A Data Lineage Model for Distributed Sub-image Processing

Johnson Mwebaze
Makerere University and
University of Groningen
Landleven 12, 9700 AV
Groningen
The Netherlands
jmwebaze@cit.ac.ug

John McFarland
University of Groningen
Landleven 12, 9700 AV
Groningen,
The Netherlands
mcfarland@astro.rug.nl

Danny Booxhorn
University of Groningen
Landleven 12, 9700 AV
Groningen,
The Netherlands
danny@astro.rug.nl

Edwin Valentijn
University of Groningen
Landleven 12, 9700 AV
Groningen,
The Netherlands
valentine@astro.rug.nl

ABSTRACT

An important challenge facing e-Science is the development of scalable systems and analysis techniques that allow client applications to locate data and services in increasingly large-scale distributed environments. e-Science Systems should achieve three main goals: (i) efficient and selective processing of data, (ii) support network collaboration without clogging distribution networks; and (iii) allow transparency of experiments through repeatability and verifiability of experiments. Several systems have addressed limited combinations of these properties, but we address all three in this work. We describe the architecture and implementation of such a framework in Astro-WISE, an astronomical approach to distributed data processing, discovery and retrieval of datasets that achieves scalability via dynamic linking (data lineage) maintained within the system. We show that lineage data collected during the processing and analysis of datasets can be reused to perform selective reprocessing (at sub-image level) on datasets while the remainder of the dataset is untouched, a rather difficult process to automate without lineage.

Categories and Subject Descriptors

H.3.4 [**Systems and Software**]: Distributed Systems; H.4 [**Information Systems Applications**]: Workflow Management; J.2 [**Physical Sciences and Engineering**]: Astronomy

General Terms

Design, Experimentation

Keywords

data lineage, provenance, target processing, scientific computing, data reduction, subimage processing

1. INTRODUCTION

The nature of astronomical (or scientific) data processing is changing. Datasets are doubling every year with archives growing in size beyond petabyte scales [17]. Likewise, the processing is increasing in complexity requiring laboriously sophisticated techniques and expertise in both data reduction and analysis. Most often such processing requires trying different data sets, trying different processing techniques, tweaking parameters, verifying data quality, repeating this data derivation and customization of the results. As a result of such analysis more data is generated as new results are derived. It is therefore crucial for the experimenter or other users of the system to understand how a result was derived and what data was used and how it was selected.

Astronomical systems must also provide scalable processing solutions, and allow seamless access to various distributed resources and archives to facilitate such analysis. However, how will users retrieve and process data they are interested in? For moving an entire dataset, a surprisingly low tech approach is often most used. However, most clients have neither the time nor resources to effectively store and analyze such huge datasets. Moreover, the cost of data transfer is not substantially cheaper today than the price of disks to store the data. Therefore, transferring a minimal set of data is critical. Additionally, the traditional approach of simply moving the data to where there is an analysis facility is inherently not scalable primarily due to network-related and client processing constraints.

We use the above examples to motivate the need to trace lineage and also the need to use lineage data to enable the effective and selective reuse of data, knowledge and experiences from previous experiments to aid both expert and non-expert distributed users in performing scientific analysis.

Data lineage (provenance) is a well-defined problem with known solutions as surveyed in [4,7,8]. The usefulness of provenance for each implementation is linked to the level of granularity at which it is collected. The requirement for this framework calls for tracing lineage on pixels. This is because most often users are interested in a source (e.g., moving, variable, or extreme in some colour index) that lies on a few pixels of an image. The approach adopted by most observation systems is processing the entire image or set of images (a single image may have up to a million sources) even when the sole source of interest may exist on only a few pixels of one or a few images. Accordingly, out of millions of images for a survey, it is nearly impossible, time consuming and wasteful to process the whole data volume.

Based on these observations, we argue that instead of processing the whole dataset (or full images), a user should only select, retrieve and process only relevant pixels on an image where the source exists. However, what happens to image operations that can not be done on a sub-image level? We claim that such a framework is possible if we can retrieve an entire backtrace of the processing history of an image at pixel level, and by using this lineage data we enable such operations that are very difficult (or impossible) to execute at sub-image level.

The goal of this paper is twofold. Firstly, we extend our lineage model presented in [14] to trace lineage at pixel level and we show that it accurately locates all pixels that were involved during processing of a result. Secondly, we show that using lineage we are able to support sub-image processing. We specifically focus on Astro-WISE however, our discussion of the requirements of sub-image processing can be applied to the general category of all observational sciences.

To the best of our knowledge, this is the first work that leverages lineage information to support sub-image processing to simplify and automate the reprocessing of objects. The rest of the paper is organized as follows; In Section 2, we briefly provide an overview of Astro-WISE. We present our pixel lineage in Section 3.1, and in Section 4, we detail the sub-image processing framework. We briefly present a use case to demonstrate the effectiveness of this framework in Section 5. We review related work in Section 6. Section 7 concludes and an outline of future work.

2. REVIEW OF Astro-WISE

Astro-WISE [3,18] is a distributed system for the scientific analysis of astronomical datasets. It is distributed over a number of sites allowing scientific communities to work with and share data (and instruments) while using their own resources. Astro-WISE stores all data from the raw data to the final science-ready product as well as the software used for the data reduction in the same system. As a result, scientists are able to develop a new workflow or pipeline from scratch, or to use an existing recipe. They are also able to process and reprocess data without losing intermediate data products in the same system using the same environment.

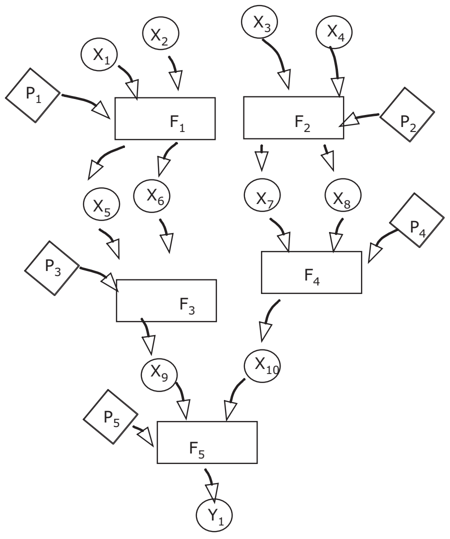


Figure 1: An example data-flow

Below, we review some terminology and introduce basic pipeline operations in Astro-WISE that serve as the basis for sub-image processing.

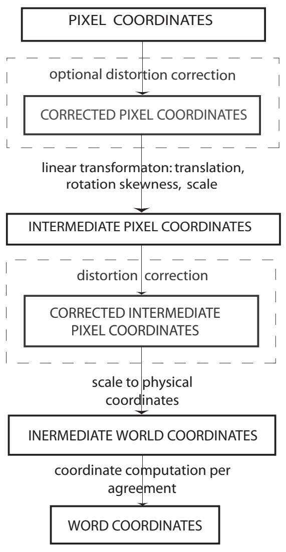
A *pipeline* is composed of modules which define specific operations and connections which specify the conceptual flow of data between modules. Pipelines are constructed on the fly using the dependency logic that is derived from lineage data. Each connection between modules is linked with an output port of one module (the source) with an input port of another module (the destination). In this work, we shall be modifying modules, connections and parameters, to adapt the existing data model to support sub-image processing. Thereafter, we make these changes on the modules as part of Astro-WISE.

Classes are associated with the various conventional images (calibration and data) and other derived data products. For example, in our system, bias exposures become instances of the `RawBiasFrame` class. These instances of classes are *objects*. *Objects* may have an associated data/image file. Classes have methods and attributes. Methods perform a task on the object while attributes are properties such as constants or links to other objects.

3. LINEAGE IN Astro-WISE

For simplicity, we represent the object data model in Astro-WISE as a graph shown in Fig. 1. The graph defines, three types of nodes; rectangles represent functions F, each F_i might have multiple inputs X and outputs Y. These are donated by a circle in the graph . Inputs/outputs could be a list of objects or images. Each F_i uses parameters P denoted as ◇→ on the graph. Parameters have a type (string, float, boolean, etc) or could be list of values, a linked list or an object. Nodes in the graph are connected with edges that show the flow of the dataflow.

Data lineage captured during processing, is information about all dependencies of the input/output variables and a complete instantiation of each function which is represented as



PIXEL COORDINATES
optional distortion correction
CORRECTED PIXEL COORDINATES
linear transformaton: translation, rotation skewness, scale
INTERMEDIATE PIXEL COORDINATES
distortion correction
CORRECTED INTERMEDIATE PIXEL COORDINATES
scale to physical coordinates
INERMEDIATE WORLD COORDINATES
coordinate computation per agreement
WORD COORDINATES

Figure 2: Flow chart showing a conversion of pixel coordinates to world coordinates.

$\langle F_{[P_1...P_n]}, [X_1....X_n], [Y_1...Y_n]\rangle$.

From this model, we explicitly assume that every output depends on every input and parameters passed to the function. This is because of the black-box nature of the services that are used for computation. When services are black boxes, we have to assume that all outputs depend on all inputs. Therefore, such lineage accounts for an output product produced during the course of a dataflow execution by displaying a connected graph of input, intermediate, output data and also parameters and attributes used during the processing. This level of granularity is in-sufficient for sub-image processing. We need to trace fine-grained lineage to include transformations at pixel/byte level. The problem of including such detailed lineage, is the size of provenance data may easily outgrow the size of the data being computed by the system. However, if we know some of the data processing and pixel transformation steps, and if we can provide a lazy inverse transformation, then we have the option to compute lineage at query time as needed rather than recording it explicitly.

3.1 Pixel Lineage

Celestial objects are identified by world coordinates. Every pixel position in a celestial image is associated with a world coordinate. Given a pixel position, for each image, the corresponding sky coordinates is computed as shown in Fig 2. Conversion from physical coordinates to world coordinates is a multi-step process that includes linear and nonlinear transformations that involve distortions corrections (e.g Eq 1). The reader is referred to [5] and [11] for the details. As the images go through the pipelines, the physical coordinates change in value, position and size as shown in Fig. 3 while the sky coordinates stay the same. Some pixels are added together to form a composite of a regridded image. The challenge is to determine which pixels, beginning from the raw images and all intermediate images were used in the construction of the final output regridded image. We denote $\mathcal{V}$ as the set of pixel transformations algorithm.

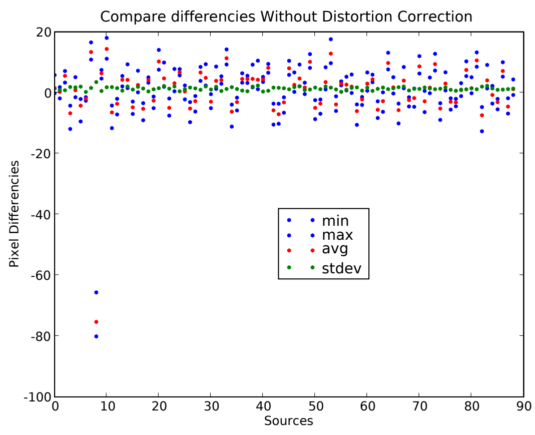
We define $F : \mathcal{V} \rightarrow \mathcal{V}$ as a function on the space of astronomical image processing, and $\delta : \mathcal{V} \times \mathcal{V} \rightarrow F$ as a function that takes an input pixels p_a and outputs p_b, then for brevity let $\delta_{ab}(p_a) = p_b$. If there exists δ_{ba} such that $\delta_{ba}\delta_{ab} = e$, where e is an identity matrix, then there exists a 1-1 mapping between pixels of the input image to pixels to an output image. Then it is possible to a find a mapping between pixels of related images or connect all pixels from raw images intermediate to the final objects. However, we can achieve this if our sequence of operations consists of invertible atomic operations. Specifically, suppose $\delta_{ab} = f_n \circ \cdots \circ f_1$ then each f_i must have a well-defined inverse. For example, if f_i is the operation of applying a distortion correction while converting from pixel coordinates to corrected pixel coordinates, f_i^{-1} is the operation of removing the distortion correction.

However, most scientific computations are treated as black boxes, moreover distortion polynomials do not have an analytic inverses implying invertibility is a very hard problem or impossible. Therefore we relax the invertibility requirement and instead switch to heuristics. We restrict our initial implementation to *TAN Distorted tangential* that employs plate solution (Eq 1) for Astrometry to cater for distortions. It is a mapping between a source detected at pixel coordinates (x, y) in frame f and its normal coordinates (ζ, η) relative to the nominal center of field f. This degree of the polynomial p is determined by the nature of distortions being accounted. a_{ij}, b_{ij}, m and c are initially unknown coefficients, to be determined by the least squares analysis of the plate coordinates of the reference stars. Equation 1 like any other distortion function has been defined in one direction and no attempt is made to find the analytic inverse [11].

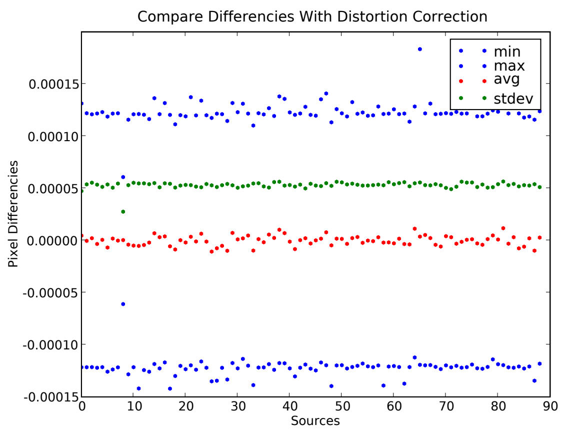
$$\begin{aligned}\zeta(x,y) &= \sum_{i+j=1}^{i+j=p} a_{ij}x^i y^j m^k c^l\,,\\ \eta(x,y) &= \sum_{i+j=1}^{i+j=p} b_{ij}x^i y^j m^k c^l\,.\end{aligned} \tag{1}$$

The relationship between input and output images to a pipeline can be derived from lineage so far captured in the system. What we need to find is the relationship between the pixels in the input images to the pixels output images. It remains to invert the pixel to world coordinate transformations. i.e Bottom-up approach in Fig. 2. Then given any pixel location, the (x, y) coordinate, and the corresponding image the pixel value can be obtained.

Using known methods for finding successively better approximations to the zeroes (or roots) of a real-valued function [12], we begin with a lazy conversion and use the results iteratively to find the best approximations of the root of Equation 1. We have implemented and tested this technique. The results of the test are shown in Fig 4. From the graph, panel (b) shows a positional accuracy of approximately 5 orders of magnitude over panel (a), an indication, that we can accurately find corresponding (x, y) positions of pixel. The algorithm can be tuned to achieve any precision, but its a trade-off between performance and accuracy. A higher precision will increase on the time the algorithm takes to converge (to find the root). With this framework, we can trace and link all pixels that have been processed in the system.



(a)



(b)

Figure 4: Panels (a) & (b) show the results of the coordinate transformation test. The difference in pixel coordinates (x,y) without and with (respectively) the distortions accounted for in the inverse transform (world to pixel). Blue is min/max difference for each sourcelist, red the average and green the standard deviation. Panel (b) shows a positional accuracy improvement of about 5 orders of magnitude over panel (a).

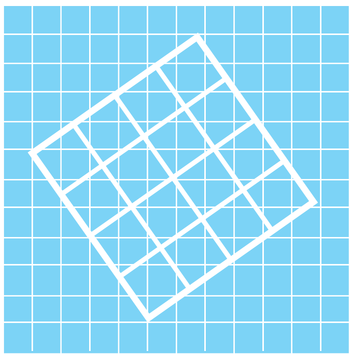


Figure 3: The Figure illustrates pixel transformations after applying linear shifts, rotations and distortions during the image regridding/co-addition process. The input grid is shown as small grey squares, whereas the output grid (re-sampled image) is represented by the large tilted/rotated ones

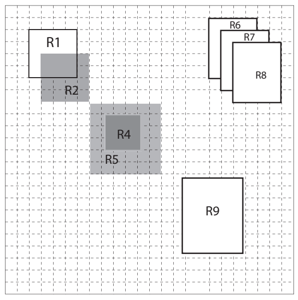


Figure 5: An example of a *CoverageMap*, Regions R1 - R9 represent processed regions which are denoted with a '1' in the pixelmap object

4. SUB-IMAGE PROCESSING

Practically, sub-image and full-image processing follow the same standards, data quality and processing requirements. However processing parameters, attributes and some modules might defer. This is because pipelines have been written and designed for instruments with fixed detector properties (e.g., image size, calibration frames, overscan regions, etc.). All metadata and processing parameters are based on an instruments or a detector. These variables must be modified to process a virtual image (sub-image).

Since no metadata (processing data) exists for sub-image processing, we use analogical reasoning to infer the necessary changes and then apply them to processing. By matching and retrieving existing pre-processed information (data lineage) in the system and we know what the relationship there is between what we want to process and what has been processed before, we should be able to determine the difference between pipelines (and objects) and then modify any new pipelines so that the new processing follows the new user processing requirements for a particular region on the frame.

4.1 Pipeline Matching and Building

The pipeline for sub-image processing is built using lineage data. Likewise all input data, attributes and parameters to be used in the sub-image pipeline are selected from the same lineage data. However, since we are processing a sub-image, selected lineage data to be used as input to the sub-image pipeline might have to be modified. When such changes occur then the part of pipeline affected by the changes must be re-run. The rerun will take data dependencies into account and only execute those parts of the pipeline affected by the changes.

Once a user requests to process a sub-image, then all provenance data needed to perform sub-image processing will be retrieved. The basic idea behind our algorithm is to search for a graph representation of the dataflow that was used in earlier runs to process the full-image from which the sub-image will be extracted. The precondition for the selection is that the same dataflow was run with the current parameters and input data. The next step is to retrieve the provenance data and intermediate data products produced by this graph from the provenance store. Then the retrieved data will be used as input to the sub-image processing pipeline that needs to be rerun.

The starting point of sub-image processing is the selection of the target, i.e. set of sky positions and a set of parameters. The system matches and selects the pipeline from all candidate pipelines by pairing nodes and computing similarity between two adjacent nodes. From the selected pipeline, the system builds a directed graph representing the data dependencies with the nodes representing objects and edges representing all dependencies attached to an object. The graph begins with the topmost node, which is the target to be made. New edges are added starting at this trigger and expanding outward, using the dependency logic derived from lineage data. The dependency graph is built and checked recursively till the last dependency (in this case raw data from the telescope).

Each node in the graph is associated to an object. Each object is identified with a unique ID. The unique ID is used as a key when searching the database for provenance information related to a particular object. Each unique ID is also associated with a specific instantiation of a class (module). Using this unique ID, we can query for all data that went into the processing of this object. If a unique ID is associated with an image, a cutout is made of the pixels of interest from this image and used as input to the module. The pixels extracted as a cutout are determined through pixel lineage.

Astro-WISE then analyzes this graph to detect modules that must be rerun, based on the processing changes required for sub-image processing. If the requested target already exists, (meaning has been processed using parameters and input data as required for the sub-image) the system will check all its dependencies up to the raw data. If all its dependencies are up-to-date then the target object is returned. If all or some dependencies are not up-todate, then these modules responsible for these dependencies will be rerun. If new versions exist of the classes that created the dependencies, then these modules will also be run-run using the new-versions of the classes. Fig. 6 shows a dependency that has been analyzed indicating which objects will be made and those that wont be reprocessed.

After assembling the pipeline and collecting all necessary input data, the sub-image is processed Source extraction is then run on the sub-image resulting in a new catalog of sky positions, and/or any other user specific processing done on the sources extracted.

4.2 Pipeline Changes

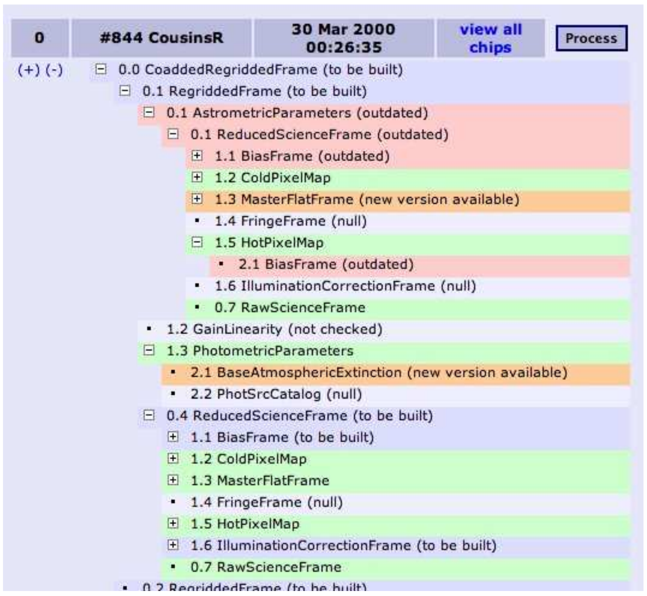
The matched and retrieved pipeline has to be modified to support sub-image processing. This also includes modifying parameters and input data. The motivation of this work is to enable users carry out a detailed analysis or a computation to a specific region on an image. For example, a user may wish to improve a given result by adjusting parameter, or might want to switch to a different algorithm on a specific region of a frame, or may want to integrate new features into existing pipelines for this purpose. In any case we build on already existing data from previous rerun to enable such changes.

We define a function Δ as a function that takes in input, the matched pipeline P_i and transforms it to another pipeline P_s (pipeline for sub-image processing). Assuming $\Delta(P_i) = P_s$, its clear that Δ is not unique for each processing. Our ultimate goal is to find Δ. i.e the changes required to transform P_i to P_s.

For each module or pipeline, we identify all tasks that are required to be changed or modified to support this new framework. Those tasks are then included in the system, as a new module, attribute or parameter without changing the overall data model. If we denote the set of all pipelines as $\mathcal{P}$, every operation performed on a pipeline, (e.g., adding parameters, modules, etc.) can be directly expressed as a (potentially partial) function $f : \mathcal{P} \rightarrow \mathcal{P}$. Then Δ is defined by such functions. Our results depend on making these functions part of Astro-WISE. These functions are dynamically loaded whenever a user requests to process a sub-image. The changes made to the modules for image analysis are specific and are beyond the scope of this paper. We present two examples below to demonstrate the kind of changes required for sub-image processing:

- **Astrometric Parameters**:A critical step for astronomical processing is deriving an 'Astrometric solution'. This is derived by fitting distortion polynomials to images, taking into account the objects seen. For accurate results, several reference stars are used to derive the final solution. For the case of sub-image processing, such a process would fail since reference stars on the sub-image will be very few. In such cases, we use the astrometric solution of a full image or another set of images of the same field and processed using the same parameters as needed to process the sub-image. The solution is then modified and fitted to the pixels of the sub-image.
- **OverScan Correction**; The Overscan is a number of rows and columns created by doing a few empty readout cycles before the detector is read. They appear as blank regions attached to the image and serve as an individual BIAS that comes along with every image. A BIAS shows the electronic noise of the camera and possible systematics. It represents the actual state of the camera at the time the exposure was taken. Each image has overscan rows and columns attached to it. During data reduction, each image must be corrected for an overscan. The overscan correction is done by smoothing the overscan regions and subtracting it from the regions with data. After this correction the overscan region is trimmed off. For the case of sub-image processing when extracting the sub-image from the full-image the algorithm must be aware of the overscan regions. The overscan regions corresponding to the sub-image are also extracted and used to perform an overscan correction on the sub-image. The trimming operation is not done in this case since the sub-images do not have overscan rows and columns.

4.2.1 *Parameter and Attribute Selection*



0	#844 CousinsR	30 Mar 2000 00:26:35	view all chips	Process

```
(+) (-)  ⊟ 0.0 CoaddedRegriddedFrame (to be built)
           ⊟ 0.1 RegriddedFrame (to be built)
             ⊟ 0.1 AstrometricParameters (outdated)
               ⊟ 0.1 ReducedScienceFrame (outdated)
                 ⊞ 1.1 BiasFrame (outdated)
                 ⊞ 1.2 ColdPixelMap
                 ⊞ 1.3 MasterFlatFrame (new version available)
                 • 1.4 FringeFrame (null)
                 ⊟ 1.5 HotPixelMap
                   • 2.1 BiasFrame (outdated)
                 • 1.6 IlluminationCorrectionFrame (null)
                 • 0.7 RawScienceFrame
             • 1.2 GainLinearity (not checked)
             ⊟ 1.3 PhotometricParameters
               • 2.1 BaseAtmosphericExtinction (new version available)
               • 2.2 PhotSrcCatalog (null)
             ⊟ 0.4 ReducedScienceFrame (to be built)
               ⊞ 1.1 BiasFrame (to be built)
               ⊞ 1.2 ColdPixelMap
               ⊞ 1.3 MasterFlatFrame
               • 1.4 FringeFrame (null)
               ⊞ 1.5 HotPixelMap
               ⊞ 1.6 IlluminationCorrectionFrame (to be built)
               • 0.7 RawScienceFrame
         • 0.2 RegriddedFrame (to be built)
```

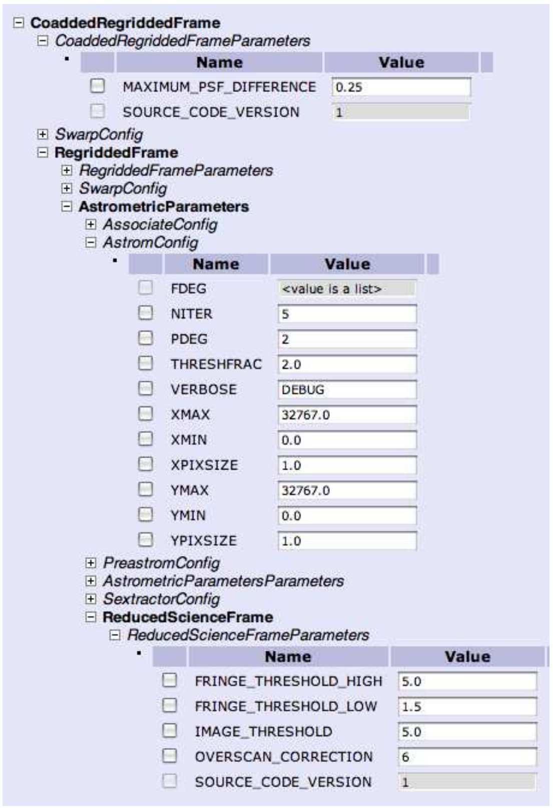
Figure 6: A tree view is given of the target(s). This tree view gives an overview of the target dependencies. Green dependencies are up-to-date, red dependencies are out-of-date and for orange dependencies indicate a newer version exists.

Lineage in Astro-WISE is stored in a database as persistent objects or links to other persistent objects. A query to the database would provide all information related to the processing history and to all locations of all stored associated files, attributes and objects. We offer a programmatic interface (in python) for querying the database for lineage information. We implemented a query language that is a natural extension to Python that incorporates data lineage in the query syntax. The query language allows standard set operators, boolean comparisons,boolean operators, and a regular expression matching function. We also identified basic operations that are useful for common querying tasks over provenance store that simplify the query syntax. Some examples are listed below;

- `get_inverse_properties(obj)` returns all objects that used *ob*
- `get_dependencies(obj)` returns all attributes of *obj*
- `get_onthefly_dependencies(obj)` returns all dependencies of *obj*
- `info(obj)` displays a lineage tree for *obj*

The example below displays a lineage tree for any object. This particular examples shows the parameters that were passed into a program called Swarp (Use for Co-addition)

```
awe> regrid = RegriddedFrame()
awe> regrid.swarpconf.info()
SwarpConfig: <astro.main.Config.SwarpConfig
           object at 0x1d893ad0>
 +-CELESTIAL_TYPE: NATIVE
 +-CENTER: ([111.428571428571, -0.957446808510638O1]
 +-INTERPOLATE: N
 +-OVERSAMPLING: 0
```



```
⊟ CoaddedRegriddedFrame
  ⊟ CoaddedRegriddedFrameParameters
```

	Name	Value
☐	MAXIMUM_PSF_DIFFERENCE	0.25
☐	SOURCE_CODE_VERSION	1

```
  ⊞ SwarpConfig
  ⊟ RegriddedFrame
    ⊞ RegriddedFrameParameters
    ⊞ SwarpConfig
    ⊟ AstrometricParameters
      ⊞ AssociateConfig
      ⊟ AstromConfig
```

	Name	Value
☐	FDEG	<value is a list>
☐	NITER	5
☐	PDEG	2
☐	THRESHFRAC	2.0
☐	VERBOSE	DEBUG
☐	XMAX	32767.0
☐	XMIN	0.0
☐	XPIXSIZE	1.0
☐	YMAX	32767.0
☐	YMIN	0.0
☐	YPIXSIZE	1.0

```
      ⊞ PreastromConfig
      ⊞ AstrometricParametersParameters
      ⊞ SextractorConfig
      ⊟ ReducedScienceFrame
        ⊟ ReducedScienceFrameParameters
```

	Name	Value
☐	FRINGE_THRESHOLD_HIGH	5.0
☐	FRINGE_THRESHOLD_LOW	1.5
☐	IMAGE_THRESHOLD	5.0
☐	OVERSCAN_CORRECTION	6
☐	SOURCE_CODE_VERSION	1

Figure 7: A web service interface for parameters selection

```
 +-PIXEL_SCALE: 0.2
 +-PROJECTION_TYPE: TAN
 +-RESAMPLING_TYPE: LANCZOS3
 +-SUBTRACT_BACK: N
 +-object_id: '83A31678D4D49F0AE0407D81E60E38BA'

---- some text missing -----
awe>
```

To change any parameter during processing, the command below would suffice,

```
awe> regrid = RegriddedFrame()
awe> regrid.swarpconf.RESAMPLING_TYPE = 'NEAREST'
```

In addition, however, we have also defined a web interface exemplified in Fig. 7 which shows the parameters that were used during processing. The same interface can be used change parameters as required for sub-image processing.

4.2.2 *Dependency CutOuts*

This service defines the *boundingbox* and provides methods to create cutouts of a requested size from the all dependent images (i.e. images generated or used in the processing of an object). The *boundingbox* defines size of the image cutouts. This service is implemented by the `Retrieve()` method explained in section 4.4

Given any $2D$-dimensional image Z with and any point $(x, y) \in Z$, we define a sub-image I as a set of points such that $\{(x, y)|(x, y) \in Z \wedge (x_0 < x < x_1), (y_0 < y < y_1)\}$ where (x_0, y_0) and (x_1, y_1) are lower and upper coordinates of the *boundingbox*. The *boundingbox* is a tuple (x_0, y_0, x_1, y_1). If C represents a set of the pixel coordinates of a celestial object, then $C \subseteq I$. The set of all points C is determined using the algorithm below;

- Select object and corresponding sky coordinates
- Get size on sky
- Scale it by the factor n (pixel scale)
- Convert into pixel coordinates and compute (x_0, y_0, x_1, y_1) for each image where the sub-image is to be extracted

4.3 Book Keeping

We describe the management of image cutouts and how we keep track of dependencies between images, cutouts, sub-images and processed regions on an Image.

Each file in Astro-WISE has a unique name, and this unique name is stored in the database as part of the metadata. In Astro-WISE, the class which represents objects that have an associated data file is called a `DataObject`. Every instance of class `DataObject` or every class which is derived from class `DataObject` has an associated data file, which is identified by the unique object identifier.

The object identifier and the object type of a `DataObject` are used as reference to identify the relationship between the data file and the `DataObject`. As soon as a user performs processing on a `DataObject`, a new `DataObject` is created and stored with a link to the parent object.

Every time we process an image a new `DataObject` is created. A request to store the `DataObject` will also store the associated image (if the image is not stored before) or a link to the image. Therefore, for every cutout we process a new `DataObject` is created and a store command would want to store the cutout. Since cutouts are made from existing images, storage of the same will duplicate pixel data in the system and will make history tracking complicated. To solve this, the `DataObject` for the sub-image and image from which the cutout was made remains the same. Therefore a request to store the new `DataObject` will store a link to the parent image from which the cutout was made, rather than storing pixel data.

This was implemented by defining three attributes, `sub-image`, `filename` and `pathname`. The `sub-image` attribute is a tuple with the coordinates of the bottom left corner and top right corner of sub-image, the `filename` attribute refers the name of image-file and the `pathname` attribute signifies two things: the filename of the sub-image, and that the file is local on the processing node. If a `DataObject` has an attribute `sub-image`, then `pathname` refers to the name of the sub-image and `filename` attribute refers to the image from which the cutout was made otherwise both the `filename` and `pathname` will refer to the same image-file.

4.4 Retrieve()

If the `DataObject` has the `sub-image` attribute set, the `retrieve` method implements arbitrary cutout coordinate specification algorithm in addition to file renaming. The `retrieve` method uses bounding box coordinates, specified by the `sub-image` attribute, and the name of the file and implements the image cutouts algorithm and retrieval of the cutouts.

For example, if a user wants to retrieve a file associated with `DataObject` (data_obj). He can use the `retrieve` method from the `DataObject` class to retrieve the file (i.e., `data_obj.retrieve()`). The `DataObject` `retrieve` method in turn calls the corresponding retrieve method from a `StorageObject` (this class defines a retrieve and a store method).

If the `DataObject` has the attribute `sub-image` set, the retrieve method will return a cutout of the data file attached to the `DataObject` specified with the `filename` attribute. The same method renames the retrieved cutout and the new name is referred to with the `pathname` attribute.

All image data is stored on the *distributed* data-servers. Clients access the data-servers using the standard HTTP protocol. Network bandwidth is a relatively scarce resource in distributed processing. We conserve our bandwidth by taking advantage of the fact that, requests for image cutouts are sent to the data-servers and only cutouts are delivered to the processing node.

This framework improves the performance of the application. Rather than loading the file into memory and extracting the sub-image, the image server reads a consecutive byte stream of the headers and requested pixel data from the disk and sends this data to the client machine for processing. This is made possible through our pixel lineage embedded into the system.

4.5 Smart Processing

The basic idea behind Astro-WISE is to process only unprocessed data. We want to minimize the processing tasks by guiding the user to useful and processed information. With this framework, users can now focus on processing regions that have objects/sources of their specific interests, rather than processing the full frame and discarding what does not interest them. If another user would like to process a region that has been processed before, the system should indicate that such a region has been processed before and returns the result.

From section 4.3, we indicated that when cutouts are made, these cutouts are not stored, but rather than a link to the DataObject where cutout was made is stored. This implies that all sub-images processed from the same frame will have the same dependency. Therefore a query to check if an object is processed might produce false results. Since the lineage of the processed sub-image (or probably the processed full frame) point to the same DataObject. To solve the problem of finding which regions (sub-images) have been processed, we introduce the idea of a *Coverage-Map* as presented in the next section.

4.5.1 Coverage-Map

The Coverage-Map is based on the observation that all DataObjects linked to the sub-image refer to the same raw frame. For example, if two or more sub-images were processed from the same raw frame, the dependency of both images will refer to the same raw science frame. Once a query is constructed to check if a sub-image is processed, it will fail or pass with wrong results.

A *Coverage-Map* as a way of representing which regions on an image have been processed. Each pixel is assigned a value of 1 (good/processed) or 0 (bad/unprocessed). Fig. 5 represents an example of a *Coverage-Map*, with regions $\mathcal{R}1 - \mathcal{R}9$ in the Fig 5 representing processed regions in a frame. The *Coverage-Map* is implemented as a pixel-map in the system. The pixel-map has the same dimensionality as the image being processed. If a user requests to process a region (sub-image) on a frame, the coverage-map is checked to establish if that region has been processed fully or partially as follows:

Given a binary $2\mathcal{D}$ image (coverage map in our case) with a regions $\mathcal{R}$ in $\{\mathcal{R}_1, \mathcal{R}_2, ...\mathcal{R}_k\}$, where $\mathcal{R}_i$ represents all processed regions in $\mathcal{R}$. if $\mathcal{R}_i$ is a set of points (x, y) over $\mathcal{R}$, given an arbitrary region $\mathcal{R}'$, we find if any points (x, y) in $\mathcal{R}'$ that do (not) exist in any $\mathcal{R}_i$. This is implemented as shown in algorithm 1. If the region has been processed before then a link to the processed frame is returned, else a False is returned implying that the region or part of the region has not been processed before.

```
Data: CoverageMap, RawImage, mathcalR'
Result: Booloen, True or Flase, if True, link to
        Proccessedframe
begin
    if RawImage has CoverageMap then
        $\mathcal{R} \longleftarrow \{R_1, R_2, ...., R_k\}$
        for $\mathcal{R}' \in \mathcal{R}$ do
            if $\mathcal{R}' \subset \mathcal{R}_i$ then
                return Link
            end
            else if $\mathcal{R}' \supset \mathcal{R}_i$ then
                return False
            end
            else if $\mathcal{R}' = \mathcal{R}_i$ then
                return False
            end
            else if $\mathcal{R}_i \supset \mathcal{R}'$ then
                return Link
            end
            else if $\mathcal{R}' \cap \mathcal{R}_i$ then
                return False
            end
            else if $\mathcal{R}' \supset R_i$ then
                return False
            end
            else
                Continue
            end
        end
    end
end
```

Algorithm 1: Checking for processed regions in the CoverageMap

4.5.2 *Update*

A User might decide to process an image by sub-images. This will create several new FileOjects and DataObjects all linked to the same RawFrame Object even if all the objects share the same input data sets, parameters and methods. (A FileObject contains information of a file, such as file size, creation date, hash value, and URI, etc.) This is inefficient, not only does it complicate data management but also complicates history tracking.

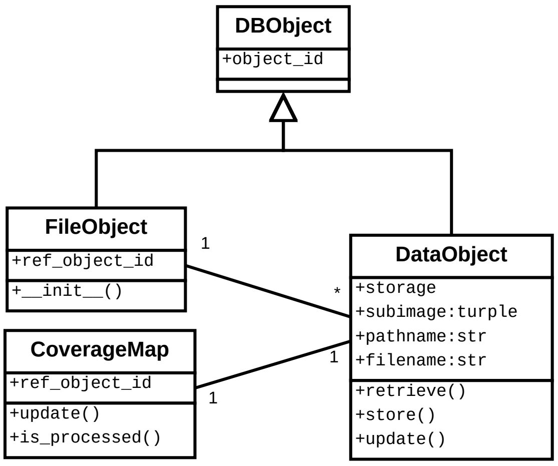


Figure 8: Class diagram to support storage of multiple processed subimages to the same FileObject

In such cases, we store all pixel data in the same FileOject. Fig. 8 shows a UML class diagram that shows the relationship between the FileObject and DataObject. For example, for all sub-images processed from the same RawFrame, only one FileObject is created and populated with pixel data that has been processed. (e.g Fig 5. The other regions that have not been processed are masked as invalid. If another region from the same RawFrame is processed, using different parameters, then a new FileOject is created and updated/stored accordingly.

Intuitively, full-images can be split up into sub-images and these sub-images processed in parallel way on different resources. After the competition of each process, another process combines the processed sub-images in to final mosaic. The advantages of such a method are obviously enormous.

4.6 MyDB and Context

Every user in Astro-WISE is provided with a database called MyDB. Within the isolated MyDB environment, users are provided with full read, write and execute privileges. Here users can create, add and delete DataObjects. However, all the processing machinery goes throughout the system. Results of an exploratory nature are stored in MyDB, where further analytical queries may be performed by the user before they can be published or viewed by other users. MyDB allows an entire explore/analyze workflow to be completed before transferring the final results to the public and also allows a personalized user query/processing history.

Context, on the other hand, allows you to filter from the ocean of data objects in Astro-WISE the subset of data objects that you want to be currently visible to you and your processes. The philosophy here is to increase efficiency in survey processing by sharing useful public data (such as calibration data). The philosophy of Context is to allow users to select which data they want to see in their data access scope in addition to public data from Astro-WISE projects. In other words, by configuring your Astro-WISE Context, you define the logical subspaces in the ensemble of data objects in Astro-WISE in which you access data and in which you create data. So with Context and MyBD, we are able to manage, any amount of public and useful data available in Astro-WISE.

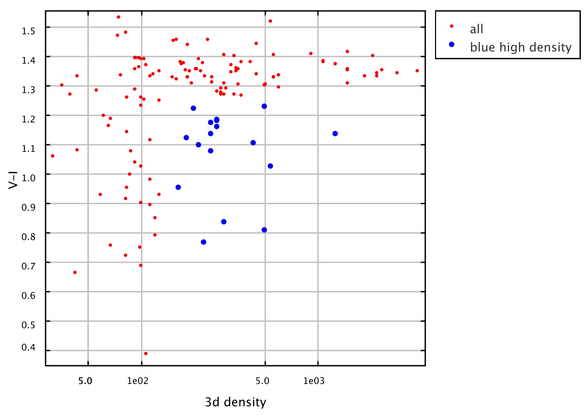


Figure 9: Density-Color of the galaxies processed from the image in Fig. 10. In this case we are interested in Blue galaxies (low V-I) in medium to high density regions which are possibly interacting with their environment. We only take the addresses of the blue galaxies and reprocess them using the sub-image pipeline

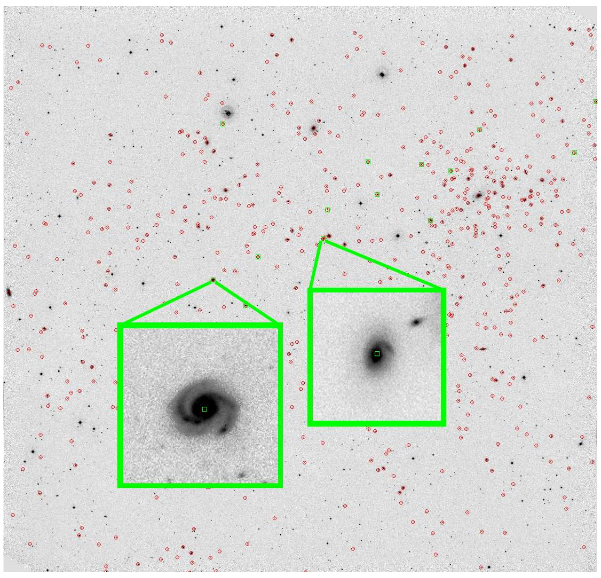


Figure 10: Science frame from which the sources of figure 9 are extracted. Sources highlighted in green are the selected blue galaxies of Fig. 9. To analyze these sources in higher detail, only their sub-images have to be processed instead of the entire image

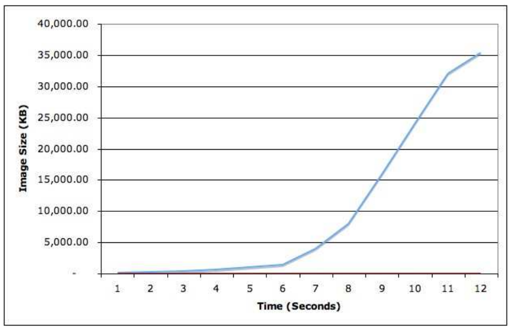


Figure 11: A plot showing the time it takes to process sub-images of different sizes

5. USE CASE: ANALYZING TRANSITIONING GALAXIES

We demonstrate the use of provenance using a use-case of analyzing transitioning galaxies. These are galaxies that fall into galaxy clusters that interact with their environment. Initially a full image is processed and an initial photometric catalog of the sources on the image is extracted. The density of galaxies around each source is calculated using the galaxy position. The magnitudes and densities of galaxies that undergo a transitional phase can be identified (the blue galaxies in Fig. 9). Out of hundreds of galaxies that were observed in this processing only transitioning (blue) galaxies will then be further analyzed by extracting cutouts from the raw images where these galaxies lie and reprocessing only these cutouts to estimate more complex and time consuming parameters such as quantifications of the internal structure of the galaxy. To identify the images required for this task and the position of the galaxy in all images, we work backwards from the galaxy through all the dependencies. The other inputs (any other sub-images, calibration objects, processing parameters, etc) to the sub-image pipeline are also selected from the initial lineage recorded during the initial processing of the full image. By performing selective processing we save hours/days/weeks of computational time.

5.0.1 File Retrievals

One common characteristic of all dataflow programming frameworks is the requirement of locally staged data for processing. A computation can not start before all required inputs are available locally at the processing node because data has to be transfered from archives to processing nodes. This is one of the performance bottlenecks in such systems. In this framework cutouts are requested from storage nodes and only the cutouts are sent over the networks.

In this test, we estimate how much time is required to download 24 sub-images from a data Service to local PC compared to downloading the 24 full images on a network connection with an estimated bandwidth of 1Gbit/s. We did four parallel downloads to a local PC of 24 (sub)images for a total of 96 images. Below are the results;

```
Sub-images
Downloaded: 24 files, 544K in 0.4s (1.22 MB/s)
```

```
Downloaded: 24 files, 544K in 0.4s (1.21 MB/s)
Downloaded: 24 files, 544K in 0.4s (1.22 MB/s)
Downloaded: 24 files, 544K in 0.7s (737 KB/s)

Full-images
Downloaded: 24 files, 384M in 22s (17.7 MB/s)
Downloaded: 24 files, 384M in 21s (17.9 MB/s)
Downloaded: 24 files, 384M in 23s (17.1 MB/s)
Downloaded: 24 files, 384M in 23s (16.6 MB/s)
```

The time in seconds is the total time that the actual transfer took, from the first byte to the last byte of each file. We notice a significant difference when transferring sub-image compared to full-images. We tried out the same test on a link of an estimated bandwidth of 6Mbit/s. In that case we only downloaded one group of 24 files sequentially to prevent very long wait times.

```
Subimages
Downloaded: 24 files, 544K in 1.3s (423 KB/s)
Full images
Downloaded: 24 files, 384M in 11m 16s (582 KB/s)
```

As you notice downloading the sub-images on a 6Mbit/s takes almost the same amount required as on a 1Gbit/s. However, the results for Full images are shocking and might render the system unusable on such links. With this framwork even users at remote locations using low bandwidth links carry out image processing just like any other person processing data locally.

6. RELATED WORK

Most of the popular astronomical software languages and packages (ITT Interactive Data Language[1], European Southern Observatory Munich Image Data Analysis System[2] [13], Image Reduction and Analysis Facility [10]) assume a standalone approach to work with the data. In the case of all these packages, the storage and access of the raw and processed data is a responsibility of the user. However, the astronomical data processing community (or the scientific community in general) is becoming very sophisticated. Data reduction and analysis has become complex [19], data rates have kept pace with advances in processing power (doubling roughly every two years), and the dimensionality of data is increasing [17]. As a result, several scalability issues have arisen which range from ensuring good performance, to handling large amounts of data, to capturing provenance, and to providing interfaces to interact with a large number of archives. Distributed or grid systems [21] have been developed to address performance and dataset concerns. Such systems like [3], provide a scalable infrastructure for running image pipelines in a distributed way.

Provenance-aware scientific workflow systems [9] have been considered as the paradigm for representing and managing complex distributed scientific computations. Systems such as those surveyed in [8] and [4] have enabled scientists to carry out complex scientific computations while capturing provenance. Despite these developments, little or no support exists in current systems to trace lineage and pixel level and most models do not allow end-users to use lineage data in scientifically meaningful way in particular to improve scientific processes. Our lineage model introduced in this paper does capture lineage at the finest detail (e.g., a pixel transformation process). This lineage captured is then used for various scientific processes but most specifically as introduced in this paper, this lineage at pixel level has been used for sub-image processing.

While tracing lineage at pixel level is missing out in most workflow systems, the level of granularity at which lineage is collected is linked to the particular needs of provenance requirements. Different provenance models are in use today and also several workflow management systems do exploit provenance information for different purposes. The use of lineage we describe in this paper is analogous to how other authors have used lineage to solve some use-cases. For example in [2], [16] and [20] provenance has been used for results verification and to prove robustness of methods, in Kepler [1] provenance has been used to enable smart "reruns" and process simplification of previously executed workflows and in [15] and [6], provenance has been used for interactive design of workflows.

We could not find anywhere in literature where sub-image processing has been done and therefore we have no comparative analysis to evaluate if there could be advantages or disadvantages of our approach against any other approach. However, we show that in this current data deluge sub-image processing as compared to full-image supports research in distributed communities by transferring and processing a minimal set of data as required for processing.

7. DISCUSSION AND CONCLUSIONS

We have described a new framework that leverages data lineage and provenance to aid in selective retrieval and processing of data. We argue that sub-image processing is a powerful methodology that will provide efficient solutions for what are otherwise manual, time-consuming tasks.

This methodology provides scalable and easy-to-use primitives for reprocessing of data. With this kind of processing, even users at remote research centres could comfortably run and process data, without limitations of huge data transfers and limitations of resources on local clients. We have proposed efficient algorithms and intuitive interfaces for realizing these primitives in an astronomical system.

However our approach is not foolproof, and there are cases where it may fail to produce the results a user expects. For example, if a user applies the methodology to a processing that involves neighboring pixels to determine a result of a pixel (e.g., derivation of astronomical parameters solution) the pipelines is likely to fail. However, when such a processing fails, or produces poor results, the user can initially process the full image and extract all parameters/needed to aid in processing of other sub-images derived from the same image. The effect shall be slower performance for the start, which improves significantly while processing other sub-images.

There are many avenues for future work. Although we reduced the domain from a full frame to a sub-image, we

[1]http://www.ittvis.com/ProductServices/IDL.aspx
[2]http://www.eso.org/sci/data-processing/software/esomidas

intend to further transform scientific systems to process pixels rather than images. We are currently investigating how we can use databases to aid in such processing. By loading data into a database and then performing all processing inside the database, then accurate lineage can be captured and traced.

8. REFERENCES

[1] I. Altintas, B. Ludaescher, S. Klasky, and M. A. Vouk. Introduction to scientific workflow management and the kepler system. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 205, New York, NY, USA, 2006. ACM.

[2] E. W. Anderson, J. P. Ahrens, K. Heitmann, S. Habib, and C. T. Silva. Provenance in comparative analysis: A study in cosmology. *Computing in Science and Engg.*, 10(3):30–37, 2008.

[3] K. G. Begeman, A. N. Belikov, D. R. Boxhoorn, F. Dijkstra, E. A. Valentijn, W.-J. Vriend, and Z. Zhao. Merging grid technologies. *Journal of Grid Computing*, 8:199– 221, 2010.

[4] R. Bose and J. Frew. Lineage retrieval for scientific data processing: a survey. *ACM Comput. Surv.*, 37(1):1–28, 2005.

[5] M. R. Calabretta and E. W. Greisen. Representations of celestial coordinates in fits. *A&A*, 395(3):1077–1122, dec 2002.

[6] T. Ellkvist, D. Koop, E. W. Anderson, J. Freire, and C. Silva. Using provenance to support real-time collaborative design of workflows. *Provenance and Annotation of Data and Processes: Second International Provenance and Annotation Workshop, IPAW 2008, Salt Lake City, UT, USA, June 17-18, 2008. Revised Selected Papers*, pages 266–279, 2008.

[7] J. Freire, D. Koop, and L. Moreau, editors. *Provenance and Annotation of Data and Processes: Second International Provenance and Annotation Workshop*. Springer-Verlag, Berlin, Heidelberg, 2008.

[8] J. Freire, D. Koop, E. Santos, and C. T. Silva. Provenance for computational tasks: A survey. *Computing in Science and Engineering*, 10(3):11–21, 2008.

[9] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers. Examining the challenges of scientific workflows. *Computer*, 40(12):24–32, 2007.

[10] P. Greenfield. Reaching for the stars with python. *Computing in Science and Engg.*, 9(3):38–40, 2007.

[11] E. W. Greisen, M. R. Calabretta, F. G. Valdes, and S. L. Allen. Representations of spectral coordinates in fits. *A&A*, 446(2):747–771, feb 2006.

[12] K. Jaan. *Numerical Methods in Engineering with Python*. Cambridge University Press, 2005.

[13] B. K, editor. *Information Handling in Astronomy - Historical Vistas*. Springer-Verlag, 2002.

[14] J. Mwebaze, D. Boxhoorn, and E. Valentijn. Astro-wise: Tracing and using lineage for scientific data processing. *Network-Based Information Systems, International Conference on*, pages 475–480, 2009.

[15] C. Scheidegger, H. Vo, D. Koop, J. Freire, and C. Silva. Querying and creating visualizations by analogy. *Visualization and Computer Graphics, IEEE Transactions on*, 13(6):1560 –1567, nov.-dec. 2007.

[16] Y. Simmhan, B. Plale, and D. Gannon. Karma2: Provenance management for data-driven workflows. *Int. J. Web Service Res.*, 5(2):1–22, 2008.

[17] A. S. Szalay. The sloan digital sky survey and beyond. *SIGMOD Rec.*, 37(2):61–66, 2008.

[18] E. A. Valentijn, J. McFarland, J. Snigula, K. Begeman, D. Boxhoorn, R. Renegelink, E. Helmich, P. Heraudeau, G. V. Kleijn, R. Vermeij, W.-J. Vriend, and M. J. Tempelaar. Astro-wise: Chaining to the universe. In *Astronomical Data Analysis Software and Systems XVI, ASP Conference Series*, volume 376, 2007.

[19] F. Wang, J. Luo, H. Deng, B. Liang, and K. Ji. C-swf: A lightweight scientific workflow system for astronomical data processing. *Computer Science and Engineering, International Workshop on*, 2:64–67, 2009.

[20] S. Wong, S. Miles, W. Fang, P. Groth, and L. Moreau. Provenance-based validation of e-science experiments. *The Semantic Web –ISWC 2005*, 2005///.

[21] J. Yu and R. Buyya. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Rec.*, 34(3):44–49, 2005.