Horizon 2020 Program (2014-2020)

A computing toolkit for building efficient autonomous applications leveraging humanistic intelligence (TEACHING)

D2.1: State-of-the-art analysis and preliminary requirement specifications for the computing and communication platform[†]

Contractual Date of Delivery	31/10/2020
Actual Date of Delivery	31/12/2020
Deliverable Security Class	Public
Editor	*Patrizio Dazzi (CNR)*
Contributors	UNIPI: Gabriele Mencagli CNR: Patrizio Dazzi, Emanuele Carlini, Alberto Gotta, Pietro Cassarà TRT: Sylvain Girbal I&M: Lorenzo Giraudi IFAG: Antonio Escobar
Quality Assurance	*Reviewer Claudio Gallicchio (UNIPI)*

[†] *The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871385.*

The TEACHING Consortium

University of Pisa (UNIPI)	Coordinator	Italy
Harokopio University of Athens (HUA)	Principal Contractor	Greece
Consiglio Nazionale delle Ricerche (CNR)	Principal Contractor	Italy
Graz University of Technology (TUG)	Principal Contractor	Austria
AVL List GmbH	Principal Contractor	Austria
Marelli Europe S.p.A.	Principal Contractor	Italy
Ideas & Motion	Principal Contractor	Italy
Thales Research & Technology	Principal Contractor	France
Information Technology for Market Leadership	Principal Contractor	Greece
Infineon Technologies AG	Principal Contractor	Germany

Document Revisions & Quality Assurance

Internal Reviewers

1. *Reviewer Claudio Gallicchio, (UNIPI)*

Revisions

Version	Date	By	Overview
1.0	30/12/2020	Editor	Final
0.3R	29/12/2020	Reviewer	Comments on draft
0.3	21/12/2020	Editor	First draft
0.2	18/12/2020	Contributors	All Contributions provided
0.1	12/11/2020	Editor	ToC

Table of Contents

List of Figures

List of Abbreviations

EC	European Commission
WP	Work Package
AI	Artificial Intelligence
ML	Machine Learning
AIaaS	AI-as-a-Service
CPS	Cyber-Physical System (CPS)
CPSoS	Cyber-Physical Systems of Systems
ADAS	Advanced driver-assistance systems
GPU	Graphics processing unit
FPGA	Field Programmable Gate Array
DSP	Data Stream Processing
IoT	Internet of Things
CUDA	Compute Unified Device Architecture
VANET	Vehicular Ad-hoc NETwork
RAN	Radio Access Network
RTOS	Real-Time Operating System
UC	Use Case
ODD	Operational Design Domain

Executive Summary

This deliverable is aimed at providing an initial report on state-of-the-art analysis, a preliminary requirements elicitation and conceptual design of the Distributed Computing and Communication platform for CPSoS, that we refer as High-Performance Computing and Communication Infrastructure (HPC^2I). That is the infrastructure standing at the basis of TEACHING platform and support the execution of project use cases.

The deliverable focuses on the several technical aspects impacting on the definition of the HPC^2I. It resorts to the scientific literature and project use cases for the identification and analysis of the relevant research and technical challenges. Such challenges eventually contribute to the elicitation of requirements, that are reported in a formal way with project-wide identifiers.

Later in the deliverable is given the definition of a conceptual, high-level system architecture, that is intended to be the basis on which to develop a concrete, full-fledged, architecture that will be presented in deliverable D2.2.

Then, this deliverable summarizes some baseline tools and technologies that are suitable candidates to be used for supporting the project activities.

Finally are drawn our conclusions.

1 Introduction

A key claim of TEACHING is that an effective AI for the CPSoS should exhibit certain key design features. Among them, the fact that machine intelligence is of distributed and pervasive nature, and the AI components can be potentially deployed in every element of the CPSoS, enabling to ***embed intelligence at the edge***, close to where the information is produced by the device or close to where the application consumes the AI predictions. As a consequence, information processing should follow as much as possible principles of locality and compositionality, respecting the inherently distributed nature of the SoS. This allows containing the deluge of noisy, redundant, heterogeneous and fast flowing data produced by the CPSoS elements, with seemingly impacting reductions in communication, storage and energy-consumption costs. Local consumption of information can also be an advantage in scenarios of unreliable connectivity or when data privacy is a key issue.

TEACHING focuses on the tools and mechanisms enabling the setup and the subsequent exploitation of the distributed and heterogeneous computing platform enabling the pervasive processing of data and allow the embedding of intelligence at the edge. Resources include different kinds of computing systems, ranging from traditional to smaller resource/energy-constrained devices, such as IoT, sensors or wearable devices. Specific activities are needed to deal with the application deployment onto the computing platform, the related networks and communications as well as on device programming and data storing. In fact, during the project will be developed solutions for enabling information exchange, dissemination and gathering among different devices, possibly belonging to different cyber-physical subsystems. TEACHING will also deal with application deployment and tuning, both in traditional computing solutions as well as on resource/energy-constrained devices. Advanced solutions and approaches for the efficient management and processing of data streams will be implemented possibly leveraging existing, state-of-the-art solutions.

Overall, all these activities will contribute to the definition of the High-Performance Computing and Communication Infrastructure (HPC^2I), supporting the execution of the TEACHING platform and its use cases.

In such a context, this deliverable – State-of-the-art analysis and preliminary requirement specifications for the computing and communication platform – is aimed at supporting the definition of the TEACHING HPC^2I by conducting an analysis of the solutions existing in the scientific literature (Section 2), to report the results of a preliminary requirement specification (Section 3), to present a conceptual architecture derived from the requirement elicitation process (Section 4) and, finally, to introduce (Section 5) some key technologies that are candidate to give a ground to the design and development activities aimed at the creation of the HPC^2I. Finally, are drawn our conclusions (Section 6).

1.1 Relationship with other deliverables

Deliverables D1.1, D2.1, D3.1, D4.1 and D5.1 (complete deliverable titles are reported later in this section), are intended to serve as a mean of verification for milestone MS1. That is the first project milestone, named *Release of the TEACHING design (requirements, specification and architecture)*.

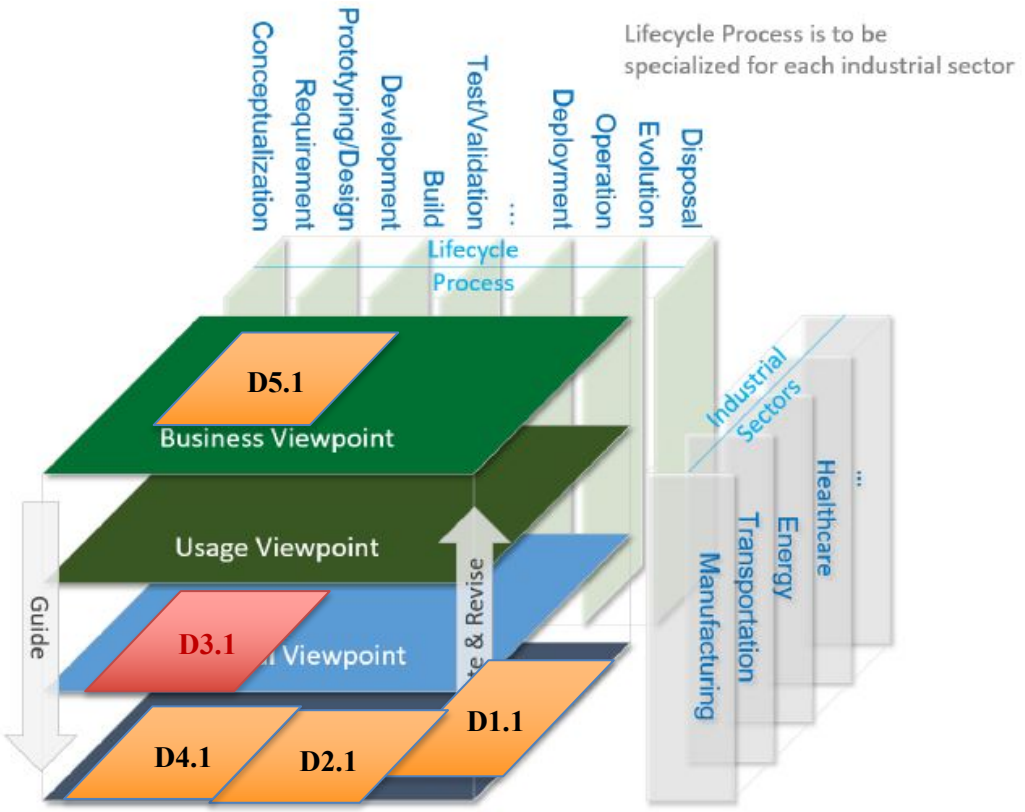

Figure 1 - Depiction of the IIRA Viewpoints from[1] and mapping of focus of TEACHING Deliverables MS1

The mapping of the viewpoints of the technical WPs, as well as the integration intentions of the TEACHING technology bricks in domain use-cases is depicted in Figure 1. Basically, D1.1 (General system perspective), D2.1 (Computing and communication infrastructure) and D4.1 (AI-as-a-Service for CPSoS) delve into technological aspects supporting the development of technologies enabling the definition of TEACHING platform. D3.1 (Engineering Methods and Architecture Patterns of Dependable CPSoS) explores the methodological aspects ensuring that TEACHING-supported CPSoS will be dependable. D5.1 (use case specifications) describes the two TEACHING use cases along with specific technological requirements as well as the challenges that use cases pose to TEACHING and its enabling technologies.

[1] https://iiot-world.com/industrial-iot/connected-industry/iic-industrial-iot-reference-architecture/

MS1 Deliverables:

D1.1 Report on TEACHING related technologies SoA and derived CPSoS requirements

D2.1 State-of-the-art analysis and preliminary requirement specifications for the computing and communication platform

D3.1 Initial Report on Engineering Methods and Architecture Patterns of Dependable CPSoS

D4.1 Initial report on the AIaaS system

D5.1 Initial use case specifications

2 State-of-the-art Analysis

This section presents a state-of-the-art analysis providing an initial scientific ground to the activities conducted by WP2. Being WP2 aimed at the definition and development of the TEACHING computing and communication platform, the analysis conducted in this section frames relevant existing results and put such works in the perspective of the WP goals. The section summarizes contributions regarding the efficient exploitation of hardware resources (both related to computing and communication aspects), as well as smart approaches on the management of such resources.

The section mostly follows the Work package breakdown into tasks. In particular, Section 2.1 is related to Task 2.2 (**High-level Efficient exploitation of multi/many-core CPUs, GPUs and FPGAs**), Section 2.2 presents state-of-the-art analysis for Task 2.3 (**High Performance Processing and Management of Data Streams**), Section 2.3 links relevant contributes in the scientific literature with the activities to be carried out with Task 2.4 (**Sensors, IoT and wearable devices in CPSoS: management, tuning and orchestration**).

The remaining of this Section provides tailored information about relevant contributions in scientific literature that could be investigated in the context of the activities to be conducted in tasks that have not been started yet: task 2.5 - **Efficient and decentralized information exchange within single CPSoS and across different CPSoSs** (Sections 2.4 and 2.5), task 2.6 - **Seamless application deployment in Cloud and Edge resources for the distributed provisioning of computing capacity** (Section 2.6) and task 2.7 - **Silicon-born dependable AI** (Section 2.7).

To ease the reading task, all the references to external work are provided as footnotes.

2.1 Efficient exploitation of heterogeneous computational resources, including multi/many cores CPUs, computing accelerators and FPGAs

The efficient exploitation of multi-core CPUs and co-processors like GPUs and FPGAs is a central goal of the TEACHING platform and of the run-time systems that will be developed during the project. During the activities of T2.2, different dimensions of the problem will be studied starting with the state-of-the-art. In terms of programming models, the general goal is to raise the abstraction level provided to the user, who is a domain expert more than a system-level programmer. For this purpose, two different families of approaches can be considered for developing parallel applications on multicores: *task-based parallel programming models* and *skeleton-based* or *pattern-based programming approaches*.

Task-based parallel programming tools are based on the general idea of implicitly or explicitly expressing a graph of tasks connected by (control- or data-) dependencies. The underlying run-time system is based on a pool of threads often pinned onto the logical cores of the CPUs. Firing tasks are efficiently scheduled on such a pool (usually without having a centralized scheduler but using *lock-free* queues and *work-stealing* techniques to balance the load). Examples of libraries adopting this approach are Intel TBB[2] and OmpSs[3]. Task-based approaches have been enhanced over the years, with additional concepts like work-sharing tasks to exploit fine-grained loop parallelism through the partitioning of the iteration space into chunks processed

[2] James Reinders. 2007. Intel threading building blocks (First. ed.). O'Reilly & Associates, Inc., USA.

[3] Barcelona Supercomputing Center, "OmpSs-2 Programming Model," accessed: 2020-05-24. [Online]. Available: https://pm.bsc.es/ompss-2.

in parallel without the same concept of logical task. Regarding support to loop parallelism, directive-based programming models like OpenMP and the aforementioned OmpSs allow users to augment loop mapping onto CPUs and data sharing rules to designate compilers for automatic parallel code generation. However, these models cannot handle irregular parallel programs efficiently[4]. Heterogeneous parallel programming (e.g., involving support for CPU-based and GPU-based computing) is suitable to be developed with task-based parallel approaches. StarPU[5] is an important project and a solid programming tool for building task graphs and offloading their processing on heterogeneous CPU+GPU systems. To use StarPU, the user is required to provide each task on all the available kinds of resources (CPU, CUDA, OpenCL) with the declarations of memory regions to be used. StarPU's runtime automatically decides where tasks should be executed (thus the version to use). This is often done using some heuristic solutions and all the required memory transfers to make task inputs available to the right resource are done by the run-time system transparently and efficiently, by avoiding additional and unnecessarily copies.

CPU-GPU co-scheduling plays a key role in heterogeneous programming systems. As said before, work stealing is a common strategy to avoid load imbalance in the runtime. Instead of keeping all workers busy most of the time, which may contribute to increase energy consumption low-powered devices, both Intel TBB and OmpSs have developed sleep-based strategies and mixed strategies based on exponential back-off. Other approaches[6] tune hardware frequency scaling to increase energy efficiency, balances[7] the load on distributed memory, deal[8] with data locality during the scheduling of tasks, and optimize[9] task processing for memory-bound applications. How to migrate all such scientific effort for hybrid systems featuring CPUs and GPUs remains a challenge in the research.

Some recent attempts are directed to make task-based programming tools compatible with the offloading of tasks on FPGA devices. OpenCL[10] is a framework providing a uniform low-level programming approach for various kinds of devices (CPU, GPU and FPGA). The device-side language is a dialect of C (OpenCL C). OpenCL programmers can create kernels running on the available devices and can offload tasks to them using command queues. This is a natural way to port a task-based programming environment on top of OpenCL, exploiting its potential to be run on FPGA devices. OpenMP extensions for supporting FPGAs are a hot topic of recent research. The work[11] by Sommer et al. provides and explicit way of offloading portions of an application to FPGA. The work is based on automatic data transfers based on task dependencies.

[4] S.Leeand, J.S.Vetter. Early Evaluation of Directive-Based GPU Programming Models for Productive Exascale Computing. In *IEEE/ACM SC*, pages 1–11, 2012.

[5] StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures CCPE - Concurrency and Computation: Practice and Experience, Special Issue: Euro-Par 2009, 23:187-198, February 2011

[6] Haris Ribic and Yu David Liu. Energy-efficient Work-stealing Language Runtimes. In *ACM ASPLOS*, pag 513–528, 2014

[7] VijayA. Saraswat, Prabhanjan Kambadur, Sreedhar Kodali, David Grove, and Sriram Krishnamoorthy. Lifeline-Based Global Load Balancing. In *PPoPP*, page 201–212, 2011

[8] Warut Suksompong, Charles E. Leiserson, and Tao B. Schardl. On the efficiency of localized work stealing. *Information Processing Letters*, 116(2):100–106, Feb 2016

[9] Shumpei Shiina and KenjiroTaura. Almost Deterministic Work Stealing. In *ACM SC*, 2019

[10] https://www.khronos.org/opencl/

[11] L. Sommer, J. Korinth, and A. Koch, "Openmp device offloading tofpga accelerators," in2017 IEEE 28th International Conference onApplication-specific Systems, Architectures and Processors (ASAP), July2017, pp. 201–205

OmpSs has also been extended in a similar direction[12] focusing on Xilinx FPGAs. At the thread level, so without considering tasks, hthreads[13] have been proposed as a unified programming model to develop application threads running on CPU or on FPGA devices based on compile-time decisions.

Another family of parallel programming framework adopt a philosophy based on the so-called *Structured Parallel Programming* (SPP) methodology. SPP consists of the use of *Parallel Patterns*, which are viable solutions to improve the quality and the efficiency of parallel software development since long time[14]. The general idea is to push the design pattern methodology, which is a de-facto standard practice for Object Oriented Programming, into the Parallel Pattern perspective, with patterns that represent high-level abstract recipes to solve recurrent problems in parallel computing. More concretely, when such patterns are directly provided through high-level programming constructs that can be explicitly instantiated by the developer, combined, and even nested in complex hierarchical structures, we speak about *Algorithmic Skeletons*[15]. Skeleton-based frameworks have been developed to support parallel programmers with the provisioning of standard programming language constructs that model and implement common, parametric, and reusable parallel computation schemes that eventually may be recognized as a practice of implementation of good parallel design patterns. As a matter of fact, combination of parallel design patterns and algorithmic skeletons are exploited in different well known parallel programming frameworks (e.g., Microsoft TPL[16], FastFlow[17], SkePU[18]).

Pattern-based frameworks have notable advantages in terms of time-to-deploy of parallel applications but also guarantee the automatic or semi-automatic applicability of different optimization strategies[19] as well as the development of autonomic management strategies[20] to achieve trade-offs between pure performance and energy consumption and, more generally, QoS requirements and Service Level Agreements with the final application users. This essential characteristic, of special importance for the TEACHING programming eco-system, is based on the formal nature of such patterns/skeletons, which often allows the definition of *cost models*

[12] Filgueras, E. Gil, D. Jim´enez-Gonz´alez, C.´Alvarez, X. Mar-torell, J. Langer, J. Noguera, and K. A. Vissers, "Ompss@zynq all-programmable soc ecosystem," inThe 2014 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA '14, Monterey,CA, USA - February 26 - 28, 2014, 2014, pp. 137–146

[13] D. Andrews, R. Sass, E. Anderson, J. Agron, W. Peck, J. Stevens,F. Baijot, and E. Komp, "Achieving programming model abstractions forreconfigurable computing,"Very Large Scale Integration (VLSI) Systems,IEEE Transactions on, vol. 16, no. 1, pp. 34–44, 2008

[14] T. Mattson, B. Sanders, and B. Massingill. Patterns for Parallel Programming. Addison-Wesley Professional, 2004

[15] M. Cole. Bringing skeletons out of the closet: A pragmatic manifesto for skeletal parallel programming. Parallel Comput., 30(3):389–406, Mar. 2004

[16] Microsoft. Task Parallel Library, 2017. https://msdn.microsoft.com/en-us/library/ dd460717(v=vs.110).aspx

[17] M. Aldinucci, M. Danelutto, P. Kilpatrick, and M. Torquati, Fast- Flow: High-Level and Efficient Streaming on Multicore. John Wiley & Sons, Ltd, 2017, ch. 13, pp. 261–280. [Online]. Available: https: //onlinelibrary.wiley.com/doi/abs/10.1002/9781119332015.ch13

[18] Öhberg, T., Ernstsson, A. & Kessler, C. Hybrid CPU–GPU execution support in the skeleton programming framework SkePU. J Supercomput 76, 5038–5056 (2020). https://doi.org/10.1007/s11227-019-02824-7

[19] K. Emoto and K. Matsuzaki. An automatic fusion mechanism for variable-length list skeletons in sketo. International Journal of Parallel Programming, 42(4):546–563, 2014

[20] G. Mencagli and M. Vanneschi. Towards a Systematic Approach to the Dynamic Adaptation of Structured Parallel Computations using Model Predictive Control. Cluster Computing, 2014, Springer. ISSN: 1386-7857, DOI: 10.1007/s10586-014-0346-3

of their performance (throughput and latency), energy consumption[21] and are suitable for being implemented on different kinds of architectures (like GPUs and FPGAs) by hiding this choice to the final user through clear and homogeneous APIs for instantiating and using the skeletons provided by the framework.

2.2 High-performance processing and management of data streams

CPSs pose urgent challenges from the programmability standpoint, where CPS-enabled applications should provide at least two major characteristics: *i)* the capability to meet (near) real-time requirements while processing *data streams* like the ones generated by the plethora of sensors available in cyber-physical environments; *ii)* being able to exploit heterogeneous hardware like the one available in embedded scenarios, i.e. low-power multi-core CPUs and accelerators like embedded GPUs and FPGAs. These two features shall be provided with sufficiently high-level programming abstractions to develop software with reduced time-to-solution.

The programming paradigm adopted in the literature for processing streams is called *Data Stream Processing* , which had its origin in the Database community with first-generation Data Stream Processing Systems[22,23] based on extensions of relational algebra languages to develop applications able to continuously compute streaming analytics from unbounded input sequences. Over the last years, to support traditional distributed environments like clusters and a wider set of streaming applications dealing with both structured and non-structured data, second-generation DSPSs like Apache Storm, Flink, Spark Streaming and many others have been publicly released to the community. However, despite their pervasive diffusion in real-world scenarios (e.g., social media analysis, networking, sensor networks) they are still inadequate to represent programming frameworks suitable for the IoT and for CPSs too. The main reason is twofold. From one side, their run-time systems do not scale on single shared-memory multicore-enabled nodes[24], due to the high overhead induced by their internal components for data (de-)serialization, resource scheduling and inter-process communication. Furthermore, they do not support any kind of accelerators, and their programming model, based on the development of *data-flow graphs* of operators, does not expose features to keep track of data locality, resource constraints and privacy that are key factors in the IoT.

Over the years, several research attempts have been conducted with the goal of making the DSP paradigm and its enabling frameworks more suitable for the IoT and for CPSs. Some works have focused on improving the scalability of the run-time systems on single nodes, by taking advantage of the shared memory to introduce optimizations during the processing. Some

[21] D. De Sensi, M. Torquati, and M. Danelutto. A reconfiguration algorithm for power-aware parallel applications. ACM Trans. Archit. Code Optim., 13(4):43:1–43:25, Dec. 2016

[22] William Thies, Michal Karczmarek, and Saman P. Amarasinghe. 2002. StreamIt: A Language for Streaming Applications. In Proceedings of the 11th International Conference on Compiler Construction (CC '02). Springer-Verlag, Berlin, Heidelberg, 179–196

[23] Daniel J. Abadi, Don Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. 2003. Aurora: a new model and architecture for data stream management. The VLDB Journal 12, 2 (August 2003), 120–139. DOI:https://doi.org/10.1007/s00778-003-0095-z

[24] S. Zhang, B. He, D. Dahlmeier, A. C. Zhou and T. Heinze, "Revisiting the Design of Data Stream Processing Systems on Multi-Core Processors," 2017 IEEE 33rd International Conference on Data Engineering (ICDE), San Diego, CA, 2017, pp. 659-670, doi: 10.1109/ICDE.2017.119

prototypes following this direction are StreamBox[25], LightSaber[26] and BriskStream[27]. The first two trade off scalability with latency by adopting an approach called *morsel-driven parallelism* where inputs are first buffered into batches of continuous inputs, and the processing of a pipeline of operators is dynamically scheduled on a pool of threads in a load balanced manner. BriskStream instead adopts the *continuous streaming model* of traditional DSPSs, more optimized for low-latency processing, and improves the scalability through the buffering of few inputs that are processed as a whole by the user functions within the operators to reduce the instruction footprint and the processor front-end stalls. However, although one step ahead towards a better lightweight run-time system, these DSPSs do not support accelerators nor do they expose programming abstractions for dealing with the IoT constraints explicitly.

DSPs more optimized for being executed on embedded devices are still at a preliminary stage of research and development. EdgeWise[28] is a recent work extending the popular Apache Storm system by keeping its high-level API while completely re-designing its run-time system to be more suitable for embedded resources. The effort is to remove the dedicated threading model of Storm, which might impair performance on limited-resource architectures due to the fair scheduling performed by the traditional OS. They propose a model where operators are logical executors dynamically scheduled onto a fixed-size pool of threads according to application-dependent scheduling policies.

Despite this progress, the research to target in a more effective way accelerators are still incomplete. According to some preliminary works[29], migrating some real-time streaming tasks to GPUs is challenging due to the need of batch processing to enforce data parallelism on devices as well as the problem of thread divergence that may cause unpredictable latency spikes. To control this, they propose a rigid scheduling of GPU kernels in periods of fixed length to provide guarantees on the latency. SABER[30] is another system for streaming on GPUs. It is based on morsel-driven parallelism, which is inadequate to keep low latency in case of live streaming tasks (where we cannot assume that data are all available in memory like in offline streaming analytics). Furthermore, they support only applications written with streaming dialects of SQL, so not suitable to cover the complexity of applications for CPSs which, instead, need APIs to enable the development of AI/ML tasks within the data-flow programming style commonly adopted in DSP.

---

[25] Hongyu Miao, Heejin Park, Myeongjae Jeon, Gennady Pekhimenko, Kathryn S. McKinley, and Felix Xiaozhu Lin. 2017. StreamBox: modern stream processing on a multicore machine. In Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '17). USENIX Association, USA, 617–629

[26] Georgios Theodorakis, Alexandros Koliousis, Peter Pietzuch, and Holger Pirk. 2020. LightSaber: Efficient Window Aggregation on Multi-core Processors. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20). Association for Computing Machinery, New York, NY, USA, 2505–2521. DOI:https://doi.org/10.1145/3318464.3389753

[27] Shuhao Zhang, Jiong He, Amelie Chi Zhou, and Bingsheng He. 2019. BriskStream: Scaling Data Stream Processing on Shared-Memory Multicore Architectures. In Proceedings of the 2019 International Conference on Management of Data (SIGMOD '19). Association for Computing Machinery, New York, NY, USA, 705–722. DOI:https://doi.org/10.1145/3299869.3300067

[28] Xinwei Fu, Talha Ghaffar, James C. Davis, and Dongyoon Lee. 2019. Edgewise: a better stream processing engine for the edge. In Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '19). USENIX Association, USA, 929–945

[29] Kai Zhang, Jiayu Hu, and Bei Hua. 2015. A holistic approach to build real-time stream processing system with GPU. J. Parallel Distrib. Comput. 83, C (September 2015), 44–57. DOI:https://doi.org/10.1016/j.jpdc.2015.05.002

[30] Alexandros Koliousis, Matthias Weidlich, Raul Castro Fernandez, Alexander L. Wolf, Paolo Costa, and Peter Pietzuch. 2016. SABER: Window-Based Hybrid Stream Processing for Heterogeneous Architectures. In Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16). Association for Computing Machinery, New York, NY, USA, 555–569

Other recent attempts have targeted traditional clusters of large servers equipped with GPU PCI-express boards. One example is Gstream[31]. Its programming model to express operators provides a simple interface where each operator is defined by extending specific base classes and overriding three methods: one for the pre-processing phase, the second for the kernel phase, the last for the post-processing step. The kernel phase is the one offloaded on the device. Despite this clear separation of concerns, the kernel function requires a fine control of the batch size to use, and the manual popping and pushing of data from sources and to producers. All these activities are low level and error prone from the viewpoint of a standard user. Furthermore, no abstraction is given to write the GPU kernel, because users need to instantiate directly CUDA code within the kernel function. G-Storm[32] proposes an extension of Apache Storm to enable GPU processing. The idea is to introduce a new operator enabled to offload the processing of an input batch to GPU. The good point with respect to Gstream is the higher-level interface: the user provides only the fine-grained function executed over each input, while the batching and the preparation of the CUDA kernel (through the Java library JCUDA) is done automatically by the run-time system. However, the approach has some specific flaws. In particular, the input buffering, processing, and the copy back of results is done without considering that the applications can require more than one GPU-enabled operator, and data communication and output routing between GPU-enabled operators should be done in a different way to avoid useless copies and to keep the data on the device's memory as much as possible to improve performance.

For what regards DSPSs targeting FPGAs, the research still lacks deep studies on this topic. Despite some attempts to implement on FPGAs specific streaming operators (e.g., windowed aggregates and joins), no past paper has tried to provide a complete system to program entire data-flow graphs with specific operators offloaded on FPGAs. The main paper which has partially tried to face this problem is Glacier[33], which provides some built-in operators (typically the ones of relational algebra) on FPGAs but lacks extendibility to cover more general-purpose tasks.

There are several open problems that represent challenges to enforce better programmability of DSP applications on embedded resources. The first challenge is how to integrate data locality, privacy and resource constraints while developing the data-flow graph of the application. Such kinds of meta information should be provided to drive the preliminary steps of the application deployment onto adequate resources. Second, the support for accelerators is still partial. To effectively use GPUs and FPGAs, the programming model that we envision clearly goes beyond the state-of-the-art. It will provide a high-level interface for implementing *any* streaming task, by requiring to the programmer to express the business logic code of the operators through lambda functions compliant with a set of acceptable signatures, while the kernel code to be offloaded on the device is in charge of the run-time system, which will also provide optimizations to enhance the co-existence of CPU-oriented and Accelerators-oriented operators through efficient data exchange mechanisms. All these steps are towards a new programming model for CPSs which will represent one of the key-enabling technologies proposed in the TEACHING project.

---

[31] Y. Zhang and F. Mueller, "GStream: A General-Purpose Data Streaming Framework on GPU Clusters," 2011 International Conference on Parallel Processing, Taipei City, 2011, pp. 245-254, doi: 10.1109/ICPP.2011.22

[32] Z. Chen, J. Xu, J. Tang, K. Kwiat and C. Kamhoua, "G-Storm: GPU-enabled high-throughput online data processing in Storm," 2015 IEEE International Conference on Big Data (Big Data), Santa Clara, CA, 2015, pp. 307-312, doi: 10.1109/BigData.2015.7363769

[33] Rene Mueller, Jens Teubner, and Gustavo Alonso. 2009. Streams on wires: a query compiler for FPGAs. Proc. VLDB Endow. 2, 1 (August 2009), 229–240. DOI:https://do.org/10.14778/1687627.1687654

2.3 Efficient data collection from Sensors, IoT and wearable devices in CPSoS

Advances in engineering enabled the development of the internet-of-things (IoT) to understand, interface, interact and engineer the physical world (systems). In more words, interconnecting billions of smart devices allows us to monitor and extract data about both the physical and the cyber communities. However, deploying a multitude of wireless sensors and agents on different application domains (e.g., environmental, healthcare, smart interconnected vehicles and trucks, smart buildings) leads not only to the hard task for communicating massive amounts of heterogeneous data, but also requires developing efficient strategies for the system modeling through the sensed data while overcoming the energy wall[34].

The IoT paradigm is referred by Bogdan et al. to a "living" interconnection of *uniquely identifiable smart embedded systems* or *things* (e.g., sensors, actuators, mobile devices) with deeply embedded cyber capabilities for sensing, processing and communicating the accumulated large amounts of heterogeneous data about the world to computational nodes for real-time analysis, interpretation and determination of closed-loop control strategies. One fundamental promising advantage, which the IoT brings, is the capability to autonomously and trustworthy process high volume of data for deciphering not only the structural and dynamical patterns of complex systems, but also for providing informed and robust control decisions.

In the smart city domain that is akin to the one of TEACHING use cases (for a detailed description refer to deliverable D5.1), the IoTs monitor the cross-dependencies between autonomous or semi-autonomous transportation and human mobility comfort, constructs dynamical coupled equations for sensitivity analysis and determines the best control decisions (e.g., controls the human mobility and adapts autonomous driving style to the best comfort of a customer).

To achieve such goals, the IoT must possess built-in intelligence / analytics capable of:

(1) extracting prescriptive information in terms of ***patterns and statistical characteristics*** of the observed processes that lead to compact system models (the term compact refers to models with minimum number of parameters given the complexity of the investigated complex system);

(2) provide predictive power by constructing ***causal compact system models*** that can explain what might happen to a set of variables or *things* of the whole if specific changes / perturbations are applied to other parts of the system.

The synergetic coupling of physical and cyber worlds raises a few grand challenges[35]:

- What is the appropriate compact model that captures the characteristics of cyber and physical processes and facilitates the analysis, design and optimization?
- What compact yet accurate models should be developed to enable the design of large-scale autonomous IoT systems-of-systems?

[34] P. Bogdan, M. Pajic, P. P. Pande and V. Raghunathan, "Making the Internet-of-Things a reality: From smart models, sensing and actuation to energy-efficient architectures," *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Pittsburgh, PA, 2016, pp. 1-10.

[35] Y. Xue, S. Rodriguez and P. Bogdan, "A spatiotemporal fractal model for a CPS approach to brain-machine-body interfaces", *Design Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 642-647, 2016

TEACHING projects captures and embeds several fundamental concepts of CPSoS like sensing, networking, and computing. This process enables description of a physical phenomenon like the comfort status of a driver or the life prediction of a complex system and leverage the ubiquitous communication paradigm to achieve a global model.

2.4 Decentralized strategies for data dissemination and organization in CPSoS

The aspects of decentralized communication and data management in TEACHING mostly apply to the vehicle use case. In fact, with the continuous advancements and developments of AI and communication technologies, an increased amount of smarter transportation vehicles is circulating in the road. These vehicles are equipped with a myriad of advanced sensors, computing, and communication technologies. Such a complex system that integrates computing, communication and control technologies can be seen a cyber-physical system with a closed feedback loop between the cyber process and the physical process[36].

Particularly, by integrating wireless communications interfaces on board, a running vehicle can exchange information with its surrounding, to enable a safer and more comfortable experience. In practice, vehicles can dynamically form a wireless network on the road, the so called Vehicular Ad-hoc NETwork (VANET). By employing different kinds of communication, such as vehicle-to-vehicle and vehicle-to-infrastructure, VANETs can support a wide variety of applications[37]. For example, in the context of TEACHING the VANET can be used to disseminate relevant data about the model to choose the proper driving model in accordance with the surrounding environment.

In terms of data dissemination protocol, a major challenge is related to the dynamcity of the vehicles and the consequent change of topology. To tackle this problem, several adaptive approaches have been proposed. For example, DV-CAST[38] uses periodic beacon messages to create and maintain a local dissemination topology. The protocol DRIVE[39] aims to propagate data in an enclosed area of interest, without maintain a neighbor list, in order to reduce the delay of the propagation. RTAD[40] proposes an adaptive meta approach, in which each vehicle is able to choose the most appropriate dissemination protocol in order to adapt to the current situation and surrounding environment. SRD[41] proposed a push-based dissemination protocol that needs only local topology information gathered with periodic beacon messages.

[36] Jia, Dongyao, et al. "A survey on platoon-based vehicular cyber-physical systems." *IEEE communications surveys & tutorials* 18.1 (2015): 263-284.

[37] Ghebleh, Reza. "A comparative classification of information dissemination approaches in vehicular ad hoc networks from distinctive viewpoints: A survey." Computer Networks 131 (2018): 15-37.

[38] Tonguz, Ozan K., Nawaporn Wisitpongphan, and Fan Bai. "DV-CAST: A distributed vehicular broadcast protocol for vehicular ad hoc networks." *IEEE Wireless Communications* 17.2 (2010): 47-57

[39] Villas, Leandro Aparecido, et al. "Drive: An efficient and robust data dissemination protocol for highway and urban vehicular ad hoc networks." *Computer Networks* 75 (2014): 381-394.

[40] Sanguesa, Julio A., et al. "RTAD: A real-time adaptive dissemination system for VANETs." *Computer Communications* 60 (2015): 53-70.

[41] Wisitpongphan, Nawaporn, et al. "Broadcast storm mitigation techniques in vehicular ad hoc networks." *IEEE Wireless Communications* 14.6 (2007): 84-94.

Finally, P2PCC[42] is an hybrid peer-to-peer protocol in which the information among the vehicles is exchanged according to a probabilistic model. If the information is not available, vehicles will communicate with the centralized server.

2.5 Efficient communication protocols in multi-homed scenarios targeting CPSoS

Most of the devices involved in a CPS and then the CPSoS are supposed to be small sized objects equipped with a limited amount of power resources. Although nowadays the concept of CPS has extended to more complex systems, the ICT field provides many protocols and paradigms to enable communications among the devices of a CPS or CPSoS, as well as the communication among the devices and service communication entities. Many of the applications for CPSs involve entities that need to communicate with each other on the move. A first example of standard definition for mobility scenario is provided by 3GPP, which has defined the Narrow-Band IoT (NB-IoT)[43] for low bandwidth and low power consuming devices. But due to the increasing demand for redundancy and some also for a greater throughput paradigm such as the Internet of Multimedia Things (IoMT)[44] has been defined to fulfil IP broadband requirements coming from Mission Critical Communications applications[45] that are mainly based on the IoT communication. In the scenario of CPSs where the involved devices are mobile, IoT multihoming and multiple-RAN (Radio Access Network) connectivity management, including automatic air interface selection and optimal weighted load-balancing between interfaces, are challenges for the reliability of future networks. Without reliable mechanisms for multihoming and load-balancing of multiple interfaces, all the Mission Critical Communication could not function. Mobility, redundancy, and bandwidth requirements may be transferred, in an CPS ecosystem, from the physical devices to the network elements, but wherever it resides, the redundancy and load-balancing function of mobile and non-mobile devices is dependent on the mechanisms for RAN multihoming. For the mobile communications Locator/ID Separation Protocol LISP[46] has recognized as a serious candidate for 5G standardization and intensively backed-up by a IETF (Internet Engineering Task Force) Work Group, LISP offers mobility, multihoming, and load-balancing that have the advantage to be applied with no changes in the Internet architecture (directly at the mobile IoT element)[47] and furthermore can be perfectly matched with SDN (Software Defined Networks) that represent another highlight of 5G architectures. A significant support for mobile communication for CPSs is provided by MEC-based (Mobile Edge Computing)[48] infrastructure and Fog Computing for IoT, as the processing can take place distributed, where the objects are,

[42] Kumar, Neeraj, and Jong-Hyouk Lee. "Peer-to-peer cooperative caching for data dissemination in urban vehicular communications." *IEEE Systems Journal* 8.4 (2013): 1136-1144.

[43] NB-IOT-feature list, *Release 13 (LTE Advanced Pro), 3GPP*, http://www.3gpp.org/images/PDF/R13 IOT rev3.pdf

[44] S. A. Alvi, B. Afzal, G. A. Shah, L. Atzori, and W. Mahmood, "Internet of multimedia things: vision and challenges," *Ad Hoc Networks*, vol. 33, pp. 87–111, 2015.

[45] M. J. Peltola, A. University, and H. Hammainen, "Economic feasibility of mobile broadband network for public safety and security," in *Proceedings of the 11th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, WiMob 2015*, pp. 67–74, are, October 2015.

[46] D.Farinacci,V.Fuller,D.Meyer,andD.Lewis,"The Locator/ID Separation Protocol (LISP)," RFC Editor RFC6830, 2013.

[47] W. Ramirez, X. Masip-Bruin, M. Yannuzzi, R. Serral-Gracia, A. Martinez, and M. S. Siddiqui, "A survey and taxonomy of ID/Locator Split Architectures," *Computer Networks*, vol. 60, pp. 13–33, 2014.

[48] "Mobile Edge Computing (MEC); Service Scenarios," in *ETSI Group Specification*, 2015, http://www.etsi.org/deliver/etsi gs/ MEC-IEG/001 099/004/01.01.01 60/gs MEC-IEG004v010101p .pdf.

thus reducing communication delays to a central processing Cloud. In this scenario the mobility could refer to the mobility of the components of a CPS (devices, sensors, IoT gateways etc.) following the global understanding of host/ user mobility; or also to Virtual Machine Mobility (VMM,) from one Edge Cloud to the other, concept also known as "Follow-Me Cloud," a method for interworking of federated clouds and distributed mobile networks[49]. The LISP protocol is a recognized solution for both the above-mentioned mobility cases, thus being differentiated from other mobility solutions. Furthermore, it is a very good option for multihoming solutions. Open-source LISP implementation are Open Overlay Router (OOR)[50] and IBM Watson[51].

LISP[52] is a host mobility protocol and a Virtual Machine Mobility protocol. In case of CPS, it can be used as host mobility, and the endpoint identifier-based addressing can be adopted as method for simplification and abstraction of network infrastructure and the RAN used as point of attachment in the network. LISP can be used for the migration of the localized processing function, for example, the migration of the IoT gateway functions from one RAN Cloud to another. Vehicle-to-vehicle communication and self-driving cars imply a lot of processing that can only be performed at the edge of the network, in the Edge Clouds, in order to minimize the delays. Migration of virtual machines during their run using LISP is a method for the processing power to follow the intelligent devices in IoT. For VMM (Virtual Machine Mobility), the processing power and applications can follow the mobility of the "thing". The benefits of using LISP for linking the Virtual Machine Mobility to the user mobility were already demonstrated by some experimental work[53]. Mobility management RFCs proposals also refer to LISP as one candidate solution. "Mobility Management for 5G Network Architectures Using Identifier-Locator Addressing"[54] specification describes the Mobility Management Architecture for 5G Networks Using Identifier-Locator Addressing in IPv6 for virtualized mobile telecommunication networks. Identifier-Locator addressing differentiates between location and identity of a network node, and it is considered the key method for 5G mobility[55]. In case of CPSs supporting critical infrastructure elements, failover mechanisms are essential in a network that is connected to multiple providers, while still being reachable via the same address (most probably an IPv6 address, but endpoint identifier addressing is very permissive). LISP implementations allow the setting of prioritization methods (weighting) for the parallel connected RANs networks, though providing load-balancing for enhance throughput, not just back-up connectivity.

[49] T. Taleb and A. Ksentini, "Follow Me cloud: interworking federated clouds and distributed mobile networks," *IEEE Network*, vol. 27, no. 5, pp. 12–19, 2013.

[50] "Open Overlay Router (OOR)," http://www.openoverlayrouter .org.

[51] "IBM Watson," https://www.ibm.com/watson

[52] A. Rodrıguez Natal, L. Jakab, M. Portoles et al., "LISP-MN: mobile networking through LISP," Wireless Personal Communications, vol. 70, no. 1, pp. 253–266, 2013.

[53] S. Secci, P. Raad, and P. Gallard, "Linking virtual machine mobility to user mobility," *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 927–940, 2016.

[54] J. Mueller and T. Herbert, *Mobility Management for 5G Net- work Architectures Using Identifier-locator Addressing*, 2016, https://tools.ietf.org/html/draft-mueller-ila-mobility-01.

[55] P. Rost, A. Banchs, I. Berberana et al., "Mobile network architecture evolution toward 5G," *IEEE Communications Magazine*, vol. 54, no. 5, pp. 84–91, 2016.

2.6 Data and software orchestration in a vertical Cloud-Edge continuum

Edge computing is an approach that has been proposed to let the computation be performed closer to the source of the workload generation. Basically, edge computing outdoes the traditional, cloud concept, by pushing applications, data and computing power away from centralized points to locations closer to the end user, to interact with the physical world[56]. Edge computing can be achieved either mostly on the edge or in strong interaction with a cloud[57]. This last scenario creates a computational continuum spanning from clouds to edge resources and allowing the joint exploitation of resources belonging the two platforms. In this setup, Edge computing can be fruitfully exploited to support AI-based solutions, as it has been demonstrated, by different evidences published in the scientific literature (see e.g., the work from Li He et al[58] and references therein). In spite of the very recent conception of the edge computing concept, some technological solutions addressing the needs of orchestrating edge-oriented applications have been proposed. Among them KubeEdge[59] is getting momentum. It is aimed at extending native containerized application orchestration capabilities to hosts at Edge.

An alternative very relevant approach is represented by the Osmotic computing[60]. Osmotic computing is a quite recently proposed paradigm to support the efficient execution of IoT services and applications at the edge. This paradigm is founded on the need for a holistic distributed system abstraction enabling the deployment of lightweight microservices on resource-constrained IoT platforms at the edge, coupled with more complex microservices running on large-scale datacenters. This paradigm is driven by the significant increase in resource capacity/capability at the network edge, along with support for data transfer protocols that enable such resources to interact more seamlessly with datacenter-based services.

2.7 Resilient, Fail-safe and energy-efficient, silicon born AI

Autonomous vehicles are an emerging research field where further research needs to be done. As a result, numerous authors are developing new AI models to improve the performance of these vehicles. However, the dataset creation for this application is not trivial due to the difficulty to find specific scenarios that fit with the desired ones while ensuring the safety and security as well as the privacy of the pedestrian, vehicles, etc. in the area. During tests of sensor stability e.g., adversarial attacks, unforeseen reactions cannot be excluded. Therefore, there is a need to find another approach to generate these datasets in a fast and efficient way. At the same time, these new algorithms and models need to be tested and evaluated, which doing so in a real vehicle contain a risk. Simulations will always only be as good as the model they represent, why real-world data is essential. A solution to the earlier mentioned challenges is the setup of a miniaturized vehicle that is equipped with state-of-the-art sensors. This way

[56] Satyanarayanan, Mahadev. "The emergence of edge computing." Computer 50.1 (2017): 30-39.

[57] Shi, Weisong, et al. "Edge computing: Vision and challenges." IEEE Internet of Things Journal 3.5 (2016): 637-646.

[58] Li, He, Kaoru Ota, and Mianxiong Dong. "Learning IoT in edge: deep learning for the internet of things with edge computing." IEEE Network 32.1 (2018): 96-101.

[59] https://kubeedge.io/en/

[60] M. Villari, M. Fazio, S. Dustdar, O. Rana and R. Ranjan, "Osmotic Computing: A New Paradigm for Edge/Cloud Integration," in IEEE Cloud Computing, vol. 3, no. 6, pp. 76-83, Nov.-Dec. 2016.

developments of algorithms and models can be speeded up. As regulations for safety and security as well as privacy restrictions can be easily overcome while having the benefits of real-world sensor data. Especially in the field of autonomous mobile cars there already is some research going on. With the open-source project named DonkeyCar[61] being the most well-known example. The use of a reduced size vehicle in order to develop and test new algorithms has been implemented by multiple authors due to multiple reasons such as safety, security or cost efficiency. An example of this is the project of Qi Z. et al.[62] where a DonkeyCar with a camera sensor was used to successfully implement algorithms based on reinforcement learning. Similarly, T. Do et al.[63] proposed a monocular vision-based autonomous vehicle prototype using Deep Convolutional Neural Networks. This system was based on a small-scale vehicle following the approach of Qi Z. et al achieving a final accuracy of 89.04% on the steering and speed decision. However, these systems were based on a single sensor. This can lead to problems due to possible noise in the data from the camera sensor as well as vulnerabilities to adversarial attacks[64]. At the same time, camera sensor data may not be reliable when driving in dark scenarios or under bad weather conditions[65]. Therefore, we propose to integrate multiple sensors leading to a sensor fusion approach where the data from different sensors will be used to complement each other.

[61] https://www.donkeycar.com/

[62] Zhang Qi and Du Tao. Self-driving scale car trained by deep reinforcement learning. ArXiv, abs/1909.03467, 2019.

[63] Q. Dang T. Do, M. Duong and M. Le. Real-time self-driving car navigation using deep neural network. 2018 4th International Conference on Green Technology and Sustainable Development (GTSD), Ho Chi Minh City, 2018, abs/1909.03467:7–12, 2018.

[64] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust Physical-World Attacks on Deep Learning Models. arXiv:1707.08945 [cs], April 2018. arXiv:1707.08945.

[65] F. Nobis, M. Geisslinger, M. Weber, J. Betz, and M. Lienkamp. A deep learning-based radar and camera sensor fusion architecture for object detection. 2019 Sensor Data Fusion: Trends, Solutions, Applications (SDF), Bonn, Germany, 2019, pages 1–7, 2019.

3 Requirement Analysis

Requirements analysis in TEACHING has been conducted in way that ensure a project-level coordination and minimize the risk of inconsistencies between different WPs. TEACHING consortium developed a centralized repository where all the requirements of the project have been collected and indexed (by assigning to each of them a unique ID). As a consequence, this section reports all those requirements that have an impact on WP2 but some requirements could be also reported in different deliverables, with the same ID, when an impact is envisioned in other WPs. As matter of fact, the design of the architecture of the *high-performance computing and communication infrastructure (HPC²I)* will be the output of the requirement analysis, and will be presented in the next deliverables of WP2. However, in Section 4 of this deliverable we anticipate some key concepts around which the overall architecture of *HPC²I* will be realized. As a consequence, the remaining of this section is organized highlighting the requirements posed to the platform by taking into account such concepts. Basically, in *HPC²I* can be envisioned two main subsystems: A Mission-critical dependable system and a Human-empowered Intelligent subsystem. The former is aimed at designing the CPSoS subsystems that are devoted to support mission-critical applications (e.g., Advanced driver-assistance systems, ADAS in short) running on on-premise specialized hardware and software co-located the controlled system. The latter is devoted to the design of the CPSoS subsystems dealing with the actual collection, analysis and exploitation of the feedback of humans to make, overall, the CPSs smarter. This last set of subsystems run on general-purpose hardware organized in a computational continuum spanning from remote clouds to edge devices co-located with on-premise resources.

3.1 Requirements on Mission-critical Dependable subsystem

3.1.1 Safety-critical in avionics

Spatial Isolation (ID_2): Spatial isolation should be ensured in the TEACHING heterogeneous setup both at software and hardware level. It includes: between the cores (usually thanks to an MMU), between the regular CPU and the accelerator (usually with explicit communication channels only), and inside the accelerator itself if it maps several machine learning algorithms.

Impacts on WP: 1, 2, 3, 4

Priority: High

Domain: Avionics

Temporal Isolation (ID_3): Temporal isolation should be ensured in the TEACHING heterogeneous setup both at software and hardware level. It includes: between the cores (usually thanks to periodic scheduling), between the regular CPU and the accelerator (usually with explicit communication channels only), and inside the accelerator itself if it maps several machine learning algorithms.

Impacts on WP: 1, 2, 3, 4

Priority: High

Domain: Avionics

3.1.2 Real-time requirements in avionics

Tight standalone WCET upper-bounds (ID_5): The worst-case execution time of the application tasks should allow to be tightly bounded with respects to execution and communication time if the task is running standalone. It includes regular tasks running on the generic core, and AI algorithms running on the accelerator

Impacts on WP: 1, 2, 3, 4

Priority: High

Domain: Avionics

Reasonable concurrent WCET upper-bounds (ID_6): While running in a multi-core concurrent execution context, the worst-case execution time of the application tasks should provide a reasonable upper bound, with safety margins not offsetting the benefits of concurrent / accelerated execution, and actually controlling over-provisioning.

Impacts on WP: 1, 2, 3, 4

Priority: High

Domain: Avionics

Monitoring features & interference channels identification (ID_7): The regular software, the AI software, the regular cores and the accelerator should all provide some monitoring features allowing us to monitor the behavior of the software on the hardware with regards to timing interference. The certification standards enforce us to identify all possible timing interference channels.

Impacts on WP: 1, 2, 3, 4

Priority: Critical

Domain: Avionics

Synchronous global system clock (ID_8): Multi-core real-time systems requires a synchronous clock for all the cores in the system to fulfill with real-time scheduling rules limiting the timing interference level. time offset between local clocks is manageable, but clock drift is not allowed.

Impacts on WP: 1, 2, 3, 4

Priority: Critical

Domain: Avionics

3.1.3 Software selection and operations for avionics

Task & Communication scheduling (ID_13): Eliminating / Bounding timing interference will require to apply rules on task and communication scheduling.

Impacts on WP: 2, 5

Priority: High

Domain: Avionics

RTOS (ID_14): CPS applications from the avionic UC being real-time and safety-critical, we require to run them with an RTOS that can guarantee scheduling of periodic tasks. Preventing preemption is not enough. The real-time operating system is responsible from providing the required task activation patterns for both synchronous periodic tasks, and asynchronous aperiodic tasks. The RTOS is also responsible for detecting deadline misses associated with these tasks.

Impacts on WP: 2, 5

Priority: High

Domain: Avionics

Monitor the GPP from the IA accelerator (ID_16): Monitoring information of the software running on the GPP hardware has to be send / made accessible from the IA accelerator without significantly impacting timing interference.

Impacts on WP: 2, 5

Priority: Low

Domain: Avionics

3.1.4 Sensor and localization data management for avionics

Periodic sensors real-time requirements (ID_17): Every sensor outputs data are generated every 200ms (5Hz periodic task), with a deadline equals to the period. Those data can possibly partially or fully be marked as invalid, depending on the flight conditions.

Impacts on WP: 2, 5

Priority: High

Domain: Avionics

Aperiodic sensors real-time requirements (ID_18): New sensors parameters asynchronously set by the pilot should be taken into account in the next sensor operational cycle, implying a deadline of 200ms.

Impacts on WP: 2, 5

Priority: High

Domain: Avionics

Periodic localization real-time requirements (ID_19): LOCC1 should operate as a periodic task at a cycle frequency of 200ms (5Hz) to match with the sensors' outputs. The low frequency BCP needs to be computed by LOCC2 every 5000ms. The magnetic deviation has to be updated by LOCC3 every 1600ms. Finally, the performance is computed every 1000ms by LOCC4. All these tasks have a deadline matching their period.

Impacts on WP: 2, 5

Priority: High

Domain: Avionics

Aperiodic localization real-time requirements (ID_20): New localization parameters asynchronously set by the pilot should be taken into account in the next operational cycle of the related task, implying a 200ms deadline for BCP related, 5s for magnetic variation related, and 1s for performance related.

Impacts on WP: 2, 5

Priority: High

Domain: Avionics

3.1.5 Flight plan and trajectories management in planes

Aperiodic flight plan real-time requirements (ID_21): After the modification of a flight plan, the new guidance information should be produced in less than 15s. Any change to a flight plan should perform a visual feedback to the pilot in less than 2s. The first two legs of a newly modified flight plan shall be available in less that 5s.

Impacts on WP: 2, 5

Priority: High

Domain: Avionics

Periodic nearest real-time requirements (ID_23): The nearest airport list shall compute by task NEARP1 in less than 3s, and displayed in less than 2s.

Impacts on WP: 2, 5

Priority: High

Domain: Avionics

3.1.6 Vehicle self-awareness in ADAS

Determine location (ID_24): CPS has to be able to determine its location in relation to the ODD. The vehicle has to be able to decide if it is inside or outside of a location-specific ODD. The location in the ODD may be required, depending on the item definition.

Impacts on WP: 1, 2, 3, 5

Priority: High Low

Domain: Automotive

Perceive relevant objects (ID_25): All entities that an automated driving system requires for its functional behavior have to be perceived, optionally pre-processed, and provided correctly. The

highest priority is placed on entities with an associated risk of collision. Sample entities include dynamic objects (e.g., other road users and characteristics of the respective movement), static instances (e.g., road boundaries, traffic guidance and communication signals) and obstacles.

Impacts on WP: 1, 2, 3, 5

Priority: High

Domain: Automotive

3.1.7 Prediction and planning for ADAS

Predict the future behaviour of relevant objects (ID_26): The relevant environment model needs to be extended by the predicted future state. The aim is to create a forecast of the environment. The intention of the relevant objects have to be interpreted in order to form the basis for predicting future motion.

Impacts on WP: 1, 2, 3

Priority: High

Domain: Automotive

Create a collision-free and lawful driving plan (ID_27): To ensure a collision-free and lawful driving policy, the following has to be respected:

- Maintain a safe lateral and longitudinal distance to other objects.
- Comply with all applicable traffic rules.
- Consider potential areas where objects may be occluded.
- In unclear situations the right of way is given, not taken.

Impacts on WP: 1, 2, 3, 5

Priority: High

Domain: Automotive

Correctly execute and actuate the driving plan (ID_28): The corresponding actuation signals for lateral and longitudinal control have to be generated based on the driving plan.

Impacts on WP: 1, 2, 3, 5

Priority: High

Domain: Automotive

3.1.8 ADAS related communications

Communicate and interact with other road users (ID_29): Automated driving vehicles are required to communicate and interact with other road users, depending on the ODD and the use cases.

Impacts on WP: 1, 2, 3, 5

Priority: Medium

Domain: Automotive

3.1.9 ADAS malfunction and under-performance detection

Determine if specified nominal performance is not achieved (ID_30): Any element of CPS can, either on its own or in combination with others, result in adverse behavior. Therefore, mechanisms are required to detect the adverse nominal performance of the system.

Impacts on WP: 1, 2, 3

Priority: High

Domain: Automotive

Detect when degradation is not available (ID_31): It has to be assured that a possible unavailability of the fail-degraded capability is detected. If the degradation strategies depend on the degradation reason, the degradation reason has to be identified.

Impacts on WP: 1, 2, 3

Priority: Medium

Domain: Automotive

Ensure safe mode transitions and awareness (ID_32): Ensure that mode transitions are performed correctly and controlled by the vehicle operator affected if necessary. The vehicle operator affected has also to be aware of the current mode and their responsibility deriving from it. For example, actuating an automated mode is permitted only when inside the ODD, and it will be deactivated prior to leaving the ODD or as a result of the vehicle operator taking control again.

Impacts on WP: 1, 2, 3, 4, 5

Priority: High

Domain: Automotive

React to insufficient nominal performance and other failures via degradation (ID_33): Due to possibly unavailable nominal performance capabilities and other failures (e.g., based on hardware faults), CPS has to degrade within a well-defined amount of time.

Impacts on WP: 1, 2, 3, 4

Priority: High

Domain: Automotive

Reduce system performance in the presence of failure for the fail-degraded mode (ID_34): The reaction in case of failures during fail-degraded mode has to be defined.

Impacts on WP: 1, 2, 3, 4

Priority: High

Domain: Automotive

Perform ODD functional adaption within reduced system constraints (ID_35): CPS operation with ODD functional adaption is actuated as nominal capabilities with new limits. Multiple functional adaptions are possible. The limitations have to be defined such that the functional adaption can be stated as safe. Therefore, it may be necessary to avoid a permanent operation. A well-defined timeframe for an additional reaction is required.

Impacts on WP: 1, 2, 3, 4, 5

Priority: High Low

Domain: Automotive

3.2 Requirements on Human-empowered Intelligent subsystem

3.2.1 Performance monitoring, evaluation and assessment

Hardware observability (ID_9): GP Core Performance Monitor Counters shall be available for the general-purpose core and memory hierarchy high low HW avionics.

Impacts on WP: 2, 5

Priority: Low

Domain: Avionics

Hardware observability (ID_10): AI Accelerator Performance Monitor Counters should be available for the AI accelerator internals medium low HW.

Impacts on WP: 2, 5

Priority: Low

Domain: Avionics

Figures on performance requirements for AI-based personalization process (ID_90): To properly dimension the computing platform and to tune the orchestration process is of paramount importance to have figures about the computing performances expected by AIaaS based applications.

Impacts on WP: 1, 2, 5

Priority: Low

Domain: Automotive

Hardware requirement of the functional modules of the AIaaS (ID_101): Each functional module of the AIaaS should have associated their specific hardware requirements (RAM, computational load, need of local storage) to assure their compatibility with the underlying hardware platform.

Impacts on WP: 2, 4

Priority: High

Domain: Automotive and Avionics

AIaaS subsytem to manage internal module violations (104): The AIaaS platform requires a subsystem to collect and react to the violations of the reliability metrics raised by the functional modules of the platform.

Impacts on WP: 2, 3, 4

Priority: Medium

Domain: Automotive and Avionics

Non-impairment of dependability (105): HW and SW architecture choices (WP1-WP2) shall not impair the reliability of the dependable subsystem; this meta requirement applies to the non-dependable part of the CPS.

Impacts on WP: 1, 2, 3, 5

Priority: High

Domain: Automotive and Avionics

3.2.2 Network communication

*Data transfer for AIaaS federation (ID_72):*The edge system must be able to send and receive sets of ML model parameters to/from the cloud over the network.

Impacts on WP: 2, 4

Priority: High

Domain: Automotive and Avionics

Inter-edge AIaaS communication (ID_79): Different modules of the AIaaS system need to exchange predictions (e.g., the output of model A to the input of model B on a different edge device).

Impacts on WP: 2, 4

Priority: Medium

Domain: Automotive and Avionics

Network coverage and available protocols (ID_86): Edge subsystem needs to be informed (i.e., properly configured) about the network channels available in the target area (e.g., cellular, satellite, Wi-Fi)

Impacts on WP: 2, 5

Priority: High

Domain: Automotive and Avionics

Cellular 3G/4G connection (ID_90): Vehicle shall be able to connect to 3G/4G network, in order to communicate with other road users and cloud.

Impacts on WP: 1, 2, 5

Priority: High

Domain: Automotive

3.2.3 Software development and deployment

Compatibility of developed SW with the chosen HW (92): The software implementation of AIaaS at the edge must run on the available hardware.

Impacts on WP: 2, 4

Priority: High

Domain: Automotive and Avionics

Definition of learning functionalities of the AIaaS (ID_94): There should be a list of functionalities from the modules of the AIaaS platforms (e.g., RNN, Classifier, etc..) with a requirement on their API

Impacts on WP: 2, 4

Priority: Critical

Domain: Automotive and Avionics

Definition of the possible pattern for the access of the Edge storage (ID_95): A pattern defines the characteristics of the data flow between the source and destination, such as: streaming/storing each value as it is available, sending batches of a assigned size, reading/writing data via queues; the actual partners depend on AIaaS system implementation as well as from the application model we will adopt for the AIaaS.

Impacts on WP: 2, 4

Priority: Medium

Domain: Automotive and Avionics

Common interface for functional modules of AIaaS (ID_96): Functional modules of the AIaaS should expose a common interface that allows orchestration operations such as initialization, creation, connection, etc.

Impacts on WP: 2, 4

Priority: High

Domain: Automotive and Avionics

AIaaS application definition (ID_97): An AIaaS application should be specified as a workflow graph of functional modules, including: its parameters, external communication channels, etc.

Impacts on WP: 2, 4

Priority: High

Domain: Automotive and Avionics

3.2.4 Data and Metadata

Common data format for the data brokering (ID_99): The data brokering should have a defined format. Functional components of the AIaaS must either use the same format or implement an adapter.

Impacts on WP: 2, 4

Priority: High

Domain: Automotive and Avionics

Common meta-data format for the data brokering (ID_100): The data brokering should have a defined format for the meta-data associated to each type of sensor data

Impacts on WP: 2, 4

Priority: High

Domain: Automotive and Avionics

Non-volatile storage for the AIaaS platform (ID_103): The AIaaS platform requires access to a non-volatile storage unit, to store data and meta-data.

Impacts on WP: 1, 2, 4, 5

Priority: High

Domain: Automotive and Avionics

3.3 Requirements on the High-Performance Computing and Communication Infrastructure as a whole

3.3.1 Communication

Communication of the AIaaS modules with the vehicle (ID_75)

Impacts on WP: 2, 3, 4

Priority: High

Domain: Automotive

Communication of the vehicle with the AIaaS modules (ID_76): The AIaaS modules must be able to receive as input the data from the sensors and the state of the vehicle

Impacts on WP: 2, 3, 4

Priority: High

Domain: Automotive and Avionics

Intra-edge AIaaS communication (ID_78): Different modules of the AIaaS system need to exchange predictions (e.g., the output of model A to the input of model B)

Impacts on WP: 2, 4

Priority: High

Domain: Automotive and Avionics

Access to vehicle sensors' data (ID_85): Vehicle sensors data (both related to humans and the vehicle itself) must be readable through a standard M2M protocol

Impacts on WP: 2, 4, 5

Priority: High

Domain: Automotive

3.3.2 Reaction to state changes

Detect changes in car state or context (ID_81): Use sensors and cameras either integrated to the car (accelerometers, gyroscopes, luminosity sensors, LIDAR, camera, IR beam and sonar) or attached by the driver to detect car's state and context (e.g., road and weather conditions, obstacles) to feed them in the human-centric personalized autonomous driving system.

Impacts on WP: 2, 4, 5

Priority: Medium

Domain: Automotive

Perform ADAS adaptation for model fine tuning (ID_82): Depending on the overall driving style of the passenger, his state, the car's context or the currently selected driving style of the passenger for the specific scenario, we change the model parameters in order to get a slightly different distribution at the variation of the model's parameters.

Impacts on WP: 2, 4, 5

Priority: Medium

Domain: Automotive

3.3.3 Software development, deployment and system description

Software packaging and deployment (ID_87): Software targeting both real-time management and personalization as well as anomaly detection should be packaged in a way allowing their deployment in the Cloud-Edge continuum.

Impacts on WP: 1, 2, 5

Priority: High

Domain: Automotive and Avionics

Figures on data production rate and QoS requirements (ID_88): There is a need for information about the actual amount of data generated by all the sensors in the controlled environment per unit of time. This is a fundamental requirement to properly dimension and tune the communication and computing platform of the CPSoS.

Impacts on WP: 2, 4, 5

Priority: High

Domain: Automotive and Avionics

3.3.4 Data management

Data brokering within the AIaaS platform (ID_98): There should be a mechanism of data brokering that mediates the access to the sensor (human, vehicle) data from the functional modules of the AIaaS.

Impacts on WP: 2, 4

Priority: High

Domain: Automotive and Avionics

Annotated data for AIaaS (avionics traces) (ID_106): Need to collect software/hardware traces from the aircraft sensors to train the AI modules.

Impacts on WP: 2, 4, 5

Priority: High

Domain: Avionics

3.3.5 Security

Secure access from application to the adaptive system of the vehicle (ID_102): The autonomous driving system should expose an interface that allows the application of the AIaaS to tune the parameter of the adaptive system of the vehicle in a secure way.

Impacts on WP: 1, 2, 3, 4, 5

Priority: Critical

Domain: Automotive

4 Design

TEACHING develops a human-aware CPSoS for autonomous safety-critical applications, relying on a distributed, energy-efficient and dependable AI, leveraging innovative edge computing platforms integrating both general purpose and specialized hardware.

A key objective of the TEACHING project is to design a computing platform supporting the development and deployment of autonomous, adaptive and dependable CPSoS applications, allowing them to exploit a sustainable human feedback to drive, optimize and personalize the provisioning of their services. To this end, TEACHING aims to realize a human-aware approach, i.e., evaluate the reactions of the human to steer the operation of the autonomous CPSoS. In the spirit of the SoS approach, TEACHING aims at realizing a computing infrastructure comprising a specialized human-centric system of sensing devices, integrated within the CPSoS, and encompassing wearables sensors for tracking bio-signals as well as touchless environmental sensors. These will serve as information sources feeding AI models specialized in the recognition and characterization of the human physiological, emotional and cognitive (PEC) state. A detailed description of the entire AI process is presented in deliverable D4.1. The reactions monitored by such a system will serve to drive CPSoS operation in synergy with the humans. The two TEACHING use cases, in autonomous driving and avionics domains, actually provide demonstrators showing how the operations of an autonomous transportation application can be reconfigured online based on the PEC reactions of the human within the CPSoS.

TEACHING high-performance computing and communication infrastructure (HPC^2I) will be the enabler of such achievements. It will be designed accordingly to an edge-distributed and federated approach, allowing to maintain important parts of the computation close to the end user and the data sources, reducing connectivity-related issues that may affect reliability and security, while enabling the exploitation of a virtually endless number of resources on Clouds, if needed. HPC^2I leverages embedded systems and specialized hardware to develop high-performance edge resources allowing to run computational demanding tasks where data is actually generated. To this end, it does integrate different specialized microcontrollers (e.g., for fail-operational aspects), to general purpose dynamically reprogrammable accelerators for heavy computations, such as FPGAs. Such computing backbone is, of course, integrated with the necessary sensing devices to realize the human-monitoring CPS.

At software system level, TEACHING manages the high heterogeneity and specificity of the hardware resources at the edge employing abstractions, communication and orchestration layer leveraging approaches borrowed from cloud- and edge-computing, enabling the management of resources and applications onto a computing continuum spanning the whole TEACHING computing platform. This will support the developer in managing the burden following from the complexity of the deployment of the different application components on the most adequate resources for their functions (data processing, safety-critical routines, cybersecurity watchdogs, etc.).

The methodology developed in TEACHING will also deliver proper programming abstractions that fit the complexity of the infrastructure and the needs of AI/ML models, whose implementation should be provided on a variety of different resources ranging from energy efficient silicon-AI, to multicores and co- processors like GPUs and FPGAs.

At the run-time system level of the infrastructure, autonomous applications on CPSoS demand a highly adaptable framework designed to continuously monitor the system execution metrics based on the workload and non-functional parameters like reliability, safety and security enforcement. In TEACHING, the high degree of reconfigurability of the platform, both at the

communication level and in terms of re-programmable hardware resources (e.g., FPGAs) when adopted, will be profitably exploited by applications running on top of it in a seamless way in order to achieve trade-offs between performance, response time and cost of running applications in terms of energy. In the remaining of this section is described the conceptual architecture of the HPC²I (Section 4.1.1), a brief description of the flows of information (Section 4.1.2) occurring between the different entities realizing the infrastructure distinguishing between the two project use cases.

4.1 High-Performance Computing and Communication Infrastructure (HPC²I)

The combined set of hardware, system software, development methodologies, orchestration strategies and network protocols used in TEACHING will realize the HPC²I, whose conceptual architecture is sketched in Figure 2 and briefly presented in Section 4.1.1.

Such infrastructure is aimed at supporting the execution of the TEACHING platform and its use cases. As such, it does need to satisfy both the specific requirements associated with the automotive and avionics use cases, basically ensuring the viability of the infrastructure as execution platform. Nevertheless, HPC²I is in charge of making possible the actual exploitation of the aforementioned human-based empowerment of the platform.

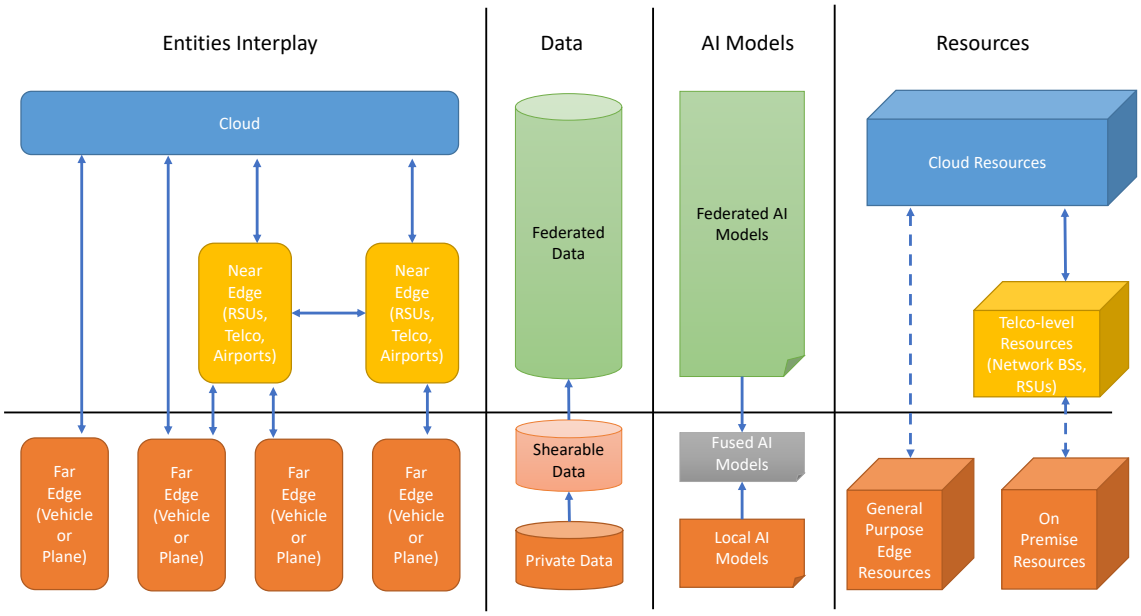

Figure 2 - Conceptual Architecture of HPC²I

Both the use cases of TEACHING have quite specific requirements in terms of hardware and software. In fact, as many other solutions targeting the avionics and automotive domains, have special needs in terms of dependability and safety that reflect in the hardware used for their execution. In the context of TEACHING project, the hardware supporting the specific needs of both the use cases will be provided by I&M[66]. Conversely, the TEACHING platform, intended

[66] Additional information about the I&M board supporting the TEACHING use cases is provided in deliverables D1.1 and D6.1

as the set of tools for development, deployment, management and communication enablement of adaptive applications based on human feedback, will be based on general purpose hardware and software. Clearly, this leads to have a computing and communication infrastructure that is highly differentiated depending on the specific requirements of the applications running on it.

The actual CPSoS nature of TEACHING platform is described in D1.1, where are detailed the features distinguishing CPSoSs from different viewpoints, contextualizing the architecture of the TEACHING platform under these perspectives. Such deliverable also provides additional details on the hardware that the HPC^{2}I will exploit, both detailing the specialized on-premise resources, as well as the cloud/edge ones. In this deliverable, instead, we focus on the actual interplay on the computational and communication resources, whose combined action realizes the HPC^{2}I. We clarify how HPC^{2}I and its key technologies will result in an enabling environment for the TEACHING platform and its use cases.

4.1.1 Conceptual architecture

HPC2I conceptual architecture is depicted in Figure 2. The figure is organized in two horizontal layers – depicted entities can be either part of the "local layer" (i.e., co-located with data generators and data consumers) or the "remote layer" (i.e., located at a different place) – and four vertical segments. Each segment describes the interaction between the two layers from a different perspective.

The resources belonging to the local layer are the "Far Edge" of HPC^{2}I, whereas the remote layer of HPC^{2}I is composed by "Clouds" and "Near Edge" resources.

As show by the second segment of Figure 2, the different layers manage different types of data. The local layer is entitled to access all the data generated locally. These pieces of information are the "private data". A subset of such data, that is not considered as much sensitive as to be kept exclusively in the Far Edge, can be shared, thus exported to the "remote layer". It can be also envisioned a pseudo-anonymization process to conduct on a subset of private data to feed the sharable data. Altogether the sharable data of the different Far Edges, realizes the Federated Data.

Private data and Federated Data are used to generate local and Federated AI models, respectively. Every Far Edge merges the Federated AI models and the Local AI models to generate Fused AI models representing a combination of the two.

It is worth to notice that Federated Data and, consequently, the Federated AI models are not necessarily a single unique instance. We envision regional-level federations of data and AI models. According with this perspective the resources at the "Near Edge" will behave as collectors of federated data at regional level as well as generators of regional-level AI models.

As we mentioned above, the local layer is composed of both on-premises resources and general-purpose edge resources, which are co-located. The former type of resources is aimed at supporting the execution of mission-critical applications, the latter type run the human-empowered AI subsystem. The remote layer is composed by resources that may be hosted in a cloud or in a Near Edge device (e.g., RSUs or Telco resources).

4.1.2 Information flows

Resources at the Far Edge can either communicate directly with a Cloud or reach it through Near Edge resources, i.e., Far Edge connects to a Near Edge that eventually transmit the payload to the remote cloud. We envision the possibility that different types of network may characterize

the link connecting the Far Edge with Clouds, the one between Far Edge and Near Edge, as well as the one between Near Edge and Clouds.

Automotive Use Case

In the automotive use case, represented in Figure 3, we envision vehicles (Vehicle1, V2, …, Vn) as the Far Edge of the system which embeds both the on-premise resources supporting the execution of ADAS, and general-purpose resources supporting the local fraction of the human-empowered AI system. As such the local layer of the system support the execution of applications that may come in the form of containers, unikernel[67] and ad-hoc software.

In case Road Side Units (RSUs in the figure) are available, the architecture is enriched by a Near Edge layer that can support more advanced features such as regional-level federation of data as well as regional-level definition of AI models. The type of software that we can expect to run on these devices is similar to the one run at the Far Edge.

It can be also envisioned an interaction occurring between different vehicles, mediated by RSUs. This type of interaction is represented in the figure below by the arrow and indicated as RSUs-mediated V2X link.

Clouds represent the centralized data center that hosts the federated data and generates the Federated AI models. On Cloud the applications are deployed using VMs and containers.

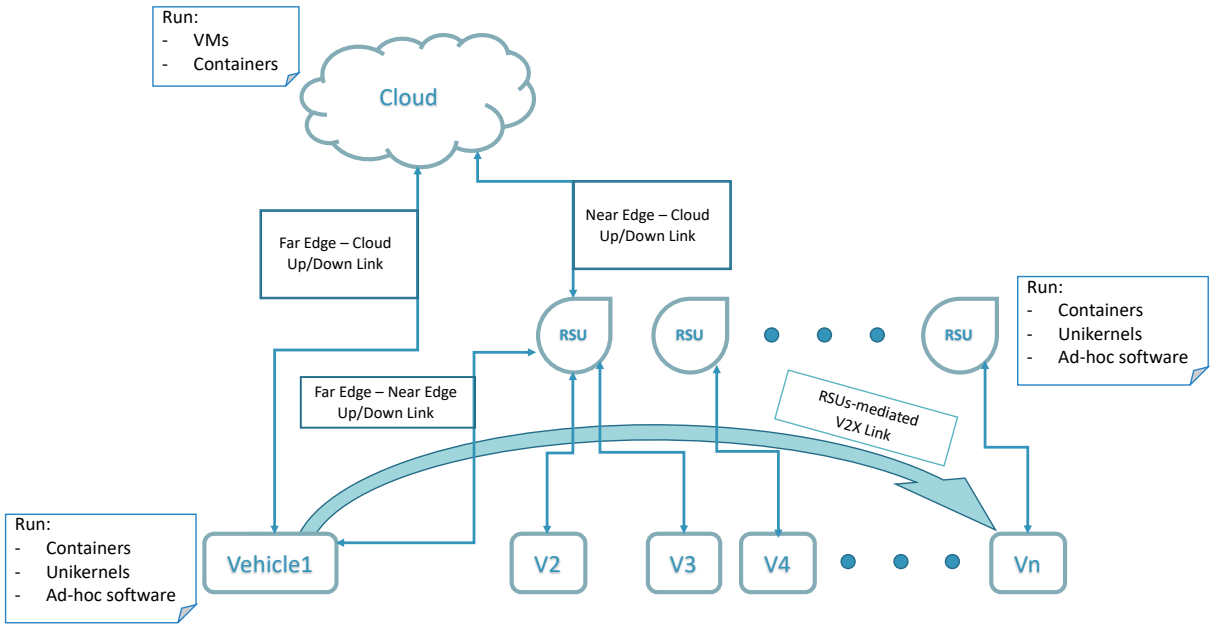

Figure 3 - Automotive Use Case

[67] http://unikernel.org/

Avionics Use Case

In the Avionics use case (Figure 4), from the HPC²I perspective, the information flow does not differ too much from the Automotive scenario. In this case, Far Edge is represented by airplanes (Plane1, P2, …, Pn). The software run at this Far Edge basically matches the one of the Automotive use case. Near Edge is represented by airports which the airplanes interface when landed. In this case we envision that the airport may be equipped with resources that are more complex than the ones embedded in an RSU. As a consequence, we expect that the software deployed and run in the Near Edge matches the one hosted in the Cloud, e.g., VMs and containers.

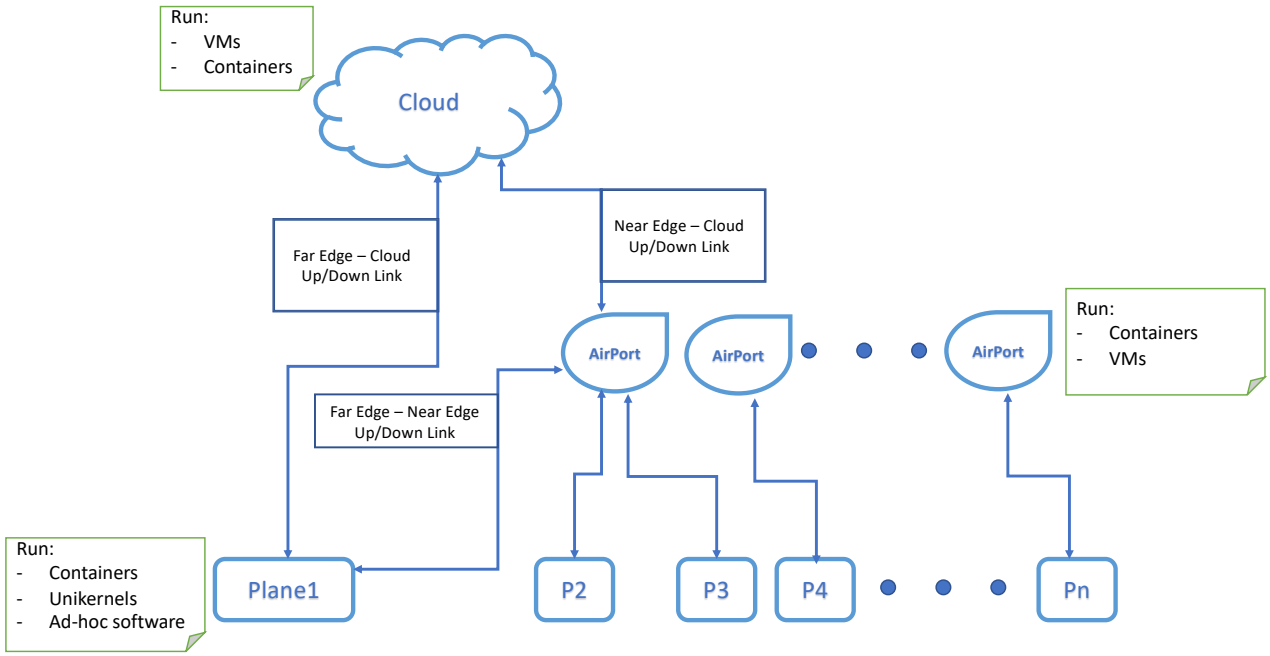

Figure 4 - Avionics Use Case

5 PRELIMINARY EVALUATIONS ON TECHNOLOGIES, PROTOCOLS AND TOOLS

This section reports some key technologies that are candidate to give a ground to the design and development activities aimed at the creation of the HPC^{2}I.

The aim of this section is to present such solutions, which have been identified during the first year of the project, that are under consideration to serve as baseline technologies on which to build the HPC^{2}I.

In particular, Section 5.1 introduce interesting technologies to exploit to simulate the TEACHING CPSoS overall. Section 5.2 presents tools enabling the AI-based processing at the edge whereas Section 5.3 focuses on the programming techniques that can be exploited, mostly at the edge to properly exploit GPUs and FPGAs. The solutions to measure the performances in a CPSoS are instead the focus of Section 5.4.

Moving from computing aspects to communication related technologies, in Section 5.5 are highlighted approaches enabling the efficient processing of streams, the communication paradigms supporting the data transfer over the network (Section 5.6), and more specifically in the area of vehicular networks (Section 5.7).

5.1 Representation and simulation of the TEACHING CPSoS

The definition of a computational environment on top of the TEACHING CPSoS is a complex task. The interaction of many computational resources with heterogenous computational capabilities and possibly variable geographical location needs to be considered. Therefore, there is the need of a software simulation that abstracts the resource and the computational task in a way to provide reproducible, controllable, and cost-effective experiments and fine-tuning methodologies prior to the actual deployment in a real cloud-edge environment. The remainder of this section describes several software that can possibly be used during the project to simulate the computational environment of TEACHING.

The core aspect is the simulation of the “cloud-edge continuum" execution environment, in which both cloud computing and edge resource participate to the execution of a common application. To properly simulate this aspect, it is necessary to take into consideration the computational capabilities of the cloud and edge resources and the characteristics of the communication channel among them.

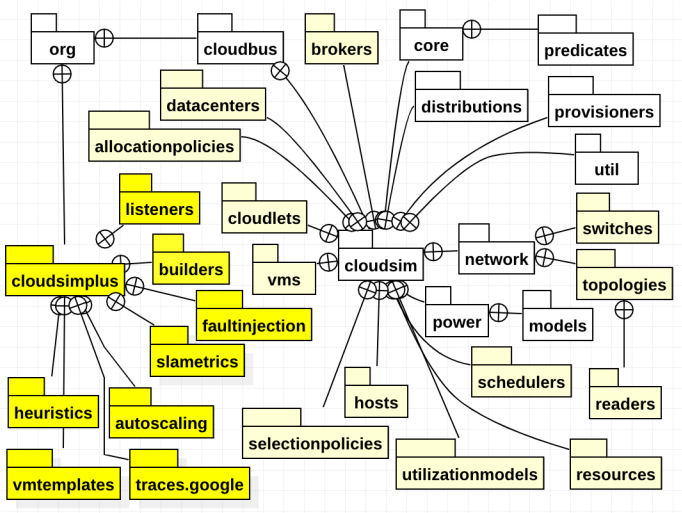

Figure 5 The modular architecture of CloudSim Plus

The CloudSim Plus[68] (CSP) simulator is an open-source simulator written in Java that has been gained a lot of momentum in the last years. At its core, CSP is based on CloudSim, a well-known and widely used simulator. CloudSim[69] is a pure cloud simulator: it focuses on providing a realistic model for the implementation of allocation policies for virtualized resources in cloud-based environment. However, CSP inherent modularity allows to personalize many functionalities to simulate the ecosystem of TEACHING (see Figure 5). Many of the functionalities that are of interest for the TEACHING project such as data center network topologies and message-passing applications, federated clouds, cloudlets definitions, distribution policies, and network definitions come already in the current distributed package.

The implementation and personalization of a given functionality it is rather easy as it is just necessary to implement the corresponding java interface. However, the addition of brand-new functionality can be more complex as it requires the definition of a new interface and to program the interaction of the new module with all the other one in the simulator. For example, in the CSP simulator there is no an explicit definition of an edge resources. While this can be implemented as a cloud resource with specific constraints in terms of hardware characteristics and geographical position, it cannot be ruled out that issues can arise when implementing a complex edge computation use case.

PureEdgeSim[70] (PES) is an open-source simulator based on CSP and adds extension for Edge and Mist computing. PES natively supports task orchestration, the definition of offloading policies and modelling of data sources such as IoT devices. These features pair nicely with the

[68] M. C. Silva Filho, R. L. Oliveira, C. C. Monteiro, P. R. M. Inácio and M. M. Freire, "CloudSim Plus: A cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness," *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, Lisbon, 2017, pp. 400-406, doi: 10.23919/INM.2017.7987304.

[69] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, Cesar A. F. De Rose, and Rajkumar Buyya, CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms, Software: Practice and Experience (SPE), Volume 41, Number 1, Pages: 23-50, ISSN: 0038-0644, Wiley Press, New York, USA, January, 2011.

[70] C. Mechalikh, H. Taktak and F. Moussa, "PureEdgeSim: A Simulation Toolkit for Performance Evaluation of Cloud, Fog, and Pure Edge Computing Environments," *2019 International Conference on High Performance Computing & Simulation (HPCS)*, Dublin, Ireland, 2019, pp. 700-707, doi: 10.1109/HPCS48598.2019.9188059.

TEACHING ecosystem, in which on-board sensor data is used to instruments the decision making that happens at the application level. Further, PES supports the definition of custom mobility models for the edge/mobile nodes, allowing to factor in the mobility of vehicles (e.g., car or airplanes) in the orchestration of the tasks at the application level (see Figure 6).

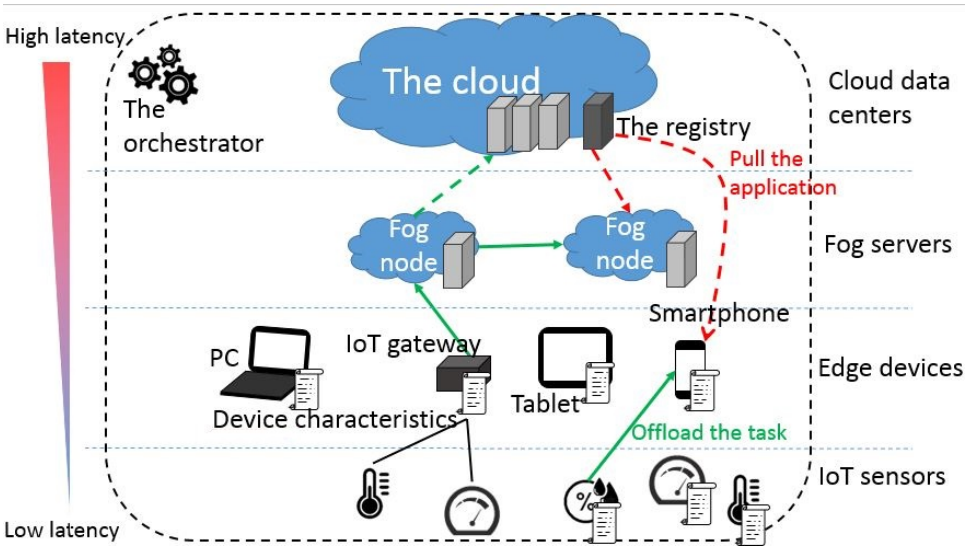

Figure 6 Architectural overview of PureEdgeSim

Several other simulations software that specifically address the cloud-edge continuum are currently being under development, however not widely used. EdgeCloudSim[71] is an open-source simulator similar to PES but showcase a limited number of features. The RECAP Simulator[72] has been developed in the context of the H2020 EU project RECAP, and it supports in the evaluation of the trade-off for different deployment solution in terms of cost, energy, resource allocation and utilization.

5.2 AI toolkit for edge devices

Together with the increase of popularity of AI, there have been an increase of popularity also in Neural Network frameworks. These allow a developer to design and train a NN model without having to deal with the low-level algorithms, but only feeding training data and choosing the architecture. Among these, the most known are TensorFlow , Caffè , and PyTorch . At the present day, it seems that there is no alternative better than the other in terms of performances. The choice can be done based on the compatibility of these frameworks with the target edge device. Furthermore, this will be the device with reduced resources, where optimizations will be more needed. An open standard is also trying to bridge together the several

[71] C. Sonmez, A. Ozgovde and C. Ersoy, "EdgeCloudSim: An environment for performance evaluation of Edge Computing systems," *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, Valencia, 2017, pp. 39-44, doi: 10.1109/FMEC.2017.7946405.

[72] J. Byrne *et al.*, "RECAP simulator: Simulation of cloud/edge/fog computing scenarios," *2017 Winter Simulation Conference (WSC)*, Las Vegas, NV, 2017, pp. 4568-4569, doi: 10.1109/WSC.2017.8248208.

libraries, called ONNX (Open Neural Network Exchange). It aims to be an exchange format for trained models so that the final result of the development will be independent from the framework used for the training phase.

In deliverable D5.1, two architectures are considered as edge devices: a Xilinx Ultrascale Zynq+ and an i.MX8 Quad Max from NXP. The first one is a System on Chip that combines ARM cores and FPGA programmable hardware. Xilinx, the company that designed it, gives access to Vitis AI, a software toolkit for accelerated AI inference, exploiting a processing unit deployed on the programmable hardware. Starting from the models described in Tensorflow and Caffe, one can optimize (using quantization and pruning) and compile them for the target. All this can be done without knowledge of hardware description languages. It claims high performance with a low energy footprint while allowing developers with neural network expertise to exploit hardware acceleration without much effort. For instance, Xilinx claims to reach 33fps while executing Resnet50 on a ZU3EG.

On the other hand, the i.MX8 Quad Max can rely on six Cortex-A cores and two GPUs for AI inference. NXP distributes eIQ, which is a set of software needed to deploy Neural Network on the device. It supports TensorFlow models, that are optimized by TensorFlowLite: It is another set of tools specific for pruning and quantization, making a TensorFlow model lighter and able to run on resource constrained devices.

The ARM NN inference engine uses the Neon accelerator inside the Arm Cores. NXP modified it to make it compatible also with the GPU present on the i.MX8.

As described in deliverable D1.1 TensorFlow/Lite is the framework that is currently the candidate to be used in TEACHING. This is also coherent with the considerations made in D4.1.

5.3 Preliminary evaluation of GPU/FPGA programming technologies

During the preliminary activities of TEACHING work package 2 have been evaluated the flexibility offered by modern high-level programming tools for GPUs and FPGAs, in order to select the right programming technologies for developing the streaming libraries needed for the project purposes. In this investigation phase, we have focused on two programming models in detail: *CUDA* for developing streaming libraries on NVIDIA GPUs, with special focus on integrated GPUs for IoT-oriented boards, and *OpenCL* for FPGAs. In the rest of this section, we will describe these two programming models and the specific features that we claim are of special interests for HPC^{2}I.

Streaming on integrated GPUs with CUDA

Our goal is to develop a streaming library able to leverage co-processors like GPUs. More specifically, owing to the requirements of the CPSoS of the TEACHING environment, we are assuming the presence of embedded devices equipped with small power-oriented ARM multicores and integrated GPUs sharing the memory with the CPU (like in the Jetson Nano Developer Kit shipped by Nvidia). The *de-facto* programming model for Nvidia GPUs is CUDA. CUDA is a programming model used to exploit GPUs for efficient execution of data-parallel kernels. CUDA kernels are special functions in C/C++ code (annotated with the `__global__` directive), which are executed asynchronously on the device. The execution model is the SIMT one (Single Instruction Multiple Threads), where each kernel consists of several

CUDA threads grouped into blocks. Threads are scheduled on the underlying CUDA cores (stream processors) in groups of 32 threads called warps, each one executing the same instruction in a SIMD lockstep manner. Warps are efficiently scheduled by the hardware/firmware resources available on the GPU device.

Developing a streaming library able to leverage integrated GPUs is of special importance for the goals of WP2. Embedded devices are often equipped with co-processors that represent powerful facilities able to accelerate specific computing tasks with limited additional power-consumption requirements. However, streaming tasks like filtering, sampling, aggregation and, more generally, stateless and stateful streaming transformations pose several challenges that often fight with the traditional way of programming GPU devices. We have identified four main challenges to obtain satisfactory performance:

- **Challenge 1**: "**enable efficient micro-batching techniques to exploit GPU processing capabilities**". The basic way to use GPUs is to extract data parallelism. In the streaming domain, each input is generally small (e.g., of few bytes, like a sensor reading in the TEACHING environment). So, we aim at extracting data parallelism by buffering inputs in small batches, whose processing is offloaded on the device for fast processing.
- **Challenge 2**: "**avoid global synchronization in the use of the GPU device by many concurrent/parallel activities on the CPU**". GPU operations like kernel executions and memory copies (host-to-device and device-to-host) are generally executed in issuing order. However, in case of multiple threads on the host triggering such kind of operations, the CUDA configuration needs special care to allow parallel execution of such operations, triggered by different CPU threads on the same device.
- **Challenge 3**: "**frequent memory allocations on the GPU-accessible memory are detrimental for performance**". To allocate GPU-accessible memory, CUDA provides some `cudaMalloc` routines whose behavior is similar to traditional `malloc` in C/C++. However, the use of `cudaMalloc` requires a global synchronization on the device: i.e. all the previously running GPU operations (kernels and copies) must be complete before the memory allocation request can be processed. If executed frequently, like expected in streaming scenarios where data continuously arrive for processing, this could generate a significant performance drop and under-utilization of the device's resources.
- **Challenge 4: "stateful streaming operators are difficult to be executed on GPU"**. GPU-based streaming operators offload the processing of batches on the device. For stateless operators, each input in the batch can be processed in parallel by a CUDA thread. So, this does not generate particular problems since the data-parallel programming style is easily enforced in such a scenario. Much more challenging is instead the micro-batch processing for stateful operators. Indeed, they often require that all the inputs of the same sub-stream (usually identified by a specific value of a key attribute in the input items) are executed sequentially by reading and updating an internal state maintained for each sub-stream. So, a CUDA kernel should enforce this constraint, i.e. all the inputs in the batch with the same key attribute must be processed sequentially by the GPU.

After our preliminary investigation, we have found specific techniques and CUDA features that represent viable solutions to face the previously described challenges:

- Micro-batching techniques should allow the fine-grained tuning of the batch size which should be adapted *automatically* by the run-time system in order to find the best balance between *processing latency* (higher with longer batches) and *throughput* (usually higher with larger batches). Furthermore, batches must be prepared as much as possible

asynchronously, by overlapping the preparation of the next batch with the transmission and processing of the previous one. This idea can be exploited thanks to the asynchronous nature of kernels in CUDA, and the possibility to use asynchronous copies that do not block the calling host thread.

- To avoid global synchronizations, we will leverage the mechanism provided by *CUDA streams*. Each CUDA stream is a sort of ordered queue of GPU operations (kernels and copies), which guarantees that all the operations issued on the same CUDA stream will be completed in the issuing order. However, no synchronization exists between GPU operations issued on different CUDA streams, which are executed concurrently/in parallel on the device. So, different operators of the streaming application running on the CPU will be composed by several host threads (C++ or Posix ones). The threads enabled to offload kernel processing on the GPU will be equipped with independent CUDA streams for maximizing parallelism without requiring global synchronizations on the device.
- To avoid the impact of frequent `cudaMalloc`, a *custom memory allocation mechanism* is needed to allow recycling already allocated GPU memory areas. Implementing a custom memory allocator is notoriously a complex problem. However, the unique characteristics of streaming applications, based entirely on the producer-consumer paradigm for exchanging data items across the data-flow graph, can be exploited to design a simple yet efficient recycling mechanism of GPU-accessible memory areas. This can be done by using *lock-free queues* that can be used by host threads to recycle areas allocated for batches whose processing has been already completed.
- Stateful kernel processing requires special care. To enforce the stateful requirements that all the inputs with the same key attribute are processed sequentially, the kernel design needs a preliminary preparation phase to compute support arrays that allow CUDA threads to access exactly one time each input in the batch, and to enforce the stateful property. Such arrays should contain the indexes of the inputs in the batch with the same key attribute, and they are followed to execute inputs with the same key sequentially by the same CUDA thread. Furthermore, since the code executed by each CUDA thread might be generic, and probably divergent among threads on the same warp, the kernels should be written in order to enforce *intra-warp parallelism* whenever it is possible and convenient. Intra-warp parallelism is a way to develop *persistent-thread kernels* in CUDA, where only one thread per warp is really working while the others are idle. This can be used, at least when the inherent parallelism within a stateful batch is limited, to increase GPU occupancy and to avoid the divergent branch problem, by using whole warp as unit of parallelism (they can be considered as conventional MIMD threads).

The picture below (Figure 7) summarizes the overall ideas of our methodology for streaming on GPUs, which will represent design choices that we will follow during the next steps of our activities for developing a streaming library with full support for integrated GPU devices on embedded resources.

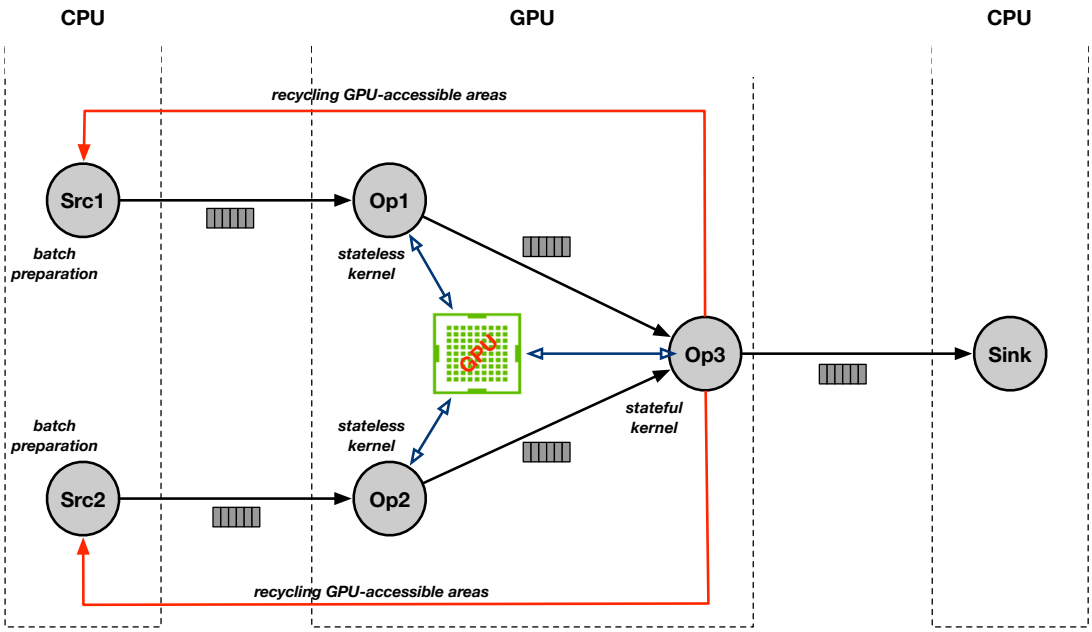

Figure 7 – Streaming on heterogeneous systems with multi-core CPUs and integrated GPUs

Other aspects of the design depend on the specific features of the underlying GPU device. In particular, for integrated GPU devices we have the opportunity to exploit a physically shared memory between CPU and GPU. If properly used, this facility would allow useless copies between host and device memories. However, there are special considerations that must be considered during the development. On Tegra architectures, like the Jetson Nano Developer Kit and other models of integrated Nvidia GPUs, there are two main ways to implement a memory area accessible by both the host and the GPU. The first is to use the unified CUDA memory (through `cudaMallocManaged` calls). The second is by exploiting pinned-memory allocated with `cudaMallocHost`.

Unified memory represents a very powerful mechanism provided by CUDA, which greatly simplifies programming without requiring explicit memory transfers. However, unified memory can be inefficient on older GPU models, because such GPU devices are not equipped with hardware page-fault engines and the whole managed memory must always be transferred on the device before processing and cannot be accessed by the host during the kernel computation. On Tegra architectures, such transfers are avoided (because of the physical memory). However, unified memory still poses some problems when used with CUDA streams. Indeed, each managed memory area must be attached to a specific CUDA stream, and cannot be accessed by the host threads when some kernels issued on that stream are running on the device. This requires a special treatment and evaluation of such attaching routines provided by CUDA.

Pinned-memory instead is much easier to use on Tegra devices. It can be safely accessed by host threads and by the device without any attachment routines to be called per CUDA stream. However, on the host the areas of pinned-memory are not cached on some Tegra devices (like the Jetson Nano), and this might generate a higher processing time for working on such memory areas by the CPU. A proper balance, and the use of the right allocation mechanism will be carefully studied in the next steps.

Streaming on FPGAs with OpenCL

In the first phase of the project activities we have studied the potential of programming using OpenCL on the Han Pilot Platform SoC board. We will briefly describe the board and we will show the specific challenges that we have preliminarily investigated.

The Han Pilot Platform is a SoC (System on Chip) board featuring and ARM multi-core CPU (dual-core ARM Cortex-A9 MPCore) and an Intel Arria10 FPGA (660K of low-power FPGA logic elements). The SoC board has been configured to use the OpenCL BSP (Board Support Package) provided by Terasic, and the full Intel Quartus Prime toolchain has been installed, with proper free academic licenses, on two server machines available at UNIPI.

The main goal of our activities is to provide powerful abstractions to use co-processors for streaming workloads. We selected OpenCL as the target framework for developing FPGA applications, because it provides a good balance between programmability (due to its higher-level API compared with traditional FPGA-based programming solutions like Verilog and HSL in general). Indeed, an OpenCL kernel is compiled by an offline compiler (called `aoc`), which generates the bitstream representation that configure the FPGA device for doing such kind of specific computational task. In this translation process, several sophisticated optimizations and heuristics are executed by the compiler to optimize the space and the resources available on the FPGA.

OpenCL allow programmers to write kernels that can be executed on different kinds of devices. The idea of a streaming library using FPGAs is to allow programmers to develop streaming DAGs (Directed Acyclic Graphs) of operators performing stateless and stateful streaming transformations like filtering, aggregation and joins. The general idea is to allow the programmer to execute a portion (usually a time-consuming one with strict latency constraints) on the FPGA devices, leaving other activities on the ARM multicore for receiving inputs from external sources, and for collecting outputs before transmitting them outside (see Figure 8 for a graphical representation of this idea). Each operator mapped on the FPGA can be implemented as:

- **Single work-item OpenCL kernels**: such kind of kernels are often called (tasks) in the OpenCL jargon. A kernel of this kind wraps a traditional sequential code in C (with specific constraints). The offline compiler will automatically pipeline and unroll the loops in the kernel code to extract parallelism between independent iterations. Such kind of automatic parallelism is called *pipeline parallelism*. Kernels of this kind are particularly suitable for streaming tasks, since they do not require having a large amount of data available in advance, but the processing can be done on a per-input basis (true streaming).
- **NDRange OpenCL kernels**: such kind of kernels represents the most common way of programming using OpenCL. Their goal is to break up the problem in finer components each one executed by an individual thread. Threads are called work-items, and they are grouped into work-groups. Parallelism is expressed by the work-items, which work in parallel on different pieces of the original input and can synchronize with other work-items in the same work-group. This parallelism model is called *data parallelism*. The offline compiler translates the code NDRagne kernels on the FPGA in a different way, without unrolling loops. So, it is of special importance to break the original problem in a large set of elements to be run by different work-items, and to balance their workload as much as possible.

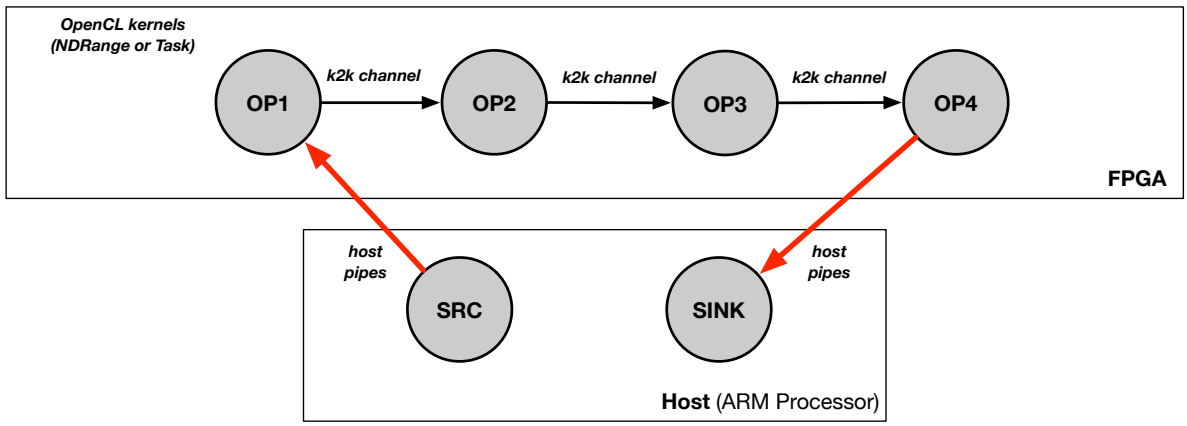

Figure 8 - Streaming on a SoC composed of an ARM processor and an Intel FPGA

The two different kinds of kernel leave space for different ways of implementing a streaming pipeline on the FPGA. The use of NDRange kernels for implementing operators enables an approach symmetric with what we plan to do with integrated GPUs and CUDA, where data parallelism is the only viable model for accelerating streaming tasks. Instead, the use of work-item kernels allows a finer parallelization, based on pipeline parallelism both inside the kernels (owing to the offline compiler translation of the kernel source code), and between kernels, which will remain active and processing different inputs in a pipeline fashion.

Another point to mention, it the OpenCL memory model and how it maps to the underlying FPGA hardware owing to the decisions taken by the offline compiler. OpenCL provides a high-level memory model whose implementation in terms of hardware resources can be different on the specific devices:

- **Host Memory**: it is a memory accessible only by the host CPU. Data are usually in this memory before being transferred to a FPGA-accessible memory.
- **Global Memory**: it is a memory accessible by both the host CPU and the FPGA device. This memory is allocated/deallocated by host threads. Memory areas allocated in the global memory should be accessed linearly by the FPGA to obtain good access latency.
- **Constant Global Memory**: it is a region of the global memory accessible in read/write mode by the Host CPU and in read-only mode by the FPGA compute units. Differently from the global memory, kernels on the GPU can load data in this kind of memory in a sort of internal cache accessible by the FPGA compute units.
- **Local Memory**: it is a memory accessible by all the work-items in the same work-group.
- **Private Memory**: it is a memory private of an individual work-item within a kernel.

On the Intel Arria10 FPGA, these logical memory entities are implemented as follows. The host memory is implemented on one or more memory banks (implemented by dedicated chips) accessible only by the host CPU. The global memory is implemented using DDR/QDR chips available on the board, while the local memory is implemented using blockRAM units presented on the FPGA. Finally, private memory is directly implemented using the registers available on the FPGA.

The other important aspect of the design is the way in which we plan to implement the data exchanging mechanism between operators mapped onto the FPGA. One important feature provided by OpenCL for Intel FPGA is represented by *Intel Channels*, which are a superset of the traditional OpenCL pipes provided by Intel FPGA vendor extension. There are three kinds of channels:

- **I/O channels**: they are used to allow data accesses from and interface of the FPGA (e.g., directly from a 10 Gb/s Ethernet port to the FPGA without passing from the host CPU).
- **Host pipes**: they allow the host CPU to send data accessible to the FPGA without passing from the global memory. They are implemented as a pinned-memory in the host and a FIFO buffer implemented on the blockRAM units on the FPGA. The implementation will stream blocks of data written by the host on the pinned-memory directly to the buffer on the FPGA device, allowing a fast and streamed way to efficiently provide inputs to the board. The same approach is used for pipelining data from the FPGA to the host.
- **Kernel-to-Kernel channels**: they are channel used by kernels on the FPGA to efficiently exchange data values. They are implemented as FIFO buffers by the offline compiler.

These facilities allow the investigation of at least two different modes for implementing operator pipelining on FPGA:

- **By-value passing**: where kernel-to-kernel channels are used to pass input values from one kernel to the next one. This is a very efficient way of implementing data forwarding between operators, done in few clock cycles on the FPGA. However, this approach has limitations related to the size of the exchanged messages that should be studied.
- **By-reference passing**: we plan to study an approach where channels are used to pass pointers to data (usually batches in that case) in global memory, which will be accessed by the next operator kernel. This approach puts a heavier load on the global memory but theoretically does not suffer from the limitations related to the size of the exchanged data items, since they are in global memory and kernel can use the channels to pass a capability for accessing them.

5.4 Performance measurement tools in a CPSoS context

For a long time, performance monitoring and profiling tools helped the HPC programmers with debugging their systems, optimizing their applications, or identifying bottlenecks. A wide variety of generic tools exists for non-time-critical systems[73], such as gprof, valgrind, or atom. These tools rely on either OS features such as multi-threading, interrupts or timers, or either on pseudo-automatic code instrumentation to collect the required timing information.

However, in **real-time systems**, such features are either not available (with enforced static scheduling), restricted or prohibited due to their impacts on time determinism. This is especially true for safety critical software that is constrained by drastic limitations due to the safety standards[74] [75] [76].

[73] Survey of Software Monitoring and Profiling Tools. B Wun. 2006

[74] Functional safety and IEC 61508 – A basic guide. International Electrotechnical Commission (IEC), Geneva, Switzerland, Nov 2002

[75] ISO 26262: Road Vehicles – Functional Safety. International Organization for Standardization (ISO), 2011

[76] DO-297: Software, Electronic, Integrated Modular Avionics (IMA): Development Guidance and Certification. Radio Technical Commission for Aeronautics (RTCA) and EURopean Organisation for Civil Aviation Equipment (EUROCAE). 1992

5.4.1 Profiling safety-critical systems

Section 5.2.1 of Deliverable D1.1 has already presented the Performance Monitor Counters (PMC) as a way to collect both timing information as well as details on the hardware resource usage, making them suitable to be exploited in Health & Usage Monitoring Systems (HUMS).

Their challenges can be summarized a. providing a way to 1) perform an accurate real-time runtime and resource usage measurement, 2) with a negligible impact on timing behaviour, 3) running outside of the operating system (avoiding system calls) to be able to profile both the OS and the running applications.

A potential solution is **METrICS**[77]: a Measurement Environment for Multi-Core Time Critical Systems that is running on top of the PikeOS[78] RTOS from SYSGO. This framework proposes accurate runtime and resource usage measurement while having a negligible impact on timing behaviour.

5.4.2 METrICS architecture

METrICS consists of several core components appearing in green in Figure 9. On the left side, we present the components actually running on the target hardware board, and on the right side the METrICS server, running on a Linux host, and in charge of driving the experimental campaign to be run on the board and collect all the gathered profiling information.

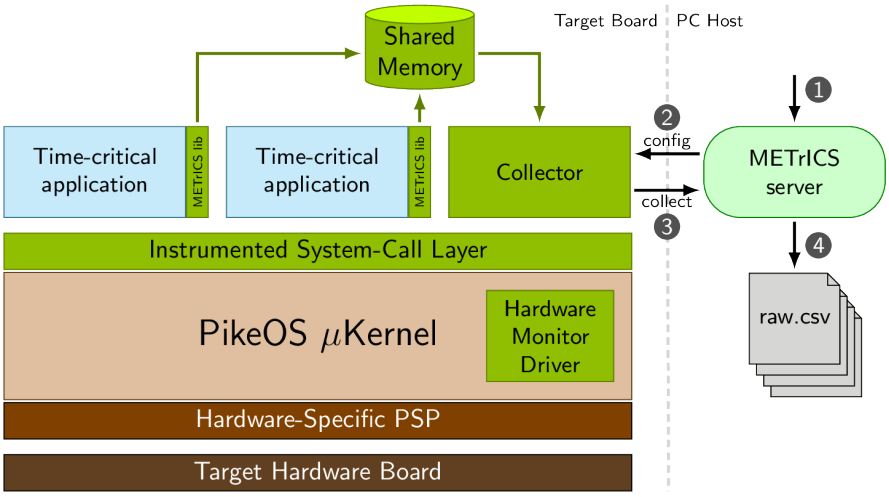

Figure 9: Architecture of the METrICS measurement tool

- The **METrICS library** is meant to be linked with the target applications to provide them with an access to the measurement probes API, allowing the collection of time and resource access information.
- The **syscall instrumentation layer** provides a way to automatically instrument each APEX system calls for ARINC-653[79] avionic applications.

[77] METrICS: A Measurement Environment for Multi-Core Time Critical Systems. S. Girbal, J. Le Rhun, H. Saoud. Embedded Real Time Software and Systems (ERTS 2018). Jan 2018. Best Paper Award

[78] PikeOS 4.2: RTOS with Hypervisor-Functionality, White paper from SYSGO AG, March 2017

[79] ARINC specification 653-2, “Avionics Application Software Standard Interface”. ARINC. December 2005

- The **Hardware Monitor kernel driver** provides the supervisor-level privilege necessary to access to hardware performance monitor counters (PMC). Such counters[80] allow us to count some hardware events, including the accesses to some shared hardware resource.
- The **collector partition** is in charge of 1) defining a shared memory space to collect measurements; 2) configuring specific measurement scenarios; 3) transferring the collected profiling information to the Linux host.
- Finally, the **METrICS server** running on the Linux host. It drives the experimental campaign and gather the collected profiling information.

5.4.3 METrICS intrusiveness

A major challenge in profiling tools is its intrusiveness in the system it monitors. METrICS distinguish **execution time intrusiveness** and **code intrusiveness**. The former limits the accuracy of the measurement due to the monitoring overhead, whereas the latter requires an effort from the developer to instrument the code of the application, which could be an issue for legacy software. METrICS focuses into limiting **execution time intrusiveness**, to have a minimal impact on the timing interference phenomenon.

A time intrusiveness had been performed of a full METrICS probe consisting of: 1) retrieving the timing information thanks to the core-dedicated special registers; 2) retrieving the performance monitor counters, again through direct register access; 3) retrieving thread-specific information from the OS; and 4) storing the collected information into the shared memory. The results are presented in Figure 10.

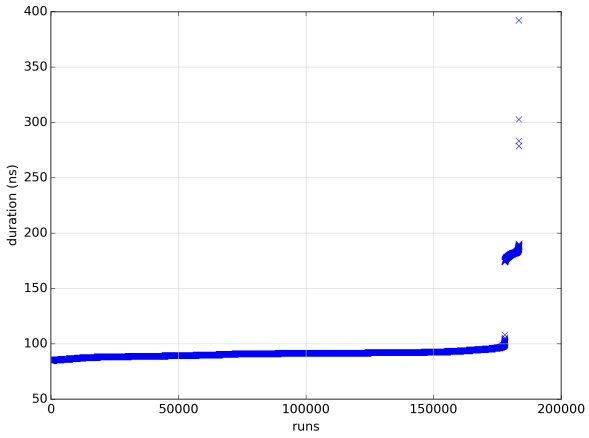

Figure 10: Completion time of a METrICS probe over 180000 runs

Over 180K runs, the probe time varies from 85ns up to 392ns. For 97% of the runs the overhead is below 110ns, and the overhead is above 191ns for only 0.002% of the cases. In comparison, the corresponding RTOS system call to only obtain current time (p4_get_time for PikeOS) requires 240ns, and it only get the current time and no PMC information. This is due to the fact that a system call involves at least two context switches, and possibly some privilege level changes. Therefore, the low intrusiveness of the overall METrICS probe makes it viable even for characterizing few micro-second long system calls of the OS.

[80] The basics of performance-monitoring hardware. B. Sprunt. Micro, IEEE, 22(4):64–71, 2002

5.4.4 Profiling Design Space

Target ARM or PowerPC embedded hardware usually provide a selection of about 250 hardware events that could be measured with performance monitor registers. Among these events around 50 of them are actually related to shared hardware resource to be profiled.

However, this architecture only provides a limited number of performance monitor registers (from 4 to 6), only allowing to concurrently profile a few hardware resources. As a consequence, a large number of runs are necessary to systematically study correlations between performance monitor counters and observed runtime (around C_{50}^{6} runs).

To perform such a large number of experiments some form of automation is necessary, and driven by the METrICS server on the Linux host. The different steps of the automated profiling process are appearing in Figure 9, with 1) the selection of target executable and test configuration, 2) the configuration of hardware counters to use, 3) the collection of measurements and, 4) the storage of result trace files.

Such experimental campaigns generate a rather large amount of raw data, making the direct statistical analysis quite difficult.

5.4.5 METrICS in the TEACHING project

Whereas METrICS has only be used so far for hardware and software characterization using statistical analysis techniques, the TEACHING project is an opportunity to couple METrICS and AI systems to learn the nominal behaviour of applications with regards to the hardware usage, allowing us to detect safety or security issues as deviation towards this nominal behaviour.

Some effort will be dedicated to port METrICS to the specific project hardware board and operating system. The hardware monitor driver being ported to a classical Linux driver to fulfil the correct privilege requirements to configure the PMCs, and the collector partition being tuned into a Linux process that will collect the traces and, during the learning phase, feed them to a potentially external AI systems to perform the learning of a “nominal behaviour”. During the operational phase, the real time trace will feed the embedded AI system that should perform the corresponding inference to detect safety / security anomalies.

5.5 Efficient processing and management of data streams

Several *Stream Processing Systems* (SPS) exist and have popularity in the research community. Some of them belong to the Apache umbrella and are well-established framework for stream processing on heterogeneous machines. In particular, they target physical clusters as well as traditional cloud environments with specific run-time mechanisms for scheduling and fault-tolerance designed traditional infrastructured distributed-memory architectures (e.g., clusters). During the first part of the project activities, we have practiced with two of them: *Apache Storm* and *Apache Flink*. In order to give a first general evaluation of these two tools, we will describe their programming model and runtime in the next subsections, furthermore, we provide a first preliminary evaluation of their performance with some final considerations about the need of SPSs oriented for being executed on edge/embedded resources like the one in the TEACHING environment.

5.5.1 Apache Storm

Apache Storm is a distributed stream processing computation framework designed to process unbounded streams of tuples (structured records of attributes). It has been acquired, developed then eventually open sourced by Twitter[81]. From the programming model perspective, it adopts a compositional API where the developer builds a graph of interconnected streaming transformations called *topology*. Supported transformations are of two types: *spouts* are the data sources, they take no inputs and generate one or more outputs; *bolts* process the incoming tuples and pass them to the next set of downstream bolts or simply end the computation, thus behaving as sinks. Data flows from producer spouts/bolts to consumer bolts using one of the several types of *partitioning strategies* at disposal, such as: random partitioning of tuples (*shuffle grouping*), partitioning based on a subset of the tuple fields (*fields grouping*), broadcasting to all the consumer instances (*all grouping*), and others. The logic to interconnect spouts and bolts and to choose the right partitioning policies is left to the programmer. Developers are in charge of implementing the business logic of the components usually by specializing several base classes (the API is available in Java/Scala) provided by the framework, then they use the TopologyBuilder class to build the actual topology.

Storm usually runs on a distributed cluster and clients submit topologies to a master node, which is called the *Nimbus*. Nimbus is responsible for distributing and coordinating the execution of the topology over the cluster machines. The actual work is done by so-called *worker nodes*. Each worker node runs one or more worker processes (each is a separated JVM). Each worker process executes spouts/bolts of a single topology by using separated threads called Executors. Each executor can run one or more replicas (tasks) of the same spout/bolt in the topology. Storm adopts this additional logical parallel concept of tasks (in addition to executor threads) to give more flexibility to application developers. Indeed, the number of executor threads can be changed at runtime while the number of tasks is fixed, to give the opportunity to scale the topology without having to shut the system down.

From the infrastructural viewpoint, each worker node run a supervisor daemon interacting with Nimbus (it is a master-worker architecture). The cluster state is maintained by Zookeeper (an open-source distributed key-value store), and Nimbus is responsible for scheduling the topologies on the worker nodes and monitoring the progress of the tuples flowing through the topology.

As for all the distributed SPS, Storm provides some sort of delivery guarantees. To exploit them, some mechanisms must be explicitly used by the programmer during the development of topologies: anchoring and acking. The former happens when a new tuple is emitted by a spout, Storm generates a unique message identifier for it and starts keeping track of the not-yet-acked tuple, then when a downstream bolt has finished processing the incoming tuple, it acks the message using one particular API call to signal that, in case of failure, the work up to that message has been safely processed. Otherwise, after a certain timeout the tuple is emitted again by the spout (thus providing a so-called *at-least-one* semantics).

5.5.2 Apache Flink

Apache Flink is a SPS developed by the Apache Software Foundation providing both streaming and batch processing with a unified API. Flink provides binding for several languages in

[81] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, et al., "Storm @twitter," in Proceedings of the 2014 ACM SIGMOD international con- ference on Management of data, pp. 147–156, 2014

addition to Java, such as Python and SQL. In the description, we will focus on the streaming capabilities of the framework, since they are of interest for the project activities.

The programming model offered by Flink is based on data transformations exposed through a fluent interface which, opposed to Storm, is strongly typed and has a declarative style. Eventually, the declarative program is mapped to a dataflow graph and deployed to a Flink cluster or to a single node instance. Developers extend the classes provided by the framework to implement their solution, e.g., by implementing the business logic code of data transformations connected in pipeline. In the Flink jargon, a *task* represents a single logical operator which can be replicated in one or more subtasks which in turn are mapped directly onto threads executed by one or more JVMs (called TaskManagers) provided that there are enough *slots* available (slots are abstract entities that are used for sharing the resources of a JVM, like the heap memory). Indeed, the number of slots determines the maximum achievable parallelism, which in case of computationally intensive tasks should not exceed the number of CPU cores/threads. Furthermore, Flink can be configured to execute more subtasks (operator replicas) on the same thread by using the so-called *operator chaining* optimization.

Flink has a completely distributed architecture mainly targeting clusters of heterogeneous nodes. Its runtime consists of two types of processes[82]. *JobManagers* coordinate the distributed execution. They schedule tasks onto the resources and coordinate the fault-tolerance protocols provided by the SPS. JabManager act as master of a master-slave architecture, and can be replicated in multiple entities for reliability reasons. The second kind of processes are TaskManager. They are in charge of executing tasks (or more specifically, the subtasks) of a dataflow graph, and buffering and exchanging the data streams. TaskManager processes are distributed among the machines of the Flink cluster. Under the hood, these processes are actors of the Akka framework [X], which provides the backbone of the Flink communication system. In addition, the JobClient submits jobs to the JobManager and waits for the result.

Flink implements fault tolerance using a combination of stream replay and checkpointing[83]. A *checkpoint* is related to a specific point in each of the input streams along with the corresponding state for each operator. A streaming dataflow can be resumed from a checkpoint while maintaining consistency (*exactly-once* processing semantics) by restoring the state of the operators and replaying the events from the point of the checkpoint. The distributed snapshotting used by Flink is implemented using *stream barriers*[84]. These barriers are injected into the data stream and flow with the records as part of it. A barrier separates the records in the data stream into the set of records that goes into the current snapshot, and the records that go into the next snapshot. When an intermediate operator has received a barrier for snapshot *n* from all of its input streams, it emits a barrier message conveying the identifier *n* to all of its outgoing streams. Once a sink operator receives the barrier *n* from all of its input streams, it acknowledges that snapshot *n* to the checkpoint coordinator. After all sinks have acknowledged a snapshot, it is considered complete.

[82] "Flink: Distributed Runtime Environment." https://ci.apache.org/projects/flink/flink-docs-release-1.10/concepts/runtime.html

[83] "Flink: Dataflow Programming Model." https://ci.apache.org/projects/flink/flink-docs-release-1.10/concepts/programming-model.html

[84] "Flink: Data Streaming Fault Tolerance." https://ci.apache.org/projects/flink/flink-docs-release-1.10/internals/stream_checkpointing.html

5.5.3 WindFlow

Traditional SPSs are far from being efficient where a single machine is concerned. Furthermore, their run-time systems have not been designed by taking into account the features of a highly heterogeneous computing environment like the one of the TEACHING eco-system, featuring embedded devices with power-limited capabilities. Furthermore, such traditional tools do not have support to co-processors like GPUs and FPGAs.

For these reasons, our activities are focused on selecting and even implementing from scratch suitable streaming libraries for achieving the goals and challenges of the project. Although traditional SPSs will remain valuable as a baseline, we are investigating the use of streaming libraries developed internally by the UNIPI team, and in particular the WindFlow[85] C++17 library. This research library adopts a *skeletal* approach to stream processing, where a rich set of predefined operators is provided to the developer such as Map, Filter, FlatMap, Accumulators and window-based aggregators. Under the hood, WindFlow uses FastFlow[86], a C++ parallel programming framework that provides a number of *parallel building blocks* which are in turn built upon a low-level runtime support based on non-blocking multi-threading with lock-less synchronizations. From the API perspective, WindFlow provides a specific construct called *MultiPipe* used to compose together operators and to build complex streaming DAGs. Internally, a MultiPipe is a composition of the FastFlow concurrent building blocks (combiners, pipes and all-to-alls) organized and nested according to formal semantics rules. As already stated by some preliminary research results, WindFlow is capable of providing significantly superior performance (higher throughput and lower processing latency) when the entire streaming DAG is run on a single multicore-based machine, where traditional SPSs exhibit instead too much overhead. For this reason, after the preliminary analysis developed in the first step of the T2.3 activities, WindFlow has been selected as a promising library to be extended for the project's purposes, thus providing support to embedded GPUs and FPGAs to perform localized streaming pipelines in the embedded resources available in the TEACHING use cases.

5.6 Communication paradigms for IoT sensors and wearable devices

In this section, we survey those Machine-to-Machine (M2M) communication paradigms and protocols for the Internet of Things (IoT) that are candidate to be exploited to support network communications of HPC^{2}I. We examine the two most diffused paradigm, i.e., request/response and publish/subscribe, and discuss the performance trade-offs in existing designs applied in the context of node-centric and data-centric network architectures. This survey contributes to compare the different solutions to derive possible outcomes on where one is more suitable in place of the other one and vice-versa.

A point worth stressing is that data transfer patterns in the M2M-driven Internet of Things will differ fundamentally from those in the classic "human-to-human" (H2H) internet. M2M communications will feature orders of magnitude more nodes than H2H, most of which create low-bandwidth, upload-biased traffic. Many M2M applications need to deliver and process information in real time, or near- real-time, and many nodes have to be extremely low-power

[85] G. Mencagli, M. Torquati, D. Griebler, M. Danelutto and L. G. Fernandes. Raising the Parallel Abstraction Level for Streaming Analytics Applications. IEEE Access, 2019, IEEE. ISSN: 2169-3536, DOI: 10.1109/ACCESS.2019.2941183

[86] M. Aldinucci, M. Danelutto, P. Kilpatrick, and M. Torquati, Fast- Flow: High-Level and Efficient Streaming on Multicore. John Wiley & Sons, Ltd, 2017, ch. 13, pp. 261–280. [Online]. Available: https: //onlinelibrary.wiley.com/doi/abs/10.1002/9781119332015.ch13

or self-powered (e.g., solar powered) devices. M2M and IoT share common features, which may be mandatory or not, which are:

a) Things: The "things" in the IoT like the "machines" in M2M, are physical entities whose identity, state (or the state of whose surroundings) is capable of being relayed to an internet-connected IT infrastructure. Almost anything to which you can attach a sensor can become a node in the Internet of Things.

b) Sensors: These are the components of "things" that gather and/or disseminate data, being on location, altitude, velocity, temperature, illumination, motion, power, humidity, blood sugar, etc. These devices are rarely "computers" as we generally understand them, although they may contain many or all of the same elements (processor, memory, storage, inputs and outputs, OS, software). The key point is that they are increasingly cheap, plentiful and can communicate either each others as it was in M2M communication platforms, or directly with the Internet or with Internet-connected devices.

c) Local-area Comms: All IoT sensors require some means of relaying data to the outside world. There's a plethora of short-range, body area, personal area, or local area wireless technologies available, including RFID, NFC, Wi-Fi, 6LoW- PAN, Bluetooth (including Bluetooth Low Energy), XBee, Zigbee, Z-Wave, and Wireless M-Bus.

d) Wide-area Comms: For long range, or wide-area, links there are existing mobile networks (using GSM, GPRS, 3G, LTE or WiMAX, for example) and satellite connections. New wireless networks such as the ultra-narrowband SIGFOX, LoRa, and NB-IoT are also emerging to cater specifically for M2M connectivity. Fixed 'things' in convenient locations could use wired Ethernet or phone lines for wide-area connections.

e) Servers, Brokers, Proxies: Some types of M2M installations, such as a smart home or office, use a local server to collect and analyze data both in real time and episodically from assets on the local area network. These on-premise servers or simpler gateways (right) usually connect also to cloud-based storage and services. According to the communication architectures, there might be intermediate servers, called brokers, which manage the distribution and forwarding of messages toward other nodes, or proxies, which relay local non-IP networks with Internet IP networks.

f) Storage and analytics: Today's internet generates a lot of data, but IoT is entirely another matter. That will require massive, scalable, storage and processing capacity, which almost invariably resides in the cloud, except for specific localized or security-sensitive cases. Service providers obviously have access here, not only to curate the data and tweak the analytics, but also for line-of-business processes such as customer relations, billing, technical support and so on.

g) User-facing services: Subsets of the data and analyses from the IoT will be available to users or subscribers, presented via easily accessible and navigable web interfaces on a full spectrum of secure client devices.

5.6.1 Node-centric communication paradigms

The client-server model is a structure that divides, in a clear way, the producers (or providers) of a service from those who are requesting it, called clients. In this paradigm, a client initiates a connection to the server, makes a request and obtains the data from the server. The clearest example of this schema is the Web itself: a server has certain information and makes it available to clients, that will connect to the server and get the information, when needed.

Such interaction schema relies on the request-response pattern: a resource is offered to a consumer when requested, not when available. Therefore, it is a client responsibility to poll for a certain resource at a given rate if it wants to be up-to-date. Again, the Web itself is a clear example of this architecture: the great majority of the available resources is available on request, and not pushed to consumers (in a solicited or unsolicited way). A set of clients connect to a set of servers to obtain the data they need.

5.6.1.1 WebSocket

The WebSocket protocol provides two-way communication over a single TCP channel. It is an IETF standard since 2011, the RFC 6455[87]. WebSocket was born to be implemented in web browsers and servers, but it can be also used for typical client-server applications, thus not relying on a web architecture. It has no relationship to the HTTP protocol, except for the handshake, that is interpreted as an Upgrade Request[88] with value containing the word *websocket*. Several browsers support this protocol nowadays.

Before WebSocket, the creation of web applications that need bidirectional communication had have the counter-effect of abusing the HTTP protocol, by using a large number of HTTP connections to poll the server for updates. A simple TCP connection is the solution, according to the design idea of the WebSocket protocol. A client opens a WebSocket based communication to a server, then they can communicate on this TCP-based channel. The WebSocket protocol provides a way to send unsolicited contents to a client, by using a TCP connection with port numbers 80/443.

Conceptually, WebSocket is just a layer on top of TCP, indeed a very simple protocol. Its simplicity is one of the main features, because the overhead is reduced (only 2 bytes per transmitted request / response[89]). Trying to summarize its features, you can list the following:

- bidirectional: there are no predefined message patterns such as request / response. The client and the server can send messages to the other part.
- full-duplex: client and server can talk in an independent way of each other; there is no such pattern as in HTTP, where only a part is talking at a time.
- single TCP connection: the same TCP connection is used by a WebSocket for its whole lifecycle, thus avoiding to open and close a TCP connection for each request.
- minimal overhead: after the initial handshake, the data is minimally framed with 2 bytes.

An unprotected WebSocket server and resource name is identified by the *ws* URI scheme name, whilst a TLS protected WebSocket server and resource name is identified by the *wss* URI scheme name.

5.6.1.2 REST - Representational state transfer

Representational state transfer (REST) is an architectural style consisting of a coordinated set of architectural constraints applied to components, connectors, and data elements[90]. The REST-style architecture consists of clients and servers. An important concept in REST is the existence

[87] http://tools.ietf.org/html/rfc6455

[88] http://tools.ietf.org/html/rfc6455

[89] http://tools.ietf.org/html/rfc6455

[90] R. Fielding and R. Taylor, "Principled design of the modern web architecture," in *Software Engineering, 2000. Proceedings of the 2000 International Conference on*, 2000, pp. 407–416.

of resources (e.g., sensors sources of specific information), each of which is referenced with a global identifier (e.g., a Uniform Resource Identifier URI in HTTP[91]). In order to manipulate these resources, components of the network (user agents and origin servers) communicate via a standardized interface (e.g., HTTP) and exchange representations of these resources (the actual documents conveying the information). REST was initially described in the context of [2], which already provides a rich and uniform vocabulary for applications based on the transfer of meaningful representational state. REST applications maximize the use of the existing, well-defined interface and other built-in capabilities, provided by the chosen network protocol, and minimize the addition of new application-specific features on top of it. Clients initiate requests to servers; servers process requests and return the appropriate replies. Such a request-response (Req-Res) paradigm accesses and manipulates the resources. This paradigm is also historically known in computer science as remote procedure call (RPC) that allows a computer program (the client) to cause a procedure to be executed in another address space (the server) by sending a request message; the remote server sends a response with the execution to the client, and the application continues its process.

In REST, a resource can be anything that can be identified by URIs (e.g., documents, images, and files). REST uses the GET, PUT, POST, and DELETE operations of HTTP to access those resources. However, the protocols used for RESTful architecture are not appropriate for resource constrained networks and devices[92]. A large overhead of HTTP implies packet fragmentation and performance degradation relatively to M2M devices. Also, TCP flow control, which is used at transport layer by HTTP, is not appropriate for M2M devices and the overhead is too high for short transactions. To extend the REST architecture for resource constrained M2M devices, constrained application protocol (CoAP)[93] has been defined. CoAP is an application protocol intended to be used in simple devices, allowing them to communicate over the Internet. CoAP includes a subset of the HTTP functionalities, optimized for M2M applications. It also supports multicast, very low overhead, and asynchronous message exchanges over a user datagram protocol (UDP) for M2M devices.

REST is a software architectural style, focused on the roles of its parts, instead of the parts themselves. The parts are:

- components: abstract software unit accessible via interface
- connectors: abstract communication medium between the components
- data: information exchanged between components by using the connectors

As software architecture, several formal constraints can be identified:

- client-server communication style: the providers and the consumers of the services are separated and independent and the communication is possible by using an interface.
- stateless communication: servers do not maintain any information related to the clients; each request is self-contained. A state information can be maintained and used to exploit authentication and authorization mechanisms within a session, but it expires in a given amount of time.

[91] T. B.-L. R. Fielding; J. Gettys; J. Mogul; H. Frystyk; L. Masinter; P. Leach, "Hypertext transfer protocol – http/1.1, request for comments: 2616," Network Working Group, Tech. Rep., June 1999

[92] W. Colitti, K. Steenhaut, N. De Caro, B. Buta, and V. Dobrota, "Rest enabled wireless sensor networks for seamless integration with web applications," in *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*, Oct 2011, pp. 867–872

[93] Z. S. K. H. C. Bormann, "Constrained application protocol (coap)," CoRE Working Group Internet-Draft, Tech. Rep., December 2013

- cache mechanism: each response from a server / producer must define itself as cacheable or not, to improve scalability and performance of the whole architecture.

5.6.1.3 WAMP - Web Application Messaging Protocol

The WebSocket Protocol is already built into modern browsers and provides bidirectional, low-latency message-based communication. However, as such, WebSocket is quite low-level and only provides raw messaging. Modern Web applications often have a need for higher level messaging patterns. This is where the WAMP (Web Application Messaging Protocol) protocol enters. WAMP adds the higher-level messaging patterns of RPC (Remote Procedure Call) and Publish/Subscribe to WebSocket - within one protocol. Technically, WAMP is an officially registered WebSocket subprotocol (runs on top of WebSocket) that uses JSON as message serialization format.

Two interaction styles are possible:

- RPC-style: three types of peer are possible: Caller, Dealer, and Callee. A Callee registers procedures with an application code to call remotely from Callers under application defined, unique names. A Dealer performs the routing of calls and results between Callers and Callees.
- Publish/Subscribe style: it is a messaging pattern involving peers of three roles: Publisher, Broker and Subscriber. A Subscriber subscribes to topics under application defined, unique names to receive events published by Publishers to such topics. A Broker performs the routing of events from Publishers to Subscribers.

A WAMP environment is called Realm, a routing and administrative domain protected by authentication and authorization. Two peers in a Realm can open a transient conversation, called Session, that runs over a Transport. The default transport is the WebSocket protocol, but also HTTP 1.0/1.1 long polling, Batched and Multiplexed Transport are possible. In the Batched schema, more than one WAMP message is transmitted per WebSocket message, while in the Multiplexed schema a single physical connection is shared between multiple logical Transport channels.

5.6.2 Data-centric communication infrastructure

For large-scale distributed systems, Data centric communication based on the publish/subscribe paradigm plays a key role on traffic volume control. More data filtering efficient and adaptive to different traffic conditions bring some solutions at the high layers of the communication model. In the context of the TEACHING project we are considering such approaches as good candidates for the development of the communication channels supported by HPC^{2}I.

The remaining of this section briefly introduces the publish/subscribe paradigm, a few approaches based on it and a potential exploitation of such solutions in TEACHING. In particular, Section 5.6.2.1 introduce the publish/subscribe paradigm. Sections 5.6.2.2, 5.6.2.3, 5.6.2.4, 5.6.2.5, 5.6.2.6 and 5.6.2.7 reports some technologies and tools that are candidate to be exploited as baseline on which to build the advanced solutions that TEACHING HPC2I is aimed at providing.

5.6.2.1 The Publish/Subscribe Paradigm

The publish/subscribe pattern is a loosely coupled, many-to-many *asynchronous* paradigm, which distinguish the entities who provide a service from those who are interested on that service. Each of these types of entity are not required to exist at the same time. The nodes

belonging to these networks are more interested in the information than the identity of the information producers. The data is therefore delivered in function of the node's interests. This paradigm is inherently anonymous, in that the communication partners are not required to identify the party they want to talk to. For example, instead of naming a publisher to receive events from, the subscriber simply describes the characteristics of the events it wants to receive[94]. Publish/subscribe messaging systems are widely used in enterprise networks, mainly because of their scalability and support of a dynamic application topology. These features are achieved by decoupling the various communicating components from each other such that it is easy to add new data sources/consumers or to replace existing modules.

When a node wants to register its interest into a certain information, it registers itself through an operation called *subscription*. That node becomes a *subscriber*. The nodes who instead produce information are called *publishers*. The entity in charge of registering the subscription requests and guaranteeing that data is delivered from the publishers to the subscribers is called *broker* (Figure 11). Several protocols based on this paradigm exist, aiming at vast application scenarios, from M2M to enterprise level function.

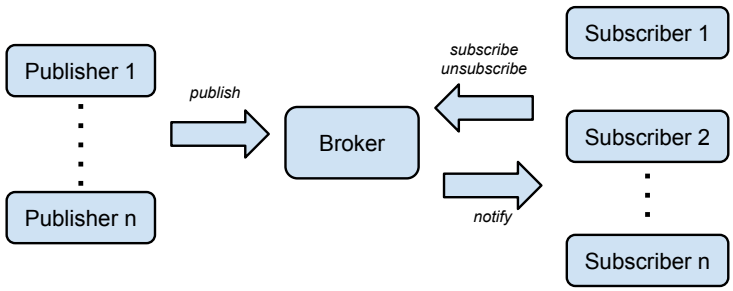

Figure 11 The publisher/subscriber paradigm.

5.6.2.2 Message Queuing Telemetry Transport

MQTT is an extremely lightweight publish/subscribe messaging transport protocol targeted for telemetry applications on constrained de- vices based on TCP/IP. The protocol subscriptions are based on *topics*, which are hierarchically organized strings. A message can be flagged as *retained*, which means the broker stores and sends that message to all the future clients which subscribe to the given topic. The protocol also supports different types of end-to-end QoS, depending on the application needs:

- QoS 0: Best effort, no guarantee of delivery
- QoS 1: At least one delivery is guaranteed but more copies of the message can be received
- QoS 2: Only one delivery is guaranteed and unique for every message.

MQTT is completely unaware about the content of the payload and it is characterized by a very small transport overhead in order to reduce the network traffic as much as possible. MQTT also implements a mechanism to notify interested parties about abnormal client disconnections implementing a *last will* feature. This mechanism allows a client to communicate to the broker a last will message it wants to be sent on behalf of the client in case it experiences an unexpected

[94] Y. Huang and H. Garcia-Molina, "Publish/subscribe in a mobile environment," *Wireless Networks*, vol. 10, no. 6, pp. 643–652, 2004

communication error. Implementations of MQTT broker are Mosquitto, HiveMQ and CloudMQTT.

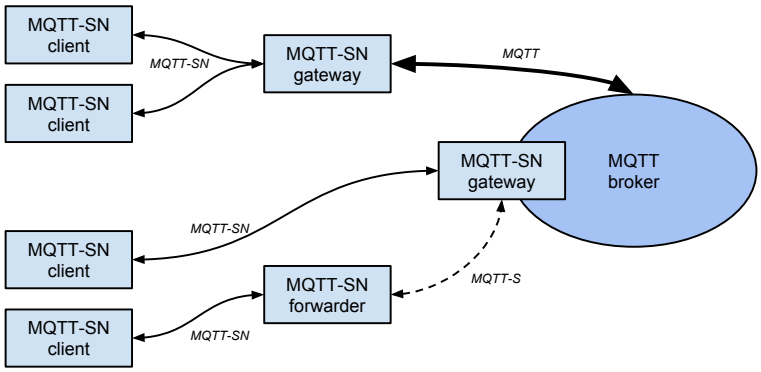

Figure 12 MQTT-SN architecture.

5.6.2.3 Message Queuing Telemetry Transport - Sensor Network

MQTT-SN is a publisher/subscriber protocol based on MQTT and targeted for Wireless Sensor Networks (WSNs). Extending the enterprise publisher/subscriber system into the WSNs also enables a seamless integration of the WSNs into the enterprise network. This makes the field data collected by the SAs (Sensors and Actuators) available to all applications as any other enterprise information and enables the control of the SAs from any enterprise application. MQTT-SN is designed aiming to an optimized implementation on low-cost, battery-operated devices with limited processing and storage resources. MQTT-SN is designed in such a way that it is unaware of the underlying networking services. Any network which provides a bi-directional data transfer service between any node and a particular one (a gateway) should be able to support MQTT- SN (Figure 12). There are three kinds of MQTT-SN components: *clients*, *gateways* and *forwarders*.

MQTT-SN *clients* connect themselves to a MQTT server via a MQTT-SN *gateway* using the MQTT-SN protocol. A MQTT-SN *gateway* may or may not be integrated with a MQTT server. In case of a stand- alone *gateway*, the MQTT protocol is used between the MQTT server and the MQTT-SN *gateway*. Its main function is the translation between MQTT and MQTT-SN. MQTT-SN *clients* can also access a *gateway* via a *forwarder* in case the *gateway* is not directly attached to their network. The forwarder simply encapsulates the MQTT-SN frames it receives on the sensor network side and forwards them unchanged to the *gateway*; in the opposite direction, it decapsulates the frames it receives from the gateway and sends them to the clients, unchanged too[95]. There are two types of MQTT-SN gateway:

- *transparent*: for each connected MQTT-S client it sets up and maintains and independent MQTT connection to the broker
- *aggregating*: all the connections from the clients are sent to the broker over an unique MQTT connection. This increases the scalability of the network when having a large number of SAs.

[95] A. Stanford-Clark and H. L. Truong, “Mqtt for sensor networks (mqtt-sn) protocol specification,” 2013

5.6.2.4 Constrained Application Protocol

In addition to request-response model, CoAP also supports publish/subscribe architecture using an extended GET method. Unlike MQTT, the publish-subscribe model of CoAP uses Universal Resource Identifier (URI) instead of topics. This means that subscribers will subscribe to a particular resource indicated by the URI *U*. When a node publishes data *D* to the URI *U*, then all the subscribers are notified about the new value as indicated in *D*. Since CoAP runs on top of the inherently not reliable UDP, it provides its own reliability mechanism through the use of *confirmable* and *non-confirmable* messages[96]. The observers register with the subject using the GET request with a special observe option activated. The subject puts the observer, if it is allowed, in the list of the interested entities and responds to the observer with an immediate state of the resource. After the initial response each subsequent notification is an additional CoAP response sent by the server in reply to the GET request and includes a complete representation of the new resource state[97]. CoAP also enables high scalability and efficiency through a more complex architecture, which in fact supports the use of caches and intermediaries (proxy) nodes that multiplex the interest of multiple subscribers in the same event into a single association[98] (Figure 13).

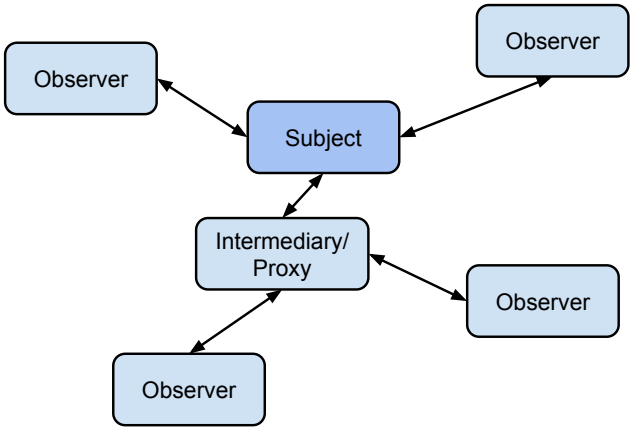

Figure 13 CoAP observer model architecture.

5.6.2.5 Extensible Messaging and Presence Protocol

XMPP is an open standard, real-time and asynchronous messaging system that delivers messages in a distributed network of presence-aware nodes. The basic protocol data unit in XMPP is an XML "stanza", which is essentially a fragment of XML. XMPP has a Publish-Subscribe functionality, specified as an extension in XEP-0060.

[96] D. Thangavel, X. Ma, A. Valera, H.-X. Tan, and C. K.-Y. Tan, "Performance evaluation of mqtt and coap via a common middleware," in *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on*. IEEE, 2014, pp. 1–6

[97] S. Bandyopadhyay and A. Bhattacharyya, "Lightweight internet protocols for web enablement of sensors using constrained gateway devices," in *Computing, Networking and Communications (ICNC), 2013 Interna- tional Conference on*. IEEE, 2013, pp. 334–340

[98] E. G. Davis, A. Calveras, and I. Demirkol, "Improving packet delivery performance of publish/subscribe protocols in wireless sensor networks," *Sensors*, vol. 13, no. 1, pp. 648–680, 2013

The protocol enables XMPP entities to create nodes (topics) at a pub/sub service and publish information at those nodes; an event notification (with or without payload) is then broadcasted to all entities that have subscribed to the node[99].

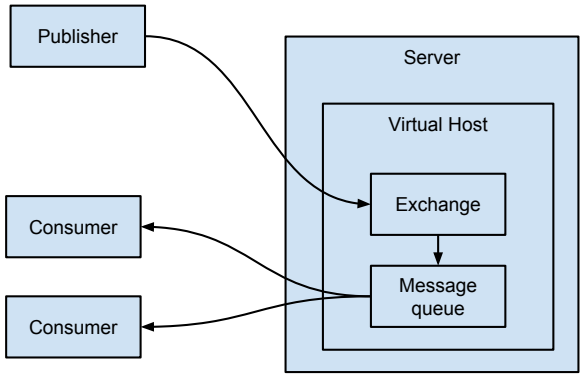

Figure 14 AMQP model

5.6.2.6 Advanced Message Queueing Protocol

The Advanced Message Queuing Protocol (AMQP) is another open standard publish/subscribe protocol that provides reliable queueing, flexible routing, and topic-based messaging. The protocol defines both the network wire-level protocol and the messaging model. Information is organized into *frames* that are sent over *channels*, which are independent threads within a single socket connection. Several brokers are available, including RabbitMQ, OpenAMQ, and Apache Qpid.

5.6.2.7 Apache KAFKA

Apache Kafka[100] is an open-source distributed event streaming platform targeting high-performance data pipelines, streaming analytics, data integration, and mission-critical applications. Kafka is a distributed system consisting of **servers** and **clients** that communicate via a TCP network protocol. It can be deployed on bare-metal hardware, virtual machines, and containers in on-premise as well as cloud environments.

Servers: Kafka is run as a cluster of one or more servers that can span multiple datacenters or cloud regions. Some of these servers form the storage layer, called the *brokers*.

Clients: distributed applications and microservices that read, write, and process streams of events in parallel, at scale, and in a fault-tolerant manner even in the case of network problems or machine failures. Kafka ships with some such clients included. Clients are available for Java and Scala including the higher-level Kafka Streams library, for Go, Python, C/C++, and many other programming languages as well as REST APIs.

[99] http://xmpp.org/about-xmpp/technology-overview/pubsub

[100] https://kafka.apache.org/intro

Kafka Terminology

An event records the fact that something happened. Sometime is also called record or message. Each event has a key, a value, timestamp and optional metadata headers.

Producers are client applications that write events and consumers are those that subscribe events (eventually reading and processing them). Producers and consumers are fully decoupled. Kafka ensures exactly-one processing property for events.

Events are organized and stored in topics. Topics in Kafka are always multi-producer and multi-subscriber: a topic can have zero, one, or many producers that write events to it, as well as zero, one, or many consumers that subscribe to these events. Events are not deleted after consumption.

Topics are partitioned, meaning a topic is spread over a number of "buckets" located on different Kafka brokers. This distributed placement of data is important for scalability because it allows client applications to both read and write the data from/to many brokers at the same time.

When a new event is published to a topic, it is actually appended to one of the topic's partitions. Events with the same event key (e.g., a customer or vehicle ID) are written to the same partition, and Kafka guarantees that any consumer of a given topic-partition will always read that partition's events in exactly the same order as they were written.

To improve data fault-tolerance and highly-available, every topic can be replicated, even across geo-regions or datacenters, so that there are always multiple brokers that have a copy of the data just in case things go wrong. A common production setting is a replication factor of 3, i.e., there will always be three copies of a piece of data. This replication is performed at the level of topic-partitions.

5.6.3 Preliminary design of the communication infrastructure

A preliminary design of the communication infrastructure, based on a data-centric paradigm, is under evaluation. We envision use of the frameworks Apache Kafka[101] and Apache Zookeeper[102] to our use cases. We are focusing on its exploitation for the automotive use case, in particular when NearEdge resources (i.e., Road Side Units – RSUs in this case) are available.

Each Far Edge is endowed with a MQTT broker to manage locally the data produced by the automotive, environment and wearable sensors. The broker uses different topics to manage private data (kept on the Far Edge), sharable data (that can be shared with the remote layer), AI models. The local subscribers, on the Far Edge, can be services such as: the local updater of the AI model, a local store manager, a Kafka client.

Resources located in the remote layer, either being Near Edges (e.g., RSUs) or Clouds access and collect sharable data as well as disseminate Federated AI models. Resources standing at the remote layer are endowed with Kafka brokers, which are able to communicate with the clients installed on each Far Edge. The subscribers to the Kafka brokers are services that process the Federated Data and generated Federated AI Models.

[101] https://kafka.apache.org

[102] https://zookeeper.apache.org

The broker in a Near Edge can also publish the data produced by the services running in the Cloud, or in the Near Edge itself. So, Kafka clients can subscribe to the topic of the updated federated AI model to refresh their local AI models, running on the Far Edges.

Note that each Far Edge is allowed to post and subscribe to its own topics thus, unless a consensus is reached, no node should be able to operate on another Far Edge topic. Only the service that performs the model federation is allowed to publish on the topic of the AI model of all the Far Edges.

In this case, we can assume that the communication between client and broker can be achieved in an encrypted way, for example using TSL, with certificates issued by a private CA. These certificates could be uploaded on the Far Edge at the same time as the TEACHING application is installed. Based on the credentials produced for each node through the certificates, Kafka's Access Control Lists can be exploited to define which hosts can perform operations on a given topic or set of topics.

Far Edges moving within the geographical area on which the communication infrastructure is deployed may need to interact with topic replicated on the Near Edges. The task of synchronizing the brokers on the Near Edges can be either achieved using Zookeeper.

Zookeeper provides a centralized service for providing configuration information, naming, synchronization and group services over large clusters in distributed systems. The goal is to make these systems easier to manage with improved, more reliable propagation of changes. ZooKeeper provides an infrastructure for cross-node synchronization by maintaining status type information in memory on ZooKeeper servers. A ZooKeeper server maintains a copy of the state of the entire system and persists this information in local log files.

5.7 Communication mechanisms for mobile vehicular networks

Traffic incidents are persistent problems in both developed and developing countries, which result in a huge loss of life and property[103]. The majority of serious road accidents occurred due to rear-end crashes, side crashes within intersections, and lane changes on one-way highways. To overcome these problems, Intelligent Transportation Systems (ITS) introduces Vehicular Ad-hoc Networks; provides the protocols that allow a vehicle to communicate with another vehicle (V2V), with the roadside infrastructure (V2I) and also provides the Vehicle-to-Pedestrian (V2P) communication, via a hardware module (On-Board Units) that can be system built-in or can be connected with the wireless communication with the vehicle. In general, V2X (Vehicle to Everything) is an umbrella term for all the communications mentioned above[104]. Each vehicle outfitted with a VANET device will act as a node in the Ad-hoc network and will be able to receive and send messages to the other vehicles or Road Side Units through the wireless network[105]. To support vehicular communications Federal Communications Commission (FCC) reserved the band of 75 MHz around 5.9 GHz for Dedicated Short-Range

[103] Hossain, Ekram, et al. "Vehicular telematics over heterogeneous wireless networks: A survey." Computer Communications 33.7 (2010): 775-793

[104] T. Bey and G. Tewolde, "Evaluation of DSRC and LTE for V2X," 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 2019

[105] K. Mehta, L. G. Malik and P. Bajaj, "VANET: Challenges, Issues and Solutions," 2013 6th International Conference on Emerging Trends in Engineering and Technology, Nagpur, 2013, pp. 78-79

Communications (DSRC)[106]. To make reliable the message exchanging among vehicles or vehicle and Road Side Units, network diversity techniques can be adopted[107].

5.7.1 Dedicated Short Range Communications

DSRC has been the V2V standard and used for direct communications between moving vehicles in the DSRC frequency range and does not depends on the cellular infrastructure. Wireless channels under vehicular environments are complicated and difficult to estimate. Packets suffer from multipath fading and Doppler shift because of the fast-moving vehicles that make the conventional communications systems such as Wi-Fi, Bluetooth unfeasible for vehicular applications. So being able to connect vehicles is one of the major challenges that autonomous vehicles are facing. Indeed, making them exchange data on the air can enable important safety and comfort application, which would otherwise be impossible to implement, or lacking fundamental functionalities. Two approaches have been proposed as of now: the first one involves the use of cellular networks, together with the upcoming 5G technology, while the second one is currently using an evolution of the IEEE 802.11a amendment, IEEE 802.11p to provide wireless access in vehicles referred as Wireless Access in Vehicular Environment (WAVE). Specific enhancements for the vehicular use case were introduced, such as the possibility to prioritize the traffic and the "Outside context of BSS" mode, allowing vehicles to communicate directly with low latency.

Low transmission latency and packet drop rate (PDR) are two of the most important characteristics of vehicular communications and the major indexes in evaluating the performance of their associated technologies. DSRC permits low-latency around 2ms in transmitting basic safety messages between vehicle to vehicle (V2V) and vehicle to roadside infrastructure (V2I). However, one of the main challenges for DSRC technology is congestion. As the number of vehicles increases in one channel. Since having the number of vehicles near to 50 significantly decreases the percentage of message delivery with the increase in latency. However, if the number of vehicles on one channel is less than 50, still if their speed increases more than 40 km/h, the message delivery rate drops to about 60 percent with a slight impact on latency[108].

Because of the communication constraints related to DSRC, researchers are trying to seek alternatives or complementary vehicular communication technologies. In recent years, there has been an increasing interest in the use of cellular technologies of 4G/LTE (long-term evolution) as well as 5G for V2X, known as cellular-V2X (C-V2X). This technology was standardized by the 3rd generation partnership (3GPP)[109]. C-V2X stands for cellular Vehicle to Everything, and it utilizes cellular technology to provide the link between the vehicle and the rest of the world, including other vehicles, and roadside units such as traffic control systems. DSRC still has a firm hold in the automotive application yet some automakers believe there is more potential in C-V2X, especially with the promise of 5G in the near future. In parallel, cellular mobile networks are also evolving rapidly. Mobile broadband systems based on 3GPP

[106] Regan MA, Oxley J, Godley S, Tingvall C: Intelligent transport systems: safety and human factors issues, no. 01/01 (Royal Automobile Club of Victoria (RACV) Ltd., 2001),. Monash University, Australia, 2001

[107] S. Barre, C. Paasch, and O. Bonaventure, "Multipath TCP: from Theory to Practice," in International Conference on Research in Networking. Springer, 2011, pp. 444–457

[108] Z. H. Mir and F. Filali, "TE and IEEE 802.11 p for vehicular networking: a performance evaluation," EURASIP Journal on Wireless Communications

[109] B. Fall, S. Niar, A. Sassi and A. Rivenq, "Adaptation of LTE-Downlink Physical Layer to V2X and T2X communications," International Journal of Engineering and Innovative Technology (IJEIT), vol. 4, no. 10, pp. 182-192, 2015

and LTE standards are already deployed globally and provide mobile connectivity with high bandwidth and low latency.

Automated driving is being seen as a technological enabler shaping the future mobility concept and enhancing the quality of modern life by providing traffic safety together with added environmental and comfort improvements. Being able to connect vehicles is one of the major challenges that autonomous vehicles are facing. Indeed, making them exchange data on the air can enable important safety and comfort applications, which would be, otherwise, complex to implement, or lacking fundamental functionalities. As mentioned earlier V2X communications are based on two main technologies called Cellular-V2X (C-V2X); that utilizes cellular technology to provide the link between the vehicle and the rest of the world, including other vehicles and infrastructure while the second one is called "Dedicated Short-Range Communications" (DSRC), based on a Wi-Fi variant known as 802.11p, operating on 5.9 GHz band. DSRC standards are being already implemented in the USA and Europe but now, with the evolution of 5G, it is believed that Cellular technology may bring more advantages and could be a strong candidate for future low-latency and high-bandwidth V2X communications. For instance, pedestrians and cyclists usually carry smartphones, and their precise position may be more seamlessly transmitted to vehicles when a cellular V2X technology is involved. There is currently an on-going debate on which systems could be better for vehicular networks: either DSRC, based on "Wi-Fi", or C-V2X.

According to a recent 5G Automotive Association white paper, "5G high reliability, low latency features can be fully applied to V2X". 5G also aims to provide a high data rate for vehicular connections. It has also been shown[110] that DSRC performance is satisfactory for most safety critical applications that require the end-to-end latency to be around 100 msec as long as the density of vehicles is moderate but as the density increases the performance is decreasing while C-V2X offers performance advantages over DSRC in terms of its additional link budget, higher resilience to interference and better non-line-of-sight (NLOS) capabilities[111].

5.6.2 Network Diversity Techniques

In communication scenarios with mobile nodes, such as vehicular, a very useful tool to increase the robustness and/or reliability connection is the use of the network diversity techniques, i.e., the possibility of using multiple connections at the same time. In this case, the mobile user is able to communicate with another node or communication entity using multiple physical communication interfaces. The communication interfaces can be homogeneous (WiFi interfaces, or LTE/5G etc.) or heterogeneous (WiFi and LTE/5G etc.). The communication protocols must be able to manage the data streams, coming from the various interfaces as a single end-to-end stream, making all management transparent to the involved communication entities.

Two suitable protocols for network diversity-based communications are the MultiPath TCP (MP-TCP)[112] and the MultiPath Real-time Transport Protocol (RTP) (MP-RTP)[113]. The MP-TCP is an evolution of TCP that allows the simultaneous use of multiple Network Interface Cards (NICs) for a single TCP connection. The MP-RTP, directly related to multimedia data

[110] M. I. Hassan, H. L. Vu and T. Sakurai, "Performance analysis of the IEEE 802.11 MAC protocol for DSRC safety applications", IEEE Trans. Veh. Technol., vol. 60, no. 8, pp. 3882- 3896, Oct. 20

[111] V2X Technology Benchmark Testing, Sep. 2018, [online] Available: https://www.fcc.gov/ecfs/filing/109271050222769

[112] S. Barre, C. Paasch, and O. Bonaventure, "Multipath TCP: from Theory to Practice," in International Conference on Research in Networking. Springer, 2011, pp. 444–457

[113] V. Singh, S. Ahsan, and J. Ott, "MPRTP: Multipath Considerations for Real-Time Media," in Proceedings of the 4th ACM Multimedia Systems Conference. ACM, 2013, pp. 190–201

delivery, the main idea is that disjoint paths, between a sender and a receiver, can be used as a single logical one to deliver data flows. Such an architecture can provide failover capabilities, or can aggregate the overall capacity to increase the achievable QoE. It is worth to notice that the bandwidth increase should be pursued by carefully choosing the links to be used so to respect any latency constraints; in fact, heterogeneous networks typically exhibit different network statistics. Such a precaution is crucial when jointly using terrestrial and NTNs. Live video streaming has been proposed as MP-TCP-based[114] as well, i.e., using elastic protocols typically used in different scenarios, as for instance typical M2M/IoT ones[115]. Typically, real-time multimedia streaming occurs over UDP instead of TCP, because the constraint posed by live feeds makes un necessary, if not even detrimental, the use of retransmissions. In the case of elastic protocol, homogeneous paths (i.e., links showing comparable network statistics) represent a condition for satisfactory performance, otherwise corrective actions are required. When considering both mobility and the use of multiple paths, MP-TCP[116] can naturally shield the application layer from the multiple handoffs occurring at lower layers in mobility conditions. The use of network diversity in mobility conditions, both urban and suburban ones, has been studied in the scientific literature. In particular, strengthen the connection reliability, lightweight FEC solutions are preferred to resource-consuming ones, such as network coding[117] in similar scenarios.

[114] B. Wang, W. Wei, Z. Guo, and D. Towsley, "Multipath Live Streaming via TCP: Scheme, Performance and Benefits," ACM Transactions on Multimedia Computing, Communications, and Applications" (TOMM), vol. 5, no. 3, p. 25, 2009

[115] M. Bacco, T. De Cola, G. Giambene, and A. Gotta, "Advances on Elastic Traffic via M2M Satellite User Terminals," in2015 International Symposium on Wireless Communication Systems (ISWCS). IEEE, 2015, pp. 226–230

[116] C. Raiciu, D. Niculescu, M. Bagnulo, and M. J. Handley, "Opportunistic Mobility with Multipath TCP," in Proceedings of the sixth international workshop on MobiArch. ACM, 2011, pp. 7–12

[117] G. Giambene, D. K. Luong, V. A. Le, T. de Cola, and M. Muham-mad, "Transport Layer Performance Combining Multipath and Network Coding in Mobile Satellite Networks," International Journal of Satellite Communications and Networking, vol. 35, no. 6, pp. 583–603, 2017

6 Conclusions

TEACHING Work Package 2 is aimed at designing and developing the distributed computing and communication platform that support the execution of TEACHING and its use cases. We refer to such platform as the High-Performance Computing and Communication Infrastructure (HPC^2I). During the first year of the project, the WP had four tasks active: **T2.1** Design of distributed computing orchestration platform for CPSoS, **T2.2** – High-level Efficient exploitation of multi/many-core CPUs, GPUs and FPGAs, **T2.3** – High Performance Processing and Management of Data Streams, and **T2.4** – Sensors, IoT and wearable devices in CPSoS: management, tuning and orchestration.

In this document we frame the WP2 activities in the scientific and technological state-of-the-art. We then relied on such analysis to conduct the WP activities. This deliverable also presents a preliminary analysis of the TEACHING HPC^2I requirements and, starting from those ones, a preliminary conceptual design of the HPC^2I architecture. Finally, the document reports some relevant technologies that are candidate to be exploited in the context of WP activities.

Following the planned project's workplan, the results of the work conducted in WP2 and described in this deliverable, along with that of the other technical WPs, will drive the efforts on the core technology building during Year 2.

The project activities currently running will be complemented by the work on the tasks starting at **M13**: Task **T2.5** – Efficient and decentralized information exchange within single CPSoS and across different CPSoSs, **T2.6** – Seamless application deployment in Cloud and Edge resources for the distributed provisioning of computing capacity and **T2.7** - Silicon-born dependable AI.

The joint effort of the many WP activities will result in the integrated mockup of the HPC^2I system in D2.2, thus contributing to the project's milestone MS2 on the first integrated setup of the TEACHING platform to be delivered at M20.