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

RECIPE

# WP3 Predictive Reliability and QoS Enforcing Methodologies

# D3.4 RECIPE Timing Analysis Tools

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: D3.4 RECIPE Timing Analysis Tools**

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

| | |
|---|---|
| **Deliverable No**: | D3.4 |
| **WP No**: | 3 |
| **WP Leader**: | R. Canal |
| **Due date**: | 30/04/2020 |
| **Delivery date**: | 07/05/2020 |

**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 |

# D3.4 RECIPE Timing Analysis Tools

| | |
|---|---|
| **Work Package:** | WP3 Predictive Reliability and QoS Enforcing Methodologies |
| **Deliverable number:** | D3.4 |
| **Deliverable title:** | RECIPE Timing Analysis Tools |
| **Due date:** | 30/04/2020 |
| **Actual submission date:** | 07/05/2020 |
| **Editor:** | R. Canal |
| **Authors:** | R. Canal, M. Fusi, F. Mazzocchetti, L. Kosmidis, F.J. Cazorla, J. Abella, C. Hernandez, R. Tornero, J. Flich |
| **Dissemination Level:** | PU |
| **No. pages:** | 14 |
| **Authorized (date):** | 07/05/2020 |
| **Responsible person:** | W. Fornaciari |
| **Status:** | Draft |

**Revision history:**

| Version | Date | Author | Comment |
|---|---|---|---|
| v.0.1 | 13/04/2020 | | Outline and contributions identified |
| v.1.0 | 27/04/2020 | | Final version after internal review |

**Quality Control:**

| | Who | Date |
|---|---|---|
| **Checked by internal reviewer** | J. Abella | 27/04/2020 |
| **Checked by WP Leader** | R. Canal | 27/04/2020 |
| **Checked by Project Technical Manager** | G. Agosta | 07/05/2020 |
| **Checked by Project Coordinator** | W. Fornaciari | 07/05/2020 |

## 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 Fédèrale 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

High-Performance Computing (HPC) systems have become ubiquitous and are no longer concentrated in supercomputing facilities and data centers. While these facilities still exist and grow, a plethora of HPC systems building on large multicores and accelerators (e.g. GPUs, FPGA-based) are nowadays deployed for a variety of applications of interest, not only for large enterprises and public institutions, but also for small and medium enterprises as well as small public and private bodies.

The proliferation of HPC systems and applications in new domains has led to new requirements related to non-functional requirements (time, power, reliability, temperature, etc) and the implementation of platforms to satisfy them [2, 8, 7, 12, 13]. In this deliverable, we provide a description of the timing analysis tools developed in this project.This prediction of the (probabilistic) worst-case execution time will -later on- guide resource management, leveraging aspects related to the efficient use of the computing platform as a whole.

The rest of the deliverable provides details on the usage and capabilities of the software developed. The specific underlying techniques implemented are described in the previous deliverable D3.2.

The tools developed are available at: RECIPE's GIT repository and they are being integrated in the run-time manager (Task 3.5)

# 2 Summary of the Timing Analysis Methodology

Timeliness is a key non-functional requirement. Timeliness can be expressed in the form of strict real-time guarantees or quality-of-service (QoS), and is quite common in HPC applications related to big data in general and sustained input processing in particular. For instance, the result of a weather prediction or large simulation modeling the propagation of hazardous substances after an accident, needs to be completed in a given short time frame in order to be useful.

Predicting how much an application can take to run requires the use of representative execution time tests during the analysis phase. However, a number of execution time conditions are hard – if at all possible – to control or even to track, so that it turns out to be virtually impossible to relate execution time measurements in the analysis phase with those that may occur once the system has been deployed.

Difficulties arise from the way the Operating System (OS) places code, stack and heap data in memory, and the impact that such allocation has on cache behavior, contention in the access to shared hardware resources (e.g. the memory controller, or shared cache buses and buffers), among other effects. Therefore, performing an arbitrarily large number of experiments during analysis does not guarantee, in general, covering execution time conditions relevant during operation since those analysis conditions may preclude, by construction, some specific memory allocations that may lead to high execution times. Hence, the lack of controllability of those effects, which are managed in non-obvious ways by the OS, defeat any attempt to predict execution times based on test campaigns that lack means to trigger specific timing conditions. In order to address this challenge, (memory placement) software randomization (SWrand) has been proposed with the aim of enabling probabilistic coverage of the different execution time conditions [6, 10]. Those techniques randomize the physical location of code and data in memory, which indirectly randomizes their placement in cache and hence, their hit/miss behavior.

- Code Randomization: SWrand copies functions' code at random memory locations during execution with the objective of randomizing their cache placement and hence, how code conflicts in first level (L1) instruction caches with OS code, dynamic libraries and other applications, and additionally with data in caches shared amongst code and data.
- Stack Randomization: Dynamic stack randomization builds upon placing the function stack at a random location for each function. This can be done using indirections and allocating the stack from the heap, as in the case of code [10]. In particular, the solution builds upon having a pool of preallocated stack frames that are given to the functions called on demand. However, in the context of HPC applications, which may run multiple threads simultaneously, managing a pool of stack frames imposes the use of synchronization primitives and may decrease performance.
- Heap Randomization: SWrand, by default, does not consider heap randomization per se since, critical real-time embedded systems are not allowed to allocate memory dynamically. Hence, specific randomization for heap objects is not needed as a goal. However, SWrand builds upon random heap object allocation provided by Stabilizer [6], a compiler pass developed within the LLVM compiler and a runtime system based on Die-Hard[3] and Heap-Layers[4] providing random allocation of objects.

Overall, SWrand can be used for HPC applications, even if they are multi-threaded. However, specific considerations need to be taken into account for code and stack randomization, as detailed above. Once those considerations are taken into account, SWrand provides means to test any cache placement probabilistically, which calls for appropriate probabilistic means to analyze execution times obtained with SWrand-based test campaigns. Once the test campaign is in place, we needed to analyze the randomly sample execution time measurements, making considerations for its reliable use in the context of HPC applications. This has been described in D3.2 as the MBPTA-CV. The use of MBPTA-CV in the context of HPC applications is viable as long as particular considerations are taken into account. Those considerations relate to choosing appropriate sample sizes, with less demanding criteria than for critical real-time embedded systems, and appropriate measurement collection protocols to achieve i.i.d. at least for maxima. Constraints for exponentiality compliance can also be decreased, and acceptable exceedance probabilities may be, in general, higher than those for critical real-time embedded systems.

Overall, following the flow summarized here and detailed in D3.2, we can compute probabilistic worst-case execution time values in an HPC context.

# 3 Timing Analysis Tool

The timing analysis tool is in charge of performing statistical analyses over the set of measurements of tasks' execution time. The tool computes the resulting statistical distribution thanks to suitable statistical algorithms. The BarbequeRTRM provides to the timing analysis tool the list of timing measurements across a pre-defined timespan. The output of the library is a statistical distribution representing this sample. The usual distribution provided by the library is the probabilistic-WCET (Worst-Case Execution Time) distribution, i.e. the statistical distribution of samples at the extremes.

The interface to the timing analsysis tool is provided as a C++ library (`libta`). Its UML class diagram is depicted in Figure 1. It has the `Request`, `Response`, and `TimingAnalyzer` classes with the same meaning. The `Request` class is a template-parameter class, so the execution times can be provided in different types, e.g. integer or float. This has been done to maintain the flexibility of using the library with any timing measurement system. Its purpose is to store the vector of execution times to be analyzed. The `Response` class is instead a generic class that must be specialized. Currently, we proposed two possible responses: a statistical pWCET distribution or a fixed WCET value. The actual choice between them is library dependent, i.e. if it exploits probabilistic or not WCET estimation methods.

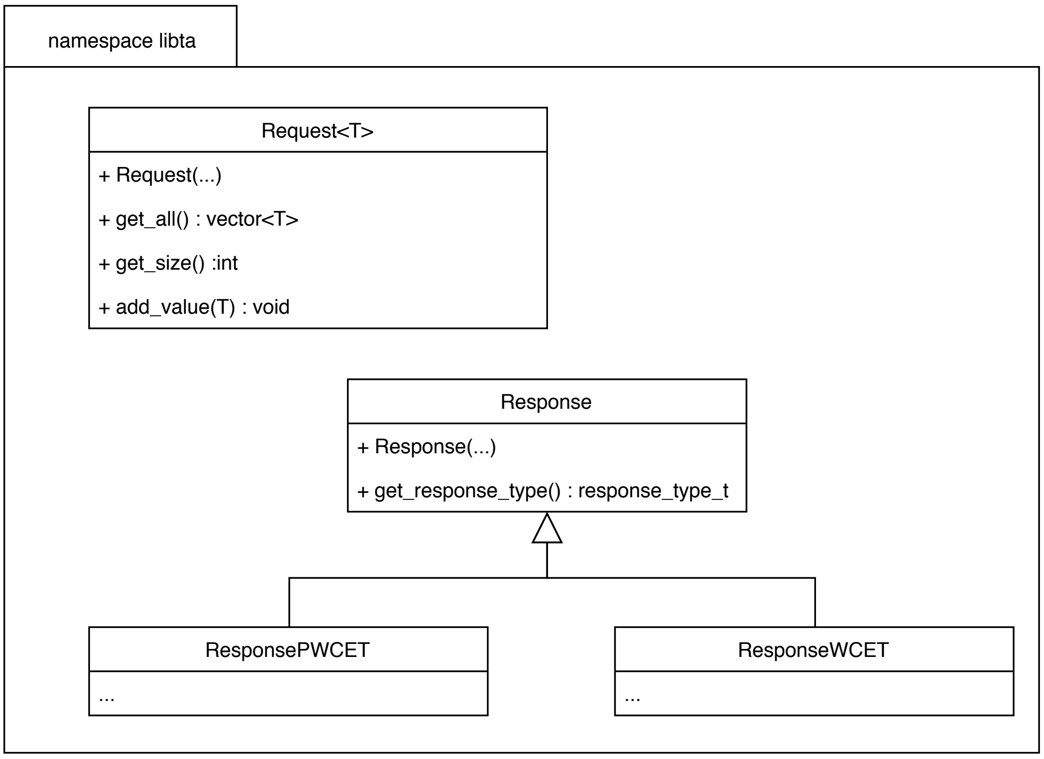

Figure 1: Timing Analisys library: UML class diagram.

## 3.1 *libta*: The API

*libta* [5] implements the technique of MBPTA-CV [1] in a header-only library. The library is template-based, so it is possible to support generic data structures. Error handling is performed

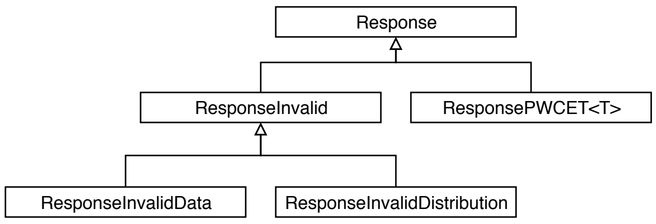

Figure 2: Inheritance diagram of `Response`.

–

through the `Either` class taken from the *neither* library [11], which offers a modern way of handling errors without the need of using return codes nor exceptions.

The API is contained with the `libta` namespace. In this section, such a namespace is not specified before class and method names for the sake of conciseness of the showed headers. The reccomendation is to use `float` and `double` datatypes to process data. Using `long double` may slow down the computation significantly.

The `Request` class represents the input for the library. It wraps a `std::vector`, but it may be easily changed if needed. These are the public methods:

`Request<T>()`: Empty class constructor.

`std::vector<T>::iterator begin()`: Begin iterator for for-range-loops.

`std::vector<T>::const_iterator cbegin() const`: Const begin iterator for for-range-loops.

`std::vector<T>::iterator end()`: End iterator for for-range-loops.

`std::vector<T>::const_iterator cend()`: Const end iterator for for-range-loops.

`void add_value(const T &time)`: Add new values to the timing array.

`const std::vector<T> & get_all() const`: Getter for the whole timing array.

`int get_size()`: Get the number of added values.

The `Response` class is a generic interface of the responses produced by the *libta* API. Most of the times it used to report errors, except for the `ResponsePWCET<T>` sub-class, which reports a valid timing estimate given an input probability. Figure 2 shows the inheritance diagram related to the `Response` class and the following list breafly describes the meaning of each `Response`:

`Response`: A generic response.

`ResponseInvalid`:A generic response with a string message.

`ResponseInvalidData`: This is returned when the a request cannot be used to estimate a distribution.

`ResponseInvalidDistribution`: This response is returned when it is not possible to produce a pWCET with a given distribution.

`ResponePWCET<T>`: A response obtained by succesfully getting a pWCET estimate.

The class `EVTDistribution<T>` represents an estimate of a heavy-tail distribution obtained by processing a `Request<T>`. A user should produce an `EVTDistribution<T>` instance using the `DistributionAnalyzer` class. This class offer the following public methods (note that the data type `EitherPWCETResponse<T>` is an alias for `Either< ResponseInvalidDistribution, ResponsePWCET<T> >`):

`EitherPWCETResponse<T> getPWCET (T probability) const`: Get the execution time that exceeds with the given probability.

`EitherPWCETResponse<T> getPWCETLow (T probability) const`:Get the execution time that exceeds with the given probability with risky assumption.

`EitherPWCETResponse<T> getPWCETHigh (T probability) const`:Get the execution time that exceeds with the given probability with safe assumption.

`T getMaxExecutionTime () const`: Get the collected maximum execution from the real values.

`const std::vector< T > getTailValues() const`: Get the real values used for the tail estimation.

`DistributionAnalyzer` is responsible of the production a heavy tail estimate through the `estimate_distribution` method, which requires a shared pointer that references a `Request` and a `const int rank_length` that tells how many points the output distribution should have. Setting a higher value of `rank_length` increases the accuracy of the result, but it slows downs the production of the curve. Such a method returns a `Either< ResponseInvalidData, EVTDistribution<T> >` instance.

## 3.2 *libta:* Call Flow

The standard usage of `libta` is shown in the Flow-Chart present in Figure 3.

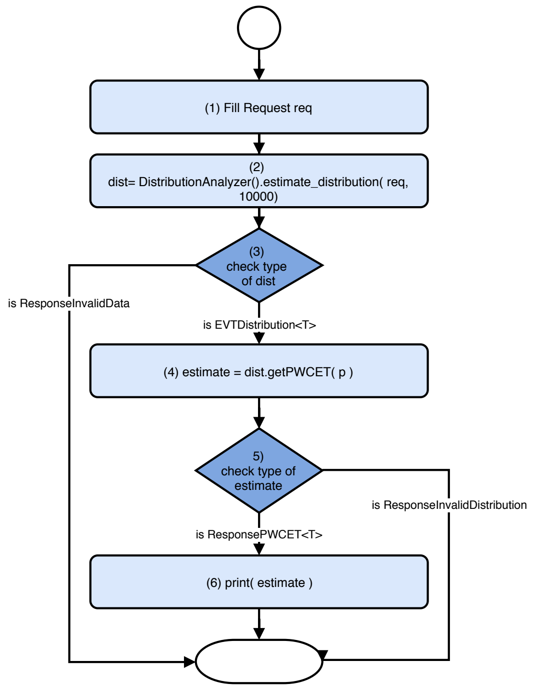

Figure 3: Flow-chart of the usage of *libta* to collect the minimum execution time that can be observed with a given probability $p$.

–

At first, a new instance of `Request` must be created and filled with samples the represents the execution time of a target (step 1 in the previously mentioned Figure), then the `estimate_distribution` method of the `DistributionAnalyzer` class produces an estimate of a heavy tail distribution in a `EVTDistribution` instance if it is possible to apply the MBPTA-CV on the given input, otherwise it returns a `ResponseInvalidData` (steps 2 and 3). This may happen if one of the number of CV values valid for estimating the heavy tail is smaller or equal than 10. If a valid distribution is returned, now it is possible to ask to the distribution the minimum execution time that can be observed with input probability $p$ by calling the `getPWCET` method (step 4). If it's possible to compute such a value, then a `ResponsePWCET` instance is returned, otherwise a `ResponseInvalidDistribution` is what the programmer will get (step 5). Note that `ResponseInvalidDistribution` is not used in the API yet, but this may change in the future.

# 4 Summary

The availability of HPC platforms nowadays has led to a plethora of HPC applications in a variety of domains and contexts. Such ubiquity of HPC has made a number of new requirements emerge. Out of those, timing guarantees are particularly important for a number of applications with real-time needs.

This work shows how software randomization and MBPTA analysis can be used reliably to perform extensive and representative execution time test campaigns for HPC applications and predicting high execution times. Our results on a parallel application used for geophysical exploration confirm our claims and show how reliable and tight pWCET bounds can be obtained for HPC applications running on HPC systems.

This deliverable has described the timing analysis tool. The novel contributions on timing analysis in HPC has been submitted (and accepted) for publication [9].This tool will be integrated in the run-time manager in the remaining tasks of this WP. We look forward to the use of this novel approaches for deriving better run time manager policies.

## References

[1] Jaume Abella, Maria Padilla, Joan Del Castillo, and Francisco J. Cazorla. Measurement-based worst-case execution time estimation using the coefficient of variation. *ACM Trans. Des. Autom. Electron. Syst.*, 22(4), June 2017.

[2] G. Agosta, W. Fornaciari, G. Massari, A. Pupykina, F. Reghenzani, and M. Zanella. Managing Heterogeneous Resources in HPC Systems. In *Proc. of PARMA-DITAM '18*, pages 7–12. ACM, 2018.

[3] Emery D. Berger and Benjamin G. Zorn. DieHard: Probabilistic memory safety for unsafe languages. In *In Proceedings of the ACM SIGPLAN 2006 Conference on Programming Language Design and Implementation*, pages 158–168. ACM Press, 2006.

[4] Emery D. Berger, Benjamin G. Zorn, and Kathryn S. McKinley. Composing high-performance memory allocators. pages 114–124, 2001.

[5] Barcelona Supercomputing Center. libta: header-only implementation of mbta-cv. https://chef.heaplab.deib.polimi.it/source/libta.

[6] Charlie Curtsinger and Emery D. Berger. STABILIZER: Statistically sound performance evaluation. *SIGARCH Comput. Archit. News*, 41(1):219–228, March 2013.

[7] J. Flich, G. Agosta, et al. Exploring manycore architectures for next-generation HPC systems through the MANGO approach. *Microprocessors and Microsystems*, 61:154 – 170, 2018.

[8] J. Flich et al. Enabling HPC for QoS-sensitive applications: The MANGO approach. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 702–707, March 2016.

[9] Matteo Fusi, Fabio Mazzocchetti, Albert Farres, Leonidas Kosmidis, Ramon Canal, Francisco J. Cazorla, and Jaume Abella. On the use of probabilistic worst-case execution time estimation for parallel applications in high performance systems. *Mathematics*, 8(3):314, Mar 2020.

[10] L. Kosmidis, C. Curtsinger, E. Quiones, J. Abella, E. Berger, and F. J. Cazorla. Probabilistic timing analysis on conventional cache designs. In *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 603–606, March 2013.

[11] LoopPerfect. neither. https://github.com/LoopPerfect/neither.

[12] Giuseppe Massari, Anna Pupykina, Giovanni Agosta, and William Fornaciari. Predictive resource management for next-generation high-performance computing heterogeneous platforms. In *Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS'19)*, Jul 2019.

[13] A. Pupykina and G. Agosta. Optimizing Memory Management in Deeply Heterogeneous HPC Accelerators. In *2017 46th Int'l Conf on Parallel Processing Workshops (ICPPW)*, pages 291–300, Aug 2017.