Published in final edited form as:
Proc Int Symp High Perform Distrib Comput. 2009 ; : 177–186. doi:10.1145/1551609.1551638.

An Integrated Framework for Parameter-based Optimization of Scientific Workflows

Vijay S. Kumar and **P. Sadayappan**
Department of Computer Science and Engineering Ohio State University Columbus, Ohio 43210 kumarvi@cse.ohio-state.edu, saday@cse.ohio-state.edu

Gaurang Mehta, **Karan Vahi**, and **Ewa Deelman**
Advanced Systems Division Information Sciences Institute University of Southern California Marina del Rey, California 90292 gmehta@isi.edu, vahi@isi.edu, deelman@isi.edu

Varun Ratnakar, **Jihie Kim**, and **Yolanda Gil**
Intelligent Systems Division Information Sciences Institute University of Southern California Marina del Rey, California 90292 varunr@isi.edu, jihie@isi.edu, gil@isi.edu

Mary Hall
School of Computing University of Utah Salt Lake City, Utah 84112 mhall@cs.utah.edu

Tahsin Kurc and **Joel Saltz**
Center for Comprehensive Informatics Emory University Atlanta, Georgia 30307 tkurc@emory.edu, jhsaltz@emory.edu

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 multi-dimensional parameter space. While some performance parameters such as grouping of workflow components and their mapping to machines do not a ect the accuracy of the output, 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 dimensions of the parameter space. Using two real-world applications in the spatial data analysis domain, we present an experimental evaluation of the proposed framework.

Keywords

scientific workflow; performance parameters; semantic representations; Grid; application QoS

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 size1. 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 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 analysis of large 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 cluster system or a potentially heterogeneous and dynamic collection of machines. Second, for each portion of a workflow mapped to such cluster systems, they enable 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 value 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 multi-dimensional 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 datasets. Another example is the version of the algorithm employed by a component to process the data. Systems should provide support to improve performance through manipulation of such parameters along multiple dimensions of the parameter space.

In this work, we classify workflow-level and component-level performance parameters into two categories: (i) *Quality-preserving* parameters (such as data chunk size) can affect the performance of an operation without affecting the quality of the analysis output, and (ii) *Quality-trading* parameters can trade the quality of the output for performance gains, and vice-versa. An example of a quality-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 performance-quality trade-offs, users may supplement their queries with application-level quality-of-service (QoS) requirements that place constraints on the quality of output [14]. 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 workflow system that supports application execution in a distributed environment and enables performance optimization via

[1]DMetrix array microscopes can scan a slide at 20×+ resolution in less than a minute. The Large Synoptic Survey Telescope will be able to capture a 3.5 Gigapixel image every 6 seconds, when it is activated.

manipulation of quality-preserving and/or quality-trading parameters. The proposed system integrates four subsystems: WINGS [10] to facilitate high-level, semantic descriptions of workflows, and Pegasus [9], Condor [19], and DataCutter [1] to support scalable workflow execution across multiple institutions and on distributed clusters within an institution. In our system, application developers and end-users can provide high-level, semantic descriptions of application structure and data characteristics. As our initial focus is on optimizing spatial data analysis applications, we have developed extensions to the core ontologies in WINGS to describe spatial datasets and 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. We have extended Condor's default job-scheduling mechanism to support performance optimizations stemming from performance-quality trade-offs. We show how parameter-based optimizations can be supported for real-world biomedical image analysis workflows using two cluster systems located at two different departments at the Ohio State University.

2. RELATED WORK

Application configuration for performance improvements in scientific workflows has been examined to varying degrees within different application domains. Grid workflow management systems like Taverna [18], Kepler [16] and Pegasus [9] 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. [11] 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 and additionally addresses 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 [3, 17]. Active Harmony [6, 7] 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 quality-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 [2] and component-based systems [8]. 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 [22, 4]. We support trade-offs based on quality of data output from the application components and integrate such parameter tuning with a standard job scheduling 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

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

system-comprehensible manner. In [4], 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 Askalon [20]. Lera et al. [15] 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. [22] 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 multidimensional data. 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.

Application 1: Pixel Intensity Quantification (PIQ)

Figure 1 shows a data analysis pipeline [5] (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*.

Application 2: Neuroblastoma Classification (NB)

Figure 2 shows a multi-resolution based tumor prognosis pipeline [12] (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 [14], 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.

These applications are different in terms of their their workflow structures and the operations they perform on the data: The NB 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 operations that are not pleasingly parallelizable and operate on entire images (with multiple focal planes), hence requiring parallel algorithms for efficient processing of out-of-core data.

4. PERFORMANCE OPTIMIZATIONS

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

4.1 Quality-preserving parameters

Chunking Strategy—Individual data elements (e.g. images) in a spatial dataset may be larger than the physical memory space on a compute resource. Relying on virtual memory alone is likely to yield poor performance. In general, the processing of large, out-of-core spatial data is supported by partitioning it into a set of *data chunks*, each of which can fit in memory, and by modifying the analysis operations 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 and data exchange between processors. In the simplest case, we can have a uniform chunking strategy, i.e. every chunk has the same shape and size. For 2-D data, this parameter is represented by a pair [*W*, *H*], where *W* and *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 inter-processor communication for that chunk will likely be, albeit the number of chunks will be smaller. The chunk-size 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 memory hierarchy and hardware architecture present on a compute resource, the chunksize can affect 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, algorithmic variants are chosen based on the *type* of the input data to 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 [13], we developed three parallel-algorithmic variants for the *warp* component of the PIQ workflow. We observed that depending on the available resources – slow/fast processor/network/disk – each variant could outperform the other. No single variant performed best under all resource conditions.

Task Granularity and Execution Strategy—A work-flow 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 [19]. 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 *threshold* and *tessellate* in the PIQ workflow) can be pipelined such that, when a component $C1$ is processing a chunk i, then the downstream component $C2$ 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 can be avoided.

4.2 Quality-trading parameters

Data resolution—Spatial data has an associated notion of quality. The resolution parameter for a data chunk takes values from 1 to n, where n represents the chunk at its highest quality. In a multi-resolution processing approach, a chunk can be processed at multiple resolutions to produce output of varying quality. Execution time increases with resolution because higher resolutions contain more data to be processed. In general, chunks need to be processed only at the lowest *target* resolution that can yield a result of adequate quality.

Processing order—The processing order parameter refers to the order in which data chunks are operated upon by the components in a workflow. Our previous work [14] with the NB workflow showed how out-of-order processing of chunks (selecting a subset of "favorable" chunks ahead of other chunks) in an image could yield improved responses (by a factor of 40%) to user queries with 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, given the high-level description of the workflow, datasets to be analyzed, a target distributed execution environment. Lastly, the *trade-off module*, implements runtime mechanisms to enable performance-quality trade-offs based on user-specified QoS requirements and constraints. The architecture of the framework is illustrated in Figure 3.

5.1 Description Module (DM)

The Description Module (DM) is implemented using the WINGS (Workflow Instance Generation and Selection) system [10]. In the WINGS system, 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. The 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 of the first component are consistent with the input data types 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 a workflow template and the input metadata, WINGS can automatically generate a detailed specification of the workflow components and data flow in the form of a DAG (also referred to as an expanded *workflow instance*) for execution.

The WINGS system provides “core” ontologies that can describe generic 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 stored in domain-specific ontologies. Workflow templates for applications within that domain are constructed using these ontologies. Next, we present the data ontology we have developed for our motivating applications.

Domain-specific Data Ontology—The core data ontology in WINGS contains OWL-based hierarchical definitions for generic data entities such as a File, a Collection of Files of the same type, and CollOfCollections. We have extended this core ontology as shown in Figure 4 to capture the data model commonly adopted in applications within the spatial data analysis domain. For instance, chunk-based analysis of out-of-core image data requires the expression of the data at various levels of abstraction: image, chunk, tile, slice, and stack. Our domain-specific data ontology defines these concepts and how they relate to each other, and allows application data flow to be 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 NB workflows. It can also be extended to support a wider range of spatial data applications. The core component ontology in WINGS was also extended to represent sequential components, component collections (bag-of-tasks style operations), and complex parallel components in our workflows.

For component collections, multiple instances of a component can be dynamically created depending on the properties of the data, and tasks corresponding to these instances are scheduled independently for execution. This enables performance improvements by adjusting both the task- and data-parallelism parameter of the application workflow as well as application performance parameters such as chunksize. The chunksize parameter dictates the unrolling of component collections in a workflow template into a bag of tasks in the workflow instance for a given input. As an example assume the chunksize value chosen for an input image results in the image being partitioned into 4 chunks. The corresponding unrolled workflow instances for the PIQ and NB workflows are shown in Figure 5. The cases where multiple `ovals' exist at the same horizontal level indicate component collections that have been automatically expanded, based on the chunksize, into multiple (in this case 4) component instances. Thus, changing the chunksize parameter can lead to different workflow structures depending on the data.

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

In our current system, we also support the notion of meta-components or explicit component-grouping. A meta-component is a combination of components in a workflow. For example, the grey rectangles for the NB workflow in Figure 5 indicate that all steps have been fused into a meta-component. By coalescing components into meta-components, a higher-granularity template of the workflow can be created. The higher-granularity template and meta-components correspond 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 resources.

5.2 Execution Module (EM)

The *Execution Module* (EM) consists of three 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 [9]. Pegasus takes resource-independent workflow descriptions (DAX) 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 software 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 [1] as explained later in this section.

Condor [19] is used to schedule tasks across machines. Pegasus submits tasks in the form of a DAG to Condor instances running locally at each Grid site.

DataCutter [1] 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) 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. 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 the use of high-speed interconnects.

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* [14]. We have investigated techniques which dynamically order the processing of data elements to speed up application execution while taking into account user-defined QoS requirements on output quality. The Trade-off Module (TM) draws from and implements the runtime support for these techniques so that quality of output can be traded for improved performance.

We provide generic support for such a reordering of data processing 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 scheme [14] in a manner that is not tied to any particular application. It relies on the semantic representations of the data chunks being processed in a batch in order to map jobs to the spatial coordinates of the chunks. Using this association, the priority queue within the custom scheduler can decide which job to schedule next, based on the spatial coordinates of the chunk belonging to that job's specification. 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 released ahead of other jobs. In this way, the customized scheduler helps exercise control over the processing order parameter. When there are no QoS requirements, 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 output accuracy for performance with user-defined quality of service requirements. For other types of performance optimization parameters, the current implementation of the framework provides support for users to specify and express the values of various performance parameters to improve performance of the workflow. We view this as a first step towards an implementation that can more 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 [21] 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 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 2×250GB SATA disks installed locally on each compute node, joined into a 437GB RAID0 volume. The second cluster, **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 flavor4 of MPI for communication in order to exploit InfiniBand support.

In the following experiments, we evaluate the performance impact of a set of parameter choices on workflow execution time. First, a set of Quality-Preserving parameters are explored, as we would initially like to tune the performance without modifying the results of the computation. We subsequently investigate the Quality-trading parameters. In our experiments, we treat different parameters independently, selecting a default value for one parameter while exploring another. The decision as to which parameter to explore first is one that can either be made by an application programmer, or can be evaluated systematically by a set of measurements, such as the sensitivity analysis found in [7].

6.1 Quality-preserving Parameters

The basic template for the PIQ workflow is one where tasks corresponding to each component and component collection are mapped to target sites by Pegasus. Our earlier work [13] showed that components like *normalize* and *autoalign* execute in less time on faster machines equipped with high-speed interconnects. Based on these experiences, we adopted a specific component placement strategy to achieve improved performance. Components *zproject*, *prenormalize*, *stitch*, *reorganize*, *warp* and the preprocessing tasks execute on the RII-MEMORY cluster, while *normalize*, *autoalign* and *mst* execute on 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 addition to mapping compute-intensive tasks to faster sites.

Effects of chunksize—For these experiments, we used a 5 GB image with 8 focal planes, where each plane has 15360 × 14000 pixels. The chunksize parameter determines the unrolling factor for component collections in the template. Different chunksize parameter

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

values resulted in workflow instances with different structures with different number of tasks, as shown in table 1.

The disparity among the number of resulting tasks for different chunksizes will grow when we have larger images (leading to more chunksize values) and more component collections. Hence, job submission and scheduling overheads play a significant role in the overall execution times. To alleviate these overheads, a possible runtime optimization is to use horizontal job clustering by Pegasus for every component collection. (The table also shows the total number of tasks for each chunksize value when tasks from component collections in the PIQ workflow are grouped into bundles of 32 tasks each.)

Figure 6 shows the overall execution time for each workflow instance corresponding to a different chunksize, both with and without using horizontal job clustering by Pegasus. This includes the time to stage-in and stage-out data for each site, the execution times for tasks from each component, and the job submission and scheduling overheads. We observe that: (1) Values at both extremes of the chunksize range do not yield good performance. The submission and scheduling of a large number of tasks causes poor performance when the smallest chunksize value is used. For the largest chunksize value, the number of chunks in an image place becomes less than the number of worker nodes available for execution. Since each task operates at the minimum granularity of a chunk, this leads to under-utilization of resources and hence, higher execution times. (2) The intermediate values for chunksize yield more or less similar performance, except for an unexpected spike at chunksize = 2560 × 2400. On further analysis of the individual component performance, we observed that the algorithm variant used for the *warp* component (which accounts for nearly 50% of the overall execution time of PIQ) performed poorly at this chunksize value. This shows that the chunking strategy can affect performance not only at the workflow level but also at the level of each component. (3) Horizontal job clustering improves performance when the unrolling factor for each component collection is large. For larger chunksize values, this factor is not high enough to justify the overheads of job clustering.

We observed similar trends for large images as well. Our experiments with 75 different chunksize values for a 17 GB image (with 3 focal planes, each plane having 36864 × 48000 pixels) showed that very small or very large chunk sizes lead to poor performance owing to reasons outline above. We also observed that chunksize values that resulted in long horizontal stripe chunks yielded optimal performance for this workflow and image class. This shows 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 parameters like chunksize based on trends generated from training data.

Effect 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. In this case, tasks corresponding to a meta-component (and not a component), are submitted to the execution module. Here, the *zproject* and *prenormalize* steps from the original template are fused to form *metacomponent1*, *normalize*, *autoalign*, *mst* are collectively *metacomponent2* while *stitch*, *reorganize*, *warp* form *metacomponent3*. *Preprocess* is the fourth meta-component in the workflow template and includes the *threshold*, *tessellate* and *prefix sum* 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 within a DataCutter instance. That is, the chunksize input parameter to a meta-component is propagated to the Data-Cutter filters set up for each component within that meta-component. Based on chunksize, DataCutter will create multiple copies of filters that handle the

processing of component collection tasks. Each such filter copy or instance will operate on a single chunk. We also manipulated the execution strategy within the *preprocess* meta-component alone, such that data within this meta-component is pipelined and streamed between its components without any disk I/O during execution.

Figure 7 shows that the overall execution time for the PIQ workflow improves by over 50% when we use the high-granularity workflow template for execution. The figure also shows the changes in execution times for individual components of the workflow. Embedding heterogeneous parallel components like *autoalign* and *warp*, as expected, makes no difference to their execution times. 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 is a batch system 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 can function like a service-oriented system, in which filters are set up on each worker node. These filters perform the desired startup operations on each node and process multiple data instances, much like a service. The input data to a collection can be streamed through these filters. Thus, effectively, the startup overheads are incurred only once per worker. 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.

6.2 Quality-trading Parameters

These experiments demonstrate performance gains obtained when one or more quality-trading parameters were changed 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 cluster5 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 quality-trading parameters.

Tuning both the resolution and processing order parameters for requirement 1—This is useful when chunks can be processed at lower resolutions so long as the resulting confidence exceeds a user-specified threshold. Figure 8 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.

[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.

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.

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 9 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.

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 10 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 quality of output 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-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 support for trading off output quality for performance in our target class of applications. The experimental evaluation of the proposed framework shows that adjustments of quality-preserving and quality-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.

Acknowledgments

This research was supported in part by the National Science Foundation under Grants #CNS-0403342, #CNS-0426241, #CSR-0509517, #CSR-0615412, #ANI-0330612, #CCF-0342615, #CNS-0203846, #ACI-0130437, #CNS-0615155, #CNS-0406386, and Ohio Board of Regents BRTTC #BRTT02-0003 and AGMT TECH-04049 and NIH grants #R24 HL085343, #R01 LM009239, and #79077CBS10.

8. REFERENCES

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

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

[3]. Chang F, Karamcheti V. Automatic configuration and run-time adaptation of distributed applications. High Performance Distributed Computing. 2000:11–20.

[4]. Chiu, D.; Deshpande, S.; Agrawal, G.; Li, R. Cost and accuracy sensitive dynamic workflow composition over Grid environments. 9th IEEE/ACM International Conference on Grid Computing; Oct. 2008; p. 9-16.

[5]. Chow SK, Hakozaki H, Price DL, MacLean NAB, Deerinck TJ, Bouwer JC, Martone ME, Peltier ST, Ellisman MH. Automated microscopy system for mosaic acquisition and processing. Journal of Microscopy. May; 2006 222(2):76–84. [PubMed: 16774516]

[6]. Chung, I-H.; Hollingsworth, J. A case study using automatic performance tuning for large-scale scientific programs. 15th IEEE International Symposium on High Performance Distributed Computing; 2006. p. 45-56.

[7]. Chung, I-H.; Hollingsworth, JK. Using information from prior runs to improve automated tuning systems. SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing; Washington, DC, USA. IEEE Computer Society; 2004. p. 30

[8]. Cortellessa, V.; Marinelli, F.; Potena, P. Automated selection of software components based on cost/reliability tradeo. Lecture Notes in Computer Science; Software Architecture, Third European Workshop, EWSA 2006; Springer; 2006.

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

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

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

[12]. Kong, J.; Sertel, O.; Shimada, H.; Boyer, K.; Saltz, J.; Gurcan, M. Computer-aided grading of neuroblastic differentiation: Multi-resolution and multi-classifier approach. IEEE International Conference on Image Processing, ICIP 2007; Oct. 2007; p. 525-528.

[13]. Kumar V, Rutt B, Kurc T, Catalyurek U, Pan T, Chow S, Lamont S, Martone M, Saltz J. Large-scale biomedical image analysis in Grid environments. IEEE Transactions on Information Technology in Biomedicine. March; 2008 12(2):154–161. [PubMed: 18348945]

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

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

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

[17]. Norris, B.; Ray, J.; Armstrong, R.; Mcinnes, LC.; Shende, S. Computational quality of service for scientific components. International Symposium on Component-based Software Engineering (CBSE7); Springer; 2004. p. 264-271.

[18]. Oinn T, Addis M, Ferris J, Marvin D, Senger M, Greenwood M, Carver T, Glover K, Pocock MR, Wipat A, Li P. Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics. 2004; 20(17):3045–3054. [PubMed: 15201187]

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

[20]. Truong H-L, Dustdar S, Fahringer T. Performance metrics and ontologies for grid workflows. Future Generation Computer Systems. 2007; 23(6):760–772.

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

[22]. Zhou, J.; Cooper, K.; Yen, I-L. A rule-based component customization technique for QoS properties. Eighth IEEE International Symposium on High Assurance Systems Engineering; March 2004; p. 302-303.

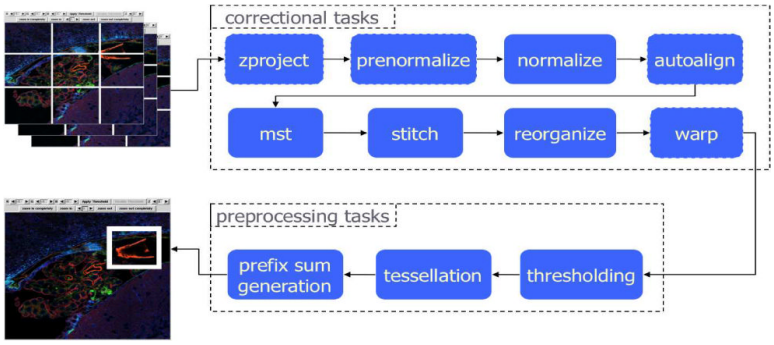

Figure 1.
PIQ workflow

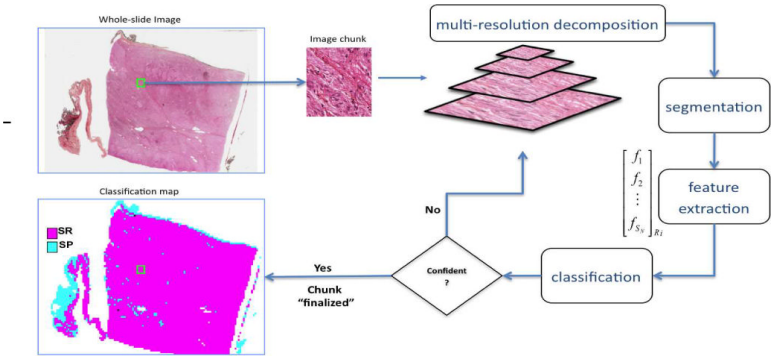

Figure 2.
NB workflow

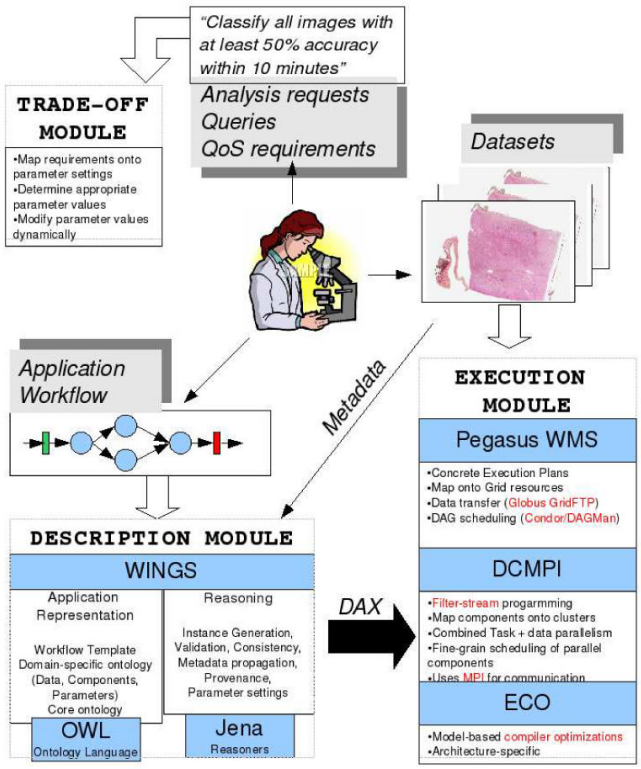

Figure 3.
Framework architecture

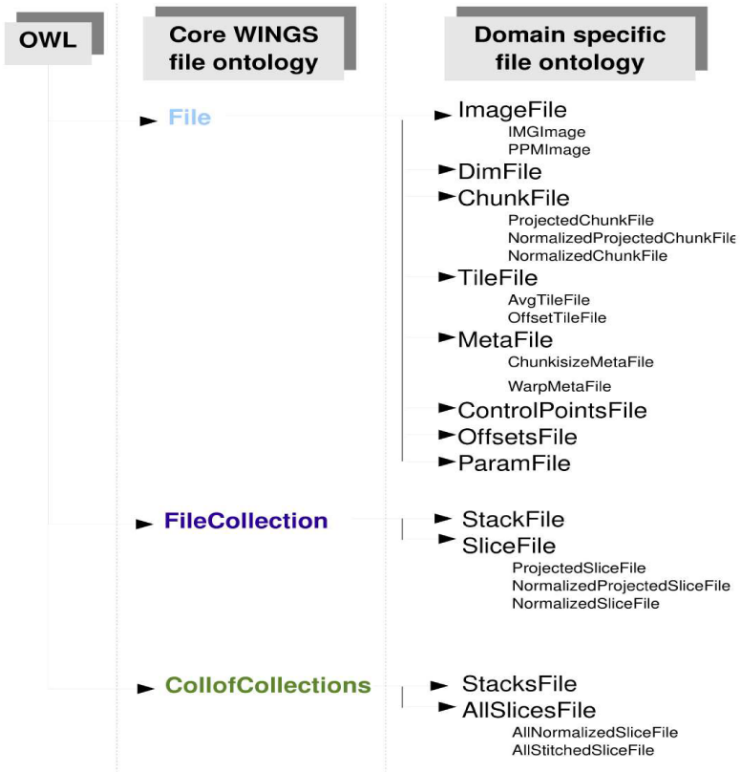

Figure 4.
Extensions to core WINGS data ontology

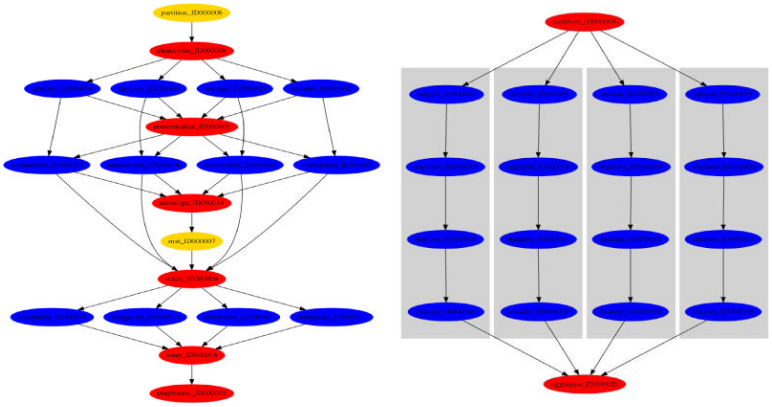

Figure 5.
Expanded workflow instance for PIQ and NB workflows (chunksize is chosen such that #chunks=4).

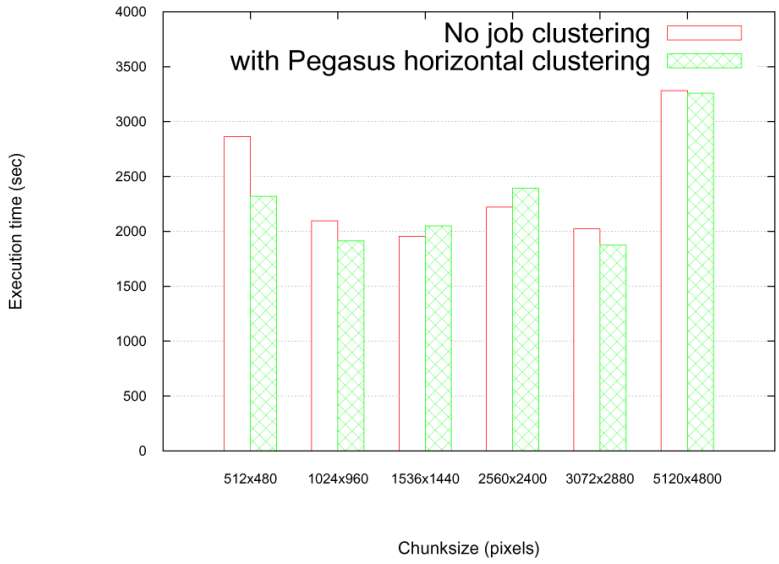

Figure 6.
Total workflow execution time for various chunksize parameter values

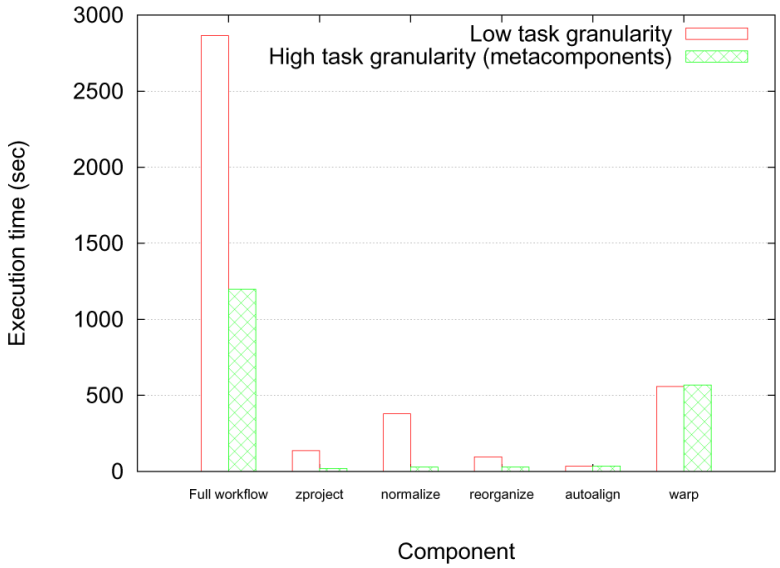

Figure 7.
Execution time with different task granularity (5GB image, 40 nodes, chunksize=512 × 480)

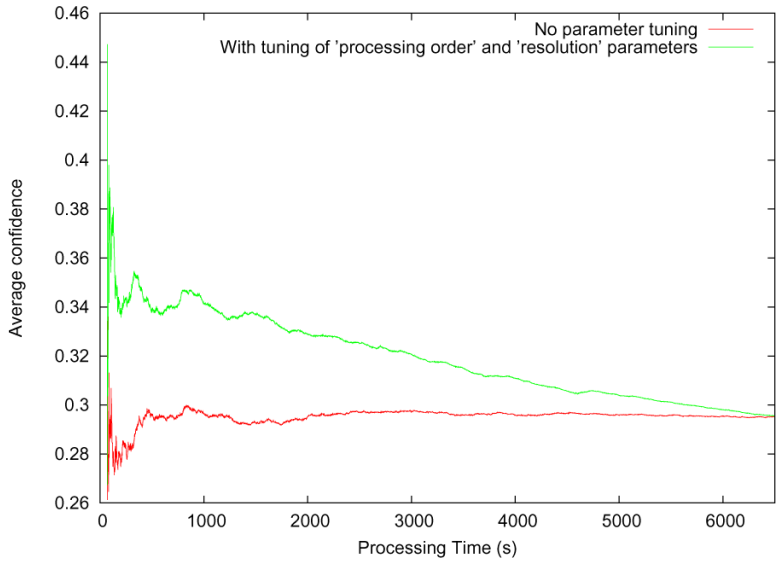

Figure 8.
Quality improvement by tuning the processing order and resolution parameters

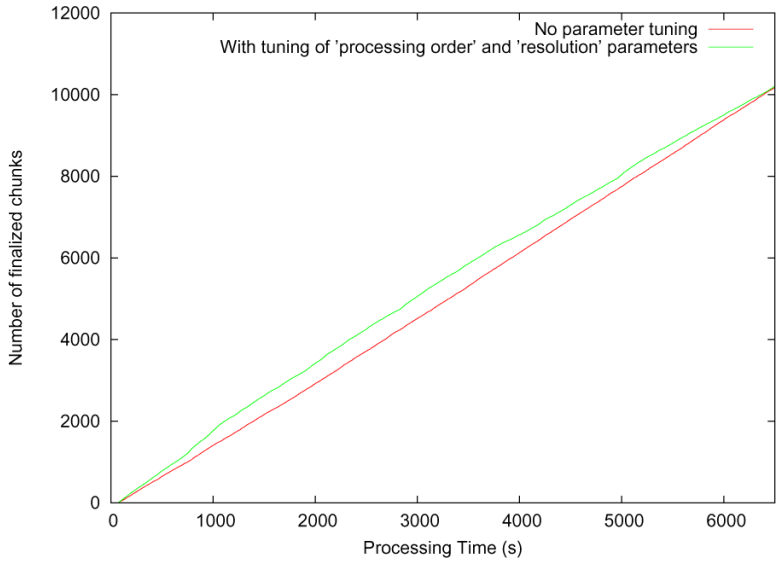

Figure 9.
Improvement in number of finalized tiles by tuning parameters

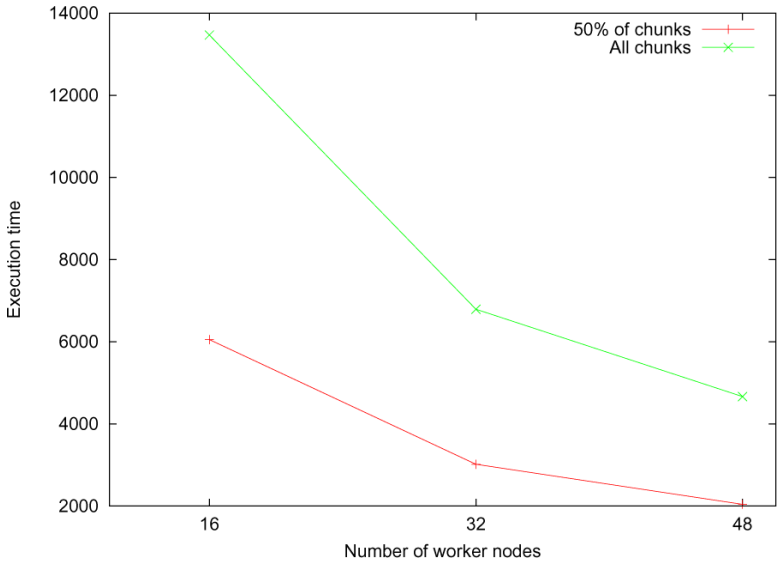

Figure 10.
Scalability with number of worker nodes

Table 1

Number of jobs in workflow instance for each chunksize

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