

Towards Optimising Distributed Data Streaming Graphs using Parallel Streams

C.S. Liew
National e-Science Centre
School of Informatics
University of Edinburgh
c.s.liew@sms.ed.ac.uk

M.P. Atkinson
e-Science Institute
University of Edinburgh
mpa@nesc.ac.uk

J.I. van Hemert
National e-Science Centre
School of Informatics
University of Edinburgh
j.vanhemert@ed.ac.uk

L. Han
National e-Science Centre
School of Informatics
University of Edinburgh
liangxiu.han@ed.ac.uk

ABSTRACT

Modern scientific collaborations have opened up the opportunity of solving complex problems that involve multidisciplinary expertise and large-scale computational experiments. These experiments usually involve large amounts of data that are located in distributed data repositories running various software systems, and managed by different organisations. A common strategy to make the experiments more manageable is executing the processing steps as a workflow. In this paper, we look into the implementation of fine-grained data-flow between computational elements in a scientific workflow as streams. We model the distributed computation as a directed acyclic graph where the nodes represent the processing elements that incrementally implement specific subtasks. The processing elements are connected in a pipelined streaming manner, which allows task executions to overlap. We further optimise the execution by splitting pipelines across processes and by introducing extra parallel streams. We identify performance metrics and design a measurement tool to evaluate each enactment. We conducted experiments to evaluate our optimisation strategies with a real world problem in the Life Sciences—EURExpress-II. The paper presents our distributed data-handling model, the optimisation and instrumentation strategies and the evaluation experiments. We demonstrate linear speed up and argue that this use of data-streaming to enable both overlapped pipeline and parallelised enactment is a generally applicable optimisation strategy.

Keywords

Scientific Workflows; Parallel Stream; Optimisation

1. INTRODUCTION

It is widely recognised that data-intensive methods are transforming the way research is conducted, as recognised in “*The Fourth Paradigm*” [21]. There are many reports identifying requirements for data-intensive computation [20, 6, 37, 23]. It is evident that improved apparatus is needed for extracting information, evidence and knowledge from the growing wealth of data. We introduce the term “datascope” for the instruments that reveal information latent in data, just as telescopes reveal the universe. Several software frameworks for composing datascope are possible: workflow technologies, map-reduce systems, query engines and desk-top tools. Each framework is subject to active research, in this paper we focus on a subclass of workflows where data is passed using data streams.

We investigate the potential for optimisation for data mining and data integration (DMI) processes to reduce the overall workflow execution time by making the best use of data streaming. We model each request as a directed acyclic graph (DAG). Tasks are handled by software components named *processing elements (PEs)* and represented as nodes in the graph. The nodes are connected in a pipelined streaming manner, which allows the overlap of PEs’ executions—a source PE continues data production while a consuming PE consumes its output permitting some of the PEs to be executed simultaneously on different portions of a data stream. The streaming technology can process requests with large-scale data by an efficient implementation of buffering in main memory. Jacobs [24] has observed that the processing speeds of memory access outperforms disk by a factor of $> 10^5$. This ratio increases when random access is needed. Section 2 describes the workflow model in detail.

Our hypothesis is that such distributed computations can be optimised if we understand the significant costs that affect streaming performance. We hypothesise candidate performance factors and examine how these relate to various cost functions. Some metrics are generic such as response time and throughput whereas others are more important for specific types of request. For instance, in a distributed computation that involves high volumes of data movement be-

tween geographically distributed data repositories, the data transmission costs may be the critical factor. Section 4.1 provides further discussion of performance metrics. We proposed a measurement framework in Section 4.2 to capture performance data when executing distributed computations at different levels of abstraction.

We propose an optimisation strategy that splits a DAG into multiple DAGs that can then be passed to separate enactment engines coupled by potentially slower data streams. The challenge is to partition automatically the initial DAG in a way that minimises a cost function. We use data collected by measurement probes to inform this decision.

In preparation for the automated optimisation, we conducted the following manual procedure.

1. Instrumented the DAG
2. Collected internal data flows and timing measurements
3. Examined the data and choose a candidate graph cut
4. Bridged the cut with long-haul data streaming
5. Measured the performance with the parts distributed across enactment engines
6. Explored various partitions and mappings to enactment engines.

The intention of these manual experiments is to establish the potential for optimisation, to identify strategies and heuristic, and to set optimisation goals.

We have conducted an experiment to evaluate our optimisation strategies in solving a real-world problem in the Life-Sciences. An experiment with EURExpress-II [22] using OGSA-DAI [14] described in Section 5. OGSA-DAI is an extensible framework which supports data streaming of heterogeneous data from multiple sources, such as: relational, XML and RDF databases, and file systems. OGSA-DAI provides an extensive library of *activities* as basic building blocks of distributed queries and workflows; they perform functions, such as: executing queries, reading files, transforming data, etc. The experimental results in Section 6 shows that a linear speed up is obtained using the proposed optimisation strategy.

The paper is structured as follows: Section 2 describes the streaming model and the optimisation problem. Section 3 presents our proposed optimisation strategy. Section 4 discusses the measurement technology. Section 5 describes the experimental setup and the real application. The results are discussed in Section 6 and related work in Section 7. We conclude and discuss further research in Section 8.

2. PROBLEM DESCRIPTION

2.1 Streaming Model

A typical distributed computation comprises a sequence of tasks that represent steps in a computational process that composes data and operations that may be independently defined. We call this a distributed data streaming graph. Such a graph can be control-driven or data-driven: the former has dependencies to show the execution ordering or control flow while the latter represents the flow of data from one task to another and execution ordering is inferred. Workflow structure is also different according to workflow engine,

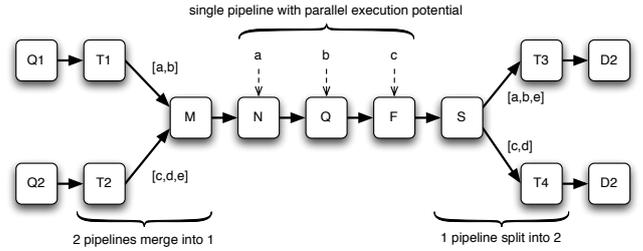


Figure 1: Pipeline example

scheduling method and enactment platform. We focus on workflows where the data flow is explicit and the control flow implicit, represented as a DAG. Let $G = \{V, E\}$ denote a DAG consisting of n steps, the set of tasks is represented by vertices, V and the edges, E denote data flows from a source task to consumer task. Figure 1 illustrates a DAG that comprises 13 tasks $V = \{Q1, Q2, T1, \dots, D1, D2\}$ with edges representing the data flow. For example, Q1 retrieves data and passes them to T1 to perform transformation.

A task is handled by executing one or more software components named *processing elements (PEs)*. PEs are connecting via *data streams* to form a DAG.

Characteristics of PEs:

- PEs have *input(s)* to receive data and *output(s)* to send data.
- PEs have different data processing rates, e.g., T1 and T2 may take different amounts of time to transform a unit of data.
- PEs may have different input consumption rates, e.g., if M is a sort merge, it may consume data from one input much faster than from the other.
- PEs start to process as soon as they have received sufficient data for the computation. They may emit data as soon as the processing on a unit of input has finished.
- Some PEs are *aggregative*, that is, they combine data from a (sub-)sequence of (sub-)units in their input to produce a single derived value in its output.
- The relationship between inputs and outputs may be specified, e.g., *a*) that a PE consumes lists from its input and generates a tuple for each list on its output that is an aggregation of the list, *b*) that a PE takes lists of tuples on its input and emits corresponding lists of tuples with the tuples partitioned between the outputs or *c*) that a PE takes lists of tuples on input *a* and tuples on input *b* consuming one list and one tuple at each step and emitting a list of tuples which are the original tuples from *a* extended by the tuples from *b*.
- A PE ceases processing when an input it requires has signalled it has no more data or when all of its consumers have indicated the no longer require data, or when it is sent a stop signal by the enactment system.

Characteristics of data streams:

- Due to different consumption rates of PEs, buffering is needed in a data stream.
- Streaming buffers can be implemented in main memory or spill onto disk.
- When a data stream connecting two PEs resides on separate machines, the stream implementation uses communication protocols.

2.2 Optimisation Problem

The DAG shown in Figure 1 is a common pattern in e-Science research which involves *a)* integrating data from distributed repositories, *b)* processing these data on distributed computing elements, and *c)* formatting and presenting the results to various targets. The execution of this DAG relies on various parameters, such as:

1. *Data source selection* — From which data repositories Q1 and Q2 select from?
2. *Enactment platform selection* — On which computing platforms are the PEs executed? Is the workload fairly distributed across the computing platforms?
3. *Pipelining* — Can a successor task start on the partial results of a predecessor task before it has completed?
4. *Parallelism* — Can some of the tasks be executed in parallel, e.g., over partitions of a unit of input?
5. *Data movement* — Can certain tasks be co-located to reduce data movement or should transfer costs be introduced to prevent two PEs competing on the same processor?
6. *Tasks sequence ordering* — Will it speed up the execution time if F (filtering based on column c) is executed before Q (quantising column b), and will it produce the same results for all inputs?

The optimisation challenge is to find a way of organising the distributed computation that will deliver the same results within the application-dependent criteria at the least cost. Various cost functions may apply, e.g., time to initial output, time to completed output on all delivery streams or amount of energy used. For example, by hand optimisation of an Astronomical application, Montage, Singh *et al.* were able to reduce total execution time by a factor of 10 [34].

Most of the workflow optimisation research focuses on improving *time-based criteria*: reducing the makespan, decreasing the response time, etc. The primary goal is to improve efficiency from the user's perspective. Alternatively, optimisation can address the overall *resource efficiency* perspective so that throughput can be increased, e.g., by using resource reservation and prediction strategies. With pay-as-you-go computing services, e.g., cloud computing, optimisation may be reformulated to reduce the *charge* or *energy* for a job.

Both response and throughput are affected by failure and overload, thus optimisation may consider how well systems handle failure, e.g., by providing alternative execution paths, cleaning up and recovering partial work after failures and re-submitting a modified workflow to complete the failing job. Similarly, it is necessary to handle increases in data volume, number of jobs and both data and workflow complexity without severe degradation.

Most users appreciate more abstract notations for specifying the workflows as they can then focus on the goals of their work. Abstraction imposes a requirement for optimisation to achieve acceptable performance and provides an opportunity for optimisation during the automated mapping to more concrete representation.

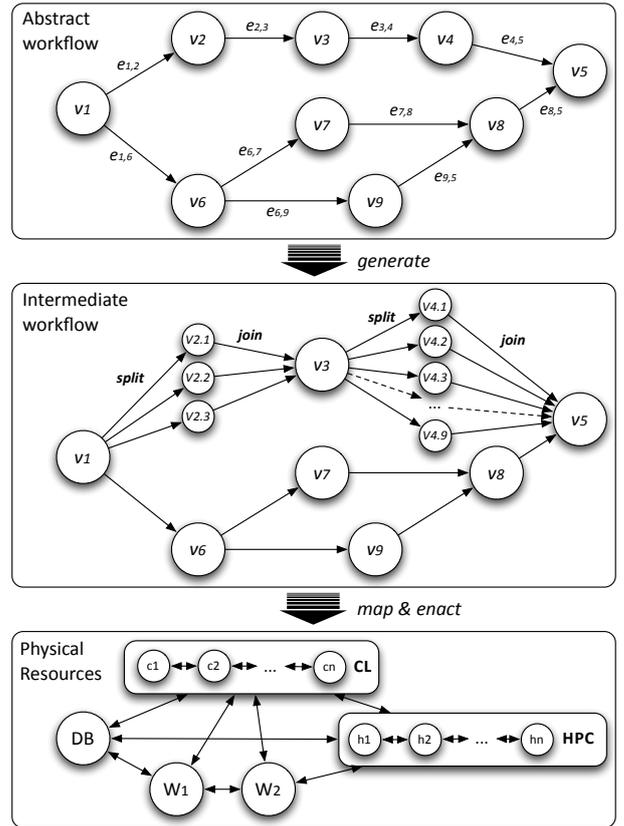


Figure 2: Mapping abstract workflow onto physical resources

3. OPTIMISATION STRATEGY

We propose an optimisation strategy that splits a DAG into multiple DAGs that can then be passed to separate enactment engines coupled by potentially slower data streams. The proposed strategy aims to minimise the total workflow cost within constraints imposed by data sources and destinations by:

1. ameliorating performance bottlenecks by parallelising,
2. distributing enactment to balance workload,
3. minimising data movement by co-locating, and
4. performing logical transformation.

The DAG is split into partitions and data staging PEs are added to handle data movement between platforms. The challenge is to partition automatically the DAG in a way that minimises a cost function.

Figure 2 illustrates how to map an abstract workflow comprising 9 tasks onto physical resources. We construct a *weighted DAG* from the given abstract DAG, D based on previously collected performance data. The weight of a node v_i is the *computational cost per unit of data* of a particular task, w_i . The weight of an edge $e_{i,j}$ that connects v_i and v_j represents the *communication cost per unit of data* by the data stream between the connected tasks. The DAG is split into k partitions, P and mapped onto heterogeneous platforms, R that consist of database server DB , workstations W_1 and W_2 , commodity cluster CL and high performance computing cluster HPC . Assume that v_3 produces a large

volume of data, and both w_3 and w_4 are high, then optimisation should map v_3 and v_4 onto *HPC*.

When optimising workflows in a pipelined streaming model, the optimiser must handle both the differences in processing behaviour and data streaming rates. As described in our previous work [4], if a PE reads all of its input tuples, and requires multiple accesses to each tuple (referred as *aggregative behaviour*), it requires the entire data to be held in memory, or repeatedly re-read from disk. If v_8 requires tuples from both inputs, e.g., a merge or join, and v_7 produces data faster than v_9 , then the stream buffer in $e_{7,8}$ will overflow to disk or have large memory requirements on the system.

The optimiser must discover the potential of a particular path in the DAG to be parallelised. For instance, if the work load of v_2 and v_4 are high, therefore splitting the data stream as multiple instances of v_2 and v_4 is beneficial. This involves two decisions: *a*) where to split the data stream, and where the data stream should be merged, and *b*) how many split instances are needed to balance data rates.

Optimising the distributed computations in a pipelined streaming model, across distributed and heterogeneous computational and data resources is a hard problem. We solve this problem in stages—the preliminary stage adopts the following assumptions, which are progressively relaxed:

1. requests tend to be repeated, either:
 - (a) entire workflows with different dataset;
 - (b) partial workflows (e.g., a sub-workflow that performs image preprocessing);
 - (c) PEs (e.g., PE that performs SQL query on relational databases),
2. enactments can be automatically instrumented, and performance data collected and stored,
3. the semantic descriptions of all PEs are stored in a *registry*, and updating operations are allowed (e.g., adding tags for optimisation),
4. data repositories are accessible across the network using selected data-integration middleware,
5. requests are data-driven, and involve common scientific type data, e.g., the recursive composition of collections (lists, sets, bags, and trees), tuples, arrays, images, and primitive types—we pass parts of these incrementally along streams to allow them to be arbitrarily large.

Our optimisation depends on the operational model shown below. We first check whether the DAG (or a subDAG) has been enacted before, and retrieve information from the *Performance Database (PDB)*. We annotate each v_i with semantic information from the *Registry (Reg)* (e.g. data unit, type structure of each input and output, parallelisable) and derived performance summaries. We then identify the *critical paths (CPs)* in the DAG. The DAG in Figure 2 indicates three execution paths, i.e. $[v_1, v_2, v_3, v_4, v_5]$, $[v_1, v_6, v_7, v_8, v_5]$ and $[v_1, v_6, v_9, v_8, v_5]$. The path with the slowest processing rate is defined as the *CP*. The optimiser focuses on optimising the *CPs*. How can it find the putative *CPs*?

We identify the critical path by instrumenting buffer for multi-input PEs, e.g., v_5 has two input streams ($e_{4,5}$ and

Read abstract DAG (D)

```

for all subDAG  $D'$  of  $D$  do
  if  $D'$  previously processed then
    Replace  $D'$  with previous optimisation  $D''$ 
    Annotate  $D''$  with semantic information from Reg
    and performance data from PDB
  else
    Annotate  $D'$  with semantic information from Reg
    and performance data from PDB
  end if
end for

Identify & Optimise Critical Paths(CPs) to produce  $D'''$ 
Instrument  $D'''$  to yield  $D''''$ 
Enact  $D''''$ 
Update PDB

```

$e_{8,5}$), thus, its performance depends on the rates of these streams. $e_{4,5}$ and $e_{8,5}$ should be instrumented. If the rates of $e_{8,5}$ is faster than $e_{4,5}$, the buffer space in $e_{8,5}$ will fill. Thus, the execution path $[v_1, v_2, v_3, v_4, v_5]$ is the critical path that needs to be optimised, with the following approaches.

Splitting data stream and executing in parallel. The split may be *horizontal* (using DB parlance) where different tuples pass along parallel paths, or *vertical*, where different elements of a tuples pass along parallel paths. By combining the semantic annotation along the *CPs*, we determine which optimisations are permissible. The final choice is based on summation of the recomputed costs, including *a*) split and merge operations, *b*) data transfers, and *c*) sorts to reconstruct order when necessary.

Splitting workflows and moving PEs to different platforms. Enacting a workflow in streaming manner on one platform may cause competition for CPU cycles, memory and bandwidth. For instance, running v_6, v_7, v_8 and v_9 on W_1 causes overload while W_2 is under utilised. We can gain speed by cutting the workflow into 2 partitions, and enacting on both platforms. We identify which PEs are anchored to a specific platform, e.g., because they access local data, and then decide which edge(s) to cut based on a heuristic function that finds an optimum within constraints, e.g.:

1. *Computational load of enactment engines.* Co-locating PEs with high computational cost per unit data may generate overload.
2. *Communication cost of data movement.* Cutting at data stream that involves large volumes of data may incur high costs.
3. *Data production rate of PEs.* Co-locating PEs with low data production rates while moving PEs with high production rates will show no overall gain.

These constraints may conflict, e.g., co-locating two PEs to reduce data movement costs, may conflict with splitting PEs to balance workload.

Transposing PEs. Some DAGs can be optimised by changing the PEs execution order. This is a common optimisation approach found in query optimisers. For instance, assume that v_3 is a projection PE and v_4 is a selection PE. Trans-

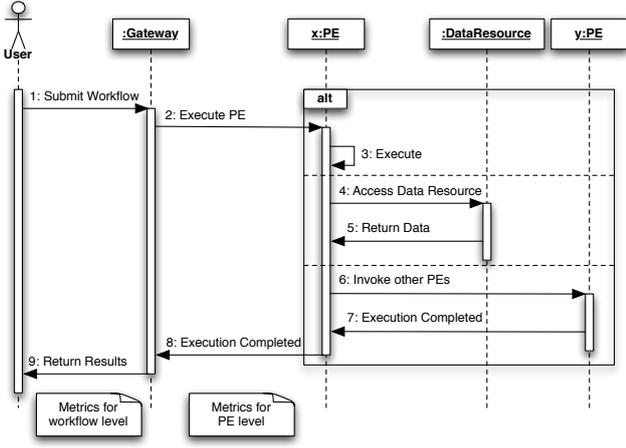


Figure 3: Multi-levels abstraction for performance analysis during workflow enactment in ADMIRE context

posing v_3 with v_4 will reduce the data size that needs to be process by v_3 , thus shorten the workflow enactment time. This requires both logical and performance information to be obtained from the *Registry* and *PDB* respectively ad used to determine logical equivalence.

Our optimisation approach is based on incremental learning from previous executions. Thus, performance data will be collected for every single run. Before we enact the optimised DAG, we instrument it by placing measurement probes in the workflow. The data collected by the measurement framework (see Section 4) after the enactment step is stored in *PDB*. In this paper, we focus on splitting data streams. Section 5 describes how we optimise a Life-Sciences application using this approach.

4. MEASUREMENT FRAMEWORK

4.1 Performance Metrics

The diversity and complexity of the scientific workflows has increased the difficulty of doing performance analysis. Truong *et al.* in [39] introduced a hierarchical abstraction for the performance analysis of workflows and identified performance metrics for different levels of abstraction in a workflow. We incorporated their model into measurement framework. We classify the performance metrics into two levels of abstraction: *request level* and *processing element level*. We define different performance metrics for both levels. Figure 3 illustrates the levels of abstraction in executing a workflow in the ADMIRE context. ADMIRE [2]—Advanced Data Mining and Integration Research for Europe—is a European-funded project that is pioneering architecture and models that deliver a coherent, extensible and flexible framework to facilitate exploration and exploitation of data. A preliminary version of the ADMIRE architecture was reported in [4].

Over the years, many studies have been conducted to understand the performance model and quality of services enacting workflows. Various performance metrics are proposed and used. Some of the metrics are commonly used in most of the studies, such as response time and throughput [26], while others provide a specific performance analysis. For in-

stance, data transfer time is used where workflows involve distributed data sources or spatially-aware optimisation [28]. In deciding what metrics to be used in our work, we consider whether the metrics: *a)* show impact of our optimisation (streaming and parallelisation), *b)* are feasible in our use case, e.g., queue waiting time is ignored in our experiment because no batch processing is involved, but is applied on heavily utilised machines, e.g., to evaluate workflow scheduler with Batch Queue Wait Time Prediction [29]. The sections below list metrics selected for our measurement framework.

4.1.1 Metrics for workflow level

We define workflow execution time as the period between the time a user submits a workflow to the ADMIRE gateway (a service that processes workflow requests and manage the workflows enactment) and receives the execution results, which includes the overall computation time for the workflow processes, the time spent in IO and waiting in process queues, and any data delivery delays.

4.1.2 Metrics for processing element level

The processing time for a PE can be measured by subtracting the time when the PE received the first block of data from the time when the last block of data has been sent out. For instance, the processing time of the SQL query PE is time spent between the receipt of the SQL expression and the delivery of last block of data to the next PE. Data volume is the total amount of data handled by a PE. There are two types of measurements that can be used here, i.e., block and tuple. The data throughput reflects the performance of a PE and is measured in terms of tuples or blocks per second. Table 1 summaries the metrics for PE level.

4.1.3 Metrics for parallel execution

Through the optimisation process, a DAG is split into multiple sub-DAGs that are then executed in parallel. Thus, we have selected two common performance indicators for parallel execution, namely *speedup* and *efficiency*.

- Speedup(S): a ratio between the execution time of a workflow (T_s) on a single processor and the execution time on multiple processors (T_m), represented as:

$$S = \frac{T_s}{T_m} \quad (1)$$

- Efficiency(E): a relationship between the speedup (S) and the number of processors (P_n) used. It is defined as:

$$E = \frac{S}{P_n} \quad (2)$$

4.2 Measuring Tools

In order to capture the performance data on every level of abstraction as discussed above, we have designed three measuring tools for our measurement framework, namely: *measurement client*, *observer* and *gatherer*. The measurement client is used to collect workflow-level performance metrics; while observer and gatherer are used in performing internal measurement on the server side to capture PE-level performance metrics.

Metric	Unit	Description
Processing Time (pt)	second	Time taken to execute a PE ($endTime - startTime$)
Data Volume (dv)	block/tuple	Amount of data been processed
Data Throughput (dt)	block/tuple per second	Amount of data been processed per second (dv/pt)

Table 1: Metrics for process element level

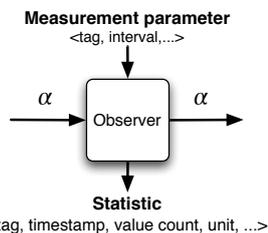


Figure 4: Design of Observer

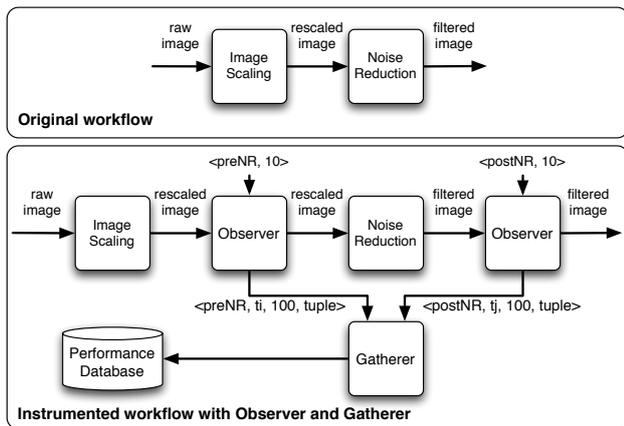


Figure 5: Using Observer in workflow

4.2.1 Measurement Client

Measurement Client is used to submit workflows to servers for enactment. It does timestamping before a request is submitted to server ($start_time$) and after the workflow is finished (end_time). Therefore, the response time of workflow execution can be obtained by:

$$workflow_execution_time = end_time - start_time$$

The data captured by the measurement client is stored in the measurement framework database. The measurement client is designed to allow single/multiple workflows to be executed in an experiment and repeated as user requests.

4.2.2 Observer

Observer is designed as a PE that can be inserted into any workflow to perform internal measurements. Observer does time stamping and is called to measure the PE-level performance. As illustrated in Figure 4, an observer receives an input stream from a previous activity, computes data flow, and outputs the data to the following activity without altering the content of the input. Observer takes separate parameter inputs for the measurement setting, which includes a unique *tag* for the particular observer and an *interval*. As shown in Figure 5, the first observer is placed before the activity *Noise Reduction* and is labelled as *preNR* while another observer *postNR* is placed after it. The interval is set to 10 for

Field Name	Data Type	Description
oid	String	unique identity given to an observer
count	Long	amount of data measured
unit	String	blocks, tuples or bytes
timestamp	Long	system timestamp

Table 2: Measurement Data Sent to Gatherer

both observers to trigger the observer does a timestamping for every 10 blocks/tuples/bytes of data read from the input stream, and to send the timestamp together with the meta-data to a gatherer as a tuple. Sending measurement results as tuples will enable the observer to connect with other existing activities such as union and merge. Table 2 shows the data sent to gatherer for each measurement.

4.2.3 Gatherer

Gatherer works closely with Observer in performing internal measurement. The measurement results collected from the observers will be streamed into a gatherer. The gatherer will add experiment metadata, e.g., experiment ID, workflow ID, and execution environment. The output from the gatherer is in tuple form and can be post processed, such as inserted into a measurement database, as shown in Figure 5, or stored in a local file system—using standard PEs.

5. EXPERIMENT

A reasonably complex and large process has been chosen as the test load for the experiments. We have selected the EURExpress-II [22] use case for the experiment where we have obtained sufficient data and a good understanding of it.

5.1 Experiment Use Case: EURExpress-II

The EURExpress-II project aims to build a transcriptome-wide atlas of gene expression for the developing mouse embryo established by RNA *in situ* hybridisation. The project annotates images of the mouse embryos by tagging images with terms from the ontology for mouse anatomy development. The data consists of mouse embryo image files and an annotation database (in MySQL) that describes the images. To date, 4 Terabytes of images have been produced and 80% of the annotation have been done manually by human curators. We will produce multiple classifiers where each classifier recognises a gene expression from a set of 1,500 anatomical components to classify the remaining 20% of images automatically. The studied example is divided into 3 stages: *Training*, *Testing* and *Development*. The training and testing stage are performed in one workflow. Datasets are split into 2 parts: for training a classifier and for test the accuracy of the trained classifier. The classifier will then be deployed to classify the remaining data.

Figure 6 outlines the test request used for measurement as explained below:

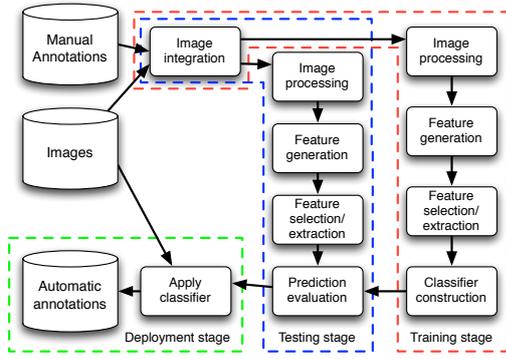


Figure 6: High level EURExpress-II workflow

1. Read raw image file and annotation database.
2. Image Scaling: Scale selected images to a standard size (320×200 pixels).
3. Noise Reduction: Apply median filtering to reduce the image noise.
4. Features Generation: Using wavelet transformation, generate the image features as matrices of wavelet coefficients. 64,000 features are generated per image of 320×200 pixels.
5. Features Extraction: Reduce the features set by selecting the representative features for constructing classifiers using Fisher Ratio analysis [16]. In our experiment, 24 most significant features are extracted from 64,000 features generated in step 4.
6. Classifier Design: Build a separate classifier for each anatomical feature which takes image features as input and outputs a rating of 'not detected', 'possible', 'weak', 'moderate' or 'strong' for anatomical features (eyes, nose, etc).
7. Evaluation: Test the classifier built in step 6 against a partition of the data not used in the preceding steps but already classified.

We create a test workload corresponding to the EURExpress-II described above as an OGSA-DAI workflow, where each data-mining task is implemented as an OGSA-DAI activity.

5.2 Implementation using OGSA-DAI

An OGSA-DAI DAG is created and deployed on our computational platform. Measuring PEs are inserted to collect performance data is collected. The instrumented DAG is executed and performance data is collected. The execution is repeated with different *data sizes* e.g., pre-process different numbers of image files over a range of 800 to 19200. The results collected are stored into an experiment database for subsequent analysis.

All of the PEs described in Section 5.1 have been developed as OGSA-DAI activities, as shown in Figure 7. For instance, an OGSA-DAI activity named *MedianFilterActivity* is developed to de-noise the raw image. Besides the DMI PEs (labelled as PE1 to PE9 in Figure 7), a set of utility PEs are needed, e.g., *ListConcatenate* activity, is used to concatenate feature streams from different machines to form a single data stream for *FeatureSelection* PE.

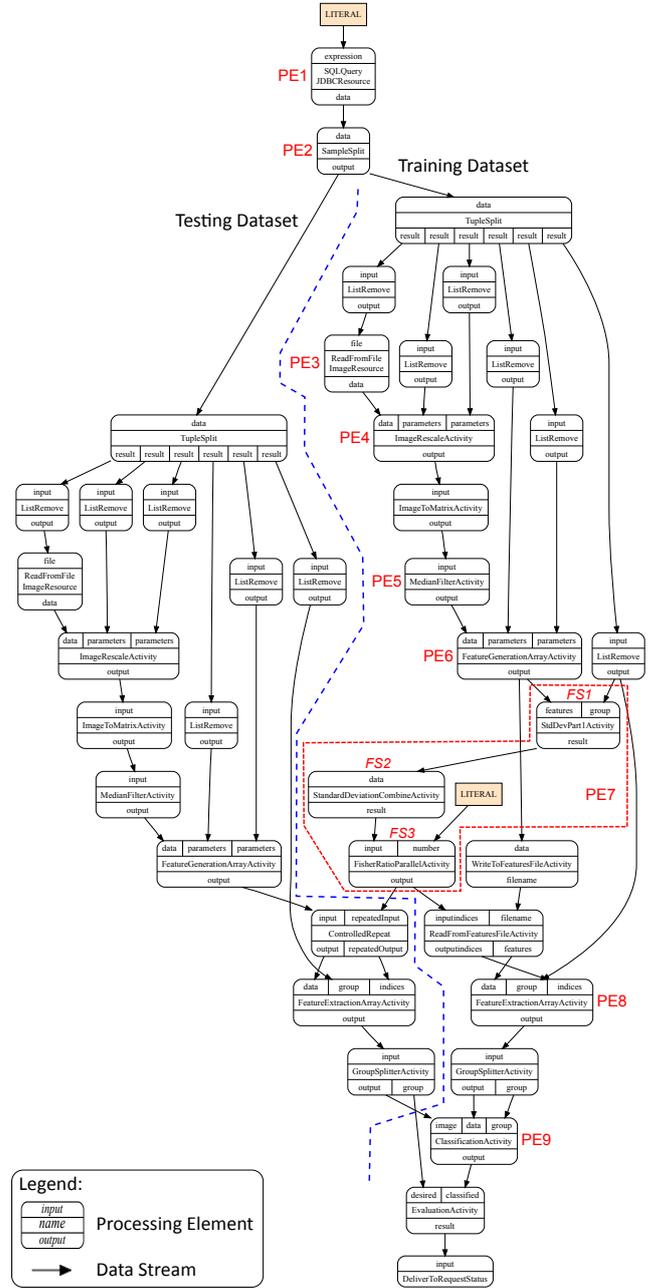


Figure 7: Implementation of PEs in EURExpress-II workflow as OGSA-DAI activities

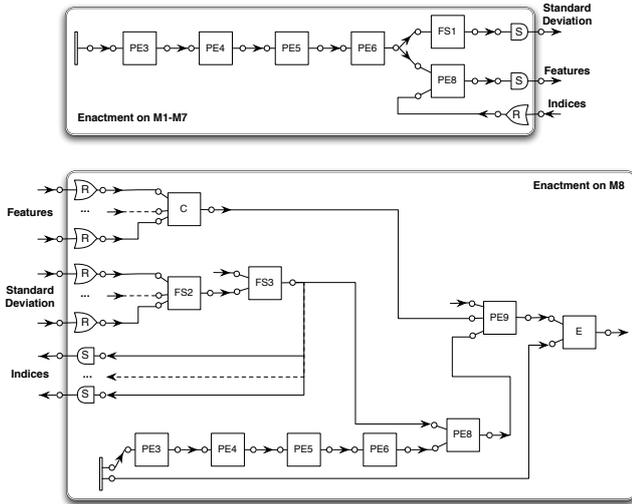


Figure 8: Enactment plan for EURExpress-II

Figure 8 illustrates a parallel enactment plan of the workflow on the experiment platform. 7 machines (M1 – M7) are used to process a training dataset. Each machine will receive a subset of the training data, read the selected image files, re-scale and de-noise the retrieved images, generate features, and perform partial feature selection (FS1). The result of FS1 will be sent through the network to M8, and combined with result from other machines for the remaining feature selection calculation (FS2 and FS3). The indices of the selected features are sent back to M1 – M7 for features extraction. Finally, M8 will receive all the extracted features of the training dataset and perform classification on its testing dataset.

Data movement between the machines is implemented using 2 OGSA-DAI activities, namely `DeliverToTCPActivity` and `ObtainFromTCPActivity`. The `DeliverToTCPActivity` creates a TCP socket and send data to another TCP host implemented in the `ObtainFromTCPActivity`. Data can be sent as primitive data type, such as integer and double, or as resizable char array. These two activities are designed to allow fine-tuning on the transmission process by adjusting char array size and socket buffer size, and to support large volume of data movement with multithreads.

5.3 Experiment apparatus

Hardware infrastructure

The experimental platform comprises 8 Intel Core Duo 2.4 GHz PCs with 2GB RAM and an Intel 2 GHz Core 2 Duo MacBook with 2GB RAM connected using a 10/100 Mbps switch. The PCs will host OGSA-DAI servers that receive workflow requests from a client and execute them. The MacBook acts as the client submitting workflow requests.

Software infrastructure

The PCs are running on Scientific Linux (version 2.6.18-92.1.22.el5PAE) and the Macbook is running on Mac OS X (version 10.5.6). In order to run as OGSA-DAI server, the PCs are installed with OGSA-DAI 3.1 GT and prerequisite software, including Java 1.6, Globus Toolkit 4.2 Web

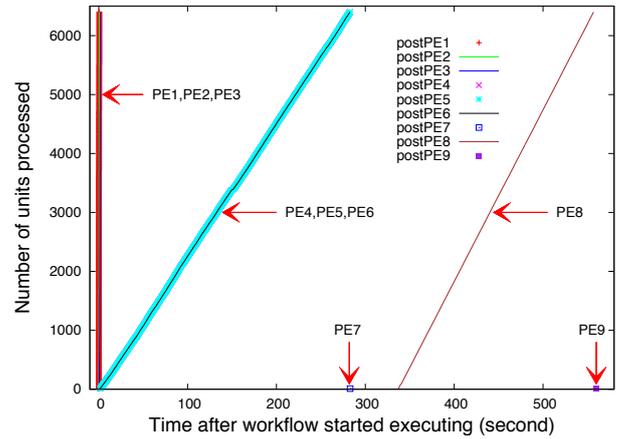


Figure 9: Timeline of PEs execution in single machine with 6400 images

Services Core, Jakarta Tomcat 5.5 and Apache ANT. The MacBook is installed with Java 1.6 to execute the submission client and MySQL (version 5.0) to store the captured measurement data.

5.4 Parameters used for experiment

In each experiment, the dataset is split across a number of machines (from 1 to 8), and the execution performance is measured. The experiment started with 800 images running on single machine, 2 machines to 8 machines. The number of images is increased for each experiment iteration to 19200 images. Each iteration is repeated 10 times.

6. RESULTS AND DISCUSSION

We first examine the results of instrumenting the workflow. Figure 9 shows the timeline of the PEs' execution measured by the internal measuring tools. The result obviously shows that most of the computation time is spent in the image preprocessing stage (PE4 and PE5) which involves massive matrix calculations, feature generation (PE6), which takes an image as the input to produce features using a wavelet transformation algorithm where each image will generate 64,000 double type values, and feature extraction PE8, which loops through the features list and extracts them for classification.

The aggregative behaviour of PE7 and PE9 are the bottleneck of the system, which breaks the streaming process. PE7 needs to read all the partial calculation of FS1, combine them using FS2 and output an array of feature indices in FS3. PE9 receives two input streams, i.e., the training dataset and testing dataset, as shown in Figure 7. In a standard k -fold validation in data mining, the dataset will be divided into k dataset, where $k - 1$ datasets are used for training the classifier, and the remaining dataset is used for testing the classifier. At i th validation, the training sub-DAG needs to process $k - 1$ times data compared to the testing sub-DAG, which results in a large difference between the data input rate between the two input streams of PE9. Thus, PE3, PE4, PE5 and PE6 in the training sub-DAG are identified to be the splitting points and executed in parallel. However, the above split causes every remote machines to send the extracted features produced by PE6 across the network to the

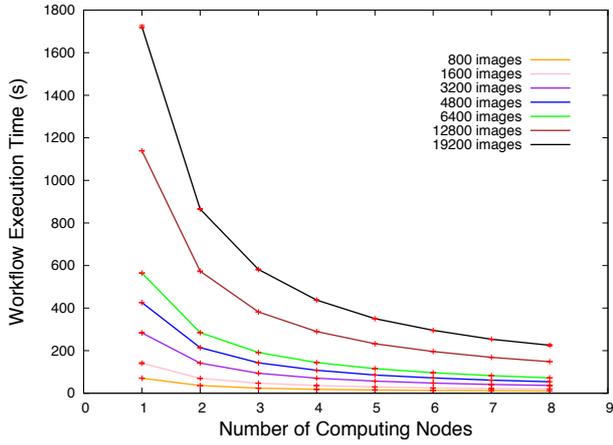


Figure 10: Workflow execution time with 99% confidence intervals

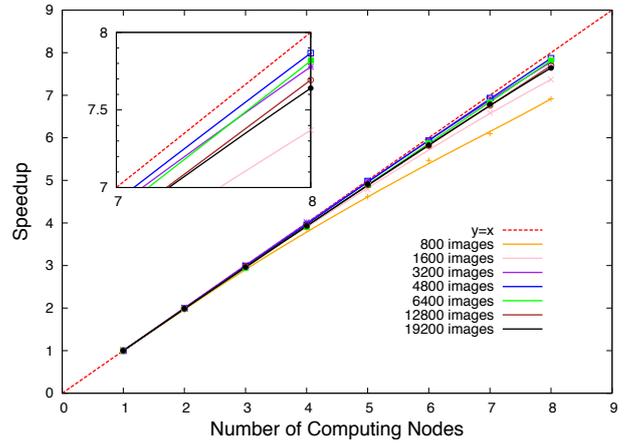


Figure 12: Speedup from parallel execution

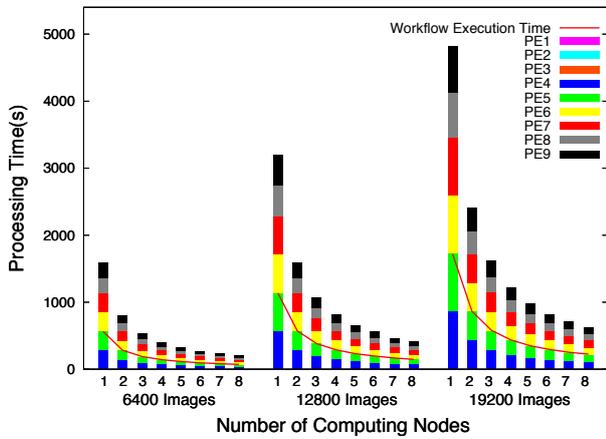


Figure 11: Non-overlapped processing time of DMI PEs compared with actual pipeline execution time

final machine that executes PE8 and PE9, which is the largest data stream in the entire workflow. To overcome this problem, we cut at PE7, as shown in Figure 8, and successfully reduce the overall communication cost.

From Figure 10, the overall processing time has been reduced with the increasing of number of computing nodes. Next, we examine the activity level metrics for 3 data samples (i.e. 6400, 12800 and 19200 images) in Figure 11. The colour histogram shows the processing time of 9 DMI PEs. PE₁, PE₂ and PE₃ show no significant impact on the overall execution. The bottleneck (i.e. image pre-processing, features generation, selection and extraction) is executed in parallel on different parts of the data sample. The total processing time of the PEs is less than the workflow execution time. For instance, the total processing time of the PEs for executing 19200 images on a single machine is 4821 second, while the real workflow execution time (indicated as red line in Figure 11) is 1721 second. In the streaming model, a PE will output the result to the next PE in the workflow as soon as it finishes the computation on a portion of its data stream.

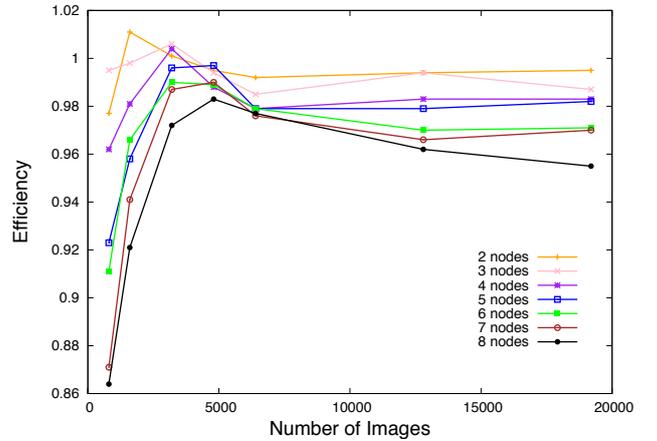


Figure 13: Efficiency of the parallel execution

Figure 12 shows a linear speedup with the number of computing nodes used. Figure 13 shows the efficiency of the parallel execution, which indicates:

1. Efficiency is close to 1 for all the selected sample sizes and number of computing nodes used.
2. Efficiency decreases when we increase the number of computing nodes.
3. Efficiency increases when we increase the sample size until a maximum is reached around 4000 to 5000 images, and drops after that.
4. The number of images at which the maximum efficiency is reached increases with the number of computing nodes used.

7. RELATED WORK

Success stories from the scientific communities demonstrate the benefits of combining efforts in terms of technology and knowledge to solve grand challenges, e.g., in astronomy [7], physics [8], meteorology [32], etc. A wide range of workflow management systems (WMS) support these e-Science activities. Roger *et al.* highlight the role of workflow technology in e-Science research [5].

In order to ease the combined data integration and task orchestration, the computation is structured as a workflow using a particular language e.g., BPEL [35] and SCUFL [30], or graph-based representation: a general Directed Acyclic Graph (DAG) with the tasks as the nodes (vertices) and data dependencies as the edges (arcs) or WMS specific DAG such as graph directed acyclic graph in XML (DAX) used in Pegasus [13]. Visualising and manipulating workflows as a graph may be more comprehensible than text for some users, however, formal programming style text is more expressive for describing workflow structures. In our work, workflows are defined using DISPEL, a language developed in the ADMIRE project—*Data-Intensive Systems, Process-Engineering Language (DISPEL)* [3], which is processed to generate DAG for optimisation and enactment.

The execution of workflows is managed by a wide range of WMS, e.g. Pegasus [13], Kepler [1], Taverna [31], Triana [38], Swift [41], etc. Some of these WMS are developed to support specific scientific domain, e.g., Taverna was first used for bioinformatics. In general, WMS provide the tools for workflow composition and resource mapping; workflow execution engines (e.g., DAGman in Condor [10]) take charge of executing the workflows on available resources. See [12, 40, 11] for reviews of these technologies. Callaghan *et al.* describe how they manage to optimise an earthquake science application, CyberShake [9].

Some of these WMS perform optimisation. In Pegasus, optimisation includes *workflow reduction* (reusing available intermediate data products), *task clustering* (reduce scheduling overhead by combining tasks), *data cleanup* (remove data no longer needed) and *partitioning* (partitions into sub-workflows with dependencies). Pegasus improves workflow performance using *placeholders*. Placeholders are units of work (either shell scripts or MPI wrappers) that are submitted to the queue of the execution engine, which once launched can be used to execute multiple tasks. The placeholder implementation is similar to the Glide-in approach [33] in Condor.

Glatard *et al.* [19] define two workflow manager architectures: *task-based*, where workflow manager is responsible in handling computing task; *service-based*, where computation is handled by external services. MOTEUR [18], is a workflow enactor that optimises service-based workflows by exploiting intrinsic parallelism (no dependencies between services), data parallelism (different data on different threads) and service parallelism (pipe-lining different services in handling independent data sets). Glatard *et al.* also propose grouping services of a workflow to reduce the overall overhead from job submission, scheduling, queuing and data movement [17].

ASKALON [15] is an example of a task-based approach workflow manager. There are three core services within the ASKALON working closely to optimise execution: *Resource Manager (GridARM)*, *Scheduler* and *Performance Prediction*. The resource manager handles negotiation, advance reservation, allocation of resources and deployment of services. Scheduler is responsible for mapping workflows and monitoring execution, while performance prediction service estimates of execution times. Together, they provide Quality of Service (QoS) by dynamically adjusting the optimised

schedules according to the infrastructure status.

Relational DBMS (e.g., Postgres, Oracle, DB2, MonetDB, SciDB) use streaming in their parallelised query evaluations [36]. These have demonstrated the power of describing operators so that their re-ordering and parallelisation can be logically inferred. In Distributed Query Processing [27, 25], reordering rules and parallelisation are extended to permit migration of operators along the data tree. We build on their algorithms and extend them to an open universe of PEs and permit DAGs as well as trees.

The principal differences between these systems and our work is that we describe the types transmitted between PEs in terms of a programming-language recursive type system with primitive scientific types and then use this structure to do finer-grained streaming. This permits measurement normalised to the units handled and hence re-use of the measurements in new workflow and platforms contexts.

8. CONCLUSION AND FUTURE WORK

We have explored and evaluated the feasibility of using streaming technology on a Life-Sciences use case—EURExpress-II. We have defined our optimisation strategy, which explores parallel optimisation opportunities in a workflow, splitting the workflow according to performance results, and executing these across distributed platforms. The results have shown a linear speed up.

The present work is aimed at facilitating distributed data-intensive computation that will be undertaken by large numbers of researchers and decision makers using modest local resources and a variety of external resources associated with their data sources. We anticipate large numbers of applications handling hundreds of gigabytes to a few tens of terabytes. To facilitate their use it is necessary to automate both the performance measurement collection and optimising transformations.

The results to-date are based on a small number of workflows in one application domain using a small set of PEs with manually organised descriptions and performance data. Our next architectural step is to build the information and data framework for optimisation so that it involves minimal manual procedures. Then we will extend the range of optimisations, the operational strategy and the experiments to demonstrate the strategy and architecture on a representative set of applications and scales. We would be pleased to cooperate in the construction of agreed data-intensive benchmarks.

9. REFERENCES

- [1] ALTINTAS, I., BERKLEY, C., JAEGER, E., JONES, M., LUDASCHER, B., AND MOCK, S. Kepler: an extensible system for design and execution of scientific workflows. *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on* (June 2004), 423–424.
- [2] ATKINSON, M., BREZANY, P., CORCHO, O., HAN, L., VAN HEMERT, J., HLUCHÝ, L., HUME, A., JANCIAK, I., KRAUSE, A., AND SNELLING, D. ADMIRE White Paper: Motivation, Strategy, Overview and Impact.

- Tech. Rep. version 0.9, ADMIRE, EPCC, University of Edinburgh, January 2009.
- [3] ATKINSON, M., VAN HEMERT, J., KRAUSE, A., LIEW, C. S., AND YAIKHOM, G. Data-intensive systems, process-engineering language (dispel): Role, motivation and rationale. Tech. rep., School of Informatics, University of Edinburgh, 2009.
 - [4] ATKINSON, M. P., VAN HEMERT, J. I., HAN, L., HUME, A., AND LIEW, C. S. A distributed architecture for data mining and integration. In *DADC '09: Proceedings of the second international workshop on Data-aware distributed computing* (2009), ACM, pp. 11–20.
 - [5] BARGA, R., AND GANNON, D. Scientific versus business workflows. *Workflows for e-Science* (2007), 9–16.
 - [6] BELL, G., HEY, T., AND SZALAY, A. COMPUTER SCIENCE: Beyond the Data Deluge. *Science* 323, 5919 (2009), 1297–1298.
 - [7] BERRIMAN, G. B., DEELMAN, E., GOOD, J., JACOB, J. C., KATZ, D. S., LAITY, A. C., PRINCE, T. A., SINGH, G., AND SU, M.-H. Generating complex astronomy workflows. *Workflows for e-Science* (2007), 19–38.
 - [8] BROWN, D. A., BRADY, P. R., DIETZ, A., CAO, J., JOHNSON, B., AND MCNABB, J. A case study on the use of workflow technologies for scientific analysis: Gravitational wave data analysis. *Workflows for e-Science* (2007), 39–59.
 - [9] CALLAGHAN, S., DEELMAN, E., GUNTER, D., JUVE, G., MAECHLING, P., BROOKS, C., VAHI, K., MILNER, K., GRAVES, R., FIELD, E., OKAYA, D., AND JORDAN, T. Scaling up workflow-based applications. *Journal of Computer and System Sciences In Press, Corrected Proof* (2009).
 - [10] COUVARES, P., KOSAR, T., ROY, A., WEBER, J., AND WENGER, K. Workflow management in condor. *Workflows for e-Science* (2007), 357–375.
 - [11] CURCIN, V., AND GHANEM, M. Scientific workflow systems - can one size fit all? In *Biomedical Engineering Conference, 2008. CIBEC 2008. Cairo International* (Dec. 2008), pp. 1–9.
 - [12] DEELMAN, E., GANNON, D., SHIELDS, M., AND TAYLOR, I. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems* 25, 5 (2009), 528 – 540.
 - [13] DEELMAN, E., SINGH, G., SU, M.-H., BLYTHE, J., GIL, Y., KESSELMAN, C., MEHTA, G., VAHI, K., BERRIMAN, G. B., GOOD, J., LAITY, A. C., JACOB, J. C., AND KATZ, D. S. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming* 13, 3 (2005), 219–237.
 - [14] DOBRZELECKI, B., KRAUSE, A., HUME, A., GRANT, A., ANTONIOLETTI, M., ALEMU, T., ATKINSON, M., JACKSON, M., AND THEOCHAROPOULOS, E. Integrating Distributed Data Sources with OGSA-DAI DQP and Views. *Philosophical Transactions of the Royal Society A* (2010), to appear.
 - [15] FAHRINGER, T., PRODAN, R., DUAN, R., HOFER, J., NADEEM, F., NERIERI, F., PODLIPNIG, S., QIN, J., SIDDIQUI, M., TRUONG, H.-L., VILLAZON, A., AND WIECZOREK, M. Askalon: A development and grid computing environment for scientific workflows. *Workflows for e-Science* (2007), 450–471.
 - [16] FISHER, R. A. The use of multiple measurements in taxonomic problems. *Annals Eugen.* 7 (1936), 179–188.
 - [17] GLATARD, T., MONTAGNAT, J., EMSELLEM, D., AND LINGRAND, D. A service-oriented architecture enabling dynamic service grouping for optimizing distributed workflow execution. *Future Generation Computer Systems* 24, 7 (2008), 720 – 730.
 - [18] GLATARD, T., MONTAGNAT, J., LINGRAND, D., AND PENNEC, X. Flexible and efficient workflow deployment of data-intensive applications on grids with moteur. *Int. J. High Perform. Comput. Appl.* 22, 3 (2008), 347–360.
 - [19] GLATARD, T., SIPOS, G., MONTAGNAT, J., FARKAS, Z., AND KACSUK, P. Workflow-level parametric study support by moteur and the p-grade portal. *Workflows for e-Science* (2007), 279–299.
 - [20] GORTON, I., GREENFIELD, P., SZALAY, A., AND WILLIAMS, R. Data-intensive computing in the 21st century. *Computer* 41, 4 (2008), 30–32.
 - [21] GRAY, J. Jim Gray on eScience: A Transformed Scientific Method. In *The Fourth Paradigm: Data-Intensive Scientific Discovery*, T. Hey, S. Tansley, and K. T. (Editors), Eds. Microsoft, 2009, pp. xix–xxxiii.
 - [22] HAN, L., VAN HEMERT, J., BALDOCK, R., AND ATKINSON, M. Automating gene expression annotation for mouse embryo. *Lecture Notes in Computer Science (Advanced Data Mining and Applications, ADMA 2009) LANI 5678* (2009), 469–478.
 - [23] INTERAGENCY WORKING GROUP ON DIGITAL DATA. Harnessing the Power of Digital Data for Science and Society. Report of the Interagency Working Group on Digital Data to the National Science and Technology Council. Tech. rep., Executive office of the President, Office of Science and Technology, Washington D.C. 20502 USA, January 2009.
 - [24] JACOBS, A. The pathologies of big data. *Commun. ACM* 52, 8 (2009), 36–44.
 - [25] KOSSMANN, D. The state of the art in distributed query processing. *ACM Comput. Surv.* 32, 4 (2000), 422–469.
 - [26] KOZIOLEK, H. Introduction to performance metrics. *Dependability Metrics* (2008), 199–203.
 - [27] LYNDEN, S., MUKHERJEE, A., HUME, A. C., FERNANDES, A. A., PATON, N. W., SAKELLARIOU, R., AND WATSON, P. The design and implementation of ogsa-dqp: A service-based distributed query processor. *Future Generation Computer Systems* 25, 3 (2009), 224 – 236.
 - [28] MEYER, L., ANNIS, J., WILDE, M., MATTOSO, M., AND FOSTER, I. Planning spatial workflows to optimize grid performance. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing* (New York, NY, USA, 2006), ACM, pp. 786–790.
 - [29] NURMI, D., MANDAL, A., BREVIK, J., KOELBEL, C., WOLSKI, R., AND KENNEDY, K. Evaluation of a workflow scheduler using integrated performance

- modelling and batch queue wait time prediction. *Supercomputing, 2006. SC '06. Proceedings of the ACM/IEEE SC 2006 Conference* (Nov. 2006), 29–29.
- [30] OINN, T., ADDIS, M., FERRIS, J., MARVIN, D., SENER, M., GREENWOOD, M., CARVER, T., GLOVER, K., POCOCK, M., WIPAT, A., AND LI, P. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics* 20, 17 (November 2004), 3045–3054.
- [31] OINN, T., LI, P., KELL, D. B., GOBLE, C., GODERIS, A., GREENWOOD, M., HULL, D., STEVENS, R., TURI, D., AND ZHAO, J. Taverna/mygrid: Aligning a workflow system with the life sciences community. *Workflows for e-Science* (2007), 300–319.
- [32] PLALE, B., GANNON, D., BROTZGE, J., DROEGEMEIER, K., KUROSE, J., MCLAUGHLIN, D., WILHELMSON, R., GRAVES, S., RAMAMURTHY, M., CLARK, R. D., YALDA, S., REED, D. A., JOSEPH, E., AND CHANDRASEKAR, V. CASA and LEAD: Adaptive cyberinfrastructure for real-time multiscale weather forecasting. *Computer* 39, 11 (2006), 56–64.
- [33] SFILIGOI, I. Making science in the grid world: using glideins to maximize scientific output. In *Nuclear Science Symposium Conference Record, 2007. NSS '07. IEEE* (26 2007–Nov. 3 2007), vol. 2, pp. 1107–1109.
- [34] SINGH, G., KESSELMAN, C., AND DEELMAN, E. Optimizing grid-based workflow execution. *Journal of Grid Computing* 3, 3 (09 2005), 201–219.
- [35] SLOMINSKI, A. Adapting bpel to scientific workflows. *Workflows for e-Science* (2007), 208–226.
- [36] STONEBRAKER, M. Stream processing. In *Encyclopedia of Database Systems*, L. Liu and M. T. Özsu, Eds. Springer US, 2009, pp. 2837–2838.
- [37] STONEBRAKER, M., BECLA, J., DEWITT, D. J., LIM, K.-T., MAIER, D., RATZESBERGER, O., AND ZDONIK, S. B. Requirements for Science Data Bases and SciDB. In *CIDR* (2009), www.crdldb.org.
- [38] TAYLOR, I., SHIELDS, M., WANG, I., AND HARRISON, A. The triana workflow environment: Architecture and applications. *Workflows for e-Science* (2007), 320–339.
- [39] TRUONG, H.-L., DUSTDAR, S., AND FAHRINGER, T. Performance metrics and ontologies for grid workflows. *Future Generation Computer Systems* 23, 6 (2007), 760 – 772.
- [40] YU, J., AND BUYYA, R. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Rec.* 34, 3 (2005), 44–49.
- [41] ZHAO, Y., HATEGAN, M., CLIFFORD, B., FOSTER, I., VON LASZEWSKI, G., NEFEDOVA, V., RAICU, I., STEF-PRAUN, T., AND WILDE, M. Swift: Fast, reliable, loosely coupled parallel computation. In *Services, 2007 IEEE Congress on* (July 2007), pp. 199–206.