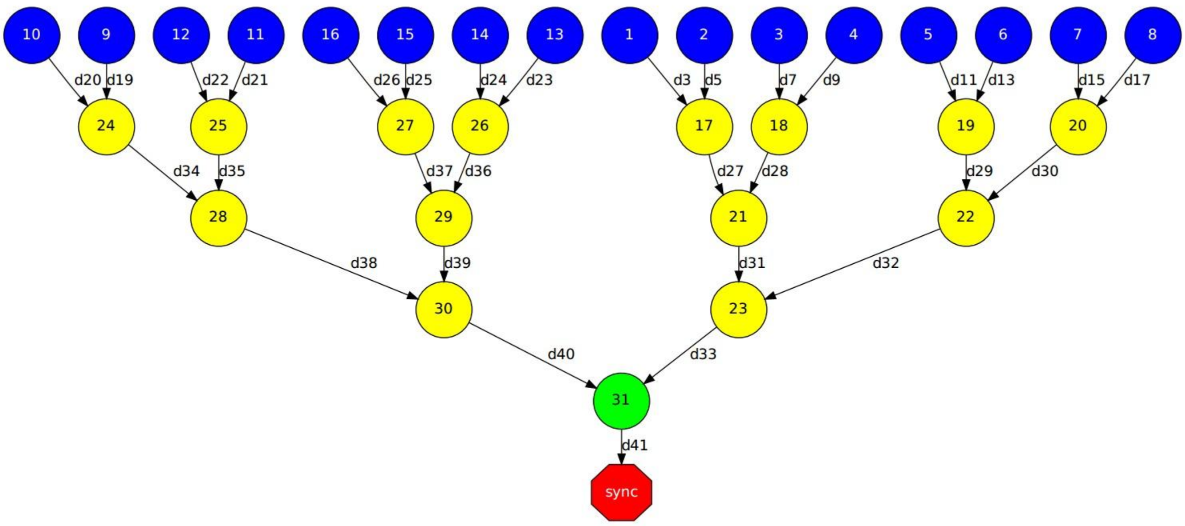


Figure 19: Caption.

There is only one task of type "scoreRatingSameSeq". This task appears in red in the trace (and in light-green in the graph). With the help of the graph we see that the

Figure 20: Caption.

"scoreRatingSameSeq" task has dependences on tasks of type "scoreRatingSameDB", in white (or yellow).

When the last task of type "hmmpfam" (in dark blue) ends, the last dependences are solved, and if we look at the graph, this means going across a path of three dependences of type "scoreRatingSameDB" (in yellow). And because of these are sequential dependences (one depends on the previous) no more than a worker can be used at the same time to execute the tasks. This is the reason of why the last three task of type "scoreRatingSameDB" (in white) are executed in worker 1.2.1 sequentially.

4.3.5.2 Numerical Analysis

Here we show another trace from a different parallel execution of the Hmmer program.

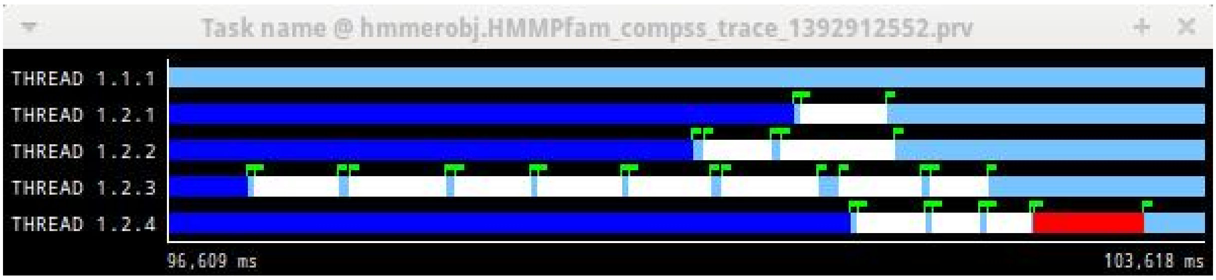


Figure 21: Caption.

Paraver offers the possibility of having different histograms of the trace events. For it just click the "New Histogram" button in the main window and accept the default options in the "New Histogram" window that will appear.

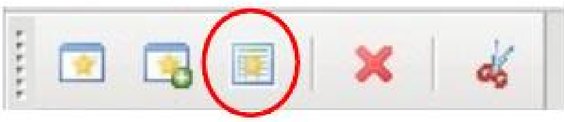
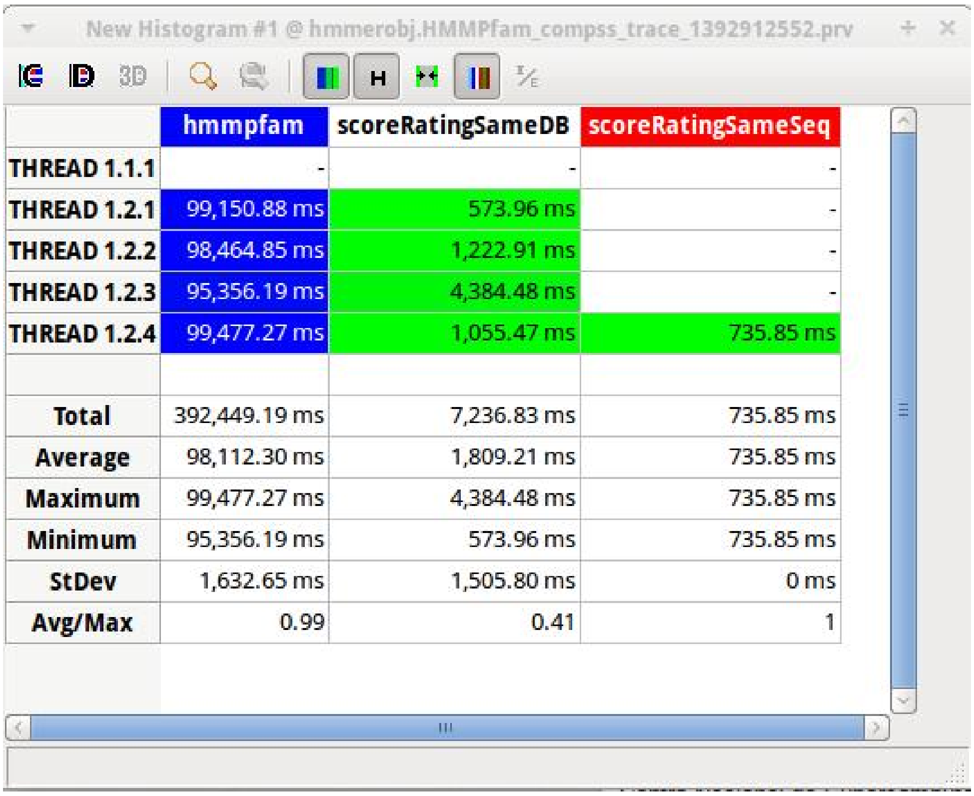


Figure 22: Paraver Menu - New Histogram

After that, the following table is shown. In this case for each worker, the time spent executing each type of task is shown. Task names appear in the same color than in the trace view. The color of a cell in a row corresponding to a worker goes in a scale from a light-green for lower values to a dark-blue for higher ones. This conforms a color based histogram.



	hmmpfam	scoreRatingSameDB	scoreRatingSameSeq
THREAD 1.1.1	-	-	-
THREAD 1.2.1	99,150.88 ms	573.96 ms	-
THREAD 1.2.2	98,464.85 ms	1,222.91 ms	-
THREAD 1.2.3	95,356.19 ms	4,384.48 ms	-
THREAD 1.2.4	99,477.27 ms	1,055.47 ms	735.85 ms
Total	392,449.19 ms	7,236.83 ms	735.85 ms
Average	98,112.30 ms	1,809.21 ms	735.85 ms
Maximum	99,477.27 ms	4,384.48 ms	735.85 ms
Minimum	95,356.19 ms	573.96 ms	735.85 ms
StDev	1,632.65 ms	1,505.80 ms	0 ms
Avg/Max	0.99	0.41	1

Figure 23: Hmmpfam histogram

The previous table also gives, at the end of each column, some extra statistical information for each type of tasks (as the total, average, maximum or minimum values, etc.).

In the window properties of the main window we can change the semantic of the statistics to see other factors rather than the time, for example, the number of bursts.

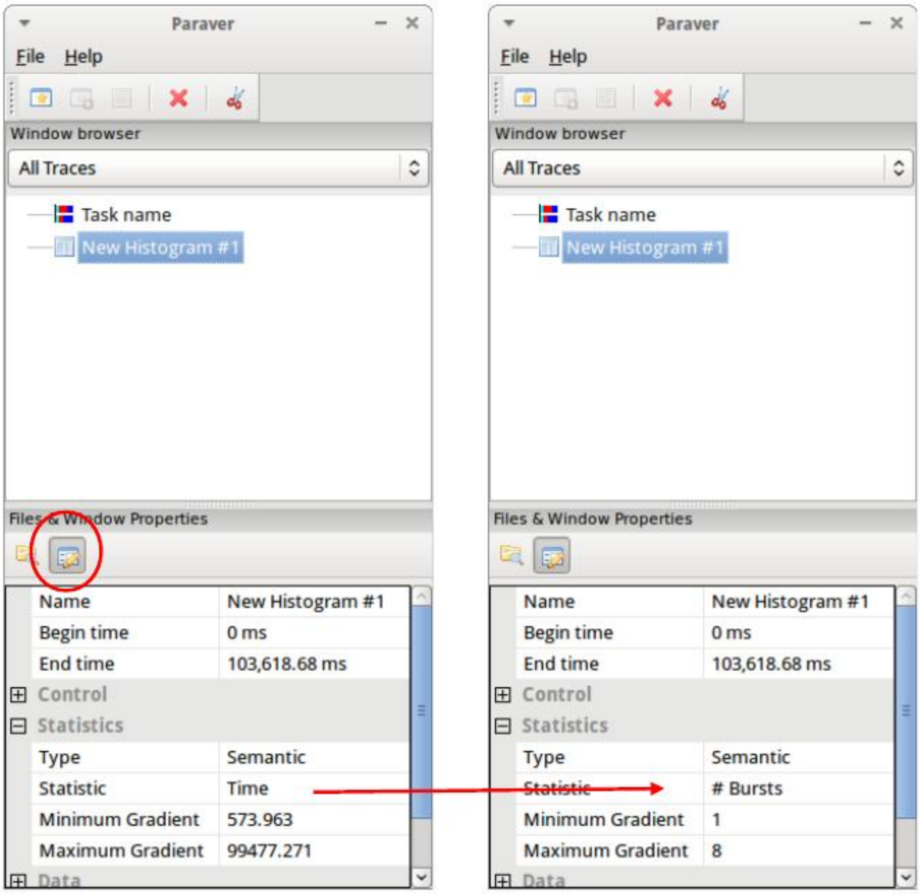
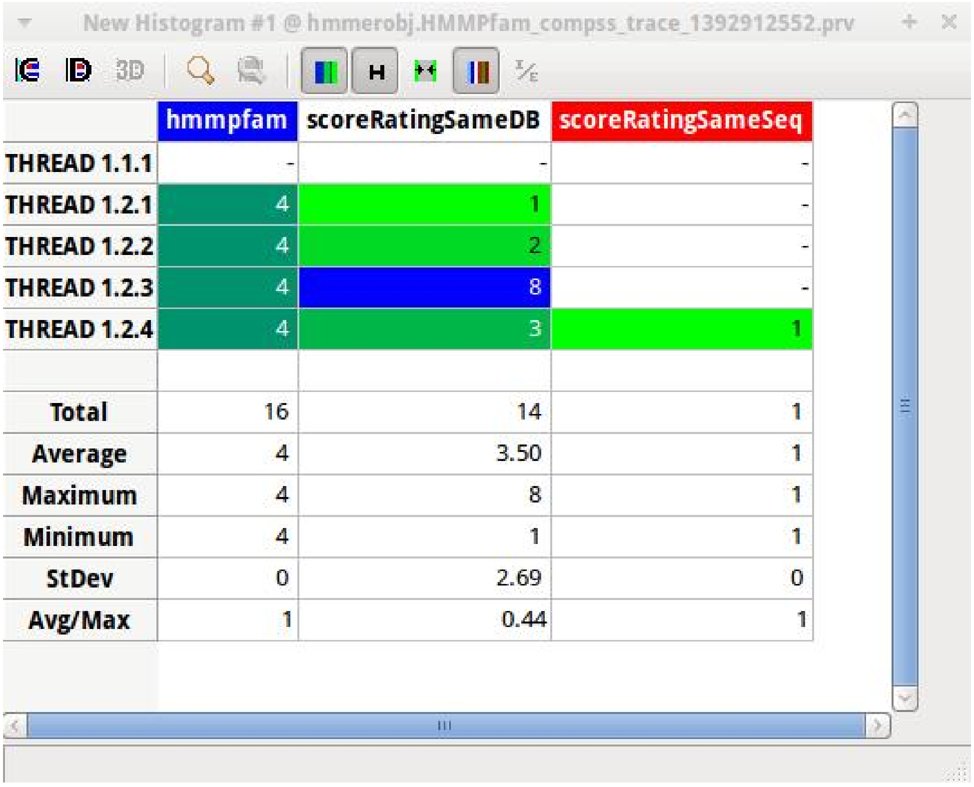


Figure 24: Paraver histogram options menu

In the same way as before, the following table shows for each worker the number of bursts for each type of task, this is, the number or tasks executed of each type. Notice the gradient scale from light-green to dark-blue changes with the new values.



	hmmpfam	scoreRatingSameDB	scoreRatingSameSeq
THREAD 1.1.1	-	-	-
THREAD 1.2.1	4	1	-
THREAD 1.2.2	4	2	-
THREAD 1.2.3	4	8	-
THREAD 1.2.4	4	3	1
Total	16	14	1
Average	4	3.50	1
Maximum	4	8	1
Minimum	4	1	1
StDev	0	2.69	0
Avg/Max	1	0.44	1

Figure 25: Hmmpfam histogram with the number of bursts

4.3.6 Other Trace examples

To end this section, let's present some other examples of COMPSs traces. COMPSs traces can be much complex as the number of workers or tasks grows. Just to illustrate this, the following pictures show traces with a greater number of workers and tasks.

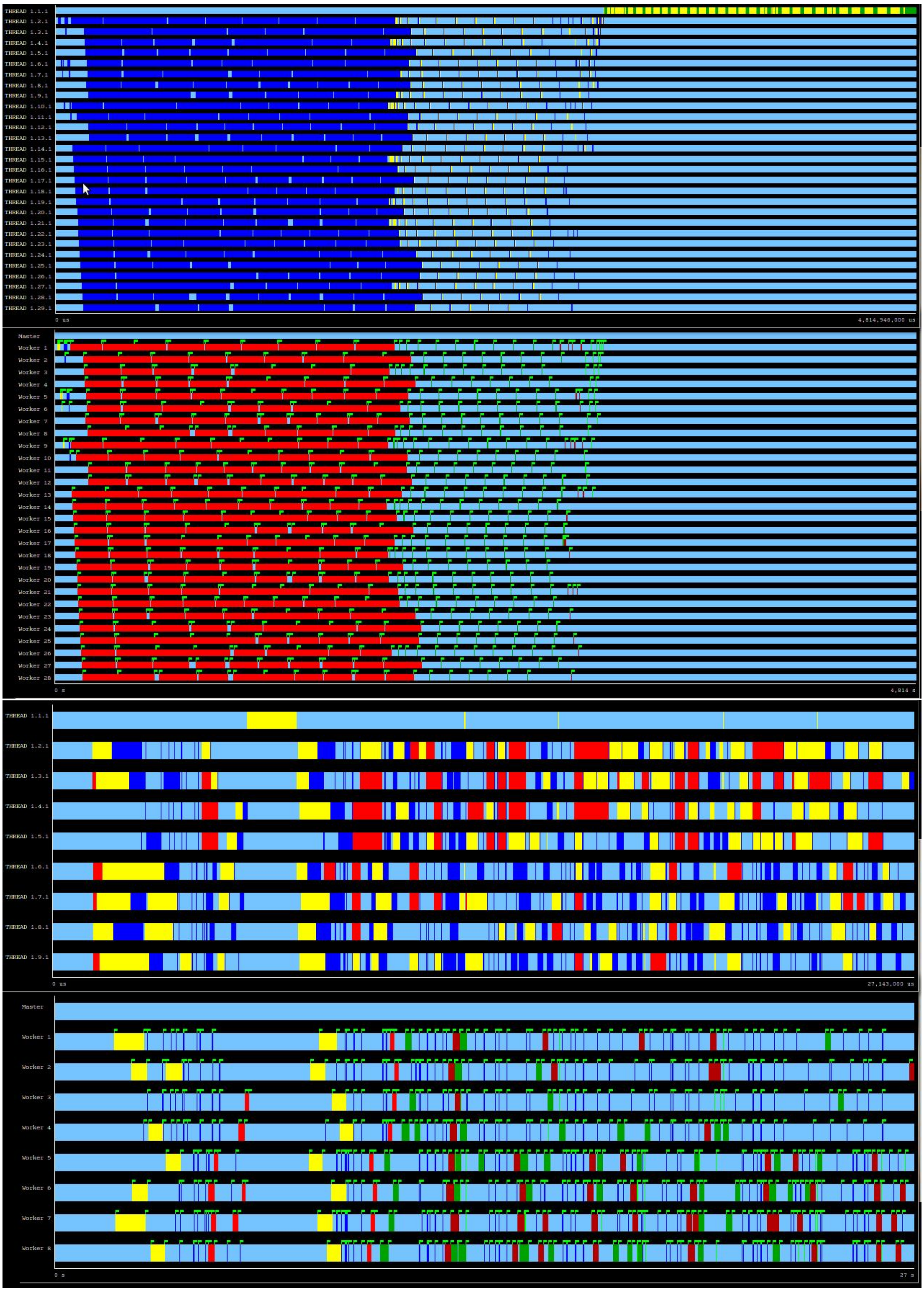


Figure 26: Examples of complex traces

4.4 IDE

The Eclipse IDE is available through the *Eclipse Market*.

5 Common Issues

This section provides answers for the most common issues that users encounter while beginning to execute COMPSs applications. Nevertheless, if your specific issues are not covered through this section, please do not hesitate to contact us at:

`support-compss@bsc.es`

5.1 How to debug

When the application does not behave as expected the first thing users must do is to run it in **debug** mode. We remember that the runcompss command allows to introduce a *-d* flag to enable the debug log level.

Once done, application execution will produce the following files:

- runtime.log
- resources.log
- jobs folder

Firstly, users should check the last lines of the runtime.log. If the file-transfers or the tasks are failing an error message will appear in this file. Secondly, if the file-transfers are successfully and the jobs are submited, users should check the *jobs* folder and look at the error messages produced inside each job. Users should notice that if there are *_RESUBMITTED* files something inside the job is failing.

5.2 Tasks are not executed

If the tasks remain in **Blocked** state means that there are no existing resouces matching the specific task constraints. This error can be potentially caused by two facts: in one hand, because the resources are not correctly loaded into the runtime and, in the other hand, because the task constraints do not really match with any resource.

In the first case, users should take a look at the *resouces.log* and check that all the resources defined in the *project.xml* file are available for the runtime. In the second case users should re-define the task constraints taking into account the resources capabilities defined into the *resources.xml* and *project.xml* files.

5.3 Jobs sistematically fail

If all the application tasks fail because all the submited jobs fail, it is probably due to the fact that there is a resource missconfiguration. In most of the cases, the resource that the application is trying to access has no passwordless access through the configured user. Users can try if this is their case by executing the following steps:

- Open the project.xml that the application is using. Remember that the default file is stored under */opt/COMPSs/ Runtime/configuration/xml/projects/project.xml*

- For each resource annotate its name and the value inside the *User* tag. Remember that if there is no *User* tag COMPSs will try to connect this resource with the same username than the one that launches the main application.

- For each annotated resourceName - user please try *ssh user@resourceName*. If the connection asks for a password then there has been a passwordless missconfiguration.

If there has been a passwordless missconfiguration you can solve it by running the following commands:

```
compss@bsc:~$ scp ~/.ssh/id_dsa.pub user@resourceName:./mydsa.pub
compss@bsc:~$ ssh user@resourceName "cat mydsa.pub >> ~/.ssh/authorized_keys; rm ./mydsa.
    pub"
```

These commands are a quick solution, for further details please check the *Additional Configuration* section inside the *COMPSs Installation Manual* available at our website `http://compss.bsc.es`.

Please find more details on the COMPSs framework at

`http://compss.bsc.es`