D3.1

Intermediate API Definitions (report)

Kaleao Ltd

DOCUMENT IDENTIFIER:	D3.1
DUE DATE:	30/06/2019
DELIVERY DATE:	30/06/2019
CLASSIFICATION:	Public
EDITORS:	Mohammed Hazeef, Monica Vatteroni
DOCUMENT VERSION:	1.0

CONTRACT START DATE:	1st January 2018
CONTRACT DURATION:	36 months

Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement nº 780245.

Partners
EXUS (Coordinator), UNIMAN, ICCS, DFKI, NEUROCOM, KALEAO, CTI DIOPHANTUS, SPARKS, iProov

Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement nº 780245.

Partners
EXUS (Coordinator), UNIMAN, ICCS, DFKI, NEUROCOM, KALEAO, CTI DIOPHANTUS, SPARKS, iProov

CONTRIBUTORS	
Name	Organization
Mohammed Hazeef	Kaleao
Monica Vatteroni	Kaleao
Giagkos Mytilinis	ICCS
Katerina Doka	ICCS
Viktor Rosenfeld	DFKI

PEER REVIEWERS	
Name	Organization
Christos Kotselidis	The University of Manchester

REVISION HISTORY		
Version	Date	Modifications
0.1	29/05/2019	Outline
0.2	14/06/2019	Integration of the contribution from partners
0.3	27/06/2019	Addressed review comments
1.0	30/06/2019	Final version After reviews

Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement nº 780245.

Partners
EXUS (Coordinator), UNIMAN, ICCS, DFKI, NEUROCOM, KALEAO, CTI DIOPHANTUS, SPARKS, iProov

1. Executive Summary

Data Processing Frameworks required to manage and allocate tasks for E2Data have been identified as Apache YARN and Flink. Both these frameworks have been fully analysed and implementation changes identified. The changes proposed for both these Frameworks are kept to the minimum, without losing the functionality required to utilize the heterogeneity of a cluster. The analysis of these frameworks has proven that implementations and testing can be done independently and their integration can be done at a later stage to include the E2Data Intelligent scheduler. With the completion of integration and testing phase, a final document with the changes proposed to these frameworks will be published.

Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement nº 780245.

Partners
EXUS (Coordinator), UNIMAN, ICCS, DFKI, NEUROCOM, KALEAO, CTI DIOPHANTUS, SPARKS, iProov

2. Contents

Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement nº 780245.

Partners
EXUS (Coordinator), UNIMAN, ICCS, DFKI, NEUROCOM, KALEAO, CTI DIOPHANTUS, SPARKS, iProov

3. Overview

Cloud infrastructures employ hardware accelerators such as GPUs, FPGAs, ASICs etc. to optimize execution of resource intensive workloads. However, these accelerators are not fully utilized due to the complexities and expertise required for manual programming. One of the proposed contribution of E2Data is to produce a scheduler on an heterogenous cloud platform that can automatically select the appropriate type and size of resources to execute data.

Cloud platforms utilize already available software infrastructures to deploy their workload on the hardware. A better interaction and close architectural abstraction was found between the interworking of Apache Flink-Yarn, that has some capability to identify heterogeneous hardware resources available on the cluster that could be utilized by the E2Data scheduler.

The aim of this document is to define and develop the necessary abstractions and enhancements inside Apache Flink and Apache Yarn. This will allow discovery of the capabilities of heterogeneous hardware and to create the API definition for Apache Flink and Yarn. This will also ensure interoperability with other Big Data and Cloud Provider suites.

These API definitions will be the inputs to modify/implement Apache Flink and Yarn for evaluation purposes as well as to allow other Big Data and Cloud software solutions to access the E2Data capabilities.

4. Apache YARN in E2Data

A basic component of E2Data that lives at the heart of its software stack is the resource management framework (RMF) it uses. The RMF is responsible for monitoring cluster utilization, dynamically acquire resources, dispatch tasks to them and provide fault-tolerance at the hardware level. It is also the RMF that the E2Data scheduler (HAIER) contacts in order to learn about the cluster status and take educated decisions on the execution of a job.

In E2Data, we use Apache YARN for managing resources. YARN features a JVM-based, distributed master-slave architecture. The *ResourceManager*, who acts as a master, is the ultimate authority that arbitrates resources among all the applications in the system. The *NodeManager* is the per-machine framework agent who acts as a slave and is responsible for the running containers: monitoring their resource usage and reporting it to the *ResourceManager*. By continuously aggregating information from all *NodeManagers*, the *ResourceManager* maintains a real-time utilization overview of the whole cluster. The adoption of YARN offers several advantages:

1. Native support for managing hardware accelerators.

Traditionally, the resource types a YARN *NodeManager* could manage were CPU, memory, disk and network. Nevertheless, as per YARN 3.0.0, hardware accelerators such as GPUs and FPGAs are also natively supported. An application can request YARN containers and specify in a straightforward manner how many and what type of accelerators (GPU/FPGA) each of them should have.

2. Easy integration with JVM-based Big Data frameworks.

E2Data considers Flink applications as first-class citizens in the Big Data processing realm and opts for their optimization. As both Flink and YARN are developed with JVM-based technologies, they can be

Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement nº 780245.

Partners
EXUS (Coordinator), UNIMAN, ICCS, DFKI, NEUROCOM, KALEAO, CTI DIOPHANTUS, SPARKS, iProov

easily integrated and tightly coupled. YARN also comprises the main RMF alternative of several other data processing frameworks (e.g., Apache Spark). By building our system on top of YARN, we can keep it Flink-independent for a large part of the E2Data codebase. This can put the foundations for establishing heterogeneous processing in a broader category of Big Data frameworks.

3. Large user-base, promising for adoption in practice.

As YARN is used by multiple Big Data frameworks, it is the de-facto standard for resource management in the Apache ecosystem. As YARN has already gained the trust of the public in both academia and industry, a solution based on it can be easily adopted in practice.

While the benefits stemming from YARN are multiple, it cannot be used intact in the E2Data architecture. The design of E2Data requires fine-grained control over the hardware resources. A GPU device of a specific type may be more advantageous than another one for a given task. Thus, the desired behavior of HAIER is to request from the *ResourceManager* a container equipped with the specific GPU. However, this is not a supported functionality in YARN as it does not make any distinction among hardware devices of the same kind.

For this reason, we have done pilot changes to Apache YARN and have implemented the additional features for the fine-grained monitoring and management of the hardware accelerators. Our version of feasibility study can be found at E2DataYARN . This source repository that is now private will soon be forked from the main Yarn development to bring E2Data changes upstream to YARN.

4.1. YARN Implementation Changes

Each *NodeManager* of a YARN cluster can be configured to manage a set of accelerator types. This can be done through the *yarn.resource-types* property of the *resource-types.xml* configuration file. By default, the only valid values for this property are: (i) yarn.io/gpu and (ii) yarn.io/fpga. In this way, GPU- and FPGA-based acceleration is enabled.

Upon starting up a new *NodeManager* daemon, the *resource-types.xml* file is checked and, if the *yarn.resource-types* property is set, YARN does the following:

1. Contacts a third-party utility (e.g., nvidia-smi, clinfo) in order to get all the required information about the configured accelerator type.
2. Initializes a *ResourcePlugin*, which is an internal structure responsible for handling the corresponding resource.
3. Registers the newly created *ResourcePlugin* to the *ResourcePluginManager*. This is a dictionary-like structure which is unique per *NodeManager*.

The key idea for providing fine-grained control over accelerators of the same type (e.g., GPU) is to instantiate multiple *ResourcePlugins* where each of them corresponds to a device of different processing capabilities. Let us examine for example the GPU case. In the default version of YARN, GPUs are managed through the *GPUResourcePlugin* and there is only one such plugin instantiated per *Node Manager*. Thus, YARN treats the same way all GPU devices of a physical node. To remedy this, we allow the existence of multiple *GPUResourcePlugins* per *NodeManager*, where each of them is customized and bound to a specific device model. For example, a *NodeManager* in the E2Data YARN may have a

Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement nº 780245.

Partners
EXUS (Coordinator), UNIMAN, ICCS, DFKI, NEUROCOM, KALEAO, CTI DIOPHANTUS, SPARKS, iProov

GPUResourcePlugin which is responsible only for the management of Tesla V100-SXM2-32GB GPUs. Therefore, monitoring or requesting accelerators of a specific type and model (e.g., Tesla V100-SXM2-32GB GPU) directly translates to accessing the corresponding plugin that is maintained in the *ResourcePluginManager*. For enabling GPU usage, YARN uses several internal structures that are managed by the corresponding *GPUResourcePlugin.* Thus, there is a chain of modifications that need to be performed In order to support multiple plugins per physical node. To that end, we have also modified the: *GpuResourceAllocator*, *GpuNodeResourceUpdateHandler*, *GpuResourceHandlerImpl* and *CGroupsHandlerImpl* YARN classes.

5. Apache Flink in E2Data

In this section, we describe the changes to the Apache Flink Big Data Framework to make it aware of heterogeneous hardware accelerators. Specifically, we add the following two capabilities:

1. Expose hardware accelerators as computing resources (task slots) on which a Flink task can be executed.
2. Enable the Flink *JobManager* to route tasks to hardware-specific task slots as determined by the HAIER scheduler.

First, we provide some background on the internal Flink implementation that provides task slots and routes tasks to them (Section 5.1). Second, we describe the necessary implementation changes to make Flink aware of heterogeneous accelerators (Section 5.2). Third, we describe changes in the public Flink API (Section 5.3).

Our changes are implemented in a dedicated branch “heterogeneous-big-data-platform”[3] of the main E2Data code repository. The implementation described in this document is tagged with “deliverable-3.1”[4]. Note that the repository is currently private.

[3] https://github.com/E2Data/flink-tornado-internal/tree/heterogeneous-big-data-platform

[4] https://github.com/E2Data/flink-tornado-internal/tree/deliverable-3.1

Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement nº 780245.

Partners
EXUS (Coordinator), UNIMAN, ICCS, DFKI, NEUROCOM, KALEAO, CTI DIOPHANTUS, SPARKS, iProov

5.1. Current Flink implementation

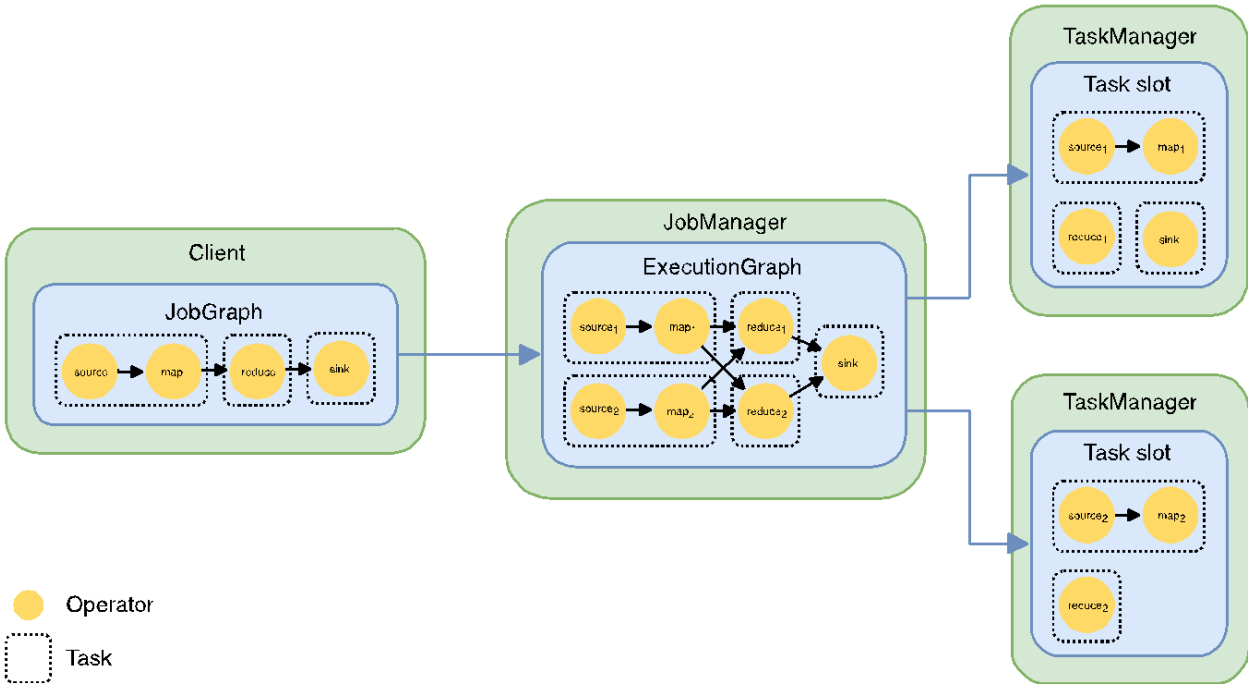

FIGURE 1 INTERPLAY BETWEEN CLIENT, JOBMANAGER, AND TASKMANAGER IN APACHE FLINK.

In Figure 1, we provide a schematic overview of the execution of a Flink dataflow program or Flink job. Three entities participate in the execution:

1. The Flink *Client* receives a Flink dataflow program from the user and constructs a *JobGraph*. The nodes of the *JobGraph* represent the operations of the Flink program (e.g., map, reduce, join, etc.) and the edges represent the dataflow. The Flink *Client* optimizes the *JobGraph* according to some criteria. Crucially, multiple operators can be chained together. These operator chains are called *tasks*.
2. The Flink *JobManager* receives the *JobGraph* and transforms it into a distributed *ExecutionGraph*. The *ExecutionGraph* consists of multiple instances of individual tasks according to the specified parallelism. Each of these instances run in parallel with others. There is typically a single *JobManager* for each Flink program.
3. The Flink *TaskManager* represents the worker nodes on which the Flink program is executed. There is typically a single *TaskManager* running on each worker node in the cluster. Each *TaskManager* provides one or more task slots according to the hardware resources (e.g., number of CPU cores) of the worker node. Upon startup, the *TaskManagers* register themselves with the *JobManager*. The *JobManager* distributes tasks to individual task slots.

Flink also includes functionality to specify available and required resources. Currently, these resources model CPU cores and the size of different memory spaces (e.g., native memory, JVM heap memory, etc.). In addition, they include a key/value map in which arbitrary resource requirements can be specified. Each task specifies the minimum and preferred resources using a *ResourceSpec* object. Each task slot specifies available resources using a *ResourceProfile* object. When allocating a task slot to a task, the Flink *JobManager* matches available *ResourceProfiles* with requested *ResourceSpecs*. However,

Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement nº 780245.

Partners
EXUS (Coordinator), UNIMAN, ICCS, DFKI, NEUROCOM, KALEAO, CTI DIOPHANTUS, SPARKS, iProov

the current Flink implementation uses default values, e.g., *ResourceSpecs.DEFAULT* and *ResourceProfiles.UNKNOWN*.

5.2. Flink Implementation Changes

In this section, we describe our implementation of the first capability, i.e., to expose hardware accelerators as task slots on which a Flink task can be executed. Note that no internal changes are necessary to implement the second capability, i.e., to route tasks to hardware-specific task slots as determined by the HAIER scheduler. Instead, it is sufficient for the HAIER scheduler to modify the minimal and preferred resources of each task (see next section).

First, we describe our changes to the Java source code of Apache Flink. Second, we describe the application logic to provide hardware-specific task slots.

5.2.1. Changes to Java Code

We introduce the Java enum *ProcessingUnitType* to model different processor types, i.e., normal CPUs and hardware accelerators. At the moment, we model CPUs, GPUs, and FPGAs. The value ANY signifies an undetermined processor resource.

```
public enum ProcessingUnitType {
  ANY, CPU, GPU, FPGA
}
```

We extend both *ResourceSpec* and *ResourceProfile* to include a *ProcessingUnitType* field as a member. In addition we provide the following default *ResourceSpec* and *ResourceProfile* instances:

```
ResourceSpec DEFAULT = new ResourceSpec(ProcessingUnitType.ANY, …);
```

```
ResourceProfile ANY = new ResourceProfile(ProcessingUnitType.ANY, …);
ResourceProfile ANY_GPU = new ResourceProfile(ProcessingUnitType.GPU, …);
ResourceProfile ANY_CPU = new ResourceProfile(ProcessingUnitType.CPU, …);
ResourceProfile ANY_FPGA = new ResourceProfile(ProcessingUnitType.FPGA, …);
```

5.2.2. Changes to application logic

When a *TaskManager* is started on a worker node, it creates a number of normal CPU task slots as given by the (existing) configuration option *taskmanager.numberOfTaskSlots*. These are configured as

Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement nº 780245.

Partners
EXUS (Coordinator), UNIMAN, ICCS, DFKI, NEUROCOM, KALEAO, CTI DIOPHANTUS, SPARKS, iProov

ResourceProfile.ANY_CPU. The *TaskManager* then reads the value of the configuration option *taskmanager.useAccelerator* (see Section 5.3). If this option is set, the *TaskManager* queries YARN to receive a list of installed hardware accelerators. These accelerators are identified by a resource string, e.g., yarn.io/gpu-teslav100sxm232gb (see Section 2). For each installed accelerator, the *TaskManager* creates a single task slot with the corresponding identifier string stored in the *extendedResources* key/value map of the task slot's *ResourceProfile*.

We extend the matching of *ResourceSpec* to an available *ResourceProfile* to include the *ProcessingUnitType* and *extendedResources*. Through the extended match, we ensure that the scheduling requested by HAIER is respected.

5.3. Flink API Changes

In this section, we describe our changes to the public Flink API. First, we describe configuration options introduced to provide hardware-specific task slots. Second, we describe how the HAIER scheduler can specify the preferred or required processor type for a specific task.

5.3.1. Configuration options

We introduce the following configuration option to the flink-conf.yaml configuration file:

Name	Values	Default	Description
taskManager.useAccelerator	true, false	false	Provide task slots for each accelerator installed on the node. If true, the *TaskManager* queries YARN to determine the installed accelerators in the node. For each accelerator, a task slot is created which specifies the YARN name of the accelerator in its *ResourceProfile* (extended resource key 'accelerator.name').

5.3.2. API usage by HAIER scheduler

The tasks of the *JobGraph* created by the Flink Client do not specify any particular resource requirements. That is, their minimal and preferred resource specifications are set to *ResourceSpec.DEFAULT*. The HAIER scheduler has to specify the type of device on which the task should be executed based on the three options listed below.

1. Leave the resource specification unchanged. In this case, the *JobManager* can schedule the task on any available task slots, i.e., normal CPU task slots and/or hardware-accelerated task slots.

Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement nº 780245.

Partners
EXUS (Coordinator), UNIMAN, ICCS, DFKI, NEUROCOM, KALEAO, CTI DIOPHANTUS, SPARKS, iProov

2. Change the resource specification to require a specific *ProcessingUnitType*, i.e., *ProcessingUnitType.GPU*. In this case, the *JobManager* can schedule the task on any available task slot that provides a corresponding *ResourceProfile*.
3. Change the resource specification to require a specific *ProcessingUnitType* and specify a YARN resource identifier in *ResourceSpec.extendedResources*, e.g., yarn.io/gpu-teslav100sxm232gb. In this case, the *JobManager* must schedule the task on the task slot created for the specific device.

6. Conclusions & Next Steps

Apache YARN, that acts as a distributed container manager can work with several data processing frameworks such as Apache Spark, Apache Flink, etc. Keeping the Implementation changes independent to Flink (per se) would allow any changes to YARN to be easily integrated with the other data processing frameworks. The changes that are proposed to be implemented in YARN are self-contained and as minimal as possible.

Apache Flink, which is chosen as the preferred Data processing Framework of E2Data, needs to be extended with additional API for the E2Data Intelligent scheduler (HAIER) to make use of the heterogeneity of the cluster. This work can be done independently to the changes in YARN. However, to make use of the real hardware resources, an integration is necessary between YARN and Flink. The changes proposed for Flink are again as minimal as possible.

6.1. Next Steps

Since both YARN and Flink can be modified independently, partners should go ahead to implement these modifications and test them to their entirety. Care should be taken that backward compatibility of these frameworks is not impaired. The modification of these frameworks should be finalised once implementation and testing are complete.

Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement nº 780245.

Partners
EXUS (Coordinator), UNIMAN, ICCS, DFKI, NEUROCOM, KALEAO, CTI DIOPHANTUS, SPARKS, iProov

Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement nº 780245.

Partners
EXUS (Coordinator), UNIMAN, ICCS, DFKI, NEUROCOM, KALEAO, CTI DIOPHANTUS, SPARKS, iProov

7. Abbreviations

RMF	Resource Management Framework
HAIER	Heterogeneity-aware, Intelligent Scheduler
YARN	Yet Another Resource Negotiator
GPU	Graphics Processing Unit
FPGA	Field Programmable Gate Array

Co-funded by the Horizon 2020 Framework Programme of the European Union under Grant Agreement nº 780245.

Partners
EXUS (Coordinator), UNIMAN, ICCS, DFKI, NEUROCOM, KALEAO, CTI DIOPHANTUS, SPARKS, iProov