A Conceptual Framework for a Flexible Data Analytics Network

Daniel Tebernum and Dustin Chabrowski
Fraunhofer Institute for Software and Systems Engineering,

Keywords: Data Analytics, Distributed System, Edge Computing, Data Provenance, Architecture.

Abstract: It is becoming increasingly important for enterprises to generate insights into their own data and thus make business decisions based on it. A common way to generate insights is to collect the available data and use suitable analysis methods to process and prepare it so that decisions can be made faster and with more confidence. This can be computational and storage intensive and is therefore often outsourced to cloud services or a local server setup. With regards to data sovereignty, bandwidth limitations, and potentially high charges, this does not always appear to be a good solution at all costs. Therefore, we present a conceptual framework that gives enterprises a guideline for building a flexible data analytics network that is able to incorporate already existing edge device resources in the enterprise computer network. The proposed solution can automatically distribute data and code to the nodes in the network using customizable workflows. With a data management focused on content addressing, workflows can be replicated with no effort, ensuring the integrity of results and thus strengthen business decisions. We implemented our concept and were able to apply it successfully in a laboratory pilot.

1 INTRODUCTION

Nowadays, data is crucial when it comes to fasten processes, optimizing business models, and uncovering knowledge that could lead to an advantage over competitors (Otto and Österle, 2016). Many companies see data as a resource they can mine, process, and then use to their advantage. Often, data is analyzed to detect hidden patterns or to get better insights into it (Miloslavskaya and Tolstoy, 2016). It is therefore essential that enterprises have a data friendly environment, where they can exploit the full potential of their data anytime and in the most flexible way. When it comes to harnessing their own data, enterprises face a variety of challenges. One can be the mere amount of available data, that is at the same time rapidly increasing. Ernst & Young claim that the "exponential data growth is a fundamental problem that is continuing to overwhelm most businesses, and it is accelerating" (Cudahy et al., 2016). Besides volume, also variety and velocity in data can be a huge challenge. In literature, these properties are often mentioned under the term *Big Data* (Chen et al., 2014). But even if the available data does not meet all requirements defined by the big data term, it can still be a hard task to get the needed insights into ones own data (Gandomi and Haider, 2015). Strategies like outsourcing data and its processing to a cloud service are not always feasible and financially reasonable. Also, a limited upstream capacity can be an obstacle when working with terabytes of data. Another strategy may be to build a local cluster that implements concepts like a data lake to run data processing pipelines. Here, too, possible costs for obtaining suitable hardware have to be taken into account. In addition, the data must first migrate to these systems. This may create unwanted duplicates that require additional storage capacity. On top of that, there is a risk that resources will be wasted, since much of the data in a data lake could be abandoned or forgotten, thus creating so-called data swamps (Buyya et al., 2018). A further challenge for enterprises becomes apparent when looking at the topic of data sovereignty. Both strategies mentioned above can be vulnerable in this case. By uploading business relevant data to the cloud, it needs to be clearly defined to what extent the enterprise's own data sovereignty has been preserved when third parties have access to it (De Filippi and McCarthy, 2012). It should also be noted that the idea of data sovereignty is not limited to the outer boundaries of an enterprise. The exchange of data between internal departments can already be an insurmountable obstacle because of sensible data or licensing issues. When using data for business decisions, addi-

Tebernum, D. and Chabrowski, D.
A Conceptual Framework for a Flexible Data Analytics Network.
DOI: 10.5220/0009827402230233
In *Proceedings of the 9th International Conference on Data Science, Technology and Applications (DATA 2020)*, pages 223-233
ISBN: 978-989-758-440-4

tional guarantees in reproducibility have to be made. Data processing must be repeatable at any time in order to strengthen the confidence in decision-making. Finally, for data-driven enterprises, it must be ensured that the environment for working with data is robust and highly available. Central cloud and even local services may be subject to malfunctioning making data-driven enterprises vulnerable to operational downtimes. For instance, in the year 2017, 600 flights were cancelled due to a failure, resulting in a loss of $112 million[1].

To overcome these problems, we suggest that data-driven enterprises extend their existing data processing environment by including the already available computing and storage resources at the edge of the computer network. This should be done in such a way that both the edge devices and local servers or cloud resources can be seamlessly integrated into one network. Our main contribution in this paper is therefore a conceptual framework that describes how to build a system that is flexible in terms of supported devices and executable code and thus forming a data analytics network. It is intended to be a template that addresses the challenges mentioned above and offers features such as availability, robustness, scalability, flexibility, and reliability when working with data. In the following we will further motivate the need for such a solution based on an enterprise use case. We will then list the relevant requirements and present our concept. A specific implementation of our concept will be developed and evaluated to show that we met our requirements. Finally, we compare our approach to existing solutions from literature and practice and discuss our results.

2 USE CASE EXAMPLE

Here, we illustrate an enterprise use case that is based on an existing real world scenario. It will motivate why a flexible and scalable data analytics network, capable of using existing edge devices, may be desirable for data-driven enterprises. The involved enterprise is an active player in the pharmaceutical sector. It carries out its own development and research and also produces its own medications. Due to this multitude of activities, a great amount of data is generated on a daily basis. It is stored locally on end devices of the employees, a central network share, or in the cloud. The data is already being used for daily work in various departments. But often, it is consumed where it is generated, resulting in data and knowledge silos. Within the company, it often appears that interesting and relevant data to one's own work can only be found by accidental contact with people from other departments. This is particularly important for the employed data scientists that want to gather more insight into the data and extract knowledge from it. As we know from literature, data scientists tend to spend around 80% of their working time in finding suitable data, maybe more (Deng et al., 2017). The enterprise is fully aware of this problem and would like to introduce a so-called data catalog. Data catalogs store metadata about data and thus are able to provide a helpful search functionality. Metadata can contain many different types of information. From technical information like file format and file size, textual descriptions, and statistical analysis to information about the data owner and the data's application areas. In general, "metadata is key to ensure that resources will survive and continue to be accessible into the future" (NISO Press, 2004). The more meaningful metadata is available, the better the search functionality in the data catalog operates. However, due to the large amount of data, it is no longer realistic to insert metadata into the catalog by hand. Still, a lot of metadata can be extracted using automated solutions. Some methods are quite trivial, while others can be highly complex. In any case, there will be a very large number of metadata services, since each type of data has different requirements for the extraction of metadata.

In combination with the potential sensitivity of the data, a cloud-only solution is not preferred. The company's employees are equipped with personal computers, which are far from being fully utilized and therefore have free computing and storage resources. These resources should help to retrieve the metadata for the data catalog. However, existing servers or cloud capacities should not be completely left out, as they could be included in more complex calculations. Since business decisions can be made on the basis of metrics from the data catalog, the extraction of metadata should also be reproducible. Furthermore, some data is only accessible to a small circle of people. The extraction of metadata can therefore not be performed on any computer, but only on designated machines. The IT department of the company therefore wants a uniform concept for the integration of the different types of end devices into a company-wide analysis network. In particular, the fact of regularly joining and leaving devices, possible errors and a feasible scaling of the solution should be considered. The metadata extraction services are implemented by different teams and require different domain knowl-

[1] https://www.datacenterknowledge.com/archives/2017/05/30/british-air-data-center-outage-feeds-outrage-at-airline-cost-cuts

edge. Therefore, the implementation of these services should focus on the functionality, i.e. on the what, and not on the how.

3 REQUIREMENTS

In this section, the most important requirements for our solution are presented, that, at the same time, define the objectives we want to reach. The requirements are derived from a literature analysis with focus on data analytics platforms (see Chapter 7) in combination with the already mentioned challenges in the previous two chapters. Table 1 shows the requirements in a structured way. In the following we will provide a more detailed explanation of these requirements.

Devices that participate in the analytics network should be able to perform any kind of given analysis task, as long as these tasks are in general technically compliant to our system (FR1). This reduces the number of different types of participants in the network, which, in turn, minimizes overall complexity. Each device itself decides how much computing and storage resources should be donated to the network (FR2). If it is known that the resources of a device must be utilized more regularly, this can be taken into account in the configuration for the analysis network. In general, however, the goal is that particularly underutilized devices in the enterprise network should donate their resources (NFR1) so there will be no limitations in daily business and no additional costs for hardware. The targeted solution should therefore be able to support a wide range of different hardware and operating systems (NFR2). This will also avoid the need to develop specific solutions for each type of device. With every additional device joining the network, the probability of them leaving it increases, whether intentionally or due to an error. It is the responsibility of the system to deal with such events appropriately in a self-orchestrating way, thus allowing the leaving and joining of devices at any time (Hoque et al., 2017) (FR3). To derive maximum benefit from the network, it should not be specialized in processing a particular type of data (FR4). This is mainly to enable a high level of domain independence. Applications for data analysis are often structured so that the programming language for the application logic, the data format, or the runtime environment are predefined. In order for the network to remain of interest in the future and to lower the entry barrier for developers, no major restrictions should be placed on the choice of programming language (NFR3). Looking at analyses, they are rarely atomic and must necessarily be performed by a single device. Often analyses can be divided into sequential and parallel steps. The description of an analysis is therefore to be defined using a workflow notation, which in turn is to be used as a recipe by the network to perform the analysis itself (FR5). By composing workflows of smaller tasks, developers can also easily reuse already existing solutions. As certain responsibilities and capabilities still need to be established in a decentralized network, an effective orchestration of the resources becomes a major challenge (Hoque et al., 2017). Therefore, our solution must be able to identify free capacities and distribute tasks in our analytics network appropriately (NFR4). In addition to the pure orchestration of resources, the distribution of data and code necessary for the analyses is also to be automated (FR6). Since the network is expected to handle a large amount of data, the distribution of data and code must be able to scale practically (NFR5). Sometimes, one does not want to distribute data or even code into a network that is not trusted one hundred percent. We already discussed the need for data sovereignty. The system must be able to restrict the execution locations and distribution of data and code (FR7). It should be noted that the result of an analysis is not very useful if it cannot be reproduced anymore (Deelman and Chervenak, 2008). While this is a common thing when it comes to scientific workflows (Liu et al., 2015), this is also crucial for data-driven enterprises, as their business decisions are powered by generated data insights. Thus, capturing the whole environment of such processes is required to enable this essential feature (Nekrutenko and Taylor, 2012) (FR8). To reduce the dependency and costs of upstream traffic to external cloud services, the system must be powerful enough to distribute code and data as well as scheduling tasks solely inside the network (NFR6). Since this comes along with a number of potential error sources, failure and fault tolerance must be provided to execute analyses effectively (NFR7).

4 CONCEPTUAL FRAMEWORK

4.1 Analysis Tasks

As described in the previous section, we want to give the opportunity of participating in our analytics network to as many computers in the network as possible. The solution should not be coupled to a specific hardware architecture and should support the most common operating systems. For this purpose, it is necessary to create an abstraction layer that allows the encapsulation of executable code. Typically, there

Table 1: List of requirements to the Data Analytics Network.

	Abbrev.	Requirement
Func. Req.	FR1	Nodes inside the network can execute heterogeneous analysis tasks.
	FR2	Nodes can specify how much computation power and storage they want to donate.
	FR3	Nodes can join and leave at any time.
	FR4	Generic types of data can be processed.
	FR5	Analyses are defined and executed as workflows.
	FR6	Distribution and execution of tasks and data in the analytics network must be done autonomously.
	FR7	The scheduling can be restricted to privacy, responsibility, and data governance constraints.
	FR8	Workflows are fully reproducible with guarantees on the data and code used.
Non-Func. Req.	NFR1	Existing but idle company resources can be incorporated.
	NFR2	Analyses can be run on heterogeneous devices.
	NFR3	There are no programming language restrictions for implementing analysis tasks.
	NFR4	Available resources should be orchestrated in an efficient way.
	NFR5	Analysis code and data is distributed in a scalable way.
	NFR6	Upstream traffic to external cloud services can be reduced.
	NFR7	A failure and fault tolerance provides high availability and robustness.

are two methods to achieve this. The first method is classical virtualization. This creates a guest system on the host system in which any code can run encapsulated. However, this method is often too computational and memory intensive, as it usually requires the virtualization of a complete operating system. Another possibility is the packaging of code inside a container environment. This provides a more lightweight form of virtualization that reuses existing components of the operating system. These packages, sometimes called container images, also have an advantage when it comes to distribution, as they are smaller and thus have lower bandwidth cost. In essence, a container image can be handled like a regular file and thus be easily distributed. In addition, this technology is widely supported (Morabito et al., 2015). Because of these properties, we decided to use container images as an important part of our concept (FR1, NFR2). By utilizing container images, we can also cover the requirement of being programming language independent (NFR3). Within a container, arbitrary environments can be created and thus also different compilers or interpreters can be used. This is intended to keep the analytics network attractive for different types of developers, as there is no need to learn a specific language. In addition, this encapsulation adds an additional isolation between the host system and the executed analytics code. To reschedule a failed task at any time and without side-effects, the *Analysis Task* (see Fig. 1 center) itself needs to be stateless and idempotent (FR8, NFR7). Because of this, we can avoid the need of capturing, distributing, and recreating previous states from other nodes.

A complete analysis rarely consists of a single task but requires various operations on a dataset. Therefore, this set of tasks is assembled with a workflow model based on a directed acyclic graph (DAG). The DAG is expressive enough to incorporate sequences, conditions, and parallelism as well as tasks and data dependencies. For scientific workflows, this model is often already sufficient as long as the workflow can be reproduced (Liu et al., 2015). To describe such workflows, we use a declarative *Workflow Definition* file (see Fig. 1 left top) (FR5).

As a general concept, we chose files to contain the input and output data of an *Analysis Task*. Considering the existing amount of heterogeneous data sources, the ubiquity of files allows to integrate a wide range of data types (FR4). For example, tabular data, documents, and multimedia can be persisted. This even applies to native technologies like relational or document-oriented databases and key-value stores. To include streaming data, e.g. for IoT analytics, these would be batched into files. In general, anything that can be represented as a file can be processed by the analytics network.

4.2 Analytics Network Orchestration

The management of the available network resources is done by an *Orchestration Component* (see Fig. 1 left). It is responsible for distributing *Analysis Tasks* to the *Worker Nodes* (see Section 4.3) in the network in a meaningful way. To make this possible, the *Worker Nodes* need to inform the *Orchestration Component* in regards to their hardware properties, like memory and processing capability, their current working status, and their working load. Based on this, the *Orchestration Component* is responsible for finding an adequate *Worker Node* to place an *Analysis Task* on (FR2, FR6, NFR4).

In order to allow the *Orchestration Component* to

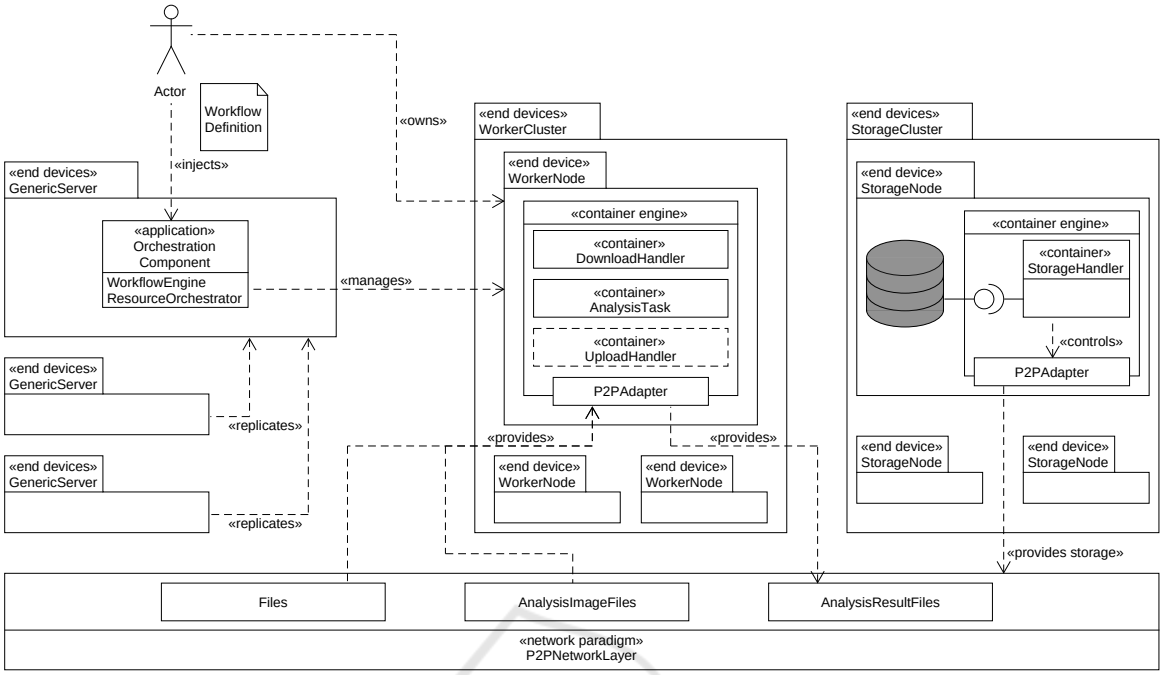

Figure 1: Architecture of the conceptual framework.

determine how analyses are to be performed and distributed in the network, it utilizes a *Workflow Engine*. This component receives a *Workflow Definition* as input and parses its structure. Based on this model, the *Workflow Engine* ensures that the tasks are executed in the described order and receive their input data. The *Workflow Definition* can also contain certain restrictions where to place a task which is taken into consideration during the scheduling process (FR7). For instance, this can be a selection of some machines or specific hardware requirements. In addition, the *Workflow Engine* actively observes the status of the individual tasks and their host machines to provide basic fault tolerance (NFR7). A common scenario in edge computing is the unexpected downtime of a node. If this happens, the orchestrator can reschedule the task (FR3). Since the *Analysis Tasks* are stateless and idempotent, this is a safe operation. Because this is the only centralized component in our concept, it is recommended to incorporate replication strategy to avoid this possible bottleneck.

4.3 Task Execution Environment

If an *Analysis Task* was implemented according to the previous mentioned conceptual definition, we need an environment for execution. Computers that want to provide computing power and meet the requirements for running *Analysis Tasks* can contact the *Orchestration Component* (FR1, FR3, NFR1). When a task is assigned to a so-called *Worker Node* (see Fig. 1 center), it fetches the specific *Analysis Task* encapsulated by an analysis image and runs the respective container. The assignment is done by the orchestrator component. To keep data management operations off the analysis image, two other containers are used additionally. This allows developers to simply implement the *Analysis Tasks* without requiring knowledge about the I/O handling. Thus, before the actual *Analysis Takes* place, a *Download Handler* (see Fig. 1 center) is spawned to provide the required input data. After the analysis is completed, an *Upload Handler* (see Fig. 1 center) is created to send the resulting files to a *Storage Cluster* (see Section 4.4). That way, the data can be made accessible for subsequent tasks of the workflow, since after finishing a task, a *Worker Node* restores its pristine state. We decided this way because of several reasons. On the one hand, we wanted to prevent possible side effects when running over a long period of time. On the other hand, we want to claim a node's resources only as long as needed (NFR4). It is important to note that the *Upload Handler* is optional if the output data is not returned to the analytics network for further workflow execution. For example, this can be useful when the last task is executed on the actor's machine.

4.4 Data Management

At the core of our conceptual framework, we want to establish a common data management. The data management is responsible for transferring and storing the data we work within our analytics network. As already pointed out in the requirements, we are focusing on file-based data. Therefore, we need a data management system that is particularly good at handling files. To keep the load off a central storage, we integrated a decentralized P2P network for the distribution based on a distributed hash table (DHT) (NFR6). The data on the nodes within the network is identified with a cryptographic hash function whose value is stored on the DHT. Peers then use the DHT to find the data for a given hash value and can retrieve it from potential multiple peers. A peer that provides data can preprocess it before sharing it in the network, thus ensuring its data sovereignty. Since analysis images and the data itself can be stored as files, we can address all required assets for a workflow with this approach (NFR5). By using only content-addressed files, we ensure that a given workflow always deals with the same data and code which enables a reproducible execution (Nekrutenko and Taylor, 2012) (FR8). An additional aspect that needs to be covered is the storage of intermediate and, if desired, final results of a workflow. Since an edge device is not as reliable as a server, we integrated a storage cluster for longterm persistence. Powerful and reliable devices can provide their free disk space by spawning a *Storage Handler* container and join a cluster of such *Storage Nodes* (see Fig. 1 right). The cluster is accessible via P2P and replicates the data between its nodes, resulting in a scalable and reliable storage system (NFR5). Regardless of the storage location, the files are always content-addressed to ensure reproducible actions later on.

4.5 Overall Picture

Now, we want to present how the individual components interact. At first, we have an actor who defines a *Workflow Description* that specifies which data is to be analyzed using which *Analysis Task* in which order. Furthermore, the actor can configure on which node the tasks will be performed, hence enabling local preprocessing of the data e.g. by specifying owned machines (FR7). By definition, this machine has to be a *Worker Node* to participate in the analytics network. Subsequent tasks could share this data using the P2P network. Then, the actor submits the *Workflow Definition* to the *Orchestration Component* to start the process. The component splits the workflow in individual tasks that are to be executed by *Worker Nodes*. The *Orchestration Component* triggers requirements matching *Worker Nodes* and tells them which data and which analysis image they have to utilize. After downloading the image and data, the node executes the task (FR1). Depending on the workflow, the result is either stored on the machine or replicated by the *Storage Cluster* provided by the P2P Network. While a workflow is processed, the *Orchestration Component* keeps track of all tasks with their generated output data. Since the data management uses content addressing for analysis code and data, these assets are always uniquely and reliably identified. In addition, tasks can be restarted at any time because of their idempotency. These features enable to generate a reproducible workflow by extending the original *Workflow Definition* with the content-addressed identifier of the intermediate and final results (FR8).

5 IMPLEMENTATION

In this section, we will describe what specific technologies we used to implement our conceptual framework. Our selection represents only one possible configuration. It is in the very nature of the conceptual framework to deal with other technologies as well. However, it is possible that when selecting other technologies, much more adaptation may be required to meet the concept. Where appropriate, we name alternative technologies and explain why we have chosen a particular one.

First, we need to decide for a container technology. In order to provide the same image for heterogeneous architectures, we make use of an image index. This index is a higher-level manifest containing a list of references pointing to an image manifest for a specific platform. Since this feature is included in the OCI specification[2], any compliant container engine is sufficient, for example *containerd*, *rkt*, or *Docker*. We chose the latter for building the images because of Docker's support for ARM-builds on x86-machines.

For the P2P network, we were looking for a content-addressed data exchange. The Interplanetary Filesystem (IPFS)[3] supports both of these properties and offers a variety of APIs to integrate it into a system. On each node in the network, we deployed an instance of an IPFS-powered component, which works as an adapter for serving and loading the data from the network. These instances are also containerized for an easy deployment. Other local containers can

[2]https://github.com/opencontainers/image-spec/blob/master/image-index.md

[3]https://ipfs.io/

retrieve data from the network by sending a HTTP request to this component, which fetches the desired data via IPFS. Since we want to access and store analysis images in the same way, we also implemented an image registry interface in the component. It is compliant with the OCI specification and translates regular calls to an image registry, e.g. downloading a single layer, to corresponding IPFS fetch requests. This enables the container engine to pull an image as usual while it gets served from the P2P network. A modification of the container client is not needed. Since both assets are provided by IPFS, we can use their content-addressed identifiers in the workflow definitions to ensure a very strong reproducibility. The storage cluster was build with the same technology. On reliable nodes with sufficient disk space, we deployed IPFS to form a cluster[4] for automatic replication and availability. There, intermediate and final results as well as analysis images can be stored and retrieved. While we chose IPFS in version 0.4.20 because of its low entry barrier and wide range of available APIs, other technologies like DAT[5] or Chord (Stoica et al., 2001) could also be great candidates in this case.

As described, the analytics network needs a way to orchestrate the containers on the various nodes in the network. This includes container placement capabilities and resource limitations when scheduling tasks as well as configuring new nodes and dealing with outages. Kubernetes[6] is a well-known orchestration tool for managing containerized applications as a cluster. It follows the client-server-architecture and manages a set of worker nodes by a master node. Workloads are described as declarative configurations and are read by so-called controllers, that watch the current cluster state and adjust it by taking actions to match the desired state. The basic scheduling unit for containers are so-called pods, which package one or more related containers. Typically, Kubernetes is used in cloud environments making use of powerful hardware and network. Since this is not guaranteed in our case, we aimed for a distribution designed for use in edge computing scenarios. K3S[7] was a perfect match for this as it includes core features only and packages them as a single binary, which speeds up the deployment and management process greatly. This way, each client can act as a Kubernetes node allowing us to make full use of Kubernetes orchestration features. By configuring resource limitations and placing labels on the nodes, the scheduling process can be adjusted to our needs. For instance, we could represent department boundaries within the nodes or mark nodes with extra powerful hardware to assign complex analysis tasks on these systems later on. Indeed, other famous container orchestration solutions like Apache Mesos[8] or Nomad[9] could be chosen, too. However, for our use case Kubernetes and K3S offered the best mix of a lightweight and powerful orchestration tool with a great engagement of developer communities and companies.

Since the resource orchestration and the workflow orchestration are deeply coupled, we need a container native solution for managing our workflows and distributing the tasks. In addition, we want to use our own data management system to provide P2P capabilities and enable local preprocessing before sending it to a storage, while still defined as a workflow. The requirement of a dynamic composition of custom workflows instead of pre-defined analytic pipelines lead us to Argo[10]. Workflows are defined as a DAG in form of a custom Kubernetes resource. Argo's own controllers then take care of instructing Kubernetes to create the pods according to the workflow. We extended the system in several places. First, we integrated our data management system so that content-addressed IDs can be specified for the images and data. Second, we customized Argo's submission of workflows to generate an extra definition of the workflow containing all the IDs of the images and the (intermediate) results. This workflow can finally be resubmitted to reproduce the original workflow as it relies on the same data. At last, we added support for the workflow engine to deal with a sudden outage of a node so that retry strategies can take place.

6 EVALUATION

We evaluated the system on a laboratory pilot by using a testworkflow for a text analysis, shown in Figure 2. Its objective is to extract the text from a local PDF document and process it with various analysis tasks, for instance keyword extraction and summary generation. In the last step, the generated analysis results are merged into a single result document. With this experimental setup we imitate a possible metadata extraction for a data catalog as described in Chapter 2. The tasks were implemented as multi-architecture images using Java, Node.js, and Python (NFR3) and dealt with different data formats (e.g. PDF, TXT, and JSON) (FR4). Indeed, other analytic workflows can

[4]https://cluster.ipfs.io
[5]https://www.datprotocol.com
[6]https://kubernetes.io
[7]https://k3s.io
[8]http://mesos.apache.org
[9]https://www.nomadproject.io
[10]https://argoproj.github.io

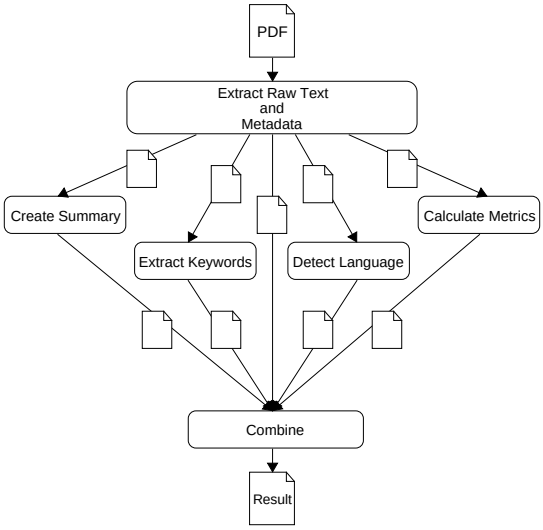

Figure 2: DAG of our evaluation workflow.

be implemented too, if they can be modeled as described in 4.1 (FR1, FR5). To demonstrate the local preprocessing capabilities, we enforced the execution of the initial extraction task of a document on its underlying node (FR7). As a testsystem, we built a small cluster, mainly consisting of Raspberry Pi Model 3B+. In addition, we included several x86-PCs and could demonstrate cross-architecture support (NFR2). However, for the actual performance evaluation, we restricted the execution to the Raspberry Pis with no resource limits (FR2). We chose these boards to represent the entire worker cluster to ensure same specifications and have reproducible results. It should be emphasized that an external storage was not required (NFR6), but could be integrated. Then, we submitted 60 instances of testworkflows and tracked the overall execution time for various worker cluster sizes (NFR2). In addition, we used a dynamic cluster that gains one worker every 15 minutes by starting Raspberry Pis gradually. The results are visualized in Figure 3 and show that the required time and cluster size are in reciprocal proportion (FR6, NFR4). It also reveals that the systems adopts new workers automatically and includes them in the scheduling process (FR3, FR6, NFR1). Furthermore, we tested whether the system is able to continue workflows when nodes randomly crash and stay offline. To simulate such scenario, we forced a shutdown on random worker node while executing a task of a workflow. With our extension, the workflow engine was capable of recognizing these situations and rescheduling the tasks to live worker nodes (NFR7). The last tested feature was reproducibility. First, we let the system generate a reproducible workflow definition of a testworkflow. Then, we submitted this definition once with modified and once with the original input data. The system behaved as expected, as the first submission failed while the last one succeeded and produced same results (FR8). While evaluating the prototype, we noticed that loading larger files ($\geq$ 1MB) via the P2P network from the IPFS cluster took an unexpected high amount of time. In fact, our tests revealed a decreased performance up to 75% compared to a direct download with HTTP. Such problems when reading large files from remote nodes in IPFS have also been reproduced in another work (Shen et al., 2019). As this is clearly an implementation problem from IPFS 0.4, it can maybe be overcome by updating to version 0.5 or by swapping IPFS in favor of DAT or other P2P frameworks utilizing content addressing. Therefore, fulfilling NFR5 in a bigger scale is a pending task.

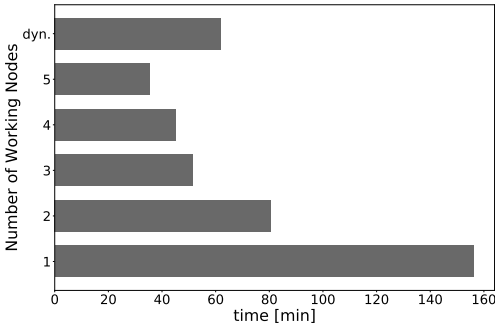

Figure 3: Worker Node scaling evaluation.

7 RELATED WORK

Based on a literature analysis, we found a number of related works in the academic world and industry that aimed to either provide a holistic analysis platform or focus on data analysis workflows with guarantees to certain execution properties. Due to the sheer amount of existing solutions, we have selected the most promising ones and classified them in regards to their analysis workflow properties. The resulting comparison is shown in Table 2. It should be noted, that a comparison between all those systems solely based on performance is not expedient due to the wide range of their differences and qualities. Even when using containerized tasks, it is difficult to establish an equal starting point because of different ways of distributing, used storage systems or processing model. Therefore we will only compare on the basis of the properties and features provided.

Especially in life sciences like bioinformatics and biomedicine, the ability to execute scalable, reproducible, and data-intensive workflows is an essential task (Nekrutenko and Taylor, 2012) (Novella et al.,

Table 2: Related work comparison.

	Nextflow	KNIME	SnakeMake	Galaxy	Airflow	Pachyderm	Data Civilizer	Kubeflow	Our Solution
Workflow editor	no	yes	no	yes	no	no	yes	no	no
Workflow model	DAG	DAG	DAG	DAG	DAG	DAG	DAG	DAG	DAG
Data sovereignty[a]	no	no	no	no	no	no[b]	no	no	yes
Distribution level	task	task	task	task	task	task	n.a.	task	task
Containerized tasks	yes	no[c]	yes	yes	yes	yes[d]	no	yes[d]	yes[d]
Language restriction	no	Java	no	no	no	no[e]	Python	no	no[e]
Reproducibility	medium[f]	medium[g]	strong[h]	strong[h]	weak	strong[h]	medium	strong[h]	very strong[i]
Automatic failover	yes	yes[j]	no	no	yes	yes	no	yes	yes
Execution on edge devices[k]	no	no	no	no	no	no	no	no	yes
Decentral distribution of code and data	no	no	no	no	no	no	no	no	yes

[a] Data can be preprocessed on their origin node without initial upload to a platform.
[b] Import to Pachyderm' storage required. [c] A KNIME instance is containerized, not a single task. [d] First-level support for containers. [e] Tasks must be defined as container images. [f] Hash is not based on content. [g] Not clearly documented how data is tracked. [h] Images are tracked by name and not by content. [i] Image *and* data are tracked by hash. [j] Only whole workflow. [k] System designed to be executed on a cluster of heterogenous (edge) devices.

2019). Therefore, various systems and platforms, such as Galaxy (Afgan et al., 2016), SnakeMake (Köster and Rahmann, 2012), Nextflow (Di Tommaso et al., 2017), and Pachyderm (Novella et al., 2019) were designed to aid researchers and ease analytic processes. All of these systems have in common that the workflow are modelled as DAGs by using a custom domain specific language (DSL). Galaxy is remarkable for a sophisticated graphical user interface to compose such workflows. While this component is clearly designed for scientific workflows, NextFlow offers a more general workflow editor. This is a feature our system is clearly missing but left out intentionally for the sake of this work's scope. In systems like Galaxy, SnakeMake, and NextFlow, the tasks can not only be executed locally but on a distributed cluster as well. It should be noted that a shared filesystem for all cluster nodes is required and is not provided by the platform itself. Such kind of storage can cause performance problems and the data must first be uploaded there. In contrast, Pachyderm and our system distribute their workload with Kubernetes, but also integrate their own data management solution. This component is responsible for the storage, provenance, and provisioning of data. While our data management is fully decentralized, Pachyderm relies on a centralized repository where data must be stored before starting a workflow. Our decentralized management allows devices to preprocess data locally before making it public to a distributed workflow. To our knowledge, this data sovereignty property is missing in other systems. An interesting feature of Pachyderm's repository is the splitting of incoming data into datums when initially loaded. In our concept, this functionality could be integrated in advance through a separate workflow for data splitting. However, it is not comparable because stream processing is deeply integrated into Pachyderm. This dataflow programming model is also used by NextFlow to assemble processes to a pipeline. Other systems like SnakeMake and our solution rely mainly on task parallelism. By utilizing containers, analysis code and its dependencies can be packaged in a lightweight and encapsulated way which is required for reproducible research (Novella et al., 2019). The container virtualization for tasks can be used in nearly all systems, but the integration varies greatly as these containers are often not a first-class citizen but a way to package scripts. In Pachyderm and our approach, running a container is the native and only way to perform a task, thus achieving a certain degree of encapsulation during the execution. Managing the input and output data is separated by the data management component so developers only need to implement data processing containers. While all those systems offer a certain grade of reproducibility, it differs in quality as truly reproducible workflows have to be executed with the same code and data. When the code is not directly provided as a script but encapsulated in an external container image, a system needs to ensure it is identical by its content and not by its tag. By using content addresses for images and data, our system tracks these assets in a reliable, uniform, and precise way.

More general data analytics platforms are Konstanz Information Miner (KNIME) by (Berthold et al., 2009) and Data Civilizer (Deng et al., 2017). Both systems are designed as a data management and workflow system to support data scientists in their daily work. By offering visualization, data cleaning and preprocessing, and analytics capabilities as well as workflow management (Rezig et al., 2019; Berthold et al., 2009), they cover a much broader range of features as our solution does. While custom analysis tasks can be created, KNIME prescribes its own

framework in Java, utilizing the Eclipse IDE environment that has to be learned and used. Inside a task, arbitrary commands can be executed, e.g. to create and run a container (Fillbrunn et al., 2017). With the newer versions of KNIME the workflows can also be distributed. The Data Civilizer relies on a table-in table-out interface while all other systems are less restricted as they deal with generic file types. In addition, data civilizer provides a GUI for editing workflows in conjunction with an I/O Specification for their workflow tasks. To our knowledge, a distributed and containerized execution is not covered. In contrast to KNIME and Data Civilizer, our approach does not include ready-to-use data wrangling and analytics tools. However, if such tools are packaged as container with a file-in file-out interface, they can be integrated seamlessly.

When dealing with workflow systems, Apache Airflow[11] is a commonly used candidate. As an orchestration tool, it only controls and schedules the actual tasks of a workflow on workers and does not include a data management system. Workflows represent a DAG and are described with a Python DSL. Therefore, it maps mostly to the workflow engine component in our system. In comparison to Argo, it does not integrate deeply into Kubernetes, but it offers to execute tasks as pods. It is also possible to run Airflow operators within an Argo workflow[12]. A system that even includes Argo as its workflow engine is Kubeflow. Kubeflow is a platform to perform and deploy machine learning workflows in a scalable and portable way on Kubernetes. Kubeflow Pipelines[13] is its submodule for orchestrating, running, and visualizing workflows. To our knowledge, there is no extra mechanism for reproducible workflows. Instead, the metadata of a workflow, e.g. used containers as well as input and output data, is recorded in a metadata database for later inspection of provenance. In contrast to our system, it relies on a central repository as storage.

An unique aspect of our systems seems to be the inclusion of edge devices for running analysis tasks and thus harvest existing idle resources. While other solutions may be able to integrate edge devices through additional effort and adjustments, this is already a fundamental part of our considerations.

[11]https://airflow.apache.org

[12]https://github.com/argoproj/data-pipeline

[13]https://www.kubeflow.org/docs/pipelines/overview/pipelines-overview

8 DISCUSSION & CONCLUSION

This paper provides a conceptual framework and an associated prototypical implementation of a flexible data analytics network. By including edge devices, we achieve local preprocessing and are able to use the power of existing idle resources. In order to make the analyses more valuable and trustworthy, they are implemented as reproducible workflows, as long as all tasks are idempotent. For instance, our solution can not guarantee a reproducible workflow if a task breaks this rule and does something like an anonymization that replaces data with random values. A possibility to overcome such issues is to use a separate prior workflow that is responsible for preprocessing. The resulting data can be used for a subsequent workflow, which in turn is reproducible again. The evaluation revealed that the requirements of building a flexible and reliable network for data analytics, which integrates edge resources dynamically, could be met. Since we could only test our system in a small scale, an evaluation in an enterprise network is a pending task we want to cover in follow-up work. Before doing so, however, it is necessary to check and solve the performance problems that were discovered when using IPFS. Another question that arises when inspecting the system's architecture is the limits of the centralized orchestration component. Although it can be operated in a high availability mode, it leaves a somewhat stale impression. While most of our other conceptual components are P2P driven and highly scalable, the orchestration component's scalability may become a limiting factor for the whole system. If feasible, we want to expand the concept in the future so that the nodes in the analysis network divide up workflows by consensus. Considering the current implementation, the development of a custom scheduler for Kubernetes could be a reasonable approach to cope with increasing numbers of worker nodes (Rausch et al., 2019).

A screening of related work revealed many good ideas, which are not present in our solution so far, but can be added at a later stage or by using specific workflow tasks. One interesting functionality is to split the data into smaller pieces at the beginning of an analysis to enable data parallel processing. For well-known file types specific preprocessing strategies could be provided. For instance, the rows of a CSV file could be initially extracted by a designated task. Another important feature, especially for end-users, would be a graphical editor to compose workflows. Currently, users are required to have deep knowledge about the available images and how they can be arranged in a meaningful way. By providing an intuitive graphical

user interface, the system can be used by a greater range of users with varying technical skills. Lastly, it should be noted that the usage of the network is not limited to analysis tasks. For instance, long-running services could be run while making use of the uniform deployment of software on diverse devices which the system provides. For instance, a workflow could deploy or update a service as the last task of an analysis.

ACKNOWLEDGEMENTS

This work was funded by the Fraunhofer-Cluster of Excellence »Cognitive Internet Technologies«.

REFERENCES

Afgan, E., Baker, D., van den Beek, M., Blankenberg, D. J., Bouvier, D., Cech, M., Chilton, J., Clements, D., Coraor, N., Eberhard, C., Grüning, B. A., Guerler, A., Hillman-Jackson, J., Kuster, G. V., Rasche, E., Soranzo, N., Turaga, N., Taylor, J., Nekrutenko, A., and Goecks, J. (2016). The galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update. *Nucleic acids research*, 44 W1:W3–W10.

Berthold, M. R., Cebron, N., Dill, F., Gabriel, T. R., Kötter, T., Meinl, T., Ohl, P., Thiel, K., and Wiswedel, B. (2009). Knime-the konstanz information miner: version 2.0 and beyond. *AcM SIGKDD explorations Newsletter*, 11(1):26–31.

Buyya, R., Srirama, S. N., Casale, G., Calheiros, R., Simmhan, Y., Varghese, B., Gelenbe, E., Javadi, B., Vaquero, L. M., Netto, M. A., et al. (2018). A manifesto for future generation cloud computing: Research directions for the next decade. *ACM Computing Surveys (CSUR)*, 51(5):105.

Chen, M., Mao, S., and Liu, Y. (2014). Big data: A survey. *Mobile networks and applications*, 19(2):171–209.

Cudahy, G., Flynn, C., Liu, J., Padmos, D., and Wanger, G. (2016). Digital supply chain: it's all about that data.

De Filippi, P. and McCarthy, S. (2012). Cloud computing: Centralization and data sovereignty. *European Journal of Law and Technology*, 3(2).

Deelman, E. and Chervenak, A. (2008). Data management challenges of data-intensive scientific workflows. In *2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 687–692.

Deng, D., Fernandez, R. C., Abedjan, Z., Wang, S., Stonebraker, M., Elmagarmid, A. K., Ilyas, I. F., Madden, S., Ouzzani, M., and Tang, N. (2017). The data civilizer system. In *Cidr*.

Di Tommaso, P., Chatzou, M., Floden, E. W., Barja, P. P., Palumbo, E., and Notredame, C. (2017). Nextflow enables reproducible computational workflows. *Nature biotechnology*, 35(4):316–319.

Fillbrunn, A., Dietz, C., Pfeuffer, J., Rahn, R., Landrum, G. A., and Berthold, M. R. (2017). Knime for reproducible cross-domain analysis of life science data. *Journal of Biotechnology*, 261:149 – 156. Bioinformatics Solutions for Big Data Analysis in Life Sciences presented by the German Network for Bioinformatics Infrastructure.

Gandomi, A. and Haider, M. (2015). Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management*, 35(2):137–144.

Hoque, S., de Brito, M. S., Willner, A., Keil, O., and Magedanz, T. (2017). Towards container orchestration in fog computing infrastructures. *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, 02:294–299.

Köster, J. and Rahmann, S. (2012). Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522.

Liu, J., Pacitti, E., Valduriez, P., and Mattoso, M. (2015). A survey of data-intensive scientific workflow management. *Journal of Grid Computing*, 13(4):457–493.

Miloslavskaya, N. and Tolstoy, A. (2016). Big Data, Fast Data and Data Lake Concepts. *Procedia - Procedia Computer Science*, 88:300–305.

Morabito, R., Kjällman, J., and Komu, M. (2015). Hypervisors vs. lightweight virtualization: A performance comparison. In *2015 IEEE International Conference on Cloud Engineering*, pages 386–393.

Nekrutenko, A. and Taylor, J. (2012). Next-generation sequencing data interpretation: Enhancing reproducibility and accessibility. *Nature reviews. Genetics*, 13:667–72.

NISO Press (2004). Understanding metadata. *National Information Standards*, 20.

Novella, J. A., Emami Khoonsari, P., Herman, S., Whitenack, D., Capuccini, M., Burman, J., Kultima, K., and Spjuth, O. (2019). Container-based bioinformatics with pachyderm. *Bioinformatics*, 35(5):839–846.

Otto, B. and Österle, H. (2016). *Corporate Data Quality*. Springer.

Rausch, T., Hummer, W., Muthusamy, V., Rashed, A., and Dustdar, S. (2019). Towards a serverless platform for edge AI. In *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*, Renton, WA. USENIX Association.

Rezig, E. K., Cao, L., Stonebraker, M., Simonini, G., Tao, W., Madden, S., Ouzzani, M., Tang, N., and Elmagarmid, A. K. (2019). Data civilizer 2.0: A holistic framework for data preparation and analytics. *Proc. VLDB Endow.*, 12(12):1954–1957.

Shen, J., Li, Y., Zhou, Y., and Wang, X. (2019). Understanding i/o performance of ipfs storage: A client's perspective. In *Proceedings of the International Symposium on Quality of Service*, IWQoS '19, New York, NY, USA. Association for Computing Machinery.

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160.