ECSEL2017-2-783162

FitOptiVis

From the cloud to the edge - smart IntegraTion and OPtimisation Technologies for highly efficient Image and VIdeo processing Systems

Deliverable: D4.2 Final run-time models and support for energy, performance and other qualities

Due date of deliverable: 31-05-2020

Actual submission date: 31-05-2020

Start date of Project: 01 June 2018 Duration: 36 months

Responsible WP4: Tampere University (of Technology)

Revision: final version

Dissemination level		
PU	Public	✓
PP	Restricted to other programme participants (including the Commission Service	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (excluding the Commission Services)	

DOCUMENT INFO

Authors (alphabetical order)

Author	Company	E-mail
Francisco Barranco	UGR	fbarranco@ugr.es
Lubomír Bulej	CUNI	lubomir.bulej@mff.cuni.cz
Santiago Cáceres	ITI	scaceres@iti.es
Tiziana Fanni	UNICA	tiziana.fanni@diee.unica.it
Dip Goswami	TUE	d.goswami@tue.nl
Keijo Haataja	HURJA	keijo.haataja@hurja.fi
Pekka Jääskeläinen	TUT	pekka.jaaskelainen@tuni.fi
Jiří Kadlec	UTIA	kadlec@utia.cas.cz
Francesca Palumbo	UNISS	fpalumbo@uniss.it
Jukka Saarinen	NOKIA	jukka.saarinen@nokia.com
Raúl Santos de la Cámarra	HIB	rsantos@hi-iberia.es
Pablo Sánchez	UC	sanchez@teisa.unican.es
Carlo Sau	UNICA	carlo.sau@diee.unica.it
Shayan Tabatabaei Nikkhah	TUE	s.tabatabaei.nikkhah@tue.nl
Luis Medina Valdés	7SOLS	luis.medina@sevensols.com

Document history

Version	Date	Change
V1.0	22-05-2020	Submitted to EU portal.

Document data

Keywords	runtime platforms, runtime models, runtime adaptation
Editor Address data	Name: Lubomír Bulej Partner: CUNI Address: Faculty of Mathematics and Physics Charles University 118 00 Prague Czech Republic Phone: +420 95155 4189

© FitOpTiVis Consortium (Public Document)

Table of Contents

© FitOpTiVis Consortium (Public Document)

© FitOpTiVis Consortium (Public Document)

© FitOpTiVis Consortium (Public Document)

1. Executive Summary

This report represents deliverable D4.2, one of the outcomes of Task 4.1 in WP4 of the FitOptiVis project. This deliverable is an incremental update of deliverable D4.1. As such, this document retains the content of the previous deliverable and provides new or updated content to reflect the progress made during the second year of the project. The document is intended to be self-contained (and is written from that perspective), so that there is no need to cross-reference the previous version of the deliverable. However, to highlight the differences between D4.2 and D4.1, we provide a dedicated Chapter 2 which lists the new and updated content along with a brief summary.

The main objective of WP4 is to deal with the complexity of application runtime management while considering a diverse set of heterogeneous platform components and configurations. The WP4 solutions provide instances of the WP2 reference architecture described in deliverable D2.1.

This deliverable provides an overview of runtime platforms which represent platform components as defined in deliverable D2.1, spanning levels of abstraction to match the needs of applications with diverse set of requirements. Consequently, our platforms include a latency-managed edge-cloud platform for latency sensitive cloud applications, a distributed OpenCL-centric heterogeneous device runtime software stack which provides a unifying backbone to applications relying on hardware accelerators, both local and remote, an OpenMP runtime built on top of the distributed OpenCL runtime, the CompSOC platform for applications targeting execution on system-on-a-chip, and a deterministic networking platform to support time-sensitive applications with mixed-criticality communication requirements.

To enable adaptive control of application quality attributes (e.g., image resolution and quality, or frame rate) in response to resource availability and the desired quality trade-off, the runtime platforms need to provide means for resource managers to control application parameters linked to individual quality attributes and to manage resources assigned to an application. Each of the platforms enables adaptation at different levels of abstraction and at different time scales. To facilitate design of the necessary management interfaces, the deliverable also reports on adaptation scenarios relevant to use cases from various partners contributing to WP4.

The content of this deliverable contributes to milestones MS5 (M18 specification update) and MS6 (M24 partial demonstrators).

© FitOpTiVis Consortium (Public Document)

2. **Document Updates**

This chapter provides a brief summary of specific content that has been updated or added to D4.2 with respect to D4.1. Naturally, Chapters 3 and 6 have been updated to reflect the new content.

4.1 Managed-Latency Edge-Cloud Environment

- Updated platform status in Section 4.1.3 to include information about the newly developed performance predictor (further elaborated in Section 5.1.3).

4.2 Heterogeneous Distributed Software Runtime

- Added description of the low overhead control protocol and the distributed event-based synchronization which has been implemented in the runtime.
- Updated the status of the internal release of pocl-remote.

4.3 Extended OpenMP Runtime Infrastructure

- Updated Section 4.3 to reflect new OpenMP offloading requirements.
- Added Section 4.3.2 describing the development of a new approach for code offloading based on an LLVM pre-processor pass and integration of the offloading methodology in an open source compiler (clang).
- Added Section 4.3.5 providing an analysis of the use of OpenMP for offloading threads in a video pipeline.

4.5 The Xilinx Zynq Platform

- Removed Section 4.5.1 (Inter-Cloud Connectivity with Arrowhead) as obsolete.

4.6 Deterministic Networking Platform

- Updated description of a TSN bridge design and implementation to reflect use-case requirements in Section 4.6.1.
- Added Section 4.6.2 with a QRML model of TSN as a platform component, with details concerning configuration parameters of individual components and run-time monitoring provided by the time synchronization component.
- Added Sections 4.6.3 and 4.6.4 describing application of TSN in different use cases.

5.1 Reconfiguration in Managed-Latency Edge-Cloud

- Added Section 5.1.3 with an overview of a performance predictor which uses a novel technique for statistical prediction of the upper bound of the response time of a service sharing the same computer with other services.

5.2 Reconfiguration on the CompSOC Platform

- Added Section 5.2.4 describing an analytical framework for budget matching, which allows to determine if the provided budget matches the required budget in presence of multiple resources of different types. The framework is an important ingredient of quality and resource management support in FitOptiVis.

5.3 Reconfiguration in Processor/Co-processor Systems

© FitOpTiVis Consortium (Public Document)

- Updated Section 5.3 to describe integration of an overlay monitoring layer in the processor-coprocessor system, which enables collection of runtime metrics and subsequently runtime estimation of selected performance indicators.
- Added Section 5.3.3 describing initial definition and evaluation of a coprocessor for the Water Supply use case, which will be used to assess the envisioned dynamic parameter adjustment strategy.

5.4 Reconfigurable 8xSIMD Floating-point Accelerators

- Introduced support for reconfigurable floating-point accelerators on the Xilinx Zynq platform. Individual subsections provide description of the accelerator architecture, design considerations, and supported reconfiguration scenarios.

5.5.1 Modelling System Variants and Configuration Changes

- Updated the description of the modelling approach to take advantage of QRML, the FitOpTiVis DSL for capturing quality and resource management models.

5.5.2 Selection and Compression of Task-Specific Features

- Updated strategies for bandwidth reduction using regions of interest and described the application of the bandwidth reduction strategies in the Habit Tracking (UC3) and Smart Grid (UC9) use cases.
- Added QRML models of system components, along with component descriptions and reconfiguration scenarios in UC3 and UC9.

5.5.3 Distributed Image Pre-Processing and Optimized Image Segmentation

- Added QRML model of the Edge component, along with description of the relevant monitored metrics and reconfiguration scenarios to enable runtime adaptation to achieve the desired performance.

© FitOpTiVis Consortium (Public Document)

3. **Introduction**

Work package 4 addresses Objective 3 of the FitOptiVis project:

> ***Objective 3**: Real-time multi-objective combinatorial optimisation; data and process distribution; run-time adaptation through virtualization; run-time quality and resource management; energy driven adaptations; workload (re-)distribution; support for run-time upgrades.*

Specifically, in WP4 the consortium develops techniques for run-time resource management within the system architecture template outlined in WP2. The main goal is to deal with the complexity of application runtime management, reconfiguration, and monitoring, while considering a diverse set of heterogeneous platform components and configurations. To increase developer productivity and to promote vendor independence with respect to compute platform, this diversity should become transparent from an application developer's point of view. Task 4.1 focuses on run-time technologies and models to support management of performance, energy, and other qualities. This deliverable reports on the outcomes of the first two years of the project.

In Chapter 4, the report provides an overview of technologies that provide basis for virtual reconfigurable platforms and concrete platform components as defined in the FitOptiVis reference architecture (see deliverable D2.1). To satisfy the diverse set of requirements found in FitOptiVis use cases, multiple concrete platforms are needed, each tailored to serve different types of requirements. For example, while real-time applications with modest latency requirements and a time frame for reconfiguration measured in seconds may be well served by a solution utilizing a general-purpose compute cluster in an edge-cloud environment, a hard real-time application implementing a tight control loop may need to utilize custom FPGA accelerators to meet its latency requirements. Building a single unified hardware, software, and tooling framework to satisfy vastly different requirements would be neither possible, nor desirable. Instead, in FitOptiVis, we aim to unify at the level of concepts, principles, and abstractions to identify and extract commonalities found in different domains.

The idea is that the technologies developed in WP4 each serve a particular purpose and are intended to be used as building blocks for implementing different use-cases and demonstrators. This is captured in Figure 1, which shows an example instantiation of the runtime technologies in a "Multi-access Edge Computing" setup, which is a computing model comprising a local edge device (a lightweight terminal computer such as a smart phone or a smart camera) connected to a nearby edge-cloud (or a cluster of GPU and/or FPGA accelerator servers) using a fast network connection. In the context of FitOptiVis, use-cases such as UC2 and UC3 are examples of such a computing model.

© FitOpTiVis Consortium (Public Document)

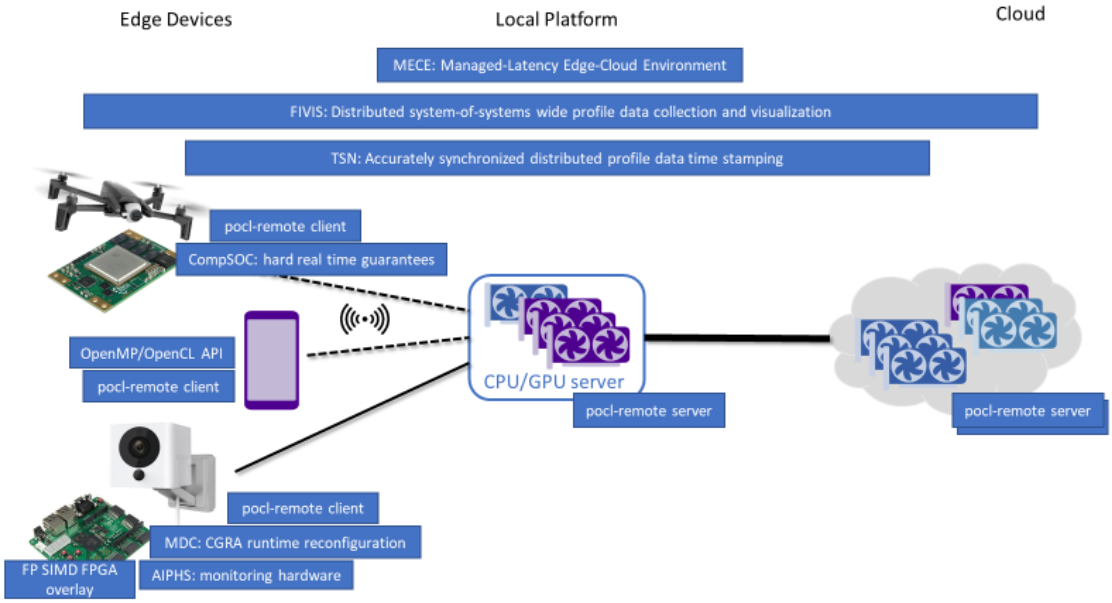


Figure 1. An example instantiation of WP4 technologies in a Multi-access Edge Computing (MEC) scenario (which applies to several use cases in the project), with different tools serving specific purposes in the implementation of the system. OpenMP serves as a layer which boosts developer productivity in defining the Terminal/Edge application. Pocl-remote is used for distributed computing across diverse heterogeneous resources. MECE handles the server-side control loop, taking into account the latencies incurred both by the potential virtualization of the server-side resources as well as the application running on the edge device. If required by an application, CompSOC can be used for managing hard real time guarantees at a System-on-a-Chip level. Different reconfiguration mechanisms (with different granularity) can be used to program FPGA devices on the fly—based on the requirements of the application being accelerated. MDC provides reconfiguration at task granularity, presenting a CGRA overlay architecture that can be implemented either as a FPGA soft core, or as an ASIC in a new SoC, FP SIMD FPGA overlays add runtime reconfigurable floating point accelerators, whereas AIPHS can generate monitoring (bus snooping) hardware to be utilized in the FPGA-based accelerators, which is discussed more thoroughly in D4.3. Finally, TSN can be used to synchronize the times of multiple server nodes to provide meaningful timestamps and accurate time triggered events as well as provide support mixed-criticality network traffic on shared network infrastructure. FIVIS enables global (system-of-systems level) profiling and performance data analysis.

A common theme of FitOptiVis systems is adaptive runtime management of various quality aspects through adjustment of configurable system parameters. The architecture of adaptive systems is often based on the MAPE-k loop [KEP03] architectural pattern, which provides a general concept of a control loop. Systems implementing such control loops can be nested to form a hierarchy of control loops operating at different time scales. This approach can be also applied in FitOptiVis, where a top-level MAPE-k loop can operate at the time frame of seconds, determining the setpoints for a lower-level control loop operating at the time scale of milliseconds or even microseconds. The presented technologies are intended for solutions operating at different time scales.

Section 4.1 describes a *multi-node managed-latency* private *edge-cloud* platform that will provide probabilistic guarantees to parts of applications (time-sensitive services) with *soft real-time* requirements which will be deployed in the edge-cloud. The platform aims to support solutions with reconfiguration time frames in seconds, which can be either general soft real-time services, or top-level adaptation control loops managing set points for lower-level control loops. By focusing on probabilistic guarantees, we aim to reduce the impact on developers by not requiring them to express application performance

© FitOpTiVis Consortium (Public Document)

requirements through many low-level metrics, but rather through a simple end-to-end metric defined on probe points provided by an application.

For interactive applications targeting shorter time frames, Section 4.2 provides a description of a *distributed*, *heterogeneous-device* runtime software stack based on OpenCL, which can be used to spread the execution of application's computational tasks to all available resources (local or remote), and which can be fully controlled from the application running on a terminal device. The foundation for an extensible and portable heterogeneous system-software stack had been laid out in the ALMARVI project, and is being extended in FitOptiVis to support new use cases in distributed and reconfigurable computing. This part directly addresses the objective of managing the complexity of a heterogeneous distributed execution platform and allowing an application to harness all available resources through a standardized API. Because OpenCL can encapsulate all types of compute devices ranging from general-purpose CPUs to fixed-function accelerators, the consortium believes that the diversity management goal is well met by relying on it as a backbone, by enabling easy support/integration path for the various hardware-software platforms developed in the project by partners, and by extending the standard whenever needed. Section 4.2 also lists potential extensions to the OpenCL API that will enable runtime monitoring, among other requirements associated with a distributed, dynamically changing environment. OpenCL supports defining heterogeneous task graphs via its command queue abstraction, which provides a basis for distributed heterogeneous task scheduling, which also helps Task 3.2 (Programming and Parallelization Support).

A higher-level programming model, OpenMP 5, is being added as an example of an end-user programming language on top of the developed stack. This addresses the goal of *transparency*. Because the OpenMP view of the platform components is more restricted than that of OpenCL, more decisions on the suitable devices for each function are delegated to the management layers in the stack, instead of relying solely on the programmer. The OpenMP 5 offloading support developed on top of the OpenCL based stack is described in Section 4.3.

For the lowest-level solutions operating at the shortest time frames, Sections 4.4 and 4.5 discuss two hardware-software platforms that are being supported and extended in the project: the CompSOC platform for composable and analysable hard real-time applications running on a single system-on-a-chip, and platform templates tailored to Xilinx Zynq-based FPGA SOCs as an easy-to-use implementation and prototyping platform. Both platforms target and support high-performance embedded computations, but place themselves in different layers of the work done within FitOptiVis: CompSOC defines a complete framework for design and implementation of hard real-time applications which utilize resource sharing, while the presented FPGA platforms enable prototyping and integrating of any hardware platforms with ease. The presented Xilinx-based platforms make a connection to the design flows in WP3 (Design-time support) allowing to prototype and utilize new hardware IP in combination with already commercialized ones running in the same system as described in WP5 (Devices and components).

An important component of any distributed system is the communication fabric. In the context of edge-cloud, we are mostly dealing with common networking technologies. However, many of the use cases are built around time-sensitive applications which need to exchange data with different levels of criticality. Section 4.6 therefore introduces a

© FitOpTiVis Consortium (Public Document)

platform component providing deterministic networking, which caters to the needs of time-sensitive applications. To this end, the platform provides a custom time-sensitive networking bridge built on top of standard networking technologies.

Chapter 5 deals with support for adaptation in the runtime platforms and applications built on those platforms. To support different trade-offs between various quality aspects (visual quality, resolution, latency) and resource usage (compute resources, I/O bandwidth, memory consumption), the architectural description of FitOptiVis applications (see Deliverable D2.1) will enable binding individual quality aspects to corresponding resource requirements. It will also expose configurable parameters that allows a runtime entity, e.g., an adaptation manager, to request a particular quality level for a specific aspect. Such an adaptation manager will then control the individual parameters to achieve a higher-level goal, e.g., best overall quality given fixed amount of resources, minimal resource usage, best quality possible, etc. The adaptation manager needs to closely co-operate (or be integrated) with the platform runtime in order to ensure that the resource requirements associated with the desired levels of different quality aspects are satisfied.

Similarly to Chapter 4, we have to deal with adaptation at different levels of abstraction corresponding to the supported runtime platforms. In Section 5.1 we, therefore, provide an overview of adaptation support in the context of the managed-latency edge-cloud platform, where the system needs to manage deployment of applications to individual nodes as well as allocation of resources such as CPU time, memory, and I/O bandwidth to co-located applications. For applications targeting systems-on-a-chip implementation and shorter time frames, Section 5.2 presents an overview of adaptation mechanisms and management interfaces on the CompSOC platform, along with mapping of CompSOC concepts to the FitOptiVis reference architecture and a mechanism to match provided and required budgets. Section 5.3 provides an overview of adaptation support for reconfigurable hardware, specifically targeting reconfigurable neural network accelerators, and Section 5.4 presents reconfigurable SIMD floating-point accelerators. Section 5.5 collects application-specific adaptation scenarios related to use cases from partners contributing to WP4, elaborating on the supported system configurations, conditions which trigger reconfiguration, monitored parameters, and other scenario-specific requirements.

Chapter 6 provides a short conclusion, briefly summarizing the advances during the first two years of the project.

In addition to the main document, the deliverable also contains a survey of existing virtualization and resource management techniques, which provides a basis for new contributions in FitOptiVis. The survey is included in Appendix A.

© FitOpTiVis Consortium (Public Document)

© FitOpTiVis Consortium (Public Document)

4. Runtime Platforms

This chapter provides an overview of technologies and concrete platforms that will serve as a basis for virtual reconfigurable platforms as defined in the FitOptiVis reference architecture. We describe the platform model and the correspondence to architectural concepts defined in WP2 (Reference architecture, virtual platform and integration), i.e., the “instantiation” of the WP2 architecture on a specific platform. Each platform serves to satisfy a different subset of the diverse requirements present in FitOptiVis use cases and is intended to applications operating at different time frames.

4.1 Managed-Latency Edge-Cloud Environment

Modern Cyber-physical Systems (CPS) rely on data from sensors and perform computationally-intensive tasks on the data (computer vision, data analytics, optimization, and decision making, learning and predictions) which often cannot be executed on edge devices due to the limited energy budget and computational power.

To obtain the necessary computational power, such systems are typically split into parts that execute on edge devices and parts that execute in the cloud. However, the connection with the physical world inherent to CPS requires these systems to operate and respond in real-time, whereas the cloud was primarily built to provide average throughput through massive scaling. The real-time requirements impose bounds on response time, and when executing tasks in the cloud, a significant part of the end-to-end response time is due to communication latency.

The concept of edge-cloud aims to tackle this problem by moving computation to computational clusters that are physically closer to edge devices. While this reduces communication latencies, edge-cloud alone does not guarantee bounded end-to-end response time, which becomes more dominated by the computation time. The reason is that while the cloud itself focuses on optimizing the average performance and the cost of computation, it does not provide any guarantees on the upper bound of the computation time of individual requests. To satisfy the needs of modern cloud-connected CPS we need an approach that can reflect the real-time requirements of modern CPSs even with cloud in the computation loop.

4.1.1 Probabilistic Latency Guarantees

Strict latency guarantees on each individual request are the domain of real-time programming, which comes at a very high price, as it forces developers to use a low-level programming language, severely limits the choice of libraries, and imposes a relatively exotic programming model of periodic non-blocking real-time tasks.

We instead advocate the use of standard cloud technologies (i.e., micro-services running in a container-based cloud such as Kubernetes) and modern high-level programming languages (e.g., Java, Scala, Python). However, we restrict ourselves to a class of applications for which soft real-time guarantees are enough (i.e., the guarantee on the end-to-end response is probabilistic, such as “in 99% of cases the response comes in 100ms and in 95% of cases the response comes in 40ms”).

It turns out that this is acceptable to a wide class of applications including augmented reality, real-time planning and coordination, video and audio processing, etc. Generally speaking, this class comprises any application that has a safe state and has a local

© FitOpTiVis Consortium (Public Document)

control loop that keeps the application in the safe state while computation is done in the cloud. Consequently, the soft real-time guarantee pertains to qualities such as availability and optimality, but not to safety. In the context of the FitOptiVis project, which generally focuses on developing distributed image and video processing pipelines, this applies to many of the use cases (augmented reality, habit tracking, municipal speed cameras, etc.).

4.1.2 Probes and Latency Requirements

One of the goals of our work is to minimize the impact of using a managed-latency edge-cloud environment on application developers. Given that we aim to use standard cloud technologies, we also envision the developer creating artefacts, e.g., for the Kubernetes (K8S) platform. The only required extension is the specification of application real-time requirements in the application deployment descriptor.

Contrary to common cloud deployment practices, we aim to spare the developer from dealing with the selection of VM type, the number of virtual CPUs, memory, IOPS, etc. Similarly, we aim to avoid specification of auto-scaling rules (including triggers), because we consider these to be implementation details of the cloud platform's internal mechanisms which the developer is not equipped to set correctly without an experiment.

We instead work with an abstraction in which the developer is responsible for providing the application and its soft real-time requirements, while the responsibility for assessing the performance of the cloud application, as well as allocating resources (i.e., the required number of virtual CPUs, memory, IOPS, etc.) and making scheduling and deployment decisions so as to ensure that the (probabilistic) guarantees are met, lies with the cloud platform. Consequently, if the platform determines that it cannot satisfy the requirements, it will not admit the application for deployment.

Specifically, when developing an edge-cloud application, the developer has to describe the application in terms of an auto-scaling micro-service with added communication latency requirements. In the specific case of the Kubernetes cloud platform, we extend the Kubernetes application deployment descriptor to allow declaration of measurement *probes,* special functions provided by the developer which the system uses to assess the performance of the application in a particular deployment scenario.

An example deployment descriptor for a sample face-recognition application is shown in Listing 1. The timing requirements for the application state that the response of the application on the "recognize" probe should be below 100 milliseconds in 99% cases, and below 50 milliseconds in 95% cases.

© FitOpTiVis Consortium (Public Document)

```
kind: Deployment
metadata:
  name: recognizer-deployment
  labels:
    app: recognizer
spec: # microservice specification
  template:
  metadata:
    labels:
      app: recognizer
  spec:
    containers:
    - name: recog
      image: repo/recog
      ports:
      - containerPort: 7777
      probes: # probes
      - name: recognize
      timingRequirements:
      - name: recognize limit
        probe: recognize
        limits:
        - probability: 0.99
          time: 100 # Max. 100ms in 99% cases
        - probability: 0.95
          time: 50 # Max. 50ms in 95% cases
```

Listing 1. Application deployment descriptor with timing requirements

A probe (or a set of probes) has to capture the essential behaviour of the application so that when invoked by the cloud-edge platform, it will provide a representative sample of the application's performance in the current deployment configuration. Expressing the application timing requirements over developer-supplied probes simplifies the specification of the contract between the application and the cloud-edge platform, and allows it to treat the application as a black-box.

4.1.3 Platform Status

The development of the managed-latency edge-cloud platform is in progress. During the first year of the project, several design iterations have been made and work on prototype implementation has been started. Inter-module interfaces, application middleware, and module prototypes have been implemented.

During the second year of the project, in addition to continued platform development, we have been investigating methods for performance prediction of co-located workloads. Specifically, we focused on developing a prediction method that uses of performance measurements collected while executing different combinations of co-located workloads to predict performance of new, previously unseen, workload combinations. In addition, we have been working on experimental methods to automatically establish the operational boundaries of the predictor.

Given the experimental nature and possibly involved installation and configuration of the prototype, we plan to make the platform available as a hosted service during the third year of the project. We will work closely with partners interested in deploying parts of their application in a managed-latency edge-cloud environment.

© FitOpTiVis Consortium (Public Document)

4.2 Heterogeneous Distributed Software Runtime

The development of a single-node heterogeneous software stack based on OpenCL was initiated in the ALMARVI project. In FitOptiVis, this stack is being extended to support a distributed edge-cloud setup that can map the architecture models defined in WP2 to concrete run-time concepts of execution platforms and their topologies while supporting new devices developed with WP3 technologies and other devices and components of WP5.

The primary questions we seek answers in the runtime stack development for are:

- What are the workloads that need to be executed on local devices given 5G, WiFi6 and other high-speed low-latency wireless network technologies?
- Where are the latency bottlenecks when offloading interactive applications across such networks to cloud-edge servers?
- Can we distribute event synchronization to minimize communication due to back-and-forth synchronization between the “application device” and the cloud-edge servers?

These questions are approached by developing a proof-of-concept heterogeneous runtime that is optimized also for low-latency tasks and which can support also other types of computation offloading in addition to those based on frame serving (e.g. cloud gaming which has become popular in the recent years).

The software stack being developed is shown in Figure 1, while an example usage context is shown in Figure 2.

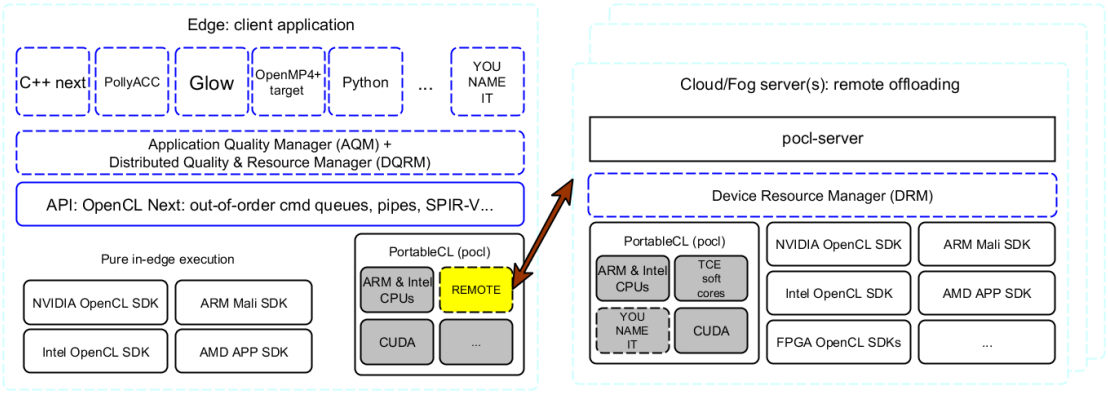


Figure 2. Multi-node heterogeneous distributed software runtime stack.

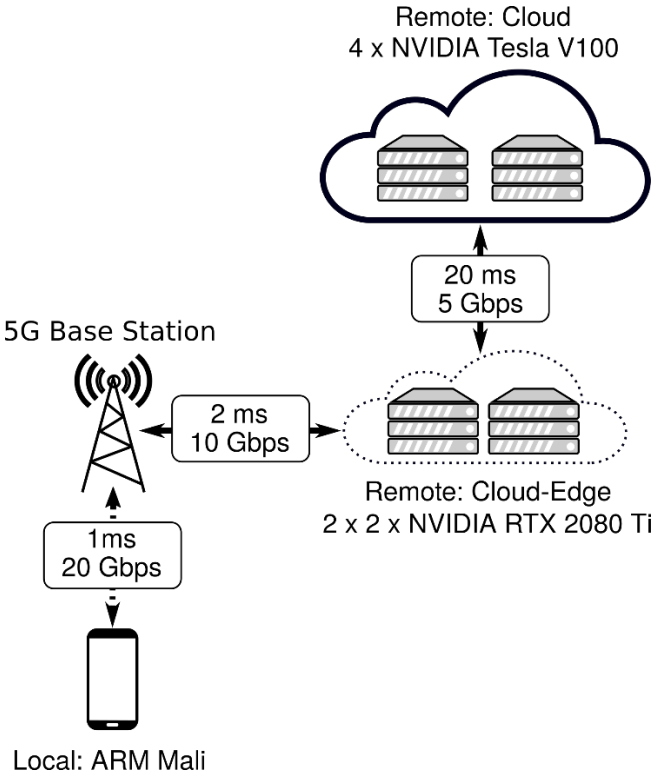


Figure 3. An example use context for the distributed runtime software stack. A terminal device (here a smartphone) deploys and starts the OpenCL application which then through a fast wireless link communicates to remote GPU devices in clusters at the cloud-edge and in the cloud.

4.2.1 OpenCL API Extension Candidates

The current notion is that OpenCL can serve as a good basis for a compute API both in local and distributed scenarios. However, already during the first year of the project, we identified the following features, which might be beneficial to add to the API (first as extensions and later as official part of the standard) to better support remote cloud-edge offloading scenarios:

- Platform: **Device Proximity**. The existing OpenCL API (practically) does not model connectivity between devices. Devices are assumed to reside in a single computer and to be accessible at most via a system bus such as PCIe or AXI, with shared external memory and/or per-device external memory. It would be beneficial to allow applications to make offloading decisions based on how efficiently devices are connected together: the API could be a platform-level query API with a possibility to query for the link between two devices. How the links are modelled and categorized is an open question at this point. E.g. 1) same shared-memory hierarchy, 2) same system bus, 3) in the same local network, 4) internet connectivity
- Device: **Link Status**. Especially with 3) of the previous item and especially with 4), the performance of the link heavily depends on the simultaneous traffic and other varying conditions (e.g. the proximity of the nearest 5G base station). It would be useful to be able to monitor historical statistical information of the link's

performance (e.g, in the past 5 seconds, or the past 5 buffer transfers). Because it is hard to isolate the network part's time from the client side code, it might be useful as an OpenCL runtime API. Of course the most important link status information is whether the link is working in the first place, as it affects the reachability of the device.

- Device: **Reachability**. In OpenCL there is already a flag for 'availability' of the device. This might be reused for scenarios where a remote device is temporarily unavailable due to networking issues.
- Command Queues: **Performance History**. Auto-tuning scenarios attempt to execute a kernel on multiple devices while varying parameters that affect execution. While the information is natural to reside on the client side of the OpenCL API, it might be useful to provide some level of support in the runtime API for querying the estimated performance of the given kernel. The kernel performance estimate might be identified with a hash and input buffer sizes or similar. It might be difficult to design this API to fit OpenCL therefore it might be better to keep it in a client-side helper API layer.
- Command Queues: **Command's Energy Consumption**. Now the profiling command queues allow storing time stamps of events. In terms of tuning the power performance, it might be interesting to also record the consumed energy in case the target supports such information. This might be difficult to get accurate as it's hard to account for which kernel consumed the energy in the processor especially if there are multiple ones running. It's worth researching at least for the dedicated GPU farm scenario where we execute one kernel at a time and might then resort to average power numbers which can be multiplied with the execution time. The OpenCL API could be connected to the profiling command queues time stamping system: the time stamps could also record "energy stamps" at a similar incremental fashion.
- Command Queues: **More Profiling/Performance Counters**: Advanced profiling information could include the cache hit miss counter values in a similar stamping fashion with the same caveat as above: in case multiple kernels are executing at the same time, it might be difficult to isolate which kernel caused which part of the cache level misses.
- Device: **Temperature Readings** of the processor/memories or any other components equipped with a temperature sensor.
- Command Queues: **Real Time Commands with Execution Cancellation**: In some soft real time cases we can just reduce quality when a kernel takes too long time. It would be useful to provide mechanism to the command queues that allow killing a kernel when a time limit is reached. This could yield a special "timeout event" which other commands could listen to and kill also the next ones that are dependent on the regular finish event that the killed command should have produced.
- Buffers: **Unreliable Buffers**: This is connected to the soft real-time case and the cancelled kernels, and not delivering full data in time, but still producing some useful data. E.g. when we produce images in a tiled fashion, it may be useful to display a partially rendered/decompressed frame, especially when applying heavy filtering on top of it or when it's assumed that the incomplete frame in general looks OK if there are enough complete frames displayed per second.

© FitOpTiVis Consortium (Public Document)

- Buffers: **File-initialized Buffers**: Some of the buffer content could be initialized from files (possibly an URI) in the system where the remote Device resides. This is currently not possible in OpenCL as it only allows initialization from an array.

4.2.2 Using pocl-remote

To provide the reader with an idea on how remote offloading works with pocl-remote, brief usage instructions are given here, while a more detailed documentation, including build instructions, can be found at:

https://github.com/cpc/pocl-fitoptivis/blob/master/doc/sphinx/source/remote.rst

On the server, the `clinfo` command must list at least one OpenCL device. The server can be then started using the following command:

```
./server/pocld <IP ADDRESS> <PORT>
```

Note that `pocld` will listen on two ports, PORT and PORT+1. The number of messages produced by the server can be adjusted by setting the `POCLD_LOGLEVEL` environment variable to the desired level before running `pocld`. The default log level is `err`. The server accepts the following log levels: `debug`, `info`, `warn`, `err`, `critical`, and `off`. On the client, the following environment variables need to be exported:

```
export POCL_DEVICES=remote
export POCL_REMOTE0_PARAMETERS=<IP ADDRESS>:<PORT>/<DEVICE ID>
```

The IP ADDRESS and PORT values are self-explanatory. PORT is the lower of the two port numbers assigned to the server. The DEVICE ID is the index of the device on the server. Valid indices range from 0 to N-1, where N is the total number of devices across all platforms on the server. The index is the order in which `pocld` lists the devices in the OpenCL platform it uses. This is the same order as displayed by `clinfo`.

The `clinfo` tool can be used to perform a "smoke test" to ensure that the distributed setup works. When configured properly, the tool should also list remote devices:

```
$ clinfo|grep pocl-remote
Device Version OpenCL 1.2 CUDA HSTR: pocl-remote 123.456.789.123:10998/0
```

A simple dot-product example can be then run by executing the example1 binary:

```
$ cd examples/example1
$ ./example1
(0.000000, 0.000000, 0.000000, 0.000000) . (0.000000, 0.000000, 0.000000, 0.000000) = 0.000000
(1.000000, 1.000000, 1.000000, 1.000000) . (1.000000, 1.000000, 1.000000, 1.000000) = 4.000000
(2.000000, 2.000000, 2.000000, 2.000000) . (2.000000, 2.000000, 2.000000, 2.000000) = 16.000000
(3.000000, 3.000000, 3.000000, 3.000000) . (3.000000, 3.000000, 3.000000, 3.000000) = 36.000000
OK
```

4.2.3 Low-Overhead Control Protocol

Use of a more general and feature-rich communication framework was foregone in favour of working directly with TCP sockets that have been configured for minimum possible OS-side latency, and packets with well-defined in-memory representation to remove any serialization and deserialization overhead associated with more general purpose portable data representations.

The protocol is implemented as plain C structures whose in-memory representation is fixed and which start with a field signifying their exact type, i.e. using the tagged union pattern. The downside of this approach is that all variants end up being padded to the size of the largest existing variant. Initial testing has shown that this introduces as much

© FitOpTiVis Consortium (Public Document)

as a couple of kilobytes in overhead, as most packets are less than a few hundred bytes. This includes buffer transfers, as the machine-learning frameworks we tested initially ended up creating dozens (up to hundreds) of less than kilobyte-sized buffers.

To address this, command-specific size value is sent before the structure and the part of the structure that goes over this size is left undefined since it will not be accessed when handling the given command. This adds slight overhead due to requiring an extra read call to the network driver, but still avoids a more traditional deserialization step.

4.2.4 Distributed Event-Based Synchronization

Inter-command synchronization is handled internally on the remote servers by utilizing event dependency information as well as buffer dependencies extracted from in-order queues. This way commands can be started as soon as they are received, given their dependencies are met and the execution can proceed independently from the host device, avoiding round-trips to the main device, which can heavily impact the perceived overall latency.

Results are sent back to the host from a separate thread that polls in-flight tasks in order to send the reply as early as possible without interrupting reception of new tasks nor execution of the current ones.

Tasks spread across multiple devices are synchronized on two levels: between devices on the remote server no extra network communication to the application is needed beyond notifying the host application of task completion, as illustrated in Figure 3. For synchronization between different remote servers, an extra thread is added for every peer to listen for incoming data migration requests. This way the host application does not need to be involved after firing off the initial migration command, and the command only needs to be dispatched to one remote server. Remote servers are assumed to be located in close proximity to each other (in the same data center), and thus to have much faster connections to each other than to the host application.

© FitOpTiVis Consortium (Public Document)

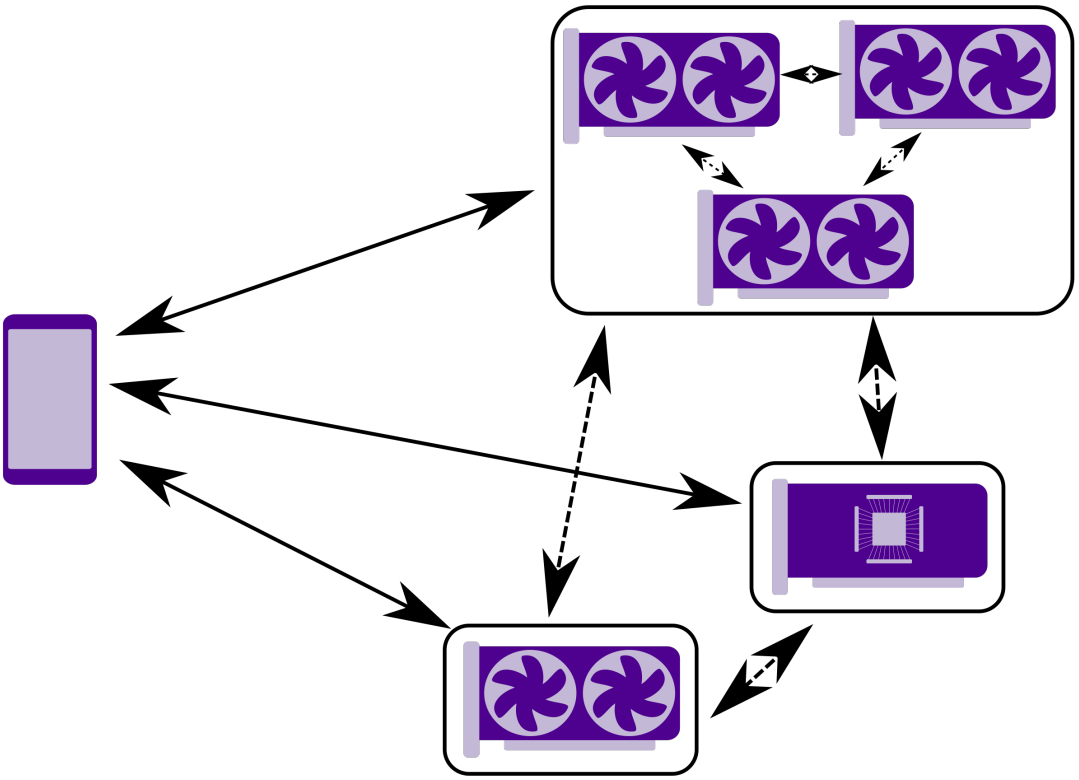


Figure 4. Multiple levels of data transfer and event signalling - application to remote server, peer-to-peer communication between servers and direct transfers between devices in a single server

4.2.5 Platform Status

The distributed OpenCL runtime is being implemented within the Portable Computing Language (POCL) open source project, with internal releases made available to the project partners until the runtime becomes mature enough for general use by the open source community, at which point the code will be published at http://code.portablecl.org.

At the time of writing this document, the latest internal release available to project partners at https://github.com/cpc/pocl-fitoptivis was labelled as version 0.7 with the following feature highlights:

- Remote code includes work to make event processing more asynchronous
- ALMAIF driver was updated and optimized; some new features (see docs for details):
- Hybrid compilation (allows running tests on TCE and then same tests on FPGA via ALMAIF, without having to change test code to load from binaries)
- Linux UIO support (possible to run programs without root)
- TCE was made thread-safe, new driver call-backs implemented, new math library functions implemented
- Glow tests added, TCE driver should now pass 90%+ of tests
- Improved support for more complicated multiple-device setups
- Android support

© FitOpTiVis Consortium (Public Document)

We now consider the runtime to be in “optimization stage”, because the main known stability issues have been resolved. The rest of the time in FitOptiVis will be spent on further scalability and performance improvements.

4.3 Extended OpenMP Runtime Infrastructure

OpenMP is the de-facto standard for parallel programming of Symmetric Multi-Processing (SMP) architectures with shared memory. During the last years, OpenMP specifications have been adding new features to support parallel programming on heterogeneous platforms. In fact, recent releases of popular compilers (such as gcc and clang) support the latest OpenMP specification (5.0), which supports runtime offloading of code to different devices such as NVIDIA GPUs, Intel Xeon-Phi co-processor, and multi-core architectures.

The OpenMP target offloading methodology differs from approaches used in other parallel programming environments such as OpenCL. In OpenMP, the code to be offloaded is precompiled for all targeted devices at build time. This main disadvantage of this approach is that all target devices must be supported by the OpenMP compiler. In contrast, OpenCL relies on compilation at runtime, which makes it easy to support new devices as they become available, but also introduces runtime overhead due to runtime compilation and compilation error management.

More importantly, though, there are devices such as FPGAs, which cannot be efficiently programmed using OpenCL—to generate an efficient FPGA implementation, the source code usually requires extensive modifications (manual or automatic code rewriting) to make it suitable for hardware synthesis. In some cases, specific hardware implementations need to be provided in a hardware definition language (HDL). This makes FPGA synthesis difficult to integrate even with traditional software build process, let alone with runtime compilation employed by OpenCL.

In FitOptiVis, the consortium is developing a new OpenMP offloading methodology which explores solutions to these limitations. The new approach, presented in the following sections, is based on two main techniques: source-code offloading and dynamic code management at runtime.

4.3.1 OpenMP Offloading Requirements

To support the new offloading methodology, the runtime implementation developed within the consortium aims to meet the following requirements:

- During compilation, the compiler should include in the executable files the code of the threads that could be allocated in different computation resources at runtime.
- There should be a methodology that allows developing new thread implementations after compilation, but before application execution. The methodology allows extracting the thread code from the executable file and defines mechanisms for dynamic loading of the new implementations.
- During execution, the runtime infrastructure should identify all the available thread implementations. The new implementations will be dynamically loaded.
- The runtime infrastructure provides dynamic thread allocation during application execution.

© FitOpTiVis Consortium (Public Document)

- The runtime library provides information about the available thread implementations and well as identified computing resources.
- The computing resource information could optionally include performance data, such as memory size and clock frequency.
- The device-specific implementation of a thread could optionally include performance data, such as memory requirements, execution time or power consumption.
- During code execution, the runtime library provides a methodology to facilitate thread runtime monitoring.

4.3.2 OpenMP Offloading Methodology

The goal of the consortium is to develop an OpenMP extension meeting the above requirements. So far, we have provide a runtime library that is capable of satisfying the following requirements:

- The runtime infrastructure can detect and dynamically load implementations of target code that were developed after the original code compilation.
- The runtime library provides basic monitoring of thread execution.
- The device-specific implementations can provide performance data for the runtime, which is then used to select the optimal implementation to execute.

In addition, we have been working on the compiler driver of an open source compiler (clang/llvm) to satisfy additional requirements that were not the sole responsibility of the runtime library, such as the inclusion of target code for different resources in executable files, or the development of the code extraction system. To comply with these requirements, we first modified the Clang compiler driver, but due to tight API bindings between the compiler driver and the Clang code generator, we had to use a different approach for code extraction.

The new approach is based on exploiting the capabilities of the LLVM framework to create an automatic code transformation tool that can directly modify the existing OpenMP code for device offloading as well as integrate the new dynamic runtime library. This approach integrates the automatic extraction of OpenMP thread code, the integration of other runtimes such the FitOptiVis dynamic runtime library, and the infrastructure to implement the target code extraction tool.

The code transformation pass needs to satisfy the following requirements:

- It must maintain code functionality, because it is only slicing the parts that define the target thread code.
- It should automatically include any additional files required by a specific target, as well as add any function declarations required after slicing and retooling of the code.
- It should generate code from which an OpenMP compiler can automatically build an executable.
- It should be integrated in our modified compiler toolchain, performing code transformation and compilation without extra intervention from the user. The compilation process is therefore split into two phases: modification of the OpenMP code and subsequent compilation.

© FitOpTiVis Consortium (Public Document)

4.3.3 The OpenMP Framework

We have been extending the standard OpenMP methodology to meet the above defined requirements in order to support a new target: the thread source code.

We provide a dynamic library to enable integration of our extension into application code. The use of the library is complemented with a code pre-processing pass that will slice and extract the target code. It also automatically provide runtime support for the dynamic library. During execution, the runtime identifies available implementations and allows selecting the desired implementation of the current thread.

The extended OpenMP framework is shown in Figure 4. The framework currently supports the activities shown in the solid-green boxes, which implement the dynamic thread-implementation management at runtime. The compilation pass embeds the thread source code in the OpenMP executable. An extraction tool can access this code in order to use a different compilation process. The new target implementation is compiled into a dynamic library that is loaded at runtime. The runtime uses an environment variable to discover the newly produced thread implementation libraries and loads them using infrastructure code that was automatically generated by the code extractor. For this reason, all implementations include a common infrastructure that allows identifying the OpenMP thread that the target implementation provides. The OpenMP applications can also access this infrastructure, which allows reconfiguring the thread target allocation at runtime.

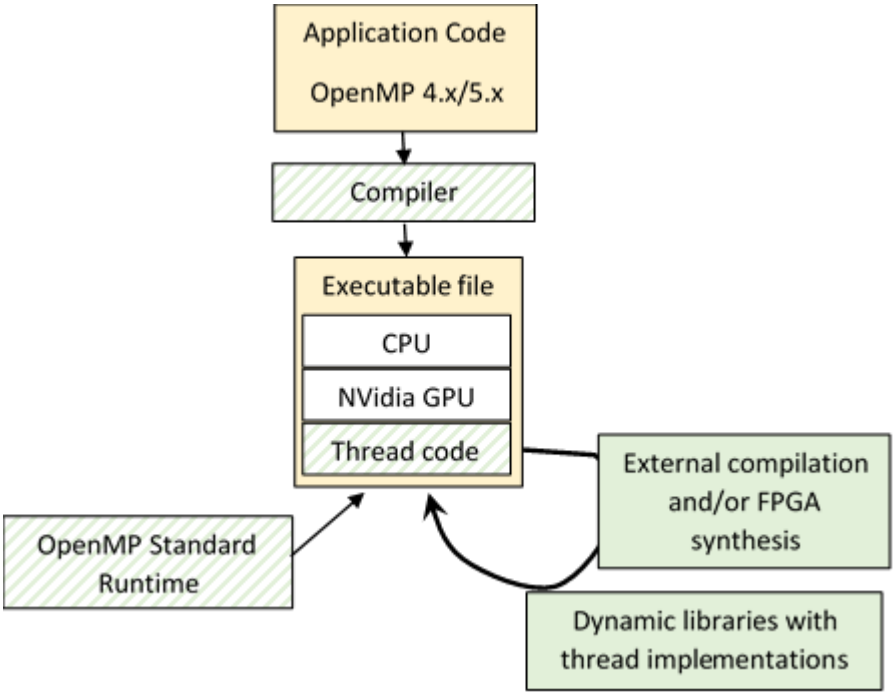


Figure 5. The Extended OpenMP framework.

4.3.4 OpenMP and OpenCL Integration

The methodology presented in the previous section has been extended to implement OpenMP threads in OpenCL, so that it fits on top of the OpenCL-centric runtime stack described in Section 4.2.

© FitOpTiVis Consortium (Public Document)

To this end, we generate an OpenCL kernel from the OpenMP thread code sections during code transformation, together with a library that synchronizes the OpenMP thread management and the OpenCL-based resource control. During execution, the application can select the OpenCL device that will execute the thread code, which is then compiled at runtime using the OpenCL API.

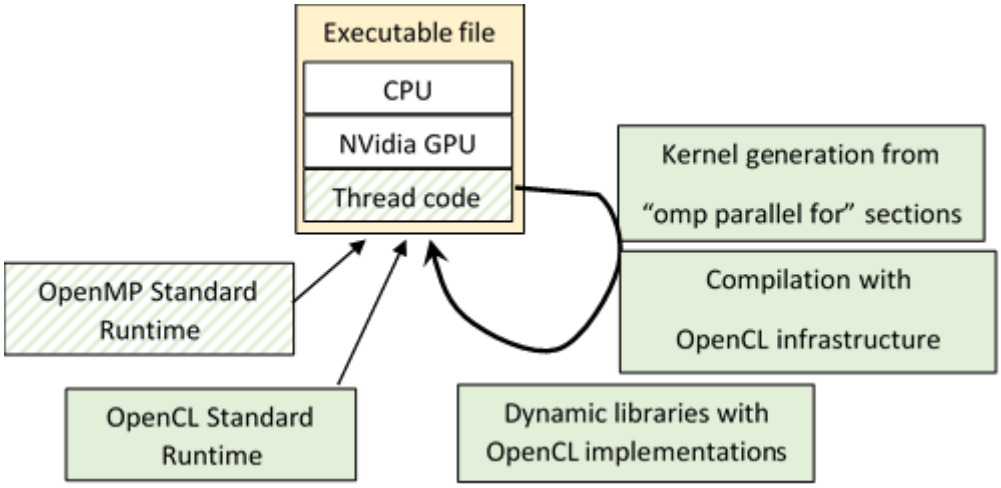


Figure 6. OpenCL integration in the OpenMP infrastructure.

This integration has the advantage of adding support for remote-device infrastructures, such as pocl-remote presented in Section 4.2, as well as any other remote device implementations.

4.3.5 Offloading OpenMP threads in a video pipeline

To demonstrate the methodology, we have developed a working example of a video pipeline, which is similar to a pipeline found in UC10, where this approach will be used. In this deliverable, we present a simple example of real-time edge extraction pipeline in which some of the OpenMP threads are offloaded to different hardware resources.

The pipeline, shown in Figure 6, consists of a camera component, which captures and relays images, a compute component, which performs scaling, filtering and edge detection on the images, and a display component, which provides the user with a side-by-side view of the original and the processed images. In particular, the compute component comprises four sub-components: a grayscale filter, a median filter, an edge detection algorithm, and a scaling algorithm. These sub-components were implemented as kernels that can be offloaded to different computing devices.

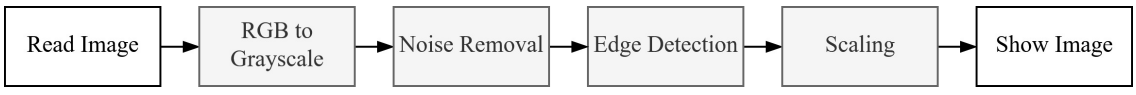


Figure 7. Architecture of the image processing pipeline (with the compute components in gray)

To implement this video pipeline, two OpenMP parallelization strategies were used:

1. Parallel section based parallelization.
2. While loop with OpenMP tasks and resource control with semaphores.

The first approach (parallel sections) is shown in Figure 7. Every thread (kernel) is implemented in an OpenMP parallel section that is executed in parallel with other sections. Each thread has an internal loop that allows maintaining the video pipeline and a barrier for synchronization with other threads. In this implementation, the video frames are transferred from one thread to another after barrier synchronization. This implementation is close to the UML/MARTE component model that has been developed in WP3, which is why we discuss this implementation in more detail.

The second parallelization strategy provides similar results. It only has an execution path (the video pipeline) which is concurrently executed by several threads. The threads use semaphores and critical sections to avoid resource access conflicts. For example, if four hardware threads or cores are allocated to execute the video pipeline and there is only one camera, only one thread will be allowed to access the camera in a particular time slot. The synchronization on resources causes the threads to execute in a pipeline fashion. The code in the critical sections could be offloaded to different target devices.

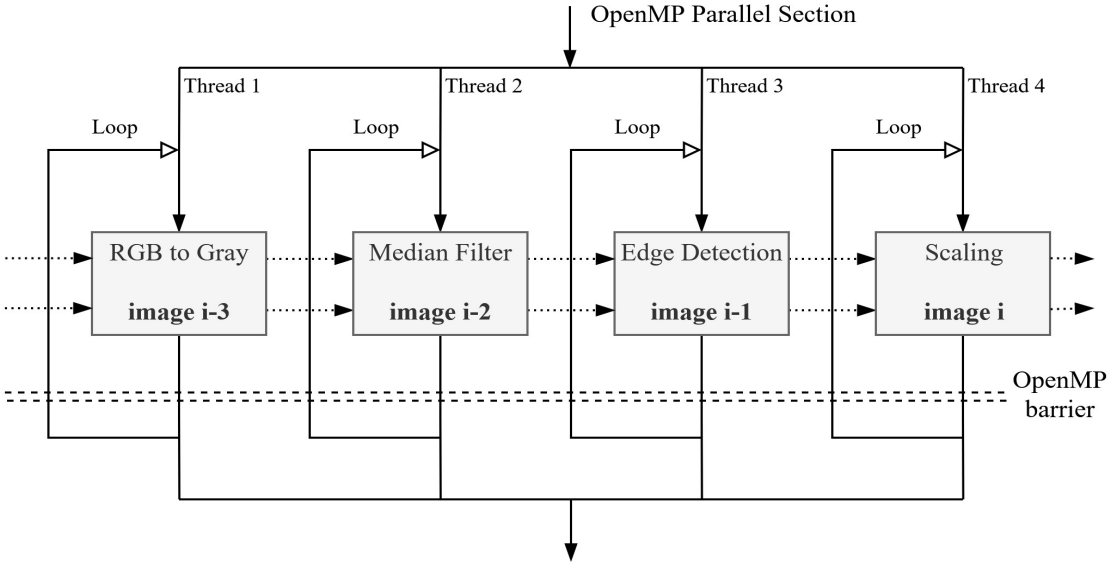


Figure 8. Image processing pipeline based on OpenMP parallel sections

Once the architecture was set, a reference CPU-based implementation was developed and profiled in order to find the bottlenecks in the system. We found that the edge detection algorithm requires more execution time than the other components. For this reason, we have explored two target offloading approaches: GPU-based implementation executed on a traditional PC, and an FPGA-based implementation executing on a Xilinx Zynq MPSoC.

For the GPU-accelerated architecture, an OpenCL kernel was generated and compiled using the standard OpenCL runtime compilation mode. For the FPGA-accelerated MPSoC, the extracted kernel code was heavily modified to take advantage of the Vivado HLS FPGA synthesis tool. Both implementations provided substantial throughput gains, as shown in Table 1 and Table 2.

Table 1 shows the frame rates achieved for a sequential CPU-only implementation, a standard OpenMP-based parallel implementation (which uses four threads executing on

© FitOpTiVis Consortium (Public Document)

four CPU cores), and a GPU-accelerated implementation (which offloads the heaviest thread on the GPU and executes the three remaining threads on three CPU cores).

Table 1. Pipeline speedup on Intel CPU + NVidia GPU

OVERALL PERFORMANCE ON PC - INTEL CORE I7-3610QM CPU + NVIDIA GT630M GPU

Implementation	Frame Rate	Speed-up
Serial (CPU-only)	1.60 FPS	x1.0
Parallel (CPU-only)	1.84 FPS	x1.2
Parallel + Offloading (GPU)	15.31 FPS	x9.6

Table 2 shows the results of executing the pipeline on the Xilinx Zynq MPSoC with FPGA offloading.

Table 2. Speedup results on Zynq MPSoC CPU+FPGA

OVERALL PERFORMANCE ON XILINX ZCU102 - ZYNQ MPSOC ARM CORTEX-A53 CPU + FPGA

Implementation	Frame Rate	Speed-up
Serial (CPU-only)	0.92 FPS	x1.0
Parallel (CPU-only)	1.13 FPS	x1.2
Parallel + Offloading (FPGA)	21.36 FPS	x23.2

We also want to highlight the need for device-specific optimization when using automatic hardware synthesis tools. If such tools are simply used on the extracted kernel code, the performance achieved by the synthesized hardware is far from impressive. However, after significant code modifications (with hardware synthesis in mind), the generated FPGA hardware can provide tremendous speedup.

This is illustrated by results in Table 3, which shows a side-by-side comparison of the throughput of two FPGA implementations of the median filter sub-component: one is synthesized directly from the extracted kernel code, while the other is synthesized from code which was heavily modified after extraction.

© FitOpTiVis Consortium (Public Document)

Table 3. Performance of hardware synthesized from unmodified and optimized kernel code

MEDIAN FILTER HW IP PERFORMANCE ESTIMATES AFTER HIGH-LEVEL SYNTHESIS

Input to Vivado HLS	Ordinary C/C++	HW-oriented C/C++
Image resolution	1920x1080	1920x1080
Clock period	3.08 ns	4.58 ns
Clock cycles	360810722	2073609
IP throughput	**0.90 fps**	**105.29 fps**

4.3.6 OpenMP Extension Status

During the first year, the consortium has developed the initial approach and infrastructure to support dynamic thread implementations, and demonstrated the integration of OpenCL into OpenMP.

Throughout the second year, we have made significant progress on slicing and integration of target code into a fat binary executable by leveraging the capabilities of the LLVM/Clang compiler. We have added a pre-processing pass to the compilation in which we analyse the thread code, separate target regions into different files, perform code substitution in the original code, and automatically load our runtime library. This makes the original OpenMP program ready for compilation by any OpenMP-compliant compiler which would generate the executable code.

The approach is illustrated on the flowchart in Figure 8. Starting from the unmodified source code, we run the pre-processing executable that will analyse, slice and modify the code to make it ready for compilation. The modified code adds support for the new target offloading style and includes the original target (kernel) source code embedded in string constants.

While this approach has been originally devised for transformations of OpenMP code to OpenMP code, once implemented, the pre-processing pass can be extended to also offload (with some limitations) OpenMP threads to OpenCL kernels.

In comparison, the original approach required modifications and extensions to the Clang OpenMP runtime libraries, extensive modifications of the code generation module of the compiler driver and of the compiler interfaces between the code generation module and the Clang Runtime Library. It also required creating an external tool for code extraction.

While such an approach may have seemed convenient and low on toolchain bloat, it ended up requiring extensive modifications to tightly coupled and not very well documented libraries and APIs. In addition, both the code generation module and the runtime library were specifically designed to adhere to OpenMP 4.5/5.0 specifications. Consequently, it is not ready for device-agnostic offloading or code injection at runtime (after compilation), because the target intermediate representation is generated in tandem with the target region delimitation and outlining.

© FitOpTiVis Consortium (Public Document)

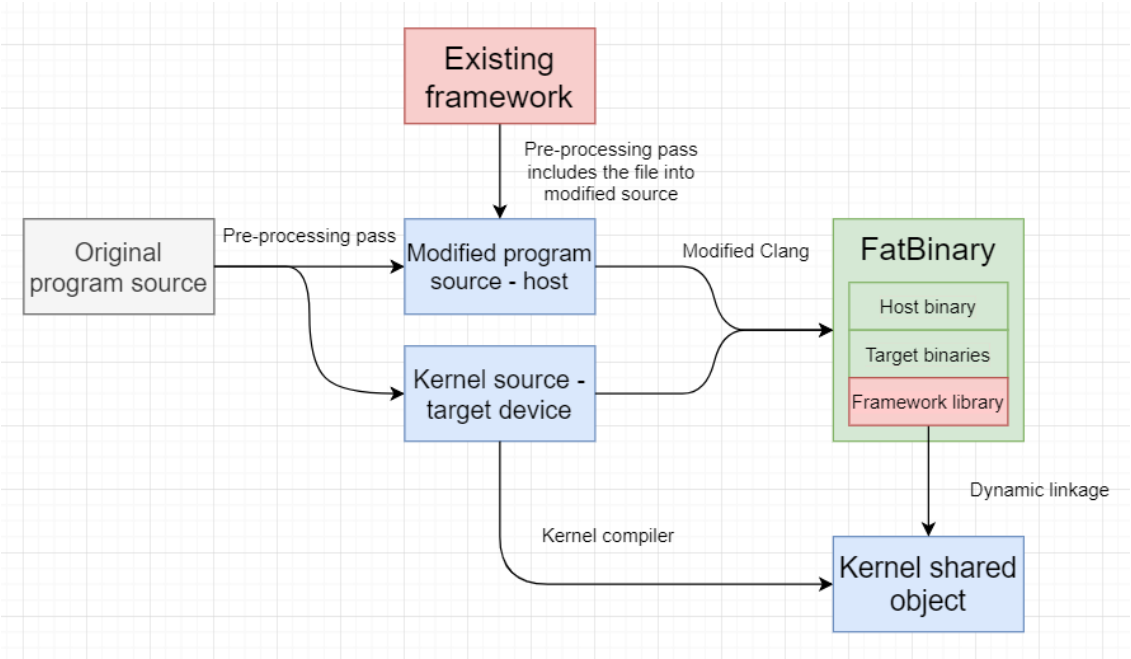


Figure 9. The pre-processing approach for integrating target code into fat binary

The improved approach aims to leverage the OpenMP code annotation pragmas to direct not only compilation for known targets, but also for unknown targets at runtime, without requiring complex modifications of undocumented LLVM/Clang code. It also simplifies the construction of the code extraction tool, which can then extract target code directly from the source code.

The main difference between the two approaches is how the extra functionality is added. The modifications and extensions to the Clang OpenMP runtime library required by the original approach would need to be accepted into the official Clang code. This would require new API endpoints for the Clang compiler to call to be introduced into the code generator.

In contrast, the improved approach adds the new functionality as an extra runtime library and code to load the library is automatically added during the pre-processing pass. This makes supporting the extensions much simpler, with minimal modifications in the code generator module (to add the source code of target regions into symbols, something requiring very little code and no extra API endpoints). Figure 9 shows where the extra functionality is integrated into the program and where the compiler modifications would occur.

© FitOpTiVis Consortium (Public Document)

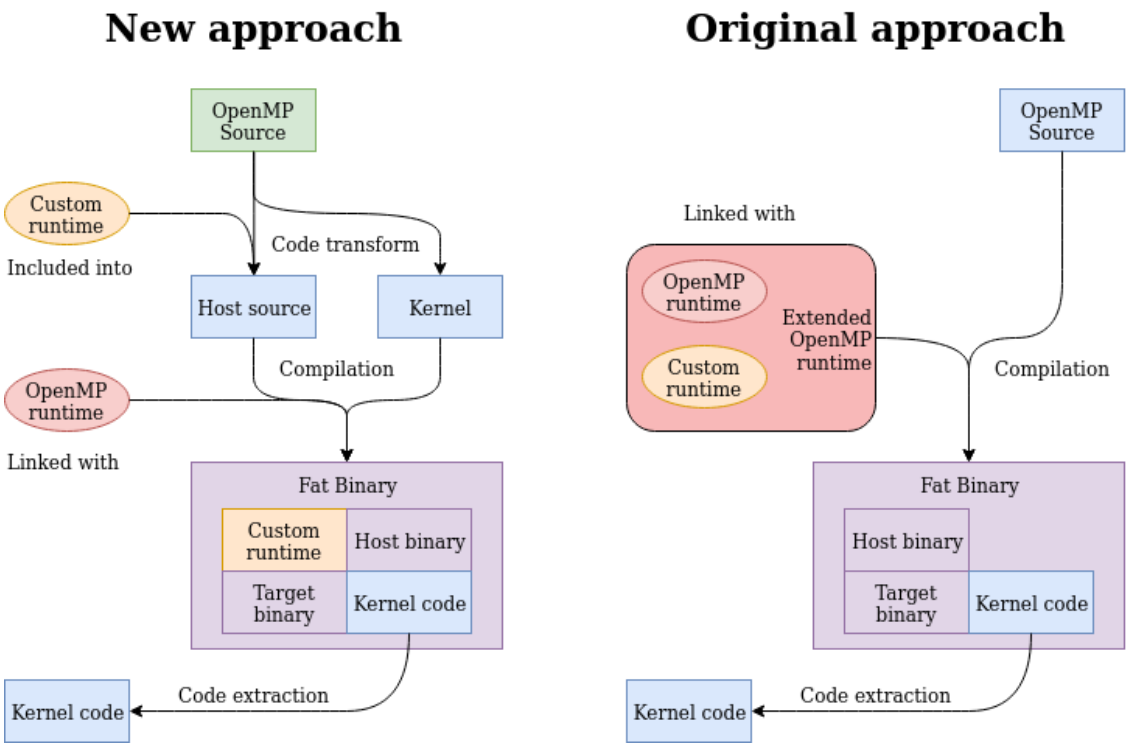


Figure 10. Comparison between the improved and the original approach to extending OpenMP

4.4 The CompSOC Platform

The CompSOC platform offers a Virtual Execution Platform (VEP) to each application. VEPs are entirely isolated from each other (space, e.g. memory, and time, e.g. TDM on processors or network-on-chip), such that each application can use its own Model of Computation and can be developed independently. This section is a summary of the platform description presented in [GOO17].

4.4.1 Hardware Architecture

MPSoCs contain multiple processors with local and shared memories. The processor's local memories are always on-chip Static Random-Access Memory (SRAM), close to the processor. Nonlocal memories shared between processors may be on-chip SRAM but often include off-chip Dynamic Random-Access Memory (DRAM). The latter has a much larger capacity (number of bits) than the on-chip memory, but at the cost of a longer execution time. Processors reach shared memories using a communication infrastructure, which is increasingly a NoC. A NoC is a miniature version of the Internet in the sense that communication is concurrent, is distributed, and is either packet based or circuit switched. As a result, it can run multiple applications of different criticalities at the same time. The CompSOC platform consists of multiple tiles interconnected by a NoC. Tile types are master tiles, slave tiles, or a mix of both, and include processor tiles, memory tiles, peripheral tiles, etc.

© FitOpTiVis Consortium (Public Document)

4.4.2 Software Architecture

The CompSOC hardware platform contains computation, communication, and storage resources. Almost all can be shared between multiple requestors, and almost all can be (re)programmed at run time. The CompSOC software extends the single hardware platform to offer multiple Virtual Execution Platforms (VEPs). A VEP is an execution platform that is a subset of the CompSOC hardware platform, in terms of time (e.g., time multiplexing a processor) or space (e.g., non-shared DMA or a region in memory). Each application runs in its own VEP, which is created, loaded, started, and possibly stopped and deleted, at run time. A CompSOC platform can run multiple VEPs concurrently, without any interference between them, i.e., composably.

4.4.3 Microkernel and RTOS

Task arbitration can be classified along several axes. First, it may be absent when there is only one task on a resource. Otherwise it is required. Second, it may be preemptive or not. Third, arbitration may be static and follow a static-order schedule or be dynamic where the order of tasks is determined at run time. Multiple applications can share the processor using a microkernel such as CoMik, which arbitrates only between applications. Each application can use virtualized RTOS, such as µC-OS III, to independently arbitrate between application tasks.

An example CompSOC platform is shown in Figure 10 [GOO17].

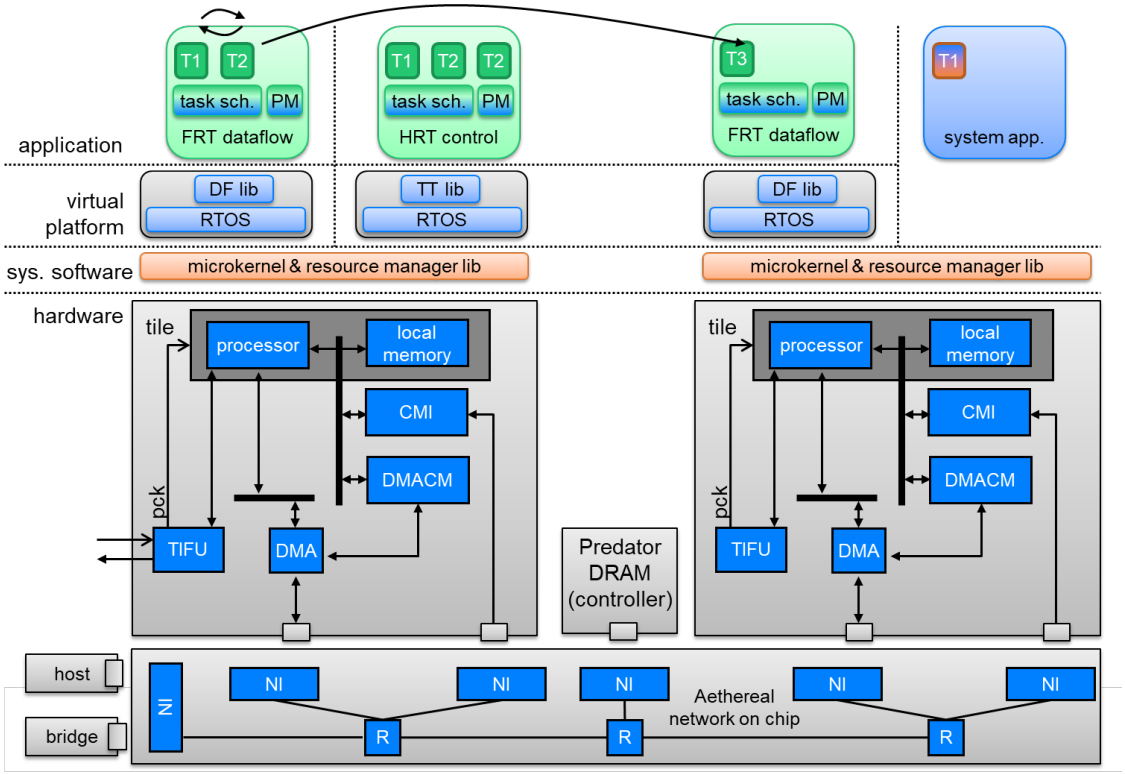


Figure 11. An example CompSOC platform.

4.4.4 FitOptiVis QRM Framework on CompSOC

The above CompSOC platform will be used to implement quality and resource management (QRM) framework envisioned in FitOptiVis. This requires several further developments including the dynamic reconfiguration mechanism and budget handling in line with what is developed in WP2. Further details of these platform adaptations are provided in Deliverable D4.3.

4.5 The Xilinx Zynq Platform

In contrast to the predecessor ALMARVI project, which only provided support for standalone boards and no board-to-board communication, the FitOptiVis project focuses on providing Peta Linux and Debian OS support, as well as enabling board-to-board communication in a local cloud.

The first version of design time and runtime support for the family of Xilinx Zynq and Zynq UltraScale+ systems has been developed by WP4 partners and released for use by project partners and general public by the end of April 2019. The new runtime provides support for Ethernet-based board-to-board communication in the local cloud, utilizing the Arrowhead framework, which is compatible with C/C++ clients running on ARM processors.

The following Xilinx Zynq systems are supported:

ZynqBerry (small). A small-size, low cost system with design time support developed in FitOptiVis. It has the Raspberry form factor and utilizes a 32bit Xilinx Zynq device (28nm) with small programmable logic area. WP4 provides support for Arrowhead-based board-to-board communication, Debian OS, and 32bit C/C++ clients. See [KAD18a], [TE0726], and [ARROW] for details.

Zynq UltraScale+ (medium). A medium-size system with design time support developed in FitOptiVis. Utilizes a 64bit Xilinx Zynq device (16nm) and reuses the carrier board and the Full HD video I/O FMC card from the ALMARVI project. WP4 provides support for Arrowhead-based board-to-board communication, 64bit Debian OS, and 64bit C/C++ clients. See [KAD18a], [KAD18b], [ARROW], [TE0820], and [TE0701] for details.

Zynq UltraScale+ (large). A large-size system with design time support developed in the FitOptiVis. The carrier board has the Mini-ITX form factor, utilizes a 64bit Xilinx Zynq device (16nm), and reuses the Full-HD video I/O FMC card from the ALMARVI project. WP4 provides support for Arrowhead-based board-to-board communication, 64bit Debian OS, and 64bit C/C++ clients. See [KAD18a], [KAD18c], [TE0820], [TE0808], and [TE080X] for details.

4.6 Deterministic Networking Platform

Time Sensitive Networking is a set of IEEE 802 standards providing reliability and determinism in Ethernet networks, which is required by time-sensitive applications. TSN enables mixed-criticality communication by allowing real-time and best-effort traffic to coexist on the same network infrastructure.

© FitOpTiVis Consortium (Public Document)

TSN is committed to standards that are fully integrated into the Ethernet protocol stack, as virtually all the functionality belongs to the IEEE 802.1 bridge layer—except the frame pre-emption capability, which is developed on top of the IEEE 802.3 MAC layer.

Deterministic end-to-end latency is supported by three key capabilities. Firstly, a time synchronization protocol (providing accuracy in the range of 50 ns) enables precise coordination between different elements in the network. Secondly, known and bounded network latencies given by link propagation delays and switch forwarding delays. And last, but not least, the isolation of critical and non-critical traffics.

4.6.1 TSN bridge design and implementation

The TSN bridge implementation within FitOpTiVis needs to provide the functionality required by the use-case requirements, which can be summarized as follows:

1. A well-known high-speed interface: Ethernet 100/1000-Base-T
2. Mixed-criticality communication between distributed processing nodes, providing deterministic services: zero packet-loss for congestion, bounded latency, and deterministic delivery for end-to-end synchronized communications.
3. Time Synchronization to facilitate a common time base for all monitoring information retrieved from heterogeneous and distributed elements. The synchronization is also required for coherent co-processing of remote nodes, especially for distributed hard real-time applications (i.e. Smart Grid).

To satisfy the requirements, the TSN bridge implements the following standards:

- **IEEE 802.1Q VLAN switching**. This standard defines the mechanisms enabling the coexistence of mixed-critical traffics over the same Ethernet network. Tagged Ethernet provides differentiation and prioritization through VLAN identification (VID) and Priority Code Point (PCP) fields. Each TSN bridge can handle up to 16 different traffic types (VID) classified in up to 4 different priorities (PCP).
- **IEEE 802.1AS gPTP**. The generalized Precision Time Protocol (gPTP) is a time synchronization protocol suitable for TSN, because it can achieve synchronization accuracy in the order of tens of nanoseconds. It supports fast failover by means of re-election of the time reference or Grandmaster (Best Master Clock Algorithm), and by processing redundant time synchronization messages (passive port role). Besides, gPTP continuously monitors the links conforming to TSN and reports key metrics such as link propagation delay, neighbour status, current time reference, and the synchronization tree (i.e. the path to the time reference or Grandmaster).
- **IEEE 802.1Qbv**. The Time-Aware Traffic Shaper (TAS) performs priority-based queuing and strict time-driven transmission scheduling based on PCP traffic priorities.

A Xilinx Zynq-7000-based platform is used to implement a 4-interface TSN bridge. This MPSoC provides an ARMv9 processing system and programmable logic. The functional architecture is shown in Figure 16. The functionality requiring deterministic behaviour, i.e., traffic switching, scheduling, and shaping, as well as timestamping and synchronization servo, are implemented in the programmable logic. The gPTP state

© FitOpTiVis Consortium (Public Document)

machines and the tasks responsible for configuration and run-time monitoring through the TSN user API are implemented in software executing on the ARM core.

The architecture of the TSN bridge consists of two main blocks:

- The networking component, which provides 1000 Base-T Ethernet connection, traffic differentiation and prioritization, in addition to priority-based, time-driven, strict arbitration of the output bandwidth. The blue modules, i.e., the redirector, the VLAN tagger (and untagger), and the Time-Aware traffic Shaper (TAS) implement the IEEE 802.1Q functionality, while the green modules, i.e., MAC, PHY, and DMA are off-the-shelf Xilinx IP-cores implementing standard IEEE 802.3 functionality. A Linux network driver is provided for this particular gateware.
- The timing component, which provides the IEEE 802.1AS functionality, i.e. highly accurate time synchronization between all TSN stations in the network. This component (orange modules) consists of a gPTP cyclic executive running on the PS and a PTP hardware clock, supported by Time Stamp Units (1G TSU's), present on each gPTP-capable interface.

Note that the TAS mechanism alone does not prevent interference between time-critical messages and lower-priority jumbo frames found in video streaming applications. This issue can be addressed by considering guard bands on the output bandwidth cyclic schedule, at the cost of available bandwidth. To avoid potential RT-QoS violations and to optimise bandwidth usage, a frame pre-emption mechanism is being considered. Frame pre-emption is a MAC sublayer enhancement described in IEEE 802.3BR that stops lower priority frame transmissions whenever TAS notices that a time-critical message should be transferred.

© FitOpTiVis Consortium (Public Document)

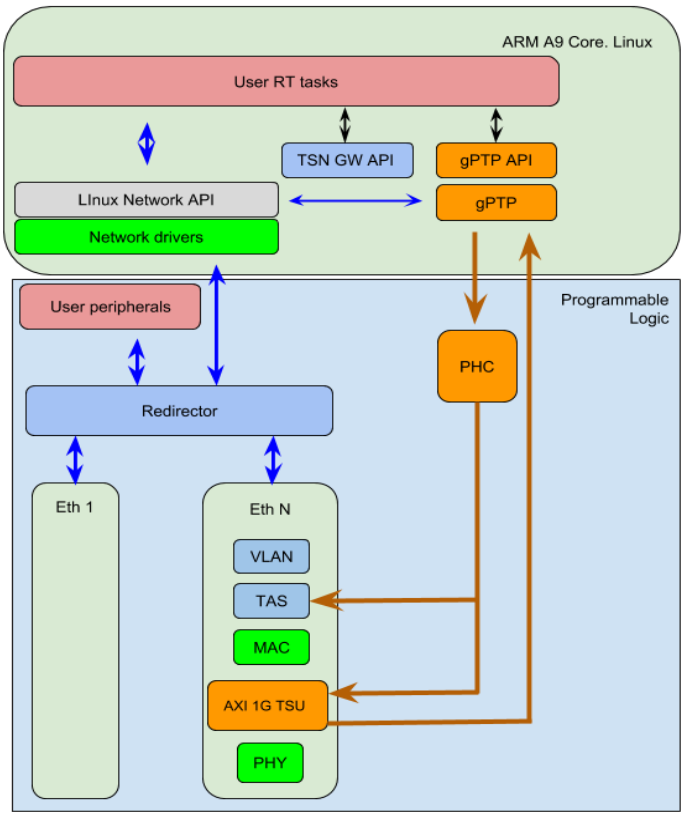


Figure 12. The architecture of the TSN bridge.

4.6.2 Modelling TSN as a platform component

This section uses the FitOpTiVis DSL defined in WP2 to provide a model of TSN as a platform component for hosting time-sensitive applications. An overview of the model is shown in Figure 17.

4.6.2.1 Application components:

Time Sensitive Application

This component represents different time-sensitive applications, implemented on top of different interconnected systems. Different time-sensitive applications typically demand connectivity with different Quality of Service (QoS), ranging from best-effort to time-critical, which is required for time synchronization or distributed hard real-time applications.

The time-sensitive application may also require time synchronization, either for time-triggered messaging or to coordinate the co-processing among distributed nodes.

4.6.2.2 Virtual execution platform

VLAN IEEE 802.1Q

This component represents the switching capability of the TSN network, attending to the traffic type and the corresponding QoS. The VLAN component provides mixed-critical

© FitOpTiVis Consortium (Public Document)

traffic support and requires strict traffic scheduling from the Time-Aware Traffic Shaper (IEEE 802.1Qbv) component.

Parameters

- **Configuration values**: Protocol field pattern for different kinds of incoming traffic which should be encapsulated into VLAN frames.
- **Translation rule**: VLAN tag to be applied, given by the VLAN VID and the VLAN PCP fields. The VLAN module supports up to 16 different traffic types and 4 different priorities.

gPTP IEEE 802.1AS

The gPTP component encapsulates the time synchronization capability of the TSN. gPTP provides the synchronization required by time-sensitive applications and the Time-Aware Traffic Shaper. The gPTP demands the maximum time-criticality from the VLAN module to keep all the networks synchronized. The IEEE 802.1AS standard defines run-time parameters and monitors which have been reflected in the DSL model.

Parameters

- **Active Interfaces**: The Zynq-7000 based platform provides four RJ-45 Ethernet interfaces. This list specifies in which one gPTP is available.
- **OperAnnounce**: Run-time message periodicity for the messages providing information about the Grandmaster and synchronization tree. The announce messages are key requirement of the Best Master Clock Algorithm.
- **OperSync**: Run-time message periodicity for the messages carrying the synchronization information.
- **OperPdelay**: Run-time message periodicity for the messages supporting the Peer to Peer delay mechanism. This mechanism is responsible for continuous monitoring of the link, the corresponding propagation delay and the remote peer status.
- **prio1**, **prio2**: These parameters control the eligibility of the current node to serve as the Grandmaster or time reference.

Qualities

- **Adjustment**: Current time drift between the Grandmaster and local-clock time.
- **Servo status**: The status of the servo performing the local clock control.
- **grandMaster**: The current time reference.
- **pathTrace**: List of the stations traversed by the synchronization messages.
- **steps_removed**: Number of stations in the pathTrace.
- **rateRatio**: the ratio between the frequencies of the local clock and the Grandmaster clock.
- **asCapable**: Remote peer capability status.
- **Role**: Current functionality mode of each interface. One of |*Master*, *Slave*, *Passive* or *Disabled.*
- **Link Delay**: Link propagation delay refreshed by the Peer-to-Peer delay mechanism.
- **Link Status**: Link status reported by the PHY.

© FitOpTiVis Consortium (Public Document)

TAS IEEE 802.1Qbv

The Time-Aware Traffic Shaper (TAS) provides the strict traffic scheduling required to isolate the different kinds of traffic and satisfy latency and bandwidth requirements. The TAS requires the bandwidth and connectivity provided by the IEEE 802.3 MAC layer and the time synchronization provided by the IEEE 802.1AS gPTP module.

Attending to the IEEE 802.1Qbv standard, the TAS requires to be configured with the base time and the scheduling table. On the one hand, the base time allows the time alignment provided the propagation delay along the TSN stream path. On the other hand, each entry of the scheduling table contains the parameters characterizing each interval of the cyclic schedule.

Parameters

- **Base Time**. POSIX `timespec` structure (seconds and nanoseconds) indicating the system time after which the cyclic scheduling is executed. Before this time, time-aware gates remain open.
- **Number of intervals**. The number of entries in the scheduling table.
- **Tick granularity of the time schedule**. Possible values are 1, 2, 4 or 8 ns.
- **Interval time**. Execution interval time length.
- **Gate configuration list**. Boolean array indicating which priority queue is opened or closed at a given interval.

4.6.2.3 Execution platform

100/1000 Base-T

This component represents the lowest layers of the Ethernet protocol stack. They provide the required connectivity and bandwidth between network elements.

© FitOpTiVis Consortium (Public Document)

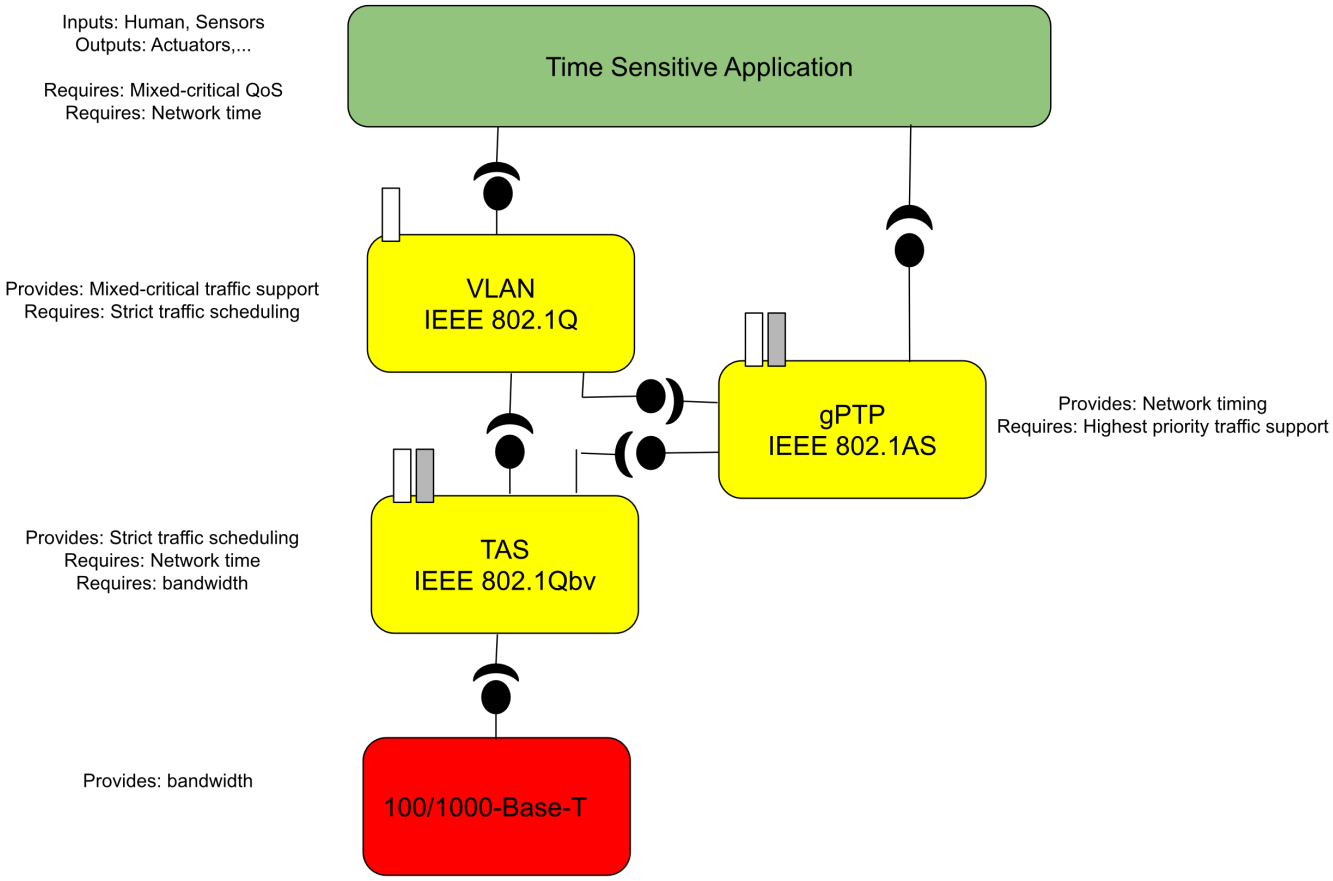


Figure 13. DSL model of the TSN platform component

4.6.3 Application in Context of UC3 (Habit Tracking)

In UC3 (Habit Tracking), gesture recognition, person tracking, augmented reality, and biometric sensors cooperate to provide a senior-friendly smart home. This scenario relies on heterogeneous devices almost all of which require 100/1000-Base-T connectivity (except for wearable sensors, which require wireless Ethernet connectivity). For the preliminary demonstrator, the following end stations have been considered:

- **Central station**. The central station collects monitoring data from the different systems. Besides, it provides co-processing capability for gesture recognition and person tracking.
- **Nvidia-based GPU platforms**. These platforms (Jetson TX2, Xavier) are used to manage CCTV cameras and perform processing in the edge for gesture recognition and person tracking.
- **Augmented reality glasses**. This wearable device is likely to be connected via a wireless Ethernet connection.
- **Biometric sensors**. Biometric sensors are connected via wireless connections and via smart phones.

© FitOpTiVis Consortium (Public Document)

Biometric sensors and augmented reality require wireless connectivity. However, because TSN is based on point-to-point links, an IPv4 tunnelling will be implemented over the wireless connectivity to adhere to TSN requirements. For the rest of devices, a 100/1000-Base-T interface will be provided.

Gesture recognition and person tracking systems require coherent processing between edge devices (Nvidia GPU platforms) and the central station. Biometric sensors generate monitoring signals which are registered and presented on the central station. Augmented reality sunglasses may show a low-quality video streaming sourced by the central station when an event is detected.

- Consequently, time synchronization is required to facilitate co-processing and coherent monitoring of the distributed systems. Furthermore, each system generates one or several heterogeneous data streams:
- Video streaming of different qualities and, hence, with changing bandwidth requirements. Video streaming is generated by gesture recognition, person tracking and augmented reality systems.
- Control streaming required by person tracking and gesture recognition for coherent processing on edge and cloud (central station).
- Low bandwidth data generated by wireless, biometric sensors
- Monitoring traffic, generated by all the systems participating in the use case, and collected on the central station.

In this context the TSN should provide a common infrastructure to facilitate the cooperation between these heterogeneous systems, by providing a common time reference. At the same time, it should provide isolation between different kinds of traffic, and provide zero-congestion, zero-loss, or bandwidth guarantee for control traffic.

4.6.4 Application in Context of UC9 (Surveillance of smart-grid critical infrastructure)

In UC9, TSN should provide connectivity to the distributed elements comprising the smart-grid and surveillance subsystems and facilitate cooperation between these subsystems.

Again, TSN provides the well-established 100/1000-Base-T Ethernet interface to interconnect equipment from different vendors:

- The Remote Terminal Unit (RTU) controls circuit breakers and disconnect switches of the electrical substation. It can be connected to other RTU's to conform a HSR ring or directly to the TSN.
- The HSR Redbox provides gateway between the HSR and TSN network.
- Nvidia-based GPU platforms performing CCTV camera control and edge processing for surveillance purposes.

Specifically, the following kinds of traffic have been identified:

- Time-critical traffic between Remote Terminal Units for the Smart Grid system.
- Control streaming between edge and cloud processing nodes.
- Video streaming of different qualities generated by surveillance processing nodes at the edge.
- Monitoring traffic generated by the different equipment and presented on central stations.

© FitOpTiVis Consortium (Public Document)

To this end, time synchronization should be provided to talker and listener nodes. Besides, time synchronization is also required for the coordination between distributed co-processing nodes of the surveillance system, and to facilitate the cooperation between surveillance and smart grid. Last, but not least, the common time allows correlating monitoring data from different sources collected on remote central stations.

© FitOpTiVis Consortium (Public Document)

5. Runtime Adaptation

To manage trade-offs between different aspects of quality (e.g., frame resolution, quality, frame rate or latency) and resource usage (e.g., CPU time, memory usage, I/O bandwidth, or energy), the runtime platforms need to be able to modify configurable parameters in response to desired quality set points and changing conditions.

In this chapter, we review the developed mechanisms for runtime reconfiguration and resource management, and introduce some of the algorithms and techniques envisioned to achieve the desired trade-offs between quality (performance) and resource usage for selected systems. The latter part of the chapter includes partner descriptions of runtime adaptation scenarios in use case-specific applications and contexts to serve as scenario descriptions for guiding the development for the duration of the project.

5.1 Reconfiguration in Managed-Latency Edge-Cloud

At the highest level of abstraction, the managed-latency edge-cloud infrastructure implements a MAPE-K self-adaptation loop [KEP03] (shown in Figure 18) to ensure that application requirements will be satisfied even in face of continuously changing conditions. To this end, the infrastructure periodically checks whether the soft real-time requirements are met and predicts near-future development. This allows the system not only to intervene after detecting a violation of application requirements, but also to act proactively ahead of time if needed.

A single adaptation loop is used to manage both the initial deployment as well as redeployment of microservices. In fact, redeployment is nearly identical to initial deployment—calculation of real-time requirements is done periodically and takes into account the current placement of microservices to prevent unnecessary relocations.

Each phase of the control loop has a distinct responsibility:

- **Monitoring**. The monitoring phase is responsible for keeping the internal model of the system up-to-date. In the context of the edge-cloud platform, the controller monitors the state of the K8S cloud (nodes, pods, and other entities such as services and deployments) as well as the state and performance of individual applications, e.g., how often.
- **Analysis**. The analysis phase is responsible for finding a deployment configuration (an assignment of application components to nodes in the cloud) that satisfies performance guarantees. A Constraint Satisfaction Problem (CSP) solver is used to find feasible solutions (in which timing requirements can be expected to hold), while the controller is responsible for evaluating the feasible solutions and choosing from among them.
- **Planning**. In the planning phase, the controller determines if the desired configuration differs from the actual configuration and if necessary, prepares a sequence of actions to bring the cloud to the desired state.
- **Execution**. In the execution phase, the controller makes actual changes to the cloud platform, following the plan of actions produced in the planning phase. In many cases, the actions can be executed in parallel, except when there are explicit precedence constraints among tasks.

The four phases execute simultaneously, sharing data through a central **knowledge** component. In its simplest form, the knowledge component can be represented by a

© FitOpTiVis Consortium (Public Document)

single centralized database. However, it is entirely possible for the knowledge component to interface with several storage back-ends that can be used for different purposes. As an example, we can consider the FIVIS data storage, analysis, and visualization platform (developed in the context of Task 4.2) to serve as the knowledge component.

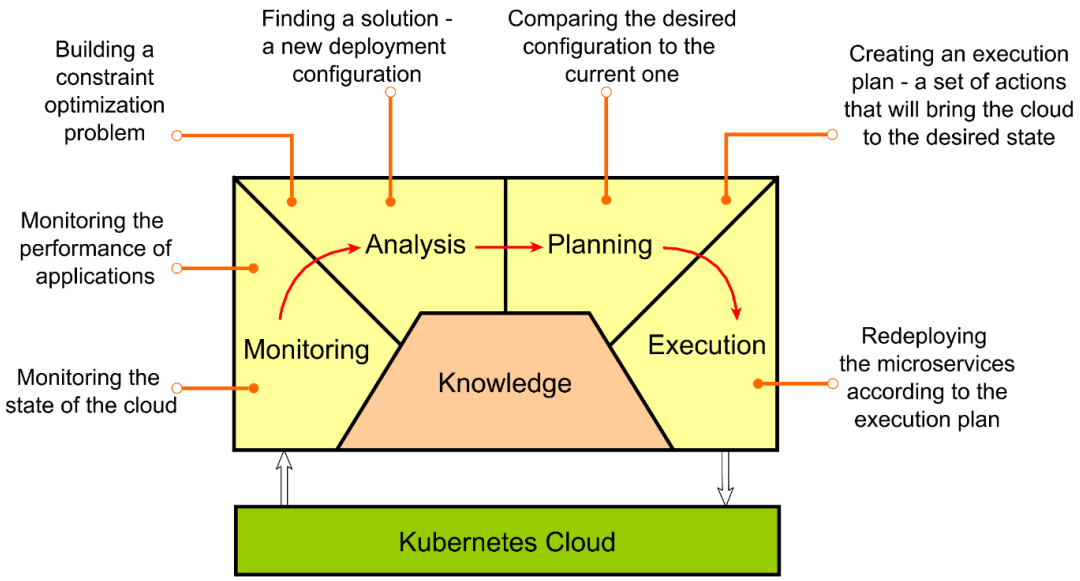


Figure 14. Self-adaptation loop of the managed-latency edge-cloud platform.

Note that this control loop applies only to management of latency in the edge-cloud platform. FitOptiVis systems in the role of edge-cloud applications will implement application-specific higher-level (higher-latency) control loops responsible for configuring the set-points (e.g., resource limits, desired framerate) for a lower-level (low-latency) control loop responsible for achieving the desired set-points on the hardware components.

5.1.1 Edge-Cloud Platform Architecture

The architecture of the edge-cloud platform shown in Figure 19 comprises a number of modules, each with distinct responsibilities in the control loop. Yellow modules (need to) run on the master node, green modules do not (need to) run on the master node, and blue modules represent a middleware layer. We now elaborate on the role of individual modules and their interaction with other modules:

- **Event Cache**. The module is responsible for persistent storage of important events, such as changes in application deployment (requests to deploy or undeploy an application) and connections from unmanaged components. Unmanaged components execute outside the edge-cloud platform (e.g., a hardware accelerator) and connect (as clients) to the managed components executing in the cloud.
- **Knowledge**. Provides data storage and query capabilities to modules directly responsible for implementing the MAPE-K control loop. Knowledge data

generally concerns cloud nodes (and their subtypes), application types and instances, and component types and instances.

- **Cloud Monitor**. Implements the monitoring phase of the MAPE-K control loop by periodically collecting information about the state of the nodes in the cloud, network latencies, and unmanaged components.
- **Analyzer**. Implements the analysis phase of the MAPE-K control loop and is responsible for finding an application deployment plan that satisfies the timing requirements of all deployed applications. The module is internally split into Solver and Predictor submodules.
    - **Solver**. Responsible for finding the best deployment plan within a given time limit. Takes into account node utilization, network latencies, and predictions of component performance in deployment scenarios considered.
    - **Predictor**. Predicts performance of managed components, taking into account the hardware they are running on and the load induced by other components running on the same hardware.
- **Planner**. Implements the planning phase of the MAPE-K control loop, which means identifying differences between the current application deployment and the desired deployment. Constructs an ordered execution plan of tasks that need to be executed to transition the system to the next state.
- **Cloud Executor.** Implements the execution phase of the MAPE-K control loop by executing planned tasks either on the Kubernetes cloud, or on the other (Managed and Unmanaged) controllers.
- **Managed Controller**. Responsible for invoking probes on managed components and for reconnecting dependencies of managed component instances. Can access all Node Controllers at runtime.
- **Unmanaged Controller**. Responsible for reconnecting dependencies of unmanaged component instances from one managed instance to another, invoking probes on the client (which invoke managed components) to observe managed component performance including communication latency, and monitoring the state of unmanaged components.
- **Node Controller**. Runs on each node and monitors the utilization of a particular node and of all the components executing on that node (using standard K8S facilities for resource monitoring). In addition, it serves as a proxy to managed component instances for the Managed Controller.
- **Probe Controller**. Serves as a central entity through which all requests for probe invocation (on Managed and Unmanaged components) have to pass. Caches and forwards the results of probe invocations.
- **Network Controller**. Responsible for making changes in network configuration and for collecting network utilization data and connection latencies.

On each node, the information about a microservice obtained during the assessment phase is used to assign each deployed microservice the resources needed to perform its tasks within the timing constraints. This resource allocation is strictly enforced using features of the operating system, containerization technology, or the virtualization platform. Specifically, we rely on resource allocation features of Docker and Linux cgroups. This is necessary to prevent microservices from exceeding their allocated share of resources (due to, e.g., a sudden spike in the number of clients), which could have a negative impact on the execution time of other microservices.

© FitOpTiVis Consortium (Public Document)

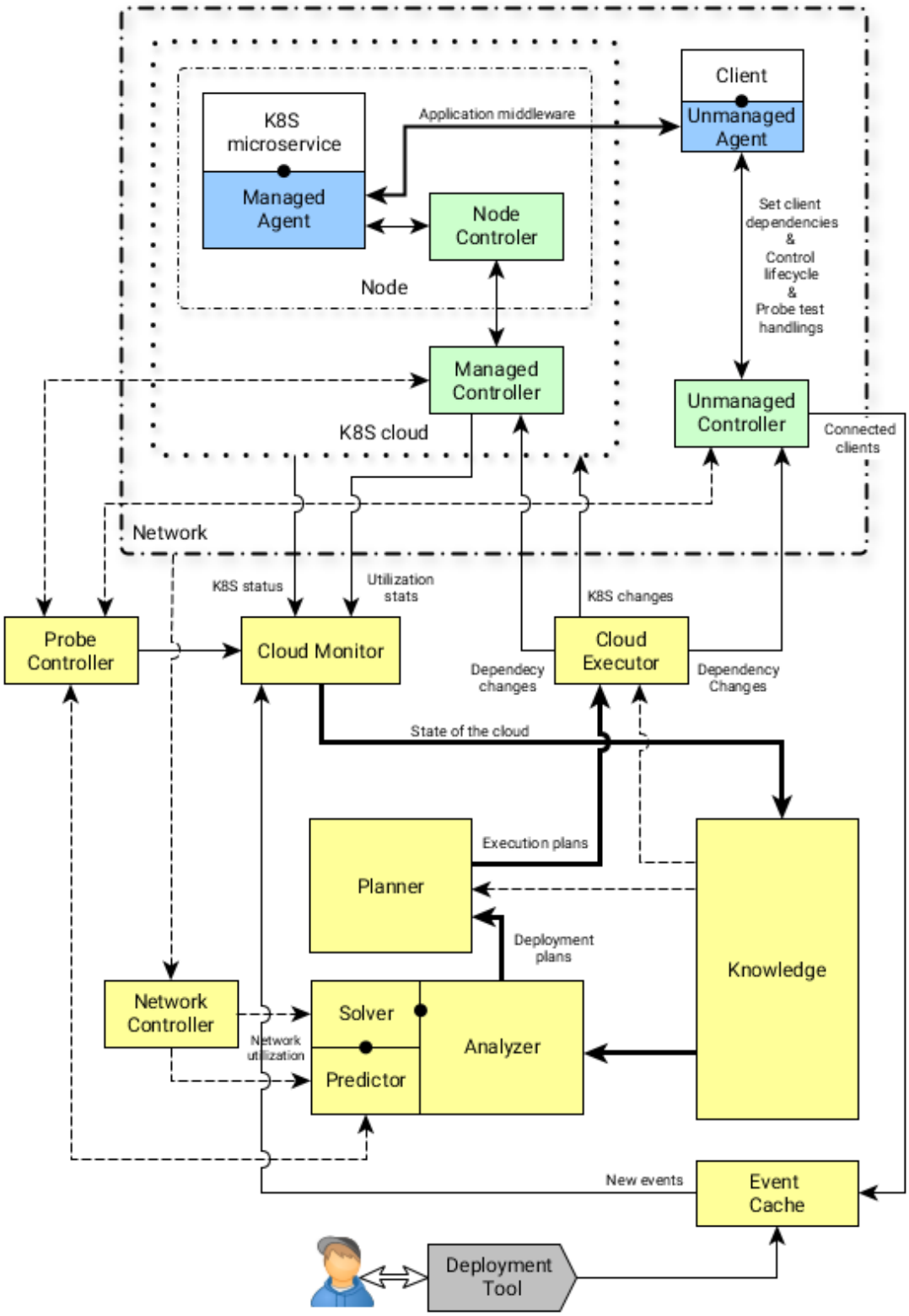


Figure 15. Architecture of the managed-latency edge-cloud platform.

5.1.2 Performance and Interference Models

To adaptively control deployment and redeployment of components in edge-cloud and thus to probabilistically guarantee end-to-end response time, the platform needs to build a model of application performance. This model needs to capture several modes of execution: baseline performance, when the application is exercised in isolation, performance under constrained resources, and performance in presence of other co-located applications sharing the physical hardware through virtualization.

Because we do not require the developer to provide the platform with apriori knowledge about application performance and resource requirements, the cloud platform needs to build the application performance model using experimental evaluation.

The model then is used to predict application performance in different situations, especially during admission control (when deploying a new application), and when optimizing the deployment of existing applications (to ensure that real-time guarantees are met, or to manage the utilization of cloud resources).

An important aspect of performance that the cloud platform needs to take into account is performance interference on shared resources (CPU caches, memory and IO bandwidth, etc.) when co-locating multiple virtual machines and/or containers on the same physical machine.

On the other hand, we generally consider the underlying network bandwidth unlimited for modelling purposes. The rationale behind this assumption is that edge-cloud applications are likely to be latency-sensitive, but not necessarily bandwidth-intensive—that would defeat the primary purpose of edge-cloud, which is to reduce communication latencies due to distance.

We also assume that edge-cloud infrastructure can generally be private, i.e., with significant level of control (like in hospital use cases). Consequently, we assume that the network infrastructure can be configured to assign time-critical network traffic a QoS class with high priority; that latency-sensitive services with guaranteed response time requirements will not saturate the network with bulk transfers; and that applications with excessive bandwidth requirements can be dealt with by proper network infrastructure design. In particular, if latency-sensitive traffic needs to coexist with bulk traffic on the same network infrastructure, we assume that solutions based on Time-Sensitive Networking will be used (see Section 4.6).

5.1.3 Performance Prediction of Co-located Workloads

One of the key responsibilities of the Analyzer module (see platform architecture in Figure 19) is finding and analysing deployment alternatives. The analysis primarily concerns application performance prediction, providing the adaptation controller with data for making decisions—both when considering an application for admission as well as when reacting to violation of application's timing requirements.

The Predictor part of the Analyzer module uses a novel performance prediction algorithm which is based on statistical characterization of application performance measurements followed by a similarity comparison, revealing performance dependencies between background workloads (i.e., microservices).

We first use performance measurements to build a structured data set and the, whenever a performance prediction of a particular scenario is needed, the relevant

© FitOpTiVis Consortium (Public Document)

prediction data are extracted into a linearized data-fitting model. This model is then solved by a constrained least-squares method, giving a reliable order statistics estimate of application performance, including its fidelity.

To build the initial data set, we perform a number of measurements for a number of scenarios involving one or more workloads. There is always a scenario in which each workloads executes in isolation, without any other workloads running in the background. For each workload, we also include various combinations of background workloads. Because this may quickly become computationally infeasible, we generally focus on collecting information for pairs of co-located workloads, which reveals first-order performance impact, i.e., how applications influence each other on given hardware platform. Scenarios involving three or more workloads are sampled depending on available resources.

For each scenario, we collect measurements on a number of parameters which characterize the application behaviour. In addition to response time, this includes CPU utilization, number of I/O operations, and memory utilization. To ensure robustness of the predictor, each scenario is measured multiple times to properly sample the influence of factors that can influence the measured parameters, but are beyond our control, such as virtual memory layout, file system state, or just-in-time compilation. With the initial data collected, we can start predicting application performance in different scenarios.

The prediction algorithm consists of three phases, and is summarized in the schema shown in Figure 20 below. Here we discuss the individual phases in more detail:

1. **Data pre-processing**. The first phase represents all computations that can be performed apriori to save the computational costs in later phases. The goal is to compute a number of statistical characteristics (for each of the given scenarios) in order to capture dependencies of all parameters of interest on the measurement conditions. This includes information about statistical distribution of the measurements, i.e., the sample mean and median, selected sample percentiles, standard and relative deviations, standard error, and the difference between the sample maximum and minimum values.
   While the characteristics such as mean or median capture typical behaviour, the sample maximum and minimum capture information about extremes. The difference between the typical and extreme behaviour is used to effectively penalize measurements with lower fidelity, improving performance prediction reliability.
   We also compute various quantities that allow revealing dependencies between performances of different workloads. In particular, these include slow-down parameters corresponding to the difference between sample percentiles of measured parameters for cases when a workload executes in isolation and when it executes together with other workloads.
2. **Task fitting**. Given the initial data, their statistical characterization, and a user-specific prediction requirement (i.e., a question), we first need to detect precomputed scenarios relevant for the prediction. We allow two types of scenario questions:
   - Q1: performance prediction for one of the already tested workloads, $\mathbf{W_i}$.
   - Q2: performance prediction for a new workload, $\mathbf{W_{n+1}}$, for which we have data measured in isolation.

© FitOpTiVis Consortium (Public Document)

The situation is simpler for Q1. The prediction must be based on the statistical characteristics of the scenarios involving $\mathbf{W_i}$. We therefore build a prediction model using all scenarios involving $\mathbf{W_i}$, except the one in which $\mathbf{W_i}$ executes in isolation. This gives us a data fitting problem, modelling the unknown correlation between the question and the preselected initial scenarios, which we then solve using the constrained least-squares method with non-negative constraints (NLS).
For Q2, the prediction is based on finding an existing workload $\mathbf{W_j}$ that most closely resembles the new workload $\mathbf{W_{n+1}}$. To find such a workload, we first compute the statistical workload characterization (see phase 1) for the scenario in which $\mathbf{W_{n+1}}$ executes in isolation and compare it to characterizations of other workloads executing in isolation. Using some similarity measure, e.g., a weighted vector norms of the difference between mean, median, and deviation for the most relevant measured parameters, we look for the lowest difference (best match), producing $\mathbf{W_j}$. Finally, we incorporate the statistical characterization of $\mathbf{W_{n+1}}$ in the data set and "rephrase" Q2 as Q1 with $\mathbf{W_j}$ in the role of $\mathbf{W_i}$ serving as a proxy for the new workload $\mathbf{W_{n+1}}$.

3. **Data-based prediction**. In the last phase, we use a weighted combination of workload dependencies to predict the behaviour in the scenario from Q1 or Q2. Specifically, we estimate percentiles of expected performance of $\mathbf{W_i}$ in Q1 by shifting the percentile observed for $\mathbf{W_i}$ executing in isolation by a linear combination of estimated weighted slowdowns.

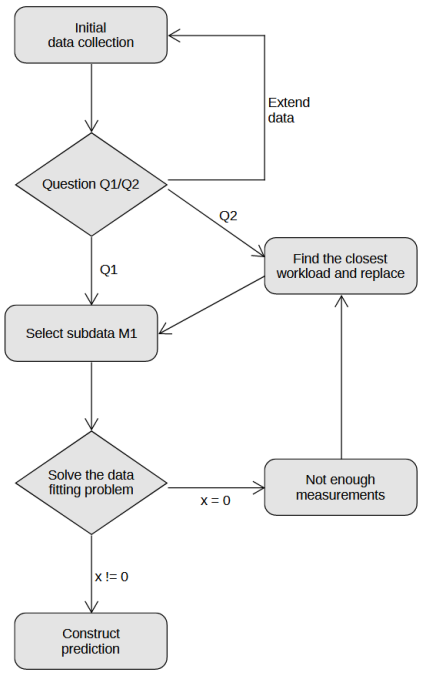


Figure 16. Overview of the performance prediction algorithm.

The interactions among co-located microservices sharing the underlying physical resources are generally complex, and often non-linear—especially when the physical resources are nearing exhaustion. Consequently, the prediction accuracy varies with

different combinations of applications and resources used, and cannot provide actionable results for all possible scenarios.

To ensure that the predictor can be used with confidence within the adaptation loop, it is critical to establish the predictor's operational boundaries and ensure that the managed system stays within the boundaries. The boundaries can be expressed as limits on the utilization of the CPU, memory, and IO resources used to characterize the workloads.

While our system currently does not support automatic discovery of the operational boundaries, our initial evaluation indicates that they could be established experimentally for a particular platform. We expect that this could be turned into an automated procedure.

The work presented here is currently under review in a scientific journal.

5.1.4 State of the Art

Cloud computing has been both a blessing and a curse. Cloud users can benefit from unprecedented availability and elasticity of resources, but the benefits come with strings attached. Cloud platforms have to continually balance the tension between efficient resource utilization (which determines costs) on the one hand, and quality-of-service guarantees demanded by latency-sensitive (LS) applications on the other hand.

Management of cloud resources has therefore become a vast and quickly moving research area, with many surveys mapping and categorizing the problems, challenges, and the state-of-the-art in various problem domains [CHE18, AMI17, HAM16, SIN15, FAN15, MAN15, GAR14]. In the context of our work we focus primarily on approaches to performance- and interference-aware self-adaptive systems which manage resource allocation and assignment in a cloud environment to achieve efficient utilization of available resources while allowing applications to meet their QoS target.

Q-Clouds [NAT10] is a QoS-aware control framework which transparently adjusts resource allocation to mitigate effects of interference on shared resources. Q-Cloud first profiles the virtual machines (VM) submitted by clients on a staging server to assess the amount of resources needed to attain the desired QoS without interference, and then manages the resources allocated to the deployed VMs in a closed control loop.

Cuanta [GOV11] is a technique for predicting performance degradation due to shard processor cache for any possible placement using a linear (as opposed to exponential) number of measurements. Applications are replaced by a synthetic clone which is tuned to mimic the application's cache pressure, and interference due to colocation is predicted based on a matrix of know interference effects between different configurations of cache clones. Even though Cuanta is not a full-fledged cloud scheduler, it was used to make better workload placement decisions for a given performance and resource constraints.

Bubble-Up [MAR11] avoids pairwise colocation profiling by characterizing the QoS degradation in LS applications using a synthetic workload with configurable memory subsystem stress test (the bubble), and the contentiousness of batch applications using a reporter workload with known sensitivity curve. The contentiousness of a batch application is mapped to a configuration of the bubble, which is then used to predict the interference inflicted by the batch application on the LS application.

© FitOpTiVis Consortium (Public Document)

Bubble-Flux [YAN13] improves on Bubble-Up by performing online profiling for LS workloads to account for workload phase changes and to identify more colocation opportunities.

Paragon [DEL13] is an online interference-aware scheduler, which uses collaborative filtering to classify incoming applications based on limited profiling signal and similarity to previously scheduled applications. It does not differentiate between batch and LS applications and schedules applications so as to minimize interference and maximize utilization. Applications are classified for interference tolerance using micro-benchmarks stressing a specific shared resource with tuneable intensity, which are run concurrently with an application to find out the interference level at which the application's performance falls below 95% of its performance in isolation.

Quasar [DEL14] improves on Paragon in that it also performs resource allocation instead of only resource assignment. Quasar extends the classification engine of Paragon to consider scale-out and scale-up scenarios, as well as different workload types with different constraints and resource allocation controls. It also provides an API that allows expressing the performance constraints regarding throughput and latency.

CloudScope [CHE15] is a representative of model-based approaches to QoS-aware cloud resource management and uses a discrete-time Markov Chain model to predict performance interference of co-located VMs. CloudScope runs within each host and collects application and VM-related metrics at runtime. The metrics serve to maintain an application-specific model capturing the proportion of the time an application uses a particular resource. The model is then used to predict slowdown due to colocation and ultimately to control placement of guest VM instances as well as adjusting the resources available to a hypervisor.

CtrlCloud [ADA17] is a performance-aware cloud resource manager and controller, which optimizes the allocation of CPU resources VMs to meet QoS targets. It maintains an online model of the relationship between allocated resource shares and the application performance, and uses a control loop to adapt the resource allocation so as to progress towards a probabilistic performance target expressed as a percentile of requests that must observe a response time within certain bounds.

Pythia [XU18] is a colocation manager which uses a linear regression model to predict combined contention on shared resources when co-locating multiple batch workloads with an LS workload. Pythia performs contention characterization for each batch workload running together with a particular LS workload and removes batch workloads that are too contentious to allow safe colocation. It then selects a small subset of batch workloads to co-locate with a latency sensitive workload and measures their combined contention to build a linear regression prediction model for contention due to multiple batch workloads.

Our selection illustrates a variety of approaches proposed over the years, each fitting a different context, yet none able to claim to solve the problem once and for all. Our approach will not be different in this aspect, but will focus on a privately-controlled cloud infrastructure. Unlike other approaches, we aim to treat all resources equally for the purpose of performance interference characterization, and rely on statistical characterization and similarity to reveal dependencies between background workloads.

© FitOpTiVis Consortium (Public Document)

5.2 Reconfiguration on the CompSOC Platform

The following presents the concept of reconfiguration and resource management framework to be realized on the CompSOC platform. This framework is an instance of the FitOptiVis architecture (see Deliverable 2.1). The mechanism to realize the framework is detailed in Deliverable 4.3. The section also describes how the concepts map to the abstractions provided by the OpenCL-centric runtime API.

5.2.1 Terminology

- **Component**: A component is a part of a platform or an application. Components can be composed to form larger components—e.g., applications or (virtual) execution platforms. They have one or more configurations, determined by component parameters, and may be reconfigurable. Component configurations have budgets and qualities. A budget can be provided or required. In OpenCL terminology, a *component* can be an OpenCL device (e.g. a GPU, CPU or an FPGA device) or an OpenCL *platform* (including all the controllable devices). It can also mean the whole OpenCL *application* including the host and the device parts, depending on the abstraction level used.
- **Task**: A task is an (application) component, which has only required budgets. In the OpenCL API, the kernels and buffer transfer *commands* are the tasks.
- **Application**: An application is a set of tasks that provides functionality to a user. In OpenCL the application consists of a main program running on a host device and a number of commands created by the program.
- **Resource**: A resource is a (platform) component, which has only provided budgets. This matches the concept of an OpenCL *device*.
- **Virtual Resource (VR)**: A virtual resource is a (platform) component, which is mapped to a single resource. In the case of pocl-remote, a virtual resource can be the device type/class/vendor for which an OpenCL kernel is optimized. Then the actual physical device will be assigned by the server-side resource manager.
- **Execution Platform (EP)**: An execution platform is the set of all resources. This matches the OpenCL *platform*.
- **Local Execution Platform (LEP)**: A LEP is the set of resources managed by a single Local Execution Platform Manager (LEPM). Every resource is part of a single LEP. Each compute server in the pocl-remote scheme can use a LEPM to manage its devices (e.g. which GPUs are dedicated to which remote application's use at which time).
- **Virtual Execution Platform (VEP)**: A VEP is a set of virtual resources that can host an application. An application has a valid deployment on a VEP when its required budgets match the budgets provided by the VEP.
- **Virtual Local Execution Platform (VLEP)**: A VLEP is a subset of a VEP that contains all VRs mapped to (resources that are part of) a single LEP. Each VLEP is managed by a Virtual Local Execution Platform Manager (VLEPM). The VEP/VLEP concepts currently do not have a direct counterpart in the OpenCL API, but these can be added within FitOptiVis as an additional initialization API by means of a runtime platform requirement description mechanism.

© FitOpTiVis Consortium (Public Document)

5.2.2 Overview

A block diagram of the proposed quality and resource management framework is depicted in Figure 21. Applications are composite components that are made up of tasks. Applications have one or more configurations, which are determined by application parameters. Applications may have certain provided qualities, and during their execution, they may be expected to provide certain quality levels (i.e., meeting QoS requirements). Each application configuration results in certain quality levels.

As shown in Figure 21, an Execution Platform (EP) is used to execute applications. In order to use the EP efficiently, applications are consolidated in an isolated manner. Subsequently, to realize this isolated consolidation, applications are deployed on Virtual Execution Platforms (VEPs). VEPs are composite platform components, which are comprised of virtual resources each of which must be mapped to a resource located in the EP. An application has a valid deployment on a VEP when its required budgets match the budgets provided by the VEP.

Applications may have certain quality requirements, which are met when they are properly configured and provided with sufficient resource budgets. Consequently, we propose a quality and resource management framework, which configures applications according to their quality requirements and ensures that application budget requirements are met. The proposed framework consists of several function blocks and databases, also shown in Figure 21. In the following section, we elaborate on the responsibilities of each block.

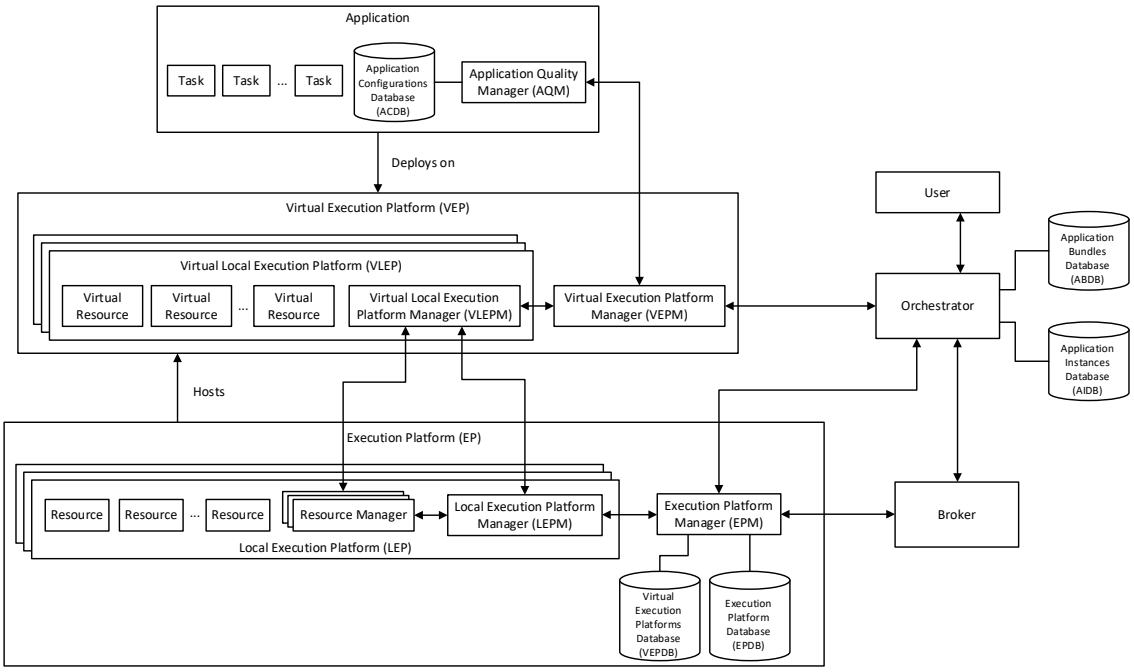


Figure 17. Block diagram of the proposed quality and resource management framework.

5.2.3 Functional Blocks

5.2.3.1 Application Quality Manager (AQM)

The Application Quality Manager is responsible for lifecycle management of an application. Each application may have one AQM task, which performs application-specific functions such as configuring application tasks with proper parameters. In particular, it has the following responsibilities:

- Configuration and reconfiguration of applications during the instantiation and reconfiguration phases, respectively. Each application task may have certain parameters that must be set before the task starts to execute. Additionally, it may be necessary to modify these parameters during task reconfiguration. The AQM configures/reconfigures the application tasks using the parameters that are given by the VEPM.
- Measuring application qualities during application execution. A quality is a measurable value that demonstrates how effectively an application is operating. Each application may have certain quality requirements that must be met during application execution. Employing an application-specific method, the AQM measures and monitors application qualities at run-time.
- Making reconfiguration decisions when certain events happen. During application execution, certain events such as workload transitions may occur which necessitate application reconfiguration including modifying application allocated resources, application parameters, and/or application state (e.g., application tasks). Such reconfiguration decisions are made by the AQM.
- Sending reconfiguration requests to VEPMs. Since the AQM is not privileged enough to modify the application VEP, it must ask VEPMs to perform reconfiguration when the application VEP must be modified.

5.2.3.2 Orchestrator

The orchestrator, which serves as the entry point of the system, manages the execution of applications (i.e., instantiation and reconfiguration) by orchestrating the EPM and VEPMs. The orchestrator is responsible for the following:

- Receiving user requests regarding running and lifecycle management of applications. As mentioned above, the orchestrator is the entry point of the system. The end user sends its requests regarding loading (i.e., running) and lifecycle management (e.g., updating quality requirements) of applications to this entity.
- Management of Application Bundles Database (ABDB) and Application Instances Database (AIDB).
- Lifecycle management of Virtual Execution Platform Managers (VEPMs). Each application VEP is managed by a VEPM, and a VEPM itself is managed by the orchestrator. VEPM lifecycle management tasks such as VEPM instantiation are performed by the orchestrator.
- Management of application deployment. To deploy an application, the Orchestrator asks the Broker to select one of the application configurations and determine a VEP to host it.

© FitOpTiVis Consortium (Public Document)

5.2.3.3 Virtual Execution Platform Manager (VEPM)

The Virtual Execution Platform Manager is responsible for the lifecycle management of the VEP an application is deployed on. This is done through orchestration of VLEPMs. For each application, there exists one and only one VEPM. Upon user requests to instantiate an application, a VLEP is created, and the VEPM is loaded onto it by the Orchestrator. Subsequently, the VEPM creates VLEPs for VLEPMs, and manages the creation of application VEP by orchestrating the VLEPMs. The VEPM has the following responsibilities:

- Lifecycle management of VLEPMs. Each application VEP is distributed among several VLEPs, each managed by a VLEPM. VLEPM lifecycle management tasks such as VLEPM instantiation are performed by the VEPM.
- Lifecycle management of application VEPs. Lifecycle operations (including creating, destroying, and reconfiguration) of application VEPs are managed by the VEPM. Since an application VEP is composed of one or more VLEPs each of which managed by a VLEPM, its lifecycle management requires the orchestration of VLEPMs, which is performed by the VEPM.

5.2.3.4 Virtual Local Execution Platform Manager (VLEPM)

The Virtual Local Execution Platform Manager is responsible for the lifecycle management of a VLEP, which is a part of an application VEP. VLEPMs are instantiated by VEPMs and are responsible for lifecycle operations of VLEPs including creating, destroying, and reconfiguration of VLEPs. To do so, each VLEPM communicates with the LEPM and Resource Managers of the LEP it is mapped on. Constrained by its access rights, a VLEPM must ask the LEPM to reserve/release virtual resources. However, for other lifecycle operations, such as allocation and initialization, it directly asks the Resource Managers.

5.2.3.5 Execution Platform Manager (EPM)

The Execution Platform Manager is responsible for managing the resources that the Execution Platform (EP) is comprised of. All the global resource-related requests are passed to this entity. Additionally, it keeps track of available resources, their costs, and resources used by VEPs. In particular, the EPM is responsible for:

- Management of Execution Platform Database (EPDB) and Virtual Execution Platforms Database (VEPDB). The information regarding available resources, resource costs, and the resource shares owned by VEPs are collected and managed by the EPM in two databases. These information are provided by LEPMs.
- Exposing resource information to the Broker. During the resource brokering process, the Broker provides the EPM with a set of application required budgets and the maximum affordable costs. Having the global view of available resources, the EPM provides the Broker with a set of VEPs meeting the required budgets and costs.

© FitOpTiVis Consortium (Public Document)

5.2.3.6 Local Execution Platform Manager (LEPM)

As mentioned before, each resource is part of a LEP and is managed by a single Local Execution Platform Manager. LEPMs are entry points of LEPs. Resource-related requests sent by remote functional blocks are received by this entity. LEPMs are responsible for:

- Management of resource reservations and allocations. In order to create VLEPs, their required resources must be reserved and allocated. The actual reservations and allocations are performed by Resource Managers. However, given the fact that each VLEP may be composed of various resources, a single entity is necessary to ensure that all the required reservations and allocations are done successfully.
- Exposing resource information to the EPM. In order to keep the global view of EP updated, each LEPM informs the EPM about the available resources and their costs.

5.2.3.7 Resource Manager (RM)

Resource managers are employed to create, configure/reconfigure, and destroy virtual resources. As shown in Figure 22, several steps must be taken for each operation. To create a virtual resource, first, its required budget – described in the Budget Descriptor – must be reserved. In this step, the required budget is being compared to the budget provided by the resource. If the reservation is successful (i.e., the provided budget is not less than the required one), a virtual resource identifier is generated, and the creation process continues with allocating the resource. During this step, the budget is programmed into the resource using the identifier. Hence, the allocation step may take more time than the reservation step. After the allocation step, the virtual resource is created and it is ready to be initialized (i.e., to be configured, e.g., load instruction

© FitOpTiVis Consortium (Public Document)

memory of a vCPU with application code). Finally, the initialized virtual resource starts running.

Similarly, several steps must be taken to destroy a virtual resource. First, the virtual resource must be stopped. Given the fact that the virtual resource may be busy at this point, stopping a virtual resource can be a slow process. After the resource becomes stopped, it may need to be reset to its initial state. Finally, the programmed budget must be released. When the budget is released, the available budget gets back to its previous state, and the virtual resource is destroyed. Besides lifecycle management of virtual

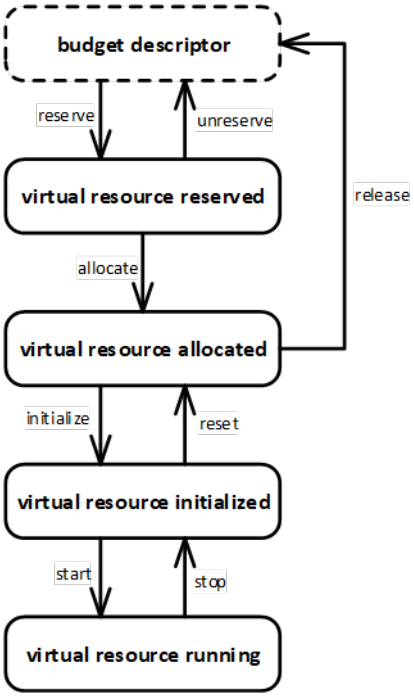


Figure 18. Lifecycle FSM of a virtual resource.

resources, RMs measure and monitor performance and costs of resources. In order to keep the LEPM updated about the status of local resources, RMs provide the LEPM with the measured performance and costs. Such provided data are maintained in the EPDB by the EPM.

5.2.3.8 Broker

The Broker, which acts as a decision maker in the system, determines the optimal configurations for all the platform and application components. For instance, when an application is planned to be instantiated, the Broker decides which application configuration should be deployed to meet the application's quality demands and which VEP configuration should be selected to host the application instance. To do so, the Broker needs to know information concerning application configurations (including their required budgets and offered qualities) and VEP configurations (including their provided budgets and costs). The former is provided by the Orchestrator using Application Bundles stored in ABDB, and the latter is provided by the EPM using the information stored in EPDB. The decisions are made in such a way that the application quality requirements are met and the aggregate cost of resources is minimized.

© FitOpTiVis Consortium (Public Document)

5.2.3.9 Databases

As shown in Figure 21, there are several databases in the proposed architecture containing information necessary for quality and resource management. Generally, the information of each component is stored in a structure called Component Bundle, shown in Figure 23. For each component configuration, the Component Bundle contains its parameters, qualities, Budget Descriptor, and initial state. Configurations are determined using the parameters. Qualities describe offered qualities of application components or costs of platform components. The Budget Descriptor, which has a hierarchical structure, describes either the provided budget of a platform component or the required budget of an application component.

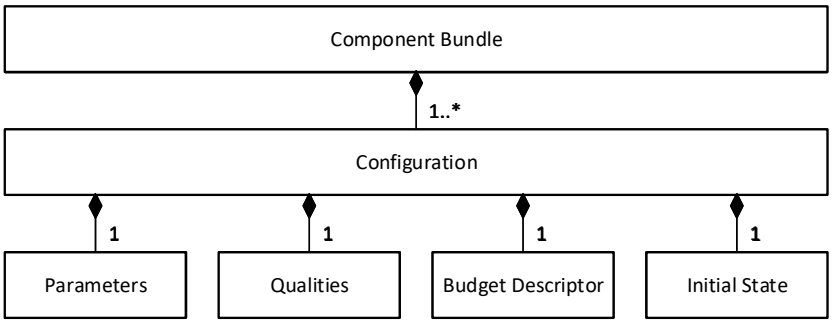


Figure 19. Structure of Component Bundle

The mentioned databases store the following information:

- Application Bundles Database (ABDB): This database stores all the application bundles. Each application bundle contains all the application configurations. This database is created and maintained by the Orchestrator.
- Application Instances Database (AIDB): It stores the bundles of application instances. Since each application instance is configured with one application configuration, the application instance bundle contains only one configuration. This database is also created and managed by the Orchestrator.
- Application Configurations Database (ACDB): The AQM needs to know about all the application configurations for making reconfiguration decisions. This database provides the AQM with this information. In essence, it stores the application bundle, which is also stored in the ABDB.
- Execution Platform Database (EPDB): It contains information of all the resources within the Execution Platform. This database is maintained by the EPM using the information collected from LEPMs.
- Virtual Execution Platforms Database (VEPDB): This database maintains information of all the created VEPs. Since each VEP is configured according to a single configuration, its bundle has only one configuration. This database is also maintained by the EPM using the information collected from LEPMs and VEPMs

© FitOpTiVis Consortium (Public Document)

5.2.4 Budget Matching

As explained before, the Broker makes decisions regarding application configurations and the VEP they deploy on. To do so, the Broker must decide on the budget connections (i.e., vertical compositions) as well. That is, the Broker makes sure that:

I. the VEPs on which applications are deployed provide enough resource budgets to applications, and that
II. the EP provides enough budgets to the VEPs that it hosts.

For this purpose, we have proposed and developed a performance analysis framework whereby the worst-case response time of applications can be determined based on their resource requirements and resource budgets provided by the EP. The overview of this framework is shown in Figure 24 below.

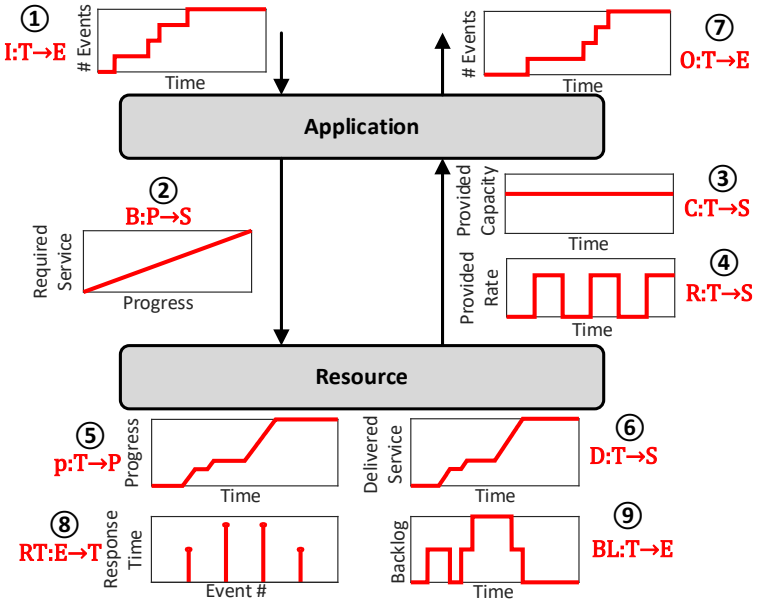


Figure 20. Overview of the performance analysis framework

In this framework, the provided budget of a resource is modelled by two functions describing the capacity of the resource and the rate at which the resource serves the requestors. The capacity of a resource describes the maximum service that the resource can deliver to applications at all the time instants (e.g., stored energy of a battery), and the rate describes the maximum service that the resource can deliver to applications at each time instant (e.g., battery power).

Resources such as processors and interconnects do not have any constraints on the total service they can allocate to applications; however, the service they deliver at each time instant is constrained by their limited bandwidth. In other words, their provided capacity is infinite, but their provided rate is limited.

Resources such as memories (space) and FPGA (area) can only accept requests when the total service they deliver to applications at that moment has not reached their capacity.

© FitOpTiVis Consortium (Public Document)

Finally, a class of resources such as batteries has limits on both their provided capacity (joules) and rate (watts). Provided budgets for an illustrative example where the EP is composed of a CPU, a HW accelerator, a memory, and a battery are shown in Figure 25 below. The horizontal axis in all plots is time.

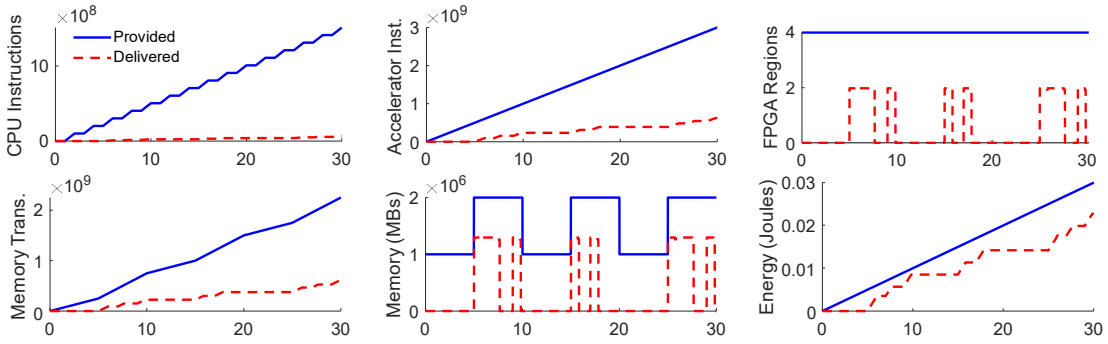


Figure 21. Example budgets for different resource classes

Applications require budgets when they are invoked by incoming events (e.g., video frames, DMA transactions, OpenCL kernel calls). Each application requires a certain service from one or more resources to handle an event. We assume that the required budget of applications can be characterized by a set of functions that specify the service that an application requires from a resource when certain application progress is made.

Application progress indicates the number of (fully or partially) processed events. Applications make progress only when they are delivered the budgets they require from all resources. At each time instant, an application makes progress until a point at which the total delivered service does not exceed the provided capacity and the service that is delivered since the previous time instant does not exceed the provided rate. The required budgets of an application used in the illustrative example are depicted in Figure 26 below. Here, the horizontal axis of all plots is progress, and the curves regularly repeat over progress.

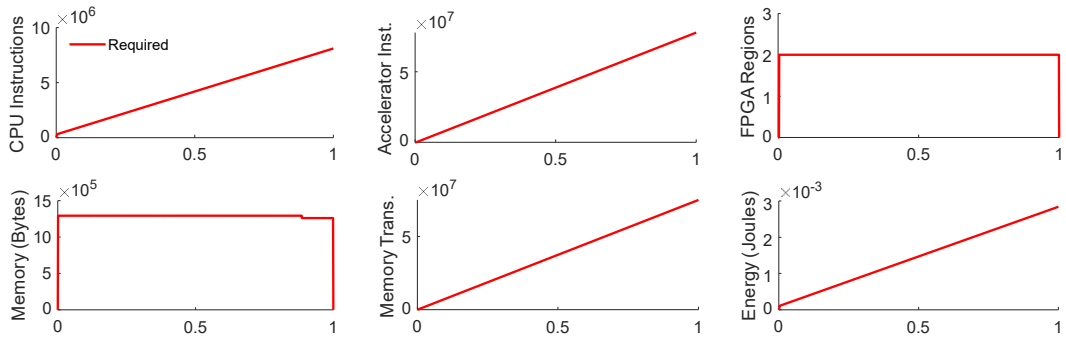


Figure 22. Resource budgets required by an application to make progress

By computing the application progress, we can obtain the timing behavior of processed events, thereby computing response times of incoming events. We say the budgets provided by the EP are matched with budgets required by applications/VEPs whenever the worst-case response time of events is not greater than the required response time of those components. In addition to computing WCRT of a given trace of a system, the framework allows us to compute an upper bound on WCRT when bounds of traces are

© FitOpTiVis Consortium (Public Document)

given. The plots in Figure 27 below show the response times, application progress, and backlog in the illustrative example when the incoming stream of events is periodic with jitter.

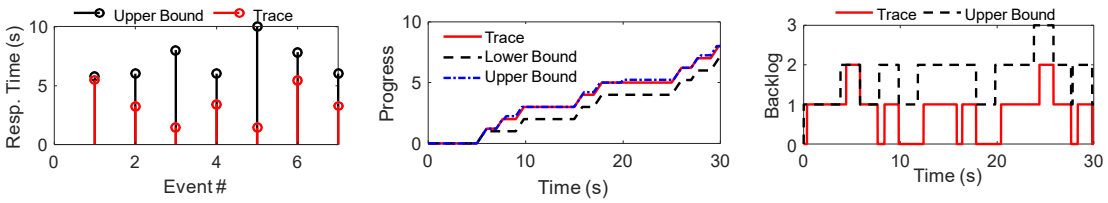


Figure 23. Example application response time, progress, and backlog

5.3 Reconfiguration in Processor/Co-processor Systems

In general, designers should be supported at design-time, to define, characterize and be able to deploy platforms that optimally match the given requirements, while guaranteeing that customized applications are still interoperable. Nevertheless, in dynamic and reactive systems, such as CPS, design-time customizability is not sufficient.

Modern systems are required to be flexible and versatile, capable of supporting multiple operational profiles corresponding to different trade-offs, and capable of switching between these profiles at runtime [BYS10] during dynamic reconfiguration. We are therefore addressing the definition of efficient run-time methodologies capable of coping with the need for flexibility at all levels of CPS systems, from edge to cloud. Here we deal specifically with run-time adaptability at the hardware component level, in particular in reference to multi-purpose co-processing units.

5.3.1 Dynamic Parameter Adjustment

The concept of dynamic parameter adjustment was introduced by Burleson et al. in [BUR01]. As illustrated in Figure 28, tuning processing in response to content variation and/or changing user/system requirements is made possible by runtime variation of different parameters. These can be classified as follows:

- **Functional parameters**. These allow tuning the output of a computation, and may include, e.g., filter and transform lengths, or quantization levels.
- **Architectural parameters**. These allow tuning guaranteed performance and energy consumption—without modifying the output of the computation. Architectural parameters include, e.g., the level of parallelism employed in the computation, which may affect throughput and energy consumption.

© FitOpTiVis Consortium (Public Document)

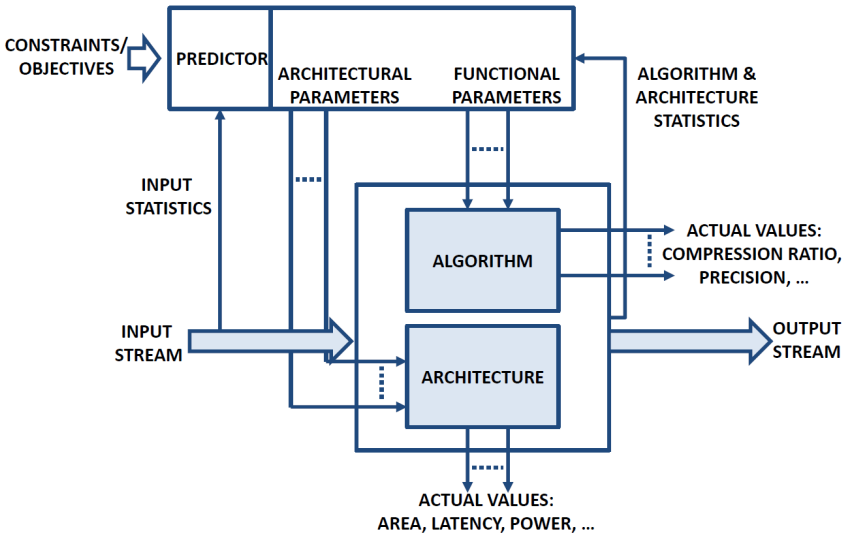


Figure 24. Dynamic parameter adjustment [BUR01].

The work of Burleson et al. refers mainly to video codec specifications, but it can be generalized to image and video processing pipelines such as the ones we are dealing with in FitOptiVis. Within the CERBERO H2020 project, which has been recently finished, a similar concept has been formalized in the definition of the adaptation loop [PFS19] shown in Figure 29.

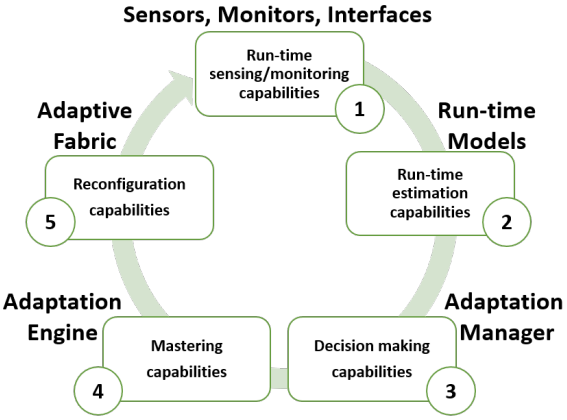


Figure 25. Self-adaptation loop as defined in the CERBERO H2020 project.

According to the formalization provided within CERBERO, self-adaptation aims at changing structure, functionality or parameters of the system in response to information coming from the environment, the user, or the system itself. Self-adaptation in CERBERO involves a feedback loop from sensors to a decision entity, decomposed as follows:

1. Run-time sensing/monitoring capabilities—to capture environment, human-commanded and system status changes with proper interfaces.
2. Run-time estimation capabilities—to estimate, during system execution, the Key Performance Indicators (KPIs) reflecting the status of the system.

© FitOpTiVis Consortium (Public Document)

3. Decision making capabilities—to define, given the evaluated KPIs and a set of predefined criteria, whether adaptation is needed to meet the expected goals, or whether to keep the execution as close as possible to its current status.
4. Mastering capabilities—to select a type of adaptation suitable for the available computing infrastructure.
5. Reconfiguration capabilities—to execute the planned changes on the available adaptable fabric.

Both above-mentioned approaches have been used as the starting points/building blocks for activities carried out within FitOptiVis. In particular, [BUR01] presents a way of describing a dynamically tuneable computing infrastructure that fits the work carried out in WP2, where a composable, customizable and reconfigurable virtual reference platform for video and image processing pipelines is defined. According to this formalism, both functional and architectural parameters can be customized to optimize a system before deployment to meet the given constraints, and to adapt the system at runtime according to varying environmental or system conditions, or to human requests. The formalization of the FitOptiVis DSL (D2.2) has allowed us to derive a subset description of the Water Supply use case (see D6.1), both in terms of application and architectural components. Adaptation support is not yet implemented yet, but initial steps have been taken to enable modelling and prediction of some of the important execution parameters, such as latency.

5.3.2 Runtime Estimation and Decision Making

The contribution specific to WP4 is at the predictor level, which should encapsulate the *runtime estimation* and *decision making* capabilities of the CERBERO adaptation loop. In processor to co-processor systems (see Deliverables 5.1 and 5.2 for more details) deployed using the Multi-Dataflow Composer (MDC, see Deliverables 3.1 and 3.2 for more details) coarse-grained functional and non-functional reconfiguration is enabled. In particular, MDC generated co-processors/accelerators are specialized hardware modules capable of accelerating different algorithms (functional reconfiguration) and/or different variants of the same algorithm (non-functional reconfiguration). Applied at a coarse-grain level, reconfiguration is very quick and takes place by simply overwriting a unique configuration register in the accelerator. Decisions on parameter tuning can be then taken at run-time, starting with the knowledge of the current state and taking into consideration varying objectives/requirements, characteristics of the processed data, and actual processing and architectural KPIs, such as the offered quality of service, throughput, or energy consumption.

We have completed the definition of the automated support for dynamic reconfiguration. The MDC tool is capable of automatically generating the APIs that enable transparent access to co-processors/accelerators from a host-processor and supports different types of coupling (e.g., loose coupling, utilizing memory-mapped communication, or tight coupling, utilizing stream-based communication), different host processors, and optionally using DMA for data transfers (WP3 and WP5 work). These APIs also enable co-processor/accelerator reconfiguration by simply changing the specific function call used to offload computation on the co-processor/accelerator.

We are also working on the definition of a proper, minimally invasive monitoring infrastructure which will enable gathering runtime data required for KPI estimation. So far, the collaboration within the consortium resulted in preliminary integration of MDC

© FitOpTiVis Consortium (Public Document)

and AIPHS. We have also presented[1] a framework for developing complex heterogeneous architectures composed of programmable processors and dedicated reconfigurable accelerators in FPGA. The framework supports customizable monitoring systems and enables control over the introduced overhead [JOINTER]. More detailed elaboration of this framework is provided in D4.3.

5.3.3 Reconfigurable Neural Network Accelerators

Given that the predictor component is partially use-case specific (with respect to the functional parameters to be tuned), we have been working on the implementation of the part of processing relevant to the Water Supply use-case. Three different neural networks provided by AITEK (*INC_net*, *Mobile_net* and *VGG_net*) have been implemented as reconfigurable hardware accelerators with the help of the MDC tool. The three networks are composed of different operators/actors, some of them parameterized and reused in multiple networks. Table 4 illustrates the composition of the networks in terms of operators/actors as well as the maximum value of parameters for each operator/actor within the neural networks. This shows that reusing operators/actors in different network implementations and support for functional parameter adaptation is feasible.

Table 4. Composition of neural networks in terms of operators/actors and maximum value of parameters.

	INC_net		Mobile_net		VGG_net	
	nr	**max param**	**nr**	**max param**	**nr**	**max param**
Input	1	1x128x128x3	1	1x128x128x3	1	1x128x128x3
Transpose	2	-	2	-	1	-
Conv	23	64x64x3x3	14	512x256x1x1	13	1024x1024x3x3
Relu	23	-	14	-	13	-
Concat	3	-	14	-	0	-
Reshape	1	-	1	-	1	-
Sigmoid	1	-	1	-	1	-
BatchNormalization	3	256	14	512	0	
MaxPool	6	-	1	-	4	-
Add	3	784	1	2	1	2
Sqrt	1	-	0	-	0	-
Reciprocal	1	-	0	-	0	-
Mul	3	784	0	-	0	-
Sub	1	784	0	-	0	-
Cast	1	2	1	2	1	2

[1]JOINTER: Joining flexible monitors with heterogeneous architectures. DATE 2020 virtual U-Booth exposition.

© FitOpTiVis Consortium (Public Document)

MatMul	1	784x2	1	512x2	1	1024x2
AveragePool	0	-	0	-	1	-
Output/Sigmoid	1	1x2	1	1x2	1	1x2

Porting to hardware is still an ongoing activity. So far we have managed to optimize some of the critical operators/actors, in particular the *Conv* operator/actor, which is responsible for all convolution calculations in the neural networks, and which turned out to be the most complex and time consuming. We started from the baseline Register Transfer Level description of the operator/actor produced by Vivado HLS without any directive or designer intervention, and produced two different variants of the design with the aim of employing more resources to increase performance:

- *paral*: this variant was obtained by applying an UNROLLING directive to the inner loop of the computation in the C implementation of the Conv operator/actor;
- *pipe*: this variant was obtained by applying a PIPELINING directive to the inner loop of the computation in the C implementation of the Conv operator/actor.

Table 5 shows the resource usage of the Conv operator/actor together with the latency of the computation code block for the three variants: *baseline*, *paral*, and *pipe*. The results show that applying either of the directives to the HLS flow leads to significantly reduced latency compared to the baseline variant. Obviously, the speed up is paid for in terms of resource—the *paral* variant requires significantly more resources than the *baseline* (about 650% more LUTs, about 490% more FF, and 3100% more DSPs when implemented in a Xilinx Artix-7 XC7A50TCSG324 FPGA). However, the *pipe* variant, which provides even higher speedup than the *paral* variant, only requires a modest increase in resources (about 16% more LUTs, and about 7% more FFs), making it ideal for implementing the neural networks.

Table 5. Resource occupancy and execution latency of the computation code block of the Conv operator/actor variants.

metric	**Conv operator/actor variant**				
	baseline	**paral**		**pipe**	
	value	**value**	**%**	**value**	**%**
LUT	1154	8653	649.83	1337	15.86
FF	608	3583	489.31	650	6.91
DSP	1	32	3100.00	1	0.00
BRAM	3	3	0.00	3	0.00
computation code block latency [us]	944.64	385.92	-59.15	189.44	-79.95

In the final year of the project, we plan to implement a simple predictor with decision making capabilities to support functional reconfiguration in the Water Supply use case, relying on the building blocks available in the project (models developed in WP2, monitoring infrastructure developed in WP4, and extension of the MDC tool expected in

© FitOpTiVis Consortium (Public Document)

WP3) and the characterization of reconfigurable accelerators we expect to complete soon. Architectural reconfiguration, enabled by monitoring data, is still ongoing work.

5.4 Reconfigurable 8xSIMD Floating-point Accelerators

In Y2, we have developed reconfigurable 8xSIMD floating-point accelerators for the Zynq and Zynq UltraScale+ platform. Detailed description of the accelerators is provided D5.2, while integration into a Linux system is described in D3.2. In this section we describe the mechanisms for run-time reconfiguration, and concrete examples of code utilizing the reconfiguration mechanism are provided in D4.3.

The internal structure of the accelerator is shown in Figure 30.

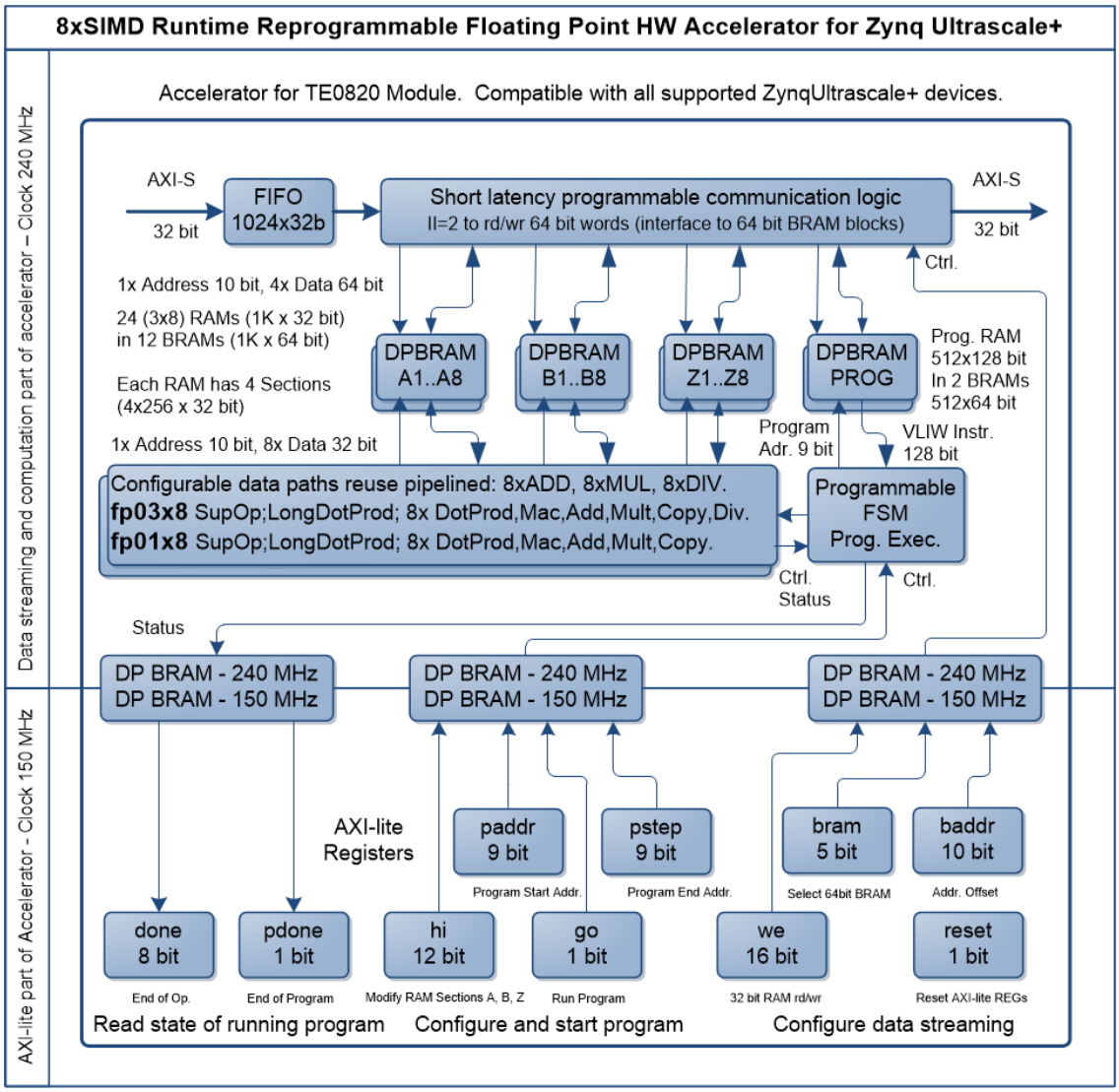


Figure 26. Run-time reconfigurable 8xSIMD floating point accelerator

5.4.1 Design Considerations and Requirements

The run-time reconfigurable floating point accelerators for the Zynq and Zynq UltraScale+ platforms have been designed and realized with respect to the following considerations and requirements:

1. Software utilizing the accelerator can be developed also directly on the board, using the C++ compiler (g++) present in the Debian OS and Xilinx data-mover support drivers.
2. The entire HW platform, comprising one or more accelerators, is provided in form of a shared library. The provided library API is compatible with C++ development practice and standard “make” can be used to build the user application.
3. The hardware of the floating point accelerators is fixed. Reconfiguration is performed by reprogramming the firmware code which defines the function of the programmable finite state machine (FSM) inside the accelerator and the function of the communication logic (see Figure 30 above).
4. Data communication is implemented as an AXI-stream and supports accelerator chaining.
5. The data communication support HW is determined at design time and cannot be changed at runtime. The following variants can be generated:
    a. Zero copy (ZC) HW data movers consuming minimal HW resources,
    b. DMA data HW data movers,
    c. Scatter gather (SG) DMA data movers with interrupts,
    d. Combination of ZC HW (DDR to Accelerator) and SG DMA HW (Accelerator to DDR)
6. All communication alternatives have to work with identical SW API. It means that the user SW code remains identical and does not need modifications at run-time.
7. Software must be able to query the list of SIMD FP operations supported by the accelerator. Based on this information, the software can be reconfigured to take advantage of supported operations.
8. The accelerator must be able to provide information on whether the HW license coming with the accelerator is valid.
9. The accelerator firmware is a simple sequence of VLIW vector instructions which support *for-loops*, *if-else*, and similar constructs. However, there is no support for checking overflow/underflow in floating point operations. Such constructs have to be implemented in the host code (executing on ARM core).
10. Computation performed in the accelerator can overlap with stream-based data communication. This is controlled by the user-space host software running on the ARM core.
11. Data are stored in 64bit-wide dual-ported blocks. This arrangement enables to use the Ultra RAM blocks (4096x64b) present in some larger Zynq UltraScale+ devices without affecting the accelerator library API or user code.

5.4.2 Reconfiguration by Change of Firmware

The accelerator executes sequences of VLIW vector instructions (firmware) stored in accelerator program memory. This firmware can be first defined in the host software and then downloaded via the streaming interface to the accelerator. The program memory will usually contain multiple different sequences of VLIW instructions.

© FitOpTiVis Consortium (Public Document)

Computation performed in the accelerator can overlap with stream-based data communication. This is controlled by the host software running on the ARM core and it can be used for run-time reconfiguration by loading a new VLIW instruction sequence to the accelerator program memory while computation is in progress.

For example, consider an application which needs to perform accelerated multiplication of 64x64 matrices (Z[64,64] = A[64,64] × B[64,64]). The application running on the host will split the matrix operation into shorter sequences of VLIW instructions and loaded instruction sequences into the accelerator program memory schedule scheduled by the application software running on the ARM host by adjusting pointers to instruction sequences to be loaded into the accelerator program memory while streaming parts of matrix B[64,64] from host DDR memory to the accelerator. Rows of the matrix are propagated as identical to all 8xSIMD memories in 8 subsequent stages. Detailed example of software using this run-time reconfiguration is presented in D4.3.

5.4.3 Reconfiguration by Temporary Change of Firmware

Application software can temporarily reconfigure the accelerator in the following steps:

1. Save accelerator context to DDR (this involves saving the content of accelerator memories and firmware to DDR; only selected parts of the context may be saved to reduce context-switching overhead),
2. Change firmware and upload it to the accelerator,
3. Execute the firmware (for example the **SupOp** instruction)
4. Read the results from accelerator data memory into ARM host memory,
5. Restore (full or partial) accelerator context from DDR.

After performing the above steps, the accelerator data and firmware is back in its original state and the application software running on the ARM host has information about the supported SIMD operations as well as about the status of the HW license.

Consider a scenario in which the application software needs to find out which SIMD operations are actually supported by the accelerator. This information is required to determine, e.g., which firmware version can be used with the accelerator. If the **DotProd** instruction is supported by the accelerator, the accelerated computation of 64x64 matrix multiplication (Z[64,64] = A[64,64] × B[64,64]) will use the instruction to improve efficiency.

Alternatively, if the **DotProd** instruction is unsupported, the application software running on the ARM host can implement an accelerated matrix multiplication using sequences of **Mac** (multiply and accumulate) instructions.

If the **Mac** instruction is also unsupported, the matrix multiplication can be implemented using the **Add** and **Mult** instructions. The performance of the matrix multiplication will be reduced by approximately 50%, but the accelerator will require less HW resources to implement. This might be necessary for some platform configurations where the programmable logic area is used by pre-defined HW accelerated video processing.

More detailed elaboration of this run-time reconfiguration is presented in D4.3.

5.4.4 Reconfiguration of Streaming Data Path

The architecture supports multiple accelerators connected in serial chains. This allows saving resources that would be otherwise spent to implement HW data movers and

© FitOpTiVis Consortium (Public Document)

enables direct communication from one accelerator to the next accelerator in the chain. However, such a connection creates dependency between accelerators and run-time reconfiguration of the data path is needed to full-fill the needed tasks.

The application software reconfigure the accelerator streaming data path at runtime to achieve the following functions:

1. Set all accelerators in the chain as “pass-through” with the exception of one, which uses the streaming data for:
    a. *Read* (reading data from the selected accelerator to ARM host DDR)
    b. *Write* (write data to the selected accelerator from ARM host DDR)
    c. Read and Write (perform read and write at the same time from two identical or different 64-bit BRAM blocks)
2. *Write* identical data to 2, 3 or 4 selected 64-bit BRAM blocks to all accelerators in the serial chain of accelerators.
3. *Read* from one selected accelerator and *write* data to another accelerator downstream in the chain of accelerators.

Consider again the case of matrix multiplication, in which run-time reconfiguration of the streaming data path can be performed to download rows of matrix B[64,64] from ARM host memory to the accelerator. Rows of matrix B are propagated to memories of all accelerators connected in the chain. Matrix B is modified in 8 run-time reconfiguration stages.

More detailed elaboration of this run-time reconfiguration is presented in D4.3.

5.5 Application-Specific Adaptation Scenarios

The following subsections provide details on adaptation specific to selected use cases.

5.5.1 Modelling System Variants and Configuration Changes

In the design phase of computer vision systems, it is most often left to the human designer to model and define the rules that can be applied dynamically at run-time to achieve adaptability (e.g., choosing a reduced frame rate or picture size). This can be theoretically made quite effective but often requires manual fine tuning of these rules.

In FitOpTiVis we aim for an incremental advancement over the current state of practice in this field. In deliverable D4.1 we proposed an approach based on generic variability modelling tools such as CVL [FLE09] and [LOP13]. These enable designers to model different system variants which then can be activated by a monitor which is analysing the performance of the system and the suitability for adaptations.

However, in the second year for the project, there have been significant advances in support for modelling, specifically the domain-specific language resulting from work within WP2. In contrast, the interest in variability-intensive modelling languages appears to be fading, both in the market and research communities. To reflect this development, we have re-evaluated our approach and instead of using CVL and related languages, we adopted FitOpTiVis DSL for modelling system configuration variants.

The FitOpTiVis DSL [D2.2, HEN20] is “an integral approach for smart integration of image-processing pipelines architecture, as well as supported tools for both design-time and run-time […]. The DSL provides a language in which systems are specified in an integral way, whereas the toolset enables automated optimisation within systems”. The

© FitOpTiVis Consortium (Public Document)

language thus provides a good platform which allows defining the functionality and possible configurations of a complex, multi-device video processing system such as the one presented in UC3 (Habit Tracking).

The proposed model (presented in D2.2) is currently simple and only covers a very particular example of adaptation: when energy levels in the system fall below a certain threshold, the clock speed and the energy profile of the board are both lowered. To support this scenario, model abstractions are generated for all participating components: the board, the camera, the facial recognizer itself, and the energy monitor. These components capture the requirements and constraints that need to be satisfied to perform configuration changes.

In WP4 we propose a mechanism based on the usage of the DSL to enable configuration changes in computer vision modules built for UC3 (Habit Tracking). The support for reconfiguration allows controlling the distribution of load between edge nodes (running on computing-power constrained ARM devices) and eventually more powerful cloud nodes (running similar algorithms but on powerful x86 nodes), changing the features of behaviour recognition, and changing the power profile of the edge nodes based on monitoring of the battery state and throttling of the CPUs/GPUs in the edge hardware.

The general architecture of the system is shown in Figure 31.

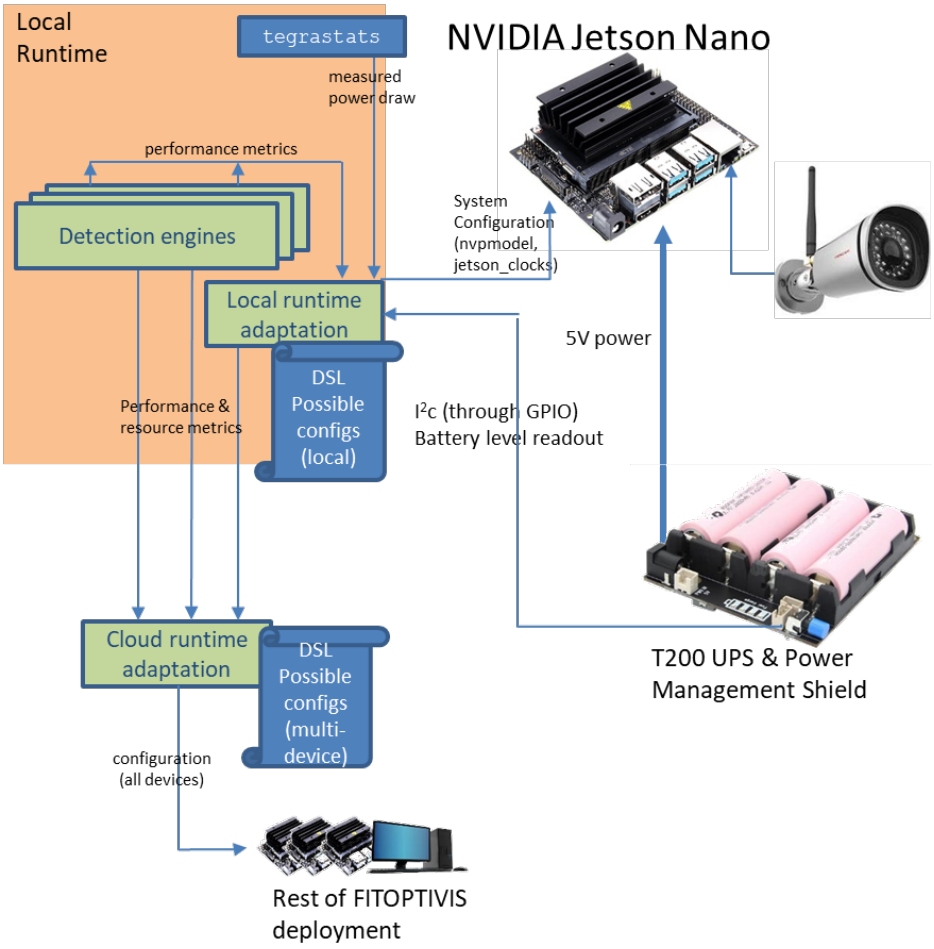


Figure 27. Architecture of the Habit Tracking adaptation system.

During Y2 we have focused mainly on modelling of the edge components (the camera, the board, and the power daughterboard). This provides us with a working basis with support for adaptation which will be expanded in the last year to support task distribution across more powerful cloud devices and other (underutilized) edge devices.

5.5.2 Selection and Compression of Task-Specific Features

During the first year of the project, a study was performed that assessed the feasibility of using visual attention to reduce data bandwidth in computer vision models for tracking and activity recognition applications. We have also considered other active-vision approaches that include changing geometric parameters of the sensor according to the task, or changing the perspective or focal length of the selected sensor. From the existing centralized methods, different metrics have been considered for resource management (runtime or not), adopting the edge-cloud paradigm.

Issues related to streaming of video include (but are not limited to) scalability when using multiple image/video sources, bandwidth of the shared network, real-time performance of the video processing components, privacy issues related to transmission of images, or the additional computational complexity when encryption is required. To address these issues, task-driven mechanisms that select (and compress) the most relevant information (e.g. through visual attention) are required. These mechanisms are part of the active vision [CHI17] concept, which covers (among others) adaptation and smart compression.

Two different strategies will be used: adaptation and task-driven selection of relevant features in order to reduce the data bandwidth and achieve real-time performance at the video processing components. The metrics upon which adaptation will be based depend on the application.

Our contributions to use cases UC3 (Habit Tracking) and UC9 (Surveillance of smart-grid critical infrastructure) will be released in form of software libraries. Regarding tools, we are currently using Python, C++ and OpenCV libraries, and GPU-accelerated solutions implemented using CUDA to achieve real-time performance on Ubuntu systems.

5.5.2.1 Application in context of UC9 (Smart-grid infrastructure surveillance)

Different computer-vision techniques have been considered to reduce computer load and network bandwidth in the context of UC9, especially techniques based on the concept of visual attention, which efficiently select relevant features and enable reduction of the required data bandwidth [BAR14]. The concept originates in biology, where perception is an active selection mechanism, and where adaptation and compression plays an important role. Many visual attention models emulating the biological process have been presented in the literature, conjugating a bottom-up saliency and a top-down modulation pathways. The saliency mechanism selects areas based on how discriminating they are with respect to their environment. The top-down modulation biases the selection with respect to the task being performed. This mechanism has been applied in many different fields such as robotics [FUJ10], autonomous navigation [LIU12], or military [CHE11].

© FitOpTiVis Consortium (Public Document)

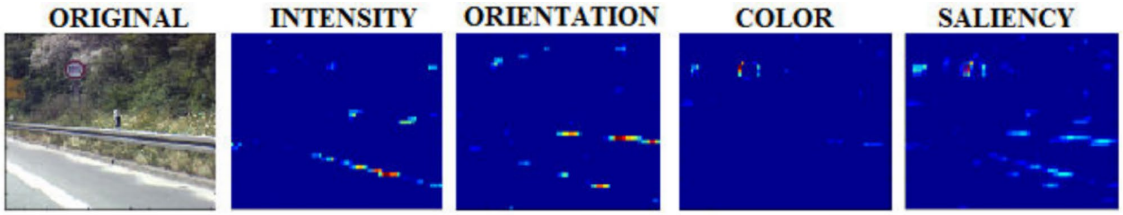


Figure 28. Example of saliency estimation from a driving scenario [BAR14]. Road mark, and the traffic sign are highlighted in the final saliency image (estimated with intensity, orientation, and color discriminative features).

In the second year of the project, another set of proven techniques has been found more efficient for the same task, with a more developed state-of-the-art. This technique is known as detection of Regions of Interest (ROIs). These regions are areas of the image in which there is a greater probability of finding elements that are relevant to a specific problem and problem domain [REN17]. In our case, these are areas with a greater probability of finding human subjects. The usual approach to detection and monitoring of moving objects in a video stream obtained by a static camera consists of background extraction. This technique allows identification of moving objects within a scene by generating a model of the frame's background [PIC04].

Figure 33 shows a frame from a test video stream and the result of applying different background extraction algorithms (MOG, MOG2, GMG, LSBP, KNN) on that stream.

© FitOpTiVis Consortium (Public Document)

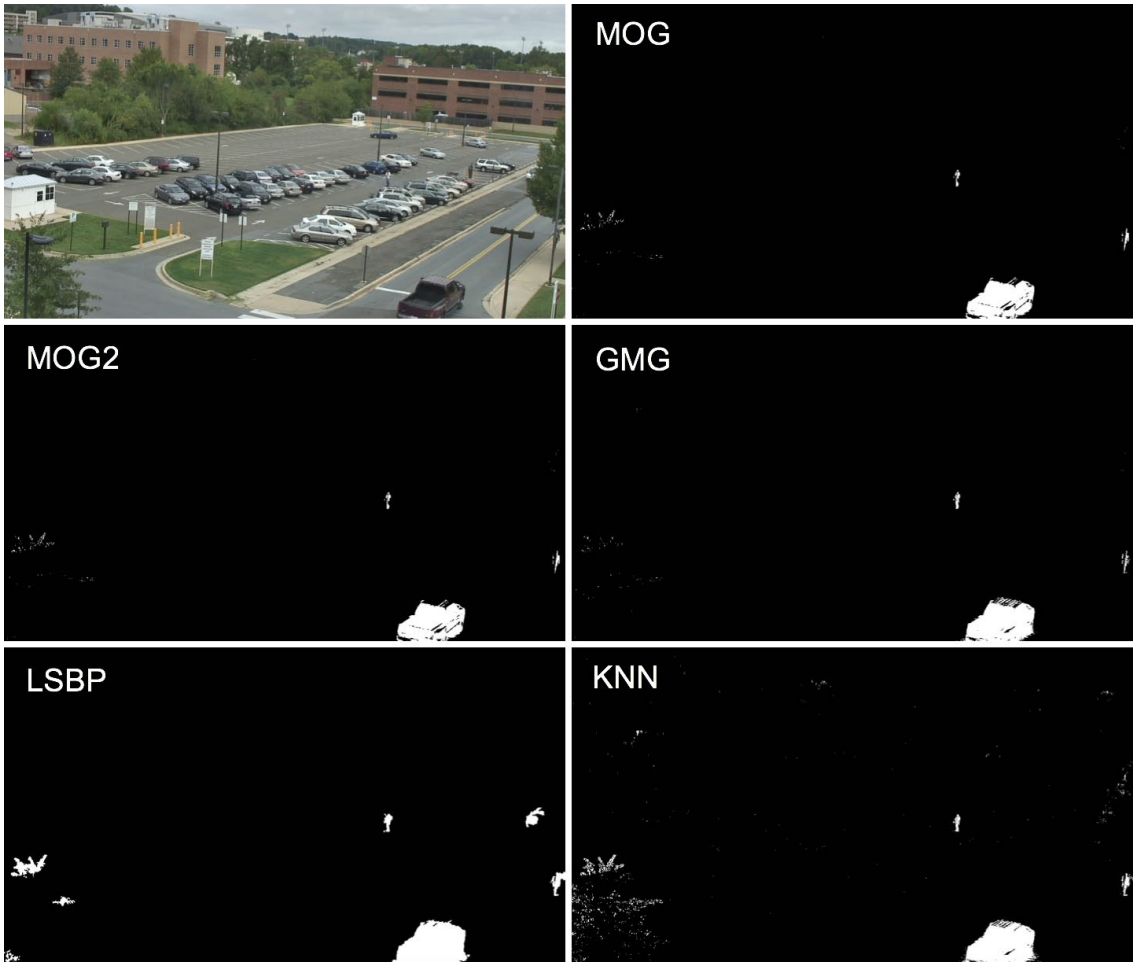


Figure 29. Impact of background extraction algorithms on detection of moving objects

Table 6 shows the performance of different background-extraction algorithms on the test stream. During testing, the MOG2 algorithm achieved the highest frame rate while producing good results with respect to background extraction, and was therefore selected for application in both UC3 and UC9.

Table 6. Performance of background extraction algorithms

Algorithm	Test FPS	**Algorithm**	Test FPS
Mixture of Gaussians (MOG) [KAE02]	33.35	**Local SVD Binary Pattern (LSBP)** [GUO16]	11.12
MOG2 [ZIV06]	125.12	**KNN** [ZIV06]	124.16
GMG [GOD12]	23.80		

In the case of UC9 (Smart-grid surveillance), the vision subsystem is focused on video-surveillance of the perimeter of an electrical substation and the main functionality is the detection of suspicious behaviour and robust tracking of suspicious targets.

Traditional object-tracking algorithms include mainly mean shift [COM00], particle filter [HUE02], frame-difference algorithms, Kalman filter [KAL60], etc., which follow the location of an object through multiple consecutive iterations. In recent years, tracking-

© FitOpTiVis Consortium (Public Document)

by-detection methods have received increased attention when dealing with a tracking problem [HAO18]. In this sense, the re-identification of persons is fundamental for the relationship between the detections of the same human subject over the time of a recording. In practice, a re-identification system consists of a person detector, a tracker, and a person matcher [ZHA19].

For our tracking application, accuracy and performance are the two qualities that are taken into account. Accuracy can be defined as the difference (in pixels) between the true location (real or labelled) and the estimated location of the target [SME14]. Performance is measured in number of frames that can be processed per second (FPS). Because performance depends on the number of targets being tracked, the number of targets to be tracked is a parameter that can be adjusted at runtime in response to desired performance. In the context of UC9, we require the system to be capable of tracking 4-5 targets in real time, and tolerate reduced performance if more targets were to be tracked.

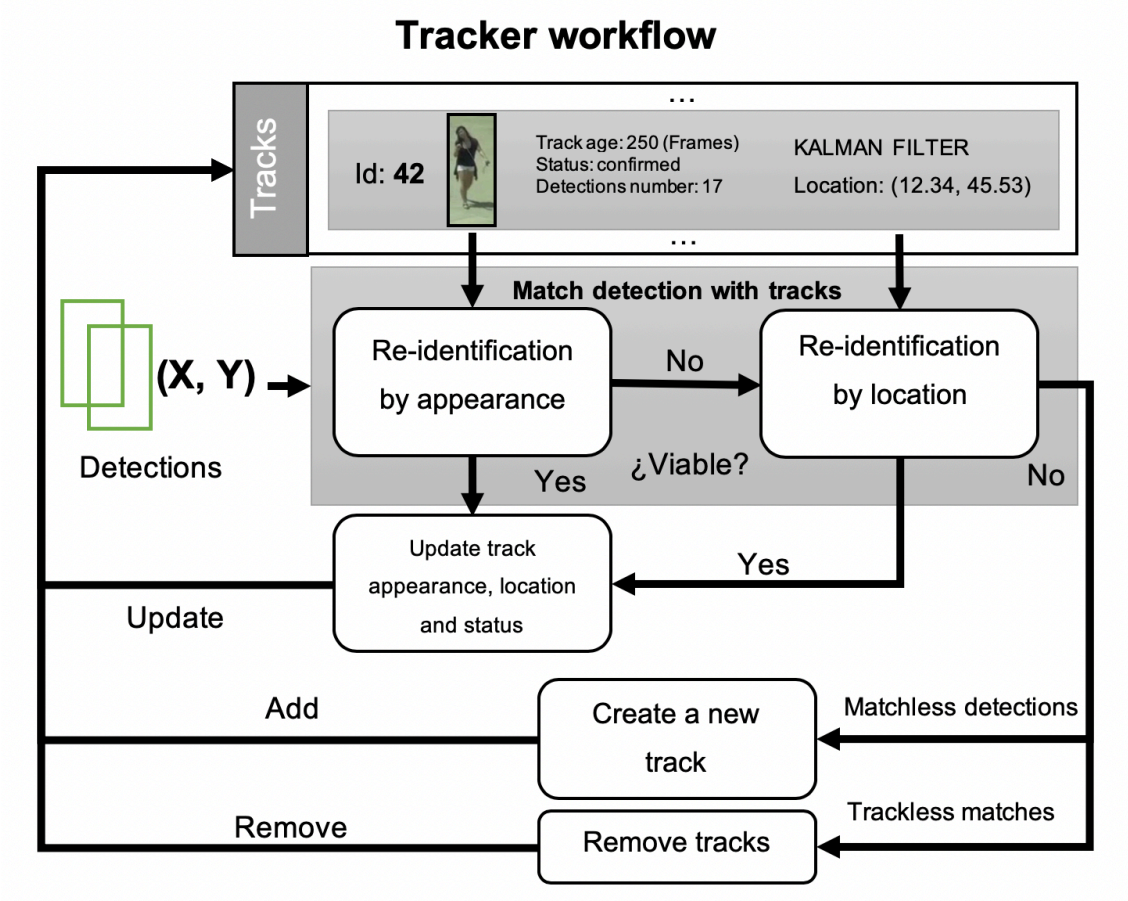


Figure 30. Operation of the tracker when human subjects have been detected

Figure 34 shows the operational workflow of the tracker when it has detected one or more human subjects. If more targets are found in the scene, the number of comparisons to be made to re-identify these detections or the number of calculations to predict the trajectories of their movements within the scene will increase and this will necessarily increase the required computational capacity.

© FitOpTiVis Consortium (Public Document)

Regarding the reduction of bandwidth when there are interesting events in the scene, we consider that applying an active scheme could reduce it to 3.7 MB/s by selecting a 128x128 (full resolution) box for the detected target, and transmitting the original image using a single color channel, scaled to ¼ of the original resolution (320p), maintaining the original frame rate, instead of transmitting a 1280p video stream at 30 frames per second that sums up to 120 MB/s.

We have also implemented a mechanism to adapt the resolution of the video feed to the saliency of the image data. Given a specific video source, more resolution and priority is assigned to a video source when a potentially interesting target is detected in its field of view. In other cases, only low-resolution images are transmitted, ensuring sensible use of network bandwidth and computational resources. Three different video qualities have been considered depending on the importance of events taking place in the scene:

- SD video (320 x 240 pixels at 5 frames per second),
- HQ video (960 x 720 pixels at 10 frames per second), and
- HD video (1280 x 960 pixels at 30 frames per second).

Modelling smart-grid infrastructure surveillance components

To model the smart-grid infrastructure surveillance system in QRML, we need to differentiate between application and platform components. The application components represent the software part of the system that deals with input from an RGB camera and outputs processed video data. The platform components represent the hardware part, specifically the edge nodes and the cloud server. The components interact at system level—processing the video stream, identifying and computing tracking data related to human targets. An overview of the model is shown in Figure 35. Here we only present a brief description of the individual components—additional details can be found in the QRML model files.

Application components

- **RGBCameraApp**: provides RGB video input at 30 fps.
- **Scaler**: adjusts image resolution and frame rate of the input video stream. Three different configurations are supported (see SD, HQ, and HD resolution/frame-rate combinations above).
- **HumanDetector**: identifies human targets in selected frames, extracts regions of interest (ROI) using a deep-learning classifier, and outputs bounding boxes containing the detections and feature vectors.
- **MultiCameraTracker**: responsible for re-identification of human subjects across different cameras and over time, using spatial-temporal information and the identified feature vectors.

Platform components

- **RGBCamera**: physical camera which provides video input to the system.
- **JetsonTX2**: edge node implementing human detection on an NVidia® Jetson TX2 board.
- **CloudCompute**: cloud server which aggregates information from cameras, puts together information from the edge nodes, and carries out robust tracking.

© FitOpTiVis Consortium (Public Document)

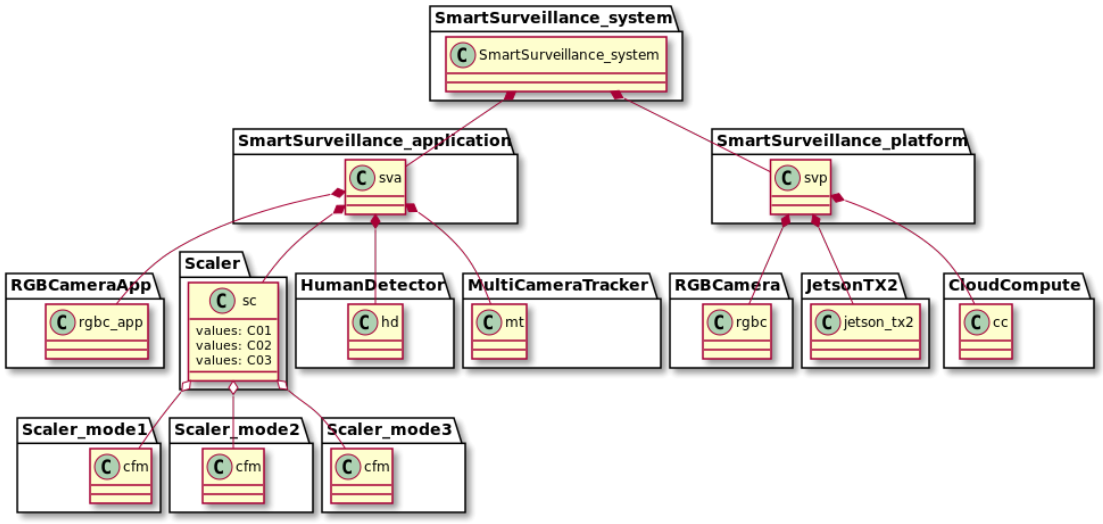


Figure 31. QRML model of the components of the smart-grid video surveillance system (UC9)

The video processing is split up between two platform components: the Nvidia® Jetson TX2 edge nodes and the high-performance cloud server. The edge nodes perform computationally demanding video processing tasks that only require local information, i.e., inference for human detection using a deep-learning model (running on 256 CUDA cores) or pre-processing and video scaling (running on 4 ARM cores). The cloud server gathers the local information and performs robust tracking in the multi-camera scenario.

The different configurations considered for system adaptation influence both image resolution and frame-rate, which in turn influences the accuracy of detection and re-identification tasks. Different configurations therefore provide different confidence levels for triggering an alarm. The goal is to reduce false positives—this is currently solved at a high cost in video-surveillance systems by involving humans in the loop.

System Adaptation and Reconfiguration

The system can operate in multiple modes, each appropriate for a different situation. When the situation changes, the system needs to adapt by switching to a configuration appropriate for the current situation. The following scenarios reflect situations which trigger system reconfiguration:

- Scenario 1: the system does not detect any people in the scene.
    - *Monitoring signal*: the number of humans detected in the scene, obtained from the **HumanDetector** component.
    - *Reconfiguration action*: taking into account the number of detections in the past and the confidence level of zero detections in a certain number of consecutive frames, the following reconfiguration sequence is sent to the nodes without detections:
        1. Obtain reconfiguration identifier from the cloud.
        2. Reconfigure **Scaler** for standard-definition (SD) video at 10 FPS.
        3. Perform only human detection video-surveillance task on SD video.
        4. Stream SD video to human operator for confirmation.
    - *Outcome*: the system only performs human detection video-surveillance tasks on SD video (the tasks related to re-identification and tracking are

© FitOpTiVis Consortium (Public Document)

suspended), providing limited (detect-only) video-surveillance functionality. As a result, it consumes less energy and computational resources, and produces less heat due to reduced edge node temperatures.

- Scenario 2: the system detects people in the scene, but not close to security perimeters.
  - *Monitoring signal*: the number of humans detected in the scene, obtained from the **HumanDetector** component.
  - *Reconfiguration action*: taking into account the number of humans detected in the past and the recurrence of human detections in a certain number of consecutive frames, the following reconfiguration sequence is sent to the nodes with detections:
    1. Obtain reconfiguration identifier from the cloud.
    2. Reconfigure **Scaler** for high-quality (HQ) video at 15 FPS.
    3. Perform all video-surveillance tasks on HQ video.
    4. Stream HQ video to human operator for confirmation.
  - *Outcome*: the system performs all video-surveillance tasks (detection, re-identification, and tracking of people) on HQ video, providing full video-surveillance functionality with reduced network bandwidth usage. This mode provides more accurate results than the detection-only mode, but is not suitable for critical situations.

- Scenario 3: the system detects people close to security perimeters.
  - *Monitoring signal*: the value of the *Suspicious Behaviour* score obtained from the **MultiCameraTracker** component. The score is derived from spatial and temporal features calculated by the human tracking component to reflect potential risk based on location and/or abnormal behaviour.
  - *Reconfiguration action*: if the *Suspicious Behaviour* score exceeds a set threshold, the following reconfiguration sequence is sent to the nodes with the highest scores:
    1. Obtain reconfiguration identifier from the cloud.
    2. Reconfigure **Scaler** for high-definition (HD) video at 30 FPS.
    3. Perform all video-surveillance tasks on HD video.
    4. Stream HD video to human operator for confirmation.
  - *Outcome:* the system performs all video-surveillance tasks (detection, re-identification, and tracking of people) on HD video, providing full video-surveillance functionality with focus on high accuracy.

5.5.2.2 Application in context of UC3 (Habit Tracking)

In the case of UC3, the vision component will be focused on the classification of a person's behaviour while indoor. It is based on the ability to understand human actions and their purpose, and usually comprises: a) extraction of features from video sequences, and b) classification and labelling of actions using the features extracted in the first step.

In recent years, several authors have been trying to identify human actions from several sources and using different technologies. One of the approaches is sensor-based activity recognition, which handles data that comes from smartphones, watches, Wi-Fi or Bluetooth [HAY15]. Another approach is using raw video as an input for perform

© FitOpTiVis Consortium (Public Document)

human action recognition (HAR) [WAN18]. In relation to HAR, various authors have used hand-crafted feature-based methods [KAN14, LAN15] and deep learning [SIM14, ULL18].

Advances to state-of-the-art in research on HAR hinge on availability of many different datasets that can be used by researchers to test their approaches [ZHU18]. Some of the video datasets are HMDB [KUE11], UCF-101 [SOO12], Charades [SIG16], Moments in Time [MON19], and Kinetics [KAY17]. The main objective of these datasets is to also provide a solution (as in the case of ImageNet [DEN09]), so that researchers can use a pre-trained model on a large action-video dataset such as Kinetics, which allows performing transfer learning, or fine tuning it to achieve a satisfactory result in other problem related to action recognition in a shorter period of time [CAR17].

After investigating the state-of-the-art in HAR, we have decided to adopt an approach based on deep learning, because of its accuracy on complex action-recognition tasks (with more than 600 different classes) [TRA18].

We have considered different deep-learning neural-network architectures, looking for one that best fits our indoor-action-recognition system. In particular, we have evaluated LRCN [DON15], 3D-ConvNet [TRA15], Two Stream [SIM14], 3D Fused Two-Stream [FEI16], and Two Stream i3D [CAR17]. The results in Table 7 show that the Two Stream I3D architecture outperforms the rest of the architectures on samples from the UCF-101 and HMDB51 datasets.

Table 7. Accuracy of different neural-network architectures for HAR

Architecture	UCF-101			HMDB51		
	RGB	Flow	RGB+Flow	RGB	Flow	RGB+Flow
LRCN	81.0	-	-	69.9	-	-
3D-ConvNet	51.6	-	-	60.0	-	-
Two-Stream	83.6	85.6	91.2	70.1	58.4	72.9
3D-Fused	83.2	85.8	89.3	71.4	61.0	74.0
Two-Stream I3D	84.5	90.6	93.4	74.1	69.6	78.7

The Two Stream I3D architecture consists of two independent 3D-ConvNet networks. One receives the input video in RGB, while the other is fed an Optical Flow estimation of the video Stream. The output of both networks can be fused in the last stage to get the Two Stream I3D network. This neural network model is pre-trained on over more than 600 classes. For this reason we will work with the Two Stream I3D model, because it allows using either one stream or both streams depending on the desired accuracy. We will be adapting the original neural network architecture to provide good performance in our Habit Tracking System.

For behaviour classification, the actions to be studied are determined by the task. Potential actions identified at this point could be cooking, preparing coffee/tea, eating, walking, cleaning the floor and actions that will trigger alarms such as accidental fall,

© FitOpTiVis Consortium (Public Document)

fainting, or lying on the floor (for the indoor scenario). The action classifier does not need to run constantly, it requires a video sequence to do the action inference, which is just a label. With respect to the qualities to be used, **precision** (fraction of positive labels that are correctly classified) and **recall** (fraction of real positives that were correctly labelled) are the most common metrics in classification tasks. These metrics allow us to develop different several neural models and compare them to choose the most appropriate one for each scenario.

Performance of the action classification process (actions labelled per second or fixed number of frames) depends on the number of actions to be classified (number of different labels that determine the complexity of the classification method). This parameter can be adapted to achieve the desired performance or other quality goals.

Moreover, the number of visual features for the classification will depend on the complexity of the used neural network architecture, and can be also adapted based on the number of classes. This will also determine the performance of the final network.

Although it is very seminal at this point, we are also considering features that can be estimated at the node, and how to send only the relevant ones to the cloud to do the final processing. In this case, not only performance is considered; but also privacy, avoiding the transmission of full images to the cloud.

We plan to perform the computation of the neural models at the edge node, and send the alarms of the critical actions detected, as well as other monitoring parameters, to the cloud.

Modelling habit-tracking action-recognition components

The goal of the system is to monitor elderly people in their own homes, recognizing potentially critical actions and situations in their daily lives. Similarly to the video-surveillance system in UC9, the model of the habit-tracking system is split between application and platform components. An overview of the model is shown in Figure 36. We again present only a brief description of the individual components—additional details can be found in the QRML model files.

Application components

- **RGBCameraApp**: provides RGB video input at 25 fps.
- **Preprocessor**: pre-processes the video stream so as to make it suitable for processing and analysis by a neural network, such as batch normalization.
- **RGBActionRecognizer**: processes the video frames of a captured action and outputs confidence levels for multiple possible action labels. The component is reconfigurable, and implements different neural networks, providing different levels of accuracy (calibrated on publicly available data sets) with different energy requirements. In particular, the component supports three configurations to guarantee real-time performance in various situations, trading accuracy for power consumption and vice-verse:
    - Configuration 1 achieves the lowest power consumption, because it only uses simple (and computationally least demanding) neural-network models. Here the low power consumption comes at the expense of lower accuracy due to use of simple models.
    - Configuration 2 uses more complex models to provide better accuracy, at the cost of higher power consumption.

© FitOpTiVis Consortium (Public Document)

  - Configuration 3 achieves the best accuracy, because it uses the most complex (and computationally most demanding) neural-network models. Here the accuracy comes at the expense of high power consumption.
- **ActionEvaluator**: uses the confidence levels assigned to different action labels to decide whether to engage the **OpticalFlowCalculation** and **OPFActionRecognizer** components to ensure that a particular action really occurred.
- **OpticalFlowComputation**: calculates Optical Flow on the cloud server (with more computing resources) to satisfy real-time requirements.
- **OPFActionRecognizer**: performs action recognition on the results of the Optical Flow calculation and outputs confidence levels for possible action labels.
- **ResultsFuser**: combines results of the RGB-based and (optionally) the Optical Flow-based neural networks to obtain final confidence levels for action labels. These are sent to the FIVIS system.

Platform components

- **RGBCamera**: physical camera which provides video input to the system.
- **JetsonXavier**: edge node implementing action recognition on an NVidia® Jetson Xavier board. The edge node sends monitoring information about energy consumption and CPU/GPU unit temperatures to the FIVIS system.
- **CloudCompute**: central node, more powerful in terms of computing resources, which mainly performs Optical Flow calculations on a set of frames.

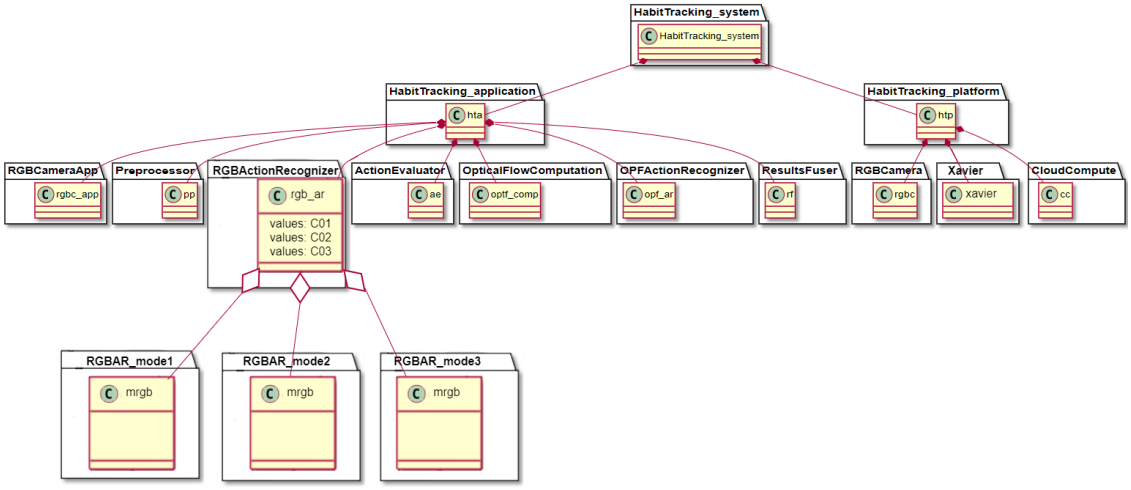


Figure 32. QRML model of the components of the habit-tracking system (UC3)

The Nvidia Jetson Xavier is an embedded compute device with 8 ARM cores and 512 CUDA cores with support for Tensor Flow. It performs well on deep-learning tasks, and provides enough computing to enable real-time operation. The device is fully-configurable and allows its performance to be adjusted at runtime by varying the number of processing cores, and the working frequency of both the CPU and GPU units.

The cloud server (**CloudCompute** component) is equipped with an Nvidia RTX2080 Ti GPU with more than 4000 CUDA cores. The server is used to perform complex computations that cannot be performed on the edge nodes. In particular, the server hosts the **OpticalFlowComputation** component, which computes Optical Flow on a set

© FitOpTiVis Consortium (Public Document)

of frames. The edge nodes use the *pocl-remote* framework to perform computation on the server.

As mentioned earlier, the **RGBActionRecognizer** component supports three configurations which use different neural network models to enable trade-off between action-recognition accuracy and power consumption. This is important to allow the system to adapt to different situations. In particular, if the active model cannot distinguish between two actions for a certain period of time, or if a confirmation is needed when a person has fallen down and is lying on the floor, a configuration with a more complex neural network model (operating at higher frequencies and with more resources) is activated to achieve better accuracy. When no critical actions take place and if there is no significant variation in the action-recognition confidence (e.g., if a person is watching TV), a configuration with less complex neural network (operating at lower frequencies and with less resources) can be activated, leading to reduced power consumption.

System Adaptation and Reconfiguration

Similar to the video-surveillance system, the habit-tracking system adapts to different situations by switching to a configuration appropriate for the current situation. The following scenarios reflect situations which trigger system reconfiguration:

- Scenario 1: the system cannot decide which action is taking place, because at least two different action labels were getting similar maximum confidence scores over a period of time.
    - *Monitoring signal*: confidence levels of recognized actions stored in the FIVIS system, and a computed signal indicating that at least two actions have similarly high confidence.
    - *Reconfiguration action*: if there are two or more ambiguous actions for a certain period of time, attempt to disambiguate by using a more complex neural network model, triggering the following reconfiguration sequence:
        1. Obtain reconfiguration identifier from the cloud.
        2. Instruct the **RGBActionRecognizer** component to load a more complex neural network model while processing the incoming video frames using the current neural network model. This is necessary to keep the system operational, because loading and preparing a new neural network model for execution takes several seconds.
        3. Once the new model is ready for execution, switch processing in the **RGBActionRecognizer** component to the new (more complex) model, and stop execution of the previous model.
        4. Increase the operating frequency of the CPU and GPU units on the (Jetson Xavier) edge node.
    - *Outcome*: the system uses a more complex neural network model in an attempt to disambiguate actions, at the cost of higher power consumption. This adaptation action can be repeated until the system reaches configuration with the most complex neural network model.

- Scenario 2: no discernible action appears to be taking place—the confidence levels for all actions are similar for a period of time (no action is reaching a high confidence value), there is no movement in the scene.

© FitOpTiVis Consortium (Public Document)

- *Monitoring signal*: confidence levels of recognized actions stored in the FIVIS system, and a computed signal indicating that all actions have similar confidence levels (no action has a significantly high value than others).
- *Reconfiguration action*: if there is no discernible action, reduce power consumption by using a less complex neural network model, triggering a reconfiguration sequence similar to Scenario 1:
  1. Obtain reconfiguration identifier from the cloud.
  2. Instruct the **RGBActionRecognizer** component to load a less complex neural network model.
  3. When ready, switch processing in the **RGBActionRecognizer** component to the new (less complex) model.
  4. Decrease the operating frequency of the CPU and GPU units on the (Jetson Xavier) edge node.
- *Outcome*: the system uses a less complex neural network model, reducing power consumption, and eventually system temperature. This adaptation action can be repeated until the system reaches configuration with the least complex neural network model.

- Scenario 3: the energy consumption and system temperature is too high (for a period of time) and action must be taken to prevent system failure or running out of battery.
  - *Monitoring signal*: energy consumption and temperature readings stored in the FIVIS system, and computed signals for each quantity indicating that a specific threshold has been exceeded over the last N minutes.
  - *Reconfiguration action*: if both signals (for energy consumption and system temperature) indicate that a threshold has been exceeded for a period of time, the following reconfiguration sequence is triggered:
    1. Obtain reconfiguration identifier from the cloud.
    2. Change the Jetson Xavier performance mode to one with lower operating frequency of the CPU and GPU units. This change (unlike switching between neural network models) is almost instantaneous.
  - *Outcome:* the overall performance of the system is reduced in exchange for reduced power consumption, and eventually lower temperature.

5.5.3 Distributed Image Pre-Processing and Optimized Image Segmentation

Multiple view geometry is a complex and resource-demanding task. Thus, image pre-processing, such as undistorting and segmenting, has to be carried out in the most efficient way. Nonetheless, precision in the segmentation process is key to offer accurate results that truthfully represent the reality. Specially, when the application is focused on industrial inspection.

As pointed out by Shi et al. [SHI16], in a system where several images taken by a number of devices have to travel to a single node, an edge-computing approach can reduce latency and bandwidth usage, while increasing throughput. This approach is based on the principle that the workload should be finished in the nearest layer with enough computation capability to the things at the edge of the network. This translates into providing the cameras with computation capability in our envisioned application. We propose a distributed image processing pipeline where low-power execution boards are

© FitOpTiVis Consortium (Public Document)

in charge of performing an initial image processing and a fast segmentation in order to increase throughput and reduce energy consumption.

Another approach for image processing distribution is based on the remote OpenCL-based software stack described in Section 4.1. A modification of this framework enables image processing pipelines to be distributed among several computational nodes located anywhere in the network. We were investigating the suitability of this approach for our foreseen final application (3D industrial inspection system), but this approach requires sending the images through the network, which increases bandwidth usage, rendering it a non-viable solution.

Therefore, we proposed a distributed image processing pipeline where low-power execution boards are in charge of performing an initial image processing and a fast segmentation in order to increase throughput and reduce energy consumption. These low-power boards are installed close to the cameras, thus, the first layer with computation capabilities is located immediately after images are captured.

The diagram in Figure 37 shows a typical configuration of this kind of system, where the number of low power execution boards and cameras can be decided at design time, while we include a new element, a 'dispatcher' to perform workload decisions at runtime.

Throughput can be increased with the distributed segmentation, as the low-power

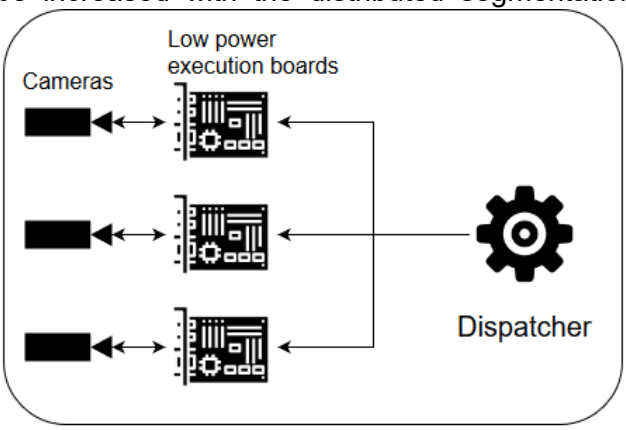


Figure 33. Typical configuration of an industrial inspection system.

execution boards can segment new captures while the main computing node is working on the previous capture.

As the first step, several low-power execution boards were evaluated on image segmentation workload to identify those most suitable in terms of computation power and cost trade-off. The table below shows the evaluated boards along with the achieved average image segmentation time in milliseconds.

Table 8. Segmentation performance and cost of low-power execution boards

Board	**Segmentation time [milliseconds]**	**Cost (approx.) [euro]**
Espresso bin	$\mu = 1045.37$; $\sigma = 2.89$	44€

© FitOpTiVis Consortium (Public Document)

Grapeboard	μ = 1839.74, σ = 23.88	190€
Raspberry Pi	μ = 3450.43; σ = 55.86	Discontinued
Raspberry Pi (optimised, neon flag)	μ = 2603.37; σ = 19.47	Discontinued
Raspberry Pi 3	μ = 1554.24; σ = 87.11	30€
Raspberry Pi 3 (optimised, neon and tune flags)	μ = 758.04; σ = 21.09	30€
Nvidia Jetson TX2	μ = 130.78; σ = 5.26	500€
Nvidia Jetson Nano	μ = 180.83; σ = 11.64	99€

The times shown in Table 8 above were obtained using the same segmentation algorithm that is currently deployed on the main computing platform. No platform-specific optimizations were applied to the algorithm.

An example of one of the hardware configurations used is shown in Figure 38 below.

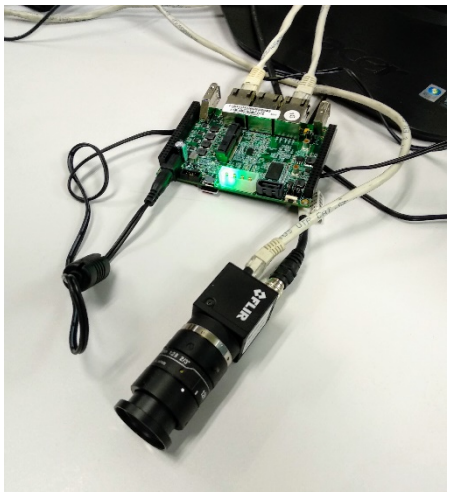


Figure 34. Marvell ESPRESSObin board connected to a HD camera.

Based on the results of the image segmentation benchmarks, the Nvidia Jetson Nano platform was selected as a cost-efficient solution providing sufficient performance. After selecting the hardware platform, the segmentation algorithms written in Python were ported to C++ and augmented with several optimizations. In order to provide hardware virtualization for the system (allowing to use other edge boards), two algorithm implementations were created: one for a CPU and one for a GPU using CUDA.

Jetson Nano capabilities using parallelization were also analysed. The results obtained are presented in Table 9.

Table 9. Nvidia Jetson Nano parallelization comparison

Number of processes	Hardware used	Time in ms
1	GPU	85
2	GPU	160
2	GPU + CPU	90/140

4	GPU	330
4	GPU + CPU	175/240
8	GPU	N/A (Crashed)
8	GPU + CPU	320/540

To plan which phases of the segmentation process should be prioritized during the optimization of the algorithm on the most promising boards, the average execution time of each phase of the algorithm was measured on the low-power execution boards. Table 10 below shows the fraction of time spent on each of the sub-tasks (only OpenCV related) in the segmentation process for the Espresso Bin board. The results are similar for the other boards tested.

Table 10. Fraction of execution time spent in different phases of the segmentation algorithm running on the Marvell EspressoBin board.

Background diff.	Blur	Erosion & Dilation	Finding contours	Gaussian Filter	Thresholding
3.63%	7.55%	29.04%	8.79%	47.37%	3.62%

Based on these results, the algorithm could benefit most from (platform-specific) optimizations in the ‘Erosion & Dilation’ and the ‘Gaussian Filter’ phases. To optimize these two phases, we are working on specific modifications of the OpenCV API to adapt to the particularities of the most promising boards

The second innovation we achieved by employing low-power execution boards installed close to the cameras is the reduction in bandwidth usage. Because images travel from the low-power boards to the main computing node already segmented, less bandwidth is used. As an example, the two images in Figure 39 below show a part processed by the system. The image on the left shows the raw data captured, while the image on the right shows the segmented image, reducing the total size of the image by about 30%. This kind of bandwidth reduction is extremely important, especially when using many cameras.

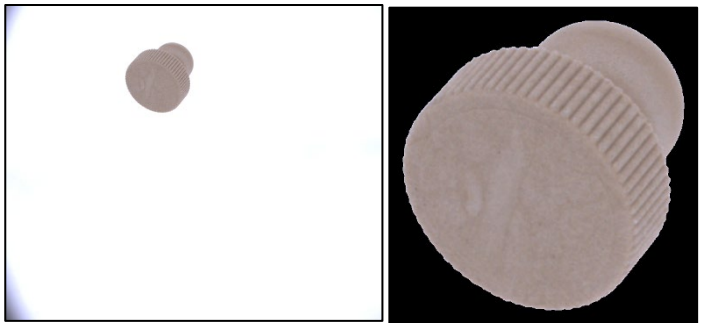


Figure 35. Example of image before (left) and after (right) segmentation

Moreover, low-power execution boards are capable of detecting incorrect captures and asking the capture system to retry a new capture of the same part. This decision is taken without the information travelling from the cameras to the computation cluster, which reduces bandwidth consumption even further.

© FitOpTiVis Consortium (Public Document)

All optimizations will be implemented in the segmentation algorithm in subsequent work, which will allow us to evaluate these innovations in comparison with the currently used approach. The optimization is focused on the following metrics:

- **Average throughput**: the number of parts processed per unit of time. To obtain a relevant evaluation, a variety of tasks with different types of parts has to be employed. This variety should also include parts that are prone to produce incorrect captures due to their shape.
- **Bandwidth usage**: the average number of bits per unit of time that are transferred from the capturing devices to the main computation node.
- Latency: the time that the system takes to process a new capture.

The diagram in Figure 40 below shows the architecture of the system.

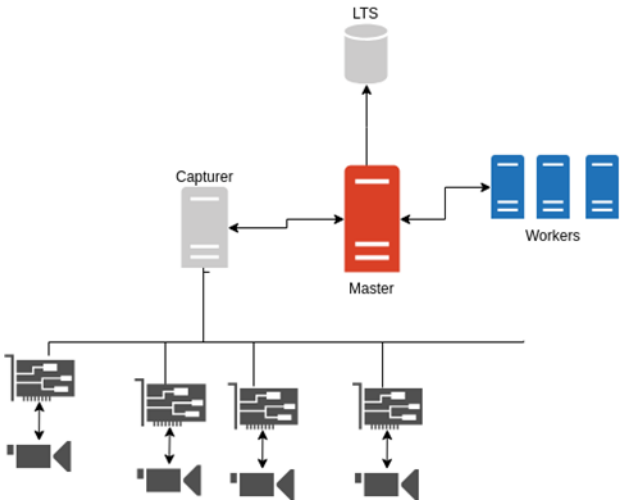


Figure 36. Capture system architecture.

The ‘Master’ node show in the diagram is a critical element for the runtime support and includes the workload ‘Dispatcher’ explained before. First, the ‘Master’ communicates with the ‘Capturer’ to request a new shoot from the cameras. When the edge boards send the images taken by the cameras, the ‘Capturer’ stores them in a queue which the ‘Master’ node is able to read, and from which the ‘Master’ distributes the captured images among worker agents (‘Workers’) who perform the computationally-intensive tasks.

© FitOpTiVis Consortium (Public Document)

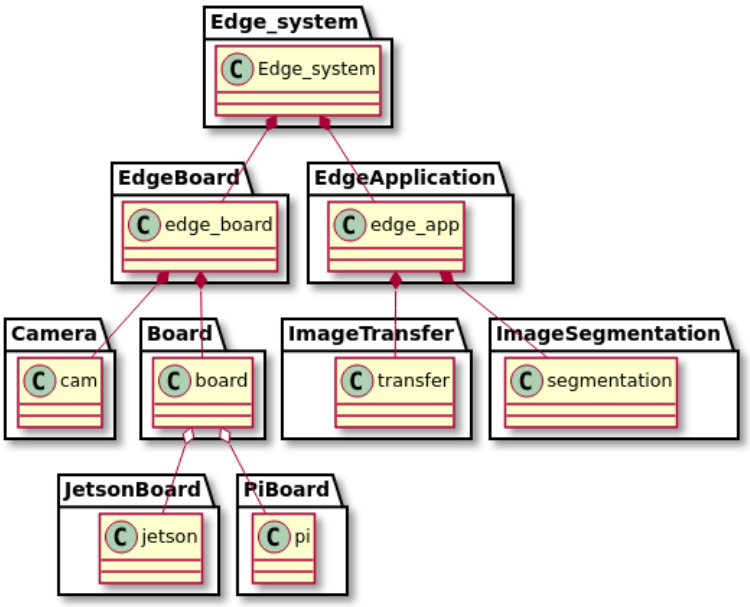


Figure 37. QRML model of the Edge component

The Edge component of the whole solution has been modelled using QRML. The diagram in Figure 41 above shows two differentiated branches: one for the hardware and one for the software. The hardware side abstracts from cameras and boards and considers different types of boards as alternatives. The software side shows two key processes related to bandwidth usage, which can be reconfigured depending on system load—one responsible for image transfer, and the other responsible for image segmentation.

Finally, the aforementioned metrics are being monitored to support reconfiguration of the system to optimize resources. The 3D Industrial Inspection use case solution requires a local store to analyse and query information to make reconfigurations in the critical execution pipeline. The logic that makes decisions about reconfiguration is implemented locally, in the same hardware that executes the pipeline and so that no delay is introduced due to communications with external servers.

In addition, monitoring data will be stored in an external platform such as FIVIS to provide operators with an overview of the system's operation and performance, but the external system will not be used to trigger reconfiguration.

Reconfiguration scenarios

Having taken the above mentioned requirements into account, the system supports the following reconfiguration scenarios:

1. **Initial (start-up) configuration**. When the system is starting, all the edge boards report the following information:
    - RAM capacity
    - GPU available
    - MTU of the Ethernet connection

Depending on the information from edge boards, the system selects the segmentation algorithm and the set of operations to be performed on the boards:

- The amount of RAM installed on the board determines whether the board will perform segmentation or not. To enable segmentation, the board must have more than 1GB of RAM.
- If GPU is available, segmentation will be performed on the GPU, otherwise it will be performed on the CPU.
- The MTU of the network connection determines the encoding of color information in the output image. If MTU is less than 4000 bytes, colors will be encoded using the Bayer format, otherwise raw RGB encoding will be used.

2. **Avoiding segmentation due to processing delays.** If the system detects that one or more boards are causing delays, it avoids image segmentation on those boards. Delays are detected by measuring the latency between receiving the signal to capture a new image and finishing the pre-processing of the whole image. The latency is measured internally by software running on each of the capture boards:
    - If the latency is greater than 300 ms the camera will skip the segmentation process.

3. **Sending RAW image.** When a board detects that the detected region of interest (ROI) is similar to the original image size, the image is sent “as captured”, in the RAW12 format (12-bit Bayer) produced by the camera. This will save bandwidth, because the detected ROI is encoded using 8-bit RGB, requiring 24 bits per pixel, whereas the RAW12 format only uses 12 bits per pixel. The downside is the need for additional processing on the worker nodes (and wasted conversion to 8-bit RGB for ROI detection on the edge boards), because the image eventually needs to be converted to RGB for processing.
    - If the image ROI is greater than 50% of the original size then the image is sent in the RAW12 format to save bandwidth, and the conversion from 12-bit Bayer format to 8-bit RGB is delegated from the edge board to the worker node.

5.5.4 Selective On-Demand Resource Loading

To achieve near real-time (soft real-time) performance on low-power mobile platforms, such as the HURJA’s Salmi Augmented Reality (AR) system, we plan to utilize smart feature extraction, segmentation, and classification algorithms to reduce bandwidth usage by only sending the necessary parts of images/videos.

Specifically in the context of the Salmi AR system, a mobile application called Extent can (upon request) download a JSON packet which consists of a list (descriptions) of wakeup images, objects, entities, and actions. Either the request can come from the Salmi MAPS website, from the Salmi AR mobile application, or directly from the Extent mobile application if the “free roam” state has been switched on (requires GPS). End-users have the option to switch the “free roam” state off at any time and when this

© FitOpTiVis Consortium (Public Document)

happens, the Extent mobile application downloads new content only upon request from an external source (currently only the Salmi system related sources are available). The Extent mobile application downloads all required wakeup images, 3D-models, textures, audio files, videos, etc. based on the instructions received via JSON packet.

To optimize the run-time performance of the Salmi AR system, all of these packets can be downloaded in advance. All files will be saved locally into end-users' mobile device (smart phone or tablet) and those will be shown to end-users based on instructions received via JSON packets as soon as matching wakeup image, object, entity, or action has been found, or when an end-user is within a certain pre-defined distance from the target. Free roam data will be removed on-the-fly from end-users' devices when each session ends. The Extent mobile application is currently being developed using C# programming language on top of the Unity 3D engine and the server back-end side is currently being developed using PHP. During our early testing phase, all description packets are in JSON format.

The runtime state of the system includes measured performance and energy usage, which can be handled by a generic data model. Relevant metrics to be monitored/evaluated are the following:

- Near real-time (soft real-time) performance: System performance can be monitored/evaluated in terms of frames-per-second or kilobits-per-second, but AR-feature robustness/performance depends highly on the selected AR-glass model. We plan to start development with state-of-the-art Magic Leap and/or HoloLens 2 glasses to ensure that all possible use cases can be implemented easily. Later on we plan to investigate the use of other (cheaper and less powerful) AR-glass options that may require more optimization of the system code to achieve the level of performance comparable with the high-end, state-of-the-art AR-glasses.
- Optimal energy usage: It is not an easy task to calculate the initial energy usage for the whole Salmi AR system before the first MVP version is fully implemented, but continuous camera feed and required advanced algorithms will present a challenge in terms of optimizing the energy usage of the system as a whole. As soon as the first MVP version is ready, we will perform extensive measurements on power usage and based on the achieved results, we will make adjustments to the implemented algorithms to enable optimal energy usage of Salmi AR system.

In addition, the system monitors the achieved level of satisfaction of all end-user groups that can be handled by a generic data model:

- The intended users of the Salmi AR system will be brain damage patients (assisted living), elderly people (assisted living), relatives (monitoring and situational awareness), nurses (home visits), and doctors (emergency cases). We have made careful plans to achieve the required level of satisfaction for all of these end-users of our Salmi AR system. However, when our first MVP version will be ready by June 2019, we cannot yet completely fulfill all of the below-mentioned end-users requirements or all the needed features, but by the end of the project, we will have fully functional version of Salmi AR system that fulfils the level of satisfaction for all of these end-user groups.

© FitOpTiVis Consortium (Public Document)

5.5.5 Algorithms and Techniques to Achieve Real-Time Performance for PCC Demo System

With the selection of MPEG V-PCC as PCC encoding and distribution scheme, Nokia can utilise available video hardware decoders to carry the major load in the decoding process. Figure 42 illustrates the current V-PCC decoding scheme block diagram, where available hardware decoding support is marked in teal.

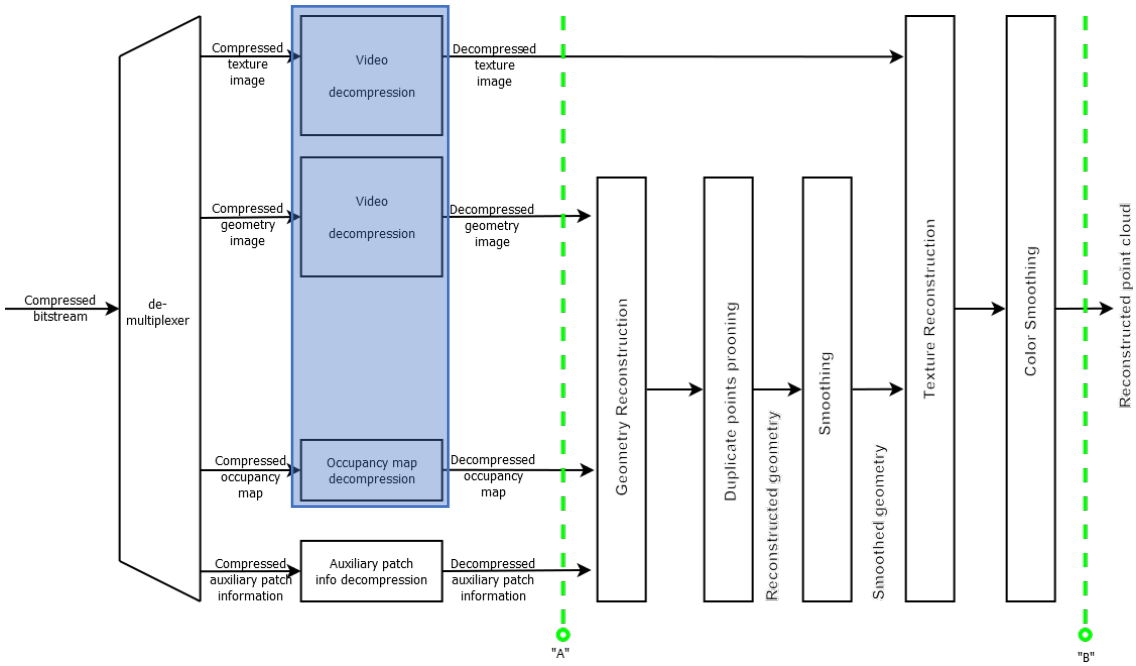


Figure 38. V-PCC TMC2 decoding structure.

For the three decoding instances, texture, geometry and occupancy video decompression, not much special attention on real-time capability is required, as the current video decoding hardware can achieve much higher levels of decoding performance than demanded by the use case. However, attention is required due to three simultaneously running video decoder instances, which must be synchronised. This aspect and any possible implications must be further investigated within FitOptiVis.

As for the real-time decoding of auxiliary patch information, little is known so far, and detailed experiments have to be carried out to assess any implications on real-time performance, e.g. maximum number of patches per frame, inter-prediction between patch auxiliary information, random access structures, etc. This investigation will also be part of the planned FitOptiVis research topics.

Finally, decoding and rendering altogether has to happen in real-time. Thus, any unnecessary data transfers, e.g. copying 3D point cloud data from the CPU to the GPU for rendering, should be avoided. Therefore, we envision V-PCC decoding straight into the GPU memory, as well as tools for partial and simultaneous decoding and playback. Such tools, together with support of the hardware video decoders, should ensure real-time capability of our PCC demo system.

© FitOpTiVis Consortium (Public Document)

6. **Conclusion**

This document summarizes the outcomes of Task 4.1 from the first two years of the project. We deal primarily with two aspects of runtime support for adaptive applications developed using the FitOptiVis approach.

The first aspect concerns runtime platforms on which the applications execute. To establish a consortium-wide awareness and agreement on platform components (as defined in deliverable D2.1) developed within the project, we present an overview of the available runtime platforms. Each platform is suitable for different kind of applications, with requirements at different levels of abstractions, and operating at different time scales.

During the second year of the project, there have been significant advances in many of the platforms. For example, the OpenCL-based Heterogeneous Distributed Software Runtime (pocl-remote, Section 4.2) added a low-overhead control protocol and event-based synchronization, the Extended OpenMP Runtime Infrastructure (Section 4.3) managed to solve the issues related to dynamic offloading of OpenMP in fat binaries and added support for pocl-remote. Deterministic Networking (Section 4.6) has become a platform component, and made progress on TSN bridge design, supporting several use cases. Most of the platforms are available to project partners, but some are still not mature enough for consortium-wide release. This concerns, e.g., the Managed-Latency Edge-Cloud Environment (Section 4.1), where development focused on components critical for adaptation (i.e., performance predictor) after completing an initial prototype. In the final year, we will focus on making all the runtime platforms available to partners in the project and provide assistance to partners targeting specific platform components. Where applicable, contributions to relevant open-source code bases will be made.

The second aspect concerns runtime adaptation and comprises two parts. The first part presents some of the mechanisms through which platforms enable adaptation. Here we emphasize progress on the performance prediction of co-located applications in the Managed-Latency Edge-Cloud Environment (Section 5.1.3), analytical methods for budget matching on the CompSOC platform (Section 5.2.4), reconfigurable neural network accelerators designed using Multi-Dataflow Composer (Section 5.3.3), and support for reconfigurable floating-point accelerators on the Xilinx Zynq platform (Section 5.4).

The second part presents application-specific adaptation scenarios from use case owners participating in WP4. During the second year of the project, most of the scenarios have been elaborated in more detail, providing specific requirements, system models based on QRML (the FitOptiVis quality and resource modelling language), or information about signals and conditions triggering reconfiguration (Sections 5.5.1, 5.5.2, and 5.5.3). The use case requirements and adaptation scenarios proved invaluable in steering the development of some of the runtime platforms and interfaces for use by adaptive resource managers developed in the context of other WP4 tasks.

Some of the platforms and applications have already adopted the reference architecture concepts from WP2, making them amenable to tool support, especially where it concerns design-time analysis and optimizations. In the final year of the project, we will finalize the instantiations of the runtime platforms within the framework of the reference architecture.

© FitOpTiVis Consortium (Public Document)

References

[ADA17] O. Adam, Y. C. Lee, A. Y. Zomaya. CtrlCloud: Performance-Aware Adaptive Control for Shared Resources in Clouds. In Proc. CCGrid, IEEE, 2017, pp. 110–119.

[AMI17] M. Amiri, L. Mohammad-Khanli. Survey on prediction models of applications for resources provisioning in cloud. Journal of Network and Computer Applications 82 (2017)93–113.

[ARROW] Documents for the Arrowhead Framework [Online]. https://forge.soa4d.org/docman/?group_id=58

[BAR14] F. Barranco, Javier Diaz, Begoña Pino, and Eduardo Ros. "Real-time visual saliency architecture for FPGA with top-down attention modulation." IEEE Transactions on Industrial Informatics 10, no. 3 (2014): 1726-1735.

[BYS10] M. Bystrom, I. Richardson, S. Kannangara, and M. de-Frutos-Lopez. 2010. Dynamic replacement of video coding elements. Image Commun. 25, 4 (April 2010), 303-313.

[BUR01] W. Burleson et al.: Dynamically parameterized algorithms and architectures to exploit signal variations for improved performance and reduced power. In Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing. IEEE, 2001, pp. 901–904 vol.2.

[CAR17] J. Carreira, A. Zisserman: Quo vadis, action recognition? A new model and the Kinetics dataset. In Proc. IEEE Conference on Computer Vision and Pattern Recognition, IEEE, 2017, pp. 6299–6308.

[CHE11] Chen, Yen-Lin, Bing-Fei Wu, Hao-Yu Huang, and Chung-Jui Fan. "A real-time vision system for nighttime vehicle detection and traffic surveillance." IEEE Transactions on Industrial Electronics 58, no. 5 (2011): 2030–2044.

[CHE15] X. Chen, L. Rupprecht, R. Osman, P. Pietzuch, F. Franciosi, W. Knottenbelt. CloudScope: Diagnosing and Managing Performance Interference in Multitenant Clouds. In Proc. MASCOTS, 2015, pp. 164–173.

[CHE18] T. Chen, R. Bahsoon, X. Yao. A Survey and Taxonomy of Self-Aware and Self-Adaptive Cloud Autoscaling Systems. ACM Comput. Surv. 51 (2018) 61:1–61:40.

[CHI17] L. Chittka, P. Skorupski: Active vision: a broader comparative perspective is needed. 2017.

[COM00] D. Comaniciu, V. Ramesh. Mean shift and optimal prediction for efficient object tracking. In Proc. Intl. Conference on Image Processing, IEEE, 2000, pp. 70–73.

[DEL13] C. Delimitrou, C. Kozyrakis. Paragon: QoS-aware Scheduling for Heterogeneous Datacenters. In Proc. ASPLOS, ACM, 2013, pp. 77–88.

[DEL14] C. Delimitrou, C. Kozyrakis. Quasar: Resource-efficient and QoS-aware Cluster Management. In Proc. ASPLOS, ACM, 2014, pp.127–144.

[DEN09] J. Deng,W. Dong, R. Socher, L. J. Li, K. Li, L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In Proc. IEEE Conf. on Computer Vision and Pattern Recognition. IEEE, 2009, pp. 248–255.

© FitOpTiVis Consortium (Public Document)

[DON15] J. Donahue, L. Anne-Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In Proc. IEEE Conf. on Computer Vision and Pattern Recognition. IEEE, 2015, pp. 2625–2634.

[FAN15] F. Faniyi, R. Bahsoon. A Systematic Review of Service Level Management in the Cloud, ACM Comput. Surv. 48 (2015) 43:1–43:27.

[FEI16] C. Feichtenhofer, A. Pinz, A. Zisserman. Convolutional two-stream network fusion for video action recognition. In Proc. IEEE Conf. on Computer Vision and Pattern Recognition. IEEE, 2016, pp. 1933–1941.

[FUJ10] T. Fujita, K. Chiba, and Claudio Privitera. "Bottom-up regions-of-interest in observation of robot hand movement: Comparisons with humans. In Proc. IEEE Intl. Conf. on Systems, Man and Cybernetics, IEEE, 2010, pp. 3768–3773.

[GAR14] M. García-Valls, T. Cucinotta. C. Lu. Challenges in real-time virtualization and predictable cloud computing. Journal of Systems Architecture 60 (2014) 726–740

[GOD12] A. B. Godbehere, A. Matsukawa, K. Goldberg: Visual tracking of human visitors under variable-lighting conditions for a responsive audio art installation. In Proc. American Control Conference. 2012.

[GOV11] S. Govindan, J. Liu, A. Kansal, A. Sivasubramaniam. Cuanta: Quantifying effects of shared on-chip resource interference for consolidated virtual machines. In Proc. SOCC, ACM, 2011, p. 22.

[GUO16] L. Guo, D. Xu, Z. Qiang. Background Subtraction Using Local SVD Binary Pattern. In Proc. IEEE Conf. on Computer Vision and Pattern Recognition Workshops. IEEE, 2016, pp. 1159–1167.

[HAY15] T. Hayashi, M. Nishida, N. Kitaoka, K. Takeda. Daily activity recognition based on DNN using environmental sound and acceleration signals. In Proc. 23rd European Signal Processing Conference. IEEE, 2015, pp. 2306–2310.

[HUE02] C. Hue, J. P. le Cadre, P. Pérez. Tracking multiple objects with particle filtering. IEEE Transactions on Aerospace and Electronic Systems, 2002.

[MAR11] J. Mars, L. Tang, R. Hundt, K. Skadron, M. L. Soffa. Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-locations. In Proc. MICRO, ACM, 2011, pp.248–259.

[GOO17] K. Goossens, M. Koedam, A. Nelson, S. Sinha, S. Goossens, Y. Li, G. Breaban, R. van Kampenhout, R. Tavakoli, J. Valencia, Ahmadi Balef, Hadi, B. Akesson, S. Stuijk, M. Geilen, D. Goswami, M. Nabi. NOC-Based Multi-Processor Architecture for Mixed Time-Criticality Applications. In Handbook of Hardware/Software Codesign, Springer, 2017.

[HAM16] A. Hameed, A. Khoshkbarforoushha, R. Ranjan, P. P.Jayaraman, J. Kolodziej, P. Balaji, S. Zeadally, Q. M.Malluhi, N. Tziritas, A. Vishnu, S. U. Khan, A. Zomaya. A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems. Computing 98 (2016) 751–774.

[HAO18] T. Hao, Q. Wang, D. Wu, J. S. Sun. Multiple person tracking based on slow feature analysis. Multimedia Tools and Applications, 77(3), 3623–3637. 2018.

© FitOpTiVis Consortium (Public Document)

[HEN20] M. Hendriks, M. Geilen, K. Goossens, R. de Jong, T. Basten. Interface Modeling for Quality and Resource Management. arXiv preprint, arXiv:2002.08181, 2020.

[JOINTER] https://youtu.be/w7EoDlxgzI0

[KAD18a] J. Kadlec, Z. Pohl, L. Kohout. Design Time and Run Time Resources for the ZynqBerry Board TE0726-03M with SDSoC 2018.2 Support. [Online]. http://sp.utia.cz/index.php?ids=projects/fitoptivis

[KAD18b] J. Kadlec, Z. Pohl, L. Kohout. Design Time and Run Time Resources for Zynq Ultrascale+ TE0820-03-4EV-1E with SDSoC 2018.2 Support". [Online]. http://sp.utia.cz/index.php?ids=projects/fitoptivis

[KAD18c] J. Kadlec, Z. Pohl, L. Kohout. Design Time and Run Time Resources for Zynq Ultrascale+ TE0808-04-15EG-1EE with SDSoC 2018.2 Support. [Online]. http://sp.utia.cz/index.php?ids=projects/fitoptivis

[KAD18c] J. Kadlec, Z. Pohl, L. Kohout. Design Time and Run Time Resources for Zynq Ultrascale+ TE0808-04-15EG-1EE with SDSoC 2018.2 Support. [Online]. http://sp.utia.cz/index.php?ids=projects/fitoptivis

[KAE02] P. KaewTraKulPong, R. Bowden. An Improved Adaptive Background Mixture Model for Real-time Tracking with Shadow Detection. In Video-Based Surveillance Systems, Springer, 2002, pp. 135–144.

[KAL60] R. E. Kalman. A new approach to linear filtering and prediction problems. Journal of Fluids Engineering, Transactions of the ASME, 1960.

[KAN14] V. Kantorov, I. Laptev. Efficient feature extraction, encoding and classification for action recognition. In Proc. IEEE Conf. on Computer Vision and Pattern Recognition. IEEE, 2014, pp. 2593–2600.

[KAY17] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, M. Suleyman. The kinetics human action video dataset. arXiv preprint, 2017, arXiv:1705.06950.

[KEP03] Kephart, J., Chess, D.: The Vision of Autonomic Computing. Computer. 36, 1, 41–50 (2003).

[KOH18] L. Kohout, J. Kadlec, Z. Pohl. Video Input/Output IP Cores for TE0820 SoM with TE0701 Carrier and and Avnet HDMI Input/Output FMC Module. [Online]. http://sp.utia.cz/index.php?ids=projects/fitoptivis

[KUE11] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, T. Serre. HMDB: a large video database for human motion recognition. In Proc. Intl. Conf. on Computer Vision, IEEE, 2011, pp. 2556–2563.

[LAN15] Z. Lan, M. Lin, X. Li, A. G. Hauptmann, B. Raj. Beyond gaussian pyramid: Multi-skip feature stacking for action recognition. In Proc. IEEE Conf. on Computer Vision and Pattern Recognition. IEEE, 2015, pp. 204–212.

[LIU12] Liu, Zhong, Weihai Chen, Yuhua Zou, and Xingming Wu. "Salient region detection based on binocular vision." IEEE Conference on Industrial Electronics and Applications (ICIEA), pp. 1862-1866. IEEE, 2012.

© FitOpTiVis Consortium (Public Document)

[MAN15] Z. A. Mann. Allocation of Virtual Machines in Cloud Data Centers—A Survey of Problem Models and Optimization Algorithms. ACM Comput. Surv. 48 (2015)11:1–11:34.

[MON19] M. Monfort, A. Andonian, B. Zhou, K. Ramakrishnan, S. A. Bargal, Y. Yan, A. Oliva. Moments in time dataset: one million videos for event understanding. IEEE transactions on pattern analysis and machine intelligence, 2019.

[NAT10] R. Nathuji, A. Kansal, A. Ghaffarkhah. Q-clouds: Managing Performance Interference Effects for QoS-aware Clouds. Proc. EuroSys 2010, ACM, 2010, pp. 237–250

[PFS19] F. Palumbo, T. Fanni, C. Sau, et al. Hardware/Software Self-adaptation in CPS: The CERBERO Project Approach. Intl. Conf. on Embedded Computer Systems, 2019.

[PIC04] M. Piccardi. Background subtraction techniques: a review. In Proc. IEEE Intl. Conf. on Systems, Man and Cybernetics. IEEE, 2004. Vol. 4, pp. 3099–3104.

[REN17] S. Ren, K. He, R. Girshick, J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017.

[SAT17] M. Satyanarayanan. The Emergence of Edge Computing. Computer 50 (2017) 30–39.

[SIG16] G. A. Sigurdsson, G. Varol, X. Wang, A. Farhadi, I. Laptev, A. Gupta. Hollywood in homes: Crowdsourcing data collection for activity understanding. In Proc. European Conf. on Computer Vision. Springer, 2016, pp. 510–526.

[SIM14] K. Simonyan, A. Zisserman. Two-stream convolutional networks for action recognition in videos. In Advances in neural information processing systems, 2014, pp. 568–576.

[SIN15] S. Singh, I. Chana. QoS-Aware Autonomic Resource Management in Cloud Computing: A Systematic Review. ACM Comput. Surv. 48 (2015) 42:1–42:46.

[SME14] A. W. M. Smeulders, D. M. Chu, R. Cucchiara, S. Calderara, A. Dehghan, M. Shah. Visual tracking: An experimental survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2014.

[SOO12] K. Soomro, A. R. Zamir, M. Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. arXiv preprint, 2012, arXiv:1212.0402.

[STA99] C. Stauffer, W. E. L. Grimson. Adaptive background mixture models for real-time tracking. In Proc. IEEE Conf. on Computer Vision and Pattern Recognition. IEEE, 1999, pp. 246–252.

[TE0726] Trenz Electronic: “TE0726 TRM”, [Online]. https://shop.trenz-electronic.de/en/27229-Bundle-ZynqBerry-512-Mbyte-DDR3L-and-SDSoC-Voucher?c=350

[TE0701] Trenz Electronic: “TE0701-06-Carrier-Board” [Online]. https://shop.trenz-electronic.de/en/TE0701-06-Carrier-Board-for-Trenz-Electronic-7-Series?c=261

[TE0808] Trenz Electronic: “UltraSOM+ MPSoC Module with Zynq UltraScale+ XCZU15EG-1FFVC900E, 4 GB DDR4”, [Online]. https://shop.trenz-

© FitOpTiVis Consortium (Public Document)

electronic.de/en/TE0808-04-15EG-1EE-UltraSOM-MPSoC-Module-with-Zynq-UltraScale-XCZU15EG-1FFVC900E-4-GB-DDR4?c=450

[TE080X] Trenz Electronic: “UltraITX+ Baseboard for Trenz Electronic TE080X UltraSOM+” [Online]. https://shop.trenz-electronic.de/en/TEBF0808-04-UltraITX-Baseboard-for-Trenz-Electronic-TE080X-UltraSOM?c=261

[TE0820] Trenz Electronic: “MPSoC Module with Xilinx Zynq UltraScale+ ZU4EV-1E, 2 Gbyte DDR4 SDRAM, 4x5cm”, [Online]. https://shop.trenz-electronic.de/en/TE0820-03-04EV-1EA-MPSoC-Module-with-Xilinx-Zynq-UltraScale-ZU4EV-1E-2-Gbyte-DDR4-SDRAM-4-x-5-cm

[TRA15] D. Tran, L. Bourdev, R. Fergus, L. Torresani, M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In. Proc. IEEE Intl. Conf. on Computer Vision. IEEE, 2015, pp. 4489–4497.

[TRA18] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, M. Paluri. A closer look at spatiotemporal convolutions for action recognition. In Proc. IEEE Conf. on Computer Vision and Pattern Recognition, IEEE, 2018, pp. 6450–6459.

[ULL18] A. Ullah, J. Ahmad, K. Muhammad, M. Sajjad, S. W. Baik. Action recognition in video sequences using deep bi-directional LSTM with CNN features. IEEE Access, (6):1155–1166, 2018.

[WAN18] J. Wang, A. Cherian, F. Porikli, S. Gould. Video representation learning using discriminative pooling. In Proc. IEEE Conf. on Computer Vision and Pattern Recognition. IEEE, 2018, pp. 1149–1158.

[XU18] R. Xu, S. Mitra, J. Rahman, P. Bai, B. Zhou, G. Bronevetsky, S. Bagchi. Pythia: Improving Datacenter Utilization via Precise Contention Prediction for Multiple Colocated Workloads. In Proc. Middleware, ACM, 2018, pp. 146–160.

[YAN13] H. Yang, A. Breslow, J. Mars, L. Tang. Bubble-flux: Precise Online QoS Management for Increased Utilization in Warehouse Scale Computers. In Proc. ICSA, ACM, 2013, pp.607–618.

[ZHU18] J. Zhu, Z. Zhu, W. Zou. End-to-end video-level representation learning for action recognition. In Proc. Intl. Conf. on Pattern Recognition. IEEE, 2018, pp. 645–650.

[ZIV06] Z. Zivkovic, F. van Der Heijden. Efficient adaptive density estimation per image pixel for the task of background subtraction. Pattern Recognition Letters, 27(7):773–780, 2006.

© FitOpTiVis Consortium (Public Document)

A. Review of Virtualization and Resource Management Techniques

This appendix provides a review of the state-of-the art in the area of virtualization and resource management techniques.

A.1 State-of-the-art in Virtualization Techniques

Virtualization refers to the abstraction of a physical component into a virtual object whereby a greater measure of utility can be obtained from the resource component offers [1]. From the hardware perspective, virtualization refers to the abstraction of computer resources whereby applications are decoupled from the hardware they execute on. While the virtualization concept has been there since 1960s, when IBM developed virtualization to enable concurrency by partitioning a mainframe into logical machines [2], it has gained extra attention in the past decade possibly due to the proliferation of cloud services. The main advantages of virtualization are:

- Consolidation: Consolidation refers to bringing together separate parts into a single or unified whole. Virtualization enables consolidation by bringing together several under-utilized execution platforms (i.e., machines) into a single execution platform, thereby reducing operating costs. This has been commonly referred to as multi-tenancy in the literature.
- Isolation: Virtualization enhances security as well as reliability by providing isolated environments where applications running in one virtual execution platform cannot affect applications running in another one. Regarding the security, less-trusted applications can be executed in separate virtual execution platforms, thereby preventing them from accessing and affecting other applications. Virtualization improves the reliability by providing isolated environments where faults and bugs in one environment cannot interfere with other environments.
- Flexibility: Virtualization provides flexible environments for applications where their allocated resources can change dynamically in response to changes in their demands. This includes modifying both the amount of resources and the mapping of virtual resources to physical ones. They are commonly called elasticity and live migration in the literature [3].

Although there are several types of virtualization (such as application virtualization, network virtualization, storage virtualization, etc.), we focus on platform virtualization (also called hardware virtualization or system virtualization in general, and server virtualization in cloud-oriented papers). By platform virtualization, we mean adding a layer between applications and the underlying hardware (called virtualization layer) which creates virtualized environments for applications to be deployed on. Based on the type of this layer, we classify the existing techniques into two classes, namely hypervisor-based virtualization and container-based virtualization, which are elaborated upon in the following sections.

A.1.1 Hypervisor-based Virtualization

For a long time, the term virtualization was used only for hypervisor-based virtualization. The hypervisor, also called Virtual Machine Monitor (VMM), is a software that abstracts

© FitOpTiVis Consortium (Public Document)

the underlying hardware into virtual components called Virtual Machines (VMs). Since the VMs need a complete execution platform (made up of various resources) to run, the hypervisor must virtualize all the underlying hardware resources (such as CPU, memory, storage, and I/O devices). The underlying hardware where the hypervisor runs is usually called the host, and the VMs that run on top of the hypervisor are called guests. Similarly, the operating system that runs on the host is called the host operating system, and the one running in a VM is called the guest operating system.

Based on the presence of the host OS, hypervisors are categorized into two classes, namely Type-1 (also called native or bare-metal) hypervisors and Type-2 (also called hosted) hypervisors. As their names imply, Type-1 hypervisors run directly on the hardware and have their own drivers, whereas Type-2 hypervisors run on top of a host OS and need its facilities to perform their tasks. The most well-known Type-1 hypervisors are:

- VMWare ESX Server [4]
- Microsoft Hyper-V [5]
- Xen [6]
- L4 microkernel family
- CoMik [7]
- XtratuM [8]
- PikeOS [9]

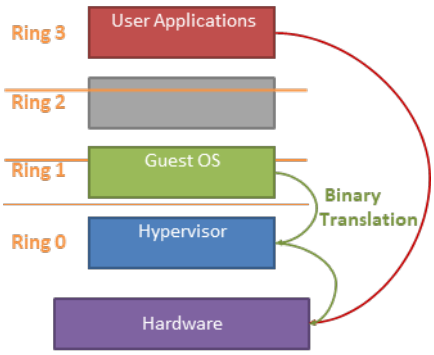
The examples of Type-2 hypervisors include but not limited to:

- Vmware Workstation and Vmware Player [10]
- VirtualBox [11]
- Parallels Desktop for Mac [12]
- QEMU [13]
- KVM [14]

Virtualization using Type-2 hypervisors is more suitable for enabling single users or small organizations to run VMs on a single machine. However, when high performance virtualization strategies are demanded, virtualization using bare-metal hypervisors, which impose less overhead due to direct interaction with the hardware, are more appropriate. Hypervisor-based virtualization approaches can be further classified into four categories which are explained next.

© FitOpTiVis Consortium (Public Document)

Full Virtualization



Full virtualization

In full virtualization, the hypervisor emulates all hardware resources on the virtual system, allowing for running unmodified guest operating systems in VMs. One of the key components that must be emulated in this method is the processor's instruction set architecture. When operating systems run within VMs, they are not privileged enough to execute privileged instructions for interrupt handling, reading and writing to devices, and virtual memory.

For instance, on the x86 architecture, there are four privilege levels (also known as rings) where the components running in level 0 are the most privileged, and the ones executing in level 3 are the least privileged. Usually, in non-virtualized systems, operating systems execute at level 0, and user applications execute at level 3. Unlike the normal instructions (e.g., ADD, SUB, etc.), the privileged instructions (e.g., HLT, invalidate a TLB entry, etc.) can only be executed by the components running in level 0. As shown in figure, in a virtualized environment, guest operating systems execute in level 1, which inhibits them from executing privileged instructions.

Since guest operating systems are unaware that they are running in a virtualized environment, they try to execute the privileged instructions similar to the case where they run in level 0. However, these attempts result in creating traps that go into the hypervisor which then emulates the expected functionality. Therefore, the guest OS never knows that it is running in a VM. Note that the non-privileged instructions execute directly on the hardware without the intermediation of the hypervisor. This technique is called trap and emulate.

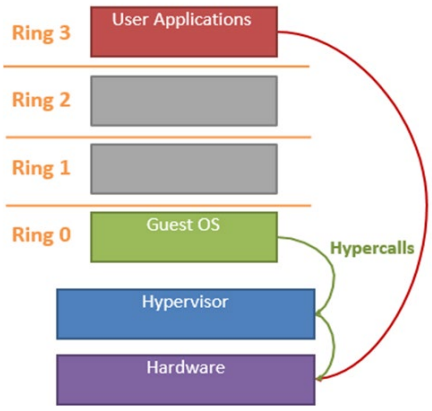
However, there are some thorny issues with this technique. In some architectures, some privileged instructions may fail silently (which are sometimes called virtualization holes). For example, some instructions execute both in the privileged mode and non-privileged mode. However, they produce different results depending on the execution mode. To overcome this issue, a common approach called binary translation is used by the hypervisor. In this approach, the hypervisor scans the unmodified operating system binaries and modifies the offending instruction sequences, making sure that they are dealt with carefully. Since every privileged instruction results in a trap into the hypervisor, the full virtualization method can cause significant performance loss in some workloads. The most well-known products that perform full virtualization are Vmware Workstation, Microsoft Virtual Server, VirtualBox, Parallels Desktop for Mac, and QEMU.

Para-virtualization

Para-virtualization (also known as OS-assisted virtualization) is an alternative approach to perform the virtualization. In this approach, the guest operating system is modified such that it is aware of being running within a VM. That is, as shown in figure, privileged instructions (i.e., non-virtualizable instructions) are replaced by calls to the hypervisor (also known as hypercalls). Therefore, compared to the full virtualization where the communication from the guest operating system to the hypervisor is always implicit via traps, in para-virtualization, the communication is explicit via hypercalls. This can offer

© FitOpTiVis Consortium (Public Document)

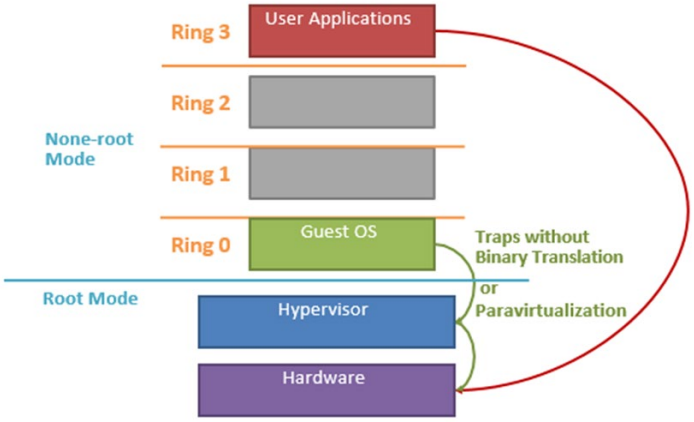
performance improvements compared to the full virtualization for some workloads. However, since the guest operating system needs to be modified, it causes compatibility and portability issues. Note that the para-virtualization does not require any changes in Application Binary Interfaces (ABIs). Hence, the applications running on top of guest operating systems do not need any modifications. The most notable hypervisors performing para-virtualization are Vmware ESX, OKL4, XtratuM, and Xen.



Para-virtualization

Hardware-assisted Virtualization

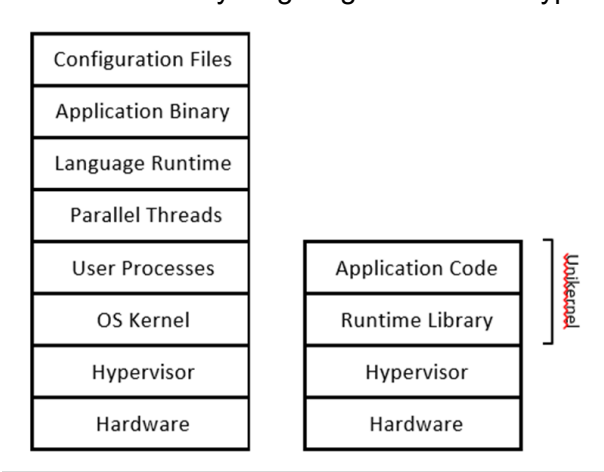
In hardware-assisted virtualization (also known as accelerated virtualization), the underlying hard-ware provides facilities to accelerate the execution of VMs. For instance, as shown in figure, a new CPU privilege mode (called root-mode) has been added to x86 processors since 2006 whereby privileged calls are automatically trapped to the hypervisor without needing to perform binary translation or para-virtualization. These virtualization extensions are introduced in Intel VT-x and AMD-V technologies for Intel and AMD processors respectively. Since the guest operating systems are not modified in hardware-assisted virtualization, it is similar to full virtualization to some extent. However, given the fact that binary translation is not required anymore, hardware-assisted virtualization is considered to be a faster approach. Note that hardware-assisted virtualization is not supported in older systems. The hypervisors that leverage hardware-assisted virtualization include, but are not limited to Vmware ESX, KVM, Hyper-V, and Xen.



Hardware assisted

Unikernels

Unikernel technology emerged in 2013 with the development of MirageOS [15]. The aim was to create specialized, single-purpose VMs whose unnecessary functionalities are removed at compile time, thereby reducing the footprint of an application running in the cloud. Unikernels are based on library operating systems proposed in the past (e.g., Exokernel [16] and Nemesis [17]). However, the hardware compatibility problems faced by these library Oss are solved in unikernels by targeting a standard hypervisor. As shown in figure, during the creation of unikernels, the required system libraries, language runtime, application binary, and configuration files are compiled into a single-address-space VM which runs directly on a standard hypervisor. Accordingly, the scheduling and resource management of unikernels are done by the hypervisor. Note that since there is only one address space, context switches between user and kernel space are not needed anymore, which results in a better performance compared to the traditional VMs. In other words, both the application and kernel components run at the privilege level 0, which is not optimal in terms of security isolation [18]. Although unikernels were first introduced for cloud applications, their lightweight nature has made them a promising solution for upcoming IoT edge applications [19].



Comparison of software layers in traditional VMs and unikernels

The most notable unikernel implementations include:

- MirageOS [15]
- HaLVM [20]
- Osv [21]
- IncludeOS [22]
- ClickOS [23]

A.1.2 Container-based Virtualization

Container-based virtualization (also known as operating system virtualization or containerization) aims at virtualizing the OS kernel rather than the hardware. It is usually considered as a lightweight alternative to hypervisor-based virtualization. The main difference between hypervisor-based virtualization and containerization is that in the former, each VM has its own OS kernel, while in the containerization, all the containers share a single kernel. Hence, containers are more lightweight than VMs. However, hypervisor-based solutions provide more flexibility by enabling the running of multiple operating systems on a single machine. A container image contains an application plus all its dependencies, libraries, and configuration files. A container is a runnable instance of a container image, which essentially is a group of processes that are isolated from

© FitOpTiVis Consortium (Public Document)

other containers or processes in the system. The OS kernel (or container engine in particular) provides this isolation. Being light-weight in nature, containers are becoming the predominant technology in resource-constrained environments such as edge- and fog-based systems [24]. The examples of containerization solutions include, but are not limited to:

- Linux Containers (LXC) [25]
- Ubuntu LXD [26]
- Windows Containers [27]
- Docker [28]
- OpenVZ [29]
- BSD Jails [30]
- Solaris Zones [31]

Since the Linux-based solutions are more common in embedded/IoT architectures, Linux containers have been more focused on. Containers in Linux are realized by leveraging two kernel features, namely control groups and namespaces. Control groups (also called cgroups) is a kernel feature that limits, accounts for and isolates the CPU, memory, disk I/O and network's usage of one or more processes. On the other hand, a cgroup is a set of processes that are bound to a set of limits defined by the cgroup filesystem. Namespaces allow for isolation of global system resources between independent processes, and they provide processes with their own system view. Processes within a namespace only see processes in the same namespace. This type of isolation prevents groups of processes from manipulating other groups. Linux provides several namespaces to isolate system resources such as process identifiers (PIDs), filesystem mount points, and network devices, to name but a few.

A.1.3 Comparison

From the previous discussions on virtualization techniques we can conclude that each approach has its own advantages and disadvantages, which makes it impossible to designate a single approach the perfect solution for virtualization. Accordingly, in this section, we compare the aforementioned techniques from various aspects. Quite a few works exist in the literature that compare virtualization techniques. Hence, to begin with, we review a group of these publications, and subsequently, we summarize the outcomes of these works.

Literature Review

A detailed performance comparison of hypervisor-based virtualization and recently proposed lightweight solutions (including the containers and unikernels) is presented in [32]. Using a number of benchmarking applications, the authors compared four virtualization solutions, namely KVM (as a hypervisor-based approach), LXC and Docker (as containerization approaches), and Osv as a unikernel approach. The considered performance aspects include CPU, Memory, Disk I/O, and Network I/O performance. The measurements show that dominance of a virtualization solution is not necessarily consistent in all the applications. For instance, in two disk performance experiments, LXC performs better than Docker in one experiment, and in the other one, the results are the other way around. However, it can be generally stated that containers outperform VMs in roughly all the experiments. For instance, containers achieve near-native performance for disk intensive benchmarks, while KVM's throughputs for disk write and read are approximately a third and a fifth of the native run, respectively. Since the

© FitOpTiVis Consortium (Public Document)

unikernel approach is not included in all the experiments, we cannot reach any conclusions about its performance compared to others. Nevertheless, they have shown that in memory performance experiments, unikernels perform worse than containers and VMs, and in the network performance experiments, they perform better than VMs but worse than containers.

The work presented in [33] compares four hypervisors (Hyper-V, KVM, vSphere, and Xen) under hardware-assisted virtualization settings in different use cases. The most important outcome of the work is that none of the hypervisors has been found superior to the others. Accordingly, effective management of hypervisor diversity with the goal of matching applications to the best platform is a significant challenge. The authors point out that a cloud environment should support different software and hardware platforms to meet various requirements. The authors have also performed experiments to measure interference caused by multiple tenants, showing that Hyper-V is sensitive to CPU, memory, and network interference. For KVM, although the response times are highly variable, none of the interfering benchmarks considerably degrade the performance. vSphere is highly sensitive to memory interference, while its sensitivity to CPU, disk, and network interference is very low. Finally, Xen's interference sensitivity on memory and network is relatively high compared to the other hypervisors. These results also support the fact that there is no dominant hypervisor with superior performance in all circumstances.

The work presented in [34] evaluates the effects of multi-tenancy on the performance of different virtualization technologies (VMs and containers) in data center environments. The authors compare LXC containers and KVM virtualization and the results show that in general, the interference caused by co-located applications is more severe in the case of containers. In the case of single-tenant scenario, LXC performance is near the performance of bare-metal execution. On the other hand, KVM imposes high performance overhead in case of I/O intensive applications. In case of co-located applications (i.e., multi-tenancy), the results for CPU intensive workloads show that containers are more susceptible to interference. However, in memory-intensive workloads, containers offer acceptable isolation, whereas KVM performs better. In disk I/O isolation experiments, the latency increases by a factor of 8 for LXC, which implies the poor disk isolation in containers. Since the disk I/O performance is not high for VMs even in the isolated cases (and therefore enough bandwidth is available for other VMs), the latency increases only two times for KVM. These measurements demonstrate that isolation is stronger in VMs. Additionally, the authors have studied the impact of virtualization solutions' capabilities on the management and development of applications. In particular, they show how the different characteristics of containers and VMs affect their management in a cluster. From the resource allocation perspective, since VMs somehow share the raw hardware, the resource allocation is also in that granularity (e.g., a fixed number of virtual CPUs). However, in the case of containers, resource control knobs offered by the OS (e.g., CPU scheduling) are more varied, which adds more dimensions to resource allocation. In other words, the resource allocation for containers involves allocation of both physical and OS resources. They also point out that dynamic resource allocation in VMs is fundamentally a hard problem, on the grounds that their virtual hardware is allocated before boot-up, and dynamically change their resource during execution requires "device hotplug" support by the guest OS. However, soft limits in containers provide a dynamic resource allocation mechanism, thereby achieving better performance on overcommitted hosts. Additionally, the authors

© FitOpTiVis Consortium (Public Document)

compared VMs and containers from migration perspective. It is stated that unlike VM migration which is mature and widely used in data centers, container migration is more challenging and not mature yet. Another comparison between VMs and containers made in this work is comparison of their images. It is shown that for the same applications, container images are considerably smaller and faster to construct, which enables faster deployment and lower storage overhead.

Several other works exist in the literature which perform such experiments to compare the virtualization techniques and solutions; [35] compares Xen and KVM; [36] compares KVM and Docker; [37] compares Xen, OpenVZ, and XenServer; [38] performs a comparison between Xen, KVM, VirtualBox, and VMWare ESX; [39] compares software and hardware techniques for x86 virtualization; and [40] presents a comparison between VMs, containers, and unikernels; to name but a few. Additionally, a survey of container-based performance evaluation is conducted in [41]. However, the outcomes of these works are in line with what we discussed above and we therefore do not review them here.

Summary and Conclusions

Based on the technique used to perform virtualization, the virtualized environment is called VM, container, or unikernel. Figure 43 compares the structure and layers of these virtualized entities. Two key points can be inferred from this figure:

© FitOpTiVis Consortium (Public Document)

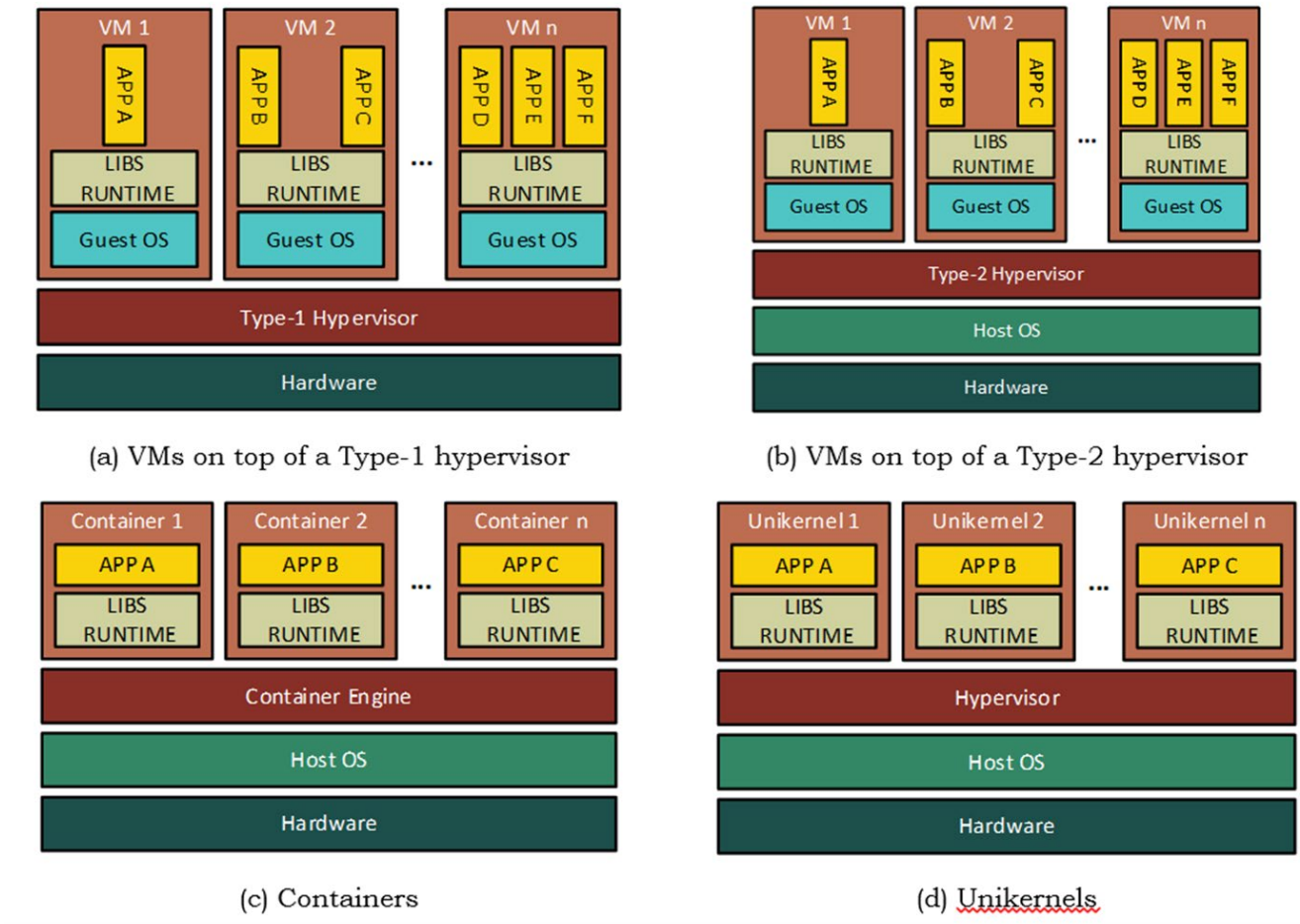


Figure 39. Structural comparison of virtualization solutions.

- Container and unikernels do not have a complete guest OS in their software stack, making them lighter than VMs.
- VMs are used to isolate complete systems – including an OS and a number of applications running on top of it – whereas containers and unikernels are employed to isolate applications.

Furthermore, we can draw an important conclusion from the results of prior works on the comparison of virtualization techniques which is the lack of a predominant virtualization solution performing better than other solutions in every circumstance. Even within a technique (such as hypervisor-based technique), each solution can only outperforms others in a few aspects, but never in all. Accordingly, to demonstrate the trade-offs between virtualization solutions, we summaries the outcomes of the prior works in Table 11.

© FitOpTiVis Consortium (Public Document)

Table 11. Comparison of virtualization solutions characteristics.

Virtualization Technology	Image Size	Boot Time	Memory Usage	Isolation	Flexibility in Resource Management	Performance	Programming Language Restrictions	Live Migration Support
Virtual Machine	~1000 MBs	~3-10 s	~100 MBs	High	Low	The worst	No	Yes
Container	~50 MBs	~<1 s	~5 MBs	Low	High	The best	No	Yes (not mature yet)
Unikernel	~<10 MBs	~<40 ms	~10 MBs	High	Low	Better than VMs, worse than containers	Yes	No

A.2 State-of-the-art in Resource Management

In a computing infrastructure, at any instance of time, resources must be effectively allocated to applications in such a way that their quality requirements are met. The dynamic nature of applications, which implies fluctuations in their resource demands, and the limited amount of available resources, which indicates that resources must be shared among applications, complicate the resource management process.

Although the infrastructure where resource management is performed span cloud infrastructures to stand-alone devices, in this work, we narrow our focus on resource management in fog/edge environments. Hong et al. [24] argue that resource management in fog/edge environments is challenging, since the applications compete for the resources which have limited capacity (e.g., limited power budget) and are heterogeneous (e.g., processors with different architectures), and their workloads change dynamically. Additionally, they argue that the cloud computing model is not practical for using in this paradigm, because it is likely to increase communication latencies when scores of devices are connected to the Internet. Consequently, applications will be adversely impacted because of the increase in communication latencies, and the overall Quality of Service (QoS) and Quality of Experience (QoE) will be degraded. Before getting into further discussions, it is worthwhile to make a distinction between the edge computing and the fog computing paradigms:

- A computing model that makes use of resources located at the edge of the network is referred to as "edge computing " [42]. Note that there is no single accepted definition of "edge" in the literature. There exists a broad definition "anything that's not a traditional data center could be the 'edge' to somebody" [43], which implies that edge of the network is somewhere nearer than data centers to the requestors.
- A model that makes use of both resources located at the edge of the network and the cloud is referred to as "fog computing" [44].

In order to study the literature, we review the following aspects of existing resource management frameworks.

A.2.1 Resource Types and Models

As discussed earlier, in the fog computing model, resources located both at the cloud and the edge of network are used to form a computing environment. These resources can be categorized under four resource types, namely compute resources, networking resources, storage resources, and power resources. In the cloud context, compute

resources are a set of Physical Machines (PMs) that are usually partitioned into several virtual machines using techniques. Each physical machine has one or more CPUs, memories, network interfaces, and I/O devices. However, most of the works only consider processing and memory capacity in their compute resource models [3]. The PMs located at the cloud must be interconnected with a high-bandwidth network. It is shown that the overall performance of cloud services is governed by the communication overhead of PMs [3], which emphasizes the importance of managing the network resources within a cloud infrastructure. Storage services provided by public cloud providers (e.g., Amazon) include various types ranging from virtual disks and database services to object stores [3]. In the cloud infrastructures, the power consuming components are servers, networking equipment, power distribution instruments, cooling appliances, and supporting infrastructure. It is estimated that energy costs account for 42% of the overall operational costs in data centers [47]. Although devising low-power hardware components and efficient application implementations can reduce these costs, power-aware resource management can substantially contribute to total cost reduction as well. A survey of such power-aware resource management techniques for cloud computing systems is presented by Hameed et al. [48].

On the other hand, in the edge computing context, Single Board Computers (SBC) and commodity products comprise the compute resources [24]. SBCs (e.g., Raspberry Pi) are small computers containing processors, memory, network, and storage devices. They have been used in some works as fog/edge nodes [49, 50]. Besides the SBCs, commodity products (e.g., laptops and smartphones) are also employed as fog/edge nodes. Networking resources (i.e., network devices) for fog/edge computing are comprised of gateways and routers, WiFi Access Points (APs), and edge racks [24]. Hong et al. [51] have proposed an approach where under-utilized laptops (resources from public crowds), desktops at the edge of the network, and servers in the cloud are utilized to execute an animation rendering service. They have proposed a prediction method based on machine-learning techniques to predict the completion time of rendering jobs according to available resources. Using a motivational example, they have demonstrated that GFLOPS (Giga Floating Point Operations per Second) is not enough to abstract computation power. Other factors such as number of cores and clock frequency must be included in the model as well. To train their prediction models, they have used datasets where CPU, RAM, disk, and network resources have been considered [52]. The budgets are described in GHz and number of cores for CPUs, GB for memories, read/write throughputs in MB/s for disks, and receive/transmit rates in MB/s for network resources.

Noreikis et al. [53] have proposed a capacity planning solution for hierarchical edge cloud consisting of edge nodes and public clouds that considers QoS requirements in terms of response delay, and diverse demands for CPU, GPU, and network resources. CPU and GPU budgets are described in utilization percentage, and network budgets for transmission and receiving are expressed in KBps, indicating the network speed. Chen et al. [54] have proposed an offloading framework—called HyFog—that accounts for device-to-device and cloud offloading techniques. They have used CPU cycles per unit time to describe compute capacity, and the network links (including cellular links and device-to-device links) are abstracted using download/upload data rates and transmission/receiving power. Wang et al. [55] have proposed the ENORM (Edge NOde Resource Management) framework that realizes fog computing by integrating the edge of the network in the computing environment. They propose a provisioning and

© FitOpTiVis Consortium (Public Document)

deployment mechanism to integrate an edge node with a cloud server. The proposed framework provisions CPU and memory to users. They are described in terms of the number of resource units each of which is one core of CPU and 200MB of RAM.

Liu et al. [56] have proposed an edge computing framework—called ParaDrop—which is implemented on WiFi Access Points or other wireless gateways. In the resource management part of ParaDrop, the controlled resources include CPU (expressed in CPU shares), memory (expressed in maximum allowed memory), and networking (traffic shaping is used to restrict the bandwidth). A fog computing architecture has been proposed by Gu et al. [57] which uses VMs for a medical cyber-physical system (MCPS). The proposed architecture utilizes computational resources in the network edge (e.g., base stations) to store and analyze the health information collected from low power sensors and actuators. Their research investigates the QoS guaranteed minimum cost resource management in fog computing supported MCPS. It is stated that the framework manages the computation capacity of base station resources; however, the resource types and models are not reported. An elastic real-time surveillance system architecture is proposed by Wang et al. [58] where surveillance cameras send images to a distributed edge cloud platform. The proposed system launches Virtualized Network Functions (VNFs) on the edge servers to execute data processing tasks. Resources are provisioned using VMs where resources are described by the number of vCPUs, size of RAM, and size of storage. Morabito et al. [59] have proposed the design of an Edge Computation Platform which leverages container-based virtualization technologies to build an environment for IoT applications. They use single board computers to create smart gateways whose CPUs, GPUs, and storage resources are being managed. There are no discussions on resource models.

An architectural framework—called Foggy—is proposed by Santoro et al. [60] which offers the functionality of negotiation, scheduling, and workload placement considering resource requirements (e.g., CPU, RAM, and disk requirements) and constraints on location and access rights. Foggy is designed to operate in Fog environments with generally more than three tiers, namely Cloud tier (with high resource capacity), Edge Cloudlets tier (with medium resource capacity), Edge Gateways tier (with low resource capacity), and Swarm of Things tier (IoT devices). In Foggy, resource refers to any computational (such as vCPUs, RAM, and disk), storage or network capacity provided by the nodes of the infrastructure. Foggy uses a set of usage profiles for characterizing the resources. For computational and storage resources, the following profiles are used: General purpose (default profile), Compute optimized, Memory optimized, and Storage optimized. For network resources, the considered profiles are Best Effort (default profile), Interactive application, Signaling and video streaming, Interactive and real-time video. Having focused on performance interference, Shekhar et al. [61] have proposed INDICES (INtelligent Deployment for ubiquitous Cloud and Edge Services) framework which performs online performance monitoring, performance prediction, network performance measurements, and server selection and application migration from the cloud to the fog. The architecture model considered in this work contains a Central Data Center (CDC) connected to a set of Micro Data Centers (MDCs) which are located at the edge. Each MDC comprises a set of computer servers which can be allocated to the CDC for its operations at a specified cost. There are no further discussions on types of resources and their models.

© FitOpTiVis Consortium (Public Document)

A.2.2 Resource Estimation Models

In order to meet application quality demands, enough resources must be allocated to applications. Accordingly, the required resources for each application should be estimated beforehand for enabling efficient resource management. This is commonly called as resource demand profiling. In this regard, we study the resource estimation techniques employed in the aforementioned works.

The completion time prediction method proposed by Hong et al. [51] utilizes an animation rendering dataset which contains a huge number of records each of which is a rendering job from an animation studio. Each record is described by the resource usage (including CPU usage in percentage and RAM usage in KBs), the characteristics of rendering jobs (e.g., number of frames, number of polygons, and image size in pixels), the network conditions (e.g., the time of sending a job), and the completion time. The capacity planning solution proposed by Noreikis et al. [53] estimates the minimum capacity required for satisfying QoS demands of real-time applications. Their developed profiler measures resource usage while executing a task on a computing node. Based on the measured usage patterns, resource demands are expressed in terms of CPU and GPU utilization (%), network latency (ms), and network bandwidth (kbps). The task execution model used in the HyFog framework [54] characterizes the resource requirements of a task by the required number of CPU cycles. However, they argue that this model can be easily extended to include other resource types. There are no discussions on how to obtain the required number of CPU cycles for a task.

The ENROM framework [55] initializes the applications using a default amount of CPU and memory. However, while the application is running, the proposed auto-scaler mechanism upscales/downscales the allocated resources dynamically. A number of metrics (e.g., round-trip application latency and hardware utilization of CPU and memory) are monitored to make scaling decisions at the auto-scaler component. Therefore, application resource requirements are not estimated beforehand, and the requirements are expressed in terms of application latency (not resource requirements). The ParaDrop framework [56] runs the requested services in virtualized environments called chutes. Resource requirements for chutes are specified in a config file which is necessary for creating chutes. CPU requirement for a chute is specified by a share value which indicates a relative share of the CPU resource that a chute gets compared to what other chutes get. The maximum amount of memory that a chute is allowed to consume is also specified in the config file. It is stated that a strategy based on shares (similar to CPU shares) is planned to be implemented for specifying the network requirements. It is not discussed in the paper how these requirements are extracted.

The resource management framework proposed by Gu et al. [57] considers the overall expected delay (including communication and processing delays) as application quality requirements. Application resource requirements are expressed by storage requirements and processing speed of applications; however, units and resource estimation methods are not discussed. The three-tier edge computing system architecture proposed by Wang et al. [58] expresses application configurations by templates in the form of a text file describing the resource assignments including IP addresses, bandwidth volumes, compute node flavors, security group and etc. There is no discussion on how to determine the resource requirements. In their experimental results, as discussed before, the VM resources are described by the number of vCPUs, size of RAM, and size of storage. Morabito et al. [59] argue that there may be

© FitOpTiVis Consortium (Public Document)

dissimilarity (e.g., in terms of CPU architecture) in different nodes in a heterogeneous environment. Therefore, for each application, different images, where hardware requirements (processor architecture, GPU, and storage) and software requirements (libraries and operating system) are described, must be provided. It is not explained how these requirements are specified.

Deployment requests in Foggy [60] contain the application component to be deployed and a set of optional deployment requirements which are expressed in terms of resource requirements and/or specific application needs such as location and access rights. Foggy employs a set of profiles to express these requirements. The profiles that are used to express application requirements are similar to the ones used to characterize the resources, explained in Section 3.2.1. There is no discussion on how to automatically determine the deployment requirements. Shekhar et al. [61] argue that the performance of an application depends on several factors including:

- the workload: the workload variation can change the performance,
- the hardware hosting platform: application performance can vary from one hardware platform to another in a heterogeneous environment,
- co-located applications that cause performance interference: hypervisors do not provide enough isolation for two reasons, namely presence of non-partitioned shared resources (e.g., cache spaces) and resource overbooking.

In their proposed framework (INDICES) they run applications in a fixed VM configuration (e.g., 2 GB memory, either 1 or 2 VCPUs). However, according to the reasons mentioned above, this fix configuration may lead to various performance levels. They leverage their built performance models to determine whether running an application on a platform causes SLO (Service-Level Objective) violations or not. Therefore, they only consider performance requirements (not resource requirements).

A.2.3 Resource Provisioning Techniques

There is no concrete definition of resource provisioning in the literature. In some works, it is used to describe the whole resource management process, while in some other works, it refers to the resource allocation procedure. In this section, we want to study how the resources are provisioned (i.e., provided) to applications.

The multimedia fog computing platform proposed by Hong et al. [51] utilizes resources from public crowds (e.g., laptops), desktops at the edge of the network, and servers in the cloud to execute animation renderings. Although the available resource dataset they have used to train their models contains resources in VMs, it is not clarified that how the resources are provisioned to users. Noreikis et al. [53] employ Docker containers to provide virtual resources to users in their capacity planning solution. In the HyFog [54] framework, applications tasks can be executed on either mobile devices or cloud servers. Resources in the former case are provided using VMs; however, resource provisioning in devices is not explained. The ENORM framework [55] leverages edge nodes to host servers offloaded from cloud servers. It is argued that edge nodes have limited hardware resources, which makes the containers more appropriate for providing resources to users. LXC containers are used in this framework. ParaDrop WiFi APs [56] are implemented on SBCs whose resources are provisioned in containers (Docker containers in their current implementation) due to their lightweight nature. Gu et al. employ VMs to provision base station resources. The surveillance system architecture proposed by Wang et al. [58] launches a group of VMs in distributed edge cloud servers

© FitOpTiVis Consortium (Public Document)

to provide resources for surveillance tasks. Lightweight characteristics of container-based virtualization are leveraged by Morabito et al. [59] to provision resources in their proposed IoT gateways.

It can be concluded from the studied works that virtualization plays a significant role in resource provisioning, and the virtualization technique for each solution is decided based on the capabilities of employed platforms.

A.2.4 Resource Allocation Strategies

In this part, we study the policies used to allocate resources to applications and map applications on resources. Hong et al. [51] decide where (i.e., which node) to run rendering jobs based on estimated available resources and predicted completion time of jobs on each node. The completion time is predicted using state-of-the-art machine learning algorithms. The details of employed decision-making policies are not discussed. The solution proposed by Noreikis et al. [53] maps long-running and latency insensitive tasks on the cloud and tasks with the shortest tolerable response delay on edge nodes. Additionally, tasks with complementary resource demands are bundled together and mapped on the same node, leading to better resource utilization. They have used the Knapsack algorithm to perform the optimization. In the HyFog framework [54], task offloading decisions are made using a three-layer graph-matching algorithm. The three-layer graph is constructed by taking the offloading space (mobiles, edge nodes, and the cloud) into account. The problem of minimizing the total task execution cost (including the energy cost per CPU cycle and transmission/receiving power costs) is mapped onto the minimum weight-matching problem over the constructed graph, and it is solved using the Edmonds's Blossom algorithm.

The ENORM framework [55] offers several mechanisms for resource management, including handshaking, deployment, auto-scaling, and termination mechanisms. The handshaking is performed between a cloud manager and edge nodes, and it is used to select a node (based on the available free resources on nodes) for application deployment. The auto-scaling mechanism periodically scales the resources allocated to applications whose latency requirements are not met. The termination mechanism terminates an edge service when either it has been idle for a long period or its QoS requirements cannot be satisfied by an edge server deployment. The ParaDrop framework [56] does not provide any resource allocation policies. Rather, the user selects an edge node (i.e., WiFi AP) to deploy its application. Gu et al. [57] investigate QoS guaranteed minimum cost resource management in fog computing supported MCPS. They formulate the cost minimization problem in a form of mixed-integer nonlinear programming (MINLP), and they linearize it as mixed-integer linear programming (MILP) problem to cope with the high complexity of solving MINLP. Further more, they propose a low-complexity two-phase LP-based heuristic algorithm to solve the MILP problem. In their problem formulation they consider four constraints, namely 1) user association constraints (each user must be associated with a base station, and a subcarrier in the BS must be allocated to the user), 2) task distribution constraints (the application data uploaded to a BS can be distributed to other BSs to get processed), 3) VM placement constraints (VMs must be deployed on BSs, and their resource requirements must not exceed the capacity of BSs), and 4) QoS constraints (the overall expected delay, including communication and processing delay, shall not exceed the application delay constraint). The total cost they seek to minimize includes the total VM deployment cost, uploading cost, and inter-BS communication cost.

© FitOpTiVis Consortium (Public Document)

Wang et al. [58] offer an elastic resource allocation mechanism in their surveillance system where computing resources are reallocated when emergency events happen. To do so, at any point in time, the closest edge nodes to the tracked object are selected to run surveillance tasks. When resource shortage happens, a part of the workload is transferred to another node whose selection depends on its distance to the monitor that captures the object's video. It can be implied that their approach minimizes the deployment costs by minimizing communication latencies. In the Foggy framework [60], to map applications to edge nodes, the orchestrator filters the nodes that can satisfy application requirements. Then, it sorts the filtered nodes according to a priority function whose details are not discussed. Subsequently, the node with the highest rank will be chosen to deploy the application. The objective of the INDICES framework [61] is to assure the SLOs (i.e., response times) for all the applications (by identifying SLO violations and migrating applications from the cloud to edge servers) while minimizing the overall deployment cost. To identify the SLO violations, application execution times are estimated using performance and interference profiles. An interference profile of an application identifies the degree to which that application will degrade the performance of other running applications on the host—called pressure—and how much its own performance will degrade due to interference from other applications—called sensitivity. Accordingly, the framework offers a performance interference-aware server selection algorithm where the SLO-violated applications are migrated to the edge nodes in such a way that the so-called pressure and sensitivity do not cause SLO violations, and furthermore, the overall deployment cost is minimized. The optimization problem is solved using a heuristic-based algorithm since the problem is an NP-Hard one.

A.2.5 Resource Management Architectures

In this section, we study the architecture introduced by the prior works to perform resource management. The framework proposed by Hong et al. [51] has three components, namely an available resource predictor, a completion time predictor, and a job scheduler. The job scheduler decides where to deploy a job based on the information provided by the resource predictor and completion time predictor. The ENORM framework [55] works across three tiers, namely the cloud tier, the edge node tier, and the user device tier. The cloud tier is where the application servers are located, and a cloud server manager runs on each application server. A cloud server manager sends requests to edge nodes, deploys services on edge nodes, and updates the global view of the application server based on the deployments. Each edge node has several components to receive requests from the cloud server manager, negotiate with it, deploy applications upon accepted requests, monitor resources and applications, and perform the auto-scaling mechanism.

The ParaDrop framework [56] has two main resource management agents, namely the ParaDrop backend and the ParaDrop daemon. The ParaDrop backend manages all the resources of the platform in a centralized manner and provides APIs for users to deploy services on the gateways. The ParaDrop daemon runs on each Access Point to perform all the functions required by the ParaDrop platform, including registering the AP to the backend, monitoring the status of AP and reporting to the backend, resource and process management, and receiving RPCs (Remote Procedure Calls) and messages from the backend and performing lifecycle management of chutes (i.e., application containers) accordingly. The architecture proposed by Wang et al. [58] consists of three tiers, namely applications tier, edge computing tier, and data tier. The applications tier

© FitOpTiVis Consortium (Public Document)

contains resource requirements of tasks, plans resource allocations and configurations, and monitors the running status of the applications. The edge computing tier contains an edge control node which performs resource orchestration (to satisfy resource requirements of tasks) and a SDN controller which monitors, configures, and manages VMs. The data tier contains terminal monitors that collect and upload real-time video data to the nearest available compute node.

The architecture proposed by Morabito et al. [59] contains an IoT Application Orchestrator which determines which software (i.e., application) is used for processing the data of a specific device as well as the best location (data center or gateway) for deploying it. It is stated that the orchestrator takes the hardware requirements (processor architecture, GPU, storage) and software requirements (libraries, operating system) of processing software into account during its decision makings. Foggy [60] is an architectural framework which offers the functionality of negotiation, scheduling, and workload placement. The management architecture is composed of an inventory, a negotiator, and an orchestrator. The inventory maintains the status of the infrastructure (i.e., available resources and their location). The negotiator decides whether to accept or reject deployment requests based on the status of the infrastructure. For the accepted deployment requests, the orchestrator deploys application components on the node that best satisfies the deployment requirements.

The architecture model considered in the INDICES framework [61] contains a Central Data Center (CDC) connected to a set of Micro Data Centers (MDCs) which are located at the edge. Each MDC comprises a set of computer servers which can be allocated to the CDC for its operations at a specified cost. A global manager on the CDC is responsible for detecting and mitigating global SLO violations. On each MDC, one of its servers acts as local manager which is responsible for data collection, performance estimation, latency measurements, and MDC-level decision making. During run-time, the global manager identifies the SLO violations, and the local managers decide where to migrate the SLO-violated applications. The works that are not discussed in this section have not made a clear discussion about their architecture.

A.2.6 Summary and Conclusions

The reviewed techniques are summarized in Table 12.

© FitOpTiVis Consortium (Public Document)

Table 12. Summary of reviewed resource management works.

Reference	Managed Resources	Resource Demand Profiling	Resource Provisioning	Resource Allocation	Architecture
Hong et al. [51]	CPU, Memory, Disk, Network	Machine learning algorithms are trained using an animation rendering dataset to predict resource demands.	VMs (not explicitly stated)	VM placement is decided according to estimated available resources and predicted completion time of jobs. Machine learning algorithms are used to predict the completion time.	A hybrid architecture comprised of an available resource predictor, a completion time predictor, and a job scheduler.
Noreikis et al. [53]	CPU, GPU, Network	A profiler extracts resource usage patterns while applications are running. Demands are expressed in CPU/GPU utilization, network latency, and network bandwidth.	Docker containers	Long-running and delay insensitive tasks run in the cloud. Delay sensitive applications run on edge nodes. Applications with complementary requirements run on the same nodes. Knapsack algorithm is used.	Not explained.
Chen et al. [54]	CPU,Network	Resource demands are expressed in the required number of CPU cycles. Profiling approach is not discussed.	VMs in cloud servers; provisioning for edge devices is not discussed.	Task offloading decisions are made using a three-layer graph-matching algorithm. The total task execution cost is minimized. Edmonds's Blossom algorithm is used.	Not explained.
Wang et al. [55]	CPU, Memory	Required resources are not estimated beforehand. Latency requirements are used to decide about resource scalings.	LXC containers	Container placement is decided according to available free resources. Scaling decisions are made based on latency violations. Termination mechanism is proposed as well.	A hybrid architecture comprised of a central cloud manager and distributed managers on edge nodes.
Liu et al. [56]	CPU, Memory, Network	Resource demands are expressed by CPU shares and the maximum memory usage. Profiling approach is not discussed.	Docker containers	Performed by users.	A hybrid architecture consisting of a centralized backed in the cloud and distributed daemons on edge nodes.
Gu et al. [57]	CPU	Resource demands are expressed by storage requirements and required processing speed.	VMs	The total cost (VM deployment cost, uploading cost, and communication cost) is minimized subject to resource and quality constraints. The problem is formulated as a MINLP problem and solved using a LP-based heuristic algorithm.	Not explained.
Wang et al. [58]	CPU, Memory, Disk	Demands are described by templates containing the resource assignments including IP addresses, bandwidth volumes, compute node flavors, security group and etc.	VMs	Reallocation decisions are made with the goal of minimizing communication latencies.	A three-tier architecture where system managers are located at the application tier, and edge node control and SDN controller are located at the edge computing tier.
Morabito et al. [59]	CPU, GPU, Disk	Hardware requirements (processor architecture, GPU, and storage) and software requirements (libraries and operating system) are described in application images. Profiling approach is not discussed.	Containers	Container placement is decided according to required hardware and software resources. Policies are not discussed.	A centralized management architecture containing an IoT orchestrator.
Santoro et al. [60]	CPU, Memory, Disk, Network	Deployment requirements are expressed using a set of profiles in terms of resource requirements and/or specific application needs such as location and access rights. Profiling approach is not discussed.	Docker containers	A priority function is used to sort the edge nodes with sufficient resources. Containers are placed on the nodes with the highest ranks.	The management architecture is composed of an inventory, a negotiator, and an orchestrator.
Shekhar et al. [61]	Not discussed	Only performance requirements are considered.	Containers inside VMs	A performance interference-aware server selection method is used to migrate SLO-violated applications to edge nodes. The optimization goal is to minimize the SLO violations and the overall deployment cost. The problem is solved using a heuristic-based algorithm.	A hybrid architecture composed of a centralized global manager on the central data center and distributed local managers on edge micro data centers.

© FitOpTiVis Consortium (Public Document)

A.3 References

[1] M. Portnoy, Virtualization essentials, vol. 19. John Wiley & Sons, 2012.

[2] J. Sahoo, S. Mohapatra, and R. Lath, “Virtualization: A survey on concepts, taxonomy and associated security issues,” in Proc. 2nd Intl. Conf. on Computer and Network Technology, pp. 222–226, IEEE, 2010.

[3] B. Jennings and R. Stadler, “Resource management in clouds: Survey and research challenges,” Journal of Network and Systems Management, vol. 23, pp. 567–619, Jul 2015.

[4] C. A. Waldspurger, “Memory resource management in VMware ESX server,” ACM SIGOPS Operating Systems Review, vol. 36, no. SI, pp. 181–194, 2002.

[5] A. Velte and T. Velte, “Microsoft virtualization with Hyper-V,” McGraw-Hill, Inc., 2009.

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, “Xen and the art of virtualization,” in ACM SIGOPS operating systems review, vol. 37, pp. 164–177, ACM, 2003.

[7] A. Nelson, A. B. Nejad, A. Molnos, M. Koedam, and K. Goossens, “Comik: A predictable and cycle-accurately composable real-time microkernel,” in Proceedings of the conference on Design, Automation & Test in Europe, p. 222, European Design and Automation Association, 2014.

[8] M. Masmano, I. Ripoll, A. Crespo, and J. Metge, “Xtratum: a hypervisor for safety critical embedded systems,” in Proc. 11th Real-Time Linux Workshop, pp. 263–272, Citeseer, 2009.

[9] R. Kaiser and S. Wagner, “The PikeOS concept: History and design,” SysGO AG White Paper. Available: http://www.sysgo.com, 2007.

[10] D. Marshall, “Understanding full virtualization, para-virtualization, and hardware assist,” VMware White Paper, p. 17, 2007.

[11] “Virtualbox.” https://www.virtualbox.org. Accessed: 2019-02-03.

[12] “Parallels desktop for Mac.” https://www.parallels.com/products/desktop. Accessed: 2019-02-03.

[13] F. Bellard, “Qemu: a fast and portable dynamic translator,” in USENIX Annual Technical Conference, FREENIX Track, vol. 41, p. 46, 2005.

[14] I. Habib, “Virtualization with KVM,” Linux Journal, vol. 2008, no. 166, p. 8, 2008.

[15] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, “Unikernels: Library operating systems for the cloud,” in Proc. 18th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS ’13), pp. 461–472, ACM, 2013.

[16] D. R. Engler, M. F. Kaashoek, et al., “Exokernel: An operating system architecture for application-level resource management,” vol. 29. ACM, 1995.

[17] I. M. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden, “The design and implementation of an operating system to support

© FitOpTiVis Consortium (Public Document)

distributed multimedia applications," IEEE Journal on Selected areas in communications, vol. 14, no. 7, pp. 1280–1297, 1996.

[18] M. J. De Lucia, "A survey on security isolation of virtualization, containers, and unikernels," tech. rep., US Army Research Laboratory Aberdeen Proving Ground United States, 2017.

[19] R. Morabito, V. Cozzolino, A. Y. Ding, N. Beijar, and J. Ott, "Consolidate iot edge computing with lightweight virtualization," IEEE Network, vol. 32, pp. 102–111, Jan 2018.

[20] "Haskell lightweight virtual machine (halvm)." https://galois.com/project/halvm. Accessed: 2019-02-03.

[21] A. Kivity, D. Laor, G. Costa, P. Enberg, N. HarEl, D. Marti, and V. Zolotarov, "Optimizing the operating system for virtual machines," in USENIX Annual Technical Conference (USENIX ATC '14), pp. 61–72, 2014.

[22] A. Bratterud, A.-A. Walla, H. Haugerud, P. E. Engelstad, and K. Begnum, "Includeos: A minimal, resource efficient unikernel for cloud services," in 7th Intl. Conf. on Cloud Computing Technology and Science (CloudCom), pp. 250–257, IEEE, 2015.

[23] J. Martins, M. Ahmed, C. Raiciu, and F. Huici, "Enabling fast, dynamic network processing with ClickOS," in Proc. 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, pp. 67–72, ACM, 2013.

[24] C. Hong and B. Varghese, "Resource management in fog/edge computing: A survey," CoRR, vol. abs/1810.00305, 2018.

[25] M. Helsley, "LXC: Linux container tools," IBM developerWorks Technical Library, vol. 11, 2009.

[26] "Lxd." https://linuxcontainers.org/lxd/introduction. Accessed: 2019-02-03.

[27] "Containers on windows." https://docs.microsoft.com/en-us/virtualization/windowscontainers/ab. Accessed: 2019-02-03.

[28] D. Merkel, "Docker: Lightweight Linux containers for consistent development and deployment," Linux J., vol. 2014, Mar. 2014.

[29] "Openvz: Open source container-based virtualization for Linux." https://openvz.org. Accessed: 2019-02-03.

[30] P.-H. Kamp and R. N. Watson, "Jails: Confining the omnipotent root," in Proc. 2nd Intl. SANE Conference, vol. 43, p. 116, 2000.

[31] D. Price and A. Tucker, "Solaris zones: Operating system support for consolidating commercial workloads," in LISA, vol. 4, pp. 241–254, 2004.

[32] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. lightweight virtualization: a performance comparison," in Proc. IEEE Intl. Conf. on Cloud Engineering, pp. 386–393, IEEE, 2015.

[33] J. Hwang, S. Zeng, F. y Wu, and T. Wood, "A component-based performance comparison of four hypervisors," in Proc. IFIP/IEEE Intl. Symp. on Integrated Network Management (IM 2013), pp. 269–276, IEEE, 2013.

© FitOpTiVis Consortium (Public Document)

[34] P. Sharma, L. Chaufournier, P. Shenoy, and Y. C. Tay, “Containers and virtual machines at scale: A comparative study,” in Proc. 17th Intl. Middleware Conference (Middleware ’16), pp. 1:1–1:13, ACM, 2016.

[35] A. Binu and G. S. Kumar, “Virtualization techniques: a methodical review of Xen and KVM,” in Proc. Intl. Conf. on Advances in Computing and Communications, pp. 399–410, Springer, 2011.

[36] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, “An updated performance comparison of virtual machines and linux containers,” in Proc. IEEE Intl. Symp. on Performance Analysis of Systems and Software (ISPASS), pp. 171–172, IEEE, 2015.

[37] A. Babu, M. Hareesh, J. P. Martin, S. Cherian, and Y. Sastri, “System performance evaluation of para virtualization, container virtualization, and full virtualization using Xen, OpenVZ, and Xenserver,” in Proc. 4th Intl. Conf. on Advances in Computing and Communications, pp. 247–250, IEEE, 2014.

[38] A. J. Younge, R. Henschel, J. T. Brown, G. Von Laszewski, J. Qiu, and G. C. Fox, “Analysis of virtualization technologies for high performance computing environments,” in Proc. 4th IEEE Intl. Conf. on Cloud Computing, pp. 9–16, IEEE, 2011.

[39] K. Adams and O. Agesen, “A comparison of software and hardware techniques for x86 virtualization,” ACM SIGOPS Operating Systems Review, vol. 40, no. 5, pp. 2–13, 2006.

[40] F. Huici, F. Manco, J. Mendes, and S. Kuenzer, “VMs, unikernels and containers: Experiences on the performance of virtualization technologies,”

[41] N. G. Bachiega, P. S. Souza, S. M. Bruschi, and S. d. R. de Souza, “Container-based performance evaluation: A survey and challenges,” in Proc. IEEE Intl. Conf. on on Cloud Engineering (IC2E), pp. 398–403, IEEE, 2018.

[42] M. Satyanarayanan, “The emergence of edge computing,” Computer, vol. 50, pp. 30–39, Jan 2017.

[43] “What is edge?” https://www.etsi.org/newsroom/blogs/entry/what-is-edge. Accessed: 2019-02-03.

[44] A. V. Dastjerdi and R. Buyya, “Fog computing: Helping the internet of things realize its potential,” Computer, vol. 49, pp. 112–116, Aug 2016.

[45] S. S. Manvi and G. K. Shyam, “Resource management for infrastructure as a service (IaaS) in cloud computing: A survey,” Journal of Network and Computer Applications, vol. 41, pp. 424– 440, 2014.

[46] P.-O. Östberg, J. Byrne, P. Casari, P. Eardley, A. F. Anta, J. Forsman, J. Kennedy, T. Le Duc, M. N. Marino, R. Loomba, et al., “Reliable capacity provisioning for distributed cloud/edge/fog computing applications,” in Proc. European Conf. on Networks and Communications (EuCNC), pp. 1–6, IEEE, 2017.

[47] J. Hamilton, “Cooperative expendable micro-slice servers (cems): low cost, low power servers for internet-scale services,” in Proc. Conf. on Innovative Data Systems Research (CIDR09), Citeseer, 2009.

[48] A. Hameed, A. Khoshkbarforoushha, R. Ranjan, P. P. Jayaraman, J. Kolodziej, P. Balaji, S. Zeadally, Q. M. Malluhi, N. Tziritas, A. Vishnu, et al., “A survey and taxonomy

© FitOpTiVis Consortium (Public Document)

on energy efficient resource allocation techniques for cloud computing systems," Computing, vol. 98, no. 7, pp. 751–774, 2016.

[49] S. J. Johnston, P. J. Basford, C. S. Perkins, H. Herry, F. P. Tso, D. Pezaros, R. D. Mullins, E. Yoneki, S. J. Cox, and J. Singer, "Commodity single board computer clusters and their applications," Future Generation Computer Systems, vol. 89, pp. 201–212, 2018.

[50] P. Bellavista and A. Zanni, "Feasibility of fog computing deployment based on docker containerization over RaspberryPi," in Proc. 18th Intl. Conf. on Distributed Computing and Networking, p. 16, ACM, 2017.

[51] H.-J. Hong, J.-C. Chuang, and C.-H. Hsu, "Animation rendering on multimedia fog computing platforms," in IEEE Intl. Conf. on Cloud Computing Technology and Science (CloudCom), pp. 336–343, IEEE, 2016.

[52] S. Shen, V. v. Beek, and A. Iosup, "Statistical characterization of business-critical workloads hosted in cloud data centers," in Proc. 15th IEEE/ACM Intl. Symp. on Cluster, Cloud and Grid Computing, pp. 465–474, May 2015.

[53] M. Noreikis, Y. Xiao, and A. Yl-Jaiski, "Qos-oriented capacity planning for edge computing," in Proc. IEEE Intl. Conf. on Communications (ICC), pp. 1–6, May 2017.

[54] X. Chen and J. Zhang, "When d2d meets cloud: Hybrid mobile task offloadings in fog computing," in Proc. IEEE Intl. Conf. on Communications (ICC), pp. 1–6, May 2017.

[55] N. Wang, B. Varghese, M. Matthaiou, and D. S. Nikolopoulos, "Enorm: A framework for edge node resource management," IEEE Transactions on Service Computing, pp. 1–1, 2018.

[56] P. Liu, D. Willis, and S. Banerjee, "Paradrop: Enabling lightweight multi-tenancy at the networks extreme edge," in Proc. IEEE/ACM Symp. on Edge Computing (SEC), pp. 1–13, Oct 2016.

[57] L. Gu, D. Zeng, S. Guo, A. Barnawi, and Y. Xiang, "Cost efficient resource management in fog computing supported medical cyber-physical system," IEEE Transactions on Emerging Topics in Computing, vol. 5, pp. 108–119, Jan 2017.

[58] J. Wang, J. Pan, and F. Esposito, "Elastic urban video surveillance system using edge computing," in Proc. Workshop on Smart Internet of Things (SmartIoT '17), pp. 7:1–7:6, ACM, 2017.

[59] R. Morabito and N. Beijar, "Enabling data processing at the network edge through lightweight virtualization technologies," in Proc. Intl. Conf. on Sensing, Communication and Networking (SECON Workshops), pp. 1–6, June 2016.

[60] D. Santoro, D. Zozin, D. Pizzolli, F. D. Pellegrini, and S. Cretti, "Foggy: A platform for workload orchestration in a fog computing environment," in IEEE Intl. Conf. on Cloud Computing Technology and Science (CloudCom), pp. 231–234, Dec 2017.

[61] S. Shekhar, A. D. Chhokra, A. Bhattacharjee, G. Aupy, and A. Gokhale, "Indices: exploiting edge resources for performance-aware cloud-hosted services," in Proc. 1st IEEE Intl. Conf. on Fog and Edge Computing (ICFEC), pp. 75–80, IEEE, 2017.

[61] Kees Goossens, Martijn Koedam, Andrew Nelson, Shubhendu Sinha, Sven Goossens, Yonghui Li, Gabriela Breaban, van Kampenhout, Reinier, Rasool Tavakoli,

© FitOpTiVis Consortium (Public Document)

Juan Valencia, Ahmadi Balef, Hadi, Benny Akesson, Sander Stuijk, Marc Geilen, Dip Goswami, and Majid Nabi. "NOC-Based Multi-Processor Architecture for Mixed Time-Criticality Applications", In Handbook of Hardware/Software Codesign, Soonhoi Ha and Jurgen Teich (editors), Springer, 2017.

© FitOpTiVis Consortium (Public Document)