**Noname manuscript No.**
(will be inserted by the editor)

# Parameterized Specification, Configuration and Execution of Data-Intensive Scientific Workflows

**Vijay S. Kumar · Tahsin Kurc · Varun Ratnakar · Jihie Kim · Gaurang Mehta · Karan Vahi · Yoonju Lee Nelson · P. Sadayappan · Ewa Deelman · Yolanda Gil · Mary Hall · Joel Saltz**

Received: date / Accepted: date

**Abstract** Data analysis processes in scientific applications can be expressed as coarse-grain workflows of complex data processing operations with data flow dependencies between them. Performance optimization of these workflows can be viewed as a search for a set of optimal values in a multidimensional parameter space consisting of input performance parameters to the applications that are known to affect their execution times. While some performance parameters such as grouping of workflow components and their mapping to machines do not affect the accuracy of the analysis, others may dictate trading the output quality of individual components (and of the whole workflow) for performance. This paper describes an integrated framework which is capable of supporting performance optimizations along multiple such parameters. Using two real-world applications in the spatial, multidimensional data analysis domain, we present an experimental evaluation of the proposed framework.

**Keywords** scientific workflow · performance parameters · semantic representations · Grid · application QoS

V. S. Kumar · P. Sadayappan
Dept. of Computer Science and Engineering, Ohio State University, Columbus, OH 43210

V. Ratnakar · J. Kim · G. Mehta · K. Vahi · Y. Lee · Y. Gil · E. Deelman
Information Sciences Institute, University of Southern California, Marina del Rey, CA 90292

M. Hall
School of Computing, University of Utah, Salt Lake City, UT 84112

T. Kurc · J. Saltz
Center for Comprehensive Informatics, Emory University, Atlanta, GA 30322

*Corresponding author*: Vijay S. Kumar
E-mail: kumarvi@cse.ohio-state.edu, phone: 614-284-6447, fax: 404-727-4992

## 1 Introduction

Advances in digital sensor technology and the complex numerical models of physical processes in many scientific domains are bringing about the acquisition of enormous volumes of data. For example, a dataset of high resolution image data obtained from digital microscopes or large scale sky telescopes can easily reach hundreds of Gigabytes, even multiple Terabytes in size[1]. These large data volumes are transformed into meaningful information via data analysis processes. Analysis processes in scientific applications are expressed in the form of workflows or networks of interdependent components, where each component corresponds to an application-specific data processing operation. Image datasets, for instance, are analyzed by applying workflows consisting of filtering, data correction, segmentation, and classification steps. Due to the data and compute intensive nature of scientific data analysis applications, scalable solutions are required to achieve desirable performance. Software systems supporting the analysis of large scientific datasets implement several optimization mechanisms to reduce execution times. First, workflow management systems take advantage of distributed computing resources in the Grid. The Grid environment provides computation and storage resources; however, these resources are often located at disparate sites managed within different security and administrative domains. Workflow systems support execution of workflow components at different sites (Grid nodes) and reliable, efficient staging of data across the Grid nodes. A Grid node may itself be a large cluster system or a potentially heterogeneous and dynamic collection of machines. Second, for each portion of a workflow mapped to such clusters, they

[1] DMetrix array microscopes can scan a slide at 20x+ resolution in less than a minute. The Large Synoptic Survey Telescope will be able to capture a 3.2 Gigapixel image every 6 seconds, when it is activated.

enable the fine-grain mapping and scheduling of tasks onto such machines.

The performance of workflows is greatly affected by certain parameters to the application that direct the amount of work to be performed on a node or the volume of data to be processed at a time. The optimal values of such parameters can be highly dependent on the execution context. Therefore, performance optimization for workflows can be viewed as a search for a set of optimal values in a multidimensional parameter space, given a particular execution context. *Workflow-level performance parameters* include grouping of data processing components comprising the workflow into 'meta-components', distribution of components across sites and machines within a site, and the number of instances of a component to be executed. These parameters impact computation, I/O, and communication overheads, and as a result, the total execution time. Another means of improving performance is by adjusting *component-level performance parameters* in a workflow. An example of such a parameter is the data chunk size in applications which analyze spatial, multidimensional datasets. Another example is the version of the algorithm employed by a component to process the data.

We classify workflow-level and component-level performance parameters into two categories:
(i) *Accuracy-preserving* parameters (such as data chunk size) can affect the performance of an operation without affecting the quality of the analysis output, and (ii) *Accuracy-trading* parameters can trade the quality of the output for performance gains, and vice-versa. An example of an accuracy-trading parameter is the 'resolution' at which image data are processed. A classification algorithm might process low-resolution images quickly, but its classification accuracy would likely be lower compared to that for higher resolution images. When optimizations involve accuracy-performance trade-offs, users may supplement their queries with application-level quality-of-service (QoS) requirements that place constraints on the accuracy of the analysis (Kumar et al, 2008b). For instance, when images in a dataset are processed at different resolutions to speed up the classification process, the user may request that a certain minimum accuracy threshold be achieved.

In this paper, we describe the design and implementation of a framework that can support application execution in a distributed environment and enable performance optimization via manipulation of accuracy-preserving and/or accuracy-trading parameters. We present an instance of our framework that integrates multiple subsystems at different levels:

- Wings (Gil et al, 2007) to facilitate high-level, semantic descriptions of application workflows
- Pegasus (Deelman et al, 2004), Condor (Thain et al, 2005), and DataCutter (Beynon et al, 2001) to support scalable workflow execution across multiple institutions and on distributed clusters within an institution
- ECO (Chen et al, 2005) to enable compiler optimizations for fine-grain computations executing on specific resources

Application developers and end-users can use our framework to provide high-level, semantic descriptions of application structure and data characteristics. As our initial focus is on addressing performance requirements in spatial, multidimensional data analysis applications, we have developed extensions to core ontologies in Wings to be able to describe spatial datasets and also to enable automatic composition and validation of the corresponding workflows. Once a workflow has been specified, users can adjust workflow-level and component-level parameters based on their QoS requirements to enable performance optimizations during execution. As part of this effort, we have also extended Condor's default job-scheduling mechanism to support performance optimizations stemming from accuracy-performance related trade-offs. We show how our framework supports parameter-based optimizations for real biomedical image analysis workflows using two cluster systems located at two different departments at the Ohio State University.

## 2 Related Work

Workflow management systems for the Grid and Services Oriented Architectures, such as Taverna (Oinn et al, 2004), Kepler (Ludäscher et al, 2006) and Pegasus (Deelman et al, 2004) seek to minimize the makespan by manipulating workflow-level parameters such as grouping and mapping of a workflow's components. Our framework extends such support by providing the combined use of task- and data-parallelism and data streaming within each component and across multiple components in a workflow to fully exploit the capabilities of Grid sites that are high-end cluster systems. Glatard et. al. (Glatard et al, 2006) describe the combined use of data parallelism, services parallelism and job grouping for data-intensive application service-based workflows. Our work is in the context of task-based workflows where execution plans are developed based on abstract workflow descriptions. We also address performance improvements by adjusting domain-specific component-level parameters.

The Common Component Architecture(CCA) forum[2] addresses domain-specific parameters for components and the efficient coupling of parallel scientific components. They seek to support performance improvements through the use of external tunability interfaces (Chang and Karamcheti, 2000; Norris et al, 2004). The Active Harmony system (Chung

[2] http://www.cca-forum.org

and Hollingsworth, 2004, 2006) is an automatic parameter tuning system that permits on-line rewriting of parameter values at run-time, and uses a simplex method to guide the search for optimal parameter values. Although we share similar motivations with the above works, we target data-intensive applications running on the Grid. We account for performance variations brought about by the characteristics of dataset instances within a domain.

Our work also supports application-level QoS requirements by tuning accuracy-trading parameters in the workflows. The performance/quality trade-off problem and tuning of quality-trading parameters for workflows has been examined before in service-based workflows (Brandic et al, 2008) and component-based systems (Cortellessa et al, 2006). But these works are geared towards system-level QoS and optimization of system-related metrics such as data transfer rates, throughput and service affinity etc. Application-level QoS for workflows has been addressed in (Zhou et al, 2004; Chiu et al, 2008). Acher et al (2008) describe a services-oriented architecture that uses ontology-based semantic modeling techniques to address the variability in results produced by Grid services for medical imaging. We support trade-offs based on quality of data output from the application components and integrate such parameter tuning within a batch system like Condor.

Supporting domain-specific parameter-based optimizations requires the representation of these parameters and their relations with various performance and quality metrics in a system-comprehensible manner. In (Chiu et al, 2008), end-users are required to provide performance and quality models of expected application behavior to the system. Ontological representations of performance models have been investigated in the context of workflow composition in the Askalon system (Truong et al, 2007). Lera et al. (Lera et al, 2006) proposed the idea of developing performance-related ontologies that can be queried and reasoned upon to analyze and improve performance of intelligent systems. Zhou et. al. (Zhou et al, 2004) used rule-based systems to configure component-level parameters. While beyond the scope of this paper, we seek to complement our framework with such approaches in the near future.

## 3 Motivating Applications

Our work is motivated by the requirements of applications that process large spatial, multidimensional datasets. We use the following two application scenarios from the biomedical image analysis domain in our evaluation. Each application has different characteristics and end-user requirements.

### 3.1 *Application 1*: **Pixel Intensity Quantification (PIQ)**

Figure 1 shows a data analysis pipeline (Chow et al, 2006) (developed by neuroscientists at the National Center for Microscopy and Imaging Research) for images obtained from confocal microscopes. This analysis pipeline quantifies pixel intensity within user-specified polygonal query regions of the images through a series of data correction steps as well as thresholding, tessellation, and prefix sum generation operations. This workflow is employed in studies that involve comparison of image regions obtained from different subjects as mapped to a canonical atlas (e.g., a brain atlas). From a computational point of view, the main end-user requirements are (1) to *minimize the execution time* of the workflow while *preserving the highest output quality*, and (2) to support the execution of potentially *terabyte-sized out-of-core image data*.

### 3.2 *Application 2*: **Neuroblastoma Classification (NC)**

Figure 2 shows a multi-resolution based tumor prognosis pipeline (Kong et al, 2007) (developed by researchers at the Ohio State University) applied to images from high-power light microscopy scanners. This workflow is employed to classify image data into grades of neuroblastoma, a common childhood cancer. Our primary goal is to optimally support user queries while simultaneously meeting a wide range of application-level QoS requirements. Examples of such queries include: *"Minimize the time taken to classify image regions with 60% accuracy"* or *"Determine the most accurate classification of an image region within 30 minutes, with greater importance attached to feature-rich regions"*. Here, accuracy of classification and richness of features are application domain-specific concepts and depend on the resolution at which the image is processed. In an earlier work, we developed heuristics that exploit the multi-resolution processing capability and the inherent spatial locality of the image data features in order to provide improved responses to such queries (Kumar et al, 2008b).

The aforementioned applications differ in their workflow structure and also the complexity of their data analysis operations. The NC workflow processes a portion or chunk of a single image at a time using a sequence of operations. The end-result for an image is an aggregate of results obtained from each independently processed chunk. PIQ, on the other hand, contains complex analysis operations such as aggregations, global computations and joins across multiple datasets, that are not easily scalable. Hence, such applications require parallel algorithms for efficient processing of out-of-core data.

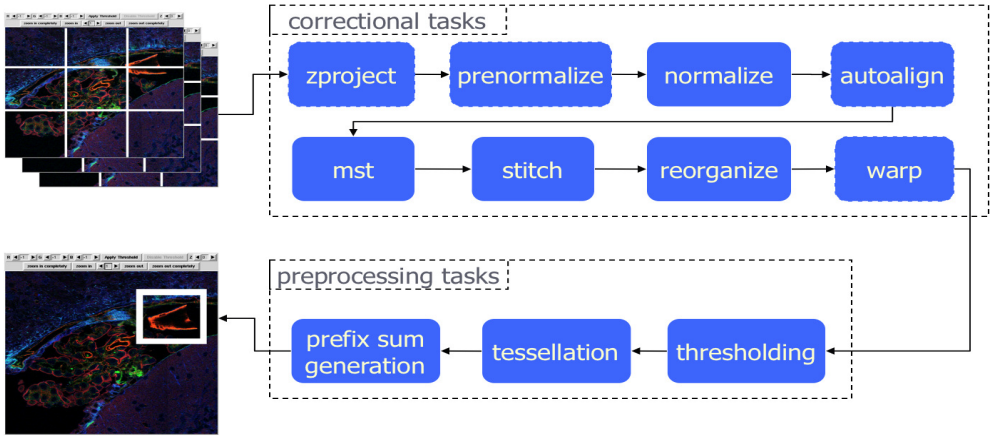



**Fig. 1** Pixel Intensity Quantification (PIQ) workflow

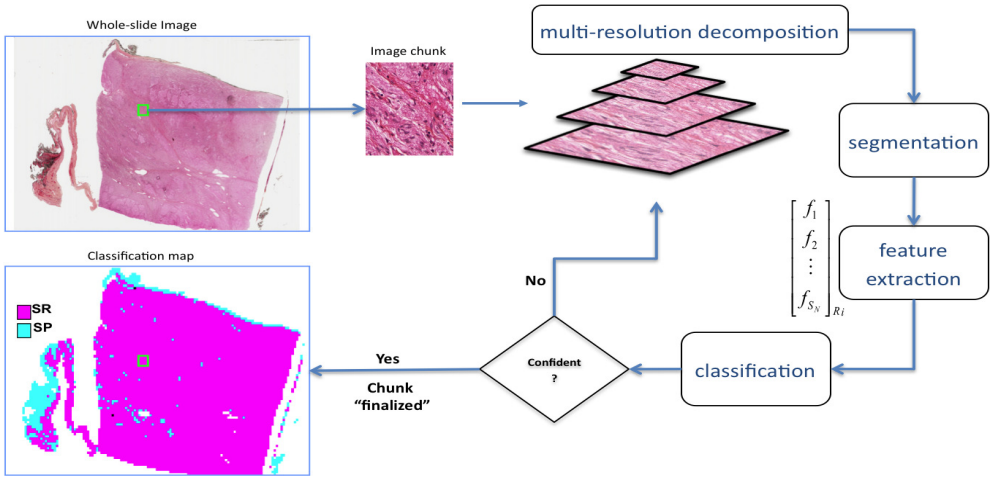



**Fig. 2** Neuroblastoma Classification (NC) workflow

## 4 Performance Optimizations

In this section we discuss several strategies for improving workflow performance. Drawing from the application scenarios, we also present parameters that impact the performance of applications in the spatial, multidimensional data analysis domain. We classify these parameters into two main categories, accuracy-preserving parameters and accuracy-trading parameters, and explain how they can influence performance and/or quality of output.

### 4.1 Accuracy-preserving Parameters

**Chunking Strategy**: Individual data elements (e.g. images) in a scientific dataset may be larger than the physical memory available on current workstations. Relying on virtual memory alone is likely to yield poor performance. In general, the processing of large, out-of-core spatial, multidimensional data is supported by partitioning it into a set of *data chunks*, each of which can fit in memory, and by modifying the analysis routines to operate on chunks of data at a time. Here, a data chunk provides a higher-level abstraction for data distribution and is the unit of disk I/O. In the simplest case, we can have a uniform chunking strategy, i.e. all chunks have the same shape and size. For 2-D data, this parameter is represented by a pair $[\mathscr{W}, \mathscr{H}]$, where $\mathscr{W}$ and $\mathscr{H}$ respectively represent the width and height of a chunk. In our work, we use this simplified strategy and refer to this parameter as `chunksize`. The `chunksize` parameter can influence application execution in several ways. The larger a chunk is, the greater the amount of disk I/O and interprocessor communication for that chunk will likely be, albeit the number of chunks will be smaller. The `chunksize` affects the number of disk blocks accessed and network packets transmitted during analysis. However, larger chunks imply a decreased number of such chunks to be processed, and this could in turn, decrease the job scheduling overheads.

Moreover, depending on the levels of memory hierarchy and hardware architecture present on a compute resource, the `chunksize` can affect the number of cache hits/misses for each component and thus, the overall execution time. For the PIQ workflow, we observed that varying `chunksize` resulted in differences in execution time. Moreover, the optimal `chunksize` for one component may not be optimal for other components; some components prefer larger chunks, some prefer square-shaped chunks over thin-striped chunks, while others may function independent of the `chunksize`.

**Component configuration**: Components in a workflow could have many algorithmic variants. Traditionally, one of these variants is chosen based on the *type* of the input data to the component and/or the type of the output data expected from the component. However, choosing an algorithmic variant can affect the application performance based on resource conditions/availability and data characteristics, even when each variant performs the analysis differently but produces the same output and preserves the output quality. In an earlier work (Kumar et al, 2008a), we developed three parallel-algorithmic variants for the *warp* component of the PIQ workflow. We observed that depending on the available resources – clusters nodes with faster processors or clusters equipped with high-speed interconnects or nodes offering higher disk I/O rates – each variant was capable of outperforming the other, and no single variant performed best under all resource conditions.

**Task Granularity and Execution Strategy**: A workflow may consist of many components. If a chunk-based processing of datasets is employed, creating multiple copies of a component instance may speed up the process through data parallelism. How components and component copies are scheduled, grouped, and mapped to machines in the environment will affect the performance, in particular if the environment consists of a heterogeneous collection of computational resources. An approach could be to treat each (component instance, chunk) pair as a task and each machine in the environment as a Grid site. This provides a uniform mechanism for execution within a Grid site as well as across Grid sites. It also enables maximum flexibility in using job scheduling systems such as Condor (Thain et al, 2005). However, if the number of component instances and chunks is large, then the scheduling and data staging overheads may assume significance. An alternative strategy is to group multiple components into *meta-components* and map/schedule these meta-components to groups of machines. Once a meta-component is mapped to a group of machines, a combined task- and data-parallelism approach with pipelined dataflow style execution can be adopted within the meta-component. When chunking is employed, the processing of chunks by successive components (such as *thresholding* and *tessellation* in the PIQ workflow) can be pipelined such that, when a component $\mathscr{C}1$ is processing a chunk $i$, then the downstream component $\mathscr{C}2$ can concurrently operate on chunk $i+1$ of the same data element. A natural extension to pipelining is the ability to stream data between successive workflow components mapped to a single Grid site, so that the intermediate disk I/O overheads are avoided.

### 4.2 Accuracy-trading Parameters

**Chunking strategy**: The `chunksize` parameter may also function as an accuracy-trading performance parameter in cases where analysis operations depend on the feature content within a data chunk. In the NC workflow, `chunksize` affects both the performance and accuracy of the analysis. The smaller the `chunksize` is, the faster the analysis of each chunk. However, small `chunksize` values (where chunks are smaller than the features being analyzed) could lead to false-negatives. If the chunk is too large, then extraneous features and noise within a chunk could impact the accuracy of analysis.

**Data resolution**: Spatial data has an inherent notion of data quality. The `resolution` parameter for spatial, multidimensional data takes values from 1 to $n$, where $n$ represents the data at its highest quality. If a multi-resolution analysis approach is adopted, then the processing of data chunks at different `resolutions` will produce output of varying quality. Execution time increases with `resolution` because higher resolutions contain more data to be processed. In some cases, data may need to be processed only at the lowest *target* resolution that can yield a result of adequate quality. It is assumed that the accuracy of analysis is maximum when data is processed at its highest resolution.

**Processing order**: The `processing order` parameter refers to the order in which data chunks are operated upon by the components in a workflow. In the NC workflow, the accuracy of analysis computed across an entire image is obtained as an aggregate of the accuracy of analysis for each chunk in the image. As aggregation is not always commutative, the order in which chunks are processed will affect the accuracy of analysis when computed across the entire image. Our previous work (Kumar et al, 2008b) with the NC workflow showed how out-of-order processing of chunks (i.e., selecting a subset of "favorable" chunks ahead of other chunks) in an image could be used to improve response (by upto a factor of 40%) to user queries that contain various quality-of-service requirements.

## 5 Workflow Composition and Execution Framework

In this section, we describe our framework to support specification and execution of data analysis workflows. The framework consists of three main modules. The *description module* implements support for high-level specification of workflows. In this module, the application structure and data characteristics for the application domain are presented to the system. This representation is independent of actual data instances used in the application and the compute resources on which the execution is eventually carried out. The *execution module* is responsible for workflow execution and takes as input, the high-level description of the workflow produced by the description module, the datasets to be analyzed and the target distributed execution environment. Lastly, the *trade-off module*, implements runtime mechanisms to enable accuracy-performance trade-offs based on user-specified quality-of-service requirements and constraints. The architecture of the framework is illustrated in Figure 3. In this paper, we evaluated a specific instance of this framework where the representative systems for each module are highlighted using blue boxes in the figure.

### 5.1 Description Module (DM)

The Description Module (DM) is implemented using the Wings (Workflow Instance Generation and Selection) system (Gil et al, 2007). In the Wings representation of a scientific workflow, the building blocks of a workflow are components and data types. Application domain-specific components are described in component libraries. A component library specifies the input and output data types of each component and how metadata properties associated with the input data types relate to those associated with the output for each component. The data types themselves are defined in a domain-specific data ontology. Wings allows users to describe an application workflow using semantic metadata properties associated with workflow components and data types at a high level of abstraction. This abstraction is known as a *workflow template*. The workflow template and the semantic properties of components and data types are expressed using the Web Ontology Language(OWL)[3]. A template effectively specifies the application-specific workflow components, how these components are connected to each other to form the workflow graph, and the type of data exchanged between the components. Figure 4(a) is a depiction of a workflow template constructed for the PIQ application. The workflow template is data-instance independent; it specifies data types consumed and produced by the components but not particular datasets. Wings can use the semantic descriptions to automatically validate a given workflow, i.e., if two components are connected in the workflow template, Wings can check whether output data types and properties of the first component are consistent with the input data types and properties of the second component. Given a workflow template, the user can specify a data instance (e.g., an image) as input to the workflow and the input argument values to each component in the workflow. Using the metadata properties of the input datasets, Wings can automatically generate a detailed specification of the workflow tasks and data flow in the form of a DAG referred to as an expanded *workflow instance* for execution. Figure 4(b) shows a workflow instance generated by Wings from the PIQ workflow template.

**Domain-specific Data Ontology:** Wings provides "core" ontologies that can describe abstract components and data types. For any new application domain, these core ontologies can be extended to capture domain-specific information. Semantic descriptions for new data types and components are maintained in domain-specific ontologies. Workflow templates for applications within that domain can then be constructed using these ontologies. The core data ontology in Wings contains OWL-based descriptions of generic data concepts such as `File`, `Collection` of `Files` of the same type, and `CollOfCollections`, as well as their metadata properties. However, our motivating applications process spatial, multidimensional data by partitioning datasets into higher-level entities called chunks. The core data ontology is not expressive enough to capture the semantics of datasets used in our motivating applications. So, we extended this core ontology (as shown in Figure 5) to express concepts and properties in the spatial, multidimensional data model so that application data flow can be conveniently described in terms of these concepts. While this ontology is not exhaustive, it is generic enough to represent typical application instances such as the PIQ and NC workflows. It can also be extended to support a wider range of data analysis applications within the domain.

Wings also provides compact representations in the template for component collections, i.e., cases where multiple copies of a component can be instantiated to perform the same analysis on different data input instances. The number of such copies for a component can be selected during the workflow instance generation stage based on the properties of the input datasets. Performance can be improved by adjusting both the task- and data-parallelism parameter as well as application performance parameters such as `chunksize`. For the PIQ and NC workflows, the `chunksize` parameter dictates the unrolling of component collections in the workflow template into a bag of component tasks in the workflow instance. As an example, assume that the value of the `chunksize` parameter chosen for a given input image was $2560 \times 2400$ pixels and resulted in each image slice being partitioned into 36 chunks. The corresponding expanded

[3] http://www.w3.org/TR/owl-ref

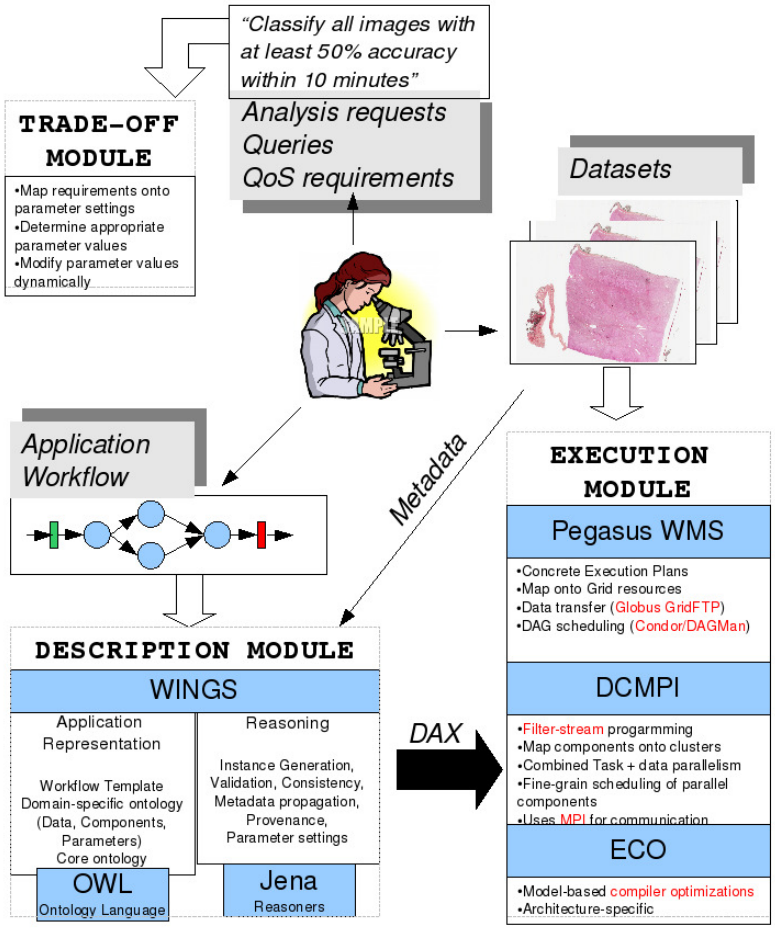
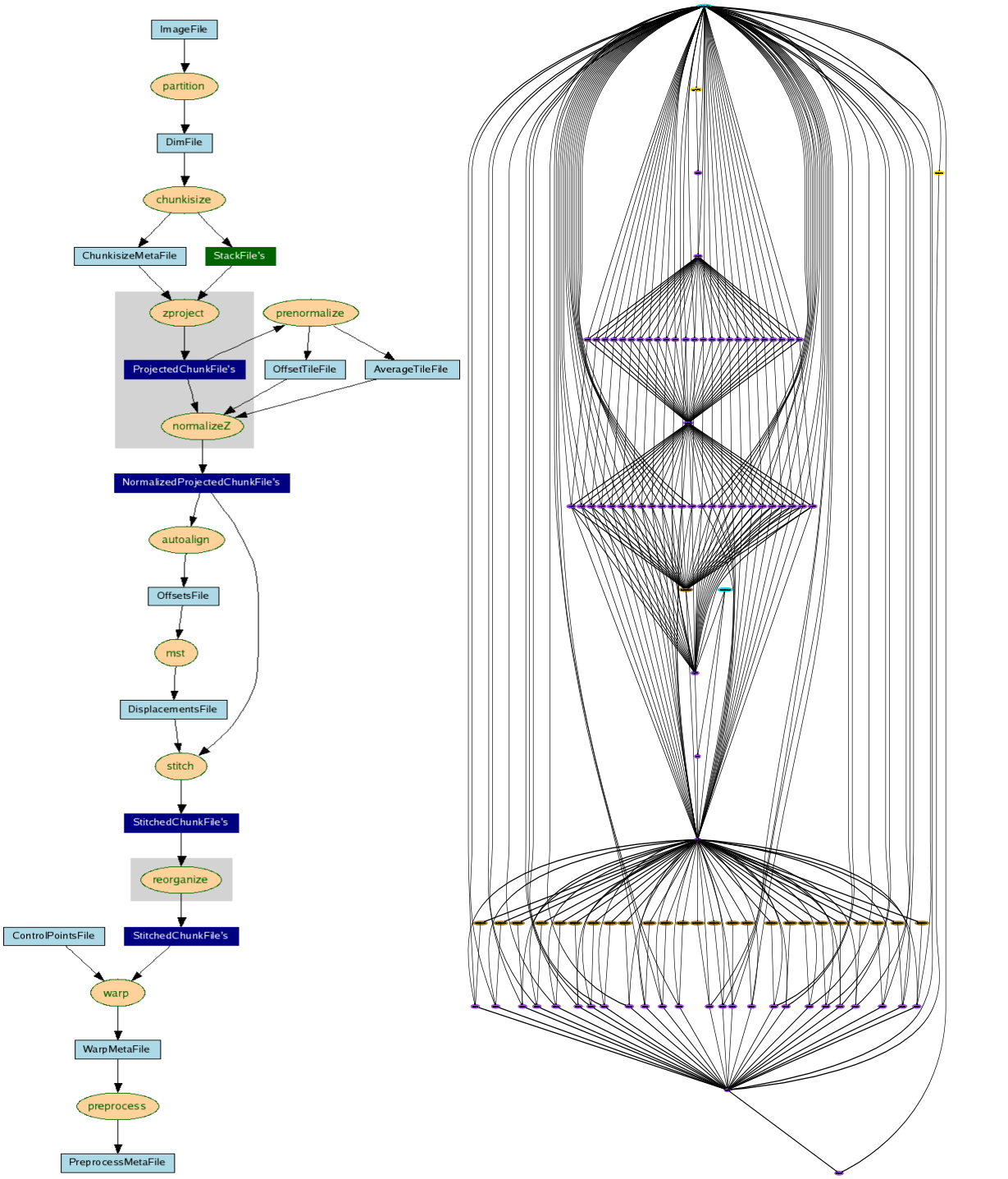



**Fig. 3** Framework architecture

workflow instance for the PIQ workflow as shown in Figure 4(b) contains 116 tasks. Component collections are shown in the workflow instance as long horizontal rows of tasks, signifying that each collection has been unrolled into 36 tasks, where each task represents an operation performed on a single chunk. Thus, `chunksize` parameter influences the structure of the resulting workflow instance.

In our current system, we also support the notion of meta-components or explicit component grouping. A meta-component can be viewed as a combination or clustering of components across multiple levels in a workflow template. Coalescing components into meta-components corresponds to the adjustment of the `task granularity` parameter for performance optimization. During execution, all tasks within a meta-component are scheduled at once and mapped to the same set of resources. Note that the use of meta-components complements the existing job clustering capabilities provided by Grid workflow systems such as Pegasus (Deelman et al, 2004). Horizontal and vertical job clustering capabilities in Pegasus are used to group tasks that are explicitly specified in a workflow instance. However, it is highly desirable to also support the following features:

– *Task parallelism*: Vertical clustering implies that workflow tasks at multiple levels are grouped into clusters so that each group can be scheduled for execution on the same processor. To improve performance, task parallelism across tasks within such groups, and also data streaming from one task to another is needed to avoid disk I/O overheads.




(a) Workflow Template **created** for PIQ

(b) Workflow Instance **generated by Wings for** `chunksize` **parameter =** $2560 \times 2400$ **pixels**

**Fig. 4** PIQ application workflow represented using Wings

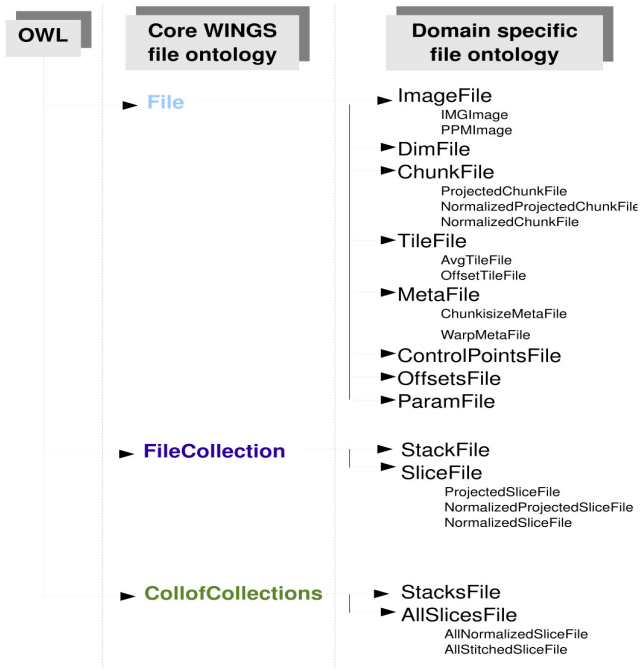



**Fig. 5** Wings data ontology extensions for spatial, multidimensional data

– *Fine-grain control*: Workflows such as PIQ also contain parallel components (e.g., MPI-style jobs) that execute across a number of nodes, but whose processes are not expressed as explicit tasks in the workflow instance. It is the responsibility of the parallel runtime system at a site to manage the processes of such components across the nodes at that site.

Meta-components can provide such task-parallelism and fine-grain control. By grouping successive components into a meta-component in the workflow template, each such component's processes can be concurrently scheduled for execution on the same compute resource. For example, the grey rectangles shown in the workflow instance in Figure 6 indicate that all processes corresponding to all four components (shown as blue ovals) in the workflow have been fused into a meta-component. One task corresponding to each meta-component will be scheduled for execution concurrently. Within each meta-component task, the execution is pipelined, and data is streamed from one component to another during execution.

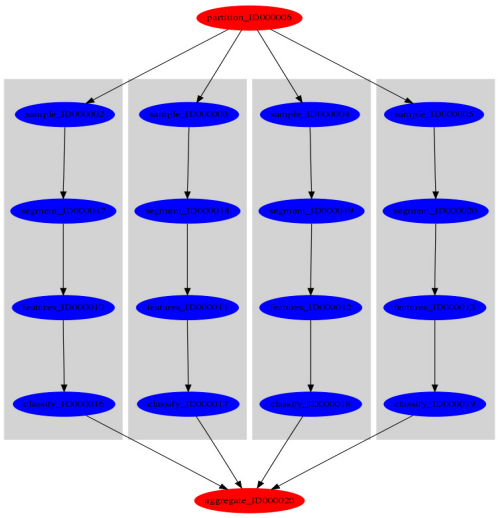


**Fig. 6** Meta-components in the NC workflow instance

## 5.2 Execution Module (EM)

The *Execution Module* (EM) consists of the following subsystems that work in an integrated fashion to execute workflows in a distributed environment and on cluster systems:

**Pegasus Workflow Management System** is used to reliably map and execute application workflows onto diverse computing resources in the Grid (Deelman et al, 2004). Pegasus takes resource-independent workflow descriptions generated by the DM and produces concrete workflow instances with additional directives for efficient data transfers between Grid sites. Portions of workflow instances are mapped onto different sites, where each site could potentially be a heterogeneous, cluster-style computing resource. Pegasus is used to manipulate the `component config` parameter, i.e. the

component transformation catalog can be modified to select an appropriate mapping from components to analysis codes for a given workflow instance. Pegasus also supports runtime job clustering to reduce scheduling overheads. Horizontal clustering groups together tasks at the same level of the workflow (e.g. the unrolled tasks from a component collection), while vertical clustering can group serial tasks from successive components. All tasks within a group are scheduled for execution on the same set of resources. However, Pegasus does not currently support pipelined dataflow execution and data streaming between components. This support is provided by DataCutter (Beynon et al, 2001) as explained later in this section.

**Condor** (Thain et al, 2005) is used to schedule tasks across machines. Pegasus submits tasks corresponding to a portion of a workflow in the form of a DAG to DAGMan meta-scheduler instances running locally at each Grid site. DAGMan resolves the dependencies between jobs and accordingly submits them to Condor, the underlying batch scheduler system.

**DataCutter** (Beynon et al, 2001) is employed for pipelined dataflow execution of portions of a workflow mapped to a Grid site consisting of cluster-style systems. A task mapped and scheduled for execution by Condor on a set of resources may correspond to a meta-component. In that case, the execution of the meta-component is carried out by DataCutter in order to enable the combined use of task- and data-parallelism and data streaming among components of the meta-component. DataCutter uses the *filter-stream* programming model, where component execution is broken down into a set of *filters* that communicate and exchange data via a stream abstraction. For each component, the analysis logic (expressed using high-level languages) like C++, Java, Matlab and Python) is embedded into one or more filters in DataCutter. Each filter executes within a separate thread, allowing for CPU, I/O and communication overlap. Multiple copies of a filter can be created for data parallelism within a component. DataCutter performs all steps necessary to instantiate filters on the target nodes and invokes each filter's analysis logic. A stream denotes a unidirectional data flow from one filter (i.e., the producer) to another (i.e., the consumer). Data exchange among filters on the same node is accomplished via pointer hand-off while message passing is used for filters on different nodes. In our framework, we employ a version of DataCutter that uses MPI for communication to exploit high-speed interconnect technology on clusters that support them.

**ECO compiler**: Within the execution of a single component task, the Empirical Compilation and Optimization (ECO) compiler can be employed to achieve targeted architecture-specific performance optimizations. ECO uses *model-guided empirical optimization* (Chen et al, 2005) to automatically tune the fine-grain computational logic (where applicable) for multiple levels of the memory hierarchy and multi-core processors on a target compute resource. The models and heuristics employed in ECO limit the search space, and the empirical results provide the most accurate information to the compiler to tune performance parameter values.

### 5.3 Trade-off Module (TM)

When large datasets are analyzed using complex operations, an analysis workflow may take too long to execute. In such cases, users may be willing to accept lower quality output for reduced execution time, especially when there are constraints on resource availability. The user may, however, desire that a certain application-level quality of service (QoS) be met. Examples of QoS requirements in image analysis include *Maximize the average confidence in classification of image tiles within t time units* and *Maximize the number of image tiles, for which the confidence in classification exceeds the user-defined threshold, within t units of time* (Kumar et al, 2008b). We have investigated techniques which dynamically order the processing of data elements to speed up application execution while meeting user-defined QoS requirements on the accuracy of analysis. The *Trade-off Module* (TM) draws from and implements the runtime support for these techniques so that accuracy of analysis can be traded for improved performance.

We provide generic support for reordering the data processing operations in our framework by extending Condor's job scheduling component. When a batch of tasks (such as those produced by expansion of component collections in a Wings workflow instance) is submitted to Condor, it uses a default FIFO ordering of task execution. Condor allows users to set the relative priorities of jobs in the submission queue. However, only a limited range (-20 to +20) of priorities are supported by the *condor_prio* utility, while a typical batch could contain tasks corresponding to thousands of data chunks. Moreover, *condor_prio* does not prevent some tasks from being submitted to the queue in the first place. In our framework, we override Condor's default job scheduler by invoking a customized scheduling algorithm that executes as a regular job within Condor's "scheduler universe" and does not require any super-user privileges. The scheduling algorithm implements a priority queue-based job reordering scheme (Kumar et al, 2008b) in a manner that is not tied to any particular application. It uses the semantic representations of data chunks in order to map jobs to the spatial coordinates of the chunks. When the custom scheduler decides which chunk to process next based on its spatial coordinates and other spatial metadata properties, it uses this association to determine the job corresponding to this chunk and

moves it to the top of the queue. The custom scheduler can be employed for any application within the spatial data analysis domain. The priority queue insertion scheme can be manipulated for different QoS requirements such that jobs corresponding to the favorable data chunks are scheduled for execution ahead of other jobs. In this way, the customized scheduler helps exercise control over the `processing order` parameter. When there are no QoS requirements associated with the user query, our framework reverts to the default job scheduler within Condor.

### 5.4 Framework Application

Our current implementation of the proposed framework supports the performance optimization requirements associated chunk-based image/spatial data analysis applications. However, the framework can be employed in other data analysis domains. The customized Condor scheduling module facilitates a mechanism for trading analysis accuracy for performance with user-defined quality of service requirements. The current implementation instance of our framework provides support for users to specify and express the values of various performance parameters to improve performance of the workflow. When dealing with application-specific performance parameters at a fine-grain computation levels, the model-guided optimization techniques in ECO can assist the user in determining the optimal parameter values. We view this as a first step towards an implementation that can automatically map user queries to appropriate parameter value settings. We target application scenarios where a given workflow is employed to process a large number of data elements (or a large dataset that can be partitioned into a set of data elements). In such cases, a subset of those data elements could be used to search for suitable parameter values (by applying sampling techniques to the parameter space) during workflow execution and subsequently refining the choice of parameter values based on feedback obtained from previous runs. Statistical modelling techniques similar to those used in the Network Weather Service (Wolski et al, 1999) can be used to predict performance and quality of future runs based on information gathered in previous runs.

## 6 Experimental Evaluation

In this section, we present an experimental evaluation of our proposed framework using the two real-world applications, PIQ and NB, described in section 3. Our evaluation was carried out across two heterogeneous Linux clusters hosted at different locations at the Ohio State University. The first one (referred to here as **RII-MEMORY**) consists of 64 dual-processor nodes equipped with 2.4 GHz AMD Opteron processors and 8 GB of memory, interconnected by a Gigabit Ethernet network. The storage system consists of 2x250GB SATA disks installed locally on each compute node, joined into a 437GB RAID0 volume. The second cluster, (referred to here as **RII-COMPUTE**), is a 32-node cluster consisting of faster dual-processor 3.6 GHz Intel Xeon nodes each with 2 GB of memory and only 10 GB of local disk space. This cluster is equipped with both an InfiniBand interconnect as well as a Gigabit Ethernet network. The RII-MEMORY and RII-COMPUTE clusters are connected by a 10-Gigabit wide-area network connection – each node is connected to the network via a Gigabit card; we observed about 8 Gigabits/sec application level aggregate bandwidth between the two clusters. The head-node of the RII-MEMORY cluster also served as the master node of a Condor pool that spanned all nodes across both clusters. A Condor scheduler instance running on the head-node functioned both as an opportunistic scheduler (for "vanilla universe" jobs) and a dedicated scheduler (for parallel jobs). The "scheduler universe" jobs in Condor, including our customized scheduling algorithm, when applicable, run on the master node. All other nodes of the Condor pool were configured as worker nodes that wait for jobs from the master. DataCutter instances executing on the RII-COMPUTE cluster use the MVAPICH flavor[4] of MPI for communication to exploit the InfiniBand interconnect. Our evaluation of the ECO compiler's automated parameter tuning was carried out independently on a Linux cluster hosted at the University of Southern California. In this cluster (referred to as **HPCC**) we used a set of 3.2 GHz dual Intel Xeon nodes each with 2 GB of memory for our evaluation.

In the following experiments, we evaluate the performance impact of a set of parameter choices on workflow execution time. First, a set of *accuracy-preserving* parameters are explored, as we would initially like to tune the performance without modifying the results of the computation. We subsequently investigate the *accuracy-trading* parameters.

### 6.1 Accuracy-preserving Parameters

We used our framework to evaluate the effects of three different accuracy-preserving parameters on the execution time for the PIQ workflow – (i) the `chunksize`, a component-level parameter, (ii) the task granularity, a workflow-level parameter, and (iii) `numActiveChunks`, a component-level parameter that is specific to the *warp* component. Two additional accuracy-preserving parameters – `component config` and `component placement` were set to their optimal values based on our prior experience with and evaluations of the PIQ workflow. In an earlier work (Kumar et al, 2008a),

[4] http://mvapich.cse.ohio-state.edu

our evaluations revealed that certain components of the PIQ workflow such as *normalize* and *autoalign* were much faster when mapped for execution onto cluster machines equipped with fast processors and high-speed interconnects. Based on these evaluations and our knowledge of the data exchange between components in the workflow, we determined an optimal `component placement` strategy which minimized overall computation time as well as the volume of data exchange between nodes. In this strategy, components *zproject, prenormalize, stitch, reorganize,warp* and the *preprocess* meta-component execute on the RII-MEMORY cluster nodes, while the *normalize, autoalign* and *mst* components are mapped to the faster processors of the RII-COMPUTE cluster. This `component placement` strategy allows for maximum overlap between computation and data communication between sites for the PIQ workflow. In another earlier work (Kumar et al, 2006), we designed multiple algorithmic variants for the *warp* and *preprocess* components of the PIQ workflow. Our evaluations helped us select the most performance-effective variant for these components depending upon the resources they are mapped to. In our experiments, we fixed the values of these latter two parameters and then set out to tune or determine optimal values for the remaining parameters. Our strategy was to treat different parameters independently, selecting a default value for one parameter while exploring the other. The decision as to which parameter to explore first is one that can either be made by an application developer, or can be evaluated systematically by a set of measurements, such as the sensitivity analysis found in (Chung and Hollingsworth, 2004).

**Effects of** `chunksize`: For these experiments, we used a 5 GB image with 8 focal planes, with each plane at a resolution $15,360 \times 14,400$ pixels. The `chunksize` parameter determines the unrolling factor for component collections in the template shown in Figure 4(a). Varying the `chunksize` parameter value affects the structure of the resulting workflow instance and the number of workflow tasks to be scheduled, as shown in table 1.

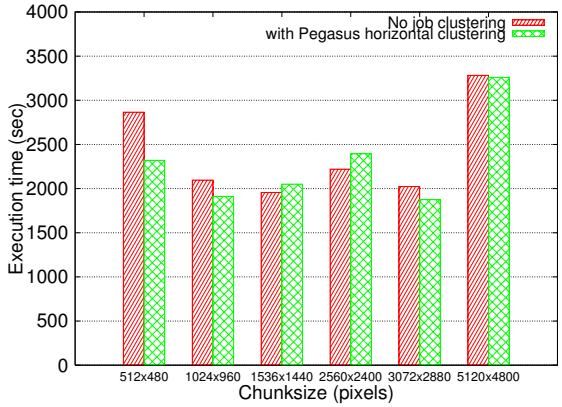
**Table 1** Number of tasks in PIQ workflow instance for a given `chunksize` parameter value

| **chunksize** (pixels) | # of chunks in a plane | **# of tasks in workflow** | |
|---|---|---|---|
| | | (no clustering) | (horizontal clustering::32) |
| $512 \times 480$ | *900* | *2708* | *95* |
| $1024 \times 960$ | *225* | *683* | *32* |
| $1536 \times 1440$ | *100* | *308* | *20* |
| $2560 \times 2400$ | *36* | *116* | *14* |
| $3072 \times 2880$ | *25* | *83* | *9* |
| $5120 \times 4800$ | *9* | *35* | *9* |

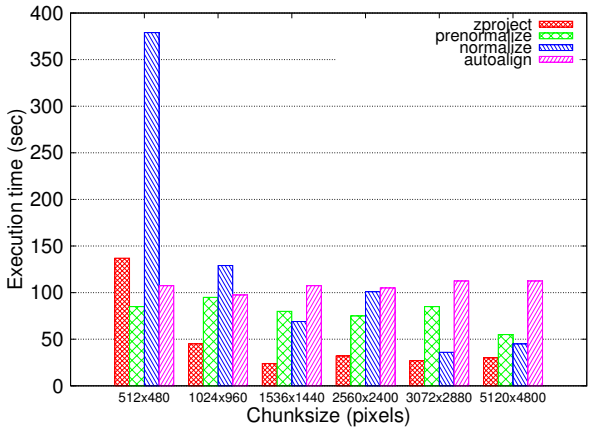
The disparity among the number of tasks in the resulting workflow instances will increase as the images grow in size, because larger images can accomodate more combinations of the chunk dimensions. If workflow templates have a larger number of component collections, then the number of tasks in the resulting workflow instance will vary more greatly with `chunksize` value. As job submission and job scheduling overheads are sizeable contributions to the overall execution time, one possible optimization technique is to employ horizontal clustering of tasks from every component collection. The table also shows the number of tasks in the resulting PIQ workflow instance for each `chunksize` value when horiontal clustering by Pegasus is used to group tasks from component collections into bundles of 32 tasks each. Ideally, with horizontal clustering, one should expect diminishing job scheduling overheads and lesser disparity in the execution times observed at different `chunksize` values.

Figure 7(a) shows the overall execution time for the PIQ workflow instance when choosing different `chunksize` parameter values, both with and without using horizontal job clustering (We used 16 RII-MEMORY nodes and 8 RII-COMPUTE nodes for these experiments). This time is inclusive of the time to stage data in and out of each site during execution, the actual execution times for each component, and the job submission and scheduling overheads. We observe that: (1) At both extremes of the `chunksize` parameter value range, the execution times are high. The job submission and scheduling overheads for the large number of tasks dominate at smallest `chunksize` value. At the largest `chunksize` value, the number of resulting chunks in an image becomes lesser than the number of worker nodes available for execution. Since each task must process at least one chunk, the under-utilization of resources leads to higher execution times. (2) The intermediate values for `chunksize` yield more or less similar performance, except for an unexpected spike at `chunksize` = $2560 \times 2400$. On further investigation of the individual component execution times, we observed that the algorithmic variant used for the *warp* component – which accounts for nearly 50% of the overall execution time of PIQ – performed poorly at this `chunksize` value. Figures 7(b) and 7(c) show how `chunksize` value affects performance at the level of each component. (3) Horizontal job clustering, as expected, lowers execution time at smaller `chunksize` values. At larger `chunksize` values, the lack of a large job pool to begin with renders job clustering ineffective.

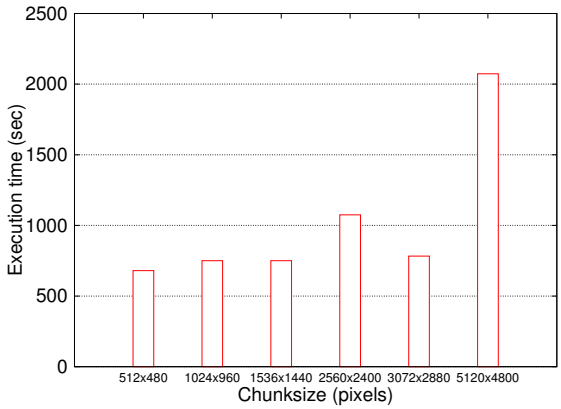
As mentioned earlier, `chunksize` value is expected to have a greater impact on the performance for larger images because of the larger range within which the parameter values can be varied. Figure 7(d) presents our evaluation results obtained for a 17 GB image (with 3 focal planes, each plane having $36864 \times 48000$ pixels). We used 32 RII-MEMORY nodes and 8 RII-COMPUTE nodes for these experiments.



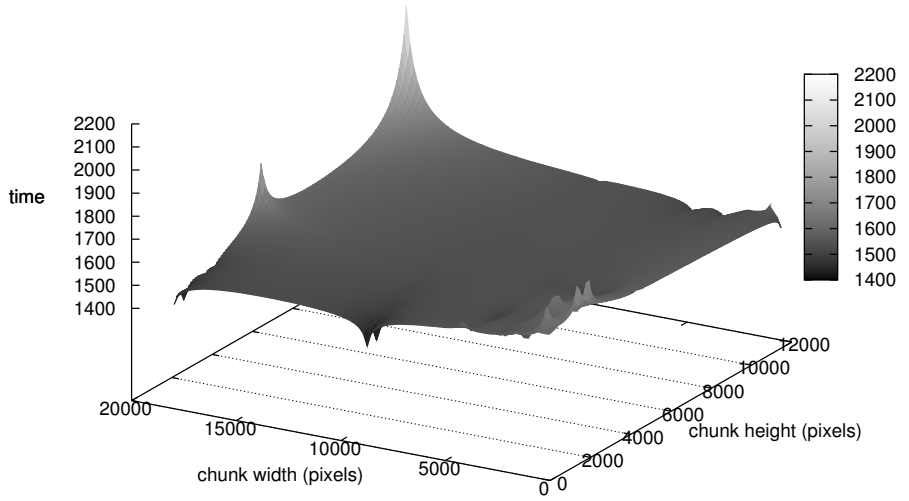

(a) **Total PIQ workflow execution time for varying `chunksize` parameter values**




(b) **Execution time for individual components of PIQ workflow**




(c) **Execution time for the *warp* component**




(d) **Total PIQ workflow execution time for varying `chunksize` parameter values (large image)**

**Fig. 7** Effect of varying `chunksize` parameter on PIQ workflow execution

The `chunksize` parameter has been shown using separate axes for the chunk width and the chunk height for better viewing of the results. The surface was plotted based on results obtained using 75 different values of the `chunksize` parameter. Again, we observe poor performance at the extreme low and high values of `chunksize` for the same reasons outlined earlier. Here, we also observed that `chunksize` values that correspond to long horizontal stripe chunks yielded better performance for the PIQ workflow and this class of data. This tells us that most analysis operations in the PIQ workflow favor horizontal striped chunks. In future endeavors, our framework will seek to determine the best values of the `chunksize` parameter by efficient navigation of the surface based on sampling techniques and trends generated from training data.

**Effects of Task Granularity**: In these experiments, we coalesced components of the PIQ workflow into meta-components to produce a workflow template with a coarser task granularity. Figure 8 illustrates an alternative workflow template for the PIQ application generated using Wings, that groups components into meta-components. Here, the *zproject* and *prenormalize* steps from the original template are fused to form *metacomp1*; *normalize, autoalign, mst* are collectively *metacomp2* while *stitch, reorganize, warp* form *metacomp3*. *Preprocess* is the fourth meta-component in the workflow template and includes the *thresholding*, *tessellation* and *prefix sum generation* components. By using this representation, our framework further reduces the number of tasks in a workflow instance. When component collections are embedded within a meta-component, they are not explicitly unrolled at the time of workflow instance generation. Instead, they are implicitly unrolled at runtime within a corresponding DataCutter instance. That is, the `chunksize` input parameter to a meta-component is propagated to the DataCutter filters set up for each component within that meta-component. Based on the value of `chunksize`, DataCutter will create multiple transparent copies of filters that handle the processing of tasks within a component collection. Each such filter copy or instance will operate on a single chunk at a time. We also manipulated the execution strategy within the *preprocess* meta-component to support task-parallelism (i.e., pipelined execution of data chunks) in order to avoid disk I/O overheads for the meta-component.

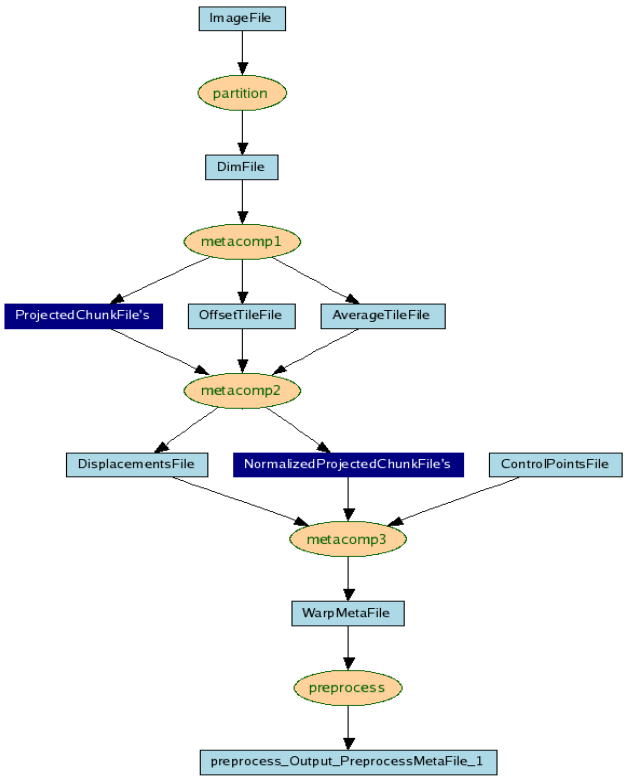



**Fig. 8** Wings workflow template for the PIQ application with meta-components

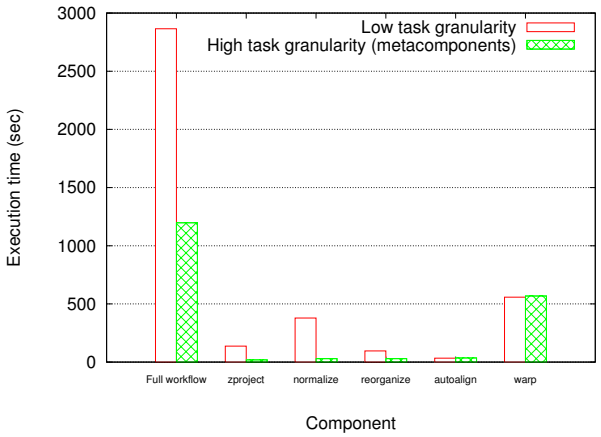



**Fig. 9** Execution time with different task granularity (5GB image, 40 nodes, `chunksize`=$512 \times 480$)

Figure 9 shows that the overall execution time for the PIQ workflow improves by over 50% when the task granularity strategy employed includes meta-components. The figure also shows the changes in execution times for individual components of the workflow. For parallel components like *autoalign* and *warp*, there is no difference in performance because the actual scheduling of processes within a component is carried out in the same way regardless of whether meta-components are used or not. However, component collections like *zproject*, *normalize* and *reorganize* benefit from the high-granularity execution. This difference is attributed to the two contrasting styles of execution that our framework can offer by integrating Pegasus and DataCutter. Pegasus supports batch-style execution where the execution of each task is treated independently. If startup costs

(e.g. MATLAB invocation or JVM startup) are involved in the execution of a task within a component collection, such overheads are incurred for each and every task in the collection. In contrast, DataCutter functions like a services-based system, where filters set up on each worker node perform the desired startup operations only once and process multiple data instances. The input data to a collection can thus be streamed through these filters. Depending on the nature of the tasks within a collection, our framework can use explicit unrolling and execution via Pegasus, or implicit unrolling within a meta-component and execution via DataCutter.

**Effects of parameter tuning for individual components**: In these experiments, we take a closer look at accuracy-preserving parameters that are specific to individual components in the workflow. Our goal here is to show how the autotuning of such parameters based on model-guided optimization techniques within the ECO compiler can reduce the execution times of the individual components, and hence, of the overall workflow. In the PIQ workflow, one of the algorithmic variants of the *warp* component, known as the On-Demand Mapper (ODM), was shown to provide best performance among all variants, as long as the amount of physical memory available on the cluster nodes was not a limiting factor in the execution (Kumar et al, 2006). The benefits of ODM derived from the fact that ODM, unlike the other variants, seeks to maintain large amounts of data in memory during execution. We identified a parameter called `numActiveChunks` that affects the execution time of the ODM variant by limiting the number of 'active' data chunks that ODM can maintain in memory on a node at any given time. The larger the number of active chunks that can be maintained in memory, the greater the potential for data reuse when the warping transformations are computed for the input data. The potential reduction in chunk reads and writes via data reuse leads to lesser execution time. The maximum value of `numActiveChunks` parameter depends on the size of the chunks and the availabe physical memory on a node. However, the optimal `numActiveChunks` value may depend on other factors such as the potential for data reuse in the ODM variant for a given memory hierarchy which need to be modeled appropriately.

We introduced a novel technique for modeling that discovers the behavior of such application-level parameters through the use of functional models and statistical methods in (Nelson, 2009). Our technique requires the component developer to identify a set of integer-valued parameters, and also specify the expected range for the optimal parameter value. The models are derived from sampling the execution time for a small number of parameter values, and evaluating how well these sample points match a functional model. To derive the functional model of the sample data, we use the curve fitting toolbox in MATLAB to find the function that best fits the data, and also to compute the $R^2$ value in order to quantify the accuracy of our model. Ultimately, we discovered that the double-exponential function ($y = a*e^{b*x} + c*e^{d*x}$) was the best overall match for the application-level parameters in our set of experiments. Using this double-exponential function model, the algorithm can select the parameter value for which the function is minimized, and can dynamically identify the neighborhood of the best result.

Our experiments were conducted on nodes of the HPCC cluster. Here, we evaluated only the *warp* component of the PIQ workflow. Specifically, we evaluated only the ODM variant of the component in order to determine the optimal value of the `numActiveChunks` parameter. Figure 10 shows results of our evaluation from warping a single slice of the image data described earlier in this section. We try to model the behavior of the `numActiveChunks` parameter for the input data using a statistical analysis approach. In this figure, the curve represents the double-exponential function obtained using five sample data points, while the remaining points obtained from the exhaustive search within the range of permissible parameter values are also shown. The area between the dotted lines represents the range of parameter values that our model predicts will have performance within 2% of the best performance. In this case, we predicted the optimal value of 74 for the `numActiveChunks` through the function model, which ended up being within 1.25% of the real best performance that one could obtain using the exhaustive search.

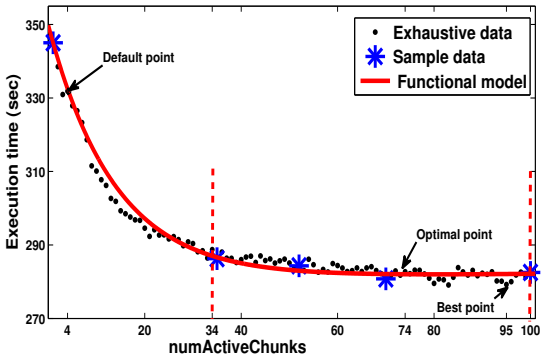



**Fig. 10** Autotuning to obtain optimal value of `numActiveChunks` parameter for the *warp* component; 8 processors

We validate our model by comparing against performance results from an exhaustive set of experiments across the entire permissible parameter range. Overall, we have achieved speedups up to 1.38X as compared to the worst-case performance from an exhaustive search while being within 0.57% and 1.72% of the best performance from the exhaustive search. We examined only 5% of the parameter search space by computing five point samples. Our experimental results show

speedups of up to 1.30X as compared to the user-specified default parameter value. While our speedup is small when placed in the context of the overall PIQ workflow, our selection of parameter values based on a statistical analysis approach derives improved performance as compared to current user-specified values.

### 6.2 Accuracy-trading Parameters

These experiments demonstrate performance gains obtained when one or more accuracy-trading parameters were modified in response to queries with QoS requirements. The optimizations in these experiments are derived from the custom scheduling approach described in section 5.3 that makes data chunk reordering decisions dynamically as and when jobs are completed. In all our experiments, we observed that the overall execution time using our custom scheduling algorithm within Condor is only marginally higher than that obtained from using the default scheduler, thereby showing that our approach introduces only negligible overheads. These experiments were carried out on the RII-MEMORY cluster [5] with Condor's master node running our customized scheduler. We carried out evaluations using multiple images (ranging in size from 12 GB to 21 GB) that are characterized by differences in their data (feature) content. We target user queries with two kinds of QoS requirements: **Requirement 1**: *Maximize average confidence across all chunks in an image within time t*; **Requirement 2**: *Given a confidence in classification threshold, maximize the number of finalized chunks for an image within time t*. These requirements can be met by tuning combinations of one or more relevant accuracy-trading parameters.

**Tuning only the `processing order` parameter for requirement 1**: This is useful when users wish to maximize average confidence across an image while processing all chunks at the highest resolution, i.e. trading quality of result for the overall image, but not the quality of the result for individual chunks. In such cases, the `processing order` parameter can be tuned such that the Condor customized scheduler prioritizes chunks that are likely to yield output with higher $(confidence/time)$ value at the maximum resolution.

Figure 11 shows that a tuned parameter helps achieve a higher average confidence at maximum resolution across all chunks. For example, after 800 seconds, the tuned parameter execution achieves an average confidence of 0.34 which is greater than the 0.28 value achieved when no parameter tuning is employed. This is because our customized scheduler tunes the `processing order` parameter to reorder chunk execution in a manner that favors jobs corresponding to chunks that yield higher confidence.

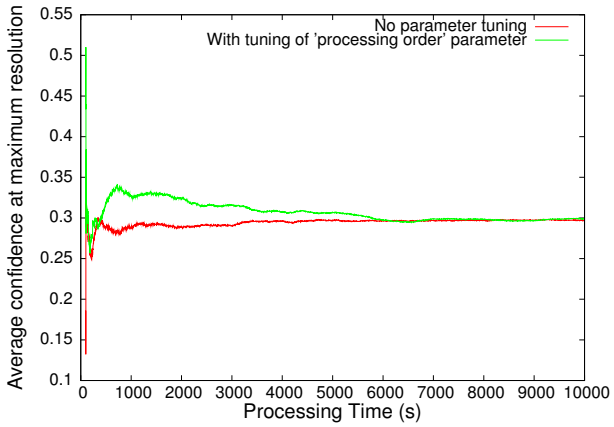



**Fig. 11** Quality Improvement by tuning only the `processing order` parameter

**Tuning both the `resolution` and `processing order` parameters for requirement 1**: This is useful when individual chunks can be processed at lower resolutions so long as the resulting confidence exceeds a user-specified threshold. Figure 12 shows how parameter tuning helps achieve higher average confidence at all points during the execution. Each chunk is iteratively processed until only that target resolution at which the confidence exceeds a threshold (set to 0.25 here). Our custom scheduler prioritizes chunks that are likely to yield output with higher $(\frac{confidence}{time})$ value at lower resolutions.

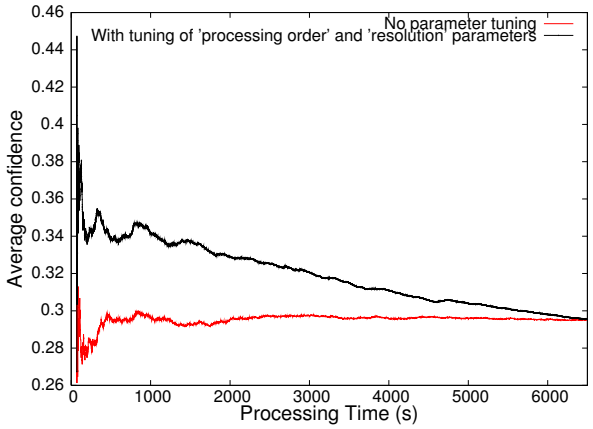



**Fig. 12** Quality improvement by tuning the `processing order` and `resolution` parameters

Results obtained for other test images exhibited similar improvement trends and also showed that our customized scheduling extensions to Condor scale well with data size.

[5] Our goal here is only to demonstrate our framework's ability to exploit performance-quality trade-offs, and not adaptability to heterogeneous sets of resources. We note that this experiment could also be carried out on the same testbed used for the PIQ application.

**Tuning both the `resolution` and `processing order` parameters for requirement 2**: Here, the customized scheduler prioritizes jobs corresponding to chunks that are likely to get finalized at lower resolutions. Figure 13 shows how parameter tuning in our framework yields an increased number of finalized chunks at every point in time during the execution. The improvement for this particular case appears very slight because the confidence in classification threshold was set relatively high as compared to the average confidence values generated by the *classify* component, and this gave our custom scheduler lesser optimization opportunities via reordering.

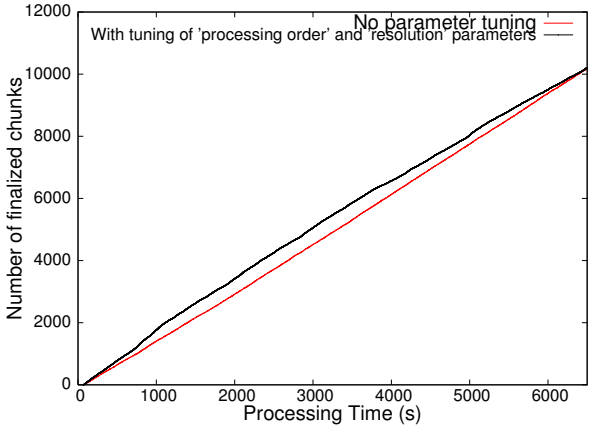



**Fig. 13** Improvement in number of finalized tiles by tuning parameters

**Scalability**: In this set of experiments (carried out as part of requirement 2), we scaled the number of worker nodes in our Condor pool from 16 to 48. Our goal here was to determine if our custom scheduler could function efficiently when more worker nodes are added to the system. Figure 14 shows how the time taken to process a certain number of chunks in an image halves as we double the number of workers. Hence, the scheduler performance scales linearly when an increasing number of resources need to be managed.

## 7 Summary and Conclusions

Many scientific workflow applications are data and/or compute-intensive. The performance of such applications can be improved by adjusting component-level parameters as well as by applying workflow-level optimizations. In some application scenarios, performance gains can be obtained by sacrificing accuracy of the analysis, so long as some minimum quality requirements are met. Our work has introduced a framework that integrates a suite of workflow description, mapping and scheduling, and distributed data processing subsystems in order to provide support for parameter-

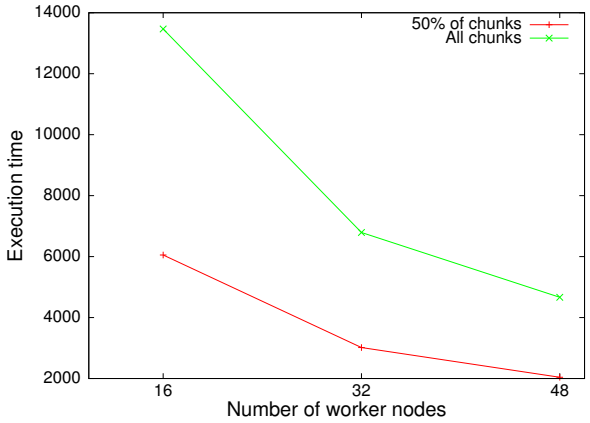



**Fig. 14** Scalability with number of worker nodes

based performance optimizations along multiple dimensions of the parameter space.

Our current implementation of the proposed framework provides support for users to manually express the values of the various performance parameters in order to improve performance. We have customized the job scheduling module of the Condor system to enable trade-offs between accuracy and performance in our target applications. The experimental evaluation of the proposed framework shows that adjustments of accuracy-preserving and accuracy-trading parameters lead to performance gains in two real applications. The framework also achieves improved responses to queries involving quality of service requirements. As a future work, we will incorporate techniques to search the parameter space in a more automated manner. We target application scenarios in which a large number of data elements or a large dataset that can be partitioned into a number of chunks are processed in a workflow. In such cases, a subset of the data elements could be used to search for suitable parameter values during workflow execution and subsequently refining the choice of parameter values based on feedback obtained from previous runs.

## 8 Acknowledgements

This research was supported in part by the National Science Foundation under Grants #CNS-0403342, #CNS-0426241, #CSR-0509517, #CSR-0615412, #ANI-0330612, #CCF-034-2615, #CNS-0203846, #ACI-0130437, #CNS-0615155, #C-NS-0406386, and Ohio Board of Regents BRTTC #BRTT02-0003 and AGMT TECH-04049 and NIH grants #R24 HL085-343, #R01 LM009239, and #79077CBS10.

## References

Acher M, Collet P, Lahire P (2008) Issues in Managing Variability of Medical Imaging Grid Services. In: MICCAI-Grid Workshop( MICCAI-Grid)

Beynon MD, Kurc T, Catalyurek U, Chang C, Sussman A, Saltz J (2001) Distributed processing of very large datasets with DataCutter. Parallel Computing 27(11):1457–1478

Brandic I, Pllana S, Benkner S (2008) Specification, planning, and execution of QoS-aware Grid workflows within the Amadeus environment. Concurrency and Computation: Practice and Experience 20(4):331–345

Chang F, Karamcheti V (2000) Automatic configuration and run-time adaptation of distributed applications. In: High Performance Distributed Computing, pp 11–20

Chen C, Chame J, Hall MW (2005) Combining models and guided empirical search to optimize for multiple levels of the memory hierarchy. In: International Symposium on Code Generation and Optimization

Chiu D, Deshpande S, Agrawal G, Li R (2008) Cost and accuracy sensitive dynamic workflow composition over Grid environments. 9th IEEE/ACM International Conference on Grid Computing pp 9–16

Chow SK, Hakozaki H, Price DL, MacLean NAB, Deerinck TJ, Bouwer JC, Martone ME, Peltier ST, Ellisman MH (2006) Automated microscopy system for mosaic acquisition and processing. Journal of Microscopy 222(2):76–84

Chung IH, Hollingsworth J (2006) A case study using automatic performance tuning for large-scale scientific programs. 15th IEEE International Symposium on High Performance Distributed Computing pp 45–56

Chung IH, Hollingsworth JK (2004) Using information from prior runs to improve automated tuning systems. In: SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing, IEEE Computer Society, Washington, DC, USA, p 30

Cortellessa V, Marinelli F, Potena P (2006) Automated selection of software components based on cost/reliability tradeoff. In: Software Architecture, Third European Workshop, EWSA 2006, Springer, Lecture Notes in Computer Science, vol 4344

Deelman E, Blythe J, Gil Y, Kesselman C, Mehta G, Patil S, Su MH, Vahi K, Livny M (2004) Pegasus: Mapping scientific workflows onto the Grid. Lecture Notes in Computer Science: Grid Computing pp 11–20

Gil Y, Ratnakar V, Deelman E, Mehta G, Kim J (2007) Wings for Pegasus: Creating large-scale scientific applications using semantic representations of computational workflows. In: Proceedings of the 19th Annual Conference on Innovative Applications of Artificial Intelligence (IAAI)

Glatard T, Montagnat J, Pennec X (2006) Efficient services composition for Grid-enabled data-intensive applications. In: Proceedings of the IEEE International Symposium on High Performance Distributed Computing (HPDC'06), Paris, France, June 19

Kong J, Sertel O, Shimada H, Boyer K, Saltz J, Gurcan M (2007) Computer-aided grading of neuroblastic differentiation: Multi-resolution and multi-classifier approach. IEEE International Conference on Image Processing, ICIP 2007 5:525–528

Kumar V, Rutt B, Kurc T, Catalyurek U, Pan T, Chow S, Lamont S, Martone M, Saltz J (2008a) Large-scale biomedical image analysis in Grid environments. IEEE Transactions on Information Technology in Biomedicine 12(2):154–161

Kumar VS, Rutt B, Kurc T, Catalyurek U, Saltz J, Chow S, Lamont S, Martone M (2006) Large image correction and warping in a cluster environment. In: SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing, ACM, New York, NY, USA, p 79

Kumar VS, Narayanan S, Kurç TM, Kong J, Gurcan MN, Saltz JH (2008b) Analysis and semantic querying in large biomedical image datasets. IEEE Computer 41(4):52–59

Lera I, Juiz C, Puigjaner R (2006) Performance-related ontologies and semantic web applications for on-line performance assessment intelligent systems. Sci Comput Program 61(1):27–37

Ludäscher B, Altintas I, Berkley C, Higgins D, Jaeger E, Jones M, Lee EA, Tao J, Zhao Y (2006) Scientific workflow management and the Kepler system: Research articles. Concurr Comput: Pract Exper 18(10):1039–1065

Nelson YL (2009) Model-guided performance tuning for application-level parameters. PhD dissertation, University of Southern California

Norris B, Ray J, Armstrong R, Mcinnes LC, Shende S (2004) Computational quality of service for scientific components. In: Proceedings of the International Symposium on Component-based Software Engineering (CBSE7), Springer, pp 264–271

Oinn T, Addis M, Ferris J, Marvin D, Senger M, Greenwood M, Carver T, Glover K, Pocock MR, Wipat A, Li P (2004) Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics 20(17):3045–3054

Thain D, Tannenbaum T, Livny M (2005) Distributed computing in practice: the Condor experience: Research articles. Concurr Comput : Pract Exper 17(2-4):323–356

Truong HL, Dustdar S, Fahringer T (2007) Performance metrics and ontologies for grid workflows. Future Generation Computer Systems 23(6):760 – 772

Wolski R, Spring N, Hayes J (1999) The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. Journal of Future Generation Computing Systems 15:757–768

Zhou J, Cooper K, Yen IL (2004) A rule-based component customization technique for QoS properties. Eighth IEEE International Symposium on High Assurance Systems Engineering pp 302–303