Case Study: Data Subscriptions Using Elastic Cloud Services

Spiros Koulouzis[1], Thierry Carval[2], Jani Heikkinen[3], Antti Pursula[3], and Zhiming Zhao[1](✉)

[1] Multiscale Networked Systems, University of Amsterdam, 1098XH Amsterdam, The Netherlands
{s.koulouzis,z.zhao}@uva.nl
[2] Ifremer, Brest, France
thierry.carval@ifremer.fr
[3] CSC - IT Center for Science, Espoo, Finland
{jani.heikkinen,antti.pursula}@csc.fi

Abstract. To perform data-centric research in environmental and earth sciences, researchers need effectively query, select and access data products from different research infrastructures. When providing observation data continuously, infrastructure is expected to create and deliver customised data products, e.g. for specific geo-regions, time durations or observation parameters, to enhance its ability to serve the research communities. Such kind of services often have time-critical requirements; some tasks need to be carried out within specific time windows when the data products are needed for real-time modelling or simulation frameworks.

Keywords: Research infrastructure · Data subscription · Cloud computing

1 Introduction

Many environmental and Earth science Research Infrastructures (RIs) act as data hubs and publishers of scientific data and serve their user communities via an integrated data portal [6]. The Euro-Argo RI [8] is a typical example of a long-established, distributed RI from the marine domain and is the European contribution to the Argo programme. Argo monitors the world's oceans measuring temperature, salinity, pressure, etc. via the 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 centres, which provide integrated, cleaned data products to various regional centres, archives and research teams; all data is then made publicly available via a common portal within 24 h of acquisition.

Due to the maturity in data acquisition, Euro-Argo seeks improved publishing methods for accessing existing curated data collections, and thus, prototypes a subscription service for their data. In contrast, to merely providing collected data freely for download and requiring researchers to monitor the core Argo dataset for updates manually, researchers are instead allowed to subscribe to specific subsets of Argo data and have updates pushed to their cloud storage, thus streamlining data delivery and accelerating data science workflows involving those data.

In this chapter, we will demonstrate how the Dynamic Real-time Infrastructure Planner developed in the project can be used for optimising virtual infrastructures for the EuroArgo research infrastructure to realise its data subscription service. The use case is prototyped based on EGI FedCloud and EUDAT's B2SAFE. This chapter is an extension of the work published in [1].

2 Data Subscription in RIs

2.1 A Data Subscription Scenario in EuroArgo

In the Euro-Argo data subscription scenario, investigators subscribe to customised views (e.g. specific regions, time durations, and observation parameters) on the Argo data using a data subscription service. Euro-Argo provides the infrastructure services needed for computing data products to match each subscription and then dispatches those products to their destinations; the subscription service can then distribute the tailored updates to investigators' private storage.

A typical subscription task can be made up of a set of input parameters:

1. An area expressed as a bounding box (geospatial data are widespread in environmental and earth science).
2. A time range (typically investigators 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 (e.g. 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 (deadlines may be expressed in terms of maximum accepted time for delivery of the data product).

To deliver the data subscription service, a distributed infrastructure is needed for computing data products and delivering subscriptions to users. The subscription scenario is often time-critical where a number of subscriptions must be fulfilled on a deadline to receive the data products. Different products may require different degrees of processing at different times and place differing levels of load on the processing infrastructure.

Such a data subscription scenario serves both end-users and application workflows for which the retrieval of subscribed-to data is a crucial input. Frequently these workflows require specific data to be delivered within a specific time window and often have firm or soft real-time requirements [9]. The type of real-time requirement is specified by the end-user or the workflow developer.

As the volume of subscriptions and the customizability of subscriptions increases, so too does the pressure on the underlying infrastructure providing the data, the bandwidth for transport and the processing capacity. At the same time, 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 quality of service during peak periods.

2.2 Generalising the Service to Different RIs

In addition to Euro-Argo, other RIs are now looking into the data subscription scenario as an approach to better serve their communities. RIs differ in several aspects, such as in their maturity in various data life cycle phases, in their internal diversity, and in collaboration between RIs. Projects such as ENVRIplus seek approaches to enable convergence through reference modelling [7, 10], helping the RIs to identify common processes and structures and to adopt best practices, and integrate to common infrastructural environment (eInfra) services.

However, several of the problems encountered in traditional data curation and publishing still exist and can be summarised as follows:

- accumulating, large, and complex datasets can only be disseminated with extensive effort
- frequently changing datasets can only be monitored with extensive effort
- unpublished, confidential data can only be disseminated to the designated audience

Moreover, as discussed in Chapter 12, other encountered challenges include lack of accounting information, lack of data provenance information, and the complexity involved in using and integrating distributed systems [5]. For example, the latter challenge emerges when data flows can exist between curation, processing, and publishing subsystems, each which can be provided by different RIs. As a result, the total number of data flows can increase to an extent not easily managed by an investigator.

To answer these problems and challenges in general, a data subscription model was proposed to change the way how subsets of frequently changing data collections are published/disseminated to designated investigators.

2.3 Data Subscription Model

Data subscriptions are built upon the well-known Publish/Subscribe messaging pattern providing advantages such as loose coupling of publishers and subscribers in time, space, and synchronization [3]. The pattern typically consists of three types of entities: publishers, subscribers, and a message broker or topologies of brokers forming a communication infrastructure. The broker can implement a messaging matching scheme through which subscribers receive only a subset of the total published messages. The two most common schemes are topic-based and content-based matching. In contrast to the latter, the topic-based matching let the publisher decide the classes of messages to which subscribers can register/subscribe to.

In this model, subscribers register to topics, more specifically to globally unique persistent data identifiers (PID) [see chapter 9]. There are several reasons for this. First, the data discovery process [see chapter 4] can be independent of creating a subscription. Second, the subscriber can create a subscription even before there is published events on the data identifier. Thus, the subscribers are truly uncoupled from the publishers in time. Third, persistent identifiers are seen as the widely accepted approach to support research data re-use and sharing, while also enabling provenance tracking, and consequently, enabling micro-attribution. Fourth, in order to provide a common and robust model, the semantically immutable persistent identifiers provide several valuable characteristics to build on, such as in direction and option for data granularity levels.

In terms of data life cycle stages, Fig. 1 shows the primary flows between the stages and a matching process of the data subscription model.

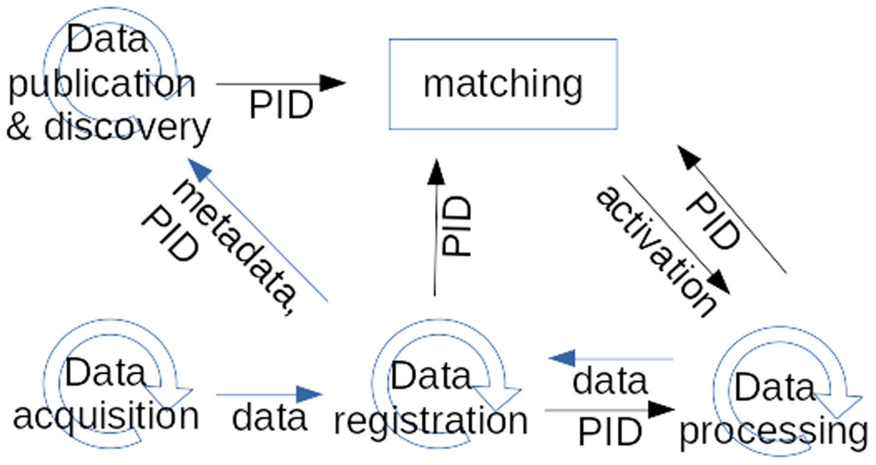

Fig. 1. The data life cycle stages in the Data Subscription Mode.

3 Architectural Design and Prototype

Figure 2 illustrates a functional depiction of the Euro-Argo data subscription scenario.

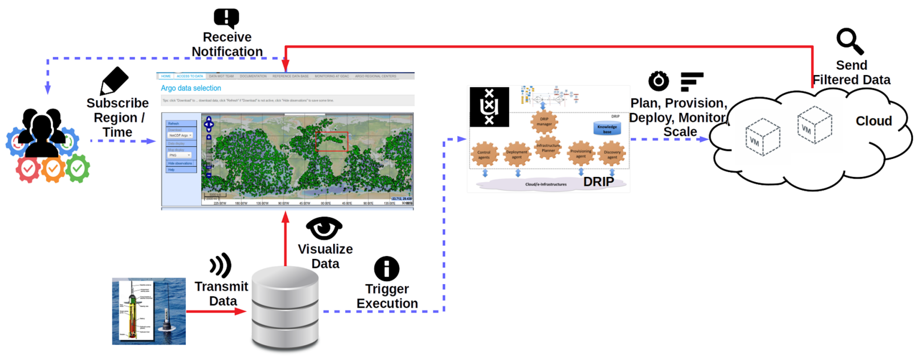

Fig. 2. The data subscription scenario is one where researchers can subscribe to the specific data they are interested in (e.g. marine data from floats in the Mediterranean) via a simple community portal, and have updates pushed to their workspaces.

3.1 Architecture Design

We applied a prototype of the data subscription service in the scenario depicted in Fig. 3. Currently, the resources from e-infrastructures such as EUDAT [4] and EGI FedCloud are used. Figure 3 shows the use-case scenario based on the use of EUDAT and EGI services. In this case, EUDAT provides services for data subscription, storage, and data transfer, while EGI FedCloud provides the services for the computing of data products for each subscription.

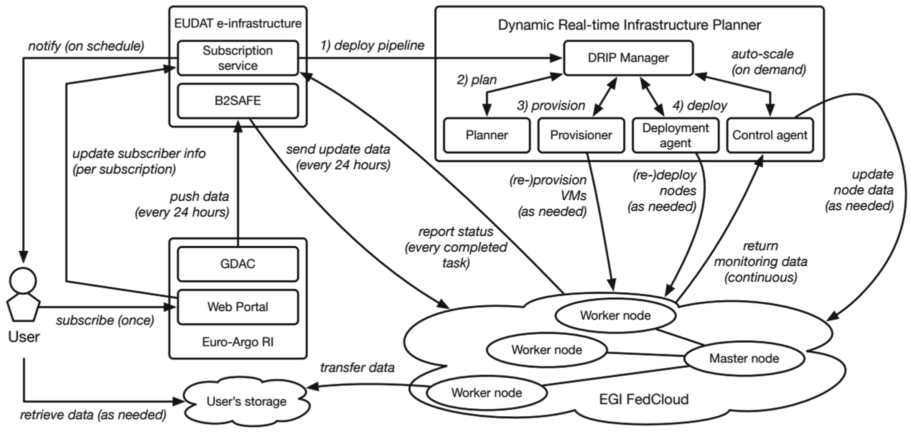

Fig. 3. A context diagram showing interactions between components in the Euro-Argo data subscription scenario. The subscription service invokes DRIP to plan, provision and deploy the subscription data processing pipeline. Subscriptions and processing are event-driven, triggered by updates pushed to the B2SAFE data repository. The deployment is scaled with demand [1].

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

1. A data selection community portal serving as the front-end;
2. The global data assembly centre of Euro-Argo [8], providing the source research dataset;
3. B2SAFE data repository [2] provided by EUDAT;
4. A deployment of DRIP [1] (deployed within EGI FedCloud);
5. A data filtering application. This is the software that actually takes the input parameters and composes the requested data product from the raw source research dataset;
6. EGI FedCloud virtual resources, forming the fundamental infrastructure for data processing and transportation;
7. EUDAT data subscription service (which maintains and matches the subscriptions defined via the data selection community 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 and optionally set a deadline for receiving the requested results. The global data assembly centre (GDAC) of Euro-Argo receives new datasets from regional centres and pushes them to the EUDAT B2SAFE data service. The subscription service itself maintains records of subscriptions including references to selected parameters and associated actions. The subscription service is based on well-defined APIs that allow connecting it with various community front-ends and infrastructure platforms. The role of DRIP then is to plan and provision a customised 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 data filtering 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 as individual tasks for parallel execution and for combining individual results into the desired data product. Partitioning input parameters should provide faster execution due to increased speed-up. The *workers* perform the actual query on the dataset based on the partitioned input parameters provided by the master node.

When new data is available to the GDAC, it pushes them to the B2SAFE service, triggering a notification to the subscription service, which consequently initiates actions on 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 data filtering application including configurations to access on 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 sub-tasks and distributes them to the workers. If the input parameters include deadlines then the master will prioritise 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[1].

3.2 Infrastructure Customisation and Performance Optimisation

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 auto-scaling option to ensure on-time delivery of the requested data, based on the total budget available for conscripting resources (not that this budget need be monetary; it could also be tied to other metrics such as energy use). However, simply adding resources is not always enough 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.

[1] The use case online demo: https://www.youtube.com/watch?v=PKU_JcmSskw&t=12s.

Two basic optimisation strategies have been investigated for partitioning and scheduling subscription tasks in order to minimise 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 dataset) and smaller sections towards the end (when observations become more detailed).

Deadline-Aware Auto-Scaling. 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:

$$Imp(task) = \left(|P| \cdot w_p\right) + \left(ttd \cdot w_p\right) + (tr \cdot w_d) + (\alpha \cdot w_a) \tag{1}$$

In Eq. 1, P is the parameter list, ttd is the time-to-deadline, tr is the time range, α is the area and w_p, w_d, w_t, w_α are the respective weights that determine each parameter's importance.

Ascertaining the prioritisation of tasks allows for smarter scaling behaviour on the part of the provisioning system by determining which parameters thresholds should be placed to trigger scaling. Figure 4 illustrates how the process of the deadline-aware auto-scheduling proceeds.

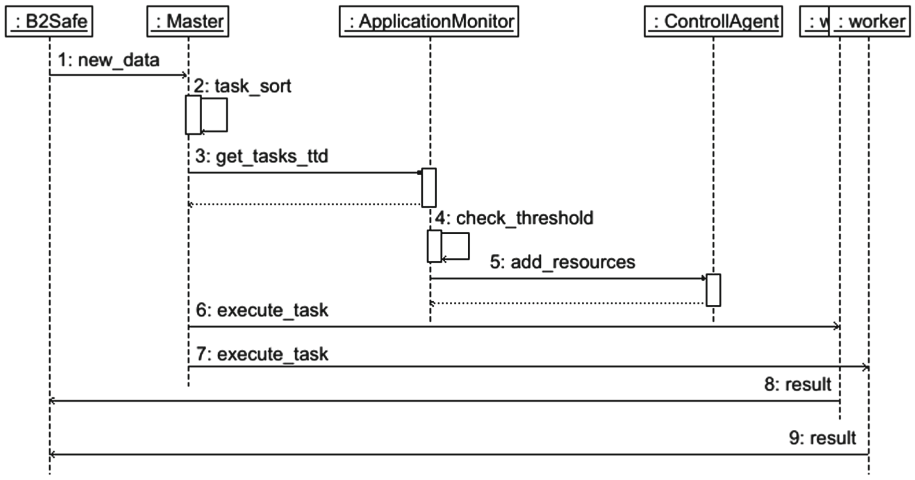

Fig. 4. Deadline-aware auto-scheduling flow. As soon as the GDAC pushes out new data, the process begins. All tasks are sorted according to Eq. 1, then the application monitor constantly evaluates the next task's time-to-deadline. If it is greater than the chosen threshold, then the controller provisions more resources.

4 Experimental Results

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

4.1 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 the end date on execution time, e.g. 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 two GBs of RAM.

We tested the logarithmic partitioning strategy under the assumption that input data are not always equally distributed, and 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 speed-up and efficiency using 1, 2, 4, and 8 VMs with one worker node per VM for both strategies. We also looked at speed-up and efficiency as we added more tasks per worker node. With speed-up, 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 utilised such that $E = S/p$.

In Table 1, we provide the correlation coefficients between execution time and each time *coverage*, *area*, *number of (other) parameters* and the *end timestamp* (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.

Figure 5 and Fig. 6 show the speedup and efficiency results. The lines indicated as 'log1' and 'log4' indicate speed-up for logarithmic partitioning while assigning 1 and 4 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 4 tasks per worker performs best (log4), which aligns with Table 1.

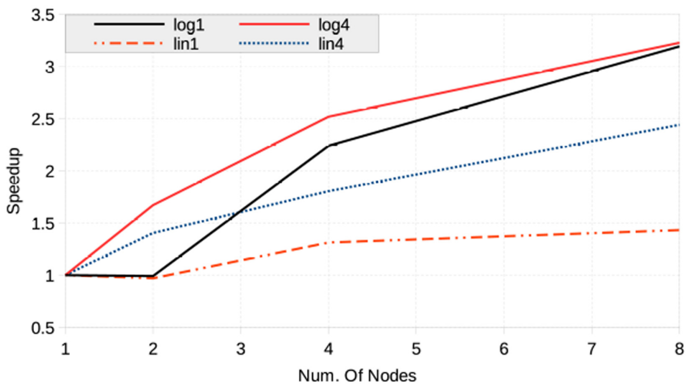

Fig. 5. Speed-up for linear and logarithmic partitioning strategies assigning 1 and 4 tasks per worker.

Table 1. Correlations with the execution time of various parameterisation options.

Correlations	Execution time
Execution time	1.00
Time coverage	0.93
Area	0.03
Num. of parameters	0.02
End timestamp	0.65

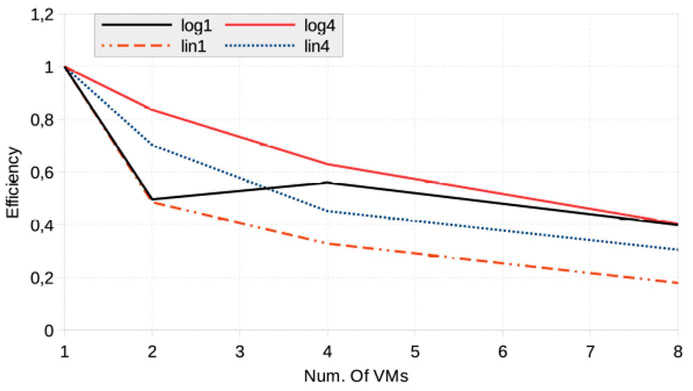

Fig. 6. Efficiency for linear and logarithmic partitioning strategies assigning 1 and 4 tasks per worker.

4.2 Deadline-Aware Auto-Scaling

Using Eq. 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 'run-away' scheduling of VMs.

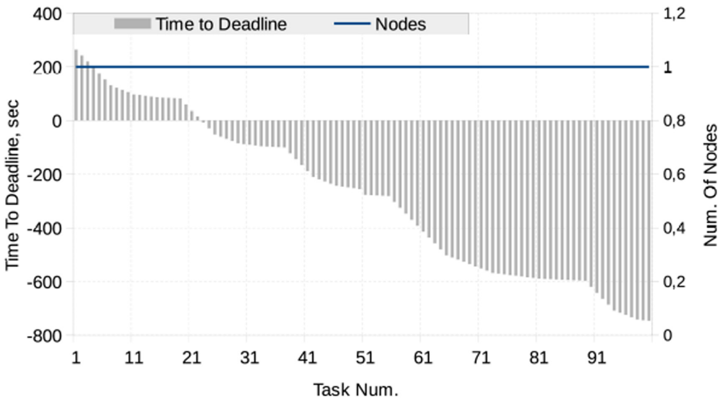

Fig. 7. Process 100 tasks with no scaling.

Figure 7, Fig. 8 and Fig. 9 show the results for each of the cases described above. In all figures the -axis represents the task number, the left-side -axis the time to the deadline (in seconds) and the right-side -axis the number of nodes used for each execution. Also, in Fig. 8 and Fig. 9 we show the threshold for triggering the addition of more resources. In Fig. 7, although the cost of the application is minimal (only one VM) after approximately 22 tasks are initiated, all deadlines are missed. In Fig. 8, we observe all tasks are processed within their deadline, but the controller over-provisions VMs for the task, reaching the specified limit of 16 workers (two workers per VM) very quickly.

Finally, in Fig. 9, we see that the controller provisions just enough workers to complete all tasks on time with the exception of the last, which overshoots its deadline by two seconds (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 quality of service provided).

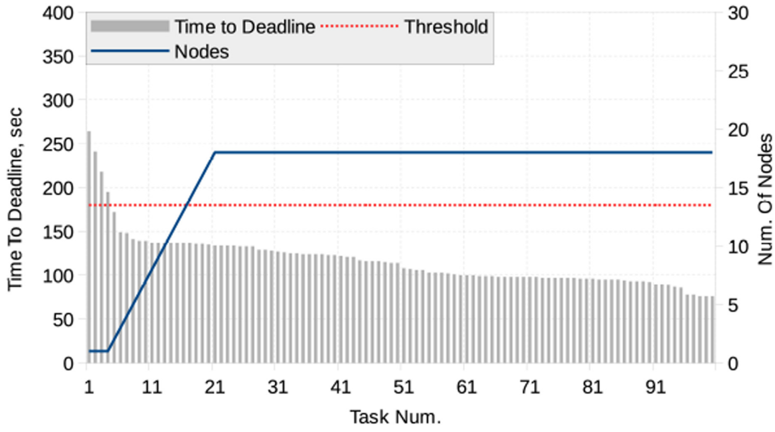

Fig. 8. Process 100 tasks with a static threshold.

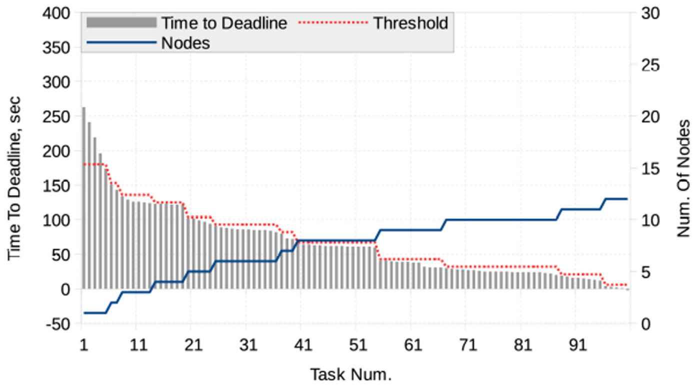

Fig. 9. Process 100 tasks with a dynamic threshold.

5 Discussion

The results presented here demonstrate that a linear partitioning strategy can provide non-linear 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 dataset 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 speed-up can be improved further if the partitioning is calibrated based on the actual end date selected for a sample. Moreover, a more linear speed-up could be achieved if the partitioning was performed based on all, rather than just one, input dimensions. This is not a trivial task; however, as the input domain may be -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. Besides identifying such appropriate hyperplanes, another challenge arises: how can we select any kind of input parameter partitioning if we cannot analyse 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 analysing 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 an impact on both the performance and requirements of the data subscription pipeline is the case where subscribers subscribe not just to one custom view on a single dataset (albeit a very rich one), but to a view that combines data from multiple datasets, possibly hosted by multiple RIs.

In this scenario, there will be multiple distinct persistent identifiers for data objects or collections. The objects and collections are curated by another entity than the subscriber. For example, when the location of the data objects or collections changes, the subscription remains valid assuming the curator updates the property of the identifier.

Moreover, 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, where different datasets are geographically dispersed and so workers may actually be deployed in different data centres to ensure performance.

A further consideration emerges from a requirement for immutable semantics of persistent identifiers. When the investigator finds a data set and metadata describing the data through the discovery process, interpretation of using the corresponding identifier in a subscription, and inclusion of an action which perceives the structural semantics of the data, need to be considered.

6 Conclusion and Future Work

In this paper, we have presented a service prototype to define data subscriptions to Euro-Argo data, connected to invoking a processing pipeline optimised with Dynamic Real Time Infrastructure Planner (DRIP) solution. We demonstrated how DRIP could be used to automatically select and provision infrastructure resources, deploy services, and optimise the runtime quality for the EUDAT data subscription service based on a study case involving the Euro-Argo research infrastructure.

The DRIP microservice suite optimises the runtime quality of service provided by a data service deployed dynamically on a virtualised 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 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 optimisation 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 datasets. Furthermore, we showed subscription matching in the presence of existing data life cycle stages through which investigators can free up time and be notified of significant events in subscribed data.

The demonstrated Data subscription service prototype is part of the EUDAT innovation portfolio. The starting point for developing the subscription service has been to provide a generic component that can be connected to different community front-ends, and that can utilise different e-infrastructure platforms for automated processing. This paper presents the successful Euro-Argo pilot case within ENVRIplus project that demonstrates the interaction of several service components from several providers: the Euro-Argo data portal, EUDAT B2SAFE data storage, the EUDAT data subscription service, the DRIP solution, and EGI FedCloud. The EUDAT Collaborative Data Infrastructure has identified the potential in the subscription model and considers it as a possible new addition to the data management services, depending on the interest of user communities and availability of development resources.

Regarding the DRIP solution, it is necessary to acknowledge the difficulty still inherent in building generic solutions for fully automated optimisation of infrastructure for arbitrary data services. Some degree of application-specific customisation is still necessary when applying infrastructure-level optimisation. However, further investigation and classification of different kinds of data service will assist in identifying the best mechanisms and heuristics for optimisation.

In this light, an important future work will be deploying DRIP as an optimisation engine for a broader range of services provided on behalf of environmental RI—by doing this, we will be able to explore a wider range of usage scenarios and so identify new optimisation 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 of distributed data streams is an important aspect of many disciplines including environmental science. Challenges such as in-time resource scaling and optimal resource placement will be studied in the context of DRIP. This will add to continuing global efforts to consolidate research infrastructure and other research support environments.

Acknowledgements. This work was supported by the European Union's Horizon 2020 research and innovation programme via the ENVRIplus project under grant agreement No 654182.

References

1. Koulouzis, S., et al.: Time-critical data management in clouds: challenges and a dynamic real-time infrastructure planner (DRIP) solution. Concurr. Comput. Pract. Exp. e5269 (2019). https://doi.org/10.1002/cpe.5269
2. Cacciari, C., Fares, M., Fiameni, G., Michelini, A., Danecek, P., Wittenburg, P.: Adoption of the B2SAFE EUDAT replication service by the epos community. In: EGU General Assembly Conference Abstracts 16 (2014)
3. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.: The many faces of publish/subscribe. ACM Comput. Surv. (CSUR) **35**(2), 114–131 (2003)
4. Gentzsch, W., Lecarpentier, D., Wittenburg, P.: Big data in science and the EUDAT project. In: Global Conference (SRII) 2014, pp. 191–194 (2014), http://doi.org/10.1109/SRII.2014.34
5. Ahanach, E. el K., Koulouzis, S., Zhao, Z.: Contextual linking between workflow provenance and system performance logs. In: 2019 15th International Conference on eScience (eScience), pp. 634–635. IEEE, San Diego (2019). https://doi.org/10.1109/eScience.2019.00093
6. Hu, Y., et al.: Deadline-aware deployment for time critical applications in clouds. In: Rivera, F.F., Pena, T.F., Cabaleiro, J.C. (eds.) Euro-Par 2017. LNCS, vol. 10417, pp. 345–357. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-64203-1_25
7. Atkinson, M., Hardisty, A., Filgueira, R., Alexandru, C., Vermeulen, A., Jeffery, K., Loubrieu, T., Candela, L., Magagna, B., Martin, P., et al.: A consistent characterisation of existing and planned RIs. ENVRIplus deliverable 5.1, submitted on 30 April 2016
8. Wong, A., Keeley, R., Carval, T.: The Argo data management team (2013) Argo quality control manual, version 2.8. Argo Data Management (2010)
9. Evans, K., et al.: Dynamically reconfigurable workflows for time-critical applications. In: Proceedings of the 10th Workshop on Workflows in Support of Large-Scale Science - WORKS 2015, pp. 1–10. ACM Press, Austin, Texas (2015). https://doi.org/10.1145/2822332.2822339
10. Zhao, Z., et al.: Reference model guided system design and implementation for interoperable environmental research infrastructures. In: 2015 IEEE 11th International Conference on e-Science, pp. 551–556. IEEE, Munich (2015). https://doi.org/10.1109/eScience.2015.41