Time-critical data management in clouds: challenges and a Dynamic Real-time Infrastructure Planner (DRIP) solution


DOI
10.1002/cpe.5269

Publication date
2020

Document Version
Final published version

Published in
Concurrency and Computation: Practice and Experience

License
CC BY

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.

UvA-DARE is a service provided by the library of the University of Amsterdam (https://dare.uva.nl)
Time-critical data management in clouds: Challenges and a Dynamic Real-Time Infrastructure Planner (DRIP) solution

Spiros Koulouzis1 | Paul Martin1 | Huan Zhou1 | Yang Hu1 | Junchao Wang1 | Thierry Carval2 | Baptiste Grenier3 | Jani Heikkinen4 | Cees de Laat1 | Zhiming Zhao1

1 Institute for Informatics, University of Amsterdam, Amsterdam, The Netherlands
2 IFREMER, Centre de Brest France, France
3 EGI Foundation, Amsterdam, The Netherlands
4 CSC IT Center for Science Ltd, Espoo, Finland

Correspondence
Zhiming Zhao, Institute for Informatics, University of Amsterdam, Science Park 904, 1098 XH Amsterdam, The Netherlands.
Email: z.zhao@uva.nl

Funding information
European Union’s Horizon 2020 Research and Innovation Programme, Grant/Award Number: 654182 (ENVRIPLUS project), 676247 (VRE4EIC project), 643963 (SWITCH project) and 654142 (EGI-Engage project)

Summary
The increasing volume of data being produced, curated, and made available by research infrastructures in the environmental science domain require services that are able to optimize the delivery and staging of data for researchers and other users of scientific data. Specialized data services for managing data life cycle, for creating and delivering data products, and for customized data processing and analysis all play a crucial role in how these research infrastructures serve their communities, and many of these activities are time-critical—needing to be carried out frequently within specific time windows. We describe our experiences identifying the time-critical requirements of environmental scientists making use of computational research support environments. We present a microservice-based infrastructure optimization suite, the Dynamic Real-Time Infrastructure Planner, used for constructing virtual infrastructures for research applications on demand. We provide a case study whereby our suite is used to optimize runtime service quality for a data subscription service provided by the Euro-Argo using EGI Federated Cloud and EUDAT’s B2SAFE services, and to consider how such a case study relates to other application scenarios.

KEYWORDS
Dynamic Real-Time Infrastructure Planner, e-infrastructure, research infrastructure, time-critical services, virtual research environment

1 INTRODUCTION

Data-centric approaches1 play an increasingly important role in scientific investigations such as climate modeling and the study of global warming in the environmental and earth science domains.2 Large-scale observations over extended periods of time are necessary for constructing and validating models of the environment and generating forecasts; therefore, it is necessary to provide sufficiently advanced computational networked infrastructure to manage both the transportation of large (distributed) data sets and the data-intensive processing of such data sets.

However, to make optimal use of the available infrastructures, researchers also need access to sophisticated research support environments that enable efficient discovery of the data, tools, etc, available for advanced data science and provide a platform for the integration of all resources into cohesive experimental investigations with replicable workflows. Examining current initiatives in Europe and beyond, three main types of research support environment can be identified:

E-infrastructures are Unified computing, storage, and network infrastructures provided via initiatives such as EGI* and EUDAT†.

e-Infrastructures focus on the management of the service life cycle of computing, storage, and network resources, and provide services to

The first five authors equally contributed to the paper.

* http://www.egi.eu/
† http://www.eudat.eu/

This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2019 The Authors Concurrency and Computation: Practice and Experience Published by John Wiley & Sons Ltd.

research communities or other user groups to provision dedicated infrastructure and to manage persistent services and their underlying storage, data processing, and networking requirements. Public e-infrastructures typically offer their services based on service-level agreements (SLAs) established at an institutional level or negotiated with specific groups. Such services are now predominantly cloud-based, using virtual machines (VMs) or containers that can be easily migrated and scaled across clusters of generic hardware.

**Research infrastructures (RIs)** are dedicated data infrastructure constructed by specific scientific communities for combining scientific data collections with integrated services for accessing, searching, and processing research data within specific scientific domains; examples include the Integrated Carbon Observation System\(^1\) for carbon monitoring in atmosphere, ecosystems, and marine environments; the European Plate Observing System\(^2\) for solid earth science; and Euro-Argo\(^3\) for collecting environmental observations from large-scale deployments of robotic floats in the world’s oceans. RIs play a key role in the *research data life cycle*, providing standard policies, protocols, and best practices for the acquisition, curation, publication, processing, and further usage of research data and other assets such as tools and simulation/modeling platforms. They typically work closely with (or effectively subsume) individual data centers dedicated to research data, sensor networks, laboratories, and experimental sites.

**Virtual research environments (VREs)** are platforms providing user-centric support for discovering and selecting data and software services from different sources, and composing and executing application workflows.\(^3\) They are also referred to as virtual laboratories\(^4\) or science gateways.\(^5\) VREs play a direct role in the *activity life cycle* of research activities performed by scientists, for example, the planning of experiments, search and discovery of resources from different sources (notably including RIs), integration of services into cohesive workflows, and collaboration with other scientists. Graphical environments, workflow management systems, and data analytics tools are typical components of such environments.

While the roles and functions of these different kinds of environments may substantially overlap, none individually fulfill all the requirements of data-centric research; in practice, all these types of research support environment must be tightly integrated (and their overlapping functions reconciled and duly delegated). In particular, e-infrastructures focus on generic ICT resources (e.g., computing or networking), RIs manage data and services focused on specific scientific domains, and VREs support the life cycle of specific research activities. Although, as already noted, the boundaries between these environments are not always entirely clear (often sharing services for infrastructure and data management\(^6\)), collectively, they represent an important trend in many international research and development projects. Figure 1 shows the abstract logical relationship between e-infrastructures, RIs, and VREs.

Performance is a crucial factor for many scenarios involving research support environments, influencing the quality of experience factors such as responsiveness to requests, to more system-level concerns such as efficient load distribution across distributed nodes in a confederation of data services. An example of a performance-critical system involving environmental data would be an early warning system where real-time sensor data have to be analyzed quickly enough to identify events and provide adequate time for response. Even in nonemergency contexts, there are many cases where RIs collect real-time data from sensors continuously for swift processing to provide ‘nearly real-time’ services to

---

\(^{1}\)https://www.icos-ri.eu/
\(^{2}\)https://www.epos-ip.org/
\(^{3}\)http://www.euro-argo.eu/
researchers—the specific example used in this paper is that of a data subscription service whereby updates to tailored subsets of a data set are pushed to subscribers within a requested deadline. Notably, these services often cut across research support environments; RIs provide the service but delegate the hosting and management of the data processing pipeline to an e-infrastructure, generally to take advantage of elastic infrastructure resources rather than provide dedicated infrastructure within their own data centers (which often operate as loose confederations with limited budgets for services beyond data curation and publication). VREs may also be involved as part of the interface with researchers, eg, to subscribe to RI services or retrieve (and process) the results from such services.

To deliver acceptable performance, time-critical applications thus rely not only on the infrastructure for parallel computing or fast communication between components but also on optimization of system-level application behavior. The customization of the infrastructure must consider performance constraints on applications at runtime as well as the utilization and cost of the underlying resources across applications.

In this paper, we present a smart infrastructure optimization engine, called the Dynamic Real-time Infrastructure Planner (DRIP), that has been developed to bridge the gap between application requirements and service delivery on the part of research support environments, to optimize the quality of service (QoS) at all levels. DRIP can be used to deploy, control, and manage the kinds of distributed data pipelines needed for advanced RI services on the kind of virtualized cloud-based infrastructures now being provided by e-infrastructures. A use case from the Euro-Argo RI will be used to demonstrate how DRIP can automatically select and provision virtual resources, deploy services, and optimize the delivery of observation data to a number of destinations in the cloud.

In the following section, we examine the scientific and technical background of this work in more detail. We describe our implemented system in Section 3 and our specific use case in more detail in Section 4. We evaluate our experiments in Section 5 and finally discuss our conclusions in Section 6.

2 Requirements and State of the Art

In this section, we analyze the basic requirements for service performance optimization for time-critical data services in research support environments, review the state of the art in real-time systems that may bear an impact on the development or operation of such data services, and summarize the essential challenges for time-critical data services on modern e-infrastructures.

2.1 Requirements

When we refer to time-critical applications, we do not usually mean speed-critical applications in the sense of applications that simply need to minimize the completion time (ie, must be run fast). True ‘real-time’ applications are characterized by bounded response time constraints on inputs, with certain consequences upon failure to meet deadlines. Based on those consequences, real-time applications are referred to as hard real time if any missed deadline leads to immediate failure of the application, soft real time if missing deadlines merely leads to degradation of user experience, and firm real time if failure is brought about by too many missed deadlines in succession. Nearly real-time applications meanwhile are those with an intrinsic yet bounded delay introduced by data processing or transmission. Note that this does not make all nearly real-time applications ‘soft’—such applications can still impose a hard requirement for processing to fall within the permitted bounds.

We might consider most processes in research support environments to be soft in so much as failure to meet deadlines is usually not immediately disastrous. However, processes that are continuously run in tandem with real-time data acquisition can be seen to be ‘firm’ due to the cascading impact of repeated failure to process their inputs on time; similarly, any highly parallelized workflow with bottlenecks in the data pipeline can suffer a precipitous drop in general QoS if delays in one parallel element impact a nonparallelized bottleneck downstream. When we refer to time-critical applications, we therefore generally mean real-time or nearly real-time applications that are ‘firm’ (or harder) in terms of the consequences of failure to meet the QoS requirements. True hard real-time constraints in research infrastructure are rare but may emerge in particular for safety-critical applications that depend on real-time observational data.

In practical terms, the ‘firmness’ of a response time constraint dictates the degree of limited resources that should be allocated to ensuring the constraint. Isolated failures do not have the same impact as failures that beget further failures. It may be possible (and desirable) in specific research support environments to be able to assign a metric to constraints based on firmness that can be translated into concrete resource-level requirements or adaptation strategies, so that this information can be passed to optimization services that must prioritize particular services or metrics.

The requirements for optimizing performance in research support environments are mainly dominated by the requirements of the data-centric research activities (the simplest but most important being the retrieval of specific data sets on request) that demand high performance or responsiveness. Within RIs, services are often developed with time constraints imposed on the acquisition, processing, and publishing of real-time observations and in scenarios such as disaster early warning. For VREs and RIs, performance factors are strongly influenced by the time needed to customize the runtime environment and to schedule the workflow applications. Steering of applications during complex experiments is also temporally bounded. Computing tasks and services provided by e-infrastructures are managed and offered to clients based on SLAs. Time constraints are also imposed on the scheduling and execution of tasks that require high-performance or high-throughput computing. The overhead introduced by the customization, reservation, and provisioning of suitable infrastructure, the monitoring of runtime behavior for
infrastructure, and the support for runtime control also needs to be reduced and maintained within minimum levels. Failure recovery for deployed services and applications in real time is also important when supporting time-critical applications; time constraints are not only imposed on failure detection but also on decision making and recovery.

2.2 State of the art

The fulfillment of most time-critical requirements for research support environments relies on optimal execution of tasks on e-infrastructures, as well as the efficient movement of data across networks. We identify several categories of time-critical applications.

**Time-critical information search and query.** Typical technologies for real-time data query and search model the search activities of users and their projected needs, for predicting future queries, optimizing catalogues, or prioritizing urgent tasks, as well as optimizing the presentation of contextual information. Information retrieval is a core part of many data management services and may require the retrieval of multiple data sets to answer a given query or considerable internal processing of data files for document-oriented data.

**Time-critical workflow execution.** Time-critical constraints on workflows are typically expressed as deadlines for completing (part of) the workflow or for responding to invocations or events within a certain time window. Scheduling the execution of such workflows requires consideration of not only individual task deadlines but also cost and occupation of resources. Algorithms based on partial critical paths can be used to solve such problems, applying metaheuristic approaches, eg, particle swarm optimization. When customizing virtual infrastructures, a common approach will (1) select suitable VMs based on certain task-VM performance matrices, (2) minimize communication costs between tasks by grouping tasks needing frequent communication in the same VM, and (3) refine the selection based on the calculation of new critical paths. Most existing work focuses on guaranteeing a single deadline encompassing the entire application, eg, the critical path-based iterative algorithm and the complete critical path algorithm. All these technologies have been widely investigated for applications modeled as directed acyclic graphs (DAGs) as DAG-based methods are popular for building data flows for data-intensive applications.

**Real-time modeling and simulation.** In data science, coupling different simulation models of individual systems can be performed to study the behaviors of complex systems, eg, combining species distribution models with weather models to study how diseases are distributed via insects and species migration at different times. Simulating physical systems does not always require the simulation to run at wall clock rates, but executing such simulations on distributed infrastructure does impose requirements on managing the simulation times of different subcomponents, eg, to control the causal relationships among events and time.

**Real-time computational steering.** Real-time steering of a computing system requires monitoring of the runtime status of both application and infrastructure. Infrastructure-level monitoring takes place at the network level and on computing and storage nodes. Monitoring service quality of cloud environments allows providers or users to evaluate compliance with SLAs. At the application level, monitoring often requires embedded probes within application components. Logging and provenance subsystems often capture the runtime status of the overall system as well. To visualize the runtime status and to allow a user to make correct decisions regarding system control, different kinds of monitoring information together with the context of the system execution have to be harmonized based on the timestamp. Semantic technologies are often used to integrate such information and to offer query interfaces to link them. Runtime steering of computing systems can take the form of adaptations of application logic at certain control points where the system actively provides time windows for users to intercede, or else, the system can be interrupted by the user during execution. The controllability of infrastructures, eg, dynamically configuring or scaling nodes, or controlling network flows, offer applications opportunities to refine the system performance.

**Real-time data acquisition.** Acquiring real-time observations is important for many RIs. The quality of communication between sensors and data processing units is crucial for timely acquisition. Software-defined sensor networks can be used to optimize communication between sensors as can applying edge computing solutions to tightly couple sensors with data processing. To make sensor data available to users in near real time, partially automating data quality control and annotation is important, but currently, most data quality control is performed manually. Standardizing this process and exploiting scalable virtualized infrastructure are recurring requirements for environmental RI.

**Real-time data transfer.** Real-time data transfer between components occurs frequently within e-infrastructures. At the network level, protocols are used to optimize data streaming throughput. Software-defined network (SDN) technologies are used to dynamically adapt network flows between data nodes, and traffic programming models such as coflow are used to reschedule runtime data transfer. At the transfer service level, dynamic schedule data transfer workers are used in the LBOCDER service to handle the balance of downloads.

**Infrastructure provisioning for time-critical applications.** Fast provisioning of virtualized infrastructure opens the possibility of runtime adaptation to meet time-critical requirements. Optimizing VM image size or directly forking runtime images from memory can reduce provisioning time. Using peer-to-peer or SDN technology to optimize image transfer among data centers is also possible. The
work of Zhou et al describes a transparent networked virtual infrastructure graph partitioning and parallel provisioning approach to map infrastructure across data centers.

**Real-time SLA.** Real-time support of virtualized infrastructure has attracted increasing interest. SLAs for real-time applications and their negotiation at runtime will be crucial for supporting real-time applications in clouds. Most approaches are based on graph mapping using key quality parameters such as execution time; improving the mapping procedure can be done by parallelizing the search procedure for matching resources and applications, preprocessing the resource information by clustering the resource information based on the SLA request, and multiobjective optimization for searching out alternative solutions.

### 2.3 Challenges for time-critical applications on e-infrastructure

In data science, the research data life cycle is considered to be of primary importance, but at each stage of that life cycle, we must also consider the life cycle of the data pipelines or data processing workflows that are needed to support each stage. Given the increasing availability and adoption of virtualized e-infrastructure and cloud services targeted towards RIs and the general research community, we are particularly interested in the life cycle for applications on virtual infrastructure (ie, configurations of networked VMs upon which data processing workflows are deployed on behalf of researchers either for specialized tasks or as part of the general data life cycle managed by RIs).

For static infrastructures, the development and configuration of a particular application (eg, a data processing pipeline or workflow) can be adapted with respect to the hardware and host architecture known to the developers. This may still require considerable technical expertise of course but can be nonetheless considered to at least represent a single initial investment to providing an efficient, performing technical solution.

In contrast, deploying application workflows on virtual infrastructures allows RIs to make use of commodity e-infrastructure resources as and when needed, rather than requiring an investment in dedicated hardware, and in principle, it offers the additional advantages of scalability and seamless migration that can to some extent be managed almost entirely by the e-infrastructure provider. It is difficult however to optimize generic virtual infrastructure for specific applications and therefore difficult to guarantee at a certain level of performance, which is a particular concern for time-critical applications.

Figure 2 illustrates the life cycle of application workflows on virtual infrastructure. Five main phases are identified:

**Virtual infrastructure planning.** Regarding the scheduling of application workflows onto a topology of VMs that ensures the availability and suitability of virtual resources at all stages of the workflow.

**Virtual infrastructure optimization.** Regarding the iterative refinement of an infrastructure plan to meet all (or a maximal subset of) requirements for performance, reliability, QoS, etc.

**Virtual infrastructure provisioning.** Regarding the actual provisioning of a planned infrastructure across one or more data centers or clouds in such a way as to create a network of resources that meet the control and data flow requirements of a distributed application.

**Software platform deployment.** Regarding the actual deployment of application elements onto the provisioned infrastructure, as well as the initialization and control of such elements at runtime.

![FIGURE 2](image.png) **The life cycle of application workflows on virtual infrastructure**
**Application monitoring and adaptation.** Regarding the monitoring of a running application with respect to selected metrics necessary for evaluating the performance and liveness of the application, as well as the ability to intelligently take measures to improve and regain a desired quality of service, eg, by automatically scaling or migrating application elements in the virtual infrastructure or reconfiguring components where practical.

While there exist a number of general solutions for managing each of these phases for the most common technologies for providing virtual infrastructure, or even subsets thereof, there is no single integrated solution for managing the entire life cycle just described. Moreover, if we want to apply such a solution to time-critical applications, then it is necessary to address additional challenges:

- To meet time requirements for discovering and retrieving data from distributed access/storage services and virtual infrastructures provided by different RIs, it is necessary to be able to define deadlines throughout the application deployment life cycle both individually and collectively.
- To develop a time-critical application, either the developer needs to be able to describe how constraints at the application level propagate down to the level of restrictions on infrastructure and QoS or the optimization services developed for the infrastructure must be able to do that for the developer.
- During the execution of time-critical applications, data sources, software components, and the execution engines of some parts of the application may have to be handled by different underlying infrastructures, making it difficult to calculate and enforce QoS constraints across the entire application/infrastructure stack.

To help address these concerns, we have developed a set of autonomous services that assist with the optimization of runtime services from infrastructure customization and provisioning to runtime control. Motivated by the requirements of time-critical data processing for scientific applications in particular, the DRIP, an ongoing development intended to eventually address all five phases described above with an emphasis on providing time-critical QoS, can now be introduced.

### 2.4 Related work

Within the cloud context, many approaches have been proposed to address the scheduling, scaling, and execution of tasks with deadlines. The majority of these proposals however adopt the viewpoint of the cloud provider which is often concerned with optimal VM placement on physical machines. In other cases, the complex scheduling algorithms consider only the planning phase and do not react at runtime to changes of performance or failures. Moreover, the majority of these approaches consider synthetic tasks and workloads, simulated cloud environments, or both.

### 3 DYNAMIC REAL-TIME INFRASTRUCTURE PLANNER

The DRIP is a service suite for planning and provisioning networks of VMs and then deploying distributed applications across those networks, managing the virtual infrastructure during runtime based on certain time-critical constraints defined with the application workflow. The DRIP service provides an engine for automating all these procedures by making use of pluggable microservices for providing specific functionalities orchestrated via a single manager component behind a RESTful Web API for easy use and retrieval of results. This approach enables a more holistic approach to the optimization of resources and meeting application-level constraints such as deadlines or SLAs. It also allows application developers to seamlessly plan a customized virtual infrastructure based on constraints on QoS, time-critical constraints, or constraints on budget. Based on such a plan, DRIP can provision a virtual infrastructure across several cloud providers and then be used for deploying application components, starting execution on demand, and managing the runtime application deployment state. Therefore, DRIP is not bound to any particular application. Instead, it is flexible and can deploy a wide range of applications on top of a customized and heterogeneous virtual infrastructure composed by multiple cloud providers to meet the application’s constraints.

#### 3.1 Architecture and functional components

The DRIP services include a number of components, interacting via an internal message brokering service orchestrated by a single manager. These components and their interaction are shown in Figure 3. All components of DRIP under the control of the DRIP manager are designed to be independently replaceable to allow for improved or alternative implementations of, for example, the planner or the provisioner. Indeed, multiple versions of a component could coexist, allowing for greater flexibility or even simply to better balance the load of requests to DRIP. The types of service that can be included in DRIP, as well as their current implementations, are now detailed:

**The DRIP manager.** The manager is a web service that allows DRIP functions to be invoked by external clients. Each request is directed to the appropriate component by the manager, which coordinates the individual components and scales them up if necessary. Resource
FIGURE 3  How services provided by the Dynamic Real-Time Infrastructure Planner are invoked by research infrastructure services to provide downstream services to users

information, credentials, performance profiles, and application workflows used by the manager and other components are all internally managed via the knowledge base as described below.

**Infrastructure planner.** The current planner uses a partial critical path algorithm\(^1\)\(^8\) optimized for workflows with multiple internal deadlines to produce efficient infrastructure topologies, selecting the most cost-effective VMs.\(^1\)\(^9\) Multiple planner components can be attached to DRIP to manage different kinds of application workflow or infrastructure topology, taking of different technologies such as SDN\(^5\)\(^1\) to customize the network topology among VMs and to optimally place network controllers for the networked VMs.\(^5\)\(^2\)

**Infrastructure provisioner.** The provisioner is responsible for automating the provisioning of infrastructure plans produced by the planner(s) onto the underlying cloud or e-infrastructure. The current provisioner can decompose the infrastructure description and provision it across multiple data centers (possibly from different providers) with transparent network configuration.\(^4\)\(^4\)

**Deployment agent.** The deployment agent installs application components onto provisioned infrastructure. The current deployment agent is able to schedule the deployment sequence based on network bottlenecks and to maximize the fulfillment of deployment deadlines for all the cloud providers currently supported by the default DRIP provisioner.\(^5\)\(^3\)

**Infrastructure control agents.** These agents provide sets of APIs that DRIP can then provide to applications to control the scaling of containers or VMs and to adapt network flows or to use itself in conjunction with a monitoring framework to automatically maintain the QoS of the deployed application.

**A knowledge base.** To allow different service components to operate effectively, DRIP provides a simple knowledge base for storing information about user credentials, the types of resource offered by clouds or e-infrastructure, and other useful data that the DRIP manager or any other component can retrieve or contribute to.

The prototype of DRIP adopts industrial and community standards. The infrastructure planner uses the Topology and Orchestration Specification for Cloud Applications (TOSCA) specification\(^8\) to get descriptions of applications and their constraints. The infrastructure provisioner uses the Open Cloud Computing Interface as its default provisioning interface and currently supports the Amazon EC2\(^1\)\(^1\), EGI FedCloud**\(^2\) and ExoGeni†† Clouds. Since DRIP relies on multiple cloud providers, it offers a best-effort approach for the provision, stability, and performance of the underlying virtual infrastructure. However, using performance and reliability models for each provider and each region DRIP is able to provide an optimal, stable, and responsive vitalized infrastructure for time-critical applications.\(^5\)\(^4\) The deployment agent can deploy overlay Docker clusters such as Docker Swarm or Kubernetes. It may also deploy any type of customized applications based on Ansible playbooks‡‡. The infrastructure control agents are a set of APIs that DRIP provides to applications to allow for scaling of containers or VMs and for adapting network flows.

---

\(^8\) [https://www.oasis-open.org/committees/tosca/](https://www.oasis-open.org/committees/tosca/)

\(^1\) [https://aws.amazon.com/cn/ec2](https://aws.amazon.com/cn/ec2)

\(^2\) [https://www.egi.eu/federation/egi-federated-cloud/](https://www.egi.eu/federation/egi-federated-cloud/)

\(^1\) [http://www.exogeni.net/](http://www.exogeni.net/)

\(^2\) [https://www.ansible.com/](https://www.ansible.com/)
manager provides a RESTful interface to allow integrated interaction with all components and uses RabbitMQ as its internal message broker to
direct requests appropriately. All DRIP software is available via https://github.com/QCAPI-DRIP/ under the Apache-2.0 license.

3.2 | How DRIP works

DRIP was developed in the context of EU H2020 projects SWITCH§§ (as part of a software workbench for time-critical, self-adaptive cloud
applications) and ENVRIPLUS¶¶ (to provide e-infrastructure optimization services for big data and scientific workflow applications). To function,
DRIP requires:

- An application description from the developer, identifying the specific components to be deployed on the provisioned infrastructure and
defining the dependencies between components that describe the application workflow and the time-critical constraints that apply to it.
- Information about the infrastructure resources (eg, VM types and network bandwidth) obtained from the cloud providers and their performance
hosting–specific applications (eg, provided by a cloud discovery and profiling service).

The application topology is currently described using TOSCA and must be part of the request made to DRIP. When a planning request comes,
the manager will direct the request to the infrastructure planner to generate a plan, which can be sent back to the user for further confirmation.
If the constraints cannot be satisfied, the planner informs the user that a plan cannot be generated. The DRIP manager stores necessary cloud
credentials on behalf of the user. The provisioning agent can provision the virtual infrastructure via interfaces offered by the cloud providers. Once
this has finished, the deployment agent will deploy all necessary components onto the provisioned infrastructure from designated repositories
and set up the control interfaces needed for runtime control of application and infrastructure.

4 | A CASE STUDY IN EURO-ARGO RESEARCH INFRASTRUCTURE

Environmental and earth science RIs support their target communities by acting as data hubs and publishers of scientific data. The Euro-Argo RI is
a typical example, being the European contribution to the Argo programme##. Argo monitors the world’s oceans measuring temperature, salinity,
pressure, etc, via the distributed deployment of robotic floats to create a roughly even network of data collecting nodes across the marine surface
of the Earth. These floats periodically send data back via satellite to data assembly centers, which provide integrated, cleaned data products to
various regional centers, archives, and research teams; all data are then made publicly available via a common portal within 24 hours of acquisition.

Euro-Argo is now prototyping a subscription service for their data. In contrast to simply providing collected data freely for download but requiring
researchers to manually monitor the core Argo data set for updates, researchers are instead allowed to subscribe to specific subsets of Argo data
and have updates pushed to their own cloud storage, thus streamlining data delivery and accelerating data science workflows involving those data.

4.1 | A data subscription service

In the data subscription service being now developed in Euro-Argo, investigators subscribe to customized views on the Argo data using a
dedicated data subscription service, selecting specific regions and time-spans, and choosing the frequency of updates. Tailored updates are then
provided on schedule to investigators’ private storage; Euro-Argo provides the infrastructure services needed for computing data products to
match each subscription and then dispatches those products to their destinations. A functional depiction of the Euro-Argo data subscription
service is illustrated by Figure 4, with the details of the implementation architecture reserved for the next section.

Data subscription is a good example of the kind of scalable, customized data service that RIs are now looking at as a way to better serve their
communities, which requires some degree of optimization of the underlying distributed infrastructure. It is also an example of a time-critical
service; subscriptions should be fulfilled on a schedule (possibly mixed for different products, leading to sporadic peaks of activity as schedules
for different products coincide), but different products may require different degrees of processing at different times and place differing levels
of load on the network to deliver to subscribers. A typical subscription task is made up of a set of inputs:

1. An area expressed as a bounding box (geospatial data being very common in environmental and earth science).
2. A time range (typically investigators will want the most recent data, but updates to past readings due to quality control or restoration of
missing data may also be of interest).
3. A list of parameters required in the data products (eg, temperature or salinity; in advanced cases, this may be a derivative parameter which
must itself be computed from some base parameters).
4. Optionally, a deadline (otherwise, a standard update schedule will be applied; deadlines might be expressed in terms of time since the last
update or simply be a regular recurring window for delivery).

§§www.switchproject.eu
¶¶www.envriplus.eu
##http://www.argo.ucsd.edu/
FIGURE 4 The data subscription scenario is one where researchers can subscribe to the specific data they are interested in (e.g., marine data from floats in the Mediterranean) via a simple portal and have all updates pushed to their workspaces periodically.

Such a data subscription service serves both end users and application workflows. Often these workflows require specific data to be delivered within a specific time window and often have a firm or soft real-time requirements. The type of real-time requirement is specified by the developer.

As the volume of subscriptions and the customizability of subscriptions increase, so does the pressure on the underlying infrastructure providing the data, the bandwidth for transport, and the processing. At the same time however, there will be periods of low activity between rounds of updates. Thus, we need a scalable infrastructure to support the data subscription processing pipeline, so as to not unnecessarily tie up resources while still permitting acceptable QoS during peak periods.

Beyond the parameters of this particular scenario, we also should consider how we manage the e-infrastructure used by RIs where there may be multiple data subscription pipelines for different kinds of data, or indeed how to easily configure and deploy new pipelines should other RIs want to replicate the Euro-Argo system for their own scientific data sets.

4.2 Service prototype

We prototyped the data subscription service based on the architecture depicted in Figure 5. Currently, the resources from e-infrastructures such as EUDAT and EGI Federated Cloud (FedCloud) are used. Figure 5 shows the use-case scenario based on the use of EUDAT and EGI services. In
this case, EUDAT provides services for storage and data transfer, while EGI FedCloud provides the services for the computing of data products for each subscription.

The data subscription service scenario thus involves the following basic components:

1. A data selection portal serving as the front end.
2. The global data assembly center of Euro-Argo providing the source research data set.
3. B2SAFE storage provided by EUDAT.
4. A deployment of DRIP (deployed within EGI FedCloud).
5. EGI FedCloud virtual resources, forming the fundamental infrastructure for data processing and transportation.
6. The subscription service (which maintains the subscriptions defined via the data selection portal).

Users interact with the subscription service via a portal, registering to receive updates for specific areas and time ranges for selected parameters such as temperature, salinity, and oxygen levels. The global data assembly center (GDAC) of Euro-Argo receives new data sets from regional centers and pushes them to the B2SAFE service. The subscription service itself maintains records of subscriptions, including selected parameters and associated actions. The role of DRIP then is to plan and provision customized infrastructure dynamically with demand and to deploy, scale, and control the data filtering application to be hosted on that infrastructure. EGI FedCloud provides actual cloud resources provisioned by DRIP.

The application itself is composed of a master node and a set of worker nodes. The master node uses a monitoring process that tracks specified metrics and interacts with the DRIP controller, which can scale out workers on demand. The master is also responsible for partitioning input parameters and distributing them to workers for parallel execution and then recombination. Partitioning input parameters should provide faster execution due to increased speedup. The workers perform the actual query on the data set based on the partitioned input parameters provided by the master node.

When new data are available to the GDAC, it pushes the to the B2SAFE service, triggering a notification to the subscription service, which consequently initiates actions with the new data. If the application is not deployed to FedCloud already, then DRIP provisions the necessary VMs and network so that the application may be deployed. Next, the deployment agent installs all the necessary dependencies along with the application, including the necessary configuration to access the Argo data. The subscription service signals to the application master node the availability of the input parameters to be processed, whereupon it partitions the input tasks into subtasks and distributes them to the workers. If the input parameters include deadlines, the master node will then prioritize them accordingly. The monitoring process keeps track of each running task and passes that information to the DRIP controller. If the programmed threshold is passed, then the controller will request more resources from the provisioner. Finally, the results of each task are pushed back to the B2SAFE service triggering a notification to the subscription service, after which it notifies the user.

4.3 Infrastructure customization and performance optimization

To meet the time-critical constraints of the data subscription service, data products for all subscriptions should be processed and distributed within a certain time window. Resources need to be elastic to support all tasks without wasting significant resources during less active periods. To this end, DRIP provides an autoscaling option to ensure on-time delivery of the requested data, based on the total budget available for conscripting resources (note that this budget need not be monetary but could instead be tied to other metrics such as energy use). However, simply adding resources is not always sufficient to provide the best possible performance for an application; to fully take advantage of the available resources, it is often necessary to change the invocation parameters of an application and partition them in a manner that will achieve good scalability and efficiency.

Two basic optimization strategies have been investigated for partitioning and scheduling subscription tasks to minimize resource usage while meeting all necessary deadlines.

Input partitioning. We investigated two types of input partitioning: linear and logarithmic. With linear partitioning, we simply divide the input range into equal parts for parallel processing. With logarithmic partitioning, we split the range into larger sections at the beginning of the range (accounting for the sparser data recorded early in the Euro-Argo data set) and smaller sections towards the end (when observations become more detailed).

Deadline-aware autoscaling. As mentioned in Section 4.2, the user has the option to specify a deadline for obtaining the requested data. To ensure on-time data delivery, the application master calculates the ‘importance’ of each task based on its deadline and input parameters:

\[ \text{Imp}(\text{task}) = (|P| \cdot w_p) + (ttd \cdot w_d) + (tr \cdot w_t) + (a \cdot w_a) \] (1)
As soon as the global data assembly center pushes out new data, the process begins. All tasks are sorted according to Equation (1); then, the application monitor constantly evaluates the next task’s time-to-deadline. If it is greater than the chosen threshold, then the controller provisions more resources.

\[
\text{Equation (1)}: \quad P, \text{ttd}, t_r, a, w_p, w_d, w_t, \text{and } w_a
\]

In Equation (1), \(P\) is the parameter list, \(\text{ttd}\) is the time-to-deadline, \(t_r\) is the time range, \(a\) is the area, and \(w_p, w_d, w_t, \text{and } w_a\) are the respective weights that determine each parameter’s importance.

Ascertaining the prioritization of tasks allows for smarter scaling behavior on the part of system provisioning by determining on which parameters thresholds should be placed to trigger scaling. Figure 6 illustrates how the process of the deadline-aware autoscheduling proceeds.

5 | EXPERIMENTAL RESULTS

In this section, we present the results of the experiments described in the previous section.

5.1 | Input partitioning

Before attempting to partition input parameters and distribute them to worker nodes, we must first identify which parameter is responsible for the most computing time when generating the data products. To do this, we generated a set of tasks on a region of randomly selected raw data requiring computing of all parameters. We performed 550 tasks spanning the Mediterranean Sea while requesting data in a time window from 1999 to 2007 and covering more than 400 possible parameters in the data products. We executed these tasks on identical VMs and measured their execution time to determine the correlation between area, time range, and the number of parameters with execution time. Additionally, we investigated the effect of end date on execution time, eg, whether execution time changes when processing three months of data leading up to 1999 rather than leading up to 2007 (indicating a general shift in the typical volume of data collected at different points in time). We use this correlation analysis to select a suitable partitioning strategy. For our experiments we used EGI’s FedCloud as our test bed; all VMs in these experiments were identical, with two cores and 2 GB of RAM.

We tested the logarithmic partitioning strategy under the assumption that input data are not always equally distributed; therefore, the load balance on the worker nodes would not be the same. For both strategies, we applied the same task with the following input parameters:

1. The Mediterranean as the target area.
2. A time range from 01/01/99 to 01/01/07.
3. 412 different additional parameters.

We measured the speedup and efficiency using one, two, four, and eight VMs with one worker node per VM for both strategies. We also looked at speedup and efficiency as we added more tasks per worker node. With speedup, we measured how much faster an application becomes when adding more VMs compared with using only one VM—the ratio of the sequential execution time to the parallel execution time \(S = T_s / T_p\). For efficiency, we measured the fraction of time in which a node is utilized such that \(E = S / p\).

In Table 1, we provide the correlation coefficients between execution time and each of time coverage, area, number of (other) parameters, and the end time stamp (of the coverage range). According to these results, the time coverage has a strong positive relation (0.93) with the execution time followed by end time (0.65). This suggests that the more dates we request to process, the more time it takes to process the request, while the other variables do not indicate any particular strong relationship with the execution time.
**TABLE 1** Correlations with execution time of various parameterization options

<table>
<thead>
<tr>
<th>Correlations</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution time</td>
<td>1.00</td>
</tr>
<tr>
<td>Time coverage</td>
<td>0.93</td>
</tr>
<tr>
<td>Area</td>
<td>0.03</td>
</tr>
<tr>
<td>Number of parameters</td>
<td>0.02</td>
</tr>
<tr>
<td>End timestamp</td>
<td>0.65</td>
</tr>
</tbody>
</table>

Figures 7 and 8 show the speedup and efficiency results. The lines indicated as 'log1' and 'log4' indicate speedup for logarithmic partitioning while assigning one and four tasks per worker, respectively. The lines indicated as 'lin1' and 'lin4' represent linear partitioning with the same assignments. These results indicate that the logarithmic partitioning with four tasks per worker performs best (log4), which aligns with Table 1.

5.2 Deadline-aware autoscaling

Using Equation (1), we ranked 100 tasks each with the same deadline but varying areas and time ranges. After ranking these tasks, we set the time-to-deadline as a metric for a monitoring process. When the time-to-deadline dropped below a certain threshold, a signal was sent to the controller to scale up the application. In this particular setup, the controller started a new VM each time it received a signal until a
specified VM limit was reached, after which the controller would start a new worker on each VM in a round-robin fashion. We examined three different cases:

1. no scaling,
2. scaling with a static threshold, and
3. scaling with a dynamic threshold.

In the case of static scaling, the controller takes no action when the time-to-deadline drops below the threshold. In the second case, the threshold was set to a static value (chosen after an empirical study). In the third case, the threshold was initially set to a specific value, but as soon as the time-to-deadline dropped below the threshold, a signal was sent to the controller to scale the application, and the new threshold value was set to the current time-to-deadline minus a selected factor. For the third case, we tried to avoid aggressive scaling in an attempt to provision only as many VMs as necessary so that we could finish all tasks in time. For this experimental setup, we specified a limit to the number of VMs to eight with two workers per VM, meaning that the maximum number of workers at any time was 16—this represented the budget limit that might be imposed by the application developer to prevent ‘runaway’ scheduling of VMs.

Figures 9, 10 and 11 show the results for each of the cases described in Section 4.3 in turn. In all figures, the x-axis represents the task number, the left-side y-axis the time to deadline (in seconds), and the right-side y-axis the number of nodes used for each execution. Also, in Figures 10 and 11, we show the threshold for triggering the addition of more resources. In Figure 9, although the cost of the application is minimal (only one VM) after approximately 22 tasks are initiated, all deadlines are missed. In Figure 10, we observe all tasks are processed within their deadline, but the controller overprovisions VMs for the task, reaching the specified limit of 16 workers (two workers per VM) very quickly. Finally, in Figure 11, we see that the controller provisions just enough workers to complete all tasks in time with the exception of the last, which overshoots its deadline by 2 s (which may or may not be unacceptable given the strictness of the deadline imposed—in this particular instance however, we deem it acceptable given the overall high QoS provided).
Discussion

The results presented here demonstrate that a linear partitioning strategy can provide nonlinear variations in speed and efficiency. This can be attributed to an unequal load distribution, where some workers were assigned far smaller loads than others, despite the data being split ‘evenly’ across a certain dimension. In the case of the Euro-Argo data set used for this experiment, this is because more recent data samples contain more data than older samples (due to improvements in data acquisition over time), which explains why the logarithmic partitioning performed better. However, the recorded speedup can be improved further if the partitioning is calibrated based on the actual end date selected for a sample. Moreover, a more linear speedup could be achieved if the partitioning is performed based on all input dimensions, rather than just one. This is not a trivial task however, as the input domain may be n-dimensional and the load may not be linear across all dimensions, making finding the appropriate hyperplanes to divide the domain into equal task loads challenging. In addition to identifying such appropriate hyperplanes, another challenge arises: How can we select any kind of input parameter partitioning if we cannot analyze the input data set in advance? In our case, we performed a correlation study to identify the relationship between the input parameter of a problem and the execution time. However, that correlation study only used a small sample and often analyzing the entire data set is not practical. To this end, it is worth investigating statistical sampling methods that may provide the most representative sample. Such a process may be complemented by an iterative process where real data coming from monitoring would help evaluate and improve both the sampling and partitioning. Historical observations on the same or similar (for a given judgement of ‘similarity’) can also contribute to selecting the best partitioning strategy.

One area that we have not investigated but which has impact on both the performance and requirements of the data subscription pipeline is the case where subscribers subscribe not only to one custom view on a single data set (albeit a very rich one) but also to a view that combines data from multiple data sets, possibly hosted by multiple RIs. In this scenario, there will be multiple sources from which to retrieve the data required for processing, and it will be necessary to consider how to join as well as partition the data in a way that accounts for factors not in play here; for example, when different data sets are geographically dispersed, workers may actually be deployed in different data centers to ensure performance.

CONCLUSION AND FUTURE WORK

In this paper, we have presented our DRIP microservice suite for optimizing the runtime QoS provided by a data service deployed dynamically on a virtualized e-infrastructure, with a particular focus on time-critical constraints such as deadlines for delivering data to a distributed set of targets. We demonstrated how DRIP can be used to automatically select and provision infrastructure resources, deploy services, and optimize the runtime quality for a data subscription service based on a study case involving the Euro-Argo research infrastructure. We demonstrated how to select an optimal strategy for partitioning the input tasks into workers using a modicum of expert knowledge concerning the specifics of an application. The results clearly show the value of integrated systems such as DRIP for dynamic optimization of data services in research support environments and how, with further investigation and development, they might be used for a number of similar applications cases involving distributed services and large, dynamic data sets.

Nevertheless, it is necessary to acknowledge the difficulty still inherent in building generic solutions for fully automated optimization of infrastructure for arbitrary data services. Some degree of application-specific customization is still necessary when applying infrastructure-level optimization. It can be hoped however that further investigation and classification of different kinds of data service will enlighten the development community as to the best mechanisms and heuristics for optimization.
In this light, an important future work will be deploying DRIP as an optimization engine for a broader range of services provided on behalf of environmental RIs—by doing this, we will be able to explore a wider range of usage scenarios and identify new optimization strategies for input partitioning and dynamic provisioning of infrastructure. For example, DRIP could consider how resource failures would have an impact on deadlines and the strategies for swiftly reacting to such events. Moreover, integrating DRIP with data processing frameworks from specific research domains will also be important for refining our approach, allowing us to work in complement with established and new frameworks for scientific data handling. For example, automated data quality checking of distributed data streams is an important aspect of many scientific disciplines. DRIP provides a natural context for studying challenges such as in-time resource scaling and optimal resource placement, and the further development of tools like DRIP will contribute to continuing global efforts to consolidate research infrastructure and other research support environments.

ACKNOWLEDGMENTS

This work was supported in part by the European Union’s Horizon 2020 Research and Innovation Programme under grants 654182 (ENVRIPLUS project), 676247 (VRE4EIC project), 643963 (SWITCH project), and 654142 (EGI-Engage project).

ORCID

Paul Martin https://orcid.org/0000-0003-1916-864X
Huan Zhou https://orcid.org/0000-0003-2319-4103
Thierry Carval https://orcid.org/0000-0003-2700-4020
Baptiste Grenier https://orcid.org/0000-0002-5686-3193
Jani Heikkinen https://orcid.org/0000-0003-4385-8288
Zhiming Zhao https://orcid.org/0000-0002-6717-9418

REFERENCES

5. Miller MA, Pfeiffer W, Schwartz T. The CIPRES science gateway: enabling high-impact science for phylogenetics researchers with limited resources. In: Proceedings of the 1st Conference of the Extreme Science and Engineering Discovery Environment: Bridging From the eXtreme to the Campus and Beyond; 2012; Chicago, IL.


33. Anadiotis ACG, Gallicco L, Milardo S, Morabito G, Palazzo S. Towards a software-defined network operating system for the IOT. Paper presented at: IEEE 2nd World Forum on Internet of Things (WF-IoT); 2015; Milan, Italy.


50. Shi J, Luo J, Dong F, Zhang J. A budget and deadline aware scientific workflow resource provisioning and scheduling mechanism for cloud. Paper presented at: 18th International Conference on Computer Supported Cooperative Work in Design (CSCWD); 2014; Hsinchu, Taiwan.


