D2.8 – Evaluation of the CLASS Software Architecture

Version 1.0

Document Information

Contract Number	780622
Project Website	https://class-project.eu/
Contractual Deadline	M42, 30th June 2021
Dissemination Level	PU
Nature	Report
Author(s)	Elli Kartsakli (BSC)
Contributor(s)	Eudald Sabaté (BSC)
Reviewer(s)	
Keywords	software architecture, CLASS scheduling strategy, edge cloud coordination

Notices: *The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No "780622".*

Change Log

Version	Author	Description of Change
V0.1	Elli Kartsakli (BSC)	Initial Draft
V0.2	Eudald Sabaté (BSC)	Introduction of results
V1.0	BSC	Final version, ready to EC review

Table of contents

Executive Summary

This deliverable presents the evaluation of the CLASS computation distribution layer, responsible for the efficient distribution of the data analytics tasks across the compute continuum and their execution providing the real-time guarantees required by the project's use cases. The key component of the computation distribution layer in the CLASS Software Architecture (SA) is the COMP Superscalar (COMPSs) framework, integrated with dataClay for the management and storage of data. To meet the real-time requirements, the baseline COMPSs scheduler has been enhanced with the implementation of four scheduling policies based on heuristics, aiming to minimize the end-to-end response time of the analytics workflow.

The performance evaluation presented in this deliverable is focused on demonstrating the performance improvement gained by the introduced scheduling heuristics, thus demonstrating the capability of the distribution layer to comply with the use case real-time requirements.

1 Introduction

The aim of WP2 for milestone MS4 has been the final release of the CLASS Software Architecture (SA), implementing all the necessary techniques to ensure the efficient distribution of the data analytics workflows across the compute continuum.

The final release of the CLASS SA has been reported in detail in D2.7 [1], whereas the performance of the different SA components has been individually evaluated in the respective final deliverables of the technical Work Packages (WPs): the data analytics platform (reported in D5.5 [2]) the cloud analytics platform (D4.6 [3]) and the edge analytics platform (D3.6 [4]). Furthermore, the end-to-end performance evaluation of the SA validating the CLASS use cases over the compute continuum infrastructure deployed in the Modena Automotive Smart Area (MASA), at the city of Modena, Italy, has bene detailed in D1.6 [5].

To complete the overall picture of the CLASS SA capabilities, this deliverable evaluates the computation distribution layer, consisting of the COMP Superscalar (COMPSs) framework, for the deployment, distribution and execution of complex analytics workflows, and dataClay, a distributed data store managing the availability of data across the compute continuum. As explained in D2.4 [6], COMPSs has been initially designed for High Performance Computing (HPC) environments, and the schedulers implemented in the COMPSs framework did not take into account real-time constraints. To that end, in the context of CLASS, four scheduling policies based on heuristics have been developed, aiming to minimize the end-to-end response time of the analytics workflow. These policies have been described in detail in D2.6 [7], and some preliminary performance results have been presented, using four well-known HPC applications, as well as a simplified version of the CLASS analytics workflow.

This deliverable will present the performance evaluation of the COMPSs analytics workflow implementing the CLASS data analytics methods for the obstacle detection and tracking analytics at the edge and the aggregation of all extracted information into the Data Knowledge Base (DKB) at the cloud. The evaluation is focused on demonstrating the performance improvement gained by the introduced scheduling heuristics, thus demonstrating the capability of the distribution layer to meet the use case real-time requirements. The evaluation has been performed considering the final release of the CLASS SA (as reported in D2.7 [1]) deployed in the MASA, considering the final version of the data analytics employed for the validation and demonstration of the project's use cases, as summarized in D1.6 [5].

The remaining of this deliverable is organised as follows. Section 2 provides a summary of the heuristic scheduling policies, first introduced in D2.6 [7]. Section 3 provides an overview of the COMPSs workflow for the CLASS data analytics, which was used for the evaluation. Finally, in Section 4, the performance evaluation results are presented and discussed.

2 The COMPSs runtime scheduler

This section will provide a high-level overview of the system model and the proposed scheduling heuristics developed in CLASS, highlighting the key concepts needed for the interpretation of the performance evaluation results. A complete description and mathematical representation of these concepts can be found in D2.6 [7].

2.1 The system model for the analysis of distributed time-sensitive workflows

COMPSs[1] is a task-based programming model and runtime framework for the development of parallel applications and their execution over distributed infrastructures [8]. The COMPSs programming model enables the developer to transform sequential code into COMPSs tasks, by simply annotating the methods that can be distributed and marking their dependencies and their directionality (i.e., IN, OUT or INOUT).

The COMPSs runtime represents a COMPSs workflow as a ***Task Dependency Graph (TDG)*** **or** ***Directed Acyclic Graph (DAG)***: each node corresponds to a COMPSs task, while the edges represent the data dependency between two tasks. Then, honoring the task dependencies and based on the implemented scheduling policy, the COMPSs runtime is responsible of distributing the tasks of the COMPSs workflow over the available computing infrastructure.

However, even though the DAG representation fully describes the task-based analytics workflow, it cannot describe heterogeneous environments in which the compute continuum is formed by resources distributed from edge to cloud, as is the case in CLASS. To tackle with this limitation, CLASS has proposed a novel system model that, in addition to the DAG, also includes a ***Digraph (directed graph) compute continuum model*** that characterizes the heterogeneous computing resources and the respective communication links.

The mathematical representation and complete description of the DAG and Digraph compute continuum system model adopted in class have been presented in detail in D2.4 [6] and D2.6 [6]. Concerning the practical implementation, the DAG is generated by COMPSs based on the definition and dependencies of the data analytics methods using the COMPSs programming model (see also Section 3.2.1 in D2.7 [1]). On the other hand, to build the Digraph compute continuum model, an initial profiling phase must take place in order to obtain the characterization of the computing and communication capabilities of the specific infrastructure on which the workflow will be executed. The profiling will generate the following information, which will be used by the COMPSs scheduler:

1) The average execution time of each task $C_{i,k}$ on each available computing resource v_i (compute node).

[1] https://www.bsc.es/research-and-development/software-and-apps/software-list/comp-superscalar

2) The average data transfer time $T_{j,i}^{transf}$, corresponding to the time needed for the transmission of a data payload between two dependent tasks v_i and v_j (e.g., the output of task v_i needed as input to v_j), which also depends on which nodes the tasks are executed on.

In the classical DAG-based system model targeting shared memory architectures, the volume of a DAG task, denoted as G, is defined as the sum of the worst-case execution time (WCET) C_i, of all the nodes of G. This value corresponds to the worst case response time of the DAG task on a dedicated single-core platform [8]. However, in the CLASS system model, this definition is not valid anymore since the response time of an application highly depends on the scheduling decisions, given that the underlying computing and communication infrastructure is heterogeneous.

Since there two factors affecting the response time of an application, i.e., the actual execution time on the computing nodes and the data transfer times, two different volumes are defined, considering a known (or static) allocation of tasks on resources:

1) the *computation volume*, defined as the sum of execution times of all tasks on the allocated computing resources, and
2) the *communication volume*, defined as the sum of all necessary data transfer times over the available communication links.

The sum of these two volumes constitutes the *workload* W, corresponding to the ***worst-case execution time upper-bound*** R^{ub} of the DAG executed on the Digraph compute continuum, for the specific allocation policy.

2.2 The CLASS scheduling heuristic policies

Based on the aforementioned system model, four heuristics have been proposed, aiming to minimize the end-to-end response time of the analytics workflow. The four proposed schemes take into account two sets of priority rules, and are briefly summarized next:

1) *Heuristics based on successors*, in which the next ready task[2] is prioritized based on two different criteria:
   - ***Largest Number of Successors in Next Level (LNSNL)***. This heuristic selects the task v_i with the largest number of *direct* successors, with the objective of increasing the number of nodes that become ready when v_i completes.
   - ***Largest Number of Successors (LNS)***. This heuristic selects the task v_i with the largest number of successors, with the objective of prioritizing the execution of those portions of the DAG with the highest number of nodes, and so potentially, the largest impact on the execution time of the application.
2) Heuristics based on the processing time, in which the minimum completion time of all ready tasks is first computed, and among the ready tasks two allocation policies are defined:

[2] A task is ready if all its direct predecessor tasks (nodes in the DAG) have been completed.

- ***Shortest Processing Time (SPT)***. This heuristic selects the task v_i with the shortest completion time, i.e., prioritizing the smallest tasks (in terms of execution time) in the fastest computing resources.
- ***Longest Processing Time (LPT)***: This heuristic selects the task v_i with the longest completion time, with the objective of prioritizing the biggest nodes in the fastest computing resources.

The pseudocode for the implemented heuristics and additional explanations can be found in Section 3.2 of D2.6 [7].

3 Overview of the COMPs workflow implementing the CLASS data analytics methods

This section will provide an overview of the COMPSs workflow that implements the data analytics executed at the edge and aggregation of data at the DKB in the cloud. For the performance evaluation, we will focus on the main part of the COMPSs workflow that is common for both CLASS use cases, namely the collision detection and the air pollution[3]. This common workflow includes the data analytics methods depicted in Figure 1, and is a subset of the overall workflow reported in D2.7 [1]. More details on the data analytics and implemented methods can be found in D2.7 [1] and D1.6 [5].

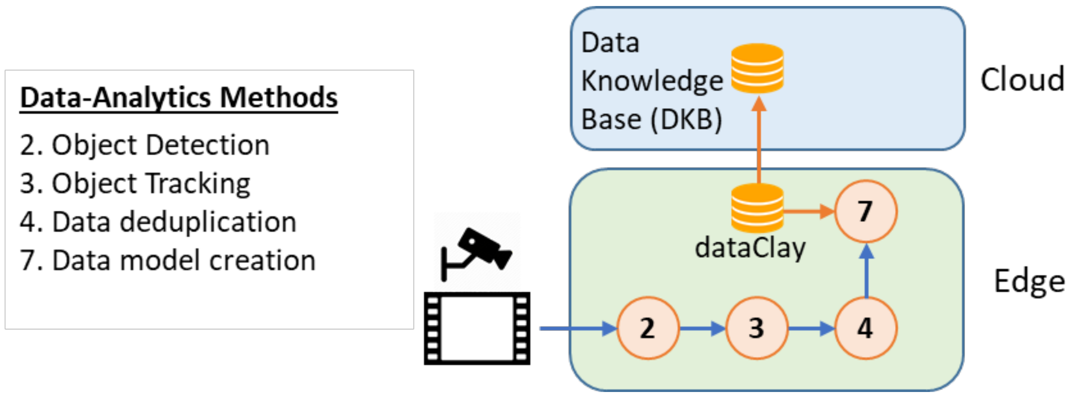

Figure 1. COMPSs task-based data analytics methods of the CLASS use cases

The workflow consists of a COMPSs application that connects to the *"object detection"* data analytics method and invokes the *"object tracking"*, which identifies and tracks objects. The detected objects from all sources go into the *"data deduplication"* method, and the output is stored in dataClay (persisting the information at the fog level) and is federated to the cloud, stored in the Data Knowledge Base (DKB).

The implementation of the COMPS analytics workflow has been described in D2.7 [1]. For convenience and to enhance readability, this information will be also included in this section.

[3] This workflow corresponds to the full COMPSs analytics methods employed for the collision use case. The air pollution use case calls one additional method that invokes the computation of the air pollution estimation, based on the data of the DKB.

Figure 2 shows the pseudo-code of the common part of the COMPSs-based Python application that implements the data analytics executed at the edge for both CLASS use cases.

```
@task(returns=list)
def get_detected_objects (camera_socket):
   return tkDNN_detected_objects(camera_id)

@task(object_list=IN, tracked_objects=IN, returns=list)
def tracker(object_list, tracked_objects):
   return track(object_list, tracked_objects)

@task(object_list=COLLECTION_IN, returns=list)
def deduplicator(tracked_objects):
   return deduplicated_obj(tracked_objects)

@task(deduplicated_objects=IN, dC_model = IN)
def create_data_model(deduplicated_obj):
   snapshot = dC_model.Create_snapshot(deduplicated_obj)
   create_air_pollution_datafile(deduplicated_obj)
   return snapshot

@task(snapshot=IN, backend_to_federate=IN)
def federate_info(snapshot, backend_to_federate):
   snapshot.federate_to_backend(backend_to_federate)

## Main function ##
while True:
     for i, socket in camera_sockets
          obj_list = get_detected_objects (socket)
          tracked_obj[i] = tracker(obj_list, tracked_obj[i])
     deduplicated_obj = deduplicator(tracked_obj)
     snapshot = create_data_model(deduplicated_obj)
     federate_info(snapshot, external_backend_id)
```

Figure 2. The COMPSs workflow for the CLASS use cases

The application iteratively executes the following functionalities encapsulated in COMPSs tasks:

- `get_detected_objects`: it connects via UDP socket, to the edge device where the "*object detection*" data analytics method (tkDNN) is being executed, namely, the edge node where the input videos from smart cameras are processed. The list of detected objects is received.
- `tracker`: it executes the "*object tracking*" data analytics method. It is implemented in C++, thus a Python binding has been implemented. Each data source (i.e., camera) is connected to a different tracker tasks, enabling the parallelization of the tracking process among multiple cameras.
- `deduplicator`: it deduplicates the objects detected by multiple sources (e.g., different cameras with partial overlapping of the covered area), returning only one copy per object.

- `create_data_model`: it executes a dataClay method to store newly detected objects or update existing objects with new events (i.e., detected positions and other relevant information), creating snapshots.
- `federate_info`: it federates the snapshots created at the fog level to the dataClay backend at the cloud (updating the DKB).

Figure 3 shows the Direct Acyclic Graph (DAG) representation of the COMPSs tasks for two iterations of the workflow considering a single video source.

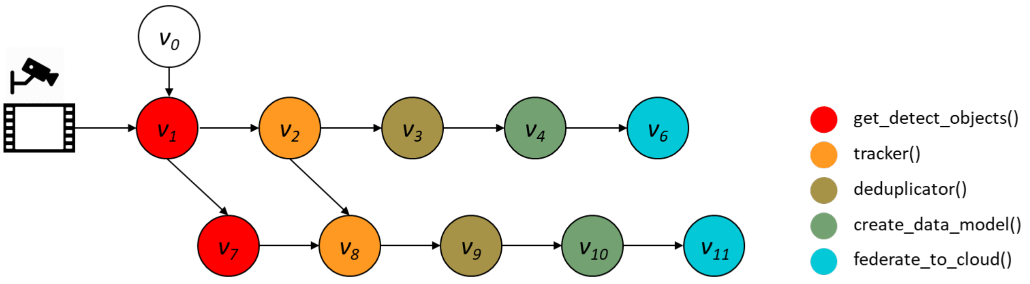

Figure 3. COMPSs DAG for two iterations of the workflow for a single camera source

In reality, the complexity of the DAG is much higher, since typically a higher number of iterations is considered for each scheduling interval (resulting to a high number of generated tasks), and multiple video sources are employed. As a reference, Figure 4 shows the DAG corresponding to 1 second of processing for a single camera.

The generated tasks of the DAG are then distributed and scheduled based on the implemented heuristics that aim to minimize the end-to-end execution time.

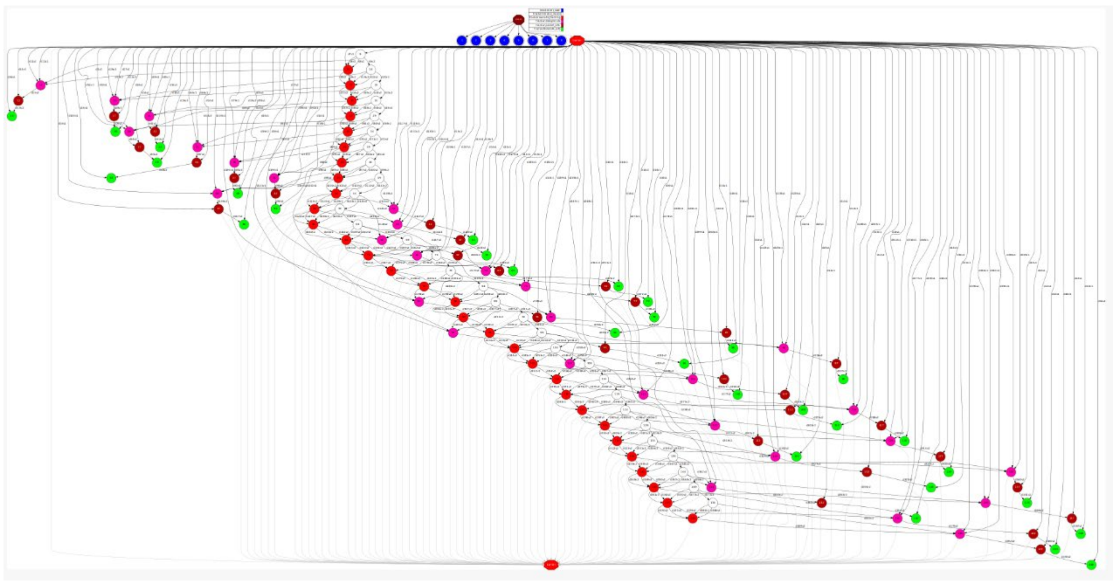

Figure 4. COMPSs DAG for 1s of processing from a single video source

4 Performance Evaluation

4.1 The compute continuum infrastructure

Figure 5 presents the compute continuum infrastructure at the MASA (fog nodes 1-4) and Modena data center (cloud), on which the COMPSs analytics workflow has been executed. The figures shows an example of the distribution of the different analytics methods on the different computing resources. The dataClay dependent tasks (i.e., the data model creation and federation) are constrained to be executed at Fog Node 4. However, the object tracking tasks can be distributed anywhere across the compute continuum, based on the scheduling policy.

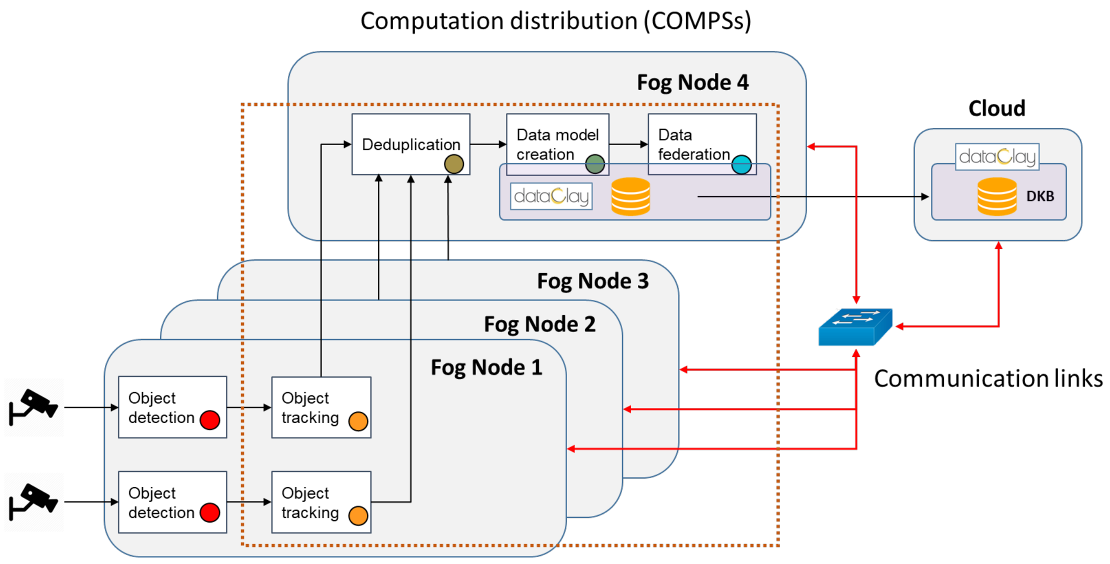

Figure 5. Description of the execution of the collision detection and air pollution estimation applications over the CLASS software architecture

4.2 Setup phase

For the implementation and evaluation of the proposed heuristic scheduling policies, the DAG and the Digraph (directed graph) compute continuum model must be obtained.

The structure of the DAG is already provided by COMPSs which dynamically builds it at runtime. To retrieve this information, the tag –g (-graph) is used when executing COMPSs (with the runcompss command). This tag generates a dot file[4], as the one of Figure 4, containing all the nodes composing the application, interconnected by edges that represent the dependencies among them. By parsing this file, all the dependencies among nodes and the identifiers of these dependencies are obtained, enabling us to extract all the necessary information for the implementation of the heuristics (for more information, see D2.7).

[4] https://graphviz.org/doc/info/lang.html

The upper bound execution time $C_{i,k}$ (see Section 2.1) is gathered by executing all tasks composing the application in the different computing resources while using all the logging and tracing functions provided by COMPSs.

As a second step, a full profiling of the compute continuum has taken place, in order to characterize the quality of the communication link. To that end, the iPerf[5] tool has been employed to retrieve measurements of the maximum achievable bandwidth between all different pairs of computing resources. These bandwidth values are used to determine the data transfer times, i.e., the time needed to transfer the dependent data between a pair of computing nodes, given their known payload and the additional overhead due to the communication protocol (i.e, Ethernet IEEE 802.3 for the wired connections among the fog and cloud resources in Modena).

4.3 Results

The performance evaluation of the proposed heuristic scheduling policies has been conducted considering two different scenarios. In the first scenario, computation is distributed among the four available computing nodes, as explained in the previous section (Figure 5). In the second scenario, the parallelization capabilities within the computing nodes are further exploited, with tasks being scheduled for concurrent execution across the four computing units of each fog node.

For both scenarios, the performance has been evaluated for one and three video sources. For all experiments, a fixed number of 400 workflow iterations has been selected, with each iteration corresponding to the complete processing of a single frame per video source, from the retrieval of the frame until the federation of the extracted information (i.e., detected, tracked and deduplicated objects with their associated position information) to the DKB in the cloud. Each experiment has been repeated 10 times, and the average execution time per iteration has been measured for each execution.

4.3.1 Scenario 1: Distributed processing across the four fog nodes

The four proposed heuristics have been compared with the First-In-First-Out (FIFO) policy implemented by the default COMPSs scheduler. Figure 6 depicts the average execution time (in seconds) per iteration of the workflow, for the five different policies. The boxplot representation has been selected, showing the five-number summary of the data, i.e., the minimum, first quartile, median, third quartile, and maximum. It can be seen that all heuristics outperform the FIFO policy, achieving lower execution times and much more stability, with all results being very close to the median value.

Similar results are obtained when the number of video sources is increased, as shown in Figure 7. A slight increase in the iteration times with respect to the first scenario (in the order of 20-30 ms) is observed, due to the increased number of detected and

[5] https://iperf.fr/

processed objects (i.e., detected, tracked and deduplicated) coming from 3 frames being processed simultaneously.

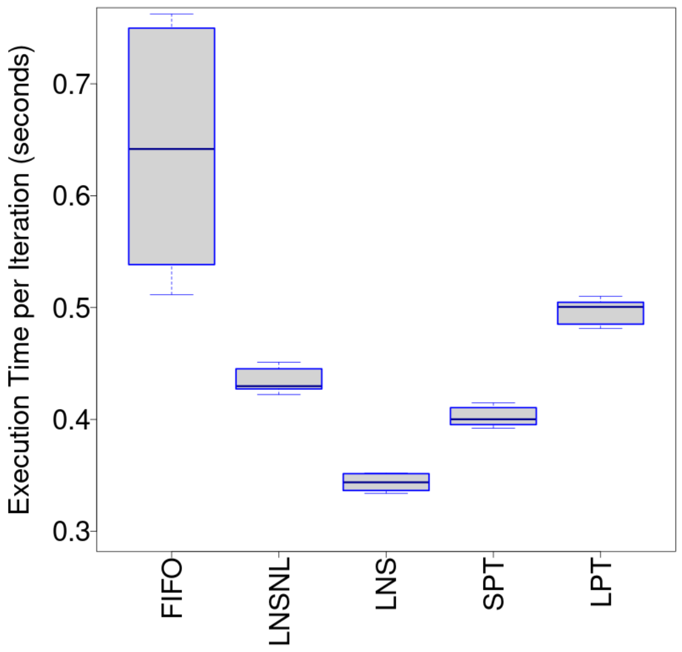

Figure 6. Execution time per iteration, with input from 1 video source

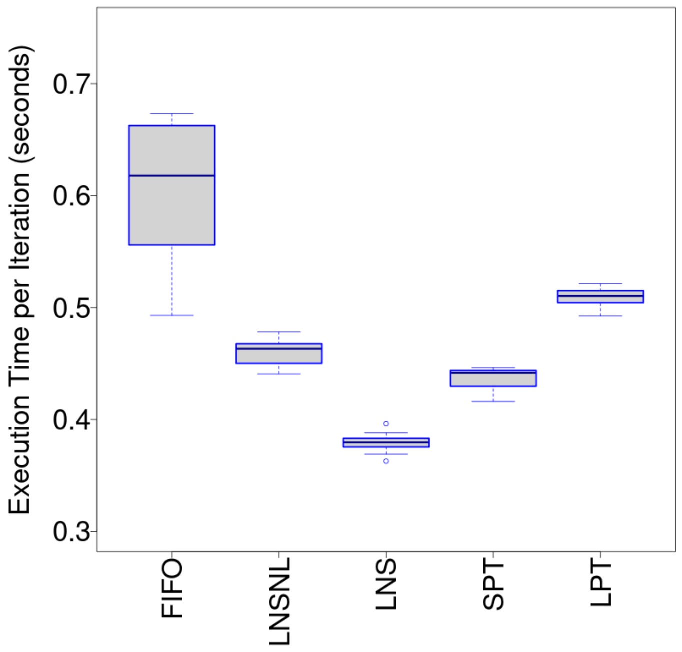

Figure 7. Execution time per iteration, with input from 3 video source

Figure 8 depicts the gain in the average execution time achieved by the different scheduling policies with respect to FIFO. The heuristics show similar behaviour for both scenarios, although the gain when three video sources are considered is reduced. The *Largest Number of Successors (LNS)* policy yields the best performance, achieving a 46.4% reduction in the average iteration execution time for a single video source, and 37.8% for three video sources. The lowest gain is achieved by the *Longest Processing Time (LPT)*, which still provides a considerable gain in time reduction with respect to the default FIFO scheduler of 22.5% for a single video source, and 16.5% for three video sources.

The performance of the heuristics is tightly related to the structure of the DAG, since the efficiency of the scheduling policy greatly depends on the number and nature of tasks and their dependencies. For the CLASS data analytics workflow, we have demonstrated that scheduling first the tasks with higher numbers of successors, which would correspond to the detection and tracking tasks based on the application DAG (Figure 3), is the approach that provides the best execution times. Prioritizing these tasks allows the retrieval and processing of more consecutive frames from the video sources (which are dependent from one frame to the next), while there is no dependency between the deduplication and federation tasks that can be executed concurrently or in a different order. On the other hand, the LPT strategy prioritizes the more time consuming analytics tasks, which in the CLASS workflow correspond to the data federation to the cloud, due to the increased time needed to transfer the data to the cloud and introduce them to the DKB.

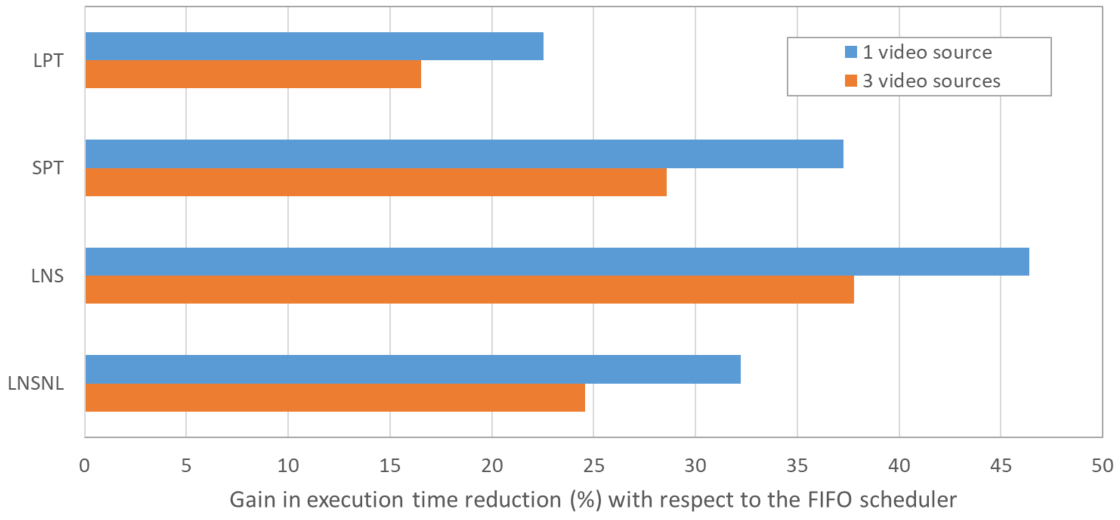

Figure 8. Reduction in execution time achieved by the proposed heuristics with respect to FIFO

Finally, Figure 9 presents the overall execution time of the workflow for 1 and 3 video sources, respectively. As expected, the obtained results show a similar behaviour as the execution times per iteration. Furthermore, these plots depict the response time upper bound (R^{ub}) for the heuristic scheduling methods, plotted as a red line. The R^{ub} has been calculated as the sum of the computation and communication volumes for each heuristic, as explained in Section 2.1. To account for the unpredictability of heterogeneous edge computing environments, a safety margin has been applied. In particular, a 50% safety margin has been added to the computation volume, while a 5% safety margin has been added to the communication volume.

The calculation of the R^{ub} is an important feature of the proposed system model, since it enable us to have a valid estimation of the response time of the workflow. This is a very important feature provided by the scheduler, since it provides some degree of predictability of the performance of the data analytics workflow.

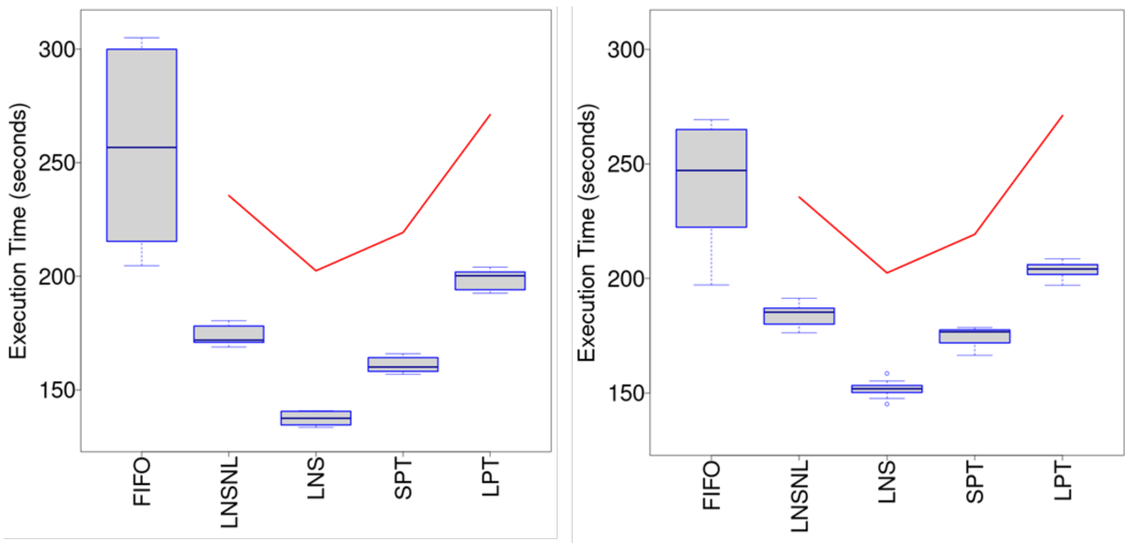

Figure 9. Total execution time of the workflow, for input from 1 (left) and 3 (right) video sources

4.3.2 Scenario 2: Exploiting the inner level of parallelism within the fog nodes

In the second scenario, the heuristics have been further improved to take into account the inner parallelism capabilities of the fog nodes, considering four cores per node. To do so, the heuristic considers two new conditions:

1. The communication cost among tasks being allocated in different cores from the same fog nodes is zero, and
2. The computation cost of tasks being allocated simultaneously in two different cores from the same fog node, is increased by a given factor, to take into account the hardware interferences (e.g., memory, interconnection network) suffered by the simultaneous execution of tasks.

These two new conditions have been included into allocation heuristics into the COMPSs framework, supporting the distribution of tasks across multiple cores of the same node. In this section the LNS and LNSNL heuristics have only been considered.

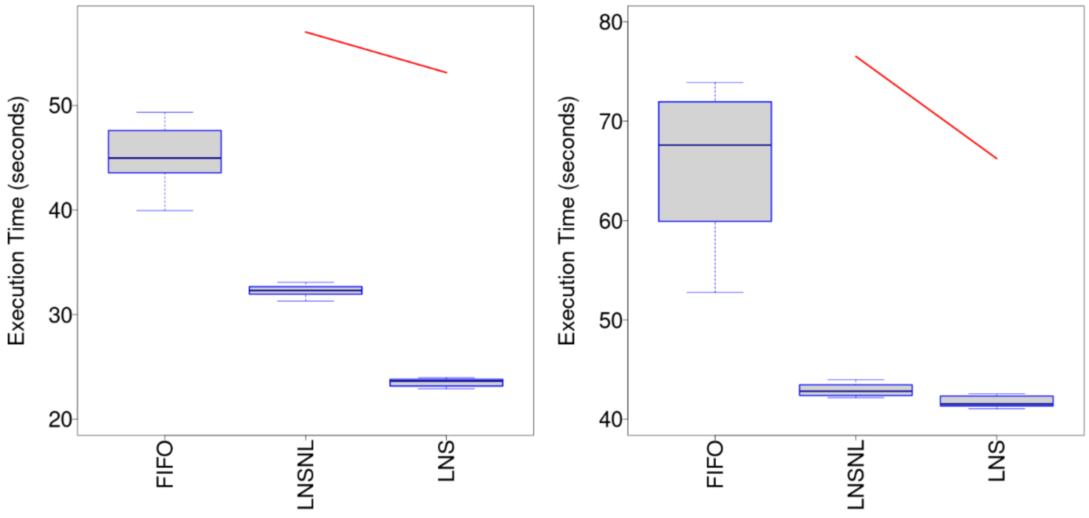

Figure 10. Total execution time of the workflow, for input from 1 (left) and 3 (right) video sources, when tasks are distributed across the four computing units of the fog nodes

Figure 10 shows the end-to-end execution time for the two heuristics LNS and LNSNL, versus the FIFO baseline. By exploiting the inner parallelism of the fog nodes, performance is significantly improved. Specifically, significant reduction in performance is achieved with respect to the first scenario when the multi-core architecture of the nodes is not fully exploited. Figure 11 shows the reduction in the end-to-end execution time with respect to the results of the first scenario (as depicted in Figure 9), with gains in the order of 70% for the LNS scheduling policy for both 1 and 3 video sources, and above 80% for the LNSNL policy.

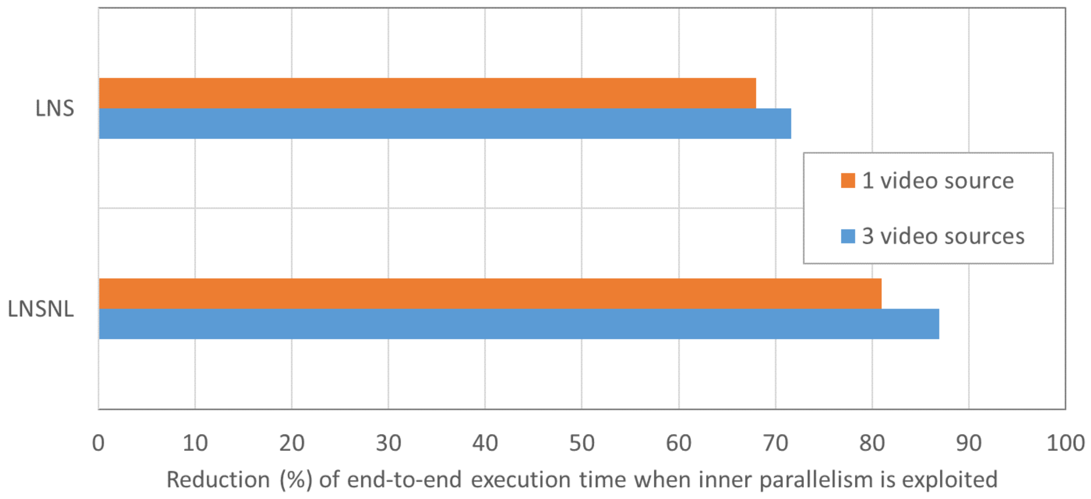

Figure 11. Reduction in end-to-end execution time achieved by the scheduling heuristics when inner parallelism in the fog nodes is exploited, with respect to the performance achieved in the first scenario (when the inner parallelism capability is available)

5 Conclusions

This document presents the CLASS allocation heuristics applied to schedule the data-analytics workflow responsible for the generation of the DKB into the MASA fog computing infrastructure. Two scenarios are considered:

i. the exploitation of parallelism across multiple fog nodes, and
ii. the exploitation of both parallelism across fog nodes and internally among cores within the same fog node.

The results shows that the full exploitation of the parallel capabilities of the MASA computing infrastructure, significantly reduces the end-to-end response time of the data-analytics workflow. Moreover, it opens the road for further investigation, encouraging the design of more sophisticated scheduling policies to match the specific requirements of the analytics. The presented schemes have yielded significant enhancements in terms of the end-to-end execution time of the data-analytics workflow and can be applied to any task-based analytics workflow.

Concretely, it remains as a future work to provide a more fine-grained control over the scheduling of the specific analytics tasks, enabling the prioritization of specific tasks based on their specific Quality of Service (QoS) requirements, could potentially provide further improvements in performance, from an analytics perspective.

Acronyms and Abbreviations

COMPSs	COMP Superscalar
D	Deliverable
DAG	Direct Acyclic Graph
DKB	Data Knowledge Base
FIFO	First-In-First-Out
HPC	High Performance Computing
MASA	Modena Automotive Smart Area
MS	Milestone
QoS	Quality of Service
SA	Software Architecture
TDG	Task Dependency Graph
WP	Work Package

References

[1] CLASS, "D2.7 - Final release of the CLASS Software Architecture," June 2021.

[2] CLASS, "D5.5 - Final Release of CLASS Big-Data Analytics Layer," June 2021.

[3] CLASS, "D4.6 - Validation of the Cloud Data Analytics Service Management and Scalability Components," March 2021.

[4] CLASS, "D3.6 - Validation of the CLASS edge computing subsystem," June 2021.

[5] CLASS, "D1.6 - Use case evaluation," June 2021.

[6] CLASS, "D2.4 - First release of the CLASS software architecture," March 2019.

[7] CLASS, "D2.6 - Second release of the CLASS Software Architecture," July 2020.

[8] R. M. Badia, J. Conejero, C. Diaz, J. Ejarque, D. Lezzi, F. Lordan, C. Ramon-Cortes and R. Sirvent, "Comp superscalar, an interoperable programming framework," *SoftwareX,* vol. 3, pp. 32-36, 2015.

[9] CLASS, "D2.1 - CLASS Software Architecture Requirements and Integration Plan," 2018.

[10] CLASS, "D5.4 - FInal release of an augmented platform for analytics workloads," July 2020.

[11] CLASS, "D1.4 - Final release of the smart city use case," July 2020.

[12] CLASS, "D1.2 - Final release of the Smart City Use-Cases," March 2019.

[13] CLASS, "D2.8 - Evaluation of the CLASS Software Architecture," June 2021.