WORKFLOW SCHEDULING FOR SERVICE ORIENTED CLOUD COMPUTING

A Thesis Submitted to the College of

Graduate Studies and Research

In Partial Fulfillment of the Requirements

For the Degree of Master of Science

In the Department of Computer Science

University of Saskatchewan

Saskatoon

By

ADNAN FIDA

Keywords: cloud, grid, scheduling, services, simulation, workflows

PERMISSION TO USE

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Sciences

176 Thorvaldson Building

110 Science Place

University of Saskatchewan

Saskatoon, Saskatchewan

S7N 5C9

ABSTRACT

Service Orientation (SO) and grid computing are two computing paradigms that when put together using Internet technologies promise to provide a scalable yet flexible computing platform for a diverse set of distributed computing applications. This practice gives rise to the notion of a computing cloud that addresses some previous limitations of interoperability, resource sharing and utilization within distributed computing.

In such a Service Oriented Computing Cloud (SOCC), applications are formed by composing a set of services together. In addition, hierarchical service layers are also possible where general purpose services at lower layers are composed to deliver more domain specific services at the higher layer. In general an SOCC is a horizontally scalable computing platform that offers its resources as services in a standardized fashion.

Workflow based applications are a suitable target for SOCC where workflow tasks are executed via service calls within the cloud. One or more workflows can be deployed over an SOCC and their execution requires scheduling of services to workflow tasks as the task become ready following their interdependencies.

In this thesis heuristics based scheduling policies are evaluated for scheduling workflows over a collection of services offered by the SOCC. Various execution scenarios and workflow characteristics are considered to understand the implication of the heuristic based workflow scheduling.

ACKNOWLEDGMENTS

I am thankful to my supervisor Dr. Ralph Deters for his continuous support and encouragement to complete my thesis. Dr. Deters provided guidance and direction that was necessary for me to move forward. I would also like extend thanks to my thesis committee members including Dr. John Cooke, Dr. Julita Vassileva, and Professor. Denard Lynch who have provided valuable feedback on this thesis. Additionally, Ms. Jan Thompson, Graduate Correspondent at the department of Computer Science, has also been very helpful throughout my program.

Finally I would also like to thank my colleagues, my employer and friends for their trust and support.

TABLE OF CONTENTS

page

PERMISSION TO USE i

ABSTRACT ii

ACKNOWLEDGMENTS iii

LIST OF TABLES vii

LIST OF FIGURES viii

LIST OF ABBREVIATIONS x

INTRODUCTION 1

RESEARCH DESCRIPTION 5

- Scenario 5
- Opportunity 6
- Problem 6
- Research Questions 7
  - System Throughput 8
  - Workflow Performance 8
  - Cloud Size 8
  - Workflow Arrival Pattern 8
  - Workflow Structure 9
  - Workflow Execution Deadlines 9

LITERATURE REVIEW 10

- Enterprise Grid Computing 10
  - Definition 10
  - Service Orientation 11
  - Scheduling 11
  - Summary 18
- Desktop Grid Computing 18
  - Definition 18
  - Service Orientation 19
  - Scheduling 19
  - Summary 23
- Scientific Grid Computing 23
  - Definition 23
  - Service Orientation 23
  - Scheduling 24
  - Summary 28
- Workflow Description Languages 28
  - Summary 30

Composite Web Services 30
Summary 32
Theoretical Work 32
Summary 35
Conclusions 35
RESEARCH GOALS 38

WORKFLOWS & SCHEDULING 40

Task Readiness 42
Size 42
Depth 43
Breadth 43
Arrival Time 44
Scheduling 44
EXPERIMENTATION 47

Simulation Methodology 47
Cloud Modeling 47
Services 48
Workflows 48
Scheduler 51
Measurements 51
Wait Time 51
Execute Time 51
Finish Time 51
Total Finish Time 52
Average Finish Time 52
Scheduling Policies 52
Workflow Prioritization 52
Service Partitioning 56
Deadlines 57
Objectives 59
RESULTS 60

Workflow Prioritization 61
Workflow and Cloud configuration 61
Results 62
System throughput 63
Workflow Performance 65
Cloud Size 67
Summary 69
Service Partitioning 70
Workflow and Cloud configuration 70
Results 72
System Throughput & Cloud Size 73
System Throughput and Heuristics 77

Workflow performance & Heuristics....80
Workflow Arrival Patterns....81
Service Utilization....81
Workflow Structure....81
Summary....82
Deadlines....83
Workflow and Cloud Configuration....83
Results....84
Summary....89
CONCLUSIONS AND FUTURE WORK....91
Research Results....91
Workflow Prioritization....92
Service Partitioning....92
Deadlines....93
Future Work....93
Conclusions....94
REFERENCES....97

LIST OF TABLES

Table page

Table 3-1. Overview of sample enterprise grid computing systems....17

Table 3-2. Overview of sample desktop grid computing systems....22

Table 3-3. Overview of sample scientific grid computing systems....27

Table 3-4. Characteristics of sample workflow description languages....29

Table 3-5. Overview of sample composite web services systems....32

Table 3-6. Overview of sample theoretical research towards workflow scheduling....34

Table 6-1. List of scheduling heuristics based on size (execution lengths)....53

Table 6-2. List of scheduling heuristics based on task dependencies....54

Table 7-1. All possible values of the experimentation parameters....60

Table 7-2. Workflow configuration parameters for the priority scheduler....62

Table 7-3. Workflow detailed configuration for the priority scheduler....62

Table 7-4. System configuration parameters for prioritization scheduler....62

Table 7-5. Workflow configuration parameters for the partitioning scheduler....71

Table 7-6. Configuration for the service partitioning scheduler....72

Table 7-6. Workflow configuration parameters for deadlines scheduler....83

Table 7-7. Deadlines scheduler run configuration....84

LIST OF FIGURES

Figure page

Figure 1-1. Resource silos with an enterprise datacenter 2

Figure 1-2. A simple Service Oriented Computing Cloud (SOCC) 3

Figure 3-1. Composite services as an execution unit for a SOCC.....31

Figure 5-1. An example DAG structured workflow 40

Figure 5-2. Workflow transformation for a single parent per child.....41

Figure 5-3. Workflow (DAG) loop unrolling 41

Figure 5-4. A workflow with four levels 42

Figure 5-5. An example workflow of size 55 (10 + 20 + 25) 42

Figure 5-6. Example workflow depth 43

Figure 5-7. An example workflow breadth of six tasks.....44

Figure 5-8. Overview of scheduling workflow collections over SOCC.....45

Figure 6-1. An example of EDAG & CDAG workflow types for the experimentation....49

Figure 6-2. Example arrival time distributions for 25 workflows 50

Figure 6-3. Example size distributions for 25 workflows.....50

Figure 6-3. A simplified scheduling scenario using size based heuristics.....54

Figure 6-4. A simplified scheduling scenario using task dependency based heuristics 55

Figure 6-5. A simplified scheduling scenario with the service partitioning.....56

Figure 7-1. Finish times for set I workflows across all scheduling heuristics.....64

Figure 7-2. Finish times for set II workflows across all scheduling heuristics.....64

Figure 7-3. Finish times for set II workflows across selected scheduling heuristics.....66

Figure 7-4. Impact of the additional number of services on the performance of heuristics68

Figure 7-5. Scheduling heuristic performance with (uniform) burst arrival pattern.....69

Figure 7-6. Total finish times for set II workflows with Arrival Time heuristic.....74

Figure 7-7. Average wait times for set I workflows with Shortest Workflow heuristic....75
Figure 7-8. Total finish times for set III workflows with Arrival Time heuristic 76
Figure 7-9 Scheduling heuristic performance for total finish time.....77
Figure 7-10 Average finish time of set II for Longest Workflow heuristic.....78
Figure 7-11. Average finish time of set III for Arrival Time heuristic.....79
Figure 7-12. Average finish time of set III for Longest Workflow heuristic 80
Figure 7-13. Scheduling heuristic performance for average finish time 81
Figure 7-14. Performance of deadlines scheduler for Set I workflows 85
Figure 7-15. Performance of deadlines scheduler for set II workflows.....86
Figure 7-16. Set I workflow incompletion due to a certain deadline type.....87
Figure 7-17. Set II workflow incompletion due to a certain deadline type 88
Figure 7-18. Set I workflow completion times (total finish times).....89
Figure 7-19. Set II workflow completion times (total finish times) 89

LIST OF ABBREVIATIONS

SO	Service Orientation
SOCC	Service Oriented Computing Cloud
SOA	Service Oriented Architecture
SaaS	Software as a Service
DAG	Directed Acyclic Graph
EDAG	Evolving Directed Acyclic Graph
CDAG	Constant Directed Acyclic Graph
ST	Shortest Task
LT	Longest Task
SW	Shortest Workflow
LW	Longest Workflow
MCT	Most Completed Tasks
MOT	Most Outstanding Tasks
LDT	Least Dependent Tasks
MDT	Most Dependent Tasks
ET	End Time
MEL	Maximum Execution Length
MPI	Message Passing Interface
RM	Reduction-Mesh
RT	Reduction-Tree
FFT	Fast Fourier Transformation
FCFS	First Come First Served

CHAPTER 1
INTRODUCTION

With the advent of the Internet technologies, many new computing paradigms have emerged over the last few years. Two such paradigms that have gained considerable attention are grid computing [1] and service oriented computing [2-4]. Grid computing evolved from the resource aggregation models to meet the ever increasing need for computing power. With the Internet, it became possible to connect distributed resources together to create a grid of computing resources that together can serve as an execution platform for diverse computing applications. At the same time existing or new applications started to appear over the Internet as services that can be consumed by other applications.

Both paradigms compliment each other and together they can further enhance the benefits offered by them individually [5]. Combined together they offer a computing environment that inherits the scalability of a grid and the flexibility of service-orientation to form a more robust distributed computing platform [6]. While the grid offers protocols to create large scale computing platforms, the Service Oriented Architecture (SOA) offers a standardize way to expose resources and applications for a grid over the Internet. Given that Internet is often referred to as a computing cloud, the notion of computing services backed by a grid over a cloud can be referred to as Services Oriented Computing Cloud (SOCC) [7-9]. Simply put, SOCC is a computing grid that offers its resources via (web) services. Every computing cloud is backed by a grid, however not every grid powers a computing cloud.

SOCC type models have attracted attention from both research organizations and commercial enterprises [10-12]. For enterprise datacenters in particular, SOCC promises to reduce the

overhead associated with the resource management and maintenance of hosting of a large number of independent applications [13]. In absence of SOCC, dedicated hardware is allocated to the applications. The physical resources are allocated to each application such that the application performance is acceptable during the peak load times. Given that peak load time windows are relatively small and occur infrequently, the physical resources allocated to each application tend to sit idle and cannot be shared by other applications that can benefit from more resources. This practice ends up creating resource silos within enterprise datacenters and requires management and maintenance of each silo independently. This scenario is depicted in figure 1-1 where two such simplified silos for two application workflows are shown for the illustration purposes.

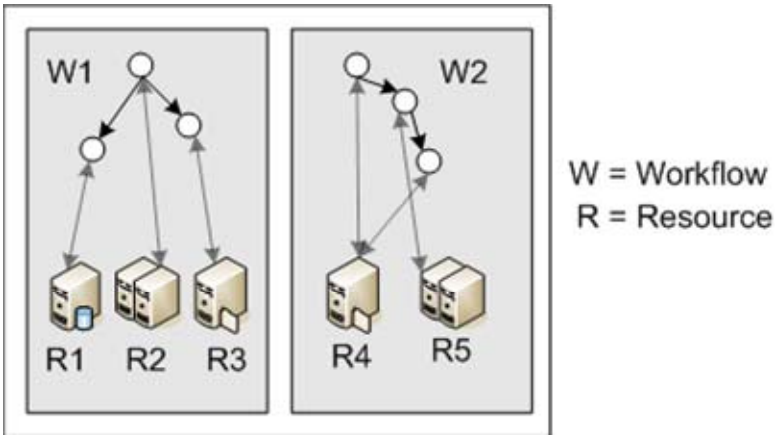

Figure 1-1. Resource silos with an enterprise datacenter

SOCC provides a way to break resource silos by exposing resources as general purposes reusable services that can be shared among many applications. For example, the figure 1-1 scenario is redrawn in figure 1-2 to show the resource sharing by introduction of a service layer. With this approach, datacenters can easily scale horizontally by adding more resources and exposing them through existing or additional services [10]. The resource maintenance overhead is reduced by having to maintain a general purpose resource cloud as opposed to a segmented

resource collection. In addition to the resource sharing, interoperability among independent applications also becomes possible. Applications themselves can offer their functionality in terms of the services that can be utilized by other applications. Application development also benefits from the service orientation as it can serve as a standard methodology when creating and deploying new applications.

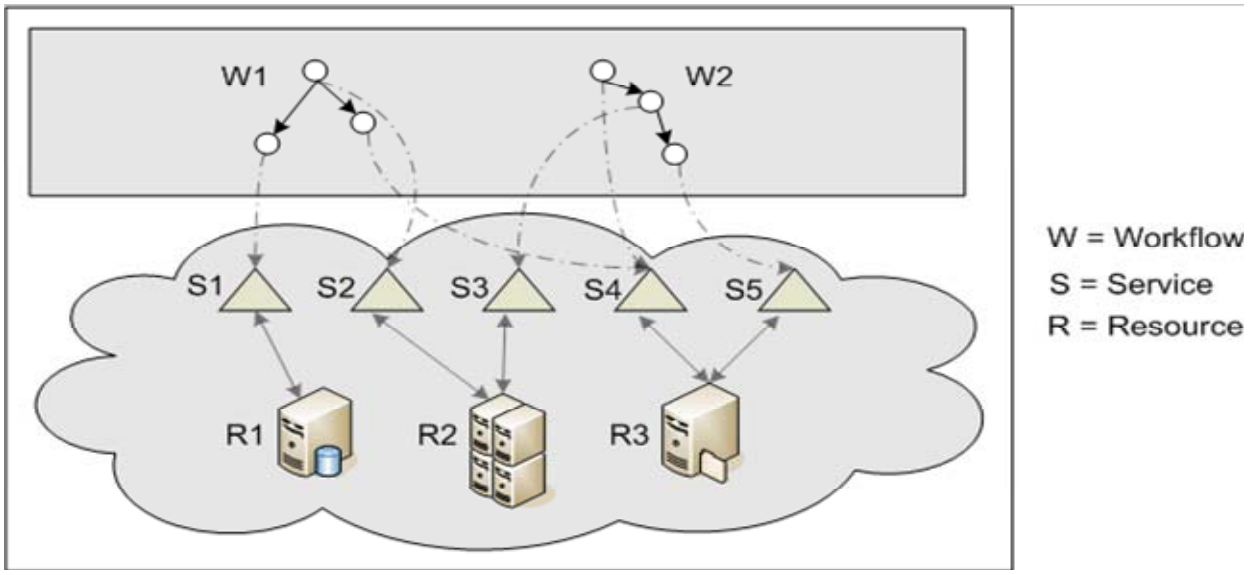

Figure 1-2. A simple Service Oriented Computing Cloud (SOCC)

Given the benefits associated with SOCC it is interesting to evaluate different application models that can benefit from such a computing platform. One such application model is workflow based. In a workflow one or more tasks together in some order are executed to complete a business process [14]. In the scope of enterprise SOCC, the workflows can represent business processes such as generating insurance quotes, performing credit card billing, generating business intelligence reports, processing data for business integration, etc.

Workflow execution over a SOCC can be conceived as one or more service calls to execute workflow tasks. Such an execution requires the scheduling of workflow tasks to the services when tasks become ready for execution following their task dependency order. In addition, in an

enterprise SOCC it is likely that multiple workflows require execution at the same time. Figure 1-1 provides an example of a simplified SOCC for two workflows. From the figure, workflow tasks are scheduled to five different services that offer three compute resources. Each workflow has its own tasks and execution order (task dependencies). Workflows can also have execution constraints associated with them such as execution completion deadlines, or other performance requirements.

Execution of multiple-workflows requires careful scheduling over an SOCC. A scheduler needs to account for all outstanding workflows in the system and perform scheduling using some technique to ensure that the system goals are met [15]. System goals can be workflow performance, uniform resource utilization, system throughput, etc.

In this work, workflow scheduling for a collection of independent workflows in a scope of a reference SOCC is studied. In particular heuristic based workflow scheduling is examined with the help of experiments. The thesis presents experimentation results that are useful when implementing workflow scheduler for SOCC. The thesis is organized as follows.

The second chapter provides a detailed research problem description. The third chapter provides the relevant work towards the scheduling within the related domains of service orientation and computing grids. In the fourth chapter, research goals and objectives are outlined. The fifth chapter provides terminology for workflow scheduling within the cloud. Experimentation methodology and setup are described in chapter six, and the seventh chapter contains the experimentation result discussion. Chapter eight concludes the thesis.

CHAPTER 2
RESEARCH DESCRIPTION

Scenario

In an enterprise datacenter implemented as an SOCC, there can be one or more collections of workflows that require execution at regular intervals. For example, every 12 hours, a collection of workflows need to be executed. Workflows are independent of each other and can run simultaneously however they are generally long running applications with execution lengths that can span multiple hours. During their execution all of the workflow tasks are performed by invoking various services. The services that are consumed by the workflows are deployed across various resources within the cloud.

The underlying resources and the corresponding services are shared among the workflows as the services offer common functionality such as database access, billing services, image processing, etc. At anytime, the number of tasks requiring execution can be much higher than the number of available services in the cloud. For example, there can be 250 tasks requiring execution while the total number of services in a cloud is 50. A large task to service ratio is desirable to control the cloud costs when service utilization across workflow tasks is not uniform (not all of the services are invoked continuously).

The scheduling of the workflows is typically performed manually by the IT staff. The scheduling entails the selection of the services and the appropriate start time for each workflow. Some manual estimation is performed by the IT staff to ensure that the final schedule provides the completion of all the workflows.

Opportunity

The workflow characteristics such as number of tasks and corresponding execution lengths are variable. Workflows continuously evolve as the underlying data changes and through the ongoing business activity. While existing workflows change their requirements, new workflows are also continuously added to the cloud to support the business growth. The changes in the existing workflow requirements and the additional new workflows require more complicated schedules by the IT staff. Given that the workflow lengths are long and the number of tasks (service calls) regularly change, it becomes increasingly cost prohibitive to devise the schedules by hand.

Such a datacenter SOCC can benefit from the automation of the workflow scheduling to reduce the human involvement and therefore reduce the costs associated with composing and maintaining the workflow schedules.

Problem

By considering the above mentioned scenario it is possible to generalize the problem of workflow scheduling automation as follows.

In a service oriented computing cloud, a workflow collection requires execution over a shared set of services. Each workflow is executed by executing all of the workflow tasks in the order of their dependencies. Therefore at anytime one or more tasks belonging to one or more workflows are ready for execution. The number of available services is generally much lower than the number of outstanding tasks and therefore, tasks must be scheduled to the services in some order. The workflow execution requires automated scheduling to avoid the human interaction and the related cost associated with manual scheduling.

The scheduler has to decide on the workflow/task execution order and the corresponding service allocation. For simplicity it can be assumed that all of the services and tasks are homogeneous. However, there are other variables in the cloud that require consideration while making scheduling decisions. These variables are the total number of services, the total number of workflows with their corresponding tasks, and the execution length of each task. Further not all of the workflows are ready for execution at the same time, and their readiness or arrival into the system is to be considered, as well.

In this work, workflows are Directed Acyclic Graph (DAG) based and therefore their scheduling over a distributed set of services is an NP complete problem [16, 17]. But approximation techniques can be used for their scheduling. It is possible to apply scheduling heuristics for workflow scheduling. For example, scheduling workflows in the order of their execution lengths and allocating a fixed number of services to the workflow are two scheduling heuristics. The heuristics can be based on certain scheduling policies. For example, in order to select among multiple ready tasks, scheduling can use a *prioritization* policy to devise certain selection heuristics such as task execution lengths. However, each scheduling policy (and the heuristics) will impact the performance of the workflows and of the cloud in some way. Further, certain policies will perform differently with different cloud configurations. For example, scheduling workflows in the order of their arrival may only work well when the overall number of tasks is small and the workflows complete their execution before more workflows arrive for execution.

Research Questions

Two scheduling policies of *workflow prioritization* and *service partitioning* are considered in this work. The prioritization policy helps to select among many ready tasks or workflows for

scheduling to a small number of services. The partitioning policy helps to allocate a fixed number of services to each workflow. Various scheduling heuristics are used to implement both policies, resulting in many scheduling scenarios. For example, in one scheduling scenario, the workflow execution length is the heuristic to prioritize the workflows. In another scenario a fixed number is used to partition the available services among the prioritized workflows. A scheduling scenario is a representation of one or more scheduling heuristics based on one or both scheduling policies.

In order to understand the impact of each scheduling policy on various scenarios the following two metrics are used.

System Throughput. The system throughput is measured in terms of the total completion time of all the workflows in the system.

Workflow Performance. The workflow performance is measured in terms of the average workflow completion time.

There are several variables that can affect the cloud behavior for the system throughput and the workflow performance. Together these variables define the cloud configuration. In this research following variables are considered to represent various cloud configurations.

Cloud Size. The number of services in a cloud available for workflow execution determines the cloud size or its capacity.

Workflow Arrival Pattern. The workflow arrival in a cloud can follow some distribution such as Uniform, Poisson, or Exponential. The workflow arrival pattern dictates the load on the cloud.

Workflow Structure. The workflow structure is described in terms of the number of tasks, their dependencies on each other, and the number of concurrent tasks. More formal details on the workflow structure are provided later in chapter 5.

Workflow Execution Deadlines. Workflows can have deadlines associated with them in terms of the completion times. For example, a workflow must finish its execution by a certain deadline after it starts the execution.

In this research, a selected set of scheduling heuristics are applied against the various values of the above mentioned cloud variables. The performance of the scheduling policies is evaluated using the two above mentioned metrics to answer the following research questions.

1. How does the cloud size impact the performance of the various scheduling policies in terms of the system throughput and the workflow performance? In general the performance of any scheduling policy will likely benefit with the increased number of services in a cloud. However, performance of various policies is compared with each other for different cloud sizes to evaluate their suitability.
2. How do the workflow arrival patterns impact the scheduling? It is possible that not all of the workflows are ready for execution at the same time therefore scheduling is an ongoing activity. Different arrival patterns cause different amounts of workload on the cloud. The performance of the scheduling policies under different loads is examined.
3. What role does a workflow structure play during scheduling? In particular what attributes of the workflow structure, such as number of tasks or the task dependencies are valuable in workflow scheduling?
4. How do the workflows that have certain completion deadlines perform with the prioritization and the partitioning policies?

More specific details on the values of the cloud configuration variables and the implemented scheduling heuristics are provided later in chapter 6 and 7.

CHAPTER 3
LITERATURE REVIEW

The research presented in this work is inspired by many elements of distributed computing including grid, workflow, and the service-orientation. The application of these elements occurs in various settings that can range from a small set of computing services to large-scale geographically dispersed high processing computing environments. Both academic and business enterprises are involved in the various modalities of the service oriented computing clouds of various sizes. Not all of the modalities provide execution for the workflow based applications. The workflow composition and usage is also diverse across the computing environments.

In this chapter research is presented to establish the context for this work. As grid computing plays an essential role for SOCC and given that a computing cloud is a relatively new paradigm, the presented work is selected based on the following criteria.

- Scheduling within the grid computing environments. E.g. enterprise, desktop, or scientific grids that provide the backbone of a computing cloud
- Service Oriented (SO) execution model
- Scheduling for single or multiple workflows including the description of workflow for scheduling purposes

The following sub sections categorize the related work with a brief description of each category. The categories are outlined in no particular order.

Enterprise Grid Computing

Definition

Enterprise grid computing refers to the commercial IT data centers that are host to various applications conducting the business operations [18, 19]. The application types include Customer Relationship Management (CRM), Enterprise Resource Planning (ERP), Business Intelligence (BI), and Management Information Systems (MIS) [20]. These applications are more and more

commonly being offered though web based portals or other Internet backed interfaces. The execution mode of these applications can be both online and offline. Further most of these applications contain many business processes that are represented as workflows. The workflows are transactional, database, and IO bound [21]. The underlying grid resources are web and application servers, databases, media storage, and infrastructure resources such as network load balancers and firewalls.

Service Orientation

Over time more and more enterprise grids are adopting the service oriented architecture (SOA) to solve the challenges of continuous change management, scale, and uniform resource utilization [22, 23]. However, such an adoption itself has costs and overhead associated with it. Existing applications need to be redesigned to work with the service oriented architectures in order to gain any benefits [24]. Therefore, during the transitional period it is common to find hybrid grids that provide certain resources as services whereas the other resources are provided in a more traditional fashion.

Scheduling

The business processes represent workflows that are invoked continuously throughout the application life cycle. The invocation times for workflows are both deterministic and nondeterministic.

Given the business value associated with the workflows, most of the scheduling occurs in predetermined and static fashion. Each workflow is allocated the required set of resources well in advance. The focus of the scheduling is on the optimization of some metric e.g. mean execution time across the several invocations of the workflow. Dedicated and redundant resources are generally used to achieve the desired performance for each workflow [24]. In some cases more

dynamic techniques to workflow scheduling have also been explored and they are described in this section.

As enterprise grids adopt service oriented architectures it is possible to outsource some of the services [25]. In such scenarios various vendors provide the services at varying costs depending on the associated QoS guarantees. Menasce et al. [26] build a theoretical model for the problem of resource allocation to workflow based enterprise applications. An example business workflow based theoretical model is used to define the scheduling/optimization problem. Workflow tasks that require various resource types (compute, data, service, and network) and various instances of each resource type with associated cost are considered in the model. The model is used to define an optimization problem of finding resource allocations that minimizes the overall execution cost while satisfying the execution time requirements. The problem is NP hard and a framework is provided to design solution heuristics. The work differentiates itself from similar problems in the domain of the multiprocessor scheduling by considering resources of multiple types. The heuristics are based on the cost of the resources and favor the least expensive resource first, unless the potential resource violates the maximum execution time. In case of a violation, backtracking is performed to select the next least expensive resource. Given the resource types, backtracking requires removal of one or more resources from the potential resources to avoid the selection of the previously rejected resources. The work is theoretical in nature but considers workflow execution for commercial enterprises. It calculates the optimum resource allocation for a workflow by simulating the workflow execution in advance. Similar to the approach in this work, heuristics are used to make the scheduling decisions. However, the developed framework only pays attention to a single workflow at a time and the underlying grid model is also not service oriented.

Elnikety et al. [27] consider workflows that are invoked via web requests. The workflows are part of a web application that spans multiple resources in the grid. They use the estimates available on the workflow execution time (cost) to perform workflow scheduling. In addition to the scheduling they also use admission control for the outstanding workflow queue in order to keep the system load under a desired threshold. The workflows share the underlying resources and have different execution lengths and resource requirements. Therefore, some scheduling is required for each workflow in order to sustain a reasonable system performance. Admission control is implemented upon request arrival and a workflow is only invoked if the overall system load does not exceed a certain threshold, therefore avoiding the system crash. Once admitted, the workflows are ordered in terms of their sizes, therefore implementing the Shortest Job First (SJF) scheduling heuristic. Experimentation reveals the workflow completion time (response time) to be reduced by a factor of 14 when compared to no scheduling and admission control. Urgaonkar et al. [28] also deal with the similar problem by monitoring a network of queues containing the workflows across the grid. They consider a multi-tier grid configuration where each tier has a corresponding workload queue. A continuous monitoring of the queue is used to establish the resource requirements and the additional resources are provisioned when necessary.

Although both of the previous studies are geared towards E-commerce web sites, they are also relevant for an enterprise grid which is a host to many large scale service oriented web applications.

Darlington et al. [29, 30] identify two categories of workflow scheduling. The first category is based on the real time data such as waiting time in the queue or the shortest remaining execution length. The second category is based on average metrics such as mean arrival time, or mean execution length. They use the second category for workflow scheduling to meet certain QoS

requirements (execution deadlines and corresponding failures) of workflows in a volatile grid. The grid model is chosen to be service-oriented and a central broker (scheduler) performs workflow task scheduling across the grid. Experimental evaluations are performed for the developed scheduling algorithms for a collection of workflows. Various workflow arrival rates are implemented to study the impact on the algorithm performance. The developed algorithms are compared with other traditional algorithms that solve similar problem(s). The evaluation metric is the workflow failure rate due to missed deadlines for different arrival rates and the mean execution times. Their algorithms perform dynamic scheduling and therefore perform better over the static algorithms and perform similar to the other dynamic algorithms. However, as the arrival rates increase the algorithm performance tend to become similar to the other traditional algorithms.

One of the issues in the enterprise grid is of resource silos as described in chapter 1. This occurs due to the dedicated allocation of resources to the workflows, in order to ensure the performance guarantees during the peak time (high load). The resource silos drastically reduce the overall resource usage outside the short length peak time windows. Resource virtualization [31, 32] is becoming a mainstream trend to break such resource silos. A physical resource is virtually divided into one or more logical resources that are exposed as services to the workflows. One issue with virtualized resource sharing among competing workflows is not satisfying QoS for some workflows, as competing workflows have different resource needs. Padala et al. [33] develop a control system that adjusts the resource sharing among applications to ensure the desired QoS and maintains the high resource utilization. The Control system implements a scheduler that is responsible for controlling physical CPU among various virtual machines executing various workflow tasks. The scheduling heuristic is Simple Earliest Deadline

First (SEDF) that distributes the CPU among virtual machines using a weight factor for a fixed time interval. The weight factor is based on the established resource utilization thresholds and ensures certain QoS for applications in terms of their response times. The CPU allocation varies from one time interval to another time interval in order to adapt to the workload on each virtual machine. The workload is estimated by using the actual workload in the last interval. A detailed simulation is conducted in conjunction with the theoretical modeling of the problem in order to provide various benchmarks on the performance of their design under various load conditions. The work does not directly perform any workflow scheduling but schedules CPU among virtual machines executing workflows. It is shown that by adapting to the workload during runtime, establishing resource utilization thresholds, and by prioritizing the applications in terms of their business value, it is possible to achieve the desired QoS.

Another approach towards datacenter resource consolidation is to assign resources based on the historic performance of a workflow. Rolia et al. [34] calculated the task to CPU assignment over historic CPU usage for each workflow. The calculation is done using two techniques of linear programming [35] and genetic programming [36]. All of the servers are consolidated into one large computing grid and available CPUs are then assigned tasks per the calculated assignments. The linear programming turned out to be very compute and time intensive yet it provided better schedules over genetic programming. Both of the algorithms did not generate optimal schedules but delivered reasonable results. In dynamic enterprises it is likely that historic performance varies considerably over time and there is a continuous effort required towards collecting application traces. However for large size enterprise grids running hundreds of applications such efforts result in considerable resource consolidation [37]. A similar approach is described by Rolia et al. [38, 39] for resource management within the enterprise grids. Their

framework utilizes profiles of resource demand for workflows. The profiles are created by the observation of the workflows for a number of weeks. In addition to the resource demand profiles, workflows also have service classes associated with various tasks. Each service class represents a resource access assurance such as Guaranteed, Best Effort, etc. Given the observed resource requirements and access assurances, schedules are created for each workflow so that the minimum amount of resources is allocated to satisfy the access and resource requirements. Admission control and monitoring are applied to ensure that the workflows meet their QoS without consuming unnecessary resources. They [40] also explore WS-Agreement [41] to reserve resources during scheduling.

The boundaries of an enterprise grid vary from one organization to another. In some cases an enterprise grid can be formed by connecting geographically dispersed datacenters together [20]. Wong et al. [42] experiment with the execution of various MPI (Message Passing Interface) applications across a geographic wide enterprise grid. Given that the MPI application tasks communicate with each other, the network and CPU heterogeneity becomes a factor while scheduling MPI applications across geographically distributed grid resources. Their study reveals the viability of the enterprise grid for MPI applications even with the higher costs of communication. The applications in general benefit from the increased number of resources although careful planning is required to ensure the feasibility of an enterprise grid for MPI applications. Jia et al. [43] also consider a geographic distributed enterprise grid as a middleware for E-commerce portals which provide an aggregate of the business processes and intelligence. In order to process the huge amount of data aggregated over multiple large size data sources, they categorize the workflows and grid into three levels of data-flow, mining-flow, and knowledge-flow. The lower level provides services to the higher level workflows and services in

order to allow development of more complex systems. Each level of the grid is data centric where raw data is transformed into useful data during the transition from lower level to the upper level. Each grid level contains dynamically generated workflows that process the data from lower level services and populate the current level data sources. The three logical levels for workflows and the grid are created to separate various business intelligence activities. The motivation is to investigate and provide techniques to dynamically generate and execute workflows to generate business intelligence data for the E-commerce. The work identifies and provides example triggers that can be used to initiate workflow generation and execution. The explicit scheduling of the workflow is not directly discussed and redundancy based techniques are suggested to deal with the workflow task contention to improve the response time. The underlying grid implementation in the architecture is based on Globus [44] toolkit.

Table 3-1. Overview of sample enterprise grid computing systems

Literature	Workflows	Workflow Contention?	Scheduling Approach	SOCC Model
Menasce et al.	Business processes	No	Heuristics – resource cost	No
Elnikety et al.	Web requests	Yes	Heuristics – Small job first	No
Urgaonkar et al.	Web requests	Yes	Approximate –Predictive & On demand	No
Darlington et al.	Scientific & Business processes	Yes	Approximate – Mean execution length	Yes
Padala et al.	Web & Database requests	Yes	Heuristic – Early deadline first	No
Rolia et al.	Business processes	Yes	Approximate – Mathematical programming	No
Wong et al.	Benchmarking applications	No	Optimal – Enumerative	No
Jia et al.	Business intelligence	Yes	Heuristic – Resource redundancy	No

Summary

Enterprise grid computing holds promise for the SOCC model for either individual or competing workflow oriented applications. Table 3-1 summaries the discussion presented in this section for the relevant characteristics to the current research. As evident from the table the related work has some elements common to workflow oriented SOCC computing. However, an explicit SOCC execution model is missing from most of the work. This is due to the costs associated with a transition to the pure SOCC base model. Additionally the definition of software as a service (SaaS) is also open to interpretation and often existing infrastructure components are termed as services although such components may not follow a more standardized definition of the SasS.

Desktop Grid Computing

Definition

A computational grid that is formed over a collection of personal computers is referred to as a desktop grid [45-47]. There are millions of personal computers connected to the Internet that are not continuously being utilized. The collective amount of idle processing time and storage across any large collection of personal computers can surpass the resources available within a super computing environment. Therefore, the motivation behind a desktop based grid is to utilize the aggregated resources (CPU, disk) over the Internet to perform computations that would otherwise require expensive high processing infrastructure. Generally, a computer owner voluntarily participates in desktop grids and does not dedicate the compute resources to the grid. Therefore, a desktop grid lacks any control over the participant computers and must operate in an unobtrusive fashion. Typically desktop grid applications make use of the idle compute resources across the grid in a best case effort. There have been numerous studies done to search/scavenge

or trace the resource availability within desktop grids to efficiently utilize the available resources and add more robustness to the application execution [48, 49].

Service Orientation

Interoperability and more standardized implementations within desktop grids are possible through the service oriented architecture [50]. Initially, desktop grids such as SETI@home [51] were created as proprietary and closed systems. Volunteers wanting to contribute their resources to multiple grid systems need to host multiple components. Given the participation in a desktop grid is on volunteer basis, service based implementation makes it viable for resource owners to host one or more grid services in order to benefit from their contribution [52]. Various efforts have been made to integrate service oriented architecture into the desktop grids [53, 54]. With this integration it is also possible to expose desktop grids to the other grid paradigms such as enterprise grids [55].

Scheduling

Given the large scale and volatility of desktop grids, scheduling has generally been for the independent task applications. However, there is work done towards the large scale desktop grids that is presented in the Theoretical Work section later in this chapter. All of the scheduling references presented here are for the independent task applications.

Jaesun et al. [56] outline the problem of CPU waste when large chunks of work are distributed to the clients by the centralized server. The waste occurs when clients are not able to compute the entire allocated chunk due to their departure from the grid. The distribution of larger chunks is necessary in order to minimize the client-server communication in light of a very large number of clients. As a solution they propose scheduling proxies that are used to distribute work among the cluster of clients. Such a proxy can collect a much larger chunk of work from the server and

then distribute smaller chunks of work among the clients within its cluster. It also monitors the clients for their availability and takes corresponding actions to get the assigned tasks completed. The clients are clustered within according to their network proximity. Event-driven simulation is used to measure the performance of their solution. The underlying work is similar to SETI@home and consists of independent tasks.

SungJin et al. [57] present a theoretical work for dealing with the fault tolerance that is associated with the grid resource failure. Such failures are either due to the departure of the resource from the grid or due to the execution of the non grid tasks on the resource. Such failures can result in a live lock problem which halts the execution of the entire job. The live lock problem occurs as the resource computer continues to execute tasks even in the case of a failure and therefore do not return a failed result to the scheduler. A heart beat based monitoring of the resources is suggested as a solution to detect the task failures for the scheduler to take the corresponding actions. A theoretical proof is given to show that the performance of their proposed fault tolerant scheduling algorithm over the other existing scheduling algorithms.

Anglano et al. [58, 59] also consider resource failure within a desktop grid while performing the task scheduling. Their fault tolerance approach is based on the knowledge that is available about the tasks and the target resources. They conclude that such a knowledge-based scheduling can result in better performance over knowledge-free scheduling. Two task selection policies combined with four resource selection policies are used to perform the scheduling. Tasks are selected based on their execution time residuals, and resource selection is based on their CPU availability and the fault-time distributions. The tasks are independent but submitted as a collection, where the entire collection needs to be executed. Simulation based experimentation is used to evaluate the scheduling policies. The experimentation reveals that the task selection

based on the longest tasks first policy performs better overall. This is due the fact that in the simulated grid environments, executing longest tasks first reduces the overall completion time of the entire collection of the submitted tasks. In addition, resource selection in order of the highest predicted CPU power yields better results over selecting resources with the predicted failure distributions.

HongSoo et al. [60] employ agents to conduct operations in a volunteer computing grid to further improve the performance of the system. The grid resources are characterized in terms of their volatility and durability. Two types of agents are created in the system. The first agent type is responsible for task allocation, load balancing, result collection, and resource monitoring. The second agent type is responsible for the task migration, check pointing, and the replication. The first type of agents performs task scheduling throughout the system and the second type of agents take actions to ensure the task completion. For example, after the initial scheduling, agents perform more aggressive check pointing and task replication for highly volatile resources. The evaluation of the agent based scheduling and execution is performed over the Korea@home system. The scheduling approach is to load balance tasks among resources based on their characterization. With an experimental evaluation, based on the throughput, they achieve a 58% improvement over the traditional First Come First Served (FCFS) based scheduling. Chakravarti et al. [61] also study the agent based task scheduling for an organic grid. The grid operations are handled via agents that have no prior system knowledge and adapt to the environment as the system state evolves. In this work, there are also two types of agents for scheduling and for computation. The scheduling agents are allocated a (logical) tree of resources. The resources closer to the top have higher performance and are preferred for scheduling over the resources at the lower levels. Overtime the tree structure is adjusted as more accurate data about resource

performance becomes available and the high performance resources are moved up. The experimental evaluations are performed to prove the viability of their proposed system. The results are not compared to any other system but the scalability, performance, and fault tolerance of the system is measured and shown to be acceptable for the application in the experimentation.

Baohua et al. [62] consider a data intensive task scheduling within a desktop grid. The challenge with any distributed data intensive execution is the overhead associated with the transfer of data to the target resource. They compare FTP and BitTorrent [63] data/file transfer protocols for load conditions under which each protocol performs better over the other. Three scheduling heuristics based on the order of task completion times are implemented with the corresponding file transfer protocols to outline the improvements over the basic RoundRobin heuristic.

Table 3-2. Overview of sample desktop grid computing systems

Literature	Applications	Scheduling Approach	SOCC Model
Jaesun et al.	Non workflow & independent task applications.	Physically distributed	No strict SOCC model, however each desktop can be abstracted as a service instance within the overall grid therefore, exhibiting the SOCC model.
SungJin et al.		Adaptive for fault tolerance	
Anglano et al.		Execution time residuals	
HongSoo et al.		Physically distributed	
Chakravarti et al.		Adaptive & physically distributed	
Baohua et al.		Heuristics – Completion times	
Kondo et al.		Heuristics – Resource performance & task replication	

Kondo et al. [64] study task execution over a desktop grid when there are more resources than the number of tasks. Given the abundance of resource there is an opportunity to consider task scheduling policies that are alternative to the simple FIFO based scheduling. The FIFO is deemed to be reasonable scheduling policy under the scenarios of resource contention. They apply three scheduling policies of Resource Prioritization, Resource Exclusion, and Task Replication to obtain the near optimal results for the tasks on hand. They observe that resource

exclusion based on task makespan with task replication is proven to be the best scheduling policy where the results come within a factor 1.7 of the optimal solution. Their simulation is trace driven from a commercial desktop grid Entropia.

Summary

In desktop grid computing systems, generally a large size of independent task applications have been considered due to the lack of complexities associated with the workflow execution. The selected sample work in this section towards the desktop grids which is relevant to my research is summarized in table 3-2. The scheduling within this domain is for the system throughput which attempts to execute maximum number of tasks possible. Although, no SOCC execution model is present, the individual desktop when exposed as a service can provide more interoperability among the competing applications.

Scientific Grid Computing

Definition

In academic and research organizations, computing grids are implemented to create high processing computing (HPC) environments. Such computing grids are formed with the collaboration of one or more organizations by networking their high processing compute resources such as clusters, storage area networks, server farms, and supercomputers. Therefore, a scientific grid operates within multiple administrative domains. The primary motivation behind such grids is to assemble computing power that otherwise would be impossible or cost prohibitive. One example of such a scientific grid is GriPhyN [65]. The target applications of such computing grids are resource intensive and research oriented. There are many types of applications being executed within such grids including workflow based applications [66, 67].

Service Orientation

The scientific grid community has adopted the SOCC implementation after the emergence of web services. In addition there have been efforts to standardize the SOCC implementations. One such effort that has gained popularity is the Open Grid Service Architecture (OGSA) [68]. Services are viewed as constructs to create virtual organizations (computing grids) by exposing the heterogeneous resources of the independent physical organizations. In particular a notion of Grid Service has been introduced to provide grid infrastructure components in a well defined fashion [5]. That is, in addition to the standardized interfaces, services are also assigned specific semantics. The semantics are used to describe the service behavior that is to be consistent across each implementation of a service. Similar to the other service oriented architectures, OGSA proposes hierarchical organization of the grid services. The core services are at the lower layers (infrastructure), which are then utilized by the domain specific higher layer services. An example of the core services include resource selection, scheduling, secure execution, data management, authentication, and fault recovery [69].

Scheduling

Application scheduling within scientific or research oriented grid computing is a well studied research area. Workflow based applications are a subset of the overall set of grid applications. Given the complexity associated with the workflows, their scheduling has also received considerable attention. Scheduling techniques roughly fall into three categories namely List Heuristics, Duplication, and Clustering [15]. The workflow task dependencies require scheduling in such a fashion that the overall execution length (makespan) of the workflow is minimized. Within List Heuristics, tasks are queued according to their execution order (task dependencies) and selected for scheduling using some priority. Scheduling with Duplication attempts to duplicate predecessors tasks in such a way that overall minimizes the workflow completion time.

Clustering based scheduling is designed to minimize the workflow makespan by reducing the communication overhead among tasks executing on different resources. Tasks requiring communication with each other are clustered or scheduled on the same resource. Yu et al. [70] perform a detailed survey of the workflow management systems and present a taxonomy to categorize various systems. Some of their work is summarized here due to its relevancy to the current research.

Condor [71], a compute-intensive grid system, provides a mechanism called DAGMan [72] to schedule workflows within the Condor enabled grid. DAGMan is a meta-scheduler that manages the workflow tasks dependencies. Its name is derived from the structure of the workflows (Directed Acyclic Graph -- DAG) that it schedules. DAGMan relies on the workflow description to submit tasks in a predefined order. However, DAGMan is limited in terms of the task dependencies. In particular it is unable to handle branching or looping. The actual scheduling of the tasks is done by the Condor scheduler.

Pegasus [73] is a workflow management component within the GriPhyN grid [65] and it performs scheduling of the abstract workflows onto the grid resources. Abstract workflows do not describe specific grid resource requirements and are generally used to increase the portability of a workflow. At runtime suitable mechanisms are used to find the actual resource mappings for the abstract workflow tasks. Pegasus uses scheduling policies of random resource selection, and of performance prediction. The performance prediction is based on the historic data that is being stored for every workflow during its execution. Pegasus converts the actual scheduling of the tasks into Condor jobs, which are then executed by the DAGMan.

Taverna [74] is a workflow management system for bioinformatics workflows. It operates within the myGrid [75] system which is a collaboration of many European research

organizations. Taverna provides its own workflow specification language to describe the workflows in terms of their input, output, data, and control flow. It also provides a user interface to manipulate workflows and select available resources for workflow tasks and users can view the workflow progress through the interface. The myGrid is a service oriented grid and offers several types of infrastructure services to the workflow management system of Taverna.

GrADS [76] is a grid system targeted towards scientific users without programming background within organizations across North America. The overall scheduling objective within GrADS is to minimize the workflow makespan. The grid resources are ranked in terms of the performance corresponding to the workflow tasks on hand. The performance is based on the execution cost plus the data movement costs to the particular resource. The resource performance ranks are then used by various scheduling heuristics to map tasks to the resources. The implemented scheduling heuristics are based on the order of task completion times. Further, GrADS continuously monitors workflow execution and performs rescheduling when a certain performance drops from the predetermined threshold. Rescheduling is either done by stop/restart or by moving tasks to the better resources if available, while considering the costs associated with the data movement.

GridFlow [77] organizes the grid into three logical layers; resource, local, and global grid. Local grid represents resources within one organization, whereas the global grid contains all of the local grids. It also provides a user interface to compose workflows. Workflows contain sub-workflows that are mapped to the local grids. Initially workflow execution is simulated to find the optimal schedule and then the actual scheduling occurs where local grids are mapped to the sub-workflows. The underlying grid system is agent based where each agent controls a local

grid. GridFlow also employs heuristic algorithms to minimize the workflow makespan and resource idle time.

Gridbus [78] is a service-oriented grid that models itself after economic principles. Grid resource management is performed by assessing supply and demand of the grid resources. The grid supports parameter sweep based workflows with the QoS specifications. The specification is expressed via a proprietary XML based language. The scheduling decisions are driven by events that occur in a tuple-space. In addition, just in time scheduling and rescheduling for failed tasks is also performed. The major difference of the Gridbus approach from other contemporary grids is the use of market-based principles to makes the scheduling decisions.

Table 3-3. Overview of sample scientific grid computing systems

Literature	Workflows	Workflow Contention	Scheduling Approach	SOCC Model
Condor	All workflows represent scientific & research applications.	All of the workflows are executed independent of each other.	Performance driven	Possible
Pegasus			Random & performance prediction	No
Taverna			Performance driven	Yes
GrADS			Performance driven heuristics	No
GridFlow			Performance driven heuristics	No
Gridbus			Market driven	Yes
ASKALON			Performance driven heuristics	Possible

ASKALON [79] provides a generic workflow development and execution framework for transparent access to a grid. The motivation behind such an access is to free the application developers from the grid level details. The transparent access is achieved via services such as scheduler, execution engine, resource manager, and performance predictions. These services are

built on top of the Globus grid framework. An XML based language is provided to develop the workflows. These workflows are then transformed into Directed Acyclic Graphs (DAGs) by loop unrolling (see chapter 5) . The DAGs are then scheduled across the grid using DAG based heuristics. The Workflows are large scale scientific applications and the goal is to study the impact of each scheduling heuristic on various workflows. Therefore, the focus of the study is on the individual workflows and their performance optimization as opposed to the collection of workflows. The study provides a good example where workflows are unrolled and transformed into large scale DAGs.

There are fuzzy boundaries between scientific and enterprise grids. Some of such differences are outlined in [18, 20, 21].

Summary

The relevant work presented in this section is summarized in table 3-3. Given that workflows in this domain represent the research applications, they are executed in isolation of other workflows and therefore workflow contention is not an issue. As seen in table 3-3, the individual workflow performance is the focus of the scheduling. In some instances, the workflow execution is performed in SOCC based environment.

Workflow Description Languages

As the name suggests, expression languages are used to describe the workflow structure, their task dependencies and the other execution requirements that are useful in workflow execution automation. It also allows for a standardization of workflow execution within a computing cloud. Most of these languages are XML based and do not always consider service oriented workflow execution. There have been quite a few workflow languages proposed over the years, however, a standard language has yet to emerge.

One of the commonly used expression languages is the Business Process Execution language for Web Services (BPEL4WS) [80]. It is built by combining some of the elements from two other expression languages of Web Services Flow Language (WSFL) [81] and XLANG [82]. It allows the description of workflow execution in both an abstract and actual manner. At the abstract level not all of the workflow execution details are specified and the execution transformation to the actual behavior is left up to the system. BPEL4WS defines interaction behaviors of various entities (workflow tasks) in terms of web services based message exchange.

Another expression language that provides an abstract description of workflow execution is the Abstract Grid Workflow Language (AGWL) [83]. The motivation behind an abstract description is to save the user from the concrete implementation details of the system. The workflow execution is described as a graph of activities that are mapped to the resources in the grid. Such a graph also describes data and control flow for the workflow execution. It supports parallel task execution including parallel loops and has data access constructs available. Such a data access mechanism allows more flexibility to the workflows that have sophisticated data needs. The limitation of any abstract language is difficulty in transformation from abstract to actual execution description. Such a transformation is left up to the system developer and will vary from one implementation to another.

Table 3-4. Characteristics of sample workflow description languages

Language	Description	Approach
BPEL4WS	Abstract/Concrete	Task execution in terms of web services calls.
AGWL	Abstract	Graph based task invocation order.
DAML-S	Concrete	Calls to semantic web services.
GSFL	Concrete	P2P oriented service calls.

In order to avoid some of the limitations with abstract workflow description, DAML-S [84] brings semantics to the service execution. By associating semantics to the service calls, it is possible to provide more consistent system behavior. However, an auto-generated implementation of the service methods to satisfy the semantics is the best way to avoid human error and interpretation of the semantics.

Another commonly known workflow language specifically designed for the grid services is the Grid Service Flow Language (GSFL) [85]. Its motivation is to provide a language that supports peer-to-peer interaction among the services for data exchange without going through an execution engine intermediary. A larger amount of data is expected to exchange hands among grid services therefore any central control is seen as a bottleneck. Further it also focuses on providing more flexibility in workflow task execution order. Certain task or corresponding services may not always be invoked given the different workflow instances and the time windows associated with the task executions.

Summary

An overview of the sample workflow description languages is shown in table 3-4. The workflow description can be in abstract terms where the implementation specific details are left out to provide more flexibility and broader coverage of the language. Each language has its own way of describing the workflow task execution.

Composite Web Services

There is also a trend [86-91] to execute workflows over non-cloud collections of services and it is generally referred to as composite web services. One or more web services are pooled together and invoked in a certain order to execute the workflow. Therefore, each composition is specific for a certain workflow. Composite web service workflow execution can be considered as an execution unit of an SOCC execution model, as shown in the figure 3-1. That is an SOCC is a

collection of composite web services. The scheduling decision specific to the unit can be considered as a local decision whereas a cloud scheduler is considered to be making global scheduling decisions. In our work no local scheduling is employed and all of the decisions are made globally in a centralized fashion.

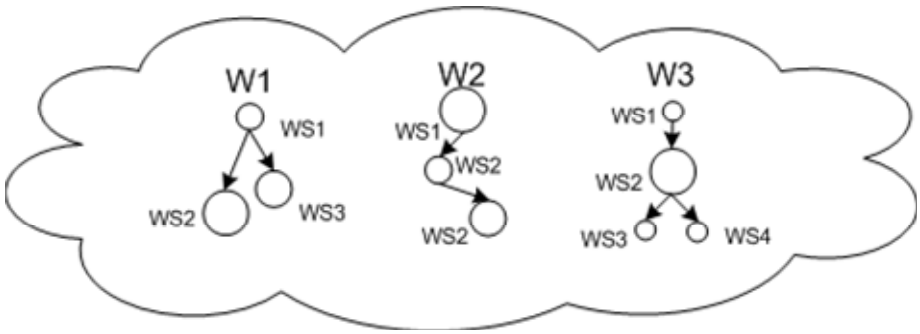

Figure 3-1. Composite services as an execution unit for a SOCC

Scheduling in composite workflows is generally focused on the optimization of multiple invocations of the workflow. Redundancy is a common technique that is used to provide improved performance and therefore, workflow scheduling concerns itself with the selection of a service instances that can meet certain QoS objectives. Response time is another metric that is generally improved with the service scheduling.

The work of Chafle et al. [92] is about decentralized orchestration of the composite web services. Orchestration refers to the act of invocation control within the web services. A centralized orchestration is generally done via a controller that can cause bottlenecks and some unnecessary data transfer overhead. With decentralized orchestration it is possible to avoid pitfalls of the centralized orchestration but it requires much more thought into design. Various design issues such as distributed error recovery and fault handling, communication overhead, and code partitioning are explored. A proof of concept implementation is provided to validate proposed design choices. The underlying composition forms a single workflow that computes a

route between origin and the destination. Scheduling is not a concern and web services calls are predetermined.

Table 3-5. Overview of sample composite web services systems

Literature	Scheduling Organization	Scheduling Approach
Chafle et al.	Decentralized	Predetermined dedicated resources
Zhang et al.	Centralized	Best Expected Execution Time

Zhang et al. [93] consider web services and grid convergence and propose a composition and scheduling architecture for the grid based web services. The composition is for the selection of a set of web services for the workflow execution. The scheduling is responsible for determining the task readiness during workflow execution and mapping the tasks to the services. The scheduling is done by matching task parameters with the resource performance. Both the composition of services and the task assignment are only for one workflow at a time. Simulation based experiments over Globus Toolkit based prototype grid is used to show the performance of their algorithms.

Summary

A composite web service (workflow) is an execution unit of the SOCC execution model. The scheduling within composite services is to optimize the multiple invocations of the single workflow and is considered local scheduling.

Theoretical Work

There has been considerable theoretical work towards DAG scheduling that is relevant to the workflow scheduling given the DAG structure of the workflow. A sample [94-96] of such work considers cloud based resources for workflow scheduling and explores scheduling approaches for various types of workflows. The workflow types are defined in terms of the workflow

structure. The workflow structure consists of many levels (see chapter 5), where each level contains tasks that can be executed in parallel. The subsequent level contains the (independent) children of the parallel parent tasks. The schedules are evaluated in terms of the number of ready tasks during each scheduling step. The maximum number of ready tasks makes it possible to minimize the makespan of the entire workflow.

Rosenberg et al. [94] first considered mesh structured workflows and theoretically proved that scheduling workflow tasks in order of their dependencies results in an optimal schedule. All of the independent tasks are at the same level and are scheduled lexicographically. The direction of task scheduling is termed as parent-oriented. That is all of the parents are executed before any of the children can be executed. Although, the selected scheduling heuristic of parent-orientation to maximize the number of eligible tasks (hence optimal) seems intuitive, it is shown that other intuitive approaches fail to produce any optimal results.

Secondly, Rosenberg et al. [95] expand their study to consider more complex workflows such as Reduction-Mesh (RM), Reduction-Tree (RT), and Fast Fourier Transformation (FFT). The scheduling evaluation criterion is extended to include minimum memory requirements in addition to the maximum number of ready tasks. The basis of the schedules developed for all three workflow types is the parent-orientation, as previously described. Such an intuitive scheduling approach previously proved to be necessary and sufficient to produce optimal results when optimality criteria only included maximization of the eligible tasks. However, parent-orientation is not sufficient when optimality criteria include minimum memory requirements as well. The study develops schedules that are optimal for maximizing eligible tasks for all three DAG types. The developed schedule for reduction-mesh DAG is also optimal for the memory requirements. The schedules for reduction-tree and FFT DAGs are approximately optimal for the

memory requirements. Additionally, it is mathematically proven [95] that any other intuitive scheduling scheme which may result in maximum eligible tasks cannot produce memory efficient schedules. On the other hand schedules that are memory efficient fail to maximize the eligible tasks. All three schedules define a particular task execution order for the consecutive scheduling steps at each diagonal level of a DAG. Together with the parent-orientation such execution orders result in the optimal results.

Table 3-6. Overview of sample theoretical research towards workflow scheduling

Literature	Workflow Structure	Scheduling Approach
Rosenberg et al-1	Mesh structured graphs	Parent Oriented & Lexicographically
Rosenberg et al-2	Reduction-Mesh (RM), -Tree (RT), Fast Fourier Transformation (FFT)	Task execution order based on the workflow structure
Rosenberg et al-3	Composites of RM, RT, FFT	Priority order

Finally Rosenberg et al. provided a framework of scheduling a large set of complex structured workflows [96]. The basis of this work is optimal schedules developed in the last two described studies. They use the previously defined workflow (DAG) types as the building blocks of much larger and complex workflows. Given the optimal schedule for each building block is known, the optimal schedules for composite workflow is obtained by identifying a priority order among the building blocks. Once the workflows are scheduled in order of the priority order of their building blocks the resulting schedule is proven to be optimal. In addition to the build block composition, the study also identifies the more important aspect which is the decomposition of the workflows into its building blocks so the optimal schedules can be identified.

Three studies together strive to provide theoretical principles towards cloud workflow scheduling. The results found in this work [94-96] are universal in nature and are applicable across many flavors of cloud computing systems.

Summary

The theoretical work presented in this section is summarized in the table 3-6. Various DAG based workflow structures are studied for their suitable scheduling policies so that the overall execution time is minimized. The work explores various task execution orders that generate maximum number of ready tasks possible and therefore, minimizing the total execution time for any workflow.

Conclusions

Given the above described literature review, various implementation and usage scenarios are seen for the following three elements of cloud computing.

- Grid computing
- Service Oriented Architecture
- Workflows based applications

There is a large amount of work towards grid computing which has been classified into three different categories of enterprise, desktop, and scientific grid computing. This categorization is based on the target execution environment setup and on the structure of the applications being considered for execution. Since a computing cloud is backed by a grid, the motivation behind this categorization is to differentiate between the purpose and objectives of various grid implementations, in order to better extract the relevant research findings for the this research. A commentary towards the Service Orientation (SO) within each category is provided to survey the current state and challenges of the Service Oriented Computing Cloud (SOCC) adoption. In addition the survey also touches on the structure of the applications and identifies workflow and non workflow based applications.

Workflow scheduling for SO environments in isolation from the grid computing is also presented in conjunction with the theoretical scheduling and description of the workflow scheduling.

In conclusion the key findings of the literature review are:

Service Orientation (SO) is in its early stages of adoption within the grid/cloud community. This is partly due to the fact that the cloud computing in itself is not a mature implementation methodology. Work continues towards the implementation challenges of a cloud and some of the work has started to look at SO to solve some of the challenges. Both SO and the grid computing emerged around the same time and are complimentary methodologies. However, their integration results in limited and constrained execution environments that cannot be used as a general purpose computing platform to fully gain the benefits associated with the SOCC. The limitations arise due to the infant nature of both methodologies. Efforts such as OGSA [68] are work towards the generalization and standardization of the SOCC based implementation.

One of the SOCC adoption challenges is migration/conversion of the existing systems and applications to make use of the SOCC. In particular some legacy application refactoring needs to occur before it can be executed in terms of services. One approach to avoid large refactoring of the application has been to wrap the existing application within a service that can then be invoked as a service. However, this approach is limiting in a sense that the application is bound to a single service for all of its functionality.

Another ongoing challenge with SOCC is the behavior of the grid services. It is possible to have multiple implementations of the same service with different behavior making it difficult to ensure the QoS associated with the service invocation.

Workflow based SO also has some challenges in terms of their description/representation, application development, and invocation/execution over distributed services. Some efforts such as [80-85] are underway to address some of these challenges by standardization.

Interoperability among SOCC systems and support for general purpose workflow execution environments is a work in progress. One outstanding item is to consider multiple workflow execution within an SOCC where workflows (and their tasks) are competing for resources.

CHAPTER 4
RESEARCH GOALS

As described in the last chapter, cloud computing borrows from many similar domains including the independent task CPU cycle harvesting based desktop computing, the enterprise datacenter grids and other extreme of the research oriented large scale computing environments. Various degrees of service orientation are present within the various modalities of the grid environments which form the cloud backbone. Going forward, more and more of the grid systems will adopt services as their construct to offer their resources, therefore aligning themselves closely with the SOCC. Workflows have often been one of the client applications of the grid environments and their execution has been explored in much detail. Expression languages to describe workflows, theoretical work towards optimal workflow scheduling and optimization of the multiple invocations of the isolated workflows are some of the efforts to execute the workflows within grids.

In order to further enrich the domain of workflow execution within a SOCC, this thesis considers competing workflow scheduling (and execution) over a pure SOCC execution model. Every cloud resource is considered to be exposed as a service and more than one workflows are competing for the services during their execution.

The problem statement formed in chapter 2 is to serve as a tool while investigating

- Workflow oriented application scheduling
- Resource competing collection of independent workflows requiring execution
- Pure service oriented computing cloud as an execution platform for the competing workflows

The motivation behind this research is to understand issues while considering SOCC as a general purpose computing platform for a collection of workflows. Both the workflow performance and the system throughput are used to evaluate various scheduling techniques.

CHAPTER 5
WORKFLOWS & SCHEDULING

A workflow is a collection of tasks connected together in a certain order for their execution. In this thesis workflows are modeled as Directed Acyclic Graphs (DAGs). It is possible to model workflows in a variety of methods including with the description languages such as BPEL4WS [80], WSFL [81], AGWL [83], GSFL [85] , previously described in chapter 3. However, DAG based modeling is a common technique that has been well studied [97]. Each node in a DAG represents a workflow task and directed links indicate the task dependencies.

In a DAG task dependencies are indicated with the unidirectional links between the tasks. The task originating the dependency is the parent task and the task depending on the parent task is the child task. Every task has a parent except for the root task that is the first task in the execution order. Figure 5-1 illustrates an example DAG structured workflow.

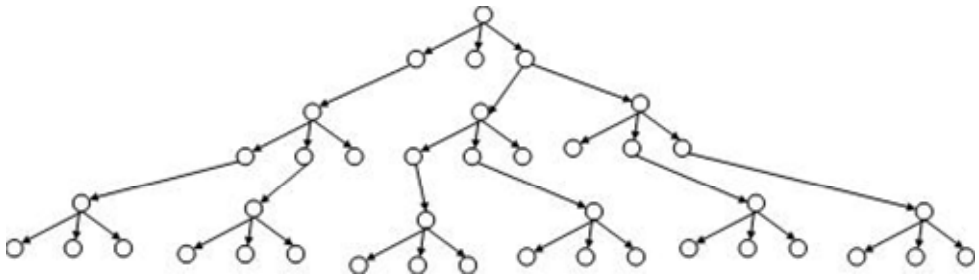

Figure 5-1. An example DAG structured workflow

Workflows with only one parent per child are considered. Although, it is possible to have multiple parents for a child, a workflow can be transformed such that each child only has one parent to simplify its structure. Such a workflow transformation creates multiple instances of a

child task for each parent, therefore, resulting in a single parent-child relationship. Figure 5-2 shows such a transformation for a simple workflow.

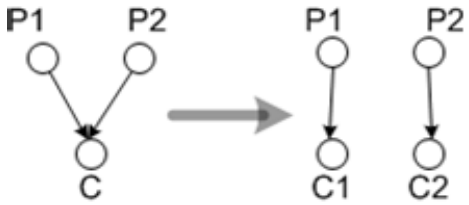

Figure 5-2. Workflow transformation for a single parent per child

Additionally, it is possible that workflows contain loops. Such loops are generally represented in a DAG by performing the loop unrolling [97, 98]. One such simplified loop unrolling is shown in figure 5-3. In this example a simple workflow repeats task C three times and therefore contains a loop. Such a loop is then unrolled by creating C1, C2, and C3 tasks which represents unique invocations of the task C.

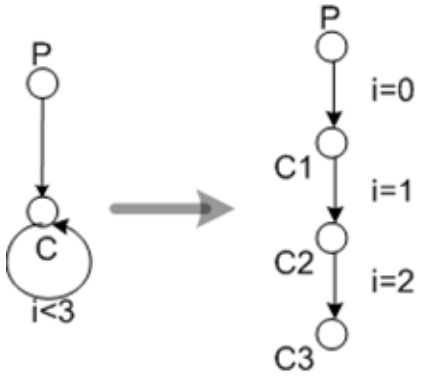

Figure 5-3. Workflow (DAG) loop unrolling

Tasks in a workflow can also have sibling tasks. One way to define sibling relationship is to consider task execution. All of the tasks that can be executed in parallel are sibling to each other. It is likely that not all of the siblings can start and finish execution at the same time depending on their readiness for execution.

Task Readiness

Task readiness for execution can be defined in two ways. One way a task becomes ready for execution is when its parent completes execution. The second way a child can become ready is when its parent and all the parent's siblings complete execution. The second definition of a child readiness is more restrictive. Tasks become ready at a slower rate, however, more tasks become ready at the same time. This also introduces a notion of workflow levels or steps for its execution. A workflow level contains tasks that become ready at the same time. All of the tasks within a level are sibling of each other. Figure 5-4 shows a workflow with four levels.

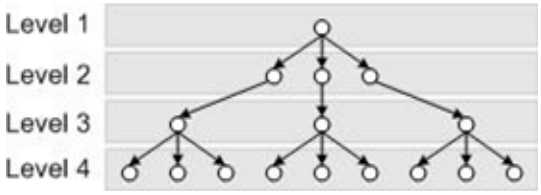

Figure 5-4. A workflow with four levels

Size

Workflow size is measured as the sum of all of its task lengths. A task length indicates the number of units to complete the execution of a task. A simple workflow with three tasks of length 10, 20, and 25 has a size of 55. Workflow size indicates total units required to complete its execution. Figure 5-5 shows an example workflow with its task sizes.

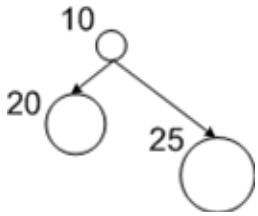

Figure 5-5. An example workflow of size 55 (10 + 20 + 25)

Depth

For a workflow without discrete levels or execution steps, the depth is simply the number of tasks along the longest execution path starting at the root task. An execution path is a chain of tasks linked together in order of their dependencies. Figure 5-6 shows an example longest execution path. For workflows with levels, the depth is equal to number of levels as seen in figure 5-4.

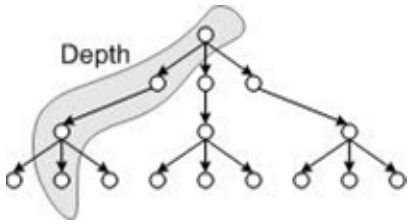

Figure 5-6. Example workflow depth

Breadth

The maximum numbers of tasks in a workflow requiring parallel execution at any time measures the breadth of the workflow. This measurement is useful when determining the number of services for a workflow. The breadth of the workflow can vary depending on the execution constraints placed on tasks. For example, when tasks execute irrespective of their siblings the breadth of the workflow is the maximum number of tasks that can become eligible for execution at anytime. In other cases, when tasks must execute in conjunction with their siblings, the breadth is the maximum number of siblings in a workflow. Figure 5-7 shows an example workflow with a breadth of six tasks.

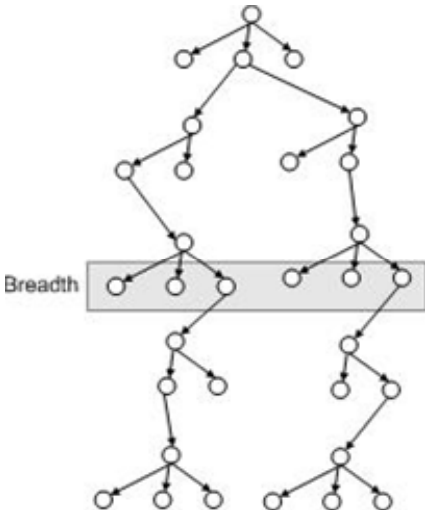

Figure 5-7. An example workflow breadth of six tasks

Arrival Time

The time at which workflows arrive in the system can be modeled to have various random distributions such as exponential, poison, uniform, etc. Chapter 6 provides more details on the implemented arrival time distributions.

Scheduling

Given the above definition of the workflow, the scheduling activity becomes a routine of allocating outstanding workflow tasks to the available services in the system. One or more queues can be implemented to track outstanding tasks of all the workflows, which are then allocated to the services using some order as the services become available. Since all of the workflows are allocated services from a same pool, they are then competing for the services within the cloud.

Figure 5-8 provides an overview example of the scheduling activity as described in this research. There are three workflows W1, W2, and W3 with ten tasks all together competing for three services S1, S2, and S3 in the example. Workflow arrival into the system is over time and workflow tasks become ready for scheduling only after their parent tasks have completed the

execution. In this example, we assume that workflow arrival is depicted from left to right, that is, W1 arrives first, followed by W2 and then W3.

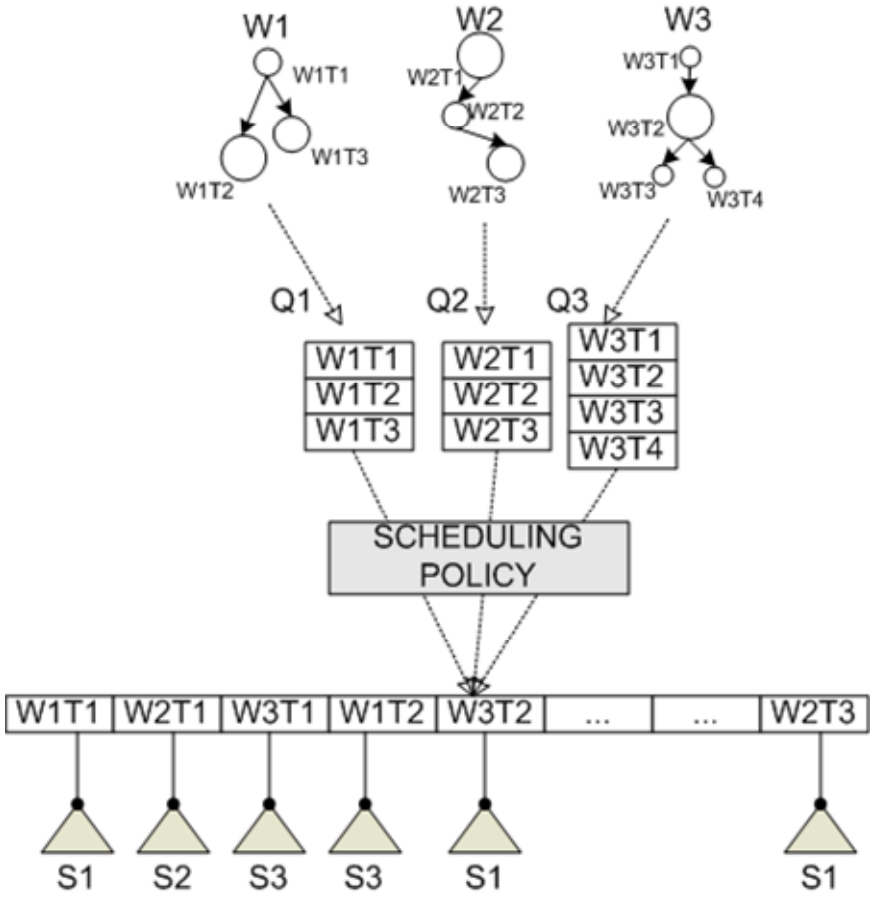

Figure 5-8. Overview of scheduling workflow collections over SOCC

The scheduler has individual queues (Q1, Q2, and Q3) for each workflow, where each queue is sorted in the order of task dependencies. These queues are loaded upon the workflow arrival into the system. Using the workflow queues, the scheduler then applies a certain scheduling policy to create the master queue for the ready tasks. The master queue is dynamic throughout the system life cycle and is always ordered by the scheduling policy. The available services are allocated to the tasks from the left to the right until all of the services are allocated or there are no more unscheduled tasks in the master queue. In figure 5-8, three tasks W1T1, W2T2, W3T3 are first allocated to the services S1, S2, and S3. After service S3 becomes available when the task

W3T3 finishes its execution, it is the allocated to the next task W1T2 in the queue. Next the service S1 becomes available and is allocated to the next task W3T2 in the queue and so on.

CHAPTER 6
EXPERIMENTATION

In order to study the scheduling of workflow collections over SOCC, simulation based experimentation is conducted. The simulation implements various scheduling scenarios that are necessary to complete the research presented in this thesis. In this chapter, experimentation setup and design details are provided to aide with the discussion of the results in the next chapter.

Simulation Methodology

A random number generation is used to prepare the input data for the simulation. Once generated, the input data is saved and used across various experimentation runs. For example the workflows are generated using various random numbers for the number of tasks, the task dependencies, and the workflow arrival times. During experimentation, the same set of workflows is used while simulating each of the scheduling heuristics for a given cloud size and the workflow arrival pattern. The consistent input data makes it possible to achieve the same output when repeating a particular experiment. The repetition is often necessary to ensure the proper simulation behavior due to software bugs.

The experiment maintains its own time clock to simulate the parallelism of the cloud. For example, to simulate the parallel execution of the scheduled tasks, the services are sequentially executed between the two clock ticks.

Cloud Modeling

The simulation models an SOCC environment to execute the various scheduling scenarios. There are three main components of the modeled cloud including the services, workflows, and the scheduler.

Services

The workflow tasks are executed by invoking the services after the scheduler assigns tasks to the available services. Only one task is assigned to a service at any time, and therefore during task execution that service is not available for scheduling. All of the services in the cloud are modeled to be homogeneous for simplicity. A cloud with multiple service types partitions the scheduling activity to pools of various service types.

Task execution by a service is modeled via a stack. Each level in a stack represents an operational unit. Therefore, at a simulation step, task execution is a computation of a single operational unit.

The services are made available through a directory. However, due to the homogeneous nature of the services no service discovery and description is implemented.

The number of services in a cloud is the cloud size or the cloud capacity. Various number of cloud sizes are used to study their affect on the workflow scheduling performance.

Workflows

A complete description of the workflow and definition of its attributes is provided in chapter 5 and is referenced in this section.

All of the workflows are generated in advance using various random number seeds. The seeds are used for the number of workflow tasks, size of each task, the workflow breadth, and the number of children for each task (task dependencies).

There are two types of workflows that are generated for the experimentation. The first type is called the Evolving DAG (EDAG) and the second type is called the Constant DAG (CDAG). Figure 6-1 shows an example of both workflow types. The EDAG workflows are evolving in terms of the number of tasks at each level, whereas the CDAG workflows do not increase the

number of tasks at each level beyond a fixed value and therefore, have somewhat constant structure.

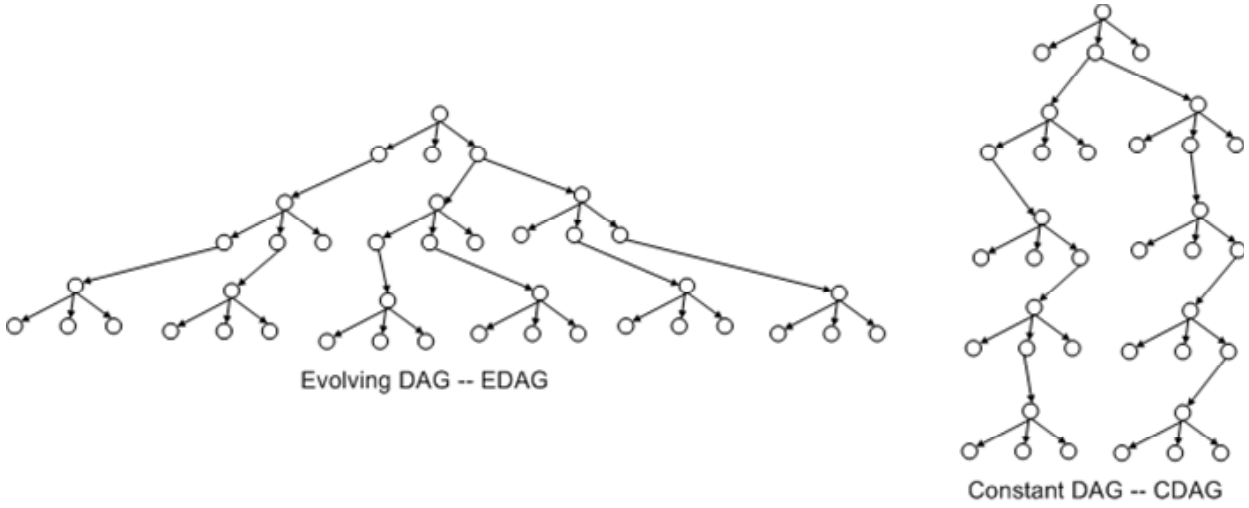

Figure 6-1. An example of EDAG & CDAG workflow types for the experimentation

The two workflow types are used to model the applications of different resources requirements. The EDAG workflows have an increasing number of resource requirements as their execution proceeds. The CDAG workflows always keep the resource requirement under a fixed value.

The workflows are modeled to arrive randomly in the cloud for the execution. Two such distributions are used to model three arrival time patterns during various experiments. The first pattern is called continuous, which is a uniform distribution, where single workflows arrive at fixed intervals. The second pattern is also a uniform distribution, but a fixed number (more than one) of workflows arrive at fixed intervals. This pattern simulates workflow arrival in bursts. The third pattern follows an exponential distribution. Figure 6-2 shows an example of these three patterns for the arrival time of 25 workflows.

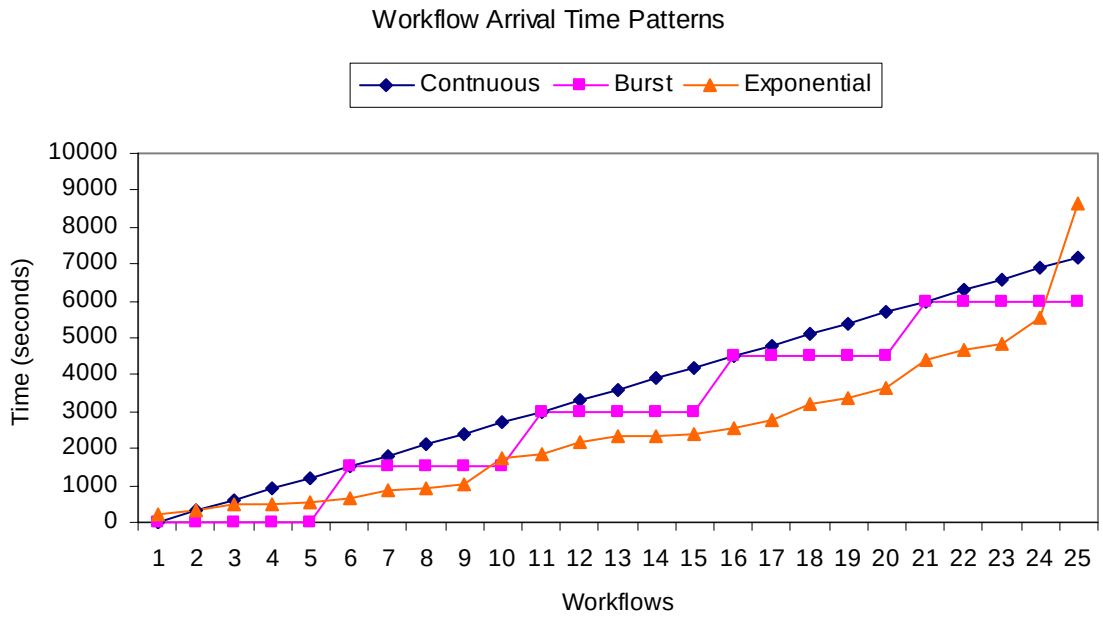

Figure 6-2. Example arrival time distributions for 25 workflows

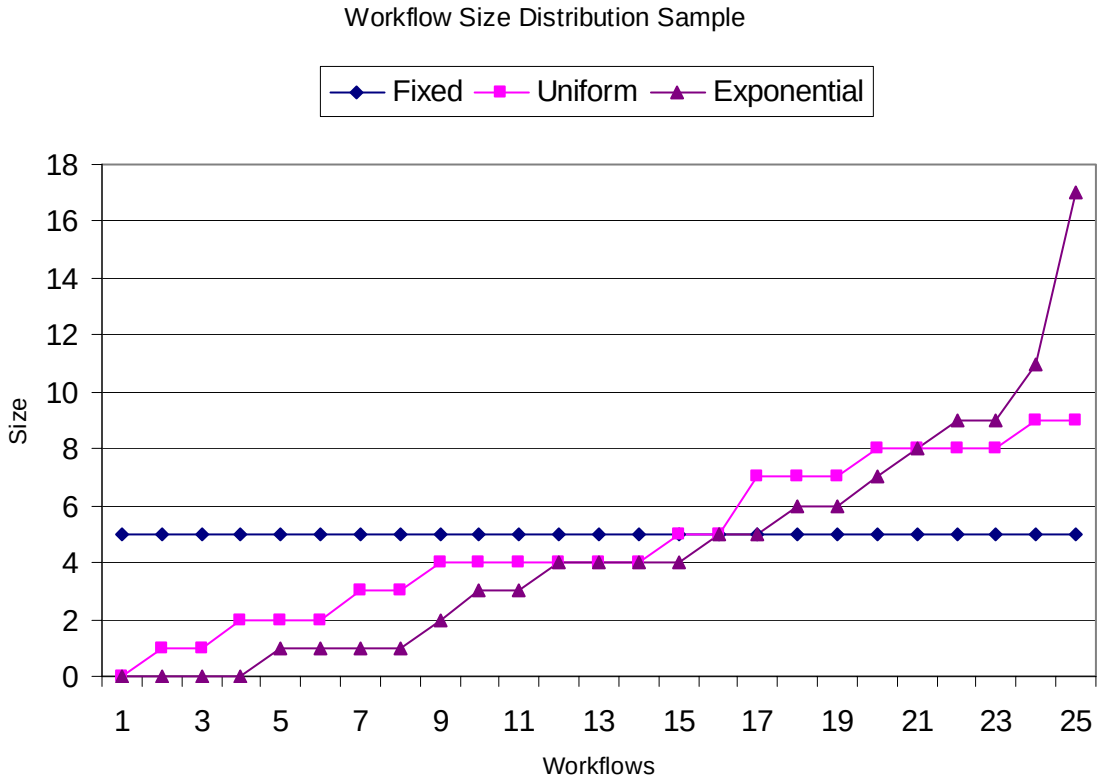

Figure 6-3. Example size distributions for 25 workflows

The figure 6-3 shows the example size distributions for a sample of workflows for illustration purposes. Three size distributions are implemented to see their impact on the scheduling policies.

Scheduler

Three implementations of the scheduler are available in the simulator to execute any of the two scheduling policies and their corresponding heuristics. The basic implementation of any scheduler is queue based. Scheduler maintains a queue to track the workflow arrival into the system. In addition, various queues are maintained to track the execution of each workflow. Scheduling is performed over a centralized scheduling queue that contains all of the ready tasks in the system belonging to the various workflows. The scheduling queue is sorted as per the scheduling policy and its corresponding heuristics. For example, for the scheduling policy of prioritization and the heuristic of shortest task, the scheduling queue sorts all of the tasks in order of their lengths regardless of their workflow association.

Measurements

The following measurements are tracked within all of the simulated experiments:

Wait Time is the time a workflow waits in the system after its arrival and before it is successfully allocated at least one service to start its execution. It is calculated by the following equation

$$\text{Wait Time} = \text{Execution Start Time} - \text{Arrival Time}$$

Execute Time is simply a difference between the workflow execution end time and the start time. It represents the total time a workflow spends executing and is indicative of workflow performance. It is calculated by the following equation

$$\text{Execute Time} = \text{End Time} - \text{Start Time}$$

Finish Time is a time at which a workflow finishes execution and exits the system. It is the sum of the workflow wait and the execute time. A workflow may or may not successfully

complete its execution when it is finished due to the associated deadline. The workflow finish time is calculated by the following equation

$$\text{Finish Time} = \text{Wait time} + \text{Execute Time}$$

Total Finish Time is the sum of all the workflow finish times in the system. This measurement indicates the overall cloud performance for a particular set of workflows.

Average Finish Time is the average of the finish time across all of the workflows belonging to a particular set. This measurement is useful in developing some idea of the individual workflow performance in a given cloud configuration.

Scheduling Policies

As previously described in chapter 2, two scheduling policies of workflow prioritization and service partitioning are implemented.

Workflow Prioritization. Under this scheduling policy the scheduler sorts the workflows and tasks using a certain priority while selecting the workflows and tasks for service allocations. There are eight different heuristics that are used for prioritization and these heuristics are outlined in the table 6-1 and 6-2.

The heuristics in table 6-1 are based on the size attribute, which is the indication of the execution lengths. Both the individual task sizes and the collective workflow sizes are used to formulate the four heuristics in table 6-1. The heuristics of ST and LT focus on the size of the ready tasks regardless of which workflows the tasks may belong to. With these two heuristics, it is possible that at the end of a scheduling step, ready tasks belonging to the different workflows are scheduled and multiple workflows are being scheduled concurrently. The heuristics of SW and LW consider the overall size of the workflow while selecting the tasks for scheduling. The rational behind consideration of task sizes versus the workflow sizes is to see which size makes the difference in terms of the system throughput and the workflow performance.

Table 6-1. List of scheduling heuristics based on size (execution lengths)

Heuristic	Acronym	Description
Shortest Task	ST	Ready tasks are selected for scheduling in ascending order of their sizes.
Longest Task	LT	Ready tasks are selected for scheduling in descending order of their sizes.
Shortest Workflow	SW	Ready tasks are selected for scheduling in ascending order of their workflow sizes.
Longest Workflow	LW	Ready tasks are selected for scheduling in descending order of their workflow sizes.

Figure 6-3 provides an overview of the scheduling heuristics based on the size attribute as listed in table 6-1. Given the dynamic nature of the scheduling, a snapshot of a very simplified system is presented in figure 6-3. There are two workflows W1, and W2 with size 100 and 200 units respectively and two services S1, and S2 within the system. In the snapshot there are three tasks W1T2, W2T1, and W2T2 sizes of 15, 10, and 30 respectively that belong to both workflows. The resulting scheduling selection order via each heuristic is shown in the figure 6-3. Although, a simplified scenario, it can be seen that each heuristic results into a different order for task selection and scheduling. Further it can also be seen that number of tasks (three) is larger than the number of services (two). After the first two tasks are assigned to the two services, the third task waits until one of the services (S1) becomes available for further assignments.

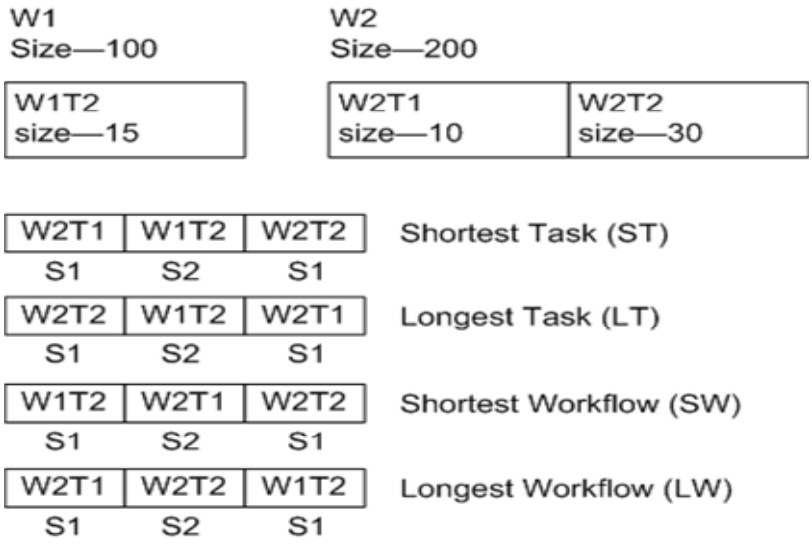

Figure 6-3. A simplified scheduling scenario using size based heuristics

The next set of four heuristics in presented in table 6-2 and it is based on the task dependencies within each workflow. The MCT and MOT pay attention to the workflow execution progress that is a direct result of the task dependencies. The LDT and MDT simply look at the number of dependent tasks to select the tasks for scheduling. The rational behind these four heuristics is to see how task dependencies impact the system throughput and the workflow performance.

Table 6-2. List of scheduling heuristics based on task dependencies

Heuristic	Acronym	Description
Most Completed Tasks	MCT	Preference is given to the ready tasks that belong to the workflow that has the most number of the completed tasks.
Most Outstanding Tasks	MOT	Preference is given to the ready tasks that belong to the workflow that has the most number of the outstanding tasks.
Least Dependent Tasks	LDT	Preference is given to the ready tasks that belong to the workflow that has the least number of the dependent tasks.
Most Dependent Tasks	MDT	Preference is given to the ready tasks that belong to the workflow that has the most number of the dependent tasks.

In figure 6-4, a snapshot of a simplified system is shown to illustrate the behavior of each heuristic presented in table 6-2. There are two workflows W1, and W2 and two services S1, and S2 in the system. The figure lists the system state information in terms of the workflow task dependencies. For example, at the time of the snapshot each workflow has a certain number of completed, outstanding, and dependent tasks. The number of dependent tasks is a constant metric and does not change throughout the workflow execution. There are three ready tasks W1T23, W2T14, and W2T15 in the snapshot. These tasks are then sorted and scheduled using each of the heuristics and results are displayed in the figure 6-4.

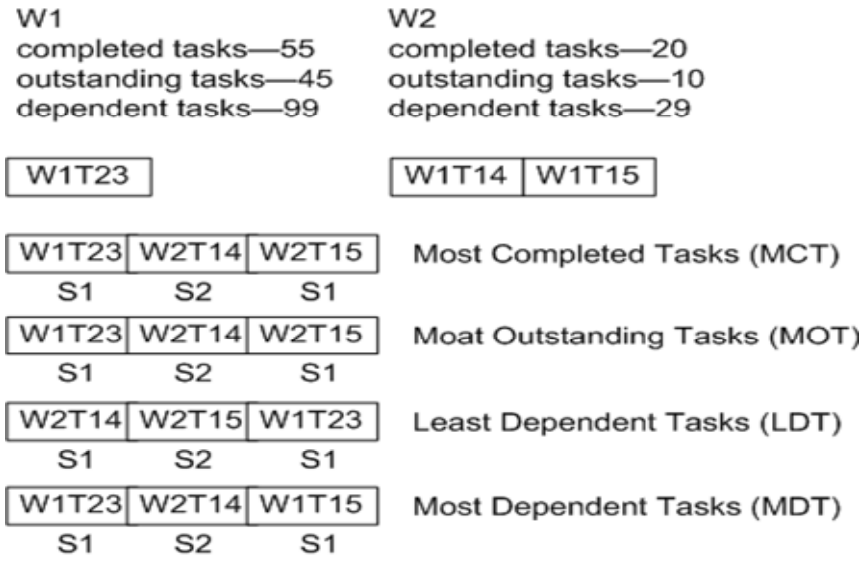

Figure 6-4. A simplified scheduling scenario using task dependency based heuristics

The implementation of this scheduler is greedy in nature. That is, using a particular heuristic, the scheduler attempts to allocate as many ready tasks as possible, given the service availability. The overall approach here is to achieve maximum service utilization by saturating the cloud with the workflows. This approach makes it possible to view the impact of each scheduling heuristic when a cloud is operating at its maximum capacity.

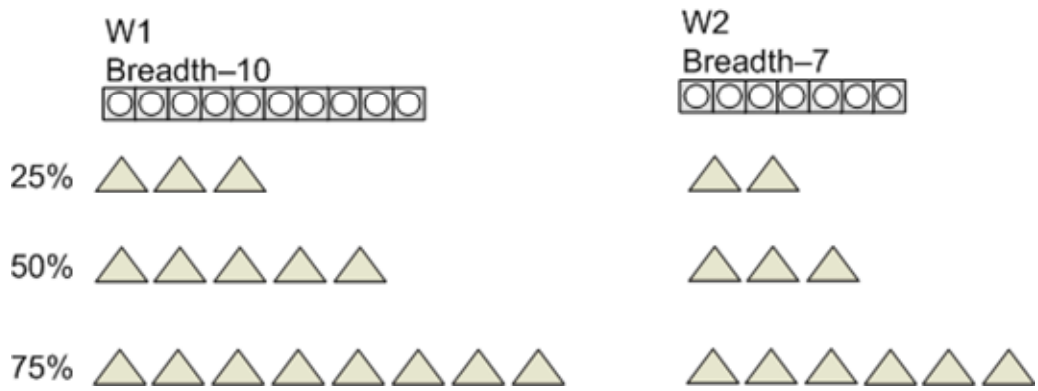

Figure 6-5. A simplified scheduling scenario with the service partitioning

Service Partitioning. The scheduling approach under this policy is to attempt a number of service allocations for a workflow up to a pre-determined threshold. The threshold is based on the workflow breadth, which is an indicator of the maximum service requirement for a workflow. Various factors of the workflow breadth are used during the simulation to determine the service count threshold. Figure 6-5, presents three such factors of 25%, 50%, and 75% to partition services among two workflows W1, and W2. As seen different breadth factors result into different number of service allocations for each workflow. Therefore, the service count threshold is directly proportional to the workflow breadth and the actual value will vary from workflow to workflow in a given set of workflows. Figure 6-5, does not restrict the number of services within the cloud and shows the ideal service allocations (service count threshold). However, in reality any cloud will have a fixed number of services and the service threshold counts determined via partitioning policy may or may not be available for allocation. In this case the workflow will only be allocated the maximum number of available services close to the service count threshold.

The rational behind this scheduling policy is to see the impact of a more controlled scheduling approach on the system and workflow performance. The control occurs in terms of the number of services partitioned/allocated among the outstanding workflows in the system. This is different than the uncontrolled scheduling policy of *prioritization,* described in the last section. With the

prioritization policy, workflows or tasks get a certain order of execution but there is no control over the number of services allocated to each workflow. With service portioning it is possible to control the number of services allocated to each workflow. It is possible to use many heuristics to decide on the number of services allocated to each workflow. In this thesis the workflow breadth is used to determine the service count that is allocated to each workflow.

Notice that the service allocation is a best case effort given the service availability in the cloud. A service is generally only available for scheduling when it is not executing any previously scheduled tasks. During the execution, services are allocated to the outstanding workflows as they become available. A workflow remains outstanding until the number of allocated services becomes equal to the specified threshold. For example, a workflow with a threshold of 5 services will remain outstanding until it has been allocated 5 services. However, outstanding status of the workflow does not impede its execution. A workflow starts execution as soon as the first service is allocated to the workflow. Therefore, during the workflow execution, the number of allocated services can be less than or equal to its threshold.

Deadlines. Implementation of this scheduler is an extension of both service partitioning and workflow prioritization schedulers. The target of this scheduler is to address workflow deadlines, when present. That is, this scheduler prioritizes deadline workflows over non deadline workflows. Deadline workflows are considered to have two deadlines. An end time (ET) deadline is by which a workflow must terminate its execution and a maximum execution length (MEL) after the workflow is scheduled. From the scheduling point of view, sufficient resources must be allocated to a deadline workflow that it meets both of its deadlines. Failure to meet either deadline results in the overall incompletion of the workflow.

The service partitioning scheduler is used as a base scheduler because it provides control over the number of services allocated to the workflows. The cloud can have workflows with or without deadlines, therefore various possibilities on how scheduling for deadlines occur exist. Three such possibilities while giving deadline workflows a priority are implemented and each subsequent implementation is an extension of the previous implementation. These three implementations are simply referred to as deadline-1, deadline-2, and deadline-3.

First (deadline-1), results obtained with the service partitioning scheduler are used. Using the best performing threshold, the workflows are scheduled in an order that prioritizes deadline workflows over non deadline workflows, among outstanding workflows in the queue. Order among the deadline workflows themselves when selecting between two deadline workflows, can be based on four attributes; the workflow arrival times, the MEL deadline, ET deadline, or the respective workflow sizes.

In the second implementation (deadline-2), the scheduler implements two separate thresholds for deadline and non deadline workflows, respectively. This is in addition to the deadline priority based workflow selection. The workflow without deadlines can be run without any time constraints, and by allocating more services to the workflows with deadlines improves their chance of meeting their deadlines. The non deadline workflows are restricted to a very small number of services (one or two) to make room for the deadline workflows. The service count threshold for the deadline workflows is set to meet their MEL. The number of services that will meet the MEL is determined and an attempt is made to allocate such a number of services. A workflow is not scheduled until the number of resources meeting its MEL becomes available.

The third implementation (deadline-3) of the scheduler is similar to the second implementation. It initially allocates the services for the deadline workflows that are sufficient to

meet the MEL deadline, but it continues to allocate more services until the optimal number of services is allocated or the workflow finishes its execution. The optimal number of resources is equal to the workflow breadth. This implementation is to favor the ET deadline. That is by attempting to allocate the optimal number of services, chances to achieve the ET may improve.

Objectives

Given the discussion presented in this chapter the experimentation objectives can be summarized as follows:

- Model and execute a Service Oriented Computing Cloud (SOCC) via simulation
- Schedule and execute DAG based workflows over the modeled SOCC
    - Implement *prioritization* and *portioning* based scheduling policies. Both scheduling policies are instantiated using many scheduling heuristics as described in the last section.
    - Schedule two DAG types of EDAG and CDAG
- Implement various cloud sizes in terms of the number of services
- Implement various workflow arrival times and workflow sizes and task dependencies
- Measure system throughput and average workflow performance for each implemented scheduling scenario
- Analyze the results to present the performance of scheduling policies in various scenarios in terms of
    - System throughput
    - Workflow performance
    - Impact of workflow structure (task dependencies)
    - Ability to meet execution deadlines

CHAPTER 7
RESULTS

There are four research questions formulated at the end of chapter 2, which are as follows.

1. How does the cloud size impact the performance of the various scheduling policies in terms of the system throughput and the workflow performance? In general the performance of any scheduling policy will likely benefit with the increased number of services in a cloud. However, performance of various policies is compared with each other for different cloud sizes to evaluate their suitability.

2. How do the workflow arrival patterns impact the scheduling? It is possible that not all of the workflows are ready for execution at the same time therefore scheduling is an ongoing activity. Different arrival patterns cause different amount of workload on the cloud. The performance of the scheduling policies under different loads is examined.

3. What role does a workflow structure play during scheduling? In particular what attributes of the workflow structure, such as number of tasks or the task dependencies are valuable in workflow scheduling.

4. How do the workflows that have certain completion deadlines perform with the prioritization and the partitioning policies?

Table 7-1. All possible values of the experimentation parameters

Scheduling Policy	Cloud Size	Workflow Arrival Patterns	Workflow Structure	Deadlines
Prioritization	Variable/Fixed	Fixed, Burst, Exponential	EDAG/CDAG	Yes/No
Partitioning	Variable/Fixed	Fixed, Burst, Exponential	EDAG/CDAG	Yes/No
Deadlines	Variable/Fixed	Fixed, Burst, Exponential	EDAG/CDAG	Yes/No

In this chapter, the results of the experimentation are outlined. The results are described to answer the four research questions above.

There are three sections in this chapter. The first two sections describe the results for *workflow prioritization* and *service partitioning* scheduling policies, respectively. The last section describes the results for the deadline oriented workflows.

Table 7-1 presents the overall experiment configuration parameters and their implemented values. Each following section first describes the parameter value of the results described in each section and the motivation behind their selection.

Workflow Prioritization

Workflow and Cloud configuration

There are two workflow sets used for scheduling using the prioritization heuristics described in table 6-1 and table 6-2. The configuration parameters for both workflow sets are shown in table 7-2, and 7-3. The sets are simply referred to as sets I, and II. Set I contains EDAG workflows and set II contains CDAG workflows. Workflows within each set are also different from each other in terms of the number of tasks, task lengths and the task dependencies. The workflows in each set follow an exponential distribution for their sizes. That is there are a large number of smaller size workflows and a small number of larger size workflows in each set. The motivation behind the two selected sets is to study the impact of the workflow structure, identified by workflow type, on the scheduling. The workflows in both sets are opposite of each other. As seen in table 7-3, set I has workflows with a large number of tasks (up to ~24000) with small execution lengths (1-10 seconds). Whereas, set II has a relatively small number of tasks (7-10) with longer task lengths (1000 seconds). As previously mentioned in chapter 6, the EDAG workflows have increasing number of resource requirements as their execution progresses, whereas the CDAG workflows generally have fixed resource requirement throughout their execution.

Table 7-2. Workflow configuration parameters for the priority scheduler

Set	Workflow Type	Workflow Count	Size	Task Readiness	Task Length
I	EDAG	100	Variable e*xponentially distributed*	Per level	Variable e*xponentially distributed*
II	CDAG	1000	Variable e*xponentially distributed*	Per level	Variable e*xponentially distributed*

Table 7-3. Workflow detailed configuration for the priority scheduler

Set	Depth	Breadth	Task Length Mean (seconds)	Number of Tasks	Average Number of Tasks	Workflow Size	Average Workflow Size
I	1 – 6	1 – 19336	10 *exponentially Distributed*	1 – 24835	2595	2 – 262141	27347
II	20	7 – 10	1000 *exponentially Distributed*	77 – 147	110	67438 – 195199	122745

In table 7-4, other experimental configuration parameters are shown. As evident, various cloud sizes, and workflow arrival time distributions are implemented in order to see their impact on the heuristic performance.

Table 7-4. System configuration parameters for prioritization scheduler

Set	Cloud Size (# of services)	Arrival Time
I	100, 500, 1000	Exponential, Continuous, Bursts
II	100, 500, 1000	Exponential, Continuous, Bursts

Results

Following is the overview of the results obtained under prioritization based scheduling

1. System throughput remains (almost) the same across all heuristics implemented that prioritize the workflows or their tasks. This is due to the fact that workflow scheduling order does not change the overall outstanding work.

2. Workflow performance significantly varies across various scheduling heuristics. The heuristics considering overall workflow size as opposed to the individual task size perform better.

3. The heuristic performance tends to diminish as the cloud size increases because more services are available for workflow execution. Therefore, with a larger cloud sizes prioritization based scheduling is sub optimal and other scheduling techniques are needed to optimize workflow performance.

4. Workflow arrival times impact the scheduling only when they introduce significant load on the cloud resources. Only exponential distribution managed to introduce significant load on the cloud during the experiment.

5. Workflow structures (EDAG or CDAG) dictate the number of workflows that can be scheduled within a cloud. Some heuristics perform differently with different workflow types. Additionally, given their resource requirement, heuristic based scheduling is only effective when cloud is under high workload.

System throughput

Given the greedy nature of the priority scheduler the overall throughput of the cloud stays relatively the same across various scheduling heuristics. Regardless of the workflow order resulting from the priority heuristics, the overall amount of work keeps the underlying services busy until all of the workflows are executed. This is revealed in figure 7-1 and 7-2. Additionally, given the scheduler attempts to allocate as many services as possible to the outstanding workflow

tasks, the service utilization is at the maximum level.

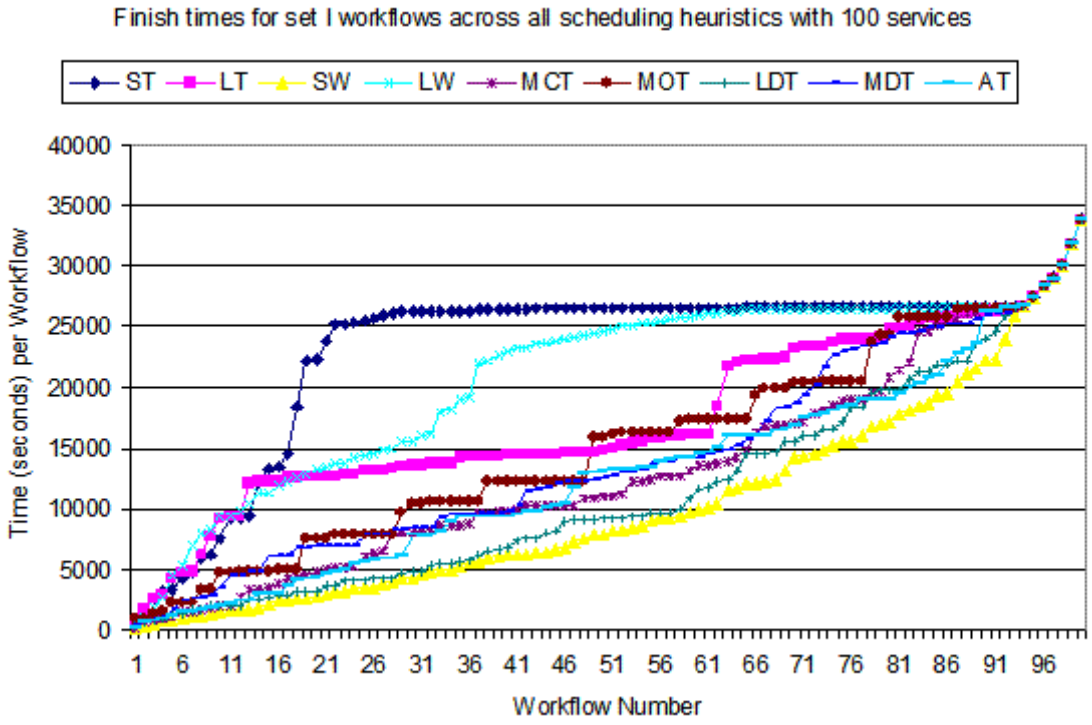

Figure 7-1. Finish times for set I workflows across all scheduling heuristics

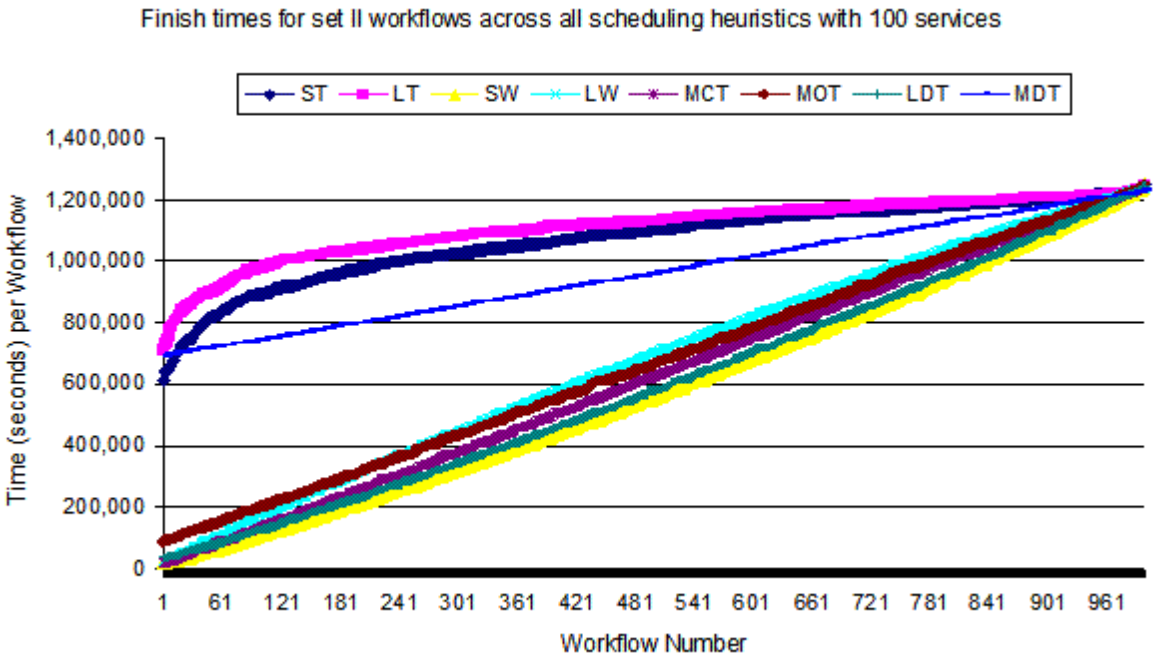

Figure 7-2. Finish times for set II workflows across all scheduling heuristics

Workflow Performance

For workflow set I (EDAG) and set II (CDAG) in conjunction with the smaller cloud size, there exist performance gaps between various heuristics. Figures 7-1 and 7-2 display the individual heuristic performance for a case where such differences are significant. The heuristic performance in 7-1 and 7-2 are charted for a cloud of 100 services when the arrival time was exponentially distributed. Overall the Shortest Workflow and the Least Dependent Tasks perform the best, whereas the Shortest Task performs the worst.

The results in figure 7-1 and 7-2 provide two insights. First the workflow selection order is important. Second, consideration of the overall workflow size rather than the individual task size makes a difference in performance. In set I, there are a larger number of small size workflows and a small number of large size workflows. Keeping the EDAG structure of the workflows in mind, scheduling smaller workflows first allows more and more workflows to be executed earlier, therefore, reducing their wait times. On the other hand, when the larger workflows are scheduled first, they consume most of the cloud resources, causing much longer wait times for a large number of smaller size workflows. In set II, the number of workflows (1000) is much larger than the number of workflows (100) in set I. The increased number of workflows was necessary to generate a similar amount of load on the cloud. The Shortest Workflow and Least Dependent Tasks continue to perform best for set II as well. However, in terms of worst performance Longest Task now beats Shortest Task. Remember that workflows in set II are of CDAG type and have long task sizes. Therefore, when Longest Tasks are scheduled first, the wait time for tasks belonging to other workflows is increased. This results is an overall longer finish times as compared to the Shortest Task. Further, Longest Workflow does not have the

poor performance as in the case of set I. The Longest Workflow becomes a viable option in set II as the size variance among workflows is much smaller than set I.

The scheduling based on the individual task sizes does not guarantee any particular order for workflows and that is the reason behind the poor performances of heuristics of Shortest Task, Longest Task, and Longest Workflow that are either based on the individual task sizes or prefer large size workflows.

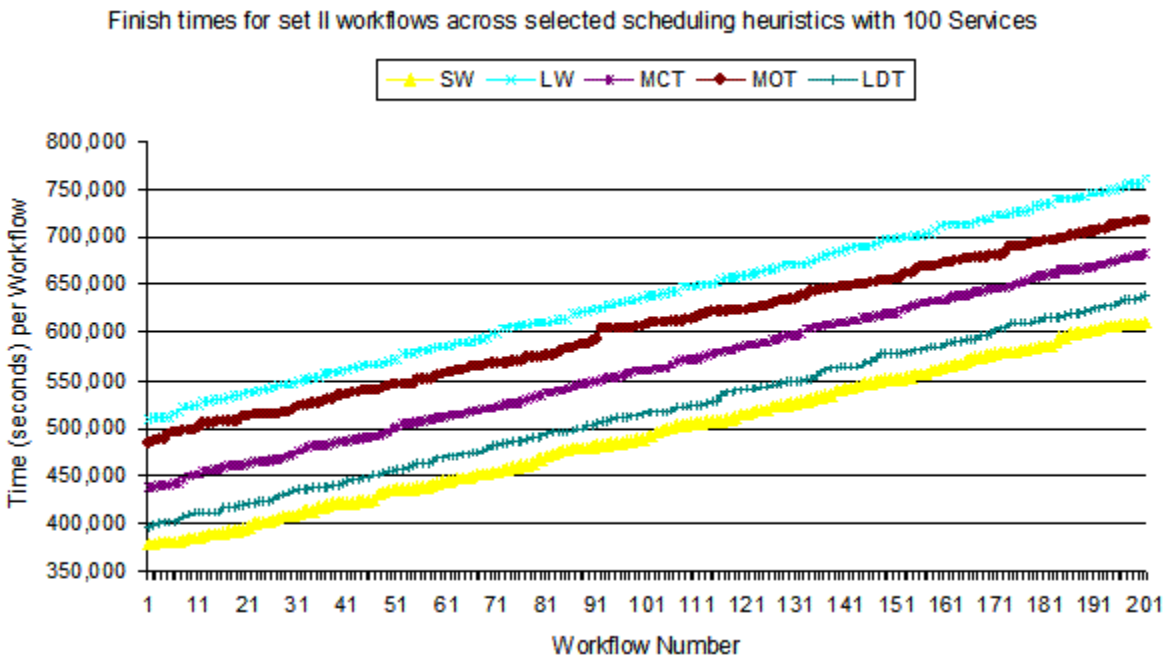

Figure 7-3. Finish times for set II workflows across selected scheduling heuristics

There are two other interesting observations available from figure 7-1. The first observation is for the performance of Most Completed Task heuristic. It comes right after Shortest Workflow and Least Dependent Tasks. This heuristic can be valuable in scenarios when workflow sizes or information on their task dependencies is not available, making Shortest Workflow and Least Dependent Task impossible. Workflow execution progress can be monitored in terms of the work completed, therefore, making Most Completed Tasks a viable option. The second observation is for the Arrival Time (AT) heuristic that is absent from the original heuristic

description in the table 6-1. This heuristic was injected to see the impact of scheduling workflows in order of their arrival. Arrival Time performance lies in the middle and there are four other heuristics that perform worse than Arrival Time. It is an interesting observation in a sense that such a simple scheduling heuristic can still yield reasonable results over other heuristics requiring more information about the workflows.

One more important thing to note is the scale in figure 7-2. Given the sheer amount of data in figure 7-2, from the initial visual inspection it may seem that most of the heuristics are performing similar to each other as they appear very close to each other. However, when compared in terms of scale, it is revealed that the performance gaps between these heuristics still remain significant to matter their selection. In figure 7-3, a close up view of the heuristic performance is shown to provide the difference in performance for heuristics that, at first, appear too close to each other in figure 7-2. As seen in figure 7-3, the performance difference between the heuristic of Shortest Workflow and the Longest Workflow is on average 120,000 seconds (2,000 minutes or 33.33 hours). That is Longest Workflow heuristic on average takes 120,000 second more than Shortest Workflow to finish the execution of each workflow shown in figure 7-3.

Cloud Size

The above results in figure 7-1 and 7-2 are for the small cloud sizes. In general, as the number of services in a cloud is increased, the performance gaps between heuristics start to diminish. This is because with more resources it is possible to execute the arriving workflows sooner, therefore, reducing the average wait time and making the selection order less important. There are 100 services for the workflows in figure 7-1. When the same workflows shown in figure 7-1 are executed with 500 services as opposed to 100 services, the results are shown in figure 7-4. As

seen the performance gaps between heuristics have reduced significantly in comparison to the heuristic performance for the same workflows but fewer services (100) in figure 7-1. With the larger cloud sizes, the service utilization is uneven across services as only a subset of services stay busy throughout the execution.

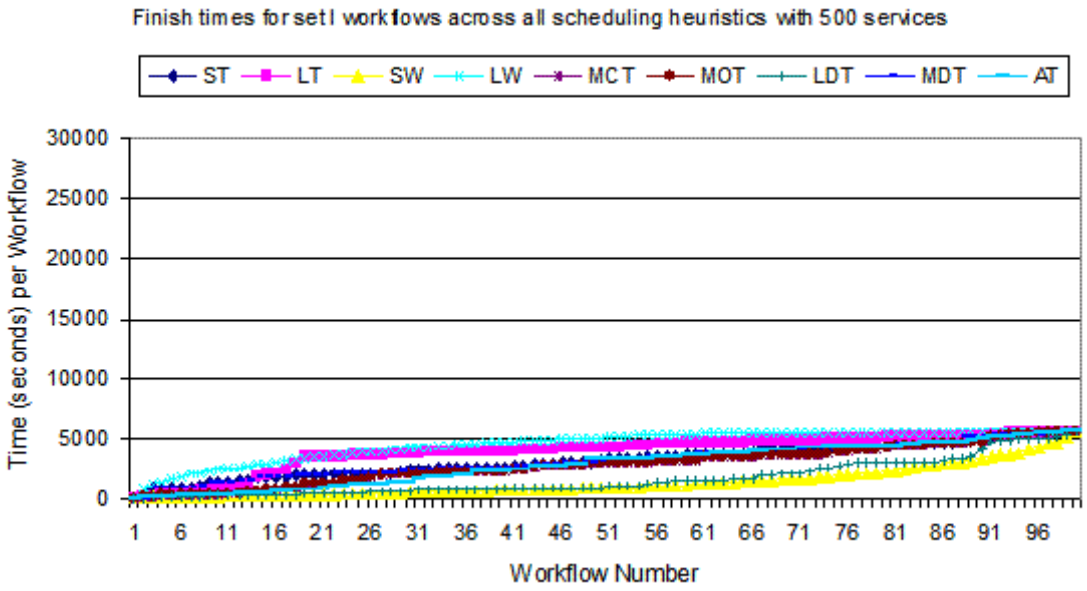

Figure 7-4. Impact of the additional number of services on the performance of heuristics

Workflow arrival time patterns

The exponential arrival time distribution causes the most amount of load on the cloud. A relatively large number of workflows arrive close to each other therefore overwhelming the system. With the uniform arrival time, whether workflows arrive continuously or in bursts with fixed intervals, the overall outstanding workflows in the system remains small. This small number of workflows is a manageable load for the cloud in general and workflows can be scheduled sooner with smaller wait times. Therefore, workflow prioritization through heuristics when workflow arrival does not cause significant load on the cloud is less effective. Figure 7-5 shows such an impact of the arrival times on the performance of heuristics for 100 services cloud with the uniform arrival time. However, with uniform arrival rate, it is possible to partition the

cloud into smaller subsets. The smaller subset cloud can then take advantage of the scheduling heuristics to deliver the reasonable performance. The freed up services in the other cloud partitions can be subsequently employed towards the other arrival time patterns or for the resource intensive workflows. Therefore, it can further reduce or eliminate the large workflows that can potentially impact the overall cloud performance.

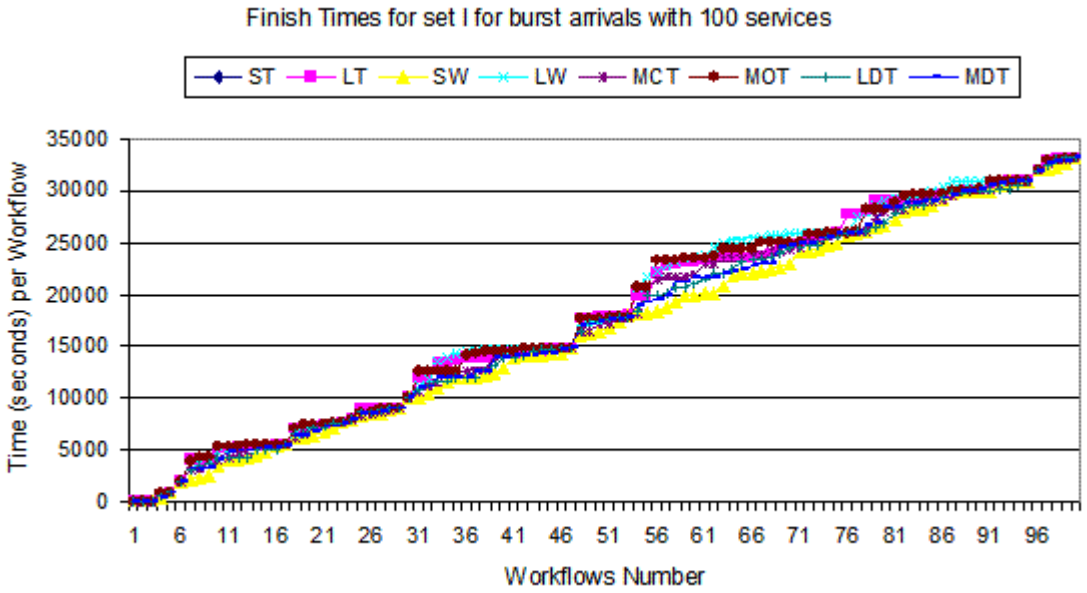

Figure 7-5. Scheduling heuristic performance with (uniform) burst arrival pattern

Summary

In a smaller size service cloud, workflow wait times are the key when scheduling continuous stream of workflows in a greedy fashion. Any scheduling technique that reduces the average wait time for the workflow performs the best. Scheduling heuristics provide a way to reduce the workflow wait times by considering the workflow sizes. When workflows are scheduled in increasing order of size, the overall wait times across workflows are minimized resulting in the best performance in terms of the average finish time. The task dependencies do not seem to make any significant differences when all of the workflows follow similar task dependency structure. However, different task dependency structure, EDAG vs. CDAG, results in different service

requirement patterns. The EDAG workflows require a very large number of services for a short period of time, whereas the CDAG workflows require a small number of services for longer periods of time. The different service requirements patterns then dictate the number of workflows that should be scheduled for heuristics to be an effective mechanism. The number of tasks in a workflow directly contributes to the workflow size and is already considered as part of the workflow size based heuristics.

The workflow wait times can also be reduced via other methods as well. First, by adding more resources to the cloud more and more workflows can be scheduled soon after their arrival. Second, by making the workflow arrival into the system more uniform, it is possible to keep the number of outstanding workflows small enough. Cloud partition is another technique that can be applied to eliminate the factors causing longer wait times. For example, small number of larger workflows can be allocated to a subset of the cloud so they do not cause longer delays for the large number of smaller workflows.

Service Partitioning

Workflow and Cloud configuration

Tables 7-5 and 7-6, show the various configurations of the simulation results of the partitioning scheduler discussed in this section. As seen, all three set contain CDAG workflows. The service partitioning based scheduling turns out to be a poor candidate for the EDAG workflows. This is due to their increasing resource requirement that always surpasses the number of services available within the cloud. In such a limited resource environment, only a greedy scheduling approach such as workflow prioritization described in previous section yields the best results for EDAG workflows. With such a greedy approach the prioritized workflows compete and consume the maximum number of services possible. Whereas with service partitioning EDAG

workflows end up with a smaller number of service allocation and therefore significantly suffer in performance.

Table 7-5 indicates three workflow-sets I, II, and III of 100 CDAG workflows each. The first set I contains 100 identical (instances of a single workflow) workflows. This set is to simulate a scenario where multiple executions of a workflow are invoked over a period of time, e.g. to process different data sets at each execution. The last two sets II and III have variable size workflows, where the overall size is uniformly and exponentially distributed, respectively. These two sets differ in terms of their workflow breadths. The last two sets are selected to study the scenarios where multiple workflows of various breadths and sizes are executed within the cloud. This is to see any relationship between workflow breadth, sizes and the number of allocated services i.e. the service partition.

Table 7-5. Workflow configuration parameters for the partitioning scheduler

Set	Workflow Type	Workflow Count	Size	Task Length	Breadth (tasks)
I	CDAG	100	Fixed	Fixed	10
II	CDAG	100	Variable *uniformly distributed*	Fixed	10
III	CDAG	100	Variable *exponentially distributed*	Fixed	1-14

Table 7-6 shows three other variables for each workflow set. Each set is executed multiple times across the indicated cloud sizes, service count thresholds, and scheduling heuristics. The service count threshold, as previously described in chapter 6, determines the service portion and it is based on the workflow breadth. The three scheduling heuristics are Arrival Time (AT),

Shortest Workflow (SW), and Longest Workflow (LW). These three heuristics provide the order in which the workflows are selected for scheduling.

Table 7-6. Configuration for the service partitioning scheduler

Set	Cloud Size (service count)	Service Count Threshold (%)	Heuristic
I	100, 200, 300	25, 50, 75	AT, SW, LW
II	100, 200, 300	25, 50, 75	AT, SW, LW
III	100, 200, 300	25, 50, 75	AT, SW, LW

The motivation behind implementing heuristics, in addition to the service count threshold, is to see any further improvements in the system. For example, if a 25% threshold performs best in some cases, is it possible to extend the performance gains by considering various orders of the workflow selection.

All of the workflows sets have exponential arrival time distribution. The other arrival time distribution does not seem to generate any significant load on the cloud to warrant any sophisticated scheduling.

Results

Following is the overview of the results obtained under the partitioning based scheduling

1. System throughput varies across various service partitions (count thresholds). The workflow size and breadth distribution dictates what service count thresholds and cloud size perform better.

2. Individual workflow performance is also affected by the partition size and on average a service threshold of 50% (1/2 of the workflow breadth) performs well.

3. Workflow arrival patterns that introduce significant load on the cloud such as exponential distribution are benefited by the service partitioning scheduling. Other arrival patterns simply do not have enough load on the cloud to require any scheduling.

4. EDAG type workflows are a poor candidate for the service partitioning scheduling and require more greedy scheduling approaches for better results. CDAG type workflows make a good candidate for service partitioning as this scheduling policy provide more control over service allocation per workflow.

5. No deadlines were implemented as part of this scheduler.

System Throughput & Cloud Size

Set I & II. While considering the total finish times (overall system throughput) there is a relationship between the cloud size and the service count threshold.

For set I and II, smaller threshold values perform better with the smaller cloud sizes. The total finish times are much lower with the smaller-size-smaller-threshold combination as compared to the smaller-size-larger-thresholds. For example in figure 7-6, total finish times for set II with Arrival Time heuristic are shown. For the cloud size of 100 and 200, service count threshold of 25% performs better over 50%, and 75%. As the number of services is increased to 300, 50% threshold becomes better. Note that the workflow in these sets (I, and II) is either identical or similar to each other. With larger thresholds it is easy to saturate the cloud services with a small number of workflows, causing the rest of the workflows to wait. Whereas, with smaller thresholds it is possible to reduce the overall workflow wait times by allocating services more evenly among a large number of similar workflows. More workflows start the execution early on, which results in an overall better system performance.

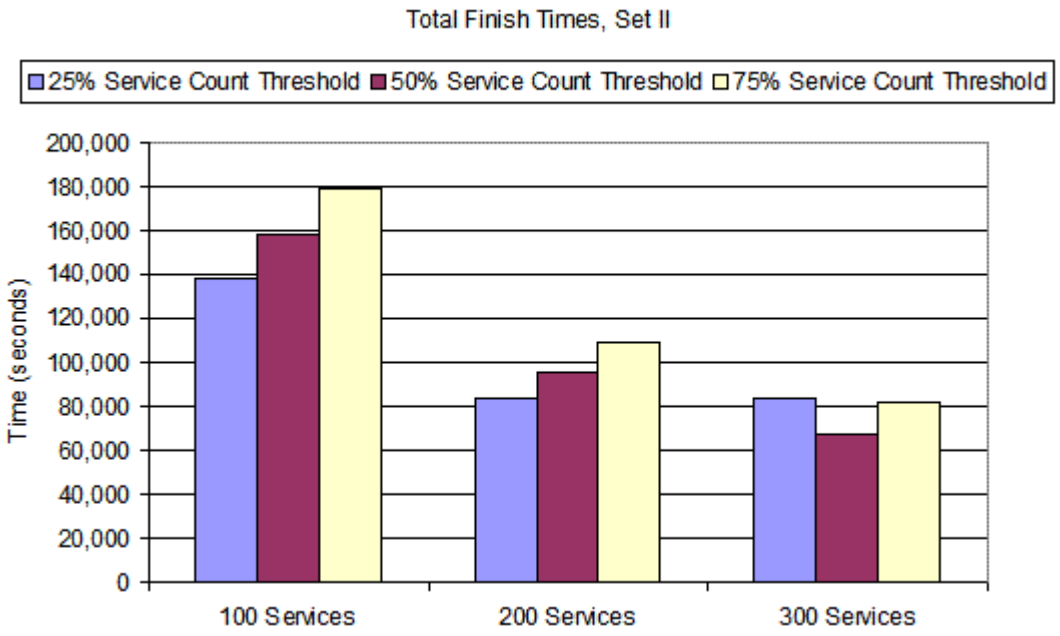

Figure 7-6. Total finish times for set II workflows with Arrival Time heuristic

Figure 7-7 shows the average wait times across the cloud sizes of each threshold for Set I. As more resources are added, it becomes possible to expand the number of services allocated to each workflow. This further expedites the workflow execution as in the case of 50% threshold and 300 resources in figure 7-6.

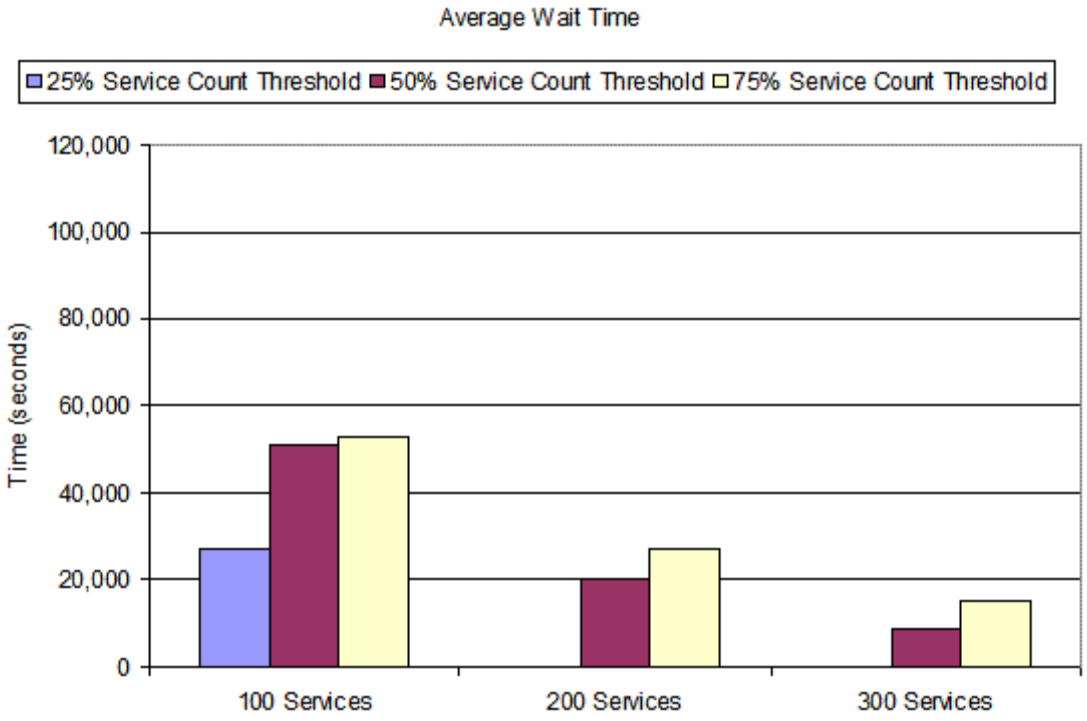

Figure 7-7. Average wait times for set I workflows with Shortest Workflow heuristic

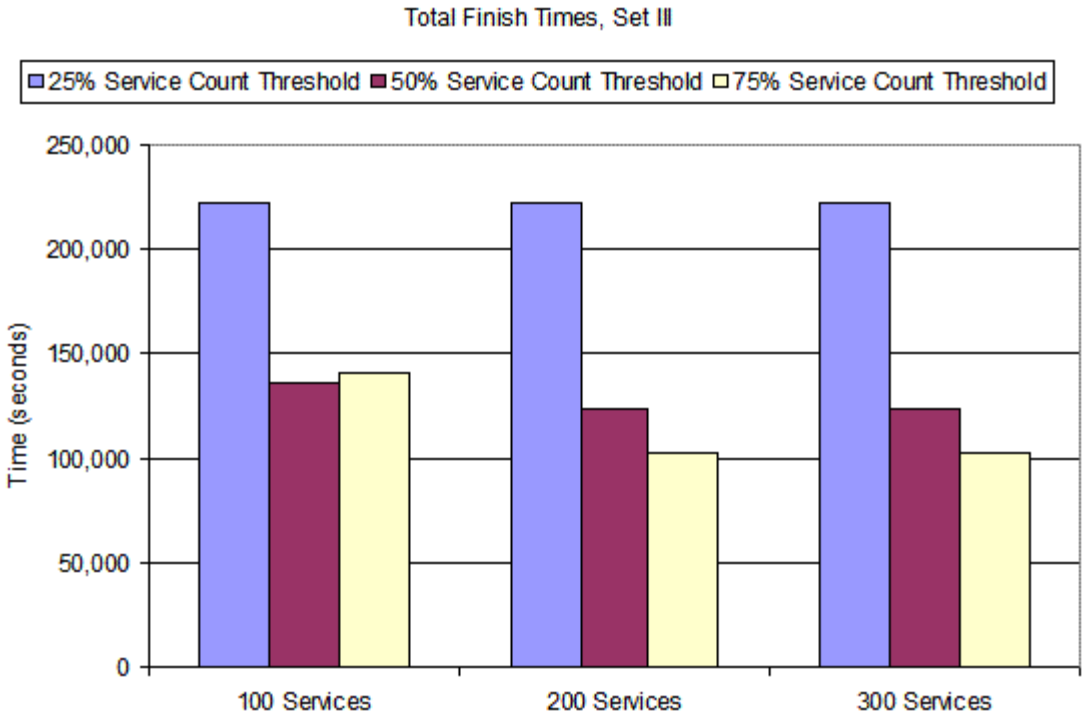

Figure 7-8. Total finish times for set III workflows with Arrival Time heuristic

Set III. For the set III, the larger threshold performs better. This is in contrast to the results obtained with set I, and II. The breadth and size of workflows in set III fluctuate quite a bit and have exponential size distribution as shown in figure 6-3. As per exponential distribution, there are a large number of relatively smaller workflows in set III and a few large workflows. The larger threshold for abundant smaller workflows results in a small number of service allocations per workflow which is easy to accommodate, yet it provides better performance for the individual workflow and the overall system performance. For example, consider a case of 40 workflows with a breadth 6 or less. Even with the large threshold of 75%, the allocations are between 1-4 services. The overall results are shown in figure 7-8. Smaller threshold performs significantly poor. This is due to the fact that the larger workflows get a very small set of services and delay the overall completion of the workflow set.

System Throughput and Heuristics

Set I. In terms of improvements related to the scheduling heuristics, for set I, there are no further improvements. This is because both Shortest Workflow and Longest Workflow heuristics are based on the workflow sizes and set I contains identical size workflows.

Set II & III. For set II, and III, Longest Workflow performs better in some cases as seen in figure 7-9. This is largely for the small cloud sizes when longest workflows in conjunction with the small threshold do not consume a lot of cloud resources, leaving more resources available for the rest of the workflows.

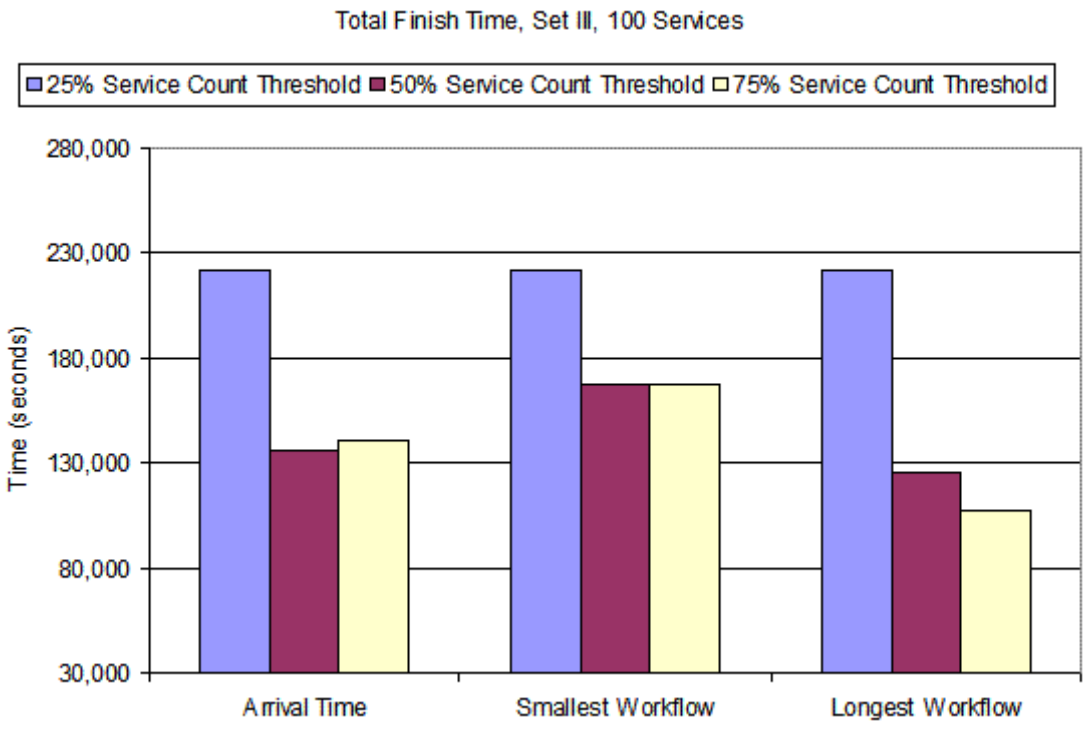

Figure 7-9 Scheduling heuristic performance for total finish time

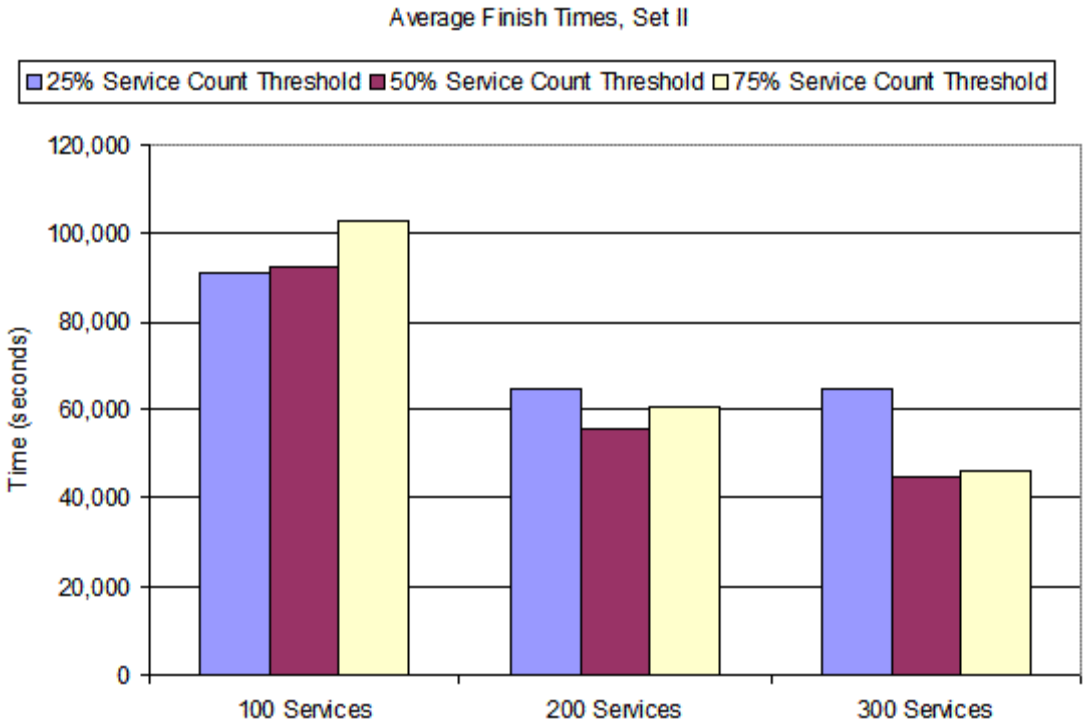

Figure 7-10 Average finish time of set II for Longest Workflow heuristic

Workflow Performance. In comparison to the total finish time, which is a measurement of the overall system performance, average finish time gives a sense of the individual workflow performance. There is no correlation between workflow sizes and the individual finish times. Workflows of various sizes arrive randomly into the system and are allocated services using some scheduling policy. It is possible that a relatively small workflow performs poorly (longer finish time) because they arrived at a time in the system when most of the services were already allocated, therefore causing it to wait for a long time.

Set I & II. In general, service count threshold of 50% results a better average performance for set I, and II workflows. This is due the fact that workflow sizes in those two sets follow a uniform size distribution. Therefore, on average most of the workflows take advantage of more resources with 50% threshold over 25% thresholds. Figure 7-10 displays average finish times for set II with Longest Workflow heuristic. As more services are added, 75% starts to perform

similar to (and eventually better) the 50% threshold. This trend is likely to continue as more services are added, ultimately arriving at an ideal scenario where all of the workflows have optimal number of services allocated to them.

Set III. For set III, 75% threshold works better in most cases. In general, given the cloud size the larger threshold provides better average performance as it results in better service allocation for a large number of smaller workflows as seen in figure 7-11.

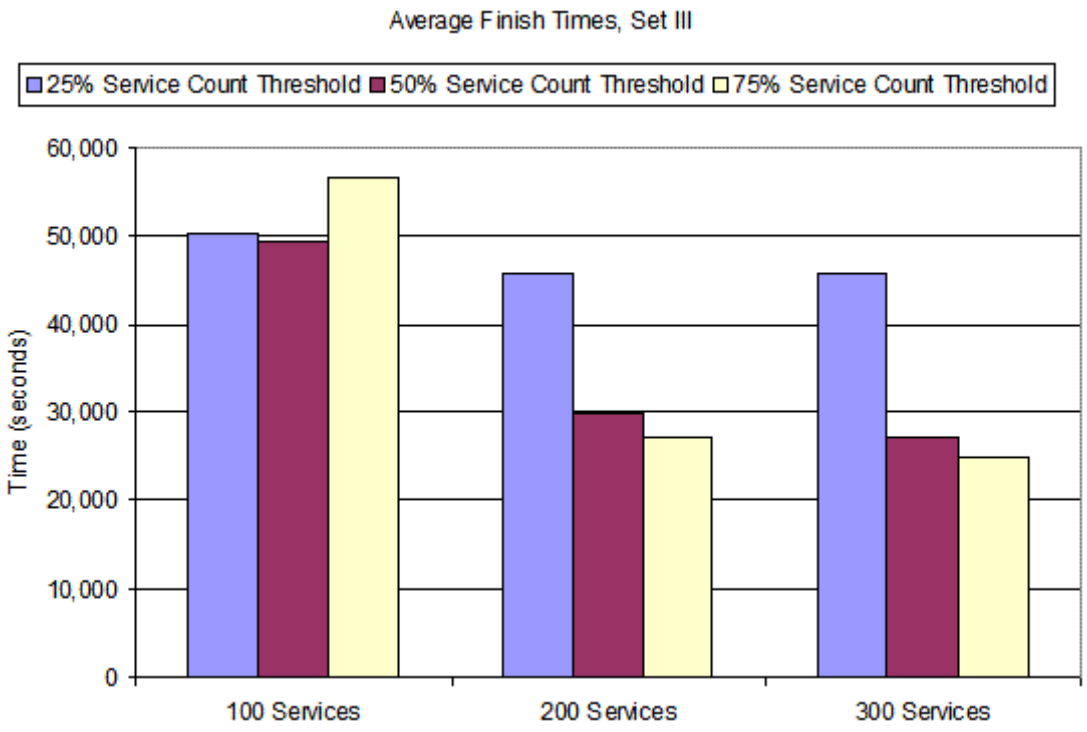

Figure 7-11. Average finish time of set III for Arrival Time heuristic

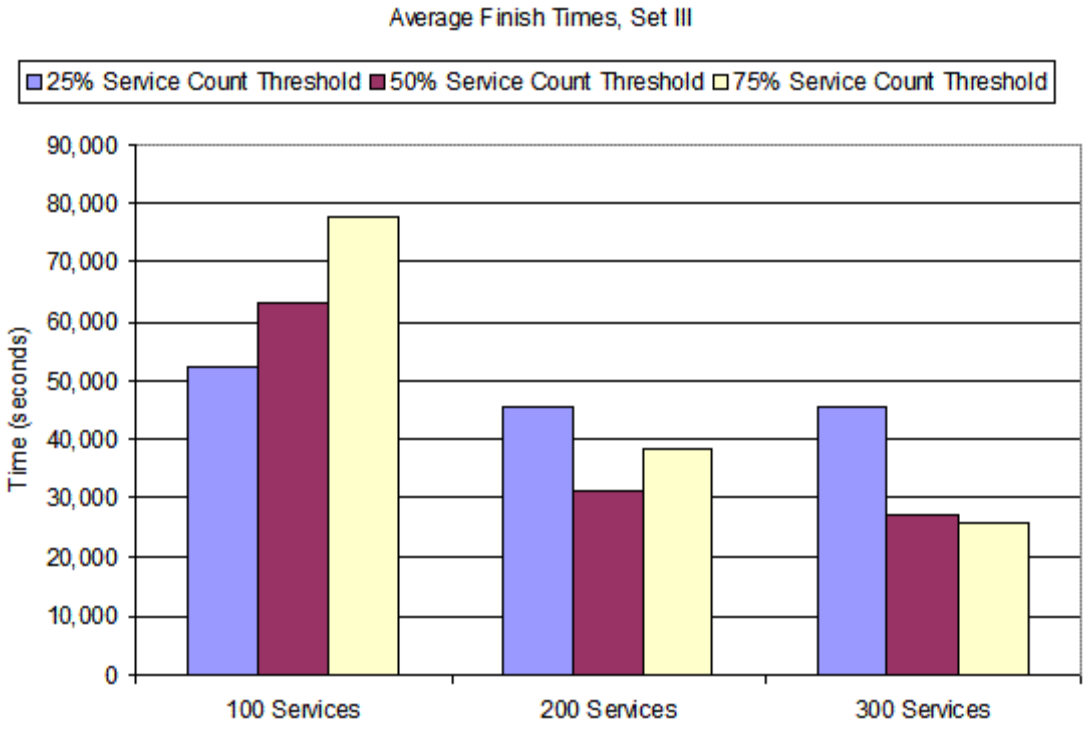

Figure 7-12. Average finish time of set III for Longest Workflow heuristic

Workflow performance & Heuristics

In terms of improvements related to the scheduling heuristics, Shortest Workflow with 50% threshold provides further improvements in average time for most cases. With the Longest Workflow heuristic a few very large workflows with larger threshold can consume more services resulting in poor average performance. In other cases, heuristics do not have any further impact on the average performance. Figure 7-13 shows improvements brought with the Smallest Workflow heuristic.

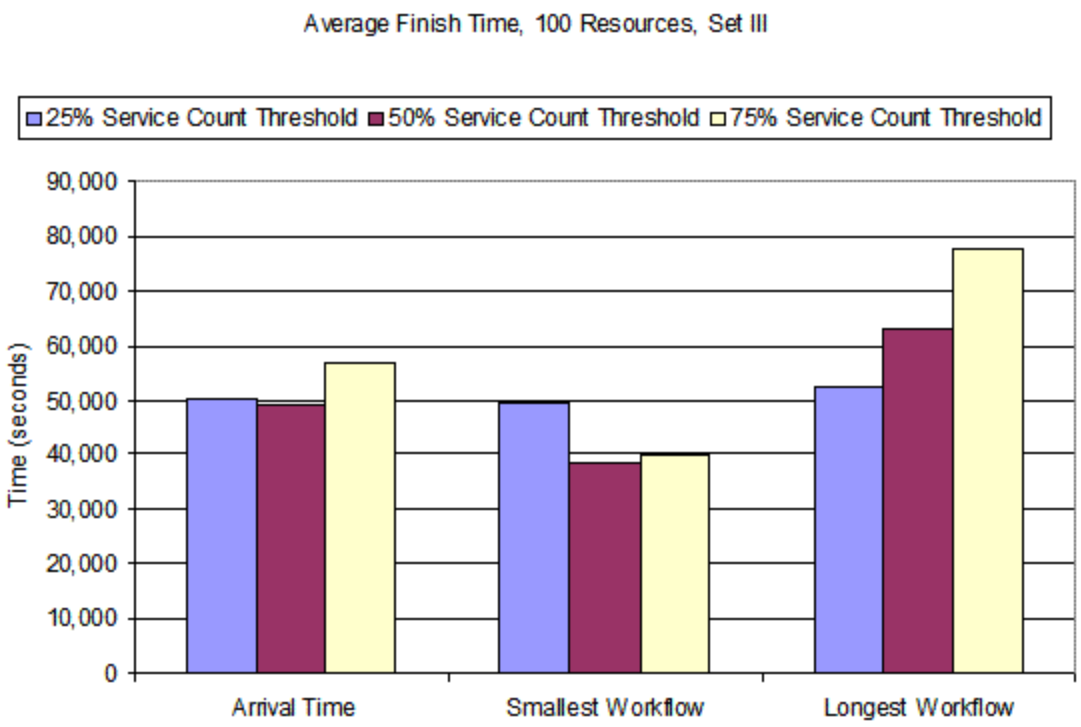

Figure 7-13. Scheduling heuristic performance for average finish time

Workflow Arrival Patterns

Only exponential arrival times are considered as with the prioritization scheduler experimentation it was revealed that uniform distribution does not generate enough load on the system to the have scheduling make a significant difference.

Service Utilization

Under service partitioning the utilization is uneven, as after meeting the specified allocation threshold no more services are allocated to the workflow even when there are services available.

Workflow Structure

Only CDAG type workflows were considered as with EDAG workflows which have a very large number of tasks, only a greedy policy such as workflow prioritization works well. The

service partitioning results into uneven distribution of services and perform much poorly for EDAG workflows.

Summary

System throughput is a function of the workflow wait time. Therefore, when more and more workflows are able to begin their execution sooner, the overall finish time across all workflows is at its minimum. That is, there is no advantage to allocating more services to a subset of workflows while making other workflows wait. With workflows similar in structure or having similar service requirements, a small service allocation to evenly distribute service results in the reduced wait times. Whereas for the workflows with considerable variation in service requirements, allocating more services to the smaller workflows produces lowest wait times. In some cases of varied service requirements, a scheduling heuristic that can restrict the larger workflows from consuming most of the cloud services, can also deliver further improvements in the total finish times.

Individual workflow performance depends on other workflows in the system. Obviously any scheme that allocates a number of services close to the optimal of service requirements for each workflow results in a better average performance. When the workflows in the system have similar service requirements, then allocating resources up to 50% of their optimum service count requirements yields the best average performance. In case of workflows with varied service requirements, best average finish times are achieved with the similar scheme as for the total finish times. That is, by allocating more services to smaller workflows and fewer services to the larger workflows, it is possible to achieve the best average times.

Smallest Workflow has the best average performance when services are equally allocated to the workflows without consideration to the variation in their services requirements. No other consistent benefits of heuristics are realized.

Deadlines

Workflow and Cloud Configuration

Two CDAG workflow sets are selected for the deadlines scheduler execution. Their configuration is shown in table 7-6. The deadlines scheduler implementation is built on top of the service partitioning implementation. Therefore, the EDAG workflows did not prove to be suitable candidates for the same reason as with service partitioning. Restricting service allocations to a fixed threshold makes it almost impossible to achieve target deadlines.

Table 7-6. Workflow configuration parameters for deadlines scheduler

Set	Workflow Types	Workflow Count	Deadline Workflows	Size	Breadth (tasks)	Deadline Generation
I	CDAG	100	25	Variable *exponentially distributed*	Variable *uniformly distributed* 1-14	Random
II	CDAG	100	25	Variable *uniformly distributed*	Fixed 10	Random

The only significant difference between two selected workflow sets in table 7-6 is the workflow sizes and their breadths. The variable vs. fixed breadth is a consideration while allocating services based on the workflow breadths. The overall number of the workflows with the deadlines in each set is set to 25% of the workflows in each set. A random selection is used to select workflows that are assigned deadlines. Workflows in each set arrive randomly by following an exponential distribution. The deadline and non deadline workflow arrival is intermixed together. Other arrival time distributions such as uniform did not pose any challenge towards achieving the deadlines as they generated relatively low load on the cloud.

The experimentation configuration is shown in table 7-7. Each workflow set is executed over a cloud of 100 services. The cloud size is kept constant for this scheduler as the objective is to study the impact of the fixed number of services, when a percentage of the overall workflows have dual deadlines. Otherwise by increasing the services in the cloud with the same number of deadline workflows, the problem becomes trivial. Each workflow is executed via all three different implementations as discussed in chapter 6, section Deadlines and for the reference purposes they are named deadline-1, deadline-2, and deadline-3.

Table 7-7. Deadlines scheduler run configuration

Set	Cloud Size (resources)	Arrival Time	Implementation
I	100	Exponential	deadline-1, deadline-2, deadline-3
II	100	Exponential	deadline-1, deadline-2, deadline-3

Results

Following is the overview of the results obtained under the partitioning based scheduling

1. Deadlines scheduler is a specialization of the workflow prioritization scheduler as the workflows are prioritized based on their deadlines.
2. Deadline based prioritization among both deadline and non deadline workflows is a best case effort towards meeting any deadlines.
3. Having dual deadlines causes more workflow execution incompletion. Either of the deadlines that is arrived first causes the workflow incompletion.

The measurements for this scheduler are the number of workflows failing to meet their deadlines, the distribution of the failure between two deadlines, and the overall impact on the average performance for both deadline and non deadline workflows.

For set I and II, the result of the various implementations towards meeting deadlines are shown in figure 7-14 and figure 7-15, respectively. These figures only include deadline

workflows and show the workflow completion status. A workflow has an incomplete status if it could not finish its execution due to either of the deadlines (ET or MEL).

Each implementation tends to improve over the previous implementation or perform the same as the previous implementation. As seen there is no significant difference between two sets across each implementation. The implementation deadline-3 improves the results for set I and does not have any improvements in case of set II. This is because there is less of a chance to find more resources with deadline-3, as the same breadth workflows overall consume services more evenly than the variable breadth workflows as in set I.

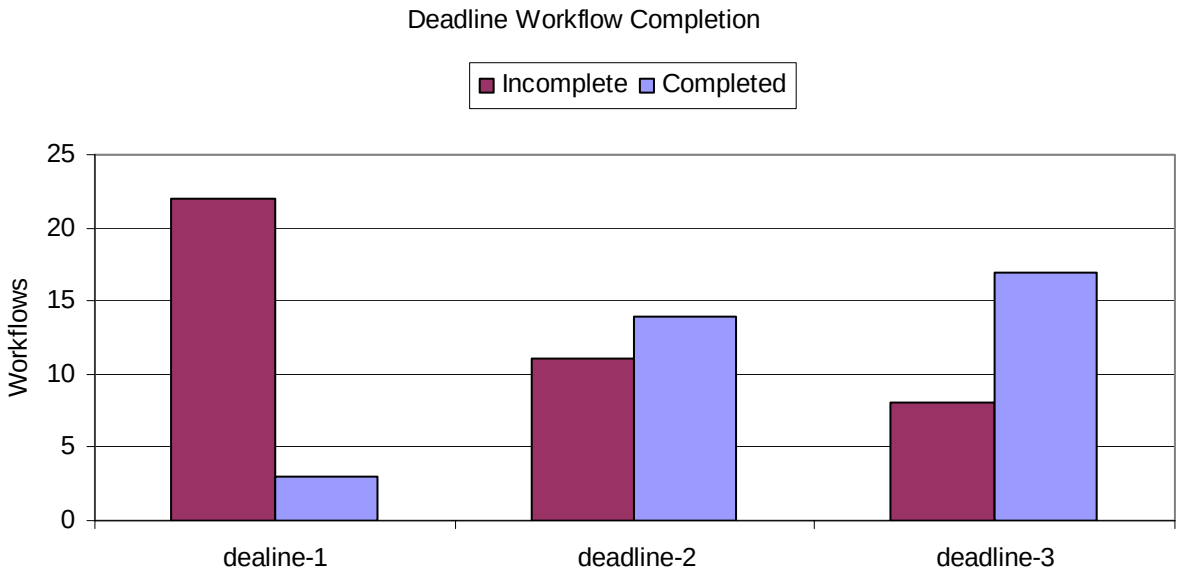

Figure 7-14. Performance of deadlines scheduler for Set I workflows

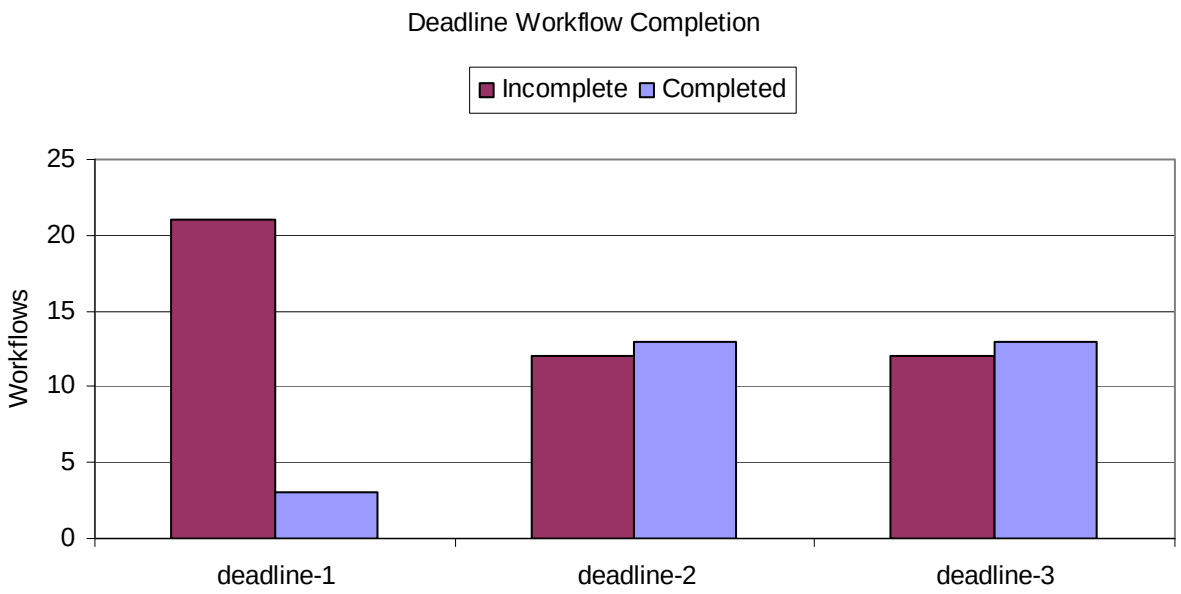

Figure 7-15. Performance of deadlines scheduler for set II workflows

The workflow incompletion is a result of failing to meet one of the two deadlines MEL or ET. The type of the deadlines that cause the workflow incompletion during the experimentation is shown in the figures 7-16 and 7-17. Both deadline-2 and deadline-3 for each set I, and II do not allocate any services to the workflows until the services necessary for MEL are available. Therefore, all of the workflow incompletion is a result of ET. Either workflows stay in the outstanding workflow queue and the ET arrives or they start executing too late to meet their ET.

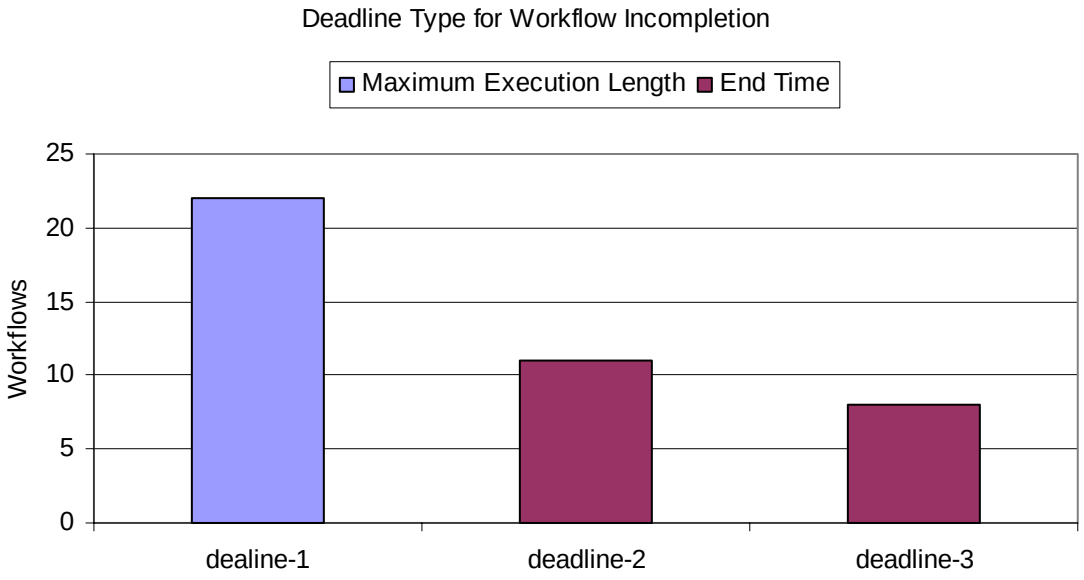

Figure 7-16. Set I workflow incompletion due to a certain deadline type

For set I, with deadline-1 all of the workflow incompletion is due to the MEL deadline. That is none of the workflows were allocated enough services that allowed them to satisfy their maximum execution lengths. For set II, deadline-1 has workflow incompletion due to both of the deadlines MEL and ET. Since workflow breadth is more evenly distributed in set II, some of the workflows are eventually allocated therefore, beating there MEL deadline but the wait time for these allocations was long enough to cause the ET deadline.

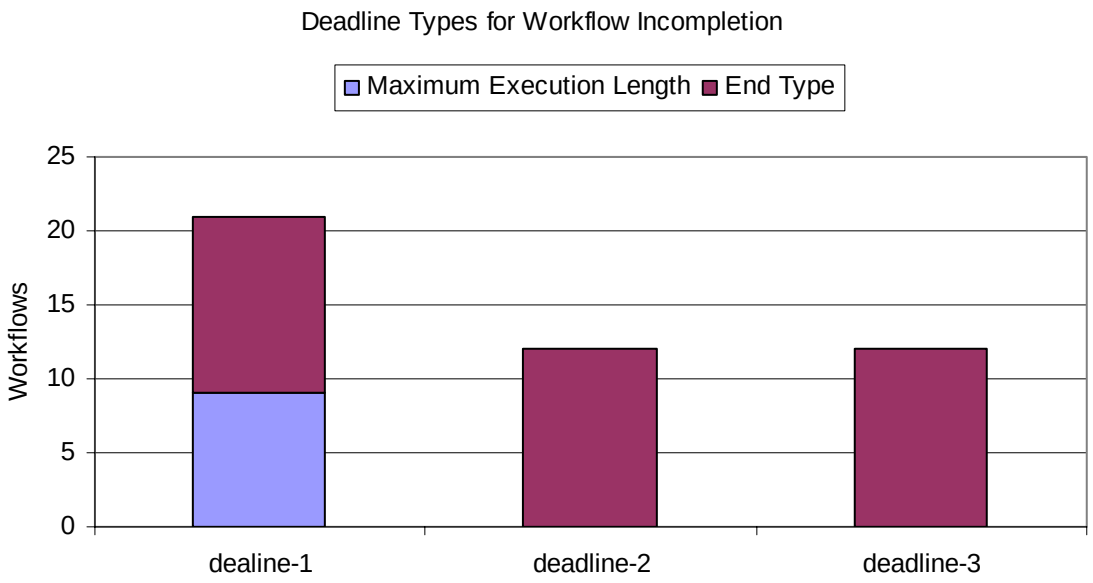

Figure 7-17. Set II workflow incompletion due to a certain deadline type

Lastly, the impact of each implementation on all of the workflows (deadline and non deadline) is shown in the figures 7-18 and 7-19. Deadline-1 has the best performance for both sets in terms of the average finish time for the workflows. However, it has the worse performance in terms of meeting the workflow deadlines as shown in figure 7-14 and 7-15. The impact of the deadline-2 and deadline 3 is similar with deadline 3 performing slightly better in some cases. For set I, the difference in performances are only for the larger (50% of the) workflows, where as with the set II, the performance difference is for most of the workflows except for the initial subset of the workflows that consumed most of the cloud services.

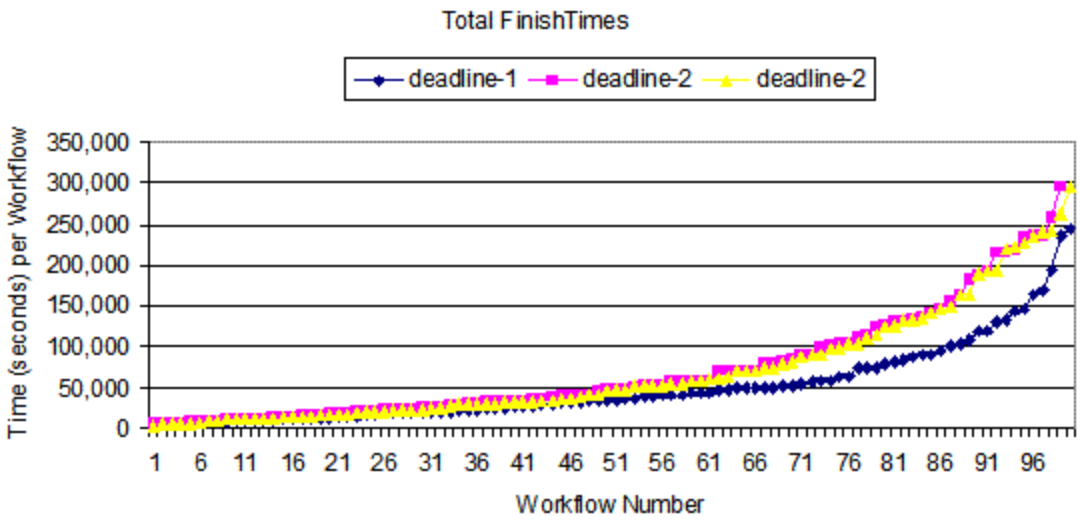

Figure 7-18. Set I workflow completion times (total finish times)

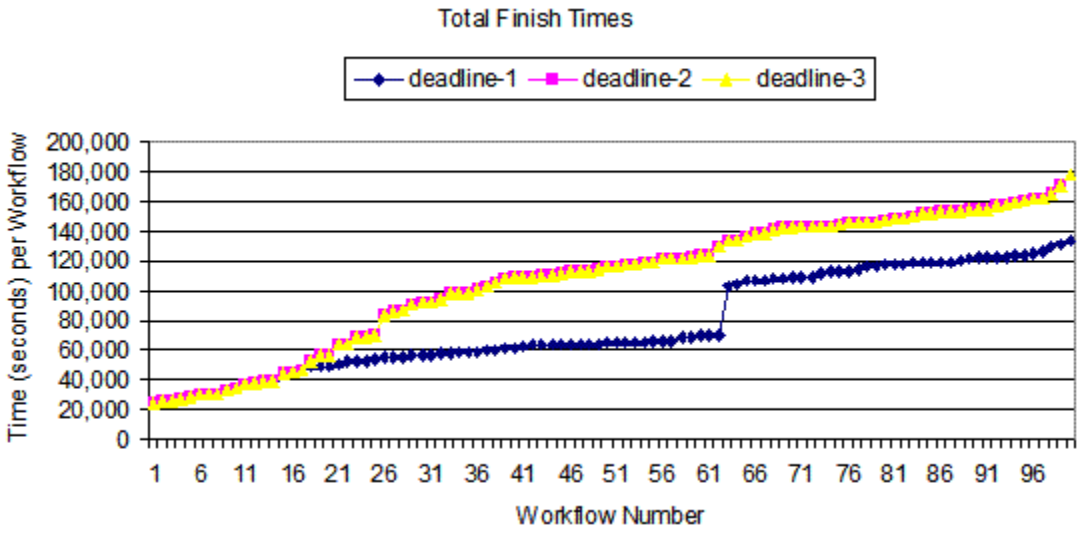

Figure 7-19. Set II workflow completion times (total finish times)

Summary

The obvious conclusion is that with the mixed mode scheduling of deadline and non deadline workflows, there is a better chance of achieving the deadlines if more services can be made available to the deadline workflows. That is by restricting the services to the non deadline workflows more services are available for the deadline workflows. However, the service

requirement distribution across deadline workflows in the system does play a role as well. For workflows with similar service requirements, there is more competition especially with the smaller cloud sizes.

In terms of which deadline caused a workflow to incomplete depends on the workflow arrival in the system. At the time of workflow arrival, a cloud may not have sufficient services to achieve both deadlines, resulting in the workflow termination. The workflows are not scheduled until they can be allocated enough services to guarantee their maximum execution lengths. This optimization is in place to save services when less than optimal service allocations for a workflow are only going to be wasted.

There is a tradeoff between the overall workflow (both deadline and non deadline) performance and the number of deadlines achieved. Given the service requirements distribution among workflows, the impact on workflow performance is felt by more workflows when they are equally competing for services.

CHAPTER 8
CONCLUSIONS AND FUTURE WORK

In this work detailed research is conducted towards workflow based application scheduling for a service oriented computing cloud (SOCC). The motivation behind this research is to identify and understand the scheduling issues while considering SOCC as a viable execution platform. For this purpose, a problem statement is formulated in chapter 2 to drive the overall research. In order to understand workflow scheduling within SOCC, all elements of the problem statement are explored. Workflow definition in terms of scheduling and execution over an SOCC is also formalized. Additionally a detailed survey is conducted to review all of the existing work in the related domains to establish the context for the research. Given the problem statement and the existing work survey, simulation based experimentation is conducted to implement the selected scheduling policies.

Research Results

The experimentation results reveal various considerations while scheduling different workflows within the SOCC based computing environment.

There are three scheduling policies implemented; *workflow prioritization, service partitioning,* and *deadlines*.

Four different variables are considered to represent the cloud configuration; *cloud size, workflow arrival, workflow structure,* and *workflow execution deadlines*.

The scheduling performance is evaluated for two metrics; *system throughput,* and *workflow performance*.

Workflow Prioritization

- System throughput is not affected by the workflow scheduling (execution) order. The overall amount of work is the same regardless of the execution order.

- Individual workflow performance significantly varies across various prioritization heuristics.

    - Heuristics considering the overall workflow size as opposed to the individual task sizes perform better.

    - Further, the heuristics that prefer the smaller size workflows have better performance when there are large number of smaller workflows and a small number of larger workflows.

    - The workflow wait time is the key factor for controlling the workflow performance. By reducing the wait time it is possible to achieve the better workflow performance.

- By increasing the cloud size, it is possible to eliminate the performance differences between various scheduling heuristics. With a large cloud size the scheduling order does not have any significant impact.

    - Workflow prioritization only matters in case of high load on the system. Under light workloads, the workflow prioritization does not matter.

- The workflow structure plays a role in scheduling performance as the structure determines the number of tasks that are ready for execution at anytime therefore, determining the number of workflows that can be scheduled simultaneously.

- For workflows with a large number of tasks, greedy scheduling seems to perform the best.

    - This also results in the maximum service utilization.

Service Partitioning

- For a large number of relatively small workflows, a more controlled scheduling such as service partitioning, which can fairly distribute the services among the workflows based on their resource requirement, is the best option.

    - For workflows with a large number of parallel tasks, service partitioning is a poor approach as it limits the number of services that can be allocated to parallel tasks.

- When the workflows in the system have similar service requirements, then allocating resources up to 50% of their optimal service requirements yields the best average performance. In case of workflows with varied service requirements, best average finish times are achieved by allocating more services to smaller workflows and fewer services to larger workflows.

- Service partitioning only matters in case of high load on the system. Under light workloads, service partitioning does not matter.

Deadlines

- A controlled service allocation can be used to achieve workflow deadlines. However, this approach is only suitable for workflows with a relatively small number of parallel tasks.
- More services can be added to help with meeting the workflow deadlines.
- In the case of both deadline and non deadline workflows, a service partitioning that is biased towards workflows with deadline helps with meeting the deadlines.
- Workflow arrival time into the system also affects the ability to meet workflow deadlines. Arrival at a busy time means there is a less likely chance that the workflow will meet its deadline.
- There is a trade off between the workflow performance and deadlines achieved. The average workflow performance is affected when deadline workflows are given preference over non deadline workflows.

In addition to the scheduling, various complimentary techniques, such as admission control and additional resource provisioning, are also possible to satisfy the workflow execution deadlines.

Future Work

The current research and results lay the ground work towards workflow execution within SOCC. In future, various extensions to the current research are possible to further enrich the problem domain.

As a next step, an actual implementation of a service oriented cloud system should be conducted. This implementation should highlight various implementation challenges that should be addressed in order to realize the benefits associated with such an execution environment. Once an actual SOCC is in place, concrete workflows representing actual work should be scheduled and executed. Execution of the concrete workflows will require some analysis of the workflows to extract the sufficient information necessary for their scheduling. It will be

necessary to develop techniques for such an analysis and it would be intriguing to find out what kind of information is feasible and useful for scheduling. Further, various service types and corresponding workflow task types should be introduced to further enrich the problem domain and get closer to real world execution scenarios.

Another important consideration is the transformation of the existing systems to the service oriented architecture. This transformation is necessary to gain benefits associated with the SOCC based execution model. Such a transformation for an existing system presents a significant investment and commitment and therefore directly impacts the adoption of the SOCC. In addition, the workflow based application development for a SOCC based environment also requires further research. There have been various workflow description languages and corresponding implementations available however a standardized approach is yet to become available.

Conclusions

There are various modalities of cloud computing emerging as the flexibility and power of such a computing model is realized. Cloud computing holds the promise to aggregate and offer software service points that can be consumed by interested entities. For academic and commercial organization this often means the outsourcing of some or all of their IT infrastructure and resources. At the same time, other entities can start to offer cloud services for parties interested in offloading their IT needs. However, general purpose cloud computing will take time to emerge. Over time more and more software services will become available within the cloud making it possible for organization to cut over to such services. However, in the meantime many independent, isolated and unique offerings of the cloud computing will continue to emerge and they will identify various integration points among clouds to deliver more general purpose

computing clouds. Enterprises can choose to implement their own clouds and at the same time consume third party cloud services when outsourcing is more feasible.

A good starting point for any enterprise looking to experience cloud computing benefits is to push its non-critical applications over to the cloud. This will allow enterprises to learn application development, deployment and maintenance over the cloud. If the cloud is maintained by a third party vendor, this will also be an opportunity for both parties to understand the application needs and develop cloud services to address such needs. Theoretically, cloud is able to support any type of application with some amount of development effort. In practice, migrating existing applications over to the cloud will take time as cloud implementations start to offer services meeting the application needs. For example, migration of an E-commerce application over to the cloud would require application re-factoring to invoke database and invoice processing requests via service calls as opposed to traditional approach of function calls.

In this research workflow oriented applications are executed over a computing cloud. These workflows varied in their structures and in terms of their execution lengths. Workflow based applications or independent task applications are most suited for cloud computing given the ease of task-to-service mapping. In other applications where there is lack of task boundaries and tasks are less autonomous more work is required to execute such applications over the cloud. A general purpose simulator developed as part of this research is used to evaluate various task-to-service mapping heuristics. A large set of execution scenarios is implemented to identify various system requirements such as size of the cloud, the work load patterns and the corresponding system performance, and the control over workflow execution. In addition workflow characteristics that are necessary or helpful for scheduling purposes are also identified.

Additionally, a detailed survey of scheduling approaches within existing systems and research is also presented for reference purposes.

In conclusion, service oriented cloud computing systems will continue to evolve and become more prevalent as they hold potential to resolve current distributed system problems. This trend will continue to prompt the development of various types of applications, tools, and supporting technologies to help with the system adoption.

REFERENCES

[1] I. Foster, C. Kesselman and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *The International Journal of High Performance Computing Applications*, vol. 15, pp. 200-222, 2001.

[2] M. P. Papazoglou, "Service-oriented computing: concepts, characteristics and directions," *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*, pp. 3-12, 2003.

[3] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris and D. Orchard. Web services architecture. Available: http://www.w3.org/TR/ws-arch/

[4] M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann and B. J. Kramer, "Service-oriented computing: A research roadmap," in *Service Oriented Computing (SOC); Dagstuhl Seminar Proceedings*, 2006,

[5] I. Foster, "Grid services for distributed system integration," *Computer*, vol. 35, pp. 37-46, 2002.

[6] S. C. Kendall, J. Waldo, A. Wollrath and G. Wyant, "A note on distributed computing," Sun Microsystems, Inc, Mountain View, CA, USA, 1994.

[7] G. Lawton, "Moving the OS to the Web," *Computer*, vol. 41, pp. 16-19, 2008.

[8] R. Ramakrishnan, "Cloud Computing¿Was Thomas Watson Right After All?" *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pp. 8-8, 2008.

[9] F. Sullivan, "I Wandered Lonely as a Cloud," *Computing in Science & Engineering*, vol. 10, pp. 88-88, 2008.

[10] I. Foster, C. Kesselman, J. M. Nick and S. Tuecke. (2002, June 22, 2002). The physiology of the grid: An open grid services architecture for distributed systems integration.

[11] H. Liu and D. Orban, "GridBatch: Cloud Computing for Large-Scale Data-Intensive Batch Applications," *Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on*, pp. 295-305, 2008.

[12] C. Moretti, J. Bulosan, D. Thain and P. J. Flynn, "All-pairs: An abstraction for data-intensive cloud computing," *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pp. 1-11, 2008.

[13] P. Lorincz, "Evolution of enterprise systems," in *Logistics and Industrial Informatics, 2007. LINDI 2007. International Symposium on*, pp. 75-80.

[14] G. C. Fox and D. Gannon, "Special Issue: Workflow in Grid Systems: Editorials," *Concurr. Comput. : Pract. Exper.*, vol. 18, pp. 1009-1019, 2006.

[15] Fangpeng Dong, Selim G. Akl, "Scheduling Algorithms for Grid Computing: State of the Art and Open Problems," *Technical Report NO. 2006-504, Queen's University, Canada*, 2006.

[16] F. Dong and S. G. Akl, "Two-Phase Computation and Data Scheduling Algorithms for Workflows in the Grid," *Icpp*, vol. 0, pp. 66, 2007.

[17] D. Fernandez-Baca, "Allocating Modules to Processors in a Distributed System," *IEEE Trans. Softw. Eng.*, vol. 15, pp. 1427-1436, 1989.

[18] Soo Hwan Yang, "Enter the grid introducing Oracle 10g," *Grid Economics and Business Models, 2004. GECON 2004. 1st IEEE International Workshop on*, pp. 157-172, 2004.

[19] K. Gor, D. Ra, S. Ali, L. Alves, N. Arurkar, I. Gupta, A. Chakrabarti, A. Sharma and S. Sengupta, "Scalable enterprise level workflow and infrastructure management in a grid computing environment," in *CCGRID '05: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 2*, 2005, pp. 661-667.

[20] S. Graupner, "Management middleware for enterprise grids," *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, vol. 1, pp. 8 pp., 2006.

[21] S. Graupner, "Policy-based resource topology design for enterprise grids," *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, vol. 1, pp. 390-397 Vol. 1, 2005.

[22] T. Boden, "The Grid Enterprise Structuring the Agile Business of the Future," *BT Technology Journal*, vol. 22, pp. 107-117, 2004.

[23] A. P. A. van Moorsel, "Grid, Management and Self-Management," *The Computer Journal*, vol. 48, pp. 325-332, January 1. 2005.

[24] C. H. Crawford, G. P. Bate, L. Cherbakov, K. Holley and C. Tsocanos, "Toward an on demand service-oriented architecture," *IBM Syst J*, vol. 44, pp. 81-107, 2005.

[25] I. M. Llorente, "A Grid Infrastructure for Utility Computing," *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2006. WETICE '06. 15th IEEE International Workshops on*, pp. 163-168, 2006.

[26] D. A. Menasce, "A framework for resource allocation in grid computing," *Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004. (MASCOTS 2004). Proceedings. the IEEE Computer Society's 12th Annual International Symposium on*, pp. 259-267, 2004.

[27] S. Elnikety, E. Nahum, J. Tracey and W. Zwaenepoel, "A method for transparent admission control and request scheduling in e-commerce web sites," in *WWW '04: Proceedings of the 13th International Conference on World Wide Web*, 2004, pp. 276-286.

[28] B. Urgaonkar and A. Chandra, "Dynamic provisioning of multi-tier internet applications," in *ICAC '05: Proceedings of the Second International Conference on Automatic Computing*, 2005, pp. 217-228.

[29] A. Afzal, "QoS-Constrained Stochastic Workflow Scheduling in Enterprise and Scientific Grids," *Grid Computing, 7th IEEE/ACM International Conference on*, pp. 1-8, 2006.

[30] Patel, Y. Darlington, J., "A novel stochastic algorithm for scheduling QoS-constrained workflows in a web service-oriented grid," in *Web Intelligence and International Agent Technology Workshops, 2006. WI-IAT 2006 Workshops. 2006 IEEE/WIC/ACM International Conference on*, pp. 437-442.

[31] S. J. Vaughan-Nichols, "New Approach to Virtualization Is a Lightweight," *Computer*, vol. 39, pp. 12-14, 2006.

[32] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield, "Xen and the art of virtualization," in *SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, 2003, pp. 164-177.

[33] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant and K. Salem, "Adaptive control of virtualized resources in utility computing environments," *SIGOPS Oper. Syst. Rev.*, vol. 41, pp. 289-302, 2007.

[34] J. Rolia, A. Andrzejak and M. F. Arlitt, "Automating enterprise application placement in resource utilities," in *DSOM*, 2003, pp. 118-129 ee = {http://sprngernk.metapress.om/openur.asp?genre=arte{\&}ssn=0302-9743{\&}oume=2867{\&}spage=118.

[35] A. Schrijver, *Theory of Linear and Integer Programming.* New York, NY, USA: John Wiley \& Sons, Inc, 1986,

[36] J. R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs.* Cambridge, MA, USA: MIT Press, 1994,

[37] D. Gmach, "Capacity Management and Demand Prediction for Next Generation Data Centers," *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pp. 43-50, 2007.

[38] J. Rolia, J. Pruyne, X. Zhu and M. F. Arlitt, "Grids for enterprise applications," in *JSSPP*, 2003, pp. 129-147 ee = {http://sprngernk.metapress.om/openur.asp?genre=arte{\&}ssn=0302-9743{\&}oume=2862{\&}spage=129.

[39] S. Singhal, "Quartermaster - a resource utility system," *Integrated Network Management, 2005. IM 2005. 2005 9th IFIP/IEEE International Symposium on*, pp. 265-278, 2005.

[40] J. P. And, "Quartermaster: Grid Services for Data Center Resource Reservation,"

[41] N. Oldham, K. Verma, A. Sheth and F. Hakimpour, "Semantic WS-agreement partner selection," in *WWW '06: Proceedings of the 15th International Conference on World Wide Web*, 2006, pp. 697-706.

[42] A. K. L. Wong and A. M. Goscinski, "Using an enterprise grid for execution of MPI parallel applications - A case study," in *PVM/MPI*, 2006, pp. 194-201 ee = {http://d.do.org/10.1007/11846802_31.

[43] Jia Hu, "Organizing Dynamic Multi-Level Workflows on Multi-Layer Grids for Developing e-Business Portals," *Web Intelligence, 2004. WI 2004. Proceedings. IEEE/WIC/ACM International Conference on*, pp. 777-778, 2004.

[44] I. Foster, "Globus toolkit version 4: Software for service-oriented systems," in *IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779*, 2005, pp. 2-13.

[45] A. A. Chien, "Architecture of the Entropia distributed computing system," *Parallel and Distributed Processing Symposium. , Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, pp. 55-59, 2002.

[46] G. Fedak, "XtremWeb: a generic global computing system," *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pp. 582-587, 2001.

[47] D. P. Anderson, "BOINC: a system for public-resource computing and storage," *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pp. 4-10, 2004.

[48] D. Kondo, M. Taufer, C. L. Brooks, H. Casanova and A. A. Chien, "Characterizing and evaluating desktop grids: an empirical study," *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, pp. 26, 2004.

[49] P. Domingues, "DGSchedSim: a trace-driven simulator to evaluate scheduling algorithms for desktop grid environments," *Parallel, Distributed, and Network-Based Processing, 2006. PDP 2006. 14th Euromicro International Conference on*, pp. 8 pp., 2006.

[50] I. T. Foster and A. Iamnitchi, "On death, taxes, and the convergence of peer-to-peer and grid computing," in *IPTPS*, 2003, pp. 118-128 ee = {http://sprngernk.metapress.om/openur.asp?genre=arte{\&}ssn=0302-9743{\&}oume=2735{\&}spage=118.

[51] E. Korpela, "SETI@home-massively distributed computing for SETI," *Computing in Science & Engineering*, vol. 3, pp. 78-83, 2001.

[52] R. Buyya, D. Abramson and J. Giddy, "A Case for Economy Grid Architecture for Service Oriented Grid Computing," *Ipdps*, vol. 02, pp. 20083a, 2001.

[53] P. Uppuluri, "P2P grid: service oriented framework for distributed resource management," *Services Computing, 2005 IEEE International Conference on,* vol. 1, pp. 347-350 vol.1, 2005.

[54] A. Luther, R. Buyya, R. Ranjan and S. Venugopal, "Alchemi: A .NET-based enterprise grid computing system," in *International Conference on Internet Computing,* 2005, pp. 269-278.

[55] A. Luther, R. Buyya, R. Ranjan and S. Venugopal, "Alchemi: A .NET-based Grid Computing Framework and its Integration into Global Grids," *CoRR,* vol. cs.DC/0402017, 2004.

[56] Jaesun Han, "Scheduling proxy: enabling adaptive-grained scheduling for global computing system," *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on,* pp. 415-420, 2004.

[57] SungJin Choi, "Volunteer availability based fault tolerant scheduling mechanism in desktop grid computing environment," *Network Computing and Applications, 2004. (NCA 2004). Proceedings. Third IEEE International Symposium on,* pp. 366-371, 2004.

[58] C. Anglano, "Fault-aware scheduling for Bag-of-Tasks applications on Desktop Grids," *Grid Computing, 7th IEEE/ACM International Conference on,* pp. 56-63, 2006.

[59] C. Anglano, "Improving the Performance of Fault-Aware Scheduling Policies for Desktop Grids (Be Lazy, Be Cool)," *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2007. WETICE 2007. 16th IEEE International Workshops on,* pp. 235-240, 2007.

[60] HongSoo Kim, "Agent-Based Autonomous Scheduling Mechanism Using Availability in Desktop Grid Systems," *Computing, 2006. CIC '06. 15th International Conference on,* pp. 174-179, 2006.

[61] A. Chakravarti, "Application-specific scheduling for the organic grid," *Cluster Computing, 2004 IEEE International Conference on,* pp. 483, 2004.

[62] Baohua Wei, "Scheduling independent tasks sharing large data distributed with BitTorrent," *Grid Computing, 2005. the 6th IEEE/ACM International Workshop on,* pp. 8 pp., 2005.

[63] J. A. Pouwelse, P. Garbacki, D. H. J. Epema and H. J. Sips, "The bittorrent P2P file-sharing system: Measurements and analysis," in *IPTPS,* 2005, pp. 205-216 ee = {http://d.do.org/10.1007/11558989_19.

[64] D. Kondo, "Resource Management for Rapid Application Turnaround on Enterprise Desktop Grids," *Supercomputing, 2004. Proceedings of the ACM/IEEE SC2004 Conference,* pp. 17-17, 2004.

[65] E. Deelman, "GriPhyN and LIGO, building a virtual data Grid for gravitational wave scientists," *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on,* pp. 225-234, 2002.

[66] M. A. Lewis, "Sleuthing out gravitational waves," *Spectrum, IEEE,* vol. 32, pp. 57-61, 1995.

[67] G. Allen, W. Benger, T. Dramlitsch, T. Goodale, H. Hege, G. Lanfermann, A. Merzky, T. Radke, E. Seidel and J. Shalf, "Cactus Tools for Grid Applications," *Cluster Computing*, vol. 4, pp. 179-188, 2001.

[68] I. Foster and D. Gannon, "The Open Grid Services Architecture Platform," 2003.

[69] H. Stockinger, "Defining the grid: a snapshot on the current view," *The Journal of Supercomputing*, vol. 42, pp. 3-17 ee = {http://d.do.org/10.1007/s11227-006-0037-9, 2007.

[70] J. Yu and R. Buyya, "A Taxonomy of Workflow Management Systems for Grid Computing," Apr. 2005.

[71] D. Thain, T. Tannenbaum and M. Livny, "Distributed computing in practice: the Condor experience." *Concurrency - Practice and Experience*, vol. 17, pp. 323-356, 2005.

[72] Condor Team. Dagman (directed acyclic graph manager).

[73] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob and D. S. Katz, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Sci. Program.*, vol. 13, pp. 219-237, 2005.

[74] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat and P. Li, "Taverna: a tool for the composition and enactment of bioinformatics workflows," *Bioinformatics*, vol. 20, pp. 3045-3054, November 22. 2004.

[75] R. D. Stevens, A. J. Robinson and C. A. Goble, "myGrid: personalised bioinformatics on the information grid," *Bioinformatics*, vol. 19, pp. i302-304, July 3. 2003.

[76] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. MellorCrumme, D. Reed, L. Torczon and R. Wolski, "The GrADS Project: Software Support for High-Level Grid Application Development," *Int. J. High Perform. Comput. Appl.*, vol. 15, pp. 327-344, 2001.

[77] J. Cao, S. A. Jarvis and S. Saini, "GridFlow: Workflow management for grid computing," C\&C Research Laboratories and NEC Europe Ltd., 2003.

[78] R. Buyya, "The Gridbus toolkit for service oriented grid and utility computing: an overview and status report," *Grid Economics and Business Models, 2004. GECON 2004. 1st IEEE International Workshop on*, pp. 19-66, 2004.

[79] T. Fahringer, R. Prodan, R. Duan, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H. Truong, A. Villazon and M. Wieczorek, "ASKALON: A grid application development and computing environment," in *GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, 2005, pp. 122-131.

[80] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte and S. Weerawarana, "Business Process Execution Language for Web Services (BPEL4WS 1.0)," *At Www*, vol. 106,

[81] F. Leymann and others, "Web Services Flow Language (WSFL 1.0)," 2001.

[82] S. Thatte, "XLANG: Web Services for Business Process Design," 2001.

[83] T. Fahringer, "Specification of grid workflow applications with AGWL: an Abstract Grid Workflow Language," *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on,* vol. 2, pp. 676-685 Vol. 2, 2005.

[84] S. A. McIlraith, "Bringing semantics to Web services," *Intelligent Systems,* vol. 18, pp. 90-93, 2003.

[85] S. Krishnan, P. Wagstrom and G. von Laszewski, "GSFL: A Workflow Framework for Grid Services," *Preprint ANL/MCS-P980-0802, Argonne National Laboratory, August,* 2002.

[86] D. Dyachuk, R. Deters, "Improving Performance of Composite Web Services International Conference on," *Service-Oriented Computing and Applications, 2007. SOCA '07. IEEE,* pp. 147-154, 19-20 June. 2007.

[87] D. Dyachuk, R. Deters, "Service level agreement aware workflow scheduling," in *Services Computing, 2007. SCC 2007. IEEE International Conference on,* 2007, pp. 715-716.

[88] D. Dyachuk, R. Deters, "Using SLA context to ensure quality of service for composite services," in *Pervasive Services, IEEE International Conference on,* 2007, pp. 64-67.

[89] Erradi, A. Maheshwari, P., "wsBus: QoS-aware middleware for reliable web services interactions," in *E-Technology, e-Commerce and e-Service, 2005. EEE '05. Proceedings. the 2005 IEEE International Conference on,* 2005, pp. 634-639.

[90] P. Siddhartha, R. Ganesan, S. Sengupta, "Smartware - A management infrastructure for web services," in *Proceedings of the 1st Workshop on Web Services: Modeling, Architecture and Infrastructure,* 2003, pp. 42-49.

[91] T. Phan, W. Li, "Heuristics-based scheduling of composite web service workloads," in *Proceedings of the 1st Workshop on Middleware for Service Oriented Computing (MW4SOC 2006),* 2006, pp. 30-35.

[92] G. B. Chafle, S. Chandra, V. Mann and M. G. Nanda, "Decentralized orchestration of composite web services," in *WWW Alt. '04: Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers \& Posters,* 2004, pp. 134-143.

[93] S. Zhang, Y. Zong, Z. Ding and J. Liu, "Workflow-oriented grid service composition and scheduling," in *ITCC '05: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II,* 2005, pp. 214-219.

[94] A. L. Rosenberg, "On scheduling mesh-structured computations for Internet-based computing," *Transactions on Computers,* vol. 53, pp. 1176-1186, 2004.

[95] A. L. Rosenberg, "Guidelines for scheduling some common computation-dags for Internet-based computing," *Transactions on Computers*, vol. 54, pp. 428-438, 2005.

[96] G. Malewicz, "On Scheduling Complex Dags for Internet-Based Computing," *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pp. 66-66, 2005.

[97] Y. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surv.*, vol. 31, pp. 406-471, 1999.

[98] A. Su, F. Berman and H. Casanova, "On the feasibility of running entity-level simulations on grid platforms," *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pp. 312-319, 2004.