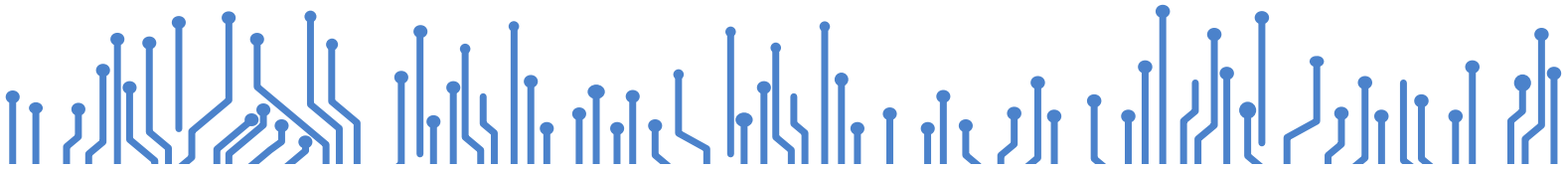
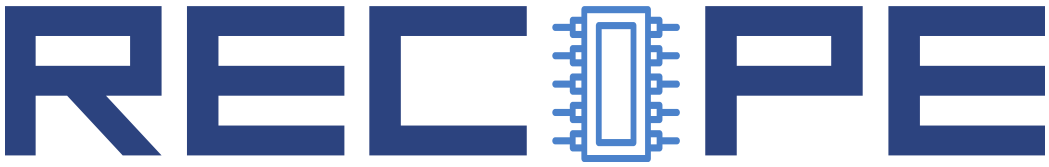
REliable Power and time-ConstraInts-aware Predictive management of heterogeneous Exascale systems





WP2 Runtime Resource Management Infrastructure

2.2 RECIPE Local Resource Manager Prototype

http://www.recipe-project.eu

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 801137

Grant Agreement No.: 801137
Deliverable: 2.2 RECIPE Local Resource Manager Prototype

Project Start Date: 01/05/2018 **Duration**: 36 months
Coordinator: *Politecnico di Milano, Italy*

Deliverable No:	2.2
WP No:	2
WP Leader:	Giuseppe Massari
Due date:	31/10/2019
Delivery date:	31/10/2019

Dissemination Level:

PU	Public Use	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

DOCUMENT SUMMARY INFORMATION

Project title:	**REliable Power and time-ConstraInts-aware Predictive management of heterogeneous Exascale systems**
Short project name:	RECIPE
Project No:	801137
Call Identifier:	H2020-FETHPC-2017
Thematic Priority:	Future and Emerging Technologies
Type of Action:	Research and Innovation Action
Start date of the project:	01/05/2018
Duration of the project:	36 months
Project website:	http://www.recipe-project.eu

2.2 RECIPE Local Resource Manager Prototype

Work Package:	WP2 Runtime Resource Management Infrastructure
Deliverable number:	2.2
Deliverable title:	RECIPE Local Resource Manager Prototype
Due date:	31/10/2019
Actual submission date:	31/10/2019
Editor:	Giuseppe Massari
Authors:	G. Massari, F. Reghenzani, R. Tornero, M. Zanella, G. Agosta, W. Fornaciari, J. Flich
Dissemination Level:	PU
No. pages:	36
Authorized (date):	30/10/2019
Responsible person:	W. Fornaciari
Status:	Plan Draft Working **Final** Submitted Approved

Revision history:

Version	Date	Author	Comment
v.0.1	04/10/2019	POLIMI	Initial skeleton
v.0.2	11/10/2019	POLIMI	Most of the sections on reliability
v.0.3	14/10/2019	POLIMI	Integration of the timing contributions
v.0.4	22/10/2019	POLIMI	Overall improvements and content extensions
v.0.5	24/10/2019	POLIMI	Draft version
v.0.6	28/10/2019	POLIMI	Ready-to-review version
v.0.7	29/10/2019	POLIMI	Integration of UPV revision
v.1.0	30/10/2019	POLIMI	Final release for submission

Quality Control:

	Who	Date
Checked by internal reviewer	UPV	29/10/2019
Checked by WP Leader	Giuseppe Massari	29/10/2019
Checked by Project Technical Manager	G. Agosta	30/10/2019
Checked by Project Coordinator	W. Fornaciari	30/10/2019

COPYRIGHT

ACKNOWLEDGEMENTS

RECIPE is a project that has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No 801137. Please see http://www.recipe-project.eu for more information.

The partners in the project are Universitat Politècnica de València (UPV), Centro Regionale Information Communication Technology scrl (CeRICT), École Polytechnique Fédérale de Lausanne (EPFL), Barcelona Supercomputing Center (BSC), Poznan Supercomputing and Networking Center (PSNC), IBT Solutions S.r.l. (IBTS), Centre Hospitalier Universitaire Vaudois (CHUV). The content of this document is the result of extensive discussions within the RECIPE ©Consortium as a whole.

DISCLAIMER

Contents

1 Introduction

As the interest for High Performance Computing (HPC) is growing, the deployment of such computing infrastructures is facing issues and challenges growing as well. Given the focus of the H2020 RECIPE project [10] and, more specifically, the goal of the current deliverable, we limit our discussion to three key aspects, for which an accurate design and implementation of the software stack is crucial.

First, new large scale application domains are emerging, for which such a class of computing sytems is required. For example, machine and deep learning, as well as stochastic models based applications, need a huge amount of memory and computational nodes to run Big Data sized algorithms. Moreover, HPC is gaining interest also for automotive, forecasting and medical applications, whose specifications typically come with critical real-time constraints.

Second, such application domains are increasing the demand of processing capabilities, raising up the power consumption of super-computing centres, and thus their contribution to the overall world-wide CO2 emissions [11]. Part of the solution to this problem comes from the adoption of more energy-efficient hardware units, like GPUs and many-core accelerators, for which specific programming models have been developed. However, this also makes it more challenging to find the optimal solutions in terms of task scheduling and resource allocation, from both the performance and the power perspective [9].

Third, in the last decade, the Mean-Time-Between-Failure (MTBF) in a typical HPC infrastructure has significantly dropped, with systems experiencing the occurrence of more than one hardware failure per day [19]. Depending on the type of failure or fault, the workload in execution may be affected in terms of performance degradation or errors in the output results carried out. These events could be not acceptable for certain classes of applications, like the aforementioned time-constrained ones.

As we already explained in D2.1, to address all the aforementioned challenges, a careful design of the system from the hardware perspective is not enough. The necessity of effective management software layers is growing with the complexity of the systems themselves. Furthermore, all the components of the software stack must cooperate, in order to build an energy-efficient [2] and fault-tolerant HPC infrastructure [15]: from the hardware abstraction layer to the upper-layer frameworks (resource managers, hypervisors, container orchestrators,...), including operating systems and programming models.

This complexity motivates the need of building a hierarchical resource management infrastructure, where each part can focus on a specific subdomain of the overall management problem. In this deliverable, we focus on the prototype of the local manager, which is the stack component in charge of solving the problem of allocating resources to the workload running on each single node of the HPC infrastructure. This means that an instance of the local resource manger will be deployed on each of the computational nodes.

The cooperation among hierarchical levels is then made possible by the implementation of suitable interfaces, between local resource manager and application programming model, other than local and global resource manager. In the former case, we could profile the applications execution at runtime, infer how they are using the computing resources and optionally redefine the

assignment. In the latter, the local resource manager would act as data provider for the global resource manager. This goes in the direction of enabling the implementing proactive policies at all the levels.

In the current deliverable, we summarize the work carried out in Task T2.2. More in detail, we provide a description of the local resource manager prototype (the BarbequeRTRM), focusing on the design and implementation effort in the time frame between months 9 and 18. The goal of this work was the introduction the hardware reliability and time-constraints management capabilities, at local resource manager level. Morever, we will briefly mention how we expect to interact with the global resource manager, with the modelling frameworks of Work Package 3 and the low-level functionalities provided by the hardware abstraction layer, as Work Package 4 outcome.

1.1 Overview and document structure

This deliverable is focused on the implementation of the local resource manager, which is the software stack component in charge of managing the resource assignment at level of single HPC node.

In deliverable D2.1, we introduced the overall status of the entire. In this document instead, we will put in evidence the recent developments aiming at making the resource manager reliability-aware and capable of implementing allocation policies that would take into account timing constraints.

In Section 2, we recall the local resource manager structure, the interfaces for providing data to the global resource manager and some new extensions specifically thought for the HPC domain.

In Section 3, we carefully describe the design of the new components the local resource manager will rely on, in order to implement reliability-aware resource management strategies. Further discussion regarding specific policy implementation are postponed to the next deliverable (D2.5), once suitable experiments have been performed.

In Section 4, we provide a picture of the stategy with aim at putting in place for addressing the time constraints management on HPC infrastructures, starting from the theoretical fundations of our approach and the moving to the role of the local resource manager.

Finally, in Section 5 we recap the status of the prototype and briefly discuss about the next steps.

2 The BarbequeRTRM

The *Local Resource Manager* represents the component of the RECIPE software stack in charge of controlling the assignment of resources, in the scope of a single node of a distributed HPC system.

In RECIPE, each computing node features a heterogeneous set of resources, for which ad-hoc resource management strategies are needed. We refer to this as *Heterogeneous Node (HN)*. The resource manager must therefore consider the heterogeneous capabilities of the processing units, in terms of timing predictability and performance, as well as the power-thermal profile. Moreover, the complexity of the target HPC node is also given by the interconnect infrastructure and the non-uniform memory accesses. In this regard, the *Hardware Abstraction Layer* can play a key role, by providing interfaces to gather HW-level profiling data about interconnect bandwidth occupancy and memory accesses. These would be exploited by the resource manager runtime policies in order to optimize performance, timing predictability and energy consumption in a more effective manner.

In this project, the basis of the development of the local resource manager is the *Barbeque Runtime Resource Manager (BarbequeRTRM)* [1]. This resource manager was born and developed thanks to previous EU-funded projects (FP7 2PARMA, HARPA and H2020 MANGO) [4].

The RECIPE Project requirements and objectives are driving the development of the BarbequeRTRM towards the introduction of new components devoted to:

- Address the increasing hardware-level reliability issues by implementing reactive and proactive countermeasures.
- Provide time-predictability guarantees to manage the execution of time-constrained applications, coming from the latest HPC application domains.

In this section, we recall the BarbequeRTRM features already introduced in deliverable D2.1, and describe this part of the software stack is being developed to target the aforementioned objectives.

2.1 Architecture

In this section, we describe the main components of the resource management framework deployed at single node level: the *BarbequeRTRM*. Figure 1 shows the overall architecture of the framework and the interfaces provided to manage applications and integrate hardware platforms supports.

The topmost layer in the figure shows the*Application Runtime Library (RTLib)* and the supported languages (C, C++, OpenCL and Python). As we explained in deliverable D2.1, the library implements what we called *Adaptive Execution Model*. On top of this, we built the MANGO programming model, to enable a straighforwad programming approach to heterogeneous target plaforms.

[1] https://bosp.dei.polimi.it/doku.php

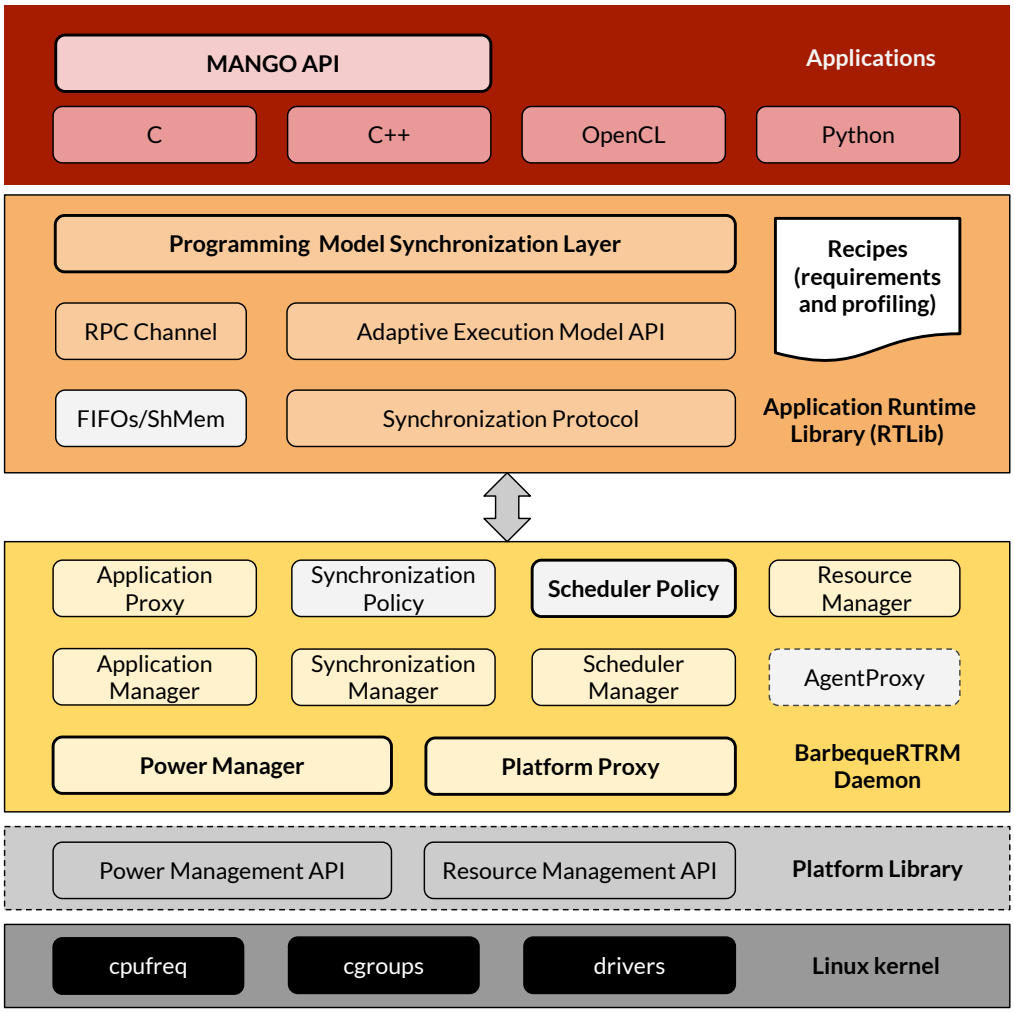


Figure 1: Overview of the BarbequeRTRM with the integration of MANGO specific components.

Other than defining a programming model, the *RTLib* covers several aspects of the interaction between application and run-time resource manager:

- To hide the low-level details of the communication channel and the custom protocol
- To provide an application programming interface for run-time manageable applications (*Adaptive Execution Model*) API
- To provide an interface for specifying application performance requirements and other profiling data (*Recipe*).

For the sake of portability, the BarbequeRTRM runs as a user-space daemon, occupying only a few resources of the host-side resources (CPUs).

The daemon and the applications communicate through a Remote Procedure Call (RPC) based protocol. Data are mainly exchanged by using named pipes; a general one for the messages coming from the application to the resource manager, plus one pipe per application for application management purposes. A further communication channel, based on shared memory, has been introduced to enable an efficient transfer of complex data structures, like for example the task-

graph representations of the applications.

According to the figure, the run-time resource manager itself is the third layer of the framework. We reported only the most important modules of its internal structure, to avoid a useless excess of details. The `Resource Manager` module is the component in charge of launching all the resource manager services. The `Application Proxy` is the endpoint of the communication with the applications. The `Application Manager` is the collector of the information regarding all the managed applications (priority, requirements, status). For example, as soon as an application is launched, a message is implicitly sent from the Application Runtime Library (RTLib) to the resource manager. The Application Proxy notifies the Application Manager, which creates an application descriptor and puts it into a waiting list. The `Resource Manager` is notified also; it recognizes that an event for which a new run of the resource allocation policy may be triggered.

The `Scheduler Manager` comes into play in cases like this. This module is responsible for loading a `Scheduling Policy` plugin and running the algorithm. As a general behavior, the policy is executed on the basis of the occurrence of specific events or the elapsing of a given period. Once it terminates, a new resource allocation schema is defined. At this point, the `Synchronization Manager` is responsible for performing a sequence of steps to synchronize the application execution with the resource manager control. Since this is actually the core of the resource manager, most of the developments required to introduce reliability and timing supports will be focused on this part.

Regarding the hardware support instead, we need suitable system interfaces for accessing low-level mechanisms to enforce the resource assignments. The `Platform Proxy` and the `Power Manager` are the two components creating the abstraction layer on top of such platform and resource-specific interfaces. For instance, the BarbequeRTRM relies on the Linux frameworks `cgroup` and `cpufreq`, to (1) bound the amount of CPU time, memory and of CPU cores set assigned to each application, and (2) set the current clock frequency of the managed cores.

For resources that are out of the direct control of the Linux operating system, like the processing units in the HN, we need to rely on a specific `Platform Library`. Similarly to the Linux case, the library must usually include a first API to control the assignment of resources to specific tasks or processes, and a second one for power management purposes. The latter is typically made of functions for runtime *monitoring* (current sensors values, hardware counters, etc. . .) and functions for performing *control* actions (e.g., resource reservation). Examples of such libraries are already available in commercial systems. AMD and NVIDIA in fact, provide their own libraries to control and get runtime data from the GPUs, respectively called *AMD Display Library (ADL)* [3] and *NVIDIA Management Library (NVML)* [16].

Similarly, in RECIPE we follow the approach developed during the MANGO project, with the *HN Library* described in Section 2.5 to provide the abstraction layer on top of the custom hardware resources deployed on FPGA. What we expect at this level is the development of additional functions (Work Package 4) that we could exploit to perform fine-grained control for both reliabilty and time-constraints management purposes.

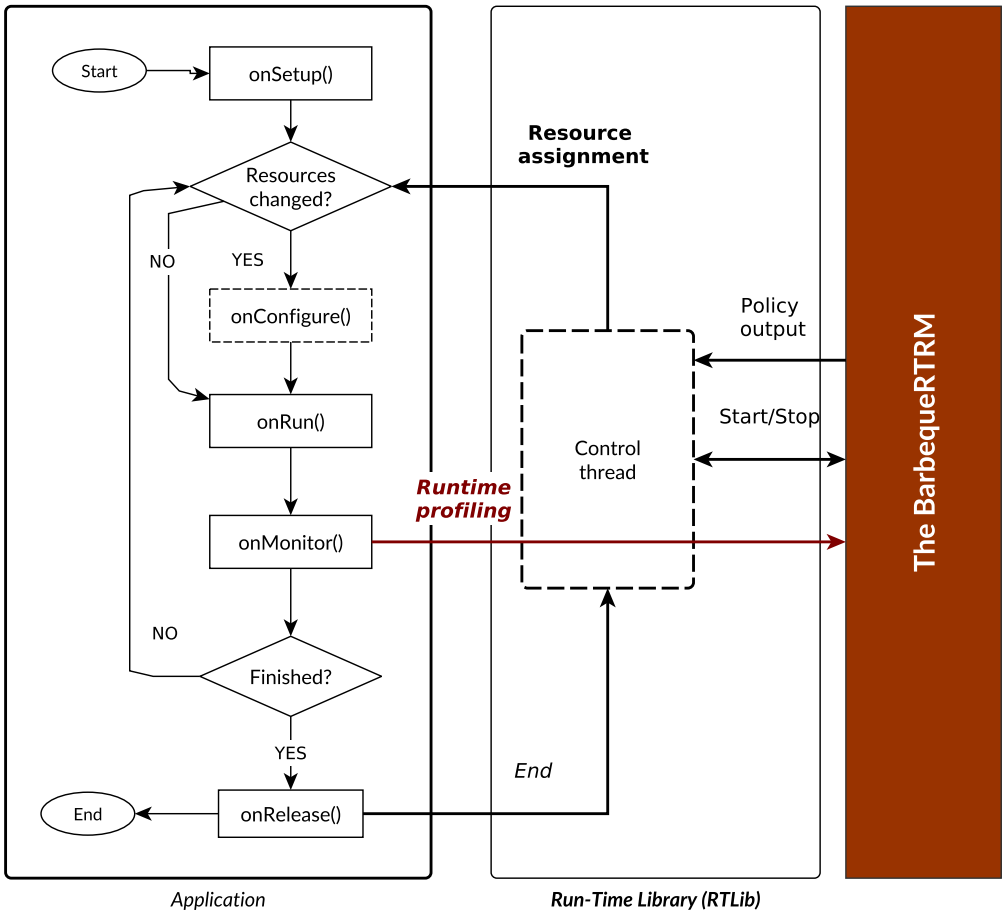


Figure 2: Adaptive Execution Model: the adaptive applications implements a BbqueEXC derived class. A control thread is responsible of the member function calls. The application is aware of the assigned resources and can negotiate.

2.2 Runtime managed programming models

In this section, we briefly mention the programming models already discussed in D2.1 and D2.4, as they are, in a certain measure, part of the the local resource manager.

We introduced the *Adaptive Execution Model (AEM)*, as a programming model thought for the implementation of reconfigurable applications (also know as "malleable" applications). The reference target of this model is the class of applications featuring a regular execution flow, which iterates over a big data set of a stream of input data (e.g., streaming processing). The application can therefore tune its parameters and level of parallelism (e.g. number of threads) according to the computing resources actually assigned. From the programming design perspective, the approach to AEM consists of defining a class derived from a base class coming with the *RTLib* and implementing the already provided member functions, whose invocation is driven by the interaction between the a control thread and the resource manager. From the programming language point of view instead, the developer can choose between four options: C, C++, Java and Python. In case of heterogeneous systems, OpenCL can by also exploited, by properly structuring the sequence of OpenCL functions calls. The BarbequeRTRM in fact, provides

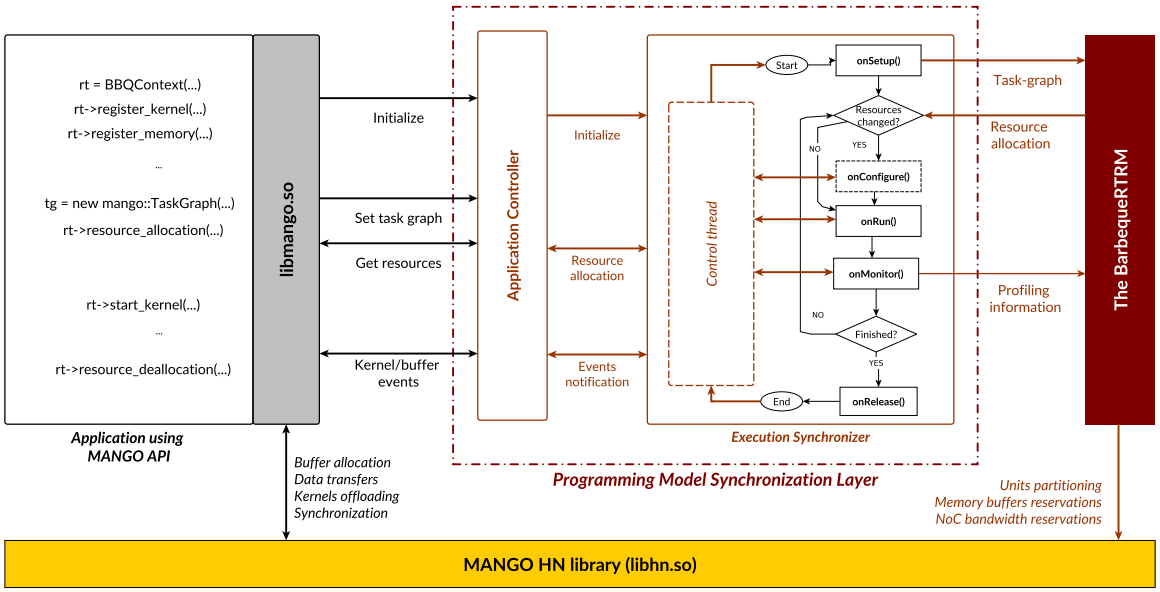


Figure 3: MANGO Programming Model: the execution is built on top of the Adaptive Execution Model.

some management capabilities of OpenCL applications, by intercepting the device requests and returning the options selected by its resource allocation policy.

During the MANGO project then, we built the *MANGO Programming Model* on top of the Adaptive Execution Model. In this case, the applications are structured into multiple tasks exchanging data through memory buffers. The local resource manager is responsible of mapping the resource assignments, at runtime, without involving the developer. This programming model is the reference choice for the application use cases targeting FPGA-based resources. From the programming language perspective, the developer can currently choose bewteen a C or C++ API, while a Python wrapper development is in progress and will be made available in a very few months from now.

In RECIPE, we do not expect to introduce disruptive changes in these programming models. To date, we planned to introduce a couple of functions that the developer can use to tune the reliabilty support, later discussed in Section 3. Finally, during the third phase of the project (Integration) we will evaluate the necessity of extending the already available interfaces to specify application requirements.

2.3 Process management

In order to properly manage HPC workload, especially legacy applications for which it would be very hard to require some porting effort, we introduced the possibility to manage generic without exploiting any of the run-time managed programming models recalled in the previous section (*AEM* and *MANGO*).

This further option has been made possible thanks to a Linux kernel feature called *Kernel Process Connector*. Thanks to this, we can observe events related to the creation or destruction

of processes in the system, which means applications starting or terminating. We integrating this kernel featuring into the BarbequeRTRM by implementing a component called `Process Listener`. Afterwards, we introduced a user-interface through which it is possible to specify the executable names of the processes we are actually interested in.

Once the name of a process matches the one for which the `Process Listener` has observed the creation (usually a fork-exec combination), a suitable process descriptor must be created. To this purpose, the classes `Process` and `Process Manager` have been implemented. This manager is a sibling of the `Application Manager`, already visible in Figure 1. This means that, while the latter will be in charge of managing the adaptive applications (implemented via AEM or MANGO programming model), the former will keep track of the status of the generic processes.

At this point, new schedulable entities have been queued, and the need of executing the resource allocation policy is triggered. This will assign the resources needed to run. This is performed by sending the optimization request notification to the `Resource Manager`. This will in turn call the policy entry point function. Further changes obviously involve the code of the policies, which now should pull schedulable entities also from the `Process Manager`.

When the termination of the managed generic process is intercepted, the management procedure follows the opposite flow. The process is marked as DISABLED, the policy is invoked, resources are released and the descriptor is destroyed.

2.4 Data provider interface

The BarbequeRTRM provides different interfaces to connect the resource manager to external components or frameworks, like for instance the Global Resource Manager.

In this subsection, we show the *Data Communication Interface*, which enables the BarbequeRTRM as data provider. More in detail, the resource manager can report about the status of the hardware resources or the running workload located at the managed node.

Data consumers can retrieve such a data by following a publish/subscribe paradigm. To this aim, the BarbequeRTRM installation comes with a suitable library (`libbbque_dci`), for the implementation of the client-side code. On the server-side instead, the `Data Manager` is the module devoted to the data collection activity.

Data clients can thus interact with this module through a set of API exposed by the library as shown by Figure 4. The communication protocol is based on a publish/subscribe schema managed by the `Data Manager` module. Third-party clients can interact with the Data Manager towards the API provided by the BarbequeRTRM Data Communication Interface (`libbbque_dci`). They can subscribe by specifying the type of data they are interested in, and the time basis according to which they will receive them. In this regard, the data exchange protocol provides two possible modes: **periodic** and **event-based** mode. The former allows the client to specify a time interval for periodic receptions of data. The latter leaves the data server in charge of sending updated data only when certain events occur.

The BarbequeRTRM sends to subscribers the status messages containing the requested data. Currently, the type of data includes *applications* and *resources* status. In the former case, the status message will contain information about the currently active applications, their resource

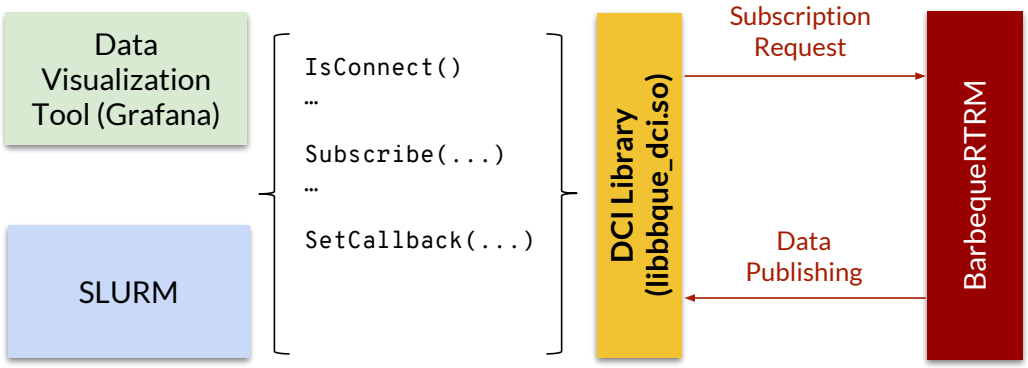


Figure 4: The interaction between the monitoring applications and the BarbequeRTRM through the Data Communication Interface.

mapping tasks, and the observed runtime performance. In the second case, the information provided includes the allocation of resources (computing units and memory) and, whenever available, their runtime status (e.g., load, temperature and power consumption).

As we said, on the `Data Manager` side (e.g., the publisher) there is a server in charge of managing incoming subscriptions. Its default TCP port is configurable at the BarbequeRTRM compilation stage, or changed later by modifying the BarbequeRTRM configuration file.

On the client side (e.g., the subscriber), the application has to instantiate the `DataClient` object. This class hides to the developer the connection details, providing member functions for the data subscription process. The object allows the client to set also a custom callback function, which will be executed everytime new data comes from the BarbequeRTRM Data Manager server.

The subscription process is based on the call of the `Subscribe(...)` function, by passing the arguments needed to specify the type of data the client is interested in, and the **mode** flag set to `SubscriptionMode_t::SUBSCRIBE`. In particular, the **Subscribe** function accepts the following parameters:

- `Filter`: it describes the type of status information the subscriber wants to receive.
- `Event`: it describes if the data delivery is periodic or event-based. In the latter case it specifies also which type of event triggers the publishing.
- `Period`: it represents the time interval to receive updates in case of periodic subscription.
- `Mode`: it indicates the mode of the request (e.g., subscribing or unsubscribing).

Multiple subscription requests can be sent through multiple subscription calls. The client can extend its subscription by sending further requests. New parameters must specify only the new data type in the filter attribute, for which it wants to receive data. If a new period is provided, this will supersede the previous one.

Similarly, the client can revoke its subscription by calling the DataClient::Subscribe(...) function, with the mode argument set to `SubscriptionMode_t::UNSUBSCRIBE`. The client can also selectively specify for which type of event or filter elements it wants to unsubscribe. A suitable

reliability management mechanism operates in case of unreachable client or server.

As mentioned above, the BarbequeRTRM will publish information by sending messages based on:

- Time period interval
- Event occured

In the first case, rate-based clients are updated according to their required rate. In the second one, each time a specific event occurs, affected status information is updated and all the subscribers will receive the new data. Three types of events are currently defined:

- **Application event**: it is triggered whenever there is a change in any application status (starting, ending...).
- **Resource event**: it is triggered whenever there is a change in any resource status (availability, thermal threshold reached...).
- **Schedule/allocation event**: it is triggered when a new allocation/schedule policy is executed.

Moreover, only the information required by the filter of each client is published to that client. These types of information are:

- **Application status**: it contains information about running applications, their performance and current resource allocation.
- **Resource status**: it contains information about available resources, their usage as well as power and thermal information.

Everytime new data comes from the BarbequeRTRM `DataManager`, the provided callback is called, so that the client can process the incoming information (i.e., storage, plotting,...).

Considering this feature of the BarbequeRTRM as the communicaton interface between the local and the global resource manager (SLURM), we need to reason on the current spectrum of data that the first can provide to the second, keeping in mind the reliability and the time constraints management goals pursued by the RECIPE project. For the first, we need to add some hardware reliability status information, that will be retrieved as it is explained in the next section. For the second instead, the BarbequeRTRM already performs a per-task monitoring of the application execution times. By forwarding this additional set of data, we enable the conditions for which the global resource manager could implement proactive policies, for preventing both reliability issues and deadline misses.

2.5 Hardware abstraction layer

In RECIPE, the hardware abstraction layer (HAL) is the entity in charge of enabling transparent and location-independent access to all heterogeneous resources in an unified way. Figure 5 illustrates the main software components this HAL is composed of.

The topmost layer in the figure is not part of the HAL, since it represents applications themselves. We have added this layer to the figure with the aim of showing the supported languages (C/C++)

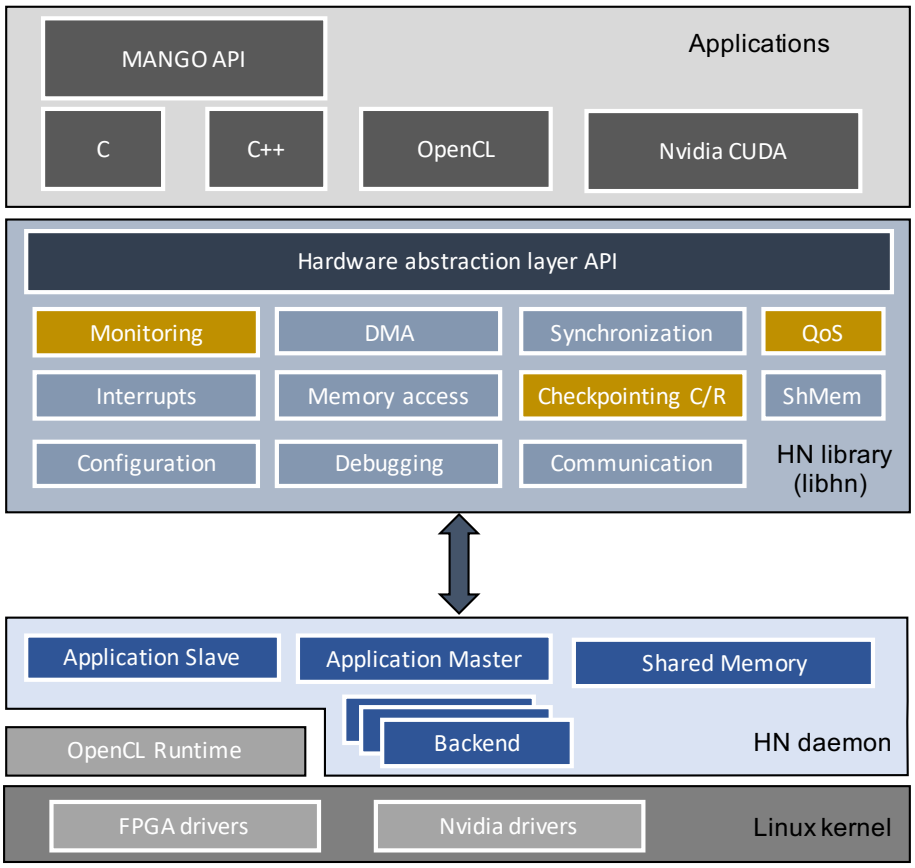


Figure 5: Overview of the components of the HAL.

and the user libraries (MANGO API, OpenCL and NVIDIA CUDA) that can interface with the bottom layers, which already implement the RECIPE hardware abstraction layer.

The next layer in the figure corresponds to the *HN library (libhn)*. This layer offers a public Application Programming interface (API) for accessing different heterogeneous resources in an unified way. Deliverable D2.1 provides a detailed description of this library, but in summary it offers:

- *Monitoring* to get status information of the heterogeneous components.
- *DMA* and *memory access* to transfer data between DDR memories located in the heterogeneous resources and hosts of the prototype.
- *Synchronization* to implement mutual exclusion for critical sections when different and concurrent collaborative kernels runs in the same FPGA.
- *QoS* to setup Quality-of-Service policies on the access to the resources.
- *Interrupts* to notify applications about kernel events occurrences.
- *Checkpointing C/R* to enable the mechanisms developed in WP4 to the upper layers.
- *ShMem* to implement an efficient data transfer mechanism between the applications and the HN daemon.
- *Configuration* to configure and run kernels, among other aspects.

- *Debugging* to debug the system.
- *Communication* to connect and communicate with the HN daemon.

As far as this document is concerned, the highlighted components of this layer will enable the BarbaqueRTRM to implement reliability and time-constraints management capabilities. To this end, the *monitoring* module will be modified to deliver the information needed for the reliability models. The *QoS* module will be created to enable the possibility of adjusting dynamically the quality of service levels each application requires. Finally, the *checkpointing C/R* will act as an entry point to enable checkpointing and restore primitives at the FPGA level. Nevertheless, the API for these modules have to be concreted in the next phase of the project, but the reader can already find a description of the fault-tolerance and QoS techniques proposed in deliverable D4.4.

The *HN daemon* is the other main component of the HAL. It is implemented as a user-space process and appears as middle-ware between the applications and hardware heterogeneous resources. It accesses the hardware through PCIe drivers mainly. For this purpose, the daemon uses a different *backend* instance for each hardware resource handled. It communicates with the application through the *HN library*, using an inter-process communication model based on Linux pipes. In this process an application connects to the *Application Master* who opens a private pipe and creates an *Application Slave*, which acts as a proxy inside the daemon. Then, all data communications between the application and daemon are carried out through its representative proxy, except for those related to high volume memory data transfers, which usually are performed using the *shared memory* mechanism enabled for that purpose.

Although the described above is the required from the point of view of the BarbaqueRTRM for satisfying its goal, we aim at analyzing the possibility of supporting location-independent resource access to native OpenCL and even NVIDIA CUDA applications as part of the activities related to WP4.

3 Reliability Management Support

In this section we describe the approach designed for handling reliability issues that could affect the hardware resources managed by the BarbequeRTRM. To this purpose, we needed to think about suitable extensions to introduce in the local resource manager, as explained in 3.1. These extensions must rely on a set of possible fault management actions (3.1), which can be performed only if properly supported by the underlying layers, i.e., hardware and operating system (3.2). Then the availability of such actions enables the possibility of implementing fault management strategies, following both a reactive and a proactive approach (3.3).

3.1 Fault management actions

In order to handle with occurrence of faults on a hardware architecture, a suitable resource manager must have the opportunity of performing low-level actions, involving the running workload and the underlying platform. These action shall allow us to react to faults or avoid them if possible, other than mitigating the consequences at application-level, like generating data corruption or leading to crashes during the execution of tasks.

To this purpose, we identified a set of common management actions, that we can implement or put in place, by integrating and exploiting tools already providing them.

Checkpoint

The execution of a task or application is dumped as a set of "image files" on a mass storage device. This action allows us to have a persistent copy of the application or task execution status. Accordingly, we can resume the execution of the task or application, later on, in case of faults leading to unexpected terminations or data corruption.

Restore

This is the complementary action with respect to *Checkpoint*. If we have a persistent copy of the task or application status, the *Restore* will consist of relaunching the execution of the task, starting from the checkpoint instead of starting over again. This possibility is crucial for critical HPC applications processing a huge amount of data (Big Data), for which a complete restart would represent a big penalty, other than a waste of time and money.

Freeze

The action of freezing a task or application consists of blocking it in a quiescent state. The application may still occupy system resources, but does not react to user input and does not provide any output. We can use this state to safely perform a migration of the application on a different node, or to recover from a transient fault by stopping it and resuming the execution with a *Restore*.

Migration

The term "migration" usually indicates the possibility of changing the set of processing resources currently allocated to run a given task or application. In most cases, this mechanism allows the operating system to balance the load of the processors and mitigate the thermal stress. If we can access information about the reliability level of a processor, we can use this action to dynamically move a task away from an "unreliable" set of resources. The complexity of the implementation of the task migration is however extremely platform-dependent. On shared memory systems, like multi-core CPU based ones, triggering a migration is trivial and does not require much additional integration effort from the resource manager perspective. The complexity of this action is typically handled by the operating system. Conversely, on heterogeneous systems, with basic runtime and featuring different memory domains, migrating a task (kernel) is a more complex operation. The interruption of a kernel require the availability of preemption or freeze mechanism, other than the possibility of moving the execution status of the kernel from one memory node to another. Lastly, the possibility of resuming the execution from the status information. Finally, one last thing to keep in mind is that, especially in HPC system, the resource manager should take into account that the migration has a cost in terms of overhead. This cost, as it should be easy to imagine, depends on the platform and the placement of the specific source and destination processors and memory nodes.

3.2 Low-level support

As we explained in the previous section, the effort to carry out, in order to allow the BarbequeRTRM to perform the aforementioned management actions, depends on the specific target platform. When the target is represented by a Linux-based system, running tasks on CPUs, the management actions described are already provided by existing tools or Linux frameworks.

The possibility of *freezing* a process, for instance, is a built-in functionality that the Linux kernel expose to user-space via the `freezer` cgroup subsystem [14]. Similarly, the possibility of *migrating* the tasks, is implicitly provided by the cgroup subsystems already exploited by the BarbequeRTRM, to enforce the resource assignments defined by the policies.

For *checkpoint* and *restore* instead, the support provided by the Linux kernel required the implementation of a specific user-space tool. To this aim, the most known tool, already available, is CRIU (Checkpoint/Restore in Userspace) [8]. This tool comes with a library, that the BarbequeRTRM can link to get access to the checkpoint/restore functionalities. This means that, most of the effort is represented by the integration of CRIU, and the introduction of a set of changes in the internal data structures of the resource manager, needed to control the execution status of the managed applications.

In RECIPE, we also target heterogeneous hardware platforms, featuring FPGA-based computing resources, like HLS accelerators and programmable GPU-like soft-core as discurred in deliverable D4.1. No tools are currently available for this custom hardware. The implementation of the management actions therefore, requires a specific development effort. As outcome of WP4, we expect the possibility of relying on a custom tool, providing a library for easy integration in the BarbequeRTRM. This would enable the possibility of performing the same management actions,

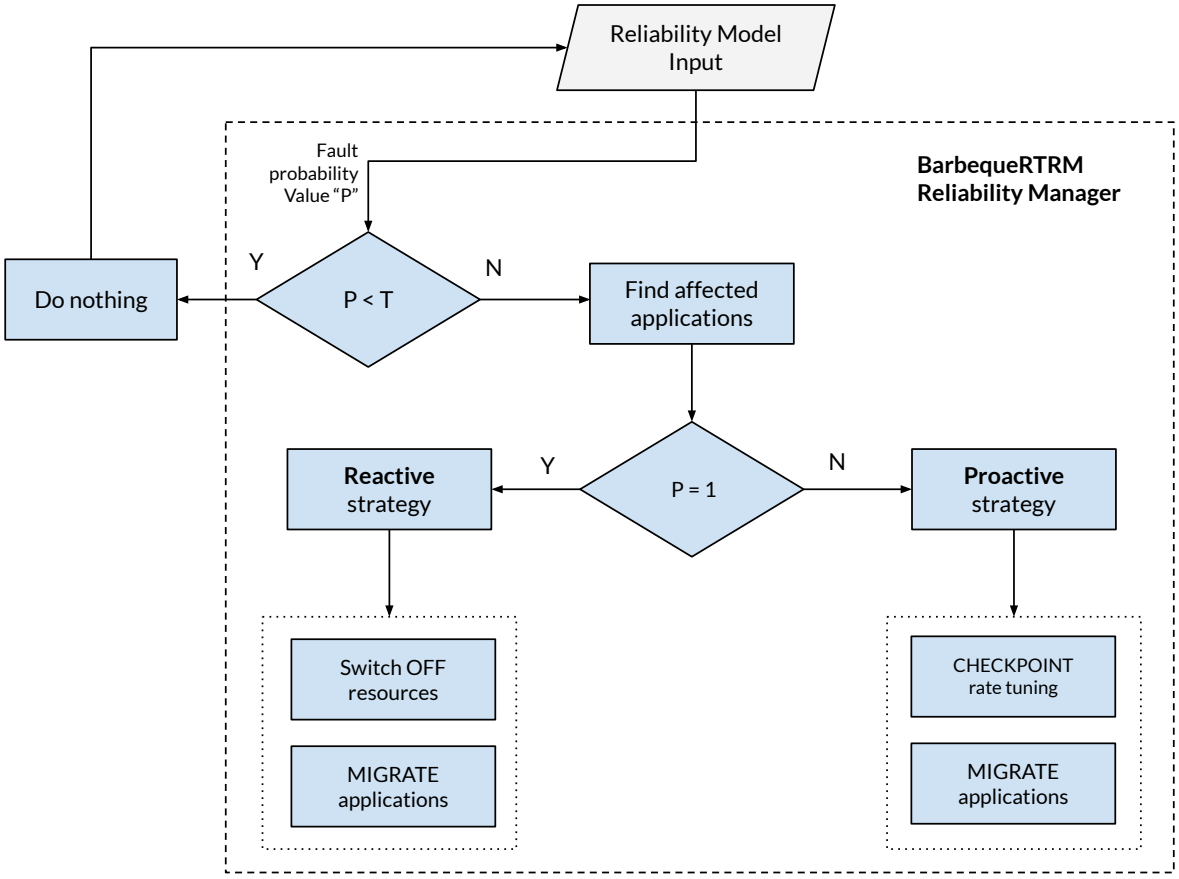


Figure 6: Early BarbequeRTRM reliability support proposal and external models interfacing.

in a portable manner, whatever the target hardware is, i.e. a homogeneous set of multi-core CPUs or a heterogeneous system.

3.3 Fault management strategy

In Figure 6, we sketched the management flow that we aim at putting in place via the BarbequeRTRM, in order to handle or prevent the occurence of faults and failures. The figure aims at providing an overview of the reliability-aware resource management capabilities. The detailed sequence of management actions will be provided in the next deliverable, once the reliability-driven policy will be implemented.

Looking at the figure, we can observe on the top the presence of a *Reliability Model* (WP3 outcome) feeds the BarbequeRTRM with data about the reliability status of the managed resources. Such data can consist of *probability values* P, in the [0..1] range, representing predictions. The interface between the resource manager and the modelling framework is described in deliverable D3.2. Then, depending on the resource and the application requirements (e.g., priority, critical level, ...), we can set a *probability threshold* T, under which the resource is considered not completely reliable. Therefore, we need check the application currently using the resources, i.e., the applications potentially affected by the probability of a fault, and decide if the fault probability

is worth considering or not. The former case would lead our local resource manager to follow a *Proactive strategy*. This means takes decisions in advance, trying to prevent the occurrence of faults and related consequences on the applications execution. A possible strategy could consist of migrating the workload on set of reliable resources or increase the checkpoint rate, so that in case of fault the loss of data is minimimized and the execution can be safely restored. Obviously the specific strategy will depend on the policy implemented.

If the probability value is $P = 1$ instead, the model is notifying the resource manager about the fact that a fault or failure has already occurred. In such a case, the resource manager must properly react. In the figure, we show a possible *Reactive strategy*, consisting of disabling the faulty resources and forcing a rescheduling of the applications. This leads to a migration of the affected workload onto a different set of processing resources. Here, the selection of the target resources depends on the actual policy in execution. For example, some policies can consider the history and rank the resources according to a reliability index.

3.4 Extensions

Reliability actions

As explained, the implementation of what we called reliability actions is deeply bound to the support provided by the hardware. This means that the `Platform Proxy` component, visible in the overall BarbequeRTRM layered architecture shown in Figure 1, has to be extended with an additional internal API. The `Platform Proxy` provides the interface between the hardware abstraction level and the resource manager internals. Therefore, we need to introduce at this level, the functions that will be called by the resource manager components involved in the reliability management strategy, to perform the freeze, the checkpoint and the restore of an application.

To this aim, we introduced the abstract class called `ReliabilityActionsIF` and show in the UML class diagram in Figure 7. The class requires the implementation of the following member functions:

- `Dump()`: perform the application checkpoint;
- `Restore()`: resume the application execution from the last checkpoint;
- `Freeze()`: freeze the application execution;
- `Thaw()`: thawn a previously frozen application;

While, the base class `Platform Proxy` provides an empty implementation of such functions. The derived class `Platform Manager` implements the functions working as dispatchers towards the `Local Platform Proxy` or the `Remote Platform Proxy` instances, depending on where the application is expected to run. In the context of RECIPE, the BarbequeRTRM operates as local resource manager, therefore the remote applications are not part the management flow. Finally, the `Local Platform Proxy` will call the implementations provided by the `Linux Platform Proxy` and the `MANGO Platform Proxy`, to perform the required action on the part of the application running on CPU and on the FPGA-based resources, respectively.

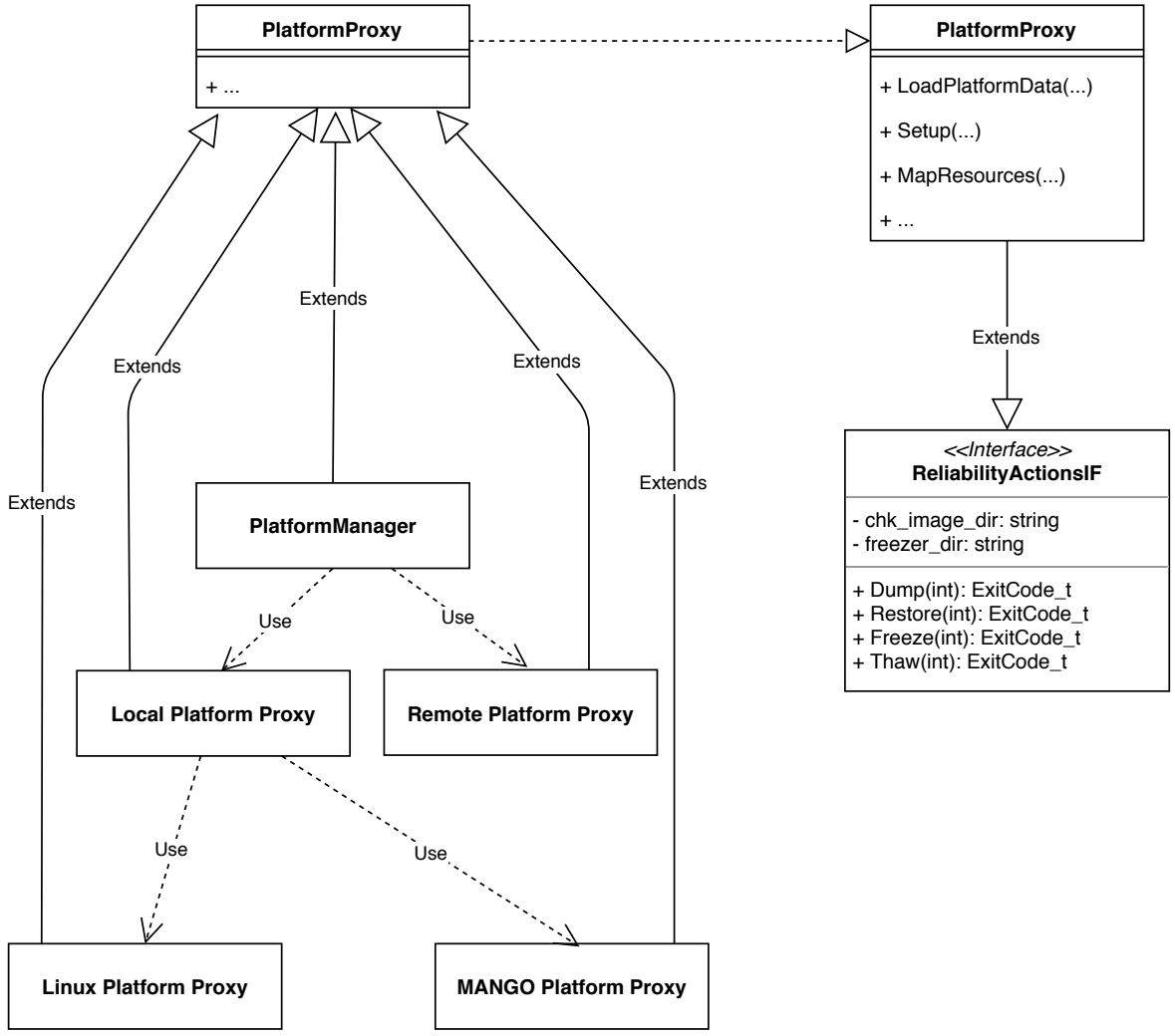


Figure 7: BarbequeRTRM: PlatformProxy interfaces extended with ReliabilityActionsIF.

Concerning task migration instead, as we said, this mechanism is already available for CPUs, while the aforementioned actions represent pre-requisites for implementing task migration among FPGA-based resources. As a consequence, by extending the `MANGO Platform Proxy` with the correct implementation of the `ReliabilityActionsIF` here described, we should be able to implement a task migration mechanism also for kernels running on heterogeneous units.

Reliability manager

The `Reliability Manager` represents the core component of the reliability-aware management support. It is in fact, the part of the local resource manager in charge of monitoring the status of the managed hardware resources. In the context of RECIPE, the monitoring task will exploit the *Reliability Model* provided as outcome of WP3. To this aim, we developed the *Hardware Reliability Library* (`libhwrel`) as interface between the resource manager and the modelling framework. More details on the library are reported in deliverable D3.2.

Once the model is available, the monitoring task can be accomplished by following two possible

approaches: *ondemand* or *periodic*. In the former case, a possible user, e.g., the resource allocation policy will require the reliability status (fault probability) related to each resource, every time the policy is executed. While, in the latter case, the `Reliability Manager` will spawn a thread in charge of periodically check the healty conditions of the hardware. In case of fault detection or fault risks, this will be the resource manager component in charge of triggering the reliability management strategy.

Workload status management

The possibility of freezing applications must be properly handled, considering two new possible status for managed applications and processes. The first one is FROZEN, and must be set once the freeze action has been successfully performed. The other one is THAWED, and must be set when we decide the resume a frozen application of process. Thawed applications must be included in the queue of applications for which a new run of resource allocation policy is needed. This require, first, to extend the set of possible expected state transitions in `Application` and `Process` descriptors, and how they are managed (and allowed) by the respective `Application Manager` and `Process Manager`. Then, current resource allocation policies must be properly modified by adding the code to loop on the queue of thawed applications.

Programming models

As we stated among the goals of the project, we aim at improving the reliability of the target HPC platforms, with no or minimum involvement from the application development side. This means that, at local resource manager level, the management layer will be responsible of performing periodic checkpoints of the applications and restoring them in case of faults. This will be done in a transparent manner with respect to the application, which means no changes and no additional function calls in the application code.

However, some application developers may want to control the checkpoint rate or force the checkpoint under certain circumstances. To this aim, we can extend the API of the two runtime managed programming models with the following self-explanatory functions, starting from the Adaptive Execution Model.

```
void SetCheckpointRate(int every_nr_run);
void CheckpointNow();
```

The first function allows the developer to configure the application checkpoint rate. The rate is expressed in number of `onRun` executions. Once the given number elapsed, the control thread sends a checkpoint request to the local resource manager. The second option instead represents a request for performing the checkpoint at the time of the function call.

Accordingly, these functions can be wrapped and made available at MANGO Programming library level. In such a case, we need to consider the two different APIs supported (C and C++). From this, for the C API we will have the following one-to-one matching, with respect to the AEM:

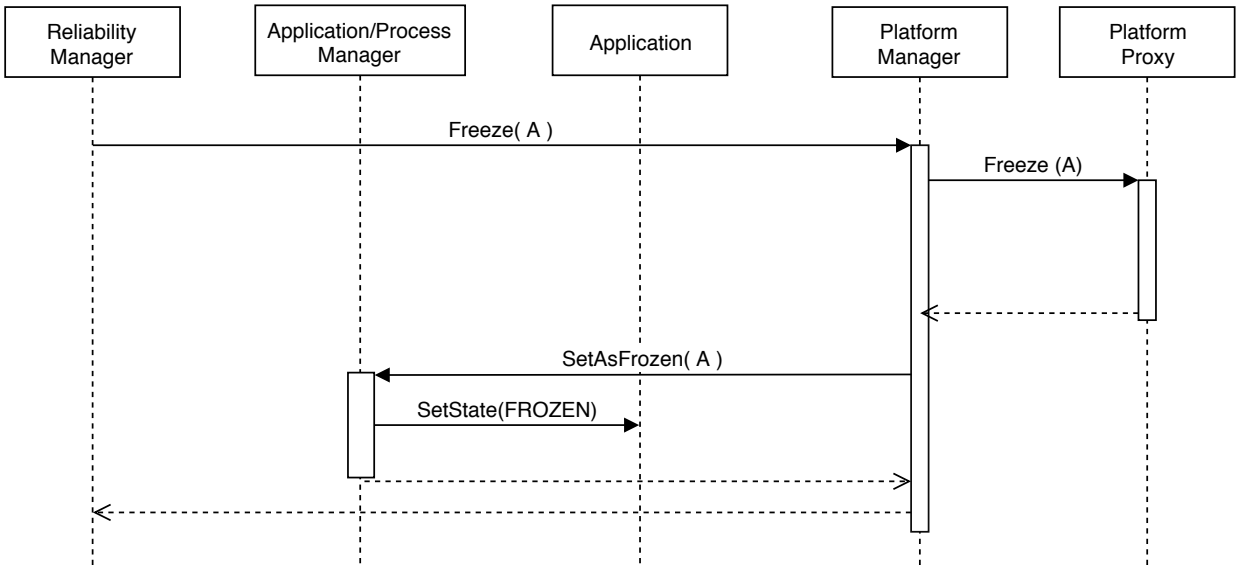


Figure 8: UML sequence diagram showing the application freezing.

```
/* C */

void mango_set_checkpoint_rate(int every_nr_run);
void mango_checkpoint_now();
```

For the C++ API, the aforementioned functions will be members of the class `BBQContext`:

```
/* C++ */

class BBQContext {

public:
  ...
  void SetCheckpointRate(int every_nr_run);
  void CheckpointNow();
};
```

Putting all together

Once the `Platform Proxy` instances have been extended with the reliability management interfaces, we can build the necessary interaction among the BarbequeRTRM components, in order to properly put in place the aforementioned reliability-driven strategy and performing the selected management actions.

In Figure 8 we sketched the UML sequence diagram showing the interaction among the BarbequeRTRM internal components. We can see the `Reliability Manager` triggering the freezing

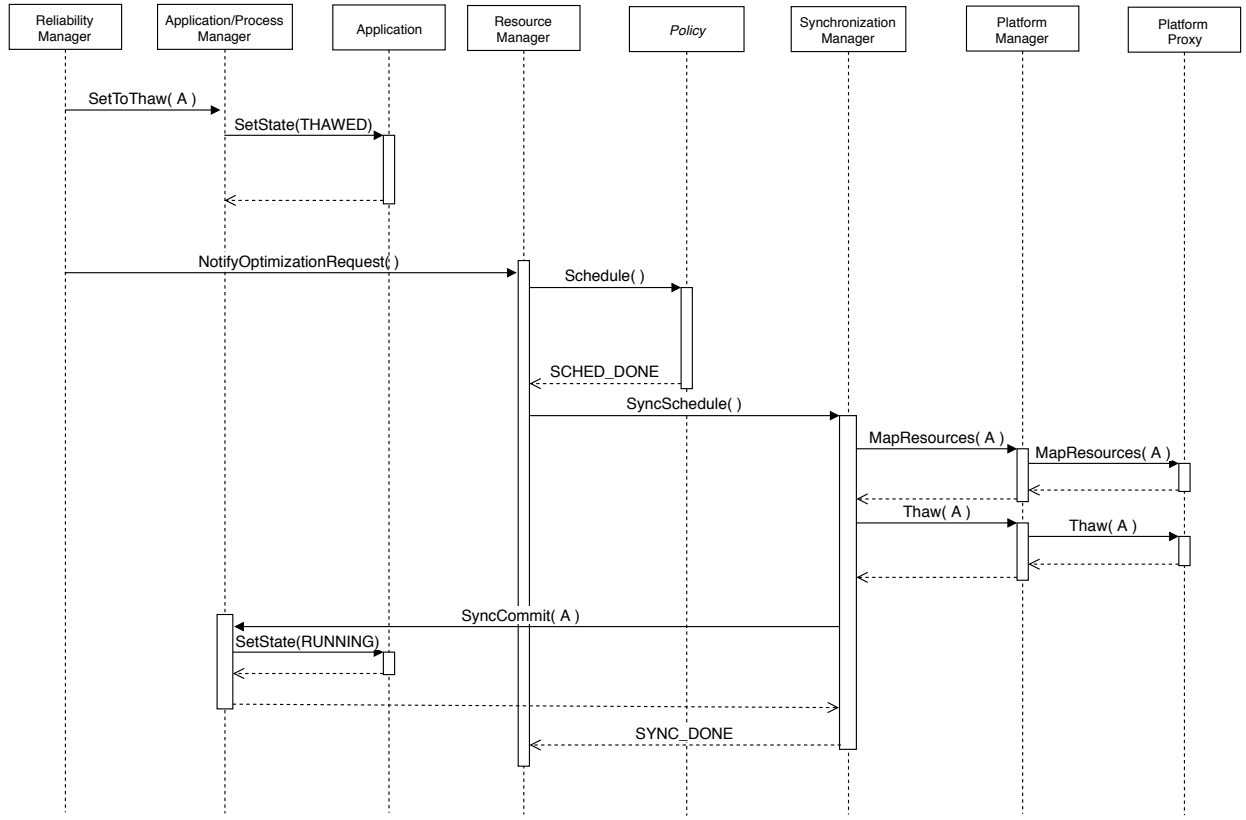


Figure 9: UML sequence diagram showing the application thawing.

request, by calling the `Freeze()` member function of `Platform Manager`. This will forward to call to the specific `Freeze()` function, implemented by each loaded `Platform Proxy`. This means that if we have a multi-tasking application (implemented by using the MANGO programming model) running on a heterogeneous set of processing units, the `Platform Manager` will call the function implemented in the `Linux Platform Proxy` for freezing the process or threads running on CPUs and the `MANGO Platform Proxy` for freezing the kernels running on FPGA processors and accelerators. Finally, once the application has been actually frozen, the `Platform Manager` can update its status information by setting it to "FROZEN" via `Application Manager`. For non-adaptive applications (e.g. generic processes), the sequence is the same except for the manager involved. In this case in fact, the `Process Manager` comes into play instead of the `Application Manager`.

Reverting the effect of a freeze, i.e. thawing an (adaptive) application or a managed process, requires the `Reliability Manager` to notify the `Application Manager` or `Process Manager` first, in order to put the description in the proper scheduling queue. To this aim, we introduced the member function `SetToThaw()` for both the managers. This has also the effect of updating the internal state of the related `Application` or `Process` instance. Then, in order to properly resume the execution, a resource allocation policy run is needed. Accordingly, an optimization request must be notified to the `Resource Manager`, afterwards the selected policy is executed. Once the policy has completed its run, the synchronization phase starts to make the resource assignments effective. The `Synchronization Manager` invokes the `MapResources()` function of the `Platform Manager`, which forwards the call to the respective `Platform Proxy` modules, depending on the type of resources. Once the assignment becomes effective, we are ready to resume the application

or process. The `Thaw()` function call follows the same flow of `MapResources`, and is responsible for resuming all the parts of the application, i.e, in case of multi-tasking application (e.g., MANGO application), the specific `Platform Proxy` version will resume the execution of kernel running on the target part of the system (e.g., CPU or HW accelerator). If the thawing is successfully completed, we can update the state of the application or process, setting to RUNNING, through the `SyncCommit()` function of the respective manager.

Checkpoint and *Restore* will follow a very similar approach, therefore we skip their description.

4 Timing Analysis Support

In the RECIPE project, one of the objectives of the local resource manager is to allocate computing resources to applications, providing the possibility of guaranteeing performance and timing constraints, making the system **real-time**.

The programming model developed during the MANGO project, requires that the application would provide a description of the inter-dependencies of the tasks composing it. In other terms, which are the shared memory area read or written by the tasks, and the timing requirements (e.g., task deadline). This description, in form of task-graph, is forwarded to the resource manager, which is responsible of allocating resources for each task (kernel) and memory buffer included in the graph. At run-time, the runtime library (RTLib), linked to both Adaptive Execution Model and MANGO applications, allows us to profile the execution time of each task and provide a feedback to the resource manager. Given the availability of these features, the BarbequeRTRM can be developed further to guarantee the application timing requirements, by introducing two functionalities: (1) analysis of the tasks execution time; (2) time-constrained resource allocation policy.

4.1 Background

The estimation of the *Worst-Case Execution Time (WCET)* is essential for real-time systems, in which the timing constraints required by the tasks must be guaranteed. A scheduling policy requires the timing analysis to estimate a WCET value for the tasks requiring the highest service level (maximum timing guarantees), that is greater or equal to the real WCETs. On the other hand, these estimations should be as tight as possible to the real WCETs, in order to minimize the resource assignment over-provisioning.

Recently, getting a non-underestimated but tight WCET has become a challenging problem. The growing computational power demand of systems, in addition, but opposed to, the reaching of technology limits, is increasing the hardware complexity of processors – such as the introduction of many-cores, multi-level caches, complex pipelines, etc. This is especially true in HPC, where the computing cluster is extremely complex. This leads to hindering the use of traditional WCET estimation techniques [12] [13] [6]. The problem is even magnified when dealing with Commercial-Off-The-Shelf (COTS) components, very diverse timing requirements and general-purpose operating systems [17] [18].

Given the aforementioned scenarios, **probabilistic real-time** has been proposed as a possible solution to WCET estimation problem. This approach is founded on the well-known *Extreme Value Theory (EVT)*, which is typically applied to natural disaster prediction, For example, to estimate the probability of unseen catastrophic floods. The use of EVT in real-time embedded systems has been proposed at the beginning of 2000s by Burns et al. [7] and Bernat et al. [5].

Probabilistic real-time based approaches can be divided into two classes [1]: *Static Probabilistic Time Analysis (SPTA)* and the *Measurement-Based Probabilistic Time Analysis (MBPTA)*. MBPTA, used in this project and subject of a subsequent deliverable D3.4, has been proposed to estimate the so-called probabilistic-WCET (pWCET), by directly sampling the execution

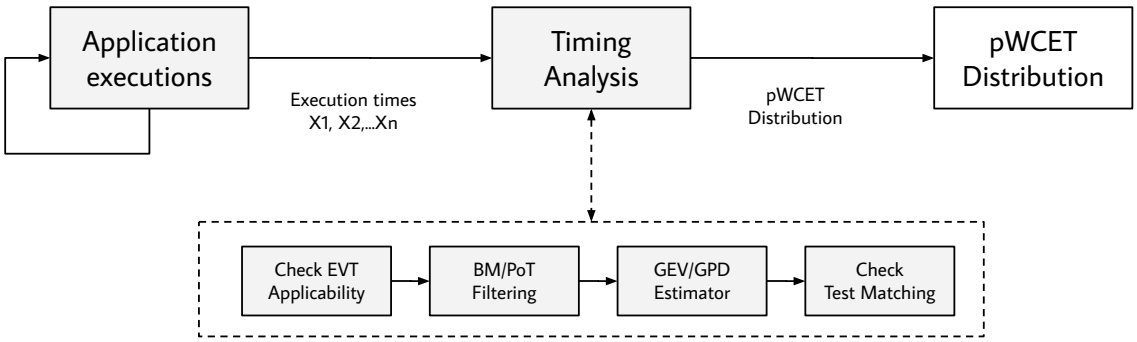


Figure 10: Offline phase of the timing analysis.

times of the tasks. Unlikely the classical WCET estimations, the pWCET is not a single value. Rather, it is a statistical distribution, characterized by the following *complementary cumulative distribution function (ccdf)*:

$$p = P(X > \overline{WCET})$$

where X is the random variable representing the task execution time. By using this distribution, it is possible to compute the probability of violation (p) of a given $\overline{WCET}$ or, vice versa, the $\overline{WCET}$ given the probability of violation (p). The pWCET is considered *safe* if the estimated distribution "upper-bounds" the worst-case execution time with a probability value equal or higher than the real one.

4.2 A two-phase strategy

The BarbequeRTRM analyzes the timing profile of applications (or tasks), in the sense that it observes the execution times of the tasks and, thanks to the related tools, described in deliverable D3.2, characterizes them with a statistical distribution. Since we are interested in guaranteeing that the worst-case execution time (WCET) of the tasks does not exceed their deadlines, the statistical distribution usually represents the WCET values and not a generic execution time. Accordingly, the distribution is called probabilistic-WCET (pWCET).

The timing analysis is initially performed offline, as shown in Figure 10, to profile the applications tasks, running on target processors, in a sort of training phase. This can also be done online, provided that the application runs multiple times and multiple deadline misses can be tolerated during this training phase.

After the training phase, the application is executed on the processing resources allocated by the time-constrained policy (see next subsection). At run-time, the BarbequeRTRM keeps gathering execution time samples of the tasks. These samples can be used to check if the behaviour of the application changes: too large values may hinder the reliability of the estimated pWCET distribution; too small values means that we are reserving more resources than necessary to the application. How to detect if the application is in any of these two situations (and then re-trigger the scheduling policy) is still an open problem. We are currently exploring two possible

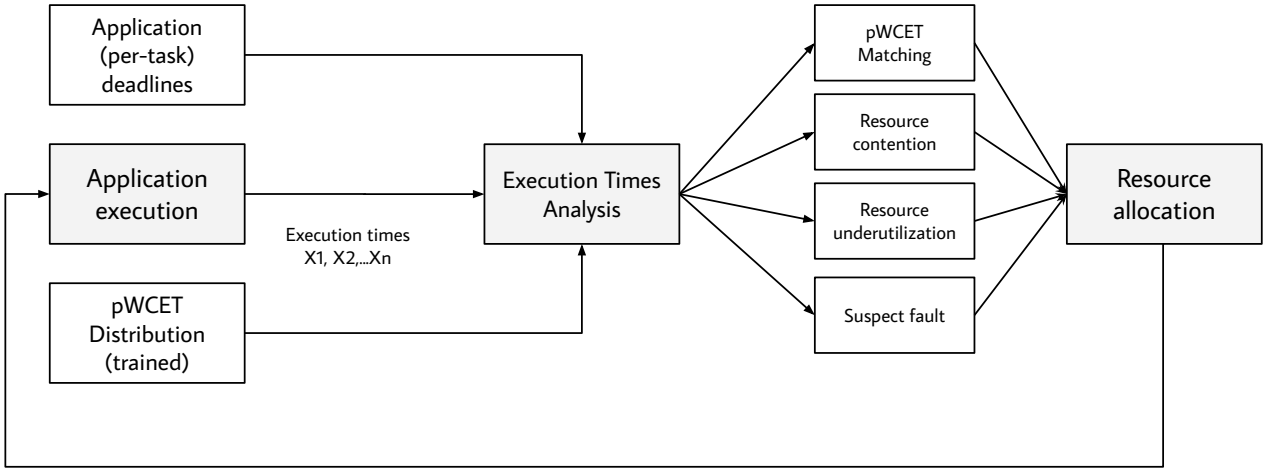


Figure 11: Online phase of the timing analysis: the execution of the application is profiled and the resource allocation policy is driven by the matching of the deviation of the execution times from the pWCET distribution.

approaches: (1) using the samples to periodically re-estimate the pWCET distribution and use a fault-detection mechanism on the distribution parameters; (2) using the samples as input of a Goodness-of-Fit (GoF) statistical test against the previously estimated distribution. Preliminary results showed that the first method is more reliable than the GoF tests, but it is also much more expensive, in computational terms. By saying more reliable we mean that it showed the occurrence of less false-positive and false-negative results. Further investigations of both approaches are needed and planned in next few months.

4.3 Time-constrained resource allocation policy design

When an application is started or a rescheduling is required by the timing analysis tool or by the reliability manager, the BarbequeRTRM invokes the scheduling/allocation policy. Writing such policy is not trivial. The resource manager has to take into account that the WCET is influenced by several factors:

- amount of resources allocated (e.g. network bandwidth assigned)
- type of resources (CPU, GPU, custom accelerator, etc.)
- presence of co-running tasks on the same CPU
- presence of co-running tasks that shares some hardware (e.g. cache, bus, etc.)
- current DVFS settings according to temperature and power constraints
- application/task accessing external I/O devices

Obtaining a deterministic relation between such factors and the application or task WCET is in general infeasible. A smart algorithm that learns online the application behaviour in according to such factors is required. According to application requirements and criticality, different priority

levels can be considered. For example, meeting the timing deadline can be more important compared to reliability issues, at least for time-sensitive applications. Vice versa, long running batch applications may require strong reliability requirements and less strict, or not even not interested in, timing constraints.

Another possible approach we would like to exploit is an approach coming from Mixed-Criticality (MC) systems: to assign a priority (criticality) level $L = \{1, 2, ..., N\}$ to each task. The higher the priority, the higher the importance of the task. Usually, the priority levels are limited and represented with $L = \{LO, MI, HI\}$. A mixed-criticality scheduler or resource manager must guarantee the execution of the jobs of tasks of a certain priority $l \in L$ even by dropping the jobs of tasks of lower priority level $l' < l$. This is the major difference with respect to the traditional priority schedulers. This concept has been studied for decades in embedded sytems but not yet in HPC. In traditional MC systems, the WCET of the tasks are estimated per priority levels, i.e. a task with priority l has $C_1, C_2, ..., C_l$ estimated WCETs. In our scenario, the different WCETs can be computed easily thanks to the estimated probability distribution function.

Having applications with different real-time requirements, makes the problem of balancing the worst-case performance with the average-case performance extremely interesting. If the tasks of an application have no strict timing constraints, or according to the MC-like case a low priority, its tasks can be scheduled in order to reach the maximum throughput, i.e. the maximum average performance. Instead, the highest priority tasks have to be scheduled with the larger probability of meeting their deadline, i.e. according to their WCET values. The co-existence of tasks with different timing constraints makes the resource allocation policy challenging, especially when tasks share resources and resource contention may inflict timing penalties to the execution of other tasks.

4.4 Extensions

The introduction of time-constraints management capabilities in the BarbequeRTRM is a much less invasive task with respect to the changes required by the reliability management support. What we need here is to profile the execution of the application at runtime and allows the resource allocation policy to access such profiling data. The joint work performed by the resource manager and the application run-time library (RTLib) already allows us to collect time samples of the application executions and its tasks, in case of MANGO application running multiple kernels. These samples will be forwarded to the *Timing Analysis Tool* shown in Figure 10, in order to get a probabilistic model of the WCET of the application (or the single tasks), as already explained. To this aim, we developed the *Timing Analysis Library* (`libta`). More details on the library can be found in deliverable D3.2.

The missing part consists of storing the probabilistic model, such that the resource manager could retrieve it at runtime. In this regard, the most straightforward strategy for us is extend the application "recipe" file, described in deliverables D2.1 and D2.4 with an additional section. Then, for the runtime exploitation, we will need utility functions to quickly check the matching between the actual execution times and the pWCET model. This, jointly to the resource control mechanisms already available or that will be integrated in the next months, complete the set of requirements to satisfy in order to enable the possibility of adding time-constraints driven resource allocation policies to the BarbequeRTRM.

Finally, the actual design of the policies will necessitate a suitable experimental campaign, consisting of 1) executing benchmarks or micro-kernels on the RECIPE target hardware, in order to validate the methodology behind the timing analysis; 2) observe the application executions in realistic scenarios, where in the resource manager has to deal with managing resource contention and notification of faults occurrences.

5 Conclusions

In this deliverable, we described the developments recently introduced in the local manager prototype (current BarbequeRTRM development version), in order to open the road to the forthcoming reliability and time-constraints management policies. The changes have required the introduction of new components, the extension of existing internal interfaces and the integration of platform-specific supports for freezing, thawing, checkpointing and restoring the execution of the managed applications.

For the time-constraints management part instead, the resource manage will feed an external timing analysis tools with profiling information, including the execution times of the applications and related tasks. The results of the analysis will be then exploited at run-time to drive the policy in the selection of the processing units to assign or to detect anomalies, due to 1) resource contention, 2) resource underutilization or 3) possible occurence of faults.

Next steps, will go in the direction of completing the integration between the local resource manager and the other components of the stack (global resource manager and programming models), other than exploiting the reliability support coming from WP4 through the hardware abstraction layer, and the modelling tools developed in WP3. These pieces, all together, will enable the possibility of actual implementing the reliability-aware and time-constraints driven resource allocation policies for applications targeting HPC systems, which is the ultimate RECIPE development effort regarding the BarbequeRTRM.

A Download and Installation

The local resource manager prototype (The BarbequeRTRM) has been made available through two public repositories. This allows the user to follow two possible approaches in the installation of the framework.

The first one consists of a stand-alone installation, including the resource manager and the programming library for implementing reconfigurable application, according to the aforementioned Adaptive Execution Model. In such a case, the repository (based on the GIT versioning system) is located at the following URL: https://bitbucket.org/bosp/bosp/src/bosp/.

The RECIPE prototype version is tagged as `recipe-v0.1`. Once the GIT respository has been cloned, the user can proceed as follows for the selection of the correct version.

```
$ cd bosp
$ git checkout recipe-v0.1
$ git submodule init  // if not downloaded yet
$ git submodule update --recursive
```

Then, for the configuration, compilation and installation we reported the steps on the official website of the BarbequeRTRM Open-Source Project (BOSP: https://bosp.deib.polimi.it/doku.php?id=installation)

The second option instead, consists of the MANGOLIBS project, which includes most of the software stack released with the H2020 MANGO project and that will be further developed during the RECIPE project. This repository allows the user to download the following components:

- BarbequeRTRM: the local resource manager and the Adaptive Execution Model library
- HN library/daemon: the hardware abstraction layer on top of the FPGA-based processing resources
- MANGO library: the programming library for MANGO and RECIPE heterogeneous platforms
- Toolchains for compiling the application kernels, targeting processing architectures used in MANGO
- Sample applications for testing and development reference purposes.

The GIT repository, in this case, is located at the following URL: https://bitbucket.org/mango_developers/mangolibs/src/master/. The reference version for this prototype release is tagged as `v0.5` and, similarly to BOSP, it can be selected as follows:

```
$ cd mangolibs
$ git checkout v0.5
$ git submodule init  // if not downloaded yet
$ git submodule update --recursive
```

The configuration, compilation and installation of all the components of the stack included in the MANGOLIBS project has been documented in the public deliverables ("MANGO User Guide") of the MANGO project. Currently, the procedure is still valid, although the functionalities that RECIPE will introduce are not all available yet.

References

[1] J. Abella, D. Hardy, I. Puaut, E. Quiones, and F. J. Cazorla. On the comparison of deterministic and probabilistic wcet estimation techniques. In *2014 26th Euromicro Conference on Real-Time Systems*, pages 266–275, July 2014.

[2] D. Aikema, C. Kiddle, and R. Simmonds. Energy-cost-aware scheduling of HPC workloads. In *2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, pages 1–7, June 2011.

[3] AMD. AMD Display Library (ADL). https://gpuopen.com/gaming-product/amd-display-adl-library/.

[4] Patrick Bellasi, Giuseppe Massari, and William Fornaciari. Effective Runtime Resource Management Using Linux Control Groups with the BarbequeRTRM Framework. *ACM Transactions on Embedded Computing Systems (TECS)*, 14(2):39, 2015.

[5] G. Bernat, A. Colin, and S. M. Petters. Wcet analysis of probabilistic hard real-time systems. In *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.*, pages 279–288. IEEE, 2002.

[6] C. Brandolese, S. Corbetta, and W. Fornaciari. Software energy estimation based on statistical characterization of intermediate compilation code. In *IEEE/ACM International Symposium on Low Power Electronics and Design*, pages 333–338, Aug 2011.

[7] A. Burns and S. Edgar. Predicting computation time for advanced processor architectures. In *Proceedings 12th Euromicro Conference on Real-Time Systems. Euromicro RTS 2000*, pages 89–96, 2000.

[8] CRIU: Checkpoint/Restore in Userspace. https://criu.org.

[9] José Flich, Giovanni Agosta, Philipp Ampletzer, David Atienza Alonso, Carlo Brandolese, Etienne Cappe, Alessandro Cilardo, Leon Dragić, Alexandre Dray, Alen Duspara, et al. Exploring Manycore Architectures for Next-Generation HPC Systems through the MANGO Approach. *Microprocessors and Microsystems*, 2018.

[10] William Fornaciari, Giovanni Agosta, David Atienza, Carlo Brandolese, Leila Cammoun, Luca Cremona, Alessandro Cilardo, Albert Farres, José Flich, Carles Hernandez, Michal Kulchewski, Simone Libutti, José Maria Martínez, Giuseppe Massari, Ariel Oleksiak, Anna Pupykina, Federico Reghenzani, Rafael Tornero, Michele Zanella, Marina Zapater, and Davide Zoni. Reliable power and time-constraints-aware predictive management of heterogeneous exascale systems. In *Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, SAMOS '18, pages 187–194, New York, NY, USA, 2018. ACM.

[11] Eduardo Camilo Inacio and Mario A.R. Dantas. A survey into performance and energy efficiency in HPC, cloud and big data environments. *International Journal of Networking and Virtual Organisations*, 14(4):299–318, 2014.

[12] R. Kirner and P. Puschner. Obstacles in worst-case execution time analysis. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 333–339, May 2008.

[13] O. Kotaba, J. Nowotsch, M. Paulitsch, S. M. Petters, and H. Theiling. Multicore in real-time systems–temporal isolation challenges due to shared resources. In *Workshop on Industry-Driven Approaches for Cost-effective Certification of Safety-Critical, Mixed-Criticality Systems*, Grenoble, France, 2013.

[14] Linux cgroup: freezer subsystem. https://www.kernel.org/doc/Documentation/cgroup-v1/freezer-subsystem.txt.

[15] Arun Babu Nagarajan, Frank Mueller, Christian Engelmann, and Stephen L. Scott. Proactive fault tolerance for hpc with xen virtualization. In *Proceedings of the 21st Annual International Conference on Supercomputing*, ICS '07, pages 23–32, New York, NY, USA, 2007. ACM.

[16] NVIDIA. NVIDIA Management Library (NVML). https://developer.nvidia.com/nvidia-management-library-nvml.

[17] Petar Radojković, Sylvain Girbal, Arnaud Grasset, Eduardo Quiñones, Sami Yehia, and Francisco J. Cazorla. On the evaluation of the impact of shared resources in multithreaded cots processors in time-critical environments. *ACM Trans. Archit. Code Optim.*, 8(4):34:1–34:25, January 2012.

[18] F. Reghenzani, G. Massari, and W. Fornaciari. Mixed time-criticality process interferences characterization on a multicore linux system. In *2017 Euromicro Conference on Digital System Design (DSD)*, pages 427–434, Wien, Aug 2017. IEEE.

[19] Marc Snir, Robert W Wisniewski, Jacob A Abraham, Sarita V Adve, Saurabh Bagchi, Pavan Balaji, Jim Belak, Pradip Bose, Franck Cappello, Bill Carlson, Andrew A Chien, Paul Coteus, Nathan A DeBardeleben, Pedro C Diniz, Christian Engelmann, Mattan Erez, Saverio Fazzari, Al Geist, Rinku Gupta, Fred Johnson, Sriram Krishnamoorthy, Sven Leyffer, Dean Liberty, Subhasish Mitra, Todd Munson, Rob Schreiber, Jon Stearley, and Eric Van Hensbergen. Addressing failures in exascale computing. *The International Journal of High Performance Computing Applications*, 28(2):129–173, 2014.