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 1.1 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 LifeSciences. 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, 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 $V = \{Q_1, Q_2, T_1, ..., D_1, D_2\}$ with edges representing the data flow. For example, $Q_1$ retrieves data and passes them to $T_1$ 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., $T_1$ and $T_2$ may take different amounts of time to transform a unit of data.
- PEs may have different input consumption rates, e.g., if $*$ 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., $a)$ a PE consumes lists from its input and generates a tuple for each list on its output that is an aggregation of the list, $b)$ 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.

![Figure 1: Pipeline example](image-url)
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, within the application-dependent criteria at the least cost. Distributed computation that will deliver the same results to various targets. The execution of this DAG relies on various parameters, such as:

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.

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 vi is the computational cost per unit of data of a particular task, wi. The weight of an edge eij that connects vi and vj 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 W1 and W2, commodity cluster CL and high performance computing cluster HPC. Assume that v3 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. \([v_1,v_2,v_3,v_4,v_5],[v_1,v_6,v_7,v_8,v_9],[v_1,v_3,v_5,v_4,v_5]\). 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 \( e_{8,5} \)), 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.

\textit{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.

\textit{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.

\textit{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 \( v_5 \) 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 3) after the enactment step is stored in PDB. In this paper, we focus on splitting data streams. Section 3 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—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 [2].

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 instance, 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 ($T_w$) on a single processor and the execution time on multiple processors ($T_m$), represented as:
  \[ S = \frac{T_w}{T_m} \]

- Efficiency($E$): a relationship between the speedup ($S$) and the number of processors ($P_n$) used. It is defined as:
  \[ E = \frac{S}{P_n} \]

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.
### Table 1: Metrics for process element level

<table>
<thead>
<tr>
<th>Metric</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing Time (pt)</td>
<td>second</td>
<td>Time taken to execute a PE (endTime − startTime)</td>
</tr>
<tr>
<td>Data Volume (dv)</td>
<td>block/tuple</td>
<td>Amount of data been processed</td>
</tr>
<tr>
<td>Data Throughput (dt)</td>
<td>block/tuple per second</td>
<td>Amount of data been processed per second (dv/pt)</td>
</tr>
</tbody>
</table>

### Figure 4: Design of Observer

#### 4.2.2 Observer

Observer is designed as a PE that can be inserted into any workflow to perform internal measurements. Observer does timestamping and is called to measure the PE-level performance. As illustrated in Figure 1, 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 4, 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 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 metadata 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 d 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 6 or stored in a local file system—using standard PEs.

### Figure 5: Using Observer in workflow

### 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:
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 measurements. 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.
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 as 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 11: Non-overlapped processing time of DMI PEs compared with actual pipeline execution time

Figure 12: Speedup from parallel execution

Figure 13: Efficiency of the parallel execution

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.

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 [5], meteorology [72], 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 such as BPEL or SCUFL, 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) take charge of executing the workflows on available resources. See [12, 40] for reviews of these technologies. Callaghan et al. describe how they manage to optimise an earthquake science application, CyberShake.

Some of these WMS perform optimisation. In Pegasus, optimisation includes workflow reduction (reusing available inter-domain data sets), 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 in Condor.

Glatard et al. 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 [15], 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.

ASKALON 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 [50]. 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


modelling and batch queue wait time prediction. 

Supercomputing, 2006. SC ’06. Proceedings of the 

[30] Oinn, T., Addis, M., Ferris, J., Marvin, D., 
Senger, 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. 

[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 

[33] Sfiligoi, I. Making science in the grid world: using 
glideins to maximize scientific output. In Nuclear 

[34] Singh, G., Kesselman, C., and Deelman, E. 
Optimizing grid-based workflow execution. Journal of 

[35] Slominski, A. Adapting bpel to scientific workflows. 

[36] Stonebraker, M. Stream processing. In 
Encyclopedia of Database Systems, L. Liu and M. T.

[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 

[38] Taylor, I., Shields, M., Wang, I., and Harrison, 
A. The triana workflow environment: Architecture 
and applications. Workflows for e-Science (2007), 
320–339.

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., Raiciu, I., 
Stef-Praun, T., and Wilde, M. Swift: Fast, 
reliable, loosely coupled parallel computation. In 
Services, 2007 IEEE Congress on (July 2007), 