GigaScience, 10, 2021, 1–14

doi: 10.1093/gigascience/giab018
Technical Note

TECHNICAL NOTE

Rapid development of cloud-native intelligent data pipelines for scientific data streams using the HASTE Toolkit

Ben Blamey [1,*], Salman Toor [1], Martin Dahlö [2,3], Håkan Wieslander [1], Philip J. Harrison [2,3], Ida-Maria Sintorn [1,3,4], Alan Sabirsh [5], Carolina Wählby [1,3], Ola Spjuth [2,3,†] and Andreas Hellander [1,†]

[1]Department of Information Technology, Uppsala University, Lägerhyddsvägen 2, 75237 Uppsala, Sweden; [2]Department of Pharmaceutical Biosciences, Uppsala University, Husargatan 3, 75237, Uppsala, Sweden; [3]Science for Life Laboratory, Uppsala University, Husargatan 3, 75237 Uppsala, Sweden; [4]Vironova AB, Gävlegatan 22, 11330 Stockholm, Sweden and [5]Advanced Drug Delivery, Pharmaceutical Sciences, R&D, AstraZeneca, Pepparedsleden 1, 43183 Mölndal, Sweden

*Correspondence address: Ben Blamey, Department of Information Technology, Uppsala University, Box 337, 75105 Uppsala, Sweden. E-mail: ben.blamey@it.uu.se http://orcid.org/0000-0003-1206-1428
†Co–senior authors.

Abstract

Background Large streamed datasets, characteristic of life science applications, are often resource-intensive to process, transport and store. We propose a pipeline model, a design pattern for scientific pipelines, where an incoming stream of scientific data is organized into a tiered or ordered "data hierarchy". We introduce the HASTE Toolkit, a proof-of-concept cloud-native software toolkit based on this pipeline model, to partition and prioritize data streams to optimize use of limited computing resources. **Findings** In our pipeline model, an "interestingness function" assigns an interestingness score to data objects in the stream, inducing a data hierarchy. From this score, a "policy" guides decisions on how to prioritize computational resource use for a given object. The HASTE Toolkit is a collection of tools to adopt this approach. We evaluate with 2 microscopy imaging case studies. The first is a high content screening experiment, where images are analyzed in an on-premise container cloud to prioritize storage and subsequent computation. The second considers edge processing of images for upload into the public cloud for real-time control of a transmission electron microscope. **Conclusions** Through our evaluation, we created smart data pipelines capable of effective use of storage, compute, and network resources, enabling more efficient data-intensive experiments. We note a beneficial separation between scientific concerns of data priority, and the implementation of this behaviour for different resources in different deployment contexts. The toolkit allows intelligent prioritization to be `bolted on' to new and existing systems – and is intended for use with a range of technologies in different deployment scenarios.

Keywords: stream processing; interestingness functions; HASTE; tiered storage; image analysis

Received: 14 September 2020; **Revised:** 26 January 2021; **Accepted:** 23 February 2021

Introduction

Large datasets are both computationally and financially expensive to process, transport, and store. Such datasets are ubiquitous throughout the life sciences, including imaging, where different types of microscopy are used to, e.g., observe and quantify effects of drugs on cell morphology. Modern imaging techniques can generate image streams at rates of up to 1 TB/hour [1]. Clearly, the processing, storage, and communication of these images can be slow, resource-intensive, and expensive, effectively becoming a bottleneck to scale experiments in support of data-driven life science. Another prominent example is human genome sequencing, where the global storage requirement is predicted to be between 2 and 40 EB (1 exabyte = 10^{18} bytes) by 2025, and with modern techniques generating data at the order of 60 GB/h [2]. Similarly, in large-scale modelling, a single computational experiment in a systems biology context can generate terabytes of data [3].

This work is motivated by some of the most critical aspects of scalable scientific discovery for spatial and temporal image data. There are 2 primary concerns: (i) not all data are equally valuable. With datasets outgrowing resources, data storage should be prioritized for data that are most relevant (or interesting) for the study at hand and poor-quality, or uninteresting, data (e.g., out-of-focus images) should be discarded or archived; (ii) when resources are limited or if decisions are required in real time, we have to be smart about how the data are (pre)processed and which subsets of the data are stored for more detailed (and potentially computer-intensive) analysis, prioritizing more interesting subsets of the data.

The general challenges of management and availability of large datasets are often popularized and summarized through the so-called "Vs of big data." Initially, the focus was on the 3 Vs: velocity, volume, and variety, but this list has since grown with the increasing number of new use cases to also include Vs such as veracity, variability, virtualization, and value [4]. During the past decade, a number of frameworks have been designed to address these challenges, offering reliable, efficient, and secure large-scale data management solutions. However, according to a white paper published by IDC [5], only 30% of the generated data is in the form that it can be efficiently analyzed. This highlights the current gap between large-scale data management and efficient data analysis. To close this gap, it is essential to design and develop intelligent data management frameworks that can help organize the available datasets for efficient analyses. In this work, we address this challenge by proposing a model that helps a developer to create a data pipeline able to make online decisions about individual data objects' priority based in actual information content, or interestingness, rather than traditional metadata (we use the generic term "data object" but note that analogous terms in various contexts include document, image, message, and blob).

A range of existing work in life science applications has discussed the challenges of transporting, storing, and analyzing data, often advocating a streamed approach. Rinehart et al. [6] explicitly discuss the constraints of cloud upload bandwidth, and its effect on overall throughput for mass spectrometry–based metabolomics. In their application, uploading large datasets from the instrument to the cloud represents a bottleneck and they advocate a stream-based approach with online analysis where data is processed when it arrives, rather than waiting for upload of the complete dataset. Hillman et al. [7] developed a stream-based pipeline with Apache Flink and Kafka for processing proteomics data from liquid chromatography–mass spectrometry. The authors note the advantages of a real-time approach to analysis: "a scientist could see what is happening in real-time and possibly stop a problematic experiment to save time." Zhang et al. [8] developed a client/server application for interactive visualization of mass spectrometry spectra, adopting a stream-based approach to achieve better user interactivity. In genomics, Kelleher et al. [9] presented the `htsget` protocol to enable clients to download genomic data in a more fine-grained fashion and allow for processing chunks as they come from the sequencer. Cuenca-Alba et al. [10] note that a single electron microscope can produce 1 TB of images per day, requiring a minimum of 1,000 CPU hours for analysis. Adapting their Scipion software [11] (intended for cryo-electron microscopy image analysis) for use in the cloud, they discuss the challenges of data transfer to/from the cloud, comparing transfer rates for different providers. Wang et al. [12] propose excluding outliers in streaming data, using an "Outlier Detection and Removal" algorithm that they evaluate on 5 bioinformatics datasets.

Rather than handling 1 particular type of data or dealing with a specific data pipeline, the aim of the present work is to distill effective architectural patterns into a pipeline model to allow for repeatable implementations of smart systems capable of online resource prioritization in scenarios involving large-scale data production, such as from a scientific instrument. Computers in the laboratory connected directly to such an instrument, used together with cloud resources, are an example of edge computing (see [13]). Under that paradigm, computational resources outside the cloud (such as mobile devices, and more conventional compute nodes) are used in conjunction with cloud computing resources to deliver benefits to an application such as reduced cost, better performance, or improved user experience. General computer science challenges include security, deployment, software complexity, and resource management/workload allocation. In our context, the streams of large data objects generated by scientific instruments create particular challenges within the edge computing paradigm because the data often need to be uploaded to the cloud for processing, storage, or wider distribution. Whilst limited compute resources at the edge are often insufficient for low-latency processing of these datasets, intelligent workload allocation can improve throughput (as discussed in Case Study 2).

In this article we propose a pipeline model for partitioning and prioritizing stream datasets into "data hierarchies" (DHs) according to an "interestingness function" (IF) and accompanying "policy," applied to objects in the stream, for more effective use of hardware (in edge and cloud contexts). We present this as a general approach to mitigating resource management challenges, with a focus on image data. Our model allows for autonomous decision making, while providing a clear model for domain experts to manage the resources in distributed systems—by encoding domain expertise via the IF. To that end, this article introduces the HASTE Toolkit , intended for developing intelligent stream-processing pipelines based on this model. Two case studies presented in this article document how microscopy pipelines can be adapted to the HASTE pipeline model.

Whilst the core ideas of intelligent data pipelines in HASTE are generally applicable to many scenarios involving scientific datasets, we here focus on case studies involving image streams, in particular from microscopy.

Background: Stream Processing and Workflow Management

A fundamental component to the pipelines presented in this article is a stream-processing engine. Systems for stream processing are generally concerned with high-frequency message influx, and those objects can be small in size, such as a few kilobytes. Examples of such data objects include sensor readings from Internet of Things (IoT) devices (such as MQTT messages), those generated from telecoms, web, and cloud applications, e-commerce and financial applications, or the aggregation and analysis log entries. Well-known examples of mature, enterprise-grade frameworks for cloud-based stream processing in these contexts include Apache Flink, Apache Spark Streaming, and Apache Log Flume. Resilience and fault tolerance are key features of these frameworks (often achieved with various forms of redundancy and replication). These frameworks are commonly used in conjunction with various queuing applications, e.g., Apache Kafka, and vendor-specific products such as AWS Kinesis; these also include basic processing functionality.

Whilst the maturity, support, documentation, features, and performance (order of megahertz message-processing throughput) boasted by these frameworks is attractive for scientific computing applications, streamed scientific data (and their processing) tend to have different characteristics: data objects used in scientific computing applications (such as microscopy images, and matrices from other scientific computing domains) can be larger in size, which can create performance issues when integrated with these enterprise frameworks described above [15]. For example, data object sizes in imaging applications could be a few megabytes.

To address this gap, we have previously developed and reported on a stream-processing framework focusing on scientific computing applications, HarmonicIO [16]. HarmonicIO sacrifices some of these features and is intended for lower-frequency applications (towards kilohertz, not megahertz), and was able to achieve better streaming performance under some conditions in 1 study for larger message sizes [15]. The HASTE Toolkit has been developed with images as the primary use case, and for this reason HarmonicIO is the default supported streaming framework in the toolkit. However, we stress here that in principle any streaming framework can be used.

Furthermore, under the emerging edge computing paradigm, there are some stream-processing frameworks available, often focusing on traditional IoT use cases. Being in their infancy, effective automated scheduling and operator placement in hybrid edge/cloud deployment scenarios remains an open research challenge for this context. Within this area, there is significant research effort concerning real-time video analysis, where images collected at the edge (from cameras) are streamed to the cloud for analysis—some degree of lossy compression is typically used in such applications.

By contrast, "workflow" frameworks are a broad class of software frameworks intended to facilitate the development of data-processing pipelines. There are a large number of such frameworks (>100 are listed in [17]). In such frameworks, one generally defines processing operations (often as the invocation of external processes), which are triggered by events such as the creation of a new file on disk, or a commit being pushed to a Git repository. Such frameworks generally handle large numbers of files, of arbitrary size, and often include some degree of fault tolerance. But in contrast to stream-processing frameworks, they may lack functionality specific for streams, such as window operations, more complex scheduling, and placing of operators, and are generally intended for higher latency and/or lower ingress rates (than the 100 kHz and up range of the stream-processing frameworks described above), and are often file system centric, with objects being written back to disk between each processing step.

The HASTE Toolkit attempts to fill a gap between these 2 classes of software (stream-processing frameworks and workflow management systems): applications where latency and high data object throughput are important (and use of a file system as a queuing platform is perhaps unsuitable for that reason) but not as high as in some enterprise stream-processing applications; whilst being flexible enough to accommodate a broad range of integration approaches, processing steps with external tools, and the large message sizes characteristic of scientific computing applications.

The priority-driven approach of the HASTE pipeline model reconciles the resource requirements of life science pipelines (characterized by streams relatively of large message, with expensive per-message processing steps) with the requirements for low latency and high throughput, allowing for real-time human supervision, inspection, and interactive analysis—as well as real-time control of laboratory equipment.

HASTE Pipeline Model

The key ideas of the HASTE pipeline model are the use of IFs and a policy to autonomously induce DHs. These structures are then used to manage and optimize different objectives such as communication, processing, and storage of the datasets. The HASTE Toolkit enables rapid constructions of smart pipelines following this model. Central to the approach is that decisions are made on the basis of actual data content rather than on a priori metadata associated with the data objects. The following subsections introduce the components of the pipeline model.

Overview

Figure 1 illustrates the proposed HASTE model and logical architecture. One or more streaming data sources generate streams of data objects. The stream then undergoes feature extraction (relevant to the context)—this data extraction can be performed in parallel, as an idempotent function of a single object. The intention is that computationally cheap initial feature extraction can be used to prioritize subsequent, more expensive, downstream processing.

An IF computes an interestingness score for each object from these extracted features. This computation can be a simple procedure, e.g., to nominate 1 of the extracted features as the interestingness score associated with the data object. In more complex cases it can also be a machine learning model trained either before the experiment or online during the experiment that generates the stream. Finally, a policy is applied that determines where to store the object within a DH, or whether to send it for further downstream processing, based on the interestingness scores.

Interestingess functions

The IF is a user-provided function, to be applied to the extracted features from the data objects in the stream. The purpose of the IF is to associate an interestingness score with each object. Examples of IFs in image analysis contexts could be features re-

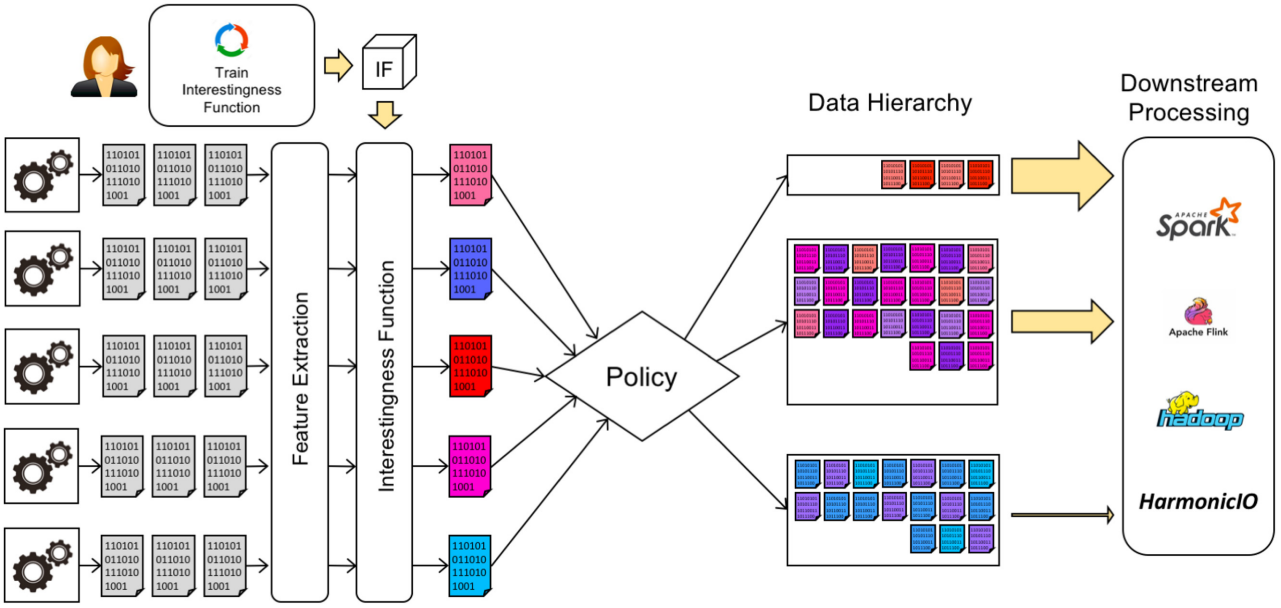

Figure 1: Logical architecture for the HASTE pipeline model. A stream of data objects is generated by 1 or more streaming sources (such as a microscope). These objects undergo online, automated feature extraction, and an IF is applied with the extracted features as input. This associates an interestingness score with each object in the stream. A user-defined policy is then used to organize the data objects into a data hierarchy to be used for optimizing subsequent communication, storage, and downstream processing.

lating to image quality, detected phenomena in images, and so forth.

The computed IF score is used for determining priority for subsequent processing, communication, and/or storage of that object. In this sense, IFs have some similarities to the concept of document (data object) "hotness" in tiered storage contexts, where a more recently accessed "hot" document would be stored in a high-performance tier. Whilst much of that line of work uses only file system information, other work takes some consideration of the application itself; e.g., Chan and Tobagi [18] model access as a Zipf distribution (for a review see [3]).

Our present work generalizes the concept of hotness in a number of ways: (i) our IFs always take consideration of semantics at the level of the scientific application—in the case of microscopy imaging this could be image focus or quality features—perhaps combined with business logic for particular color channels, etc.—rather than file system semantics (such as file access history). This approach allows an immediate, online decision about the object's interestingness rather then inferring it from subsequent access patterns. (ii) Tiered storage is just 1 potential application of HASTE: we use IFs to prioritize data objects for storage, compute, and communication. (iii) With HASTE, the intention is that users configure IFs themselves, together with the associated policy. Currently, the output of the IF is scalar valued. This is intended to ensure smooth integration in cases where the IF is a machine learnt model, outputting a probability, rank, or some other statistical measure.

Furthermore, we propose general software abstractions for these ideas and demonstrate the potential benefits of online interestingness-based prioritization in 2 case studies: both in terms of the optimization of various resources (compute, communication, storage), but also from an experimental and scientific viewpoint—selecting the best data (or outliers) for inspection and further analysis.

Policies for inducing data hierarchies

In applications using tiered storage, more interesting data objects would be saved in higher performance, more expensive tiers—readily accessible for downstream processing (while less interesting objects could be cheaply archived)—explored in Case Study 1. Whereas, in edge computing contexts, we may want to prioritize data objects for computation at the cloud edge, to make more effective use of that resource—explored in Case Study 2. In both cases we refer to these structures as DHs. In a HASTE pipeline DHs are "induced" within the source data by the IF and a policy. The policy takes the interestingness score as input and applies a set of rules to determine how an object is placed within the DH, e.g., its allocation within a tiered storage system; or where it should be stored or processed downstream. Listing 1 shows how a user can define a policy, a simple dictionary mapping intervals of interestingness scores to the tiers (which are configured separately). In this article we demonstrate 2 forms of policy: the aforementioned interval model (in which the tier is determined directly from the interestingness score, Case Study 1) and a priority-based policy, in which data objects are queued (according to their interestingness) for upload and processing (as in Case Study 2).

A benefit of the HASTE pipeline model is the clear role separation—all the domain-specific knowledge is effectively encapsulated within the IF whilst the choice of how to form DHs and optimize storage tier allocation is encapsulated entirely within the policy. This allows the scientific question of what constitutes an interesting data object, and the computing infrastructure, or indeed, budgetary, concerns of how to make best use of computing resources (including storage), to be separated and worked on by team members with different expertise. Importantly, this decoupling allows the possibility for IFs to be reused among scientists, and between contexts where the data may be similar but the dataset size, and available computing infrastructure, may be different.

The HASTE Toolkit

The HASTE Toolkit implements the core functionality needed for rapidly constructing smart pipelines based on the proposed model.

HASTE Storage Client

The HASTE Storage Client (HSC) serves as the main entry point for the user. It is configured with the IF, the policy, and the configuration associated with the tiers, and processes each data object arriving in the stream. It can be installed as a stand-alone Python module (see https://github.com/HASTE-project/HasteStorageClient, version 0.13 was used for this study). It allows a DH to be realized within HASTE as tiered storage. It is a library with the core prioritization functionality: it invokes the IF on incoming data objects and applies the policy to form the data hierarchy. The extracted features are used to compute interestingness scores, which, along with other metadata and logging information, are saved in a database by the HSC. It is intended to be adopted within the Python-based stream-processing framework of choice; an example can be found at https://github.com/HASTE-project/HasteStorageClient/blob/master/example.py. An existing pipeline can be adapted to use HASTE according to the following steps:

- Install the HSC from PyPI `pip install haste-storage-client` or from source.
- Configure 1 or more storage tiers (on a HASTE-compatible storage platform; at the time of writing, supported platforms are OpenStack Swift, Pachyderm [19], and POSIX-compatible filesystems).
- Define an IF for the context—it can use spatial, temporal, or other metadata associated with the data object.
- Run feature extraction on the object prior to invoking the HSC.
- Deploy a MongoDB instance. The scripts https://github.com/HASTE-project/k8s-deployments/ can be adapted for this purpose.

Other key components of the HASTE Toolkit

This section lists other various components in the HASTE Toolkit and describes how they relate to the key ideas of IFs, DHs, and policies.

The HASTE Agent. A command line application (developed for the microscopy use case in Case Study 2), which uploads new documents on disk to the cloud, whilst performing intelligently prioritized pre-processing of objects waiting to be uploaded, to minimize the overall upload duration (see https://github.com/HASTE-project/haste-agent; v0.1 was used for this study). The functionality of this tool is discussed in detail in Case Study 2.

The HASTE Gateway. Cloud gateway service, which receives data objects in the cloud and forwards them for further processing. it is deployed as a Docker container (see https://github.com/HASTE-project/haste-gateway; v0.10 was used in this study).

The HASTE Report Generator. An auxiliary command line tool for exporting data from the Extracted Features Database (see https://github.com/HASTE-project/haste-report-generator).

The Extracted Features Database. MongoDB is used by the HSC to hold a variety of the metadata: extracted features, interestingness scores, and tier/DH allocation.

Tiered Storage. Tiered storage is 1 way that a DH can be realized. The HSC allows existing storage to be organized into a tiered storage system, where tiers using various drivers built into the HSC can be configured. In Case Study 2 the tiers are file system directories, into which image files are binned according to the user-defined policy. In other deployments, less expensive disks/cloud storage could be used for less interesting data. Note that the policy can also send data deemed unusable (e.g., quality below a certain threshold) directly to trash. Tiered storage drivers are managed by the HASTE storage client.

Our GitHub project page (https://github.com/HASTE-project) showcases other components relating to various example pipelines developed within the HASTE project, including IFs developed for specific use cases, as well as scripts for automated deployment.

Whilst the supporting services (MongoDB, RabbitMQ) are proven industry-grade applications, we acknowledge that the HASTE components lack some features for reliable use in production settings, e.g., the business logic for the generation of a new HASTE Stream ID (which would be perhaps integrated with laboratory automation software) and more control over what happens to images that are corrupt or otherwise cannot be processed.

Experiments and Results

In this section we illustrate the utility of the toolkit in 2 real-world case studies chosen to demonstrate how the HASTE pipeline model can be realized in practice to optimize resource usage in 2 very different infrastructure and deployment scenarios. Table 1 summarizes the objectives of the case studies.

Case Study 1 concerns data management for a high-content screening experiment in a scientific laboratory at Uppsala University, Sweden. A small on-premises compute cluster running Kubernetes [20] provides the necessary infrastructure to handle the immediate data flow from the experiment, but both storage capacity and manual downstream analysis are concerns. We use the HASTE Toolkit to build a pipeline that captures the input data as an image stream and bins images into tiers according to image quality. The overall goal is to organize the images into tiers for subsequent processing, both to ensure that the best images are allocated to high-performance storage for high-performance analysis and to help the scientist prioritize manual work to appropriate subsets of data.

Case Study 2 concerns processing an image stream from a transmission electron microscope (TEM). During streaming, there is an opportunity to pre-process images using a desktop PC co-located with the microscope, before being uploaded for stream processing in the cloud. This is an example of an edge computing [13] scenario, where very limited but low-latency local infrastructure is leveraged together with a large cloud infrastructure. The ultimate goal is real-time control of the microscope (see Fig. 6), and consequently end-to-end latency is a key concern. This latency is constrained by image upload time. Here we develop a pipeline using the HASTE tools with an IF that predicts the effectiveness of pre-processing individual images at the edge prior to cloud upload. See Table 1 for an overview of the case studies.

Case Study 1—Smart data management for high-content imaging experiments

This case study focuses on adoption of the HASTE Toolkit in a high-content microscopy setting—the input is a stream of images arriving from an automated microscope. This deployment uses an on-premises compute cluster running Kubernetes with

Table 1: Overview of the 2 case studies used in this article

Parameter	Case Study 1—Cell profiling	Case Study 2—Real-time processing with a TEM
Application	High-content imaging	Real-time control of microscopy
Priorities	Storage	Communication and compute
Goal	Tiered storage allocation	Reduce end-to-end latency for cloud upload
Deployment setting	On-premises cloud, Kubernetes	Cloud edge & public cloud (SNIC)
Interestingness function (IF)	CellProfiler Pipeline—image quality	Estimation of size reduction (sampling, splines)
Policy	Fixed interestingness thresholds	Dynamic interestingness rank

a local NAS. While we want online analysis, we consider this a "high-latency" application—images can remain unprocessed for some seconds or minutes until compute resources are available. This is a contrast to Case Study 2, where low-latency processing is a goal.

Image quality is an issue in microscopy: images can have debris, can be out of focus, or can be unusable for some other reason relating to the experimental set-up. Such images can disrupt subsequent automated analysis and are distracting for human inspection. Furthermore, their storage, computation, and transportation have avoidable performance and financial costs.

For this case study, the HASTE Toolkit is used to prioritize storage. The developed IF is a CellProfiler pipeline performing "out of focus" prediction using the `imagequality` plugin [21]. The policy is a fixed threshold used to bin images into a DH according to image quality.

Figure 2 illustrates the key components of the architecture:

- Client—monitors the source directory for new image files, adding the name of each file to the queue (see https://github.com/HASTE-project/cellprofiler-pipeline/tree/master/client; v3 was used for this study).
- Queue—a RabbitMQ queue to store filenames (and associated metadata). Version 3.7.15 was used for this study.
- Worker—waits for a filename message (on the queue), runs a CellProfiler pipeline on it, and computes an interestingness score from the CellProfiler features (according to a user-defined function) (see https://github.com/HASTE-project/cellprofiler-pipeline/tree/master/worker; v3 was used for this study).

The deployment scripts for Kubernetes and Helm used to deploy these services for this study are available at https://github.com/HASTE-project/k8s-deployments. To replicate the results, follow the steps in the readme file.

The image, together with its interestingness score and metadata, is passed to the HASTE Storage Client, which allocates the images to Tiered Storage/DH and saves metadata in the Extracted Feature Database. Each image is processed independently, which simplifies scaling.

The HASTE Toolkit simplifies the development, deployment, and configuration of this pipeline—in particular, the interaction between the file systems used in the input image stream and archive of the processed images. When using our image-processing pipeline, user effort is focused on (i) defining a suitable IF and (ii) defining a policy that determines how the output of that function relates to DH allocation (storage tiers). Both of these are declared within the Kubernetes deployment script. When developing the pipeline itself, one is able to provide the interestingness score (the output of the IF) and the policy as arguments to the HASTE tools, and delegate responsibility to applying the policy (with respect to the storage tiers), recording all associated metadata to the Extracted Feature Database.

In this case study, the client, queue, and workers are all deployed from Docker images. However, use of neither Docker or Kubernetes is required for HASTE. Auto-scaling is configured for the workers: they are scaled up when processing images and scaled back down again when idle. A message containing the image filenames (and other metadata) is queued, but the file content is read from the NAS for processing and tiering.

The code for the worker is an extension of DistributedCellProfiler (released as part of CellProfiler v3.0; version 3.1.8 was used for this study) [22], which runs within AWS [23]. The key benefit of our containerized system is that because it runs in Docker and is not dependent on AWS services, it can be used for local deployments in laboratory settings, so that images do not need to be uploaded to the public cloud for processing. Alternatively, our system can be used with any cloud computing provider able to host Docker containers. We use the open-source message broker RabbitMQ in place of Amazon SQS (simple queue service). Our Kubernetes deployment scripts handle the necessary configuration, and a benefit of RabbitMQ is that it has a built-in management web GUI. A helper script is provided to configure the credentials for the management GUI.

Evaluation

For validation of this case study we simulated analysis and tiering using a high-content screening dataset previously collected in the laboratory, consisting of 2,699 images of cortical neuronal cells, imaged with an ImageXpress XLS; the dataset is available at [24]. In doing so, we demonstrate that our system is able to handle a large number of images. To simulate the microscope, the images were copied into the source directory, triggering messages from the client, which were read by workers to analyze the images (with CellProfiler) to extract the relevant features from the results, apply the IF, and allocate them to the tiers according to the policy. Running in our laboratory Kubernetes environment, 17 workers were able to process images simultaneously.

We use the PLLS (Power Log Log Slope) feature as the basis of our interestingness score because it has been shown to be a robust measure of image focus [25]. In this case study, we use the logistic function f as an IF, applying it to the PLLS feature x, to compute the interestingness score. The logistic function has output in the range (0,1):

$$f(x) = \frac{1}{1 + e^{-k(x - x_0)}}.$$

The PLLS values will depend on a number of factors (such as magnification, number of cells, stainings, and exposure times). The parameters of this IF can be chosen to fit the modality, based on a sample of pre-images for calibration. In this case, we chose ($k = 4.5$, $x_0 = -1.4$). The policy is defined to map the interestingness score in the intervals (i/4, (i

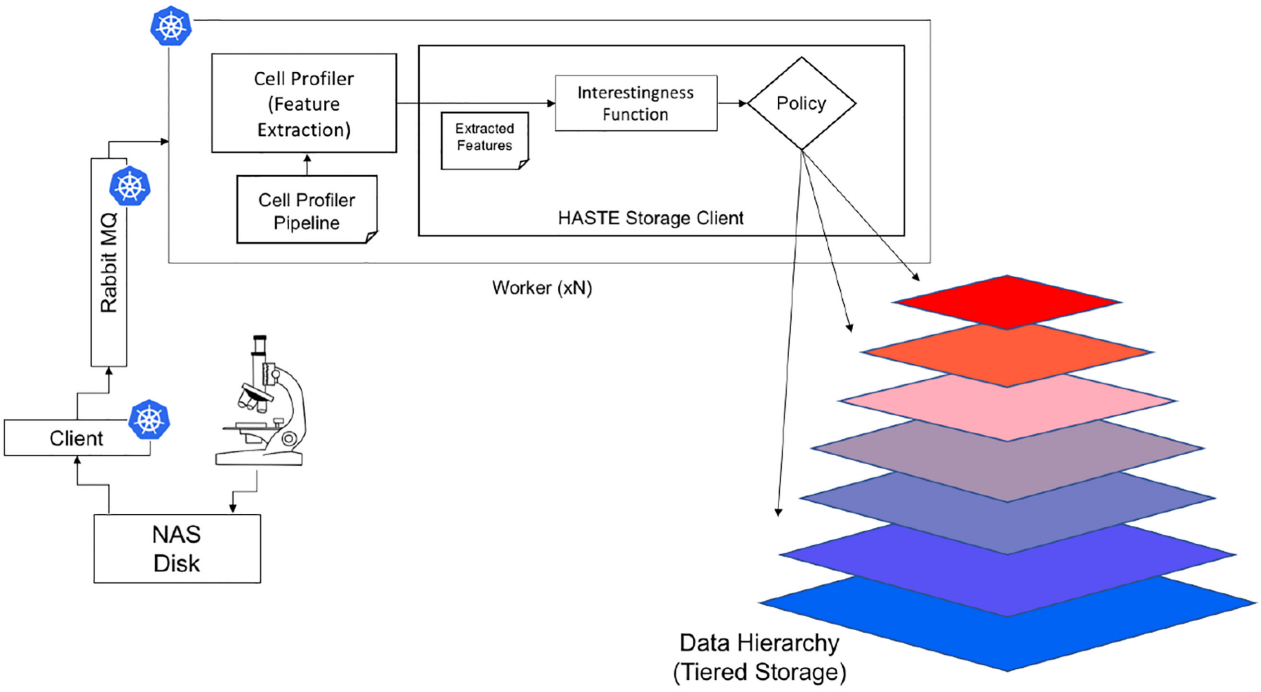

Figure 2: Architecture for Case Study 1. In this case study, the DH is realized as storage tiers. Images streamed from the microscope are saved to disk (network attached storage [NAS]). This disk is polled by the "client," which pushes a message about the new file to RabbitMQ. Workers pop these messages from the queue, analyze the image, and move it to the storage tiers configured in the data hierarchy, using the HASTE Storage Client, configured with an appropriate IF and policy. Icons indicate the components running as Kubernetes "pods."

+ 1)/4) for $i \in (0, 1, 2, 3)$ to the respective storage tiers. Figure 3 shows histograms of the PLLS feature and interestingness score.

For this evaluation, these tiers were simply directories on disk. Any storage system compatible with the HSC could be used; the key idea is that different storage platforms (with different performance and cost) can be used for the different tiers. In this case, we simply use the tiers as a convenient way to partition the dataset for further analysis and inspection. Figure 4 shows examples of the images according to tiers, and Table 2 shows the results.

```
interestingness_function(features):
    plls = features['PLLS']
    int_score = 1/(1 + exp(-(4.5) * (plls - (-1.4))))
    return int_score
```

See: https://github.com/HASTE-project/cellprofiler-pipeline/blob/master/worker/haste/pipeline/worker/LogisticInterestingnessModel.py

```
storage_policy:
 [ [0.,   0.25, tierD],
   [0.25, 0.50, tierC],
   [0.50, 0.75, tierB],
   [0.75, 1.00, tierA] ]
```

See: https://github.com/HASTE-project/k8s-deployments/blob/master/pipeline_worker.yaml

Listing 1: Pseudocode for image tier placement (Case Study 1). The IF is the logistic function, applied to the previously extracted PLLS feature. The policy shows thresholds for the different tiers.

Case Study 2—Prioritizing analysis of TEM images at the cloud edge

This case study is concerned with the prioritized processing of a stream of images from a microscope (according to an IF), applied to a hybrid edge/cloud stream-processing deployment context. In this example, we show how the HASTE tools can facilitate better use of constrained upload bandwidth and edge compute resources. The image stream comes from MiniTEM™, a 25-keV TEM [26] (Vironova, Sweden), connected to a desktop PC from which the microscope is operated and the image stream received, via proprietary driver software. The stream-processing application pre-processes the TEM images locally (i.e., at the cloud edge), to reduce their image size, with the effect of reducing their upload time to the cloud and hence the end-to-end processing latency.

The purpose of the pipeline is to automate a typical workflow for TEM analysis, which proceeds as follows: a sample is loaded into the microscope (in this case a tissue sample), and the operator performs an "initial sweep" over the sample at low magnification to locate target (i.e., interesting) regions of the sample. In the conventional workflow, the search for target areas of the sample is done by human inspection. The operator then images identified target areas of the sample at higher magnification for subsequent visual/digital analysis.

Automating this process entails the detection of target regions of the sample using an automated image-processing pipeline, based on a set of images from the initial sweep. Such a pipeline would output machine-readable instructions to direct the microscope to perform the high-magnification imaging, reducing the need for human supervision of sample imaging. The image-processing pipeline used to detect target regions can be costly and slow and could hence preferably be performed in

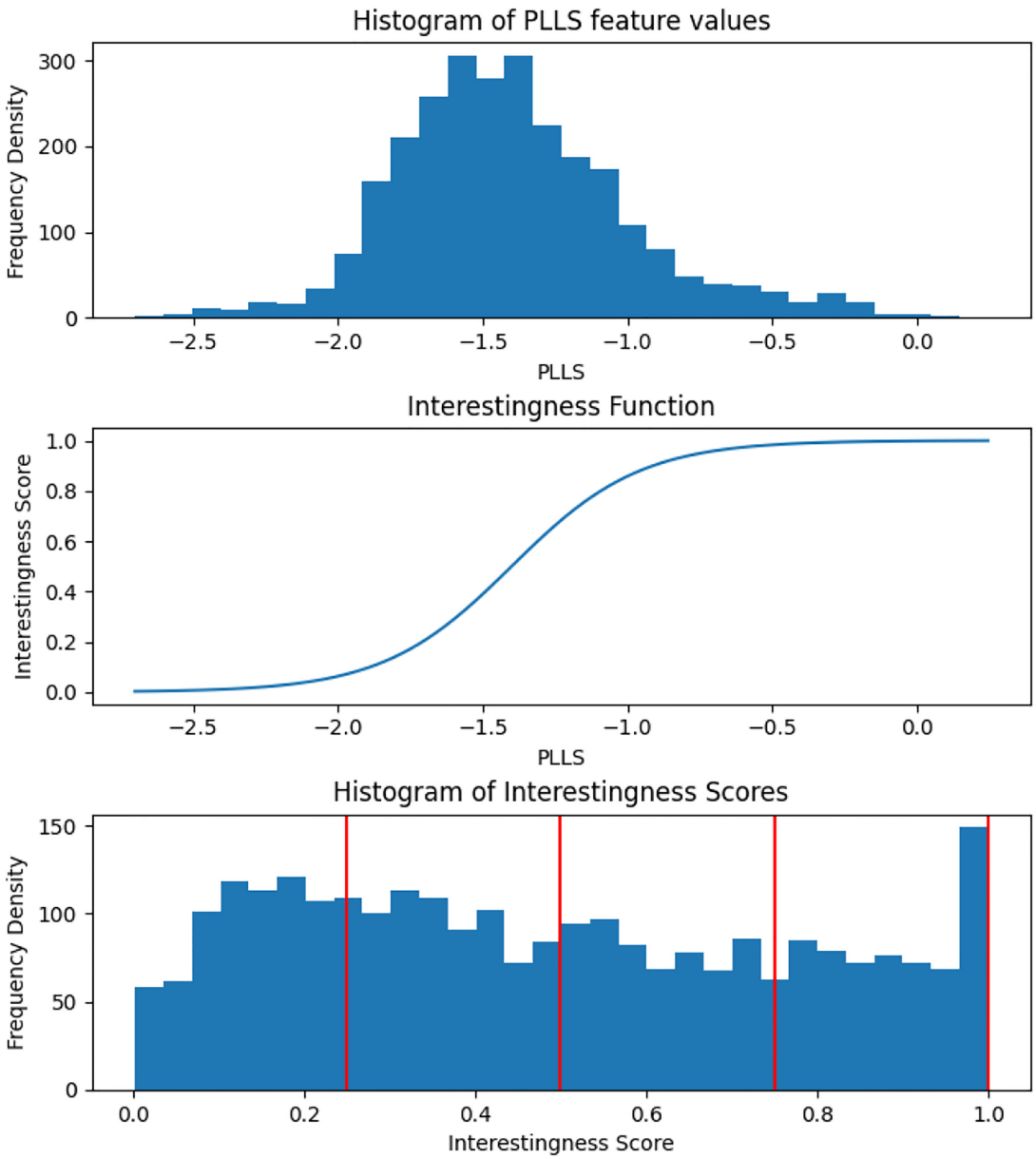

Figure 3: Histograms of the PLLS feature scores (top), and when converted to an interestingness score (bottom), by application of the Logistic Function (the IF for Case Study 1, middle). The vertical lines on the bottom plot indicate tier boundaries configured in the policy; cf. example images in Fig. 4.

Table 2: Image allocation for Case Study 1

Tier	Image count	Data (MB)
A	726	6,789
B	731	6,836
C	606	5,667
D	636	5,947
Total	2,699	25,239

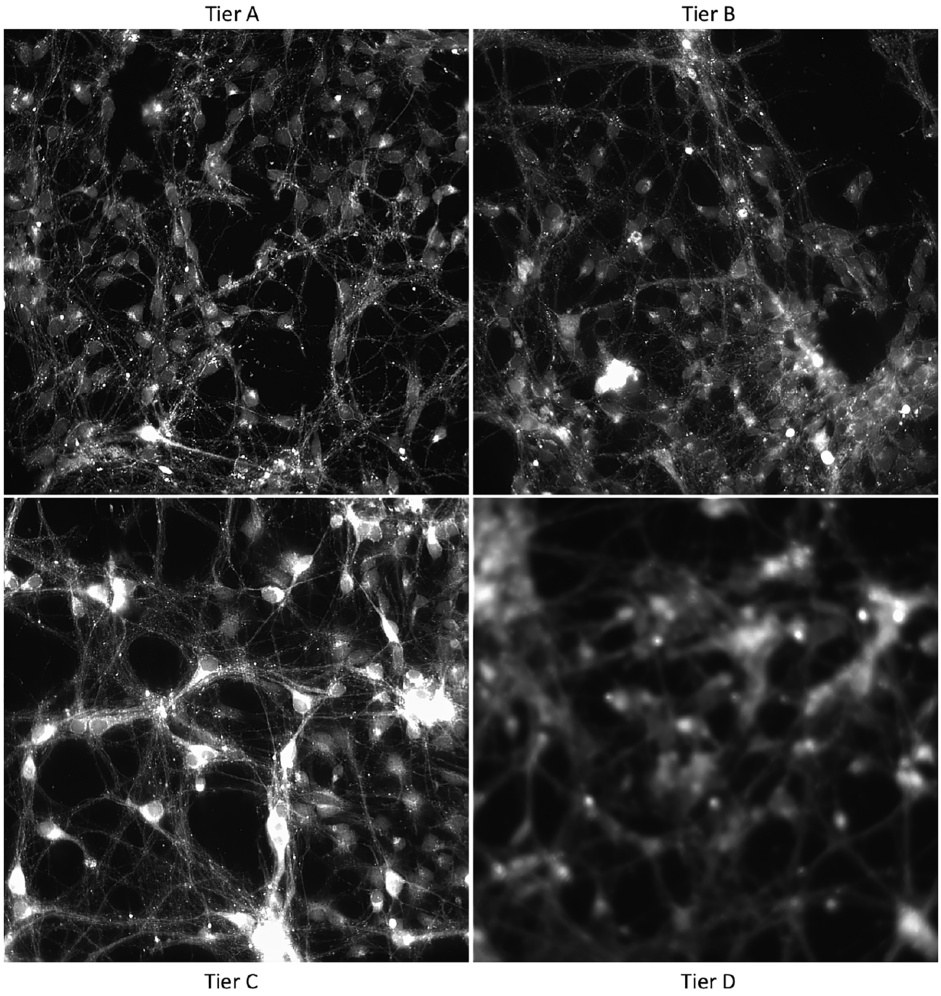

Figure 4: Example images from the high-content screening dataset (Case Study 1), according to automatically assigned tier. Tier A is the most in-focus, with the highest PLLS feature values and interestingness scores.

the cloud. Performing image processing in the cloud has several advantages: it allows short-term rental of computing resources without incurring the costs associated with up-front hardware investment and on-premises management of hardware. Machines with GPUs for deep learning, as well as secure, backed-up storage of images in the cloud, are available according to a pay-per-use model. With our overall aim of supporting a real-time control loop, and given the expense of the equipment, sample throughput is important. Despite images being compressed as PNGs, upload bandwidth is a bottleneck. Note that PNG compression is lossless, so as not to interfere with subsequent image analysis. Consequently, we wish to upload all the images from the initial sweep into the cloud as quickly as possible, and this is what is targeted here.

A pre-processing operator would reduce the compressed image size to an extent depending on the image content. However, this operator itself has a computational cost, but because of the temporary backlog of images waiting to be uploaded, there is an opportunity to pre-process some of the waiting images to reduce their size (see Fig. 5). The available upload bandwidth with respect to the computational cost of the pre-processing operator means that (in our experiment) there is insufficient time to pre-process all images prior to upload. In fact, to pre-process all of them would actually increase end-to-end latency, due to the computational cost of the pre-processing operation and limited file size reduction for some images (content dependent). The solution is to prioritize images for upload and pre-processing, respectively, whilst both processes, as well as the enqueuing of new images from the microscope, are occurring concurrently.

Feature extraction, the interestingness function, and policy

Samples for TEM analysis are typically supported by a metal grid, which then obscures (blocks) regions of the sample in (in this case) a honeycomb pattern. The blocked regions appear black in the images. As the sample holder moves under the camera, a sequence of images are captured. These images can be indexed according their position in this sequence. The extent to which the sample is obscured in a particular image is a piecewise smooth (but irregular) function of image index. This irregularity is dependent on the particular magnification level and on the speed and direction of the sample holder movement. Images can be pre-processed to remove noise from blocked regions of the image, reducing the size of the image under PNG compression. The extent of file size reduction (under our pre-processing operator) is related to the extent to which the grid obscures the image.

Consequently, the predicted extent of file size reduction can be modelled with linear spline interpolation, based on the actual file size reduction of images sampled from the queue, described

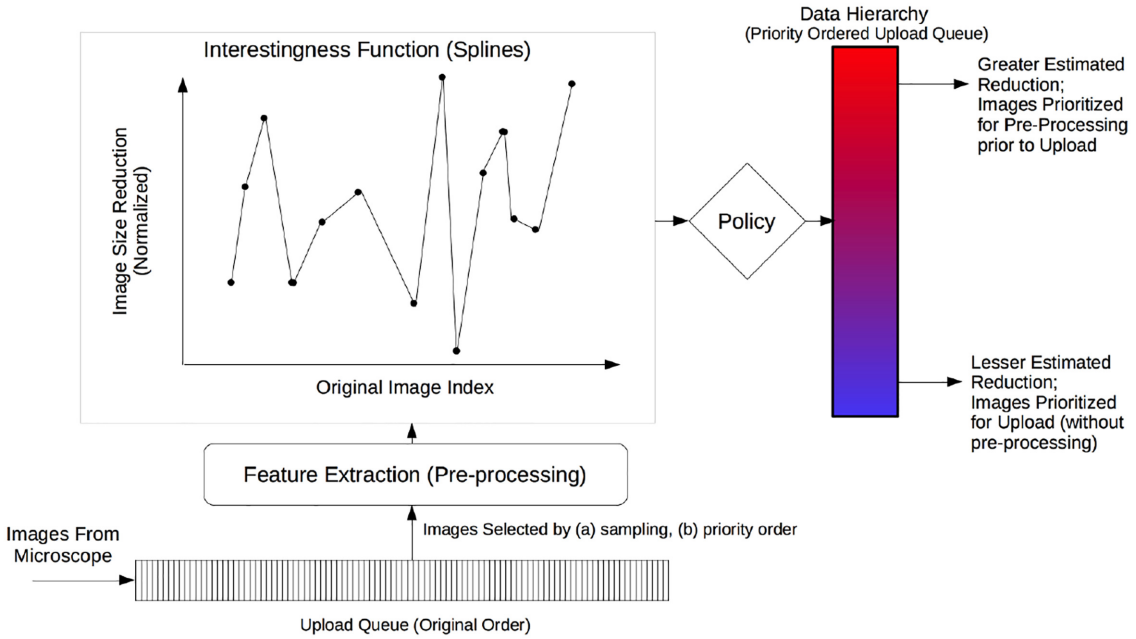

Figure 5: Architecture for Case Study 2, showing internal functionality of the HASTE Desktop Agent at the cloud edge. Images streamed from the microscope are queued at the edge for uploading after (potential) pre-processing. The DH is realized as a priority queue. Images are prioritized in this queue depending on the IF, which estimates the extent of their size reduction under this pre-processing operator: those with a greater estimated reduction are prioritized for processing (vice versa for upload). This estimate is calculated by interpolating the reduction achieved in nearby images (see Fig. 7). This estimated spline is the IF for this case study.

in more detail in [27]. The file size reduction corresponds to feature extraction in the HASTE pipeline model, and the spline estimate (the estimate of message size reduction) can be encapsulated as an IF (see Fig. 1). The HASTE tools, specifically the HASTE Agent, allow that IF to be used as a scheduling heuristic to prioritize upload and local (pre-)processing, respectively (i.e., corresponding to the policy inducing the DH in HASTE).

Available compute resources at the cloud edge are prioritized on those images expected to yield the greatest reduction in file size (normalized by the compute cost, i.e., CPU time, incurred in doing so). Conversely, upload bandwidth is prioritized on (i) images that have been processed in this way, followed by (ii) those images for which the extent of file size reduction is expected to be the least—under the aim of minimizing the overall upload time.

An important distinction between this setting and that in Case Study 1 is that the IF and DH are dynamic in this case study.

The HASTE Agent manages the 3 processes occurring simultaneously: new images are arriving from the microscope, images are being pre-processed, and images are being uploaded.

Evaluation

When evaluated on a set of kidney tissue sample images [28] our edge-based processing approach was, naturally, able to significantly reduce the end-to-end latency, when compared to performing no edge processing at all. However, our splines-based prioritization approach was able to further reduce the end-to-end latency when compared to a baseline approach for prioritization [27]. This improvement was obtained with relative ease due to the HASTE Toolkit. To reproduce this case study, follow the step-by-step guide at https://github.com/HASTE-project/haste-agent/blob/master/readme.md.

To verify the pre-processing operator, it was applied to all images after the live test was performed. Figure 7 shows how the image size reduction (y-axis–normalized with computational cost) can be modelled as a smooth function of the document index (x-axis). The colors and symbols show which images were processed prior to upload either on the basis of searching (black crosses) or on the basis of the IF and those selected for pre-processing (blue dots) and those that were not (orange crosses). As can be seen and expected there is 1 peak (the central one) where more images should optimally have been scheduled for pre-processing prior to upload. That they were not is a combination of the heuristics in the sampling strategy, and the uploading speed; i.e., they were simply uploaded before the IF (the spline estimate) was a sufficiently good estimate to schedule them for timely pre-processing. The blue line in Fig. 7 corresponds to the final spline.

Discussion

This article has discussed an approach to the design and development of smart systems for processing large data streams. The key idea of a HASTE pipeline is based on prioritization with an IF and the application of a policy. We demonstrated in 2 distinct case studies that this simple model can yield significant performance gains for data-intensive experiments. We argue that IFs (and the prioritization and binning that they achieve) should be considered more a "first-class citizen" in the next generation of workflow management systems and that the prioritization of data using IFs and policies are useful concepts for designing and developing such systems.

The ability to express informative IFs is critical to the efficiency of a HASTE pipeline. IFs are chosen by the domain expert to quantify aspects of the data to determine online prioritization. In this work we provide 2 examples of increasing complexity. In Case Study 1, the IF is a static, idempotent function of a single image, which can be checked against a static threshold to determine a priority "bin" or tier to store the image. In Case Study 2, the prioritization of the queue of images wait-

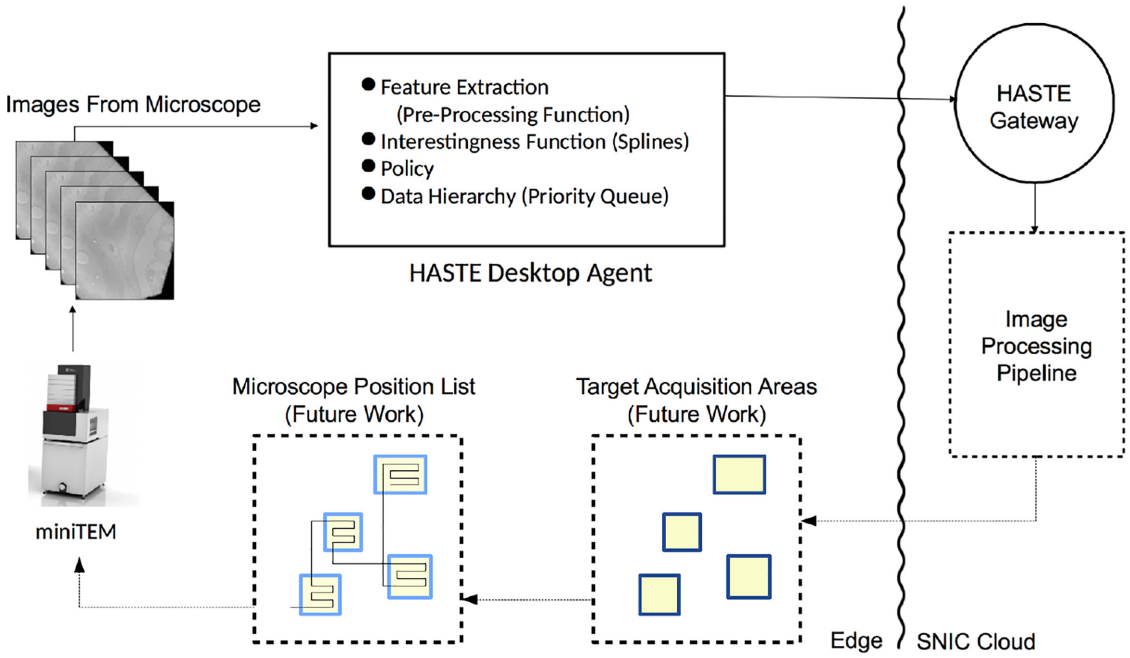

Figure 6: Architecture of the intended application: full control loop for the MiniTEM, with automatic imaging of target areas identified in initial scan. Control of microscope acquisition is future work. The internals of the HASTE Desktop Agent (where the HASTE model is applied) are shown in Fig. 5.

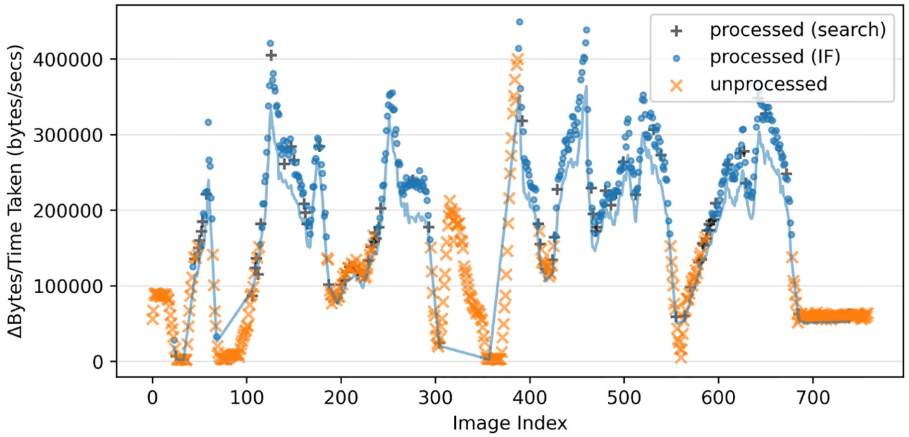

Figure 7: Image size reduction (normalized by CPU cost) over index, showing which images are processed at the edge. Those marked "processed" were processed at the cloud edge prior to upload (and vice versa)—selected either to search for new areas of high/low reduction or to exploit known areas (using the IF). The line shows the final revision of the splines estimation of the message size reduction (the IF). Note how this deviates from the true value (measured independently for illustration purposes on the same hardware) in regions of low reduction. Note the oscillating pattern, which is an artifact of movement over the grid in the miniTEM. Adapted from [27].

ing to be uploaded is revised online, as the underlying model is revised. The strength of our proposed model is that, having defined an IF, by making small changes to the policy, the user is able to reconfigure the pipeline for different deployment scenarios and datasets, with different resulting resource allocation. The HASTE Toolkit is an initial implementation of this vision. An avenue for future work will explore the creation of IFs through training in real time, using active learning and potentially also reinforcement learning.

The policy-driven approach of resource prioritization proposed under the HASTE pipeline model can be generalized to optimize utilization of different forms of constrained computational resources. In some contexts (such as Case Study 1) we are concerned with processing data streams for long-term storage, so storage requirements (and associated costs) are the key concern. In other contexts, with a focus on real-time control, automation, and robotics, the priority can be more about achieving complex analysis with low latency. In Case Study 2 this is manifest as a need to achieve edge-to-cloud upload in the shortest possible time.

The different policies for the 2 case studies reflect this: in Case Study 1, the user defines a policy to bin images according their interestingness score (i.e., image quality); these thresholds are pre-defined by the user. That is to say, the user decides explicit interestingness thresholds, and this determines the resources (in this case, storage) that are allocated and the final

cost. In similar deployment scenarios where cloud storage is used (especially blob storage) costs would depend on the number of images within each interestingness bound. Whereas in Case Study 2, by modelling the predicted extent of message size reduction as an IF within the HASTE tools, we can define a policy to prioritize image processing and upload with the goal of minimizing the total upload time for the next step in the pipeline.

These policies induce 2 forms of DH: in Case Study 2, the DH is manifest as a priority queue, updated in real time as new images arrive, are pre-processed, and eventually removed, whereas the available resources (CPU, network) are fixed. By contrast, the data hierarchy in Case Study 1 is static, defined by fixed thresholds on interestingness score; in this case, it is the resources (in this case, storage and consequent processing) that are variable, determined by how many images end up in each tier of the hierarchy.

Finally we note that the IF and policy could also be used to prioritize data based on some measure of confidence. In many scientific analyses there exists a significant amount of uncertainty in several steps of the modelling process. For example in a classification setting the class labels predicted can be highly uncertain. If in the top tier of the hierarchy we would place only those data points for which we are confident in the predicted label, downstream analysis would see a reduction in noise and an increased separability of the (biological) effects under study, as discussed in [29].

Conclusion

In this article we have proposed a new model for creating intelligent data pipelines, and presented a software implementation, the HASTE Toolkit. We have shown how these tools can be leveraged in imaging experiments to organize datasets into DHs. We have shown benefits in terms of cost reduction and performance improvement, in terms of compute resources of various kinds. In our case studies, we have studied some typical deployment scenarios and shown how prioritization can be achieved in these scenarios. Conceptualizing data analysis pipelines around IFs allows better use of various computing resources and provides a conceptual structure for us to think about the involvement of humans in such pipelines (and their monitoring), as well as a means of managing scientific experimentation—either with instruments or through simulation.

The proposed HASTE pipeline model is intended as a means of bringing structure to large scientific datasets—a means of curating a “data lake” [30] whilst avoiding creating a “data swamp” [31, 32]. It is effectively a design pattern creating a data hierarchy from “runtime knowledge” about the dataset—extracted in real time. The HASTE Toolkit is intended to help scientists achieve this. Our contribution is 2-fold: both the HASTE pipeline model as a concept, and a Python implementation. It would be possible to re-implement the design pattern in other programming languages as needed.

A key contribution made by the HASTE Toolkit is the design of an API that allows the user to express how they would like their data to be prioritized whilst hiding from them the complexity of implementing this behaviour for different constrained resources in different deployment contexts. The toolkit should allow intelligent prioritization to be ”bolted on” to new and existing systems and is consequently intended to be usable with a range of technologies in different deployment scenarios.

In the general context of big data, the HASTE Toolkit should be seen as an effort to address challenges related to data streams and efficient placement and management of data. It provides the technical foundation for automatically organizing incoming datasets in a way that makes them self-explanatory and easy to use based on the features of data objects rather than traditional metadata. It also enables efficient data management and storage based on data hierarchies using dynamic policies. This lays the foundation for domain experts to efficiently select the best-suited data from a massive dataset for downstream analysis.

Future work will focus on further improving the reliability, testing, and robustness of the software and its documentation. We also plan to investigate how the cloud function-as-a-service functionality (e.g., OpenFaaS, or AWS Lambda) could be used to further decouple IFs and allow them to be adjusted more readily.

Availability of Supporting Source Code and Requirements

The HASTE Toolkit:

- Project home page: https://github.com/HASTE-Project
- Operating systems: Linux, MacOSX
- Programming languages: Python 3
- Licence: BSD 3
- RRID:SCR_020932, https://scicrunch.org/browse/resources/SCR_020932
- biotools ID: haste_toolkit

Pipeline Example Application (Case Study 1):

- Source Code: https://github.com/HASTE-project/cellprofiler-pipeline
- Version 3, Licence: BSD 3.
- Client App, (Docker Image: benblamey/haste_pipeline_client:v1 Based on python:3.7.2 parent Docker image.)
- Worker App, (Docker Image: benblamey/haste_pipeline_worker:v3. Based on ubuntu:16.04 parent Docker image. CellProfiler v3.1.8, OpenJDK 8, Numpy 1.16.3)
- Deployment Scripts and Documentation: https://github.com/HASTE-project/k8s-deployments

Components of the HASTE Toolkit (and other dependencies) used in this article:

- HASTE Storage Client, v0.13 (BSD 3), https://github.com/HASTE-project/HasteStorageClient
- HASTE Gateway - v0.10, Python 3, (BSD 3) benblamey/haste-gateway:latest, https://github.com/HASTE-project/haste-gateway
- HASTE Agent, v0.1, Python 3, (BSD 3) https://github.com/HASTE-project/haste-agent
- RabbitMQ, Version 3.7.15, Erlang 22.0, https://www.rabbitmq.com/
- MongoDB, Version 4.0.9, https://www.mongodb.com/

Data Availability

An archival copy of the code (HASTE toolkit and pipeline example applications) is available via the GigaScience repository, GigaDB [33].

Abbreviations

API: application programming interface; AWS: Amazon Web Services; CPU: central processing unit; DH: data hierarchy; GPU:

graphics processing unit; GUI: graphical user interface; HASTE: Hierarchical Analysis of Spatial Temporal Data; HSC: HASTE Storage Client; IF: interestingness function; IoT: Internet of Things; MQTT: Message Queuing Telemetry Transport; NAS: network-attached storage; PLLS: Power Log Log Slope; SNIC: Swedish National Infrastructure for Computing; TEM: transmission electron microscope.

Competing Interests

The authors declare that they have no competing interests.

Funding

The HASTE Project (Hierarchical Analysis of Spatial and Temporal Image Data, http://haste.research.it.uu.se/) is funded by the Swedish Foundation for Strategic Research (SSF) under award No. BD15-0008 and the eSSENCE strategic collaboration for eScience.

Authors' Contributions

B.B.: Conceptualization, Project Administration, Investigation, Formal Analysis, Software, Methodology, Validation, Data Curation, Writing - Original Draft, Writing - Review & Editing, Visualization.
S.T.: Conceptualization, Methodology, Writing - Original Draft, Writing - Review & Editing, Funding Acquisition.
M.D.: Software, Methodology.
H.W.: Methodology, Writing - Review & Editing
P.H.: Methodology, Writing - Review & Editing.
I.M.S.: Conceptualization, Investigation, Software, Methodology, Resources, Writing - Original Draft, Writing - Review & Editing, Visualization.
A.S.: Conceptualization, Supervision, Methodology, Writing - Original Draft, Writing - Review & Editing.
C.W.: Conceptualization, Supervision, Project Administration, Methodology, Validation, Funding Acquisition, Writing - Original Draft, Writing - Review & Editing, Visualization.
O.S.: Conceptualization, Supervision, Project Administration, Methodology, Validation, Resources, Funding Acquisition, Writing - Original Draft, Writing - Review & Editing, Visualization.
A.H.: Conceptualization, Supervision, Project Administration, Software, Methodology, Validation, Funding Acquisition, Writing - Original Draft, Writing - Review & Editing, Visualization.

Acknowledgments

Thanks to Anders Larsson and Oliver Stein for help with software deployment and testing for Case Study 1. Thanks to Polina Georgiev for providing the images used in the evaluation of Case Study 1. Resources from The Swedish National Infrastructure for Computing (SNIC) [34] were used for Case Study 2.

References

1. Ouyang W, Zimmer C. The imaging tsunami: computational opportunities and challenges. Curr Opin Syst Biol 2017;**4**:105–13.
2. Stephens ZD, Lee SY, Faghri F, et al. Big data: astronomical or genomical?. PLoS Biol 2015;**13**(7):e1002195.
3. Blamey B, Wrede F, Karlsson J, et al. Adapting the secretary hiring problem for optimal hot-cold tier placement under top-K workloads. In: 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID) Larnaca, Cyprus; 2019:576–583.
4. Sivarajah U, Kamal MM, Irani Z, et al. Critical analysis of big data challenges and analytical methods. J Bus Res 2017;**70**:263–86.
5. Reinsel D, Gantz J, Rydning J. Data Age 2025: The Digitization of the World from Edge to Core (Seagate White Paper); 2018. https://www.seagate.com/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf. An IDC White Paper – #US44413318. Accessed: April 2020
6. Rinehart D, Johnson CH, Nguyen T, et al. Metabolomic data streaming for biology-dependent data acquisition. Nat Biotechnol 2014;**32**(6):524–7.
7. Hillman C, Petrie K, Cobley A, et al. Real-time processing of proteomics data: the internet of things and the connected laboratory. In: 2016 IEEE International Conference on Big Data; 2016:2392–9.
8. Zhang Y, Bhamber R, Riba-Garcia I, et al. Streaming visualisation of quantitative mass spectrometry data based on a novel raw signal decomposition method. Proteomics 2015;**15**(8):1419–27.
9. Kelleher J, Lin M, Albach CH, et al. Htsget: a protocol for securely streaming genomic data. Bioinformatics 2019;**35**(1):119–21.
10. Cuenca-Alba J, del Cano L, Gómez Blanco J, et al. ScipionCloud: an integrative and interactive gateway for large scale cryo electron microscopy image processing on commercial and academic clouds. J Struct Biol 2017;**200**(1):20–7.
11. de la Rosa-Trevín JM, Quintana A, del Cano L, et al. Scipion: a software framework toward integration, reproducibility and validation in 3D electron microscopy. J Struct Biol 2016;**195**(1):93–9.
12. Wang D, Fong S, Wong RK, et al. Robust high-dimensional bioinformatics data streams mining by ODR-ioVFDT. Sci Rep 2017;**7**(1):43167.
13. Shi W, Dustdar S. The promise of edge computing. Computer 2016;**49**(5):78–81.
15. Blamey B, Hellander A, Toor S. Apache Spark Streaming, Kafka and HarmonicIO: a performance benchmark and architecture comparison for enterprise and scientific computing. In: Gao W, Zhan J, Fox G , et al., eds. Benchmarking, Measuring, and Optimizing: Bench 2019, Denver, CO, USA. Cham: Springer; 2019, doi:10.1007/978-3-030-49556-5˙30.
16. Torruangwatthana P, Wieslander H, Blamey B, et al. HarmonicIO: scalable data stream processing for scientific datasets. In: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA. IEEE; 2018:879–82.
17. Awesome Pipeline. https://github.com/pditommaso/awesome-pipeline.Accessed: March 2020
18. Chan SG, Tobagi FA. Modeling and dimensioning hierarchical storage systems for low-delay video services. IEEE Trans Comput 2003;**52**(7):907–19.
19. Novella JA, Emami Khoonsari P, Herman S, et al. Container-based bioinformatics with Pachyderm. Bioinformatics 2019;**35**(5):839–46.
20. Kubernetes. Kubernetes Documentation. https://kubernetes.io/docs/home/. Accessed May 2020
21. Bray MA, Carpenter AE. Quality control for high-throughput imaging experiments using machine learning in Cellprofiler. In: Johnston PA, Trask OJ . High Content Screening: A Powerful Approach to Systems Cell Biology and Phenotypic

Drug Discovery Methods in Molecular Biology. New York, NY: Springer; 2018:89–112.
22. McQuin C, Goodman A, Chernyshev V, et al. CellProfiler 3.0: next-generation image processing for biology. PLoS Biol 2018;**16**(7), doi:10.1371/journal.pbio.2005970.
23. Distributed-CellProfiler. https://github.com/CellProfiler/Distributed-CellProfiler. Accessed May 2020
24. Polina G, Blamey B, Ola S. Snat10 Knockout Mice Cortical Neuronal Cells (ImageXpress XLS Example Images). 2020. http://doi.org/10.17044/scilifelab.12811997.v1.
25. Bray MA, Fraser AN, Hasaka TP, et al. Workflow and metrics for image quality control in large-scale high-content screens. J Biomol Screen 2012;**17**(2):266–74.
26. Vironova AB. MiniTEM: Automated Transmission Electron Microscopy Analysis. https://www.vironova.com/our-offering/minitem/. Accessed June 2020
27. Blamey B, Sintorn IM, Hellander A, et al. Resource- and message size-aware scheduling of stream processing at the edge with application to realtime microscopy. arXiv 2019:1912.09088.
28. Blamey B, Ida-Maria S. HASTE miniTEM Example Images (Dataset). 2020. https://doi.org/10.17044/scilifelab.12771614.v1.
29. Wieslander H, Harrison PJ, Skogberg G, et al. Deep learning and conformal prediction for hierarchical analysis of large-scale whole-slide tissue images. IEEE J Biomed Health Inform 2021;**25**(2):371–80.
30. Dixon J. Pentaho, Hadoop, and Data Lakes. 2010. https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/. Accessed February 2020
31. Brackenbury W, Liu R, Mondal M, et al. Draining the data swamp: a similarity-based approach. In: Proceedings of the Workshop on Human-In-the-Loop Data Analytics HILDA'18, Houston, TX, USA. New York: ACM; 2018, doi:10.1145/3209900.3209911.
32. Hai R, Geisler S, Quix C. Constance: an intelligent data lake system. In: Proceedings of the 2016 International Conference on Management of Data SIGMOD '16, San Francisco, CA, USA. New York: ACM; 2016: 2097–100.
33. Blamey B, Salman T, Martin D, et al. Supporting data for "Rapid development of cloud-native intelligent data pipelines for scientific data streams using the HASTE Toolkit." GigaScience Database 2021. http://dx.doi.org/10.5524/100872.
34. Toor S, Lindberg M, Falman I, et al. SNIC Science Cloud (SSC): a national-scale cloud infrastructure for Swedish Academia. In: E-Science (e-Science), 2017 IEEE 13th International Conference On IEEE, Auckland. IEEE; 2017: 219–27.