AGH

University of Science and Technology in Krakow

Faculty of Computer Science, Electronics and Telecommunications

DEPARTMENT OF COMPUTER SCIENCE

MASTER OF SCIENCE THESIS

PIOTR BRYK

Computer Science

STORAGE-AWARE ALGORITHMS FOR SCHEDULING OF WORKFLOW ENSEMBLES IN CLOUDS

SUPERVISOR:

Maciej Malawski Ph.D

Krakow 2015

Aware of criminal liability for making untrue statements I declare that the following thesis was written personally by myself and that I did not use any sources but the ones mentioned in the dissertation itself.

. . . .

SIGNATURE

Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

Wydział Informatyki, Elektroniki i Telekomunikacji

KATEDRA INFORMATYKI

PRACA MAGISTERSKA

PIOTR BRYK
Informatyka

ALGORYTMY SZEREGOWANIA ZBIORÓW GRAFÓW ZADAŃ NA CHMURZE OBLICZENIOWEJ UWZGLĘDNIAJĄCE MAGAZYNOWANIE DANYCH

PROMOTOR:
dr inż. Maciej Malawski

Kraków 2015

Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie, i nie korzystałem ze źródeł innych niż wymienione w pracy.

. . . .

PODPIS

I would like to express my gratitude to Maciej Malawski, my supervisor, for his support, guidance and patience throughout the course of this thesis. I am also grateful to Gideon Juve and Ewa Deelman for sharing their expertise and providing insightful comments on my thesis.

Abstract

This thesis focuses on data-intensive workflows, and addresses the problem of scheduling workflow ensembles under cost and deadline constraints in Infrastructure as a Service (Infrastructure-as-a-Service) clouds. The previous research in this area ignores file transfers between workflow tasks, which, as we show, often have a large impact on workflow ensemble execution. In this thesis we propose and implement a simulation model for handling file transfers between tasks, featuring the ability to dynamically calculate bandwidth and supporting a configurable number of replicas, thus allowing us to simulate various levels of congestion. The resulting model is capable of representing a wide range of storage systems available on clouds: from in-memory caches (such as memcached), to distributed file systems (such as NFS servers) and cloud storage (such as Amazon Simple Storage Service or Google Cloud Storage). We observe that file transfers may have a significant impact on ensemble execution; for some applications up to 90% of execution time is spent on file transfers. Next, we propose and evaluate a novel scheduling algorithm that minimizes the number of transfers by taking advantage of data caching and file locality. We find that for data-intensive applications it perform better than other scheduling algorithms. Additionally, we modify the original scheduling algorithms to effectively operate in environments where file transfers take non-zero time.

The thesis is organized as follows. Chapter 1 introduces the topic of this thesis and explains basic concepts and terms used in following chapters. Chapter 2 presents related work and state of the art research. Chapter 3 describes the problem and aforementioned storage model. In Chapter 4 we introduce alterations to the algorithms presented in the base research. Chapter 5 contains a description of the evaluation procedure we employed and Chapter 6 discusses the results of our evaluation. Finally, Chapter 7 outlines general conclusions and explores possibilities for future work.

Contents

List of figures 3

List of tables 4

1. Introduction 5

1.1. Motivation 5

1.2. Scientific workflows 6

1.3. Cloud computing 7

1.4. Problem statement 9

1.5. Goals of the thesis 10

1.6. Organization of the Thesis 10

1.7. Summary 10

2. State of the art 11

2.1. State of the art review 11

2.2. Summary 12

3. Proposed storage and execution model 13

3.1. Execution model 13

3.2. Storage and file transfer model 14

3.3. Performance metric 18

3.4. Summary 18

4. Storage and locality-aware algorithms 19

4.1. Storage aware scheduling algorithms 19

4.2. File locality-aware scheduling algorithm 20

4.3. Summary 22

5. Evaluation procedure 24

5.1. Experiment setup 24

5.2. Summary 25

6. Discussion of results 26

6.1. No storage system ... 26
6.2. In-memory storage ... 27
6.3. Distributed storage system ... 30
6.4. NFS storage ... 34
6.5. Summary ... 34
7. Conclusions and future work ... 35
7.1. Conclusions ... 35
7.2. Future work ... 36
A. Source Code ... 37
B. Publications ... 38

List of Figures

1.1 Graphical representation of a scientific workflow for LIGO application. 7

3.1 State diagram of a Virtual Machine that is requested to execute a task. 15

3.2 Virtual Machines staging in files from storage systems with different number of replicas. 16

3.3 Sample storage bandwidth function. 17

4.1 Schedules produced by locality-aware algorithms with cache and with no cache. 22

6.1 Average number of completed workflows within budget and deadline constraints for scheduling algorithm families. 27

6.2 Average algorithm exponential score for SIPHT and CyberShake applications running in an environment with an in-memory storage system. 28

6.3 Average algorithm exponential score in the function of the number of replicas of in-memory storage system. 29

6.4 CyberShake ensemble execution generated by the DPDS algorithm on a cloud with distributed storage system. 30

6.5 Average exponential score for applications running in an environment with a distributed storage system. 31

6.6 Average number of completed workflows in an environment with a distributed storage system. . . 32

6.7 Exponential score in environment with distributed storage system and disabled Virtual Machine cache. 33

6.8 Average ratio of total time spent on transfers to total ensemble execution time, per application for distributed storage system. 33

6.9 Average cache bytes hit ratio per application for distributed storage system. 33

List of Tables

4.1 Summary of algorithms analyzed in this thesis. 23

5.1 Summary of input parameters for all experiments run 25

6.1 Average exponential score of all application and algorithm pairs in the NFS server storage experiment. 34

1. Introduction

This chapter presents introduction to this work. In Section 1.1 we describe main motivation behind research that we conducted in this thesis. Section 1.2 introduces the concept of scientific workflows. Similarly, Section 1.3 gives an overview of cloud computing characteristics and models. Section 1.4 defines the problem we tackle in the thesis. Main goals of the thesis are outlined in Section 1.5, while organisation of the document is presented in Section 1.6.

1.1. Motivation

Today, workflows are frequently used to model large-scale distributed scientific applications. They enable the expression of multi-step task workloads in an easy to understand, debug and maintain manner. By using scientific workflows multiple researchers can collaborate on the design of a single distributed application. This is because workflows are arranged as directed acyclic graphs (DAGs), where each node is a standalone task and edges are dependencies between the tasks. These dependencies are typically input/output files that need to be transferred between tasks. With the rise in popularity of workflows in the scientific community, specialized management systems have emerged to provide dedicated execution environments. For example Pegasus [23], which is used in a number of scientific domains like astronomy or bioinformatics, is a system that can execute workflows on desktops, clusters, grids or clouds. Another example is HyperFlow [11], which is a programming and workflow description model that makes it easy to manage and develop complex, distributed workflows. Executing workflows is a non-trivial task, especially on clouds where resource provisioning and deprovisioning, cost accounting, and resource setup must be taken into account. Additionally, large-scale computations are often composed of several inter-related workflows grouped into *ensembles* consisting of workflows that have a similar structure, but may differ in their input data, number of tasks, and individual task sizes.

In this thesis, we focus on *data-intensive* workflows. For example, Montage [31] is a software toolkit for constructing science-grade mosaics in the Flexible Image Transport System (FITS) format by composing multiple astronomical images. An example Montage workflow consists of 10,429 tasks, requires 4.93 CPU hours to execute, reads 146.01 GB of input, and writes 49.93 GB of output [34]. CyberShake [18] workflows, which are used to generate seismic hazard maps, are similarly data-intensive. A typical CyberShake run requires 9,192.45 CPU hours to execute and reads 217 TB of data [34]. In cloud environments, where typical sustained global storage throughput is in the order of 10-20 MiB/s [17], data transfer times are not negligible and may take a significant portion of total workflow execution time. Several studies confirm that the transfers may also comprise a significant amount of workflow execution cost [33, 44, 55]. Dedicated workflow-aware file systems have even been proposed to address this issue [21].

Although data transfers play an important role in the performance of workflows on clouds, most existing research work on scheduling does not address this issue adequately. Workflow scheduling algorithms often assume that data is transferred directly between execution units. This is the case for HEFT and other, similar heuristics [50, 9, 16]. However, in clouds global storage systems are the most popular means of file transfers [33]. Examples of such storage systems include object storage (e.g., Google Cloud Storage (GCS) [7]), shared NFS folders, parallel file systems (e.g., Lustre) or even in-memory data stores (e.g., memcached [25]). The goal of this work is to develop models of such storage systems and use them to evaluate scheduling algorithms.

1.2. Scientific workflows

Scientific workflows are directed acyclic graphs (DAGs) of tasks with additional metadata describing the tasks and runtime requirements. Figure 1.1 shows graphical representation of a LIGO application workflow. Workflows are frequently used to describe multi-stage computational or data processing applications. In particular, they are widely adapted in bioinformatics or cheminformatics domains. They allow for easy debugging, because every step in the computational process is an isolated task with clearly defined input and output data. With this characteristics workflow runtime environments, such as Pegasus [23], can distribute the workflow on distinct compute units, for example Virtual Machines or containers. Another important characteristics of workflows is that they facilitate collaboration between researchers from remote sites. This is because each task can be developed independently, as long as its input and output files are well defined.

Historically, workflows have been used in other domains, such as manufacturing or handling business processes. In those use cases workflows are sets of dependant activities or processes that are required to produce a service or product. Typically, flow charts are employed to visualize business processes, which is another popular way of graphical presentation of workflows. In manufacturing, workflows are used, for example, when processing parts through various stages of production. In business, workflows are used to formalize processes within an organization, for example gathering document approvals or sending emails.

Workflow management system (WMS) is a service that can run and monitor scientific workflows. The main goal of such system is to provide easy to use environment where users can submit their workflows for execution, and analyze results. WMS is responsible for scheduling workflows onto underlying infrastructure, ensuring Quality-of-Service (QoS) and managing fault tolerance. When a workflow task fails or underlying infrastructure resource (e.g., disk or CPU) breaks then it is the responsibility of WMS to reschedule task or heal failing resource. This makes it easy to execute very large scientific workflows which can run for days on multiple physical machines, because underlying runtime failures are handled by the system. Additionally, some workflow management systems facilitate application monitoring by providing interactive dashboards that show execution progress in real time.

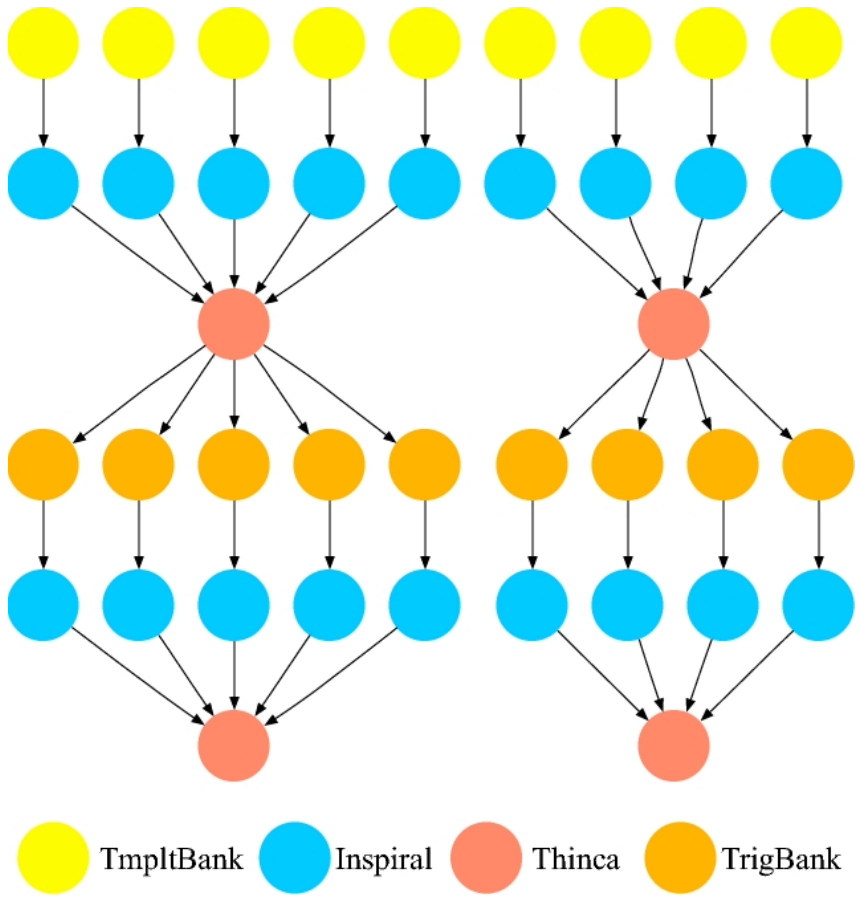

Figure 1.1: Graphical representation of a scientific workflow for LIGO application. Circles are tasks and arcs are dependencies between the tasks, realized as output file transfers.

Source: http://pegasus.isi.edu

This is an important characteristic, especially for long-running workflows, because the researcher that uses WMS can see what execution stage a workflow is at. Another important feature of modern workflow management systems is that they increase portability and reusability of workflows. This means user workflows can be run on variety of environments, without being changed. For example, the same workflow can be run on scientific grid or public cloud (e.g., PL-Grid [36], Amazon EC2 [1], Google Cloud [5], and others).

1.3. Cloud computing

Cloud computing is a popular term that describes variety of Internet services. US National Institute of Standards and Technology (NIST) describes cloud computing as a model for enabling on-demand and ubiquitous access to compute, storage and programming resources such as networks, servers, disks or application programming interfaces [43]. The burden of capacity management, hardware provisioning, data center management is offloaded to cloud infrastructure provider. Cloud computing has a few core characteristics that describe it. Furthermore it then can be divided into service and deployment models.

Key characteristics:

- *on-demand resource access* - users can provision resources on-demand, when they have a need for. For example, when web application usage grows, then system administrator can provision more web servers. Similarly, when resources are not needed anymore, they can be deprovisioned without additional costs.
- *network access* - resources can be accessed and managed through network to allow for easy service composability and reduce interaction with human operators.
- *shared resources* - cloud providers divide resources into pools that multiple customers share. For example, one physical machine can host many Virtual Machines for different customers or one physical disk can store data for different users. Such approach requires strict security model so that nefarious users cannot access others' data.
- *elasticity* - from perspective of the customer cloud appears as infinite source of resources that can be rented for desired amount of time and scale up/down according to actual usage.
- *easy maintenance* - cloud provider manages entire underlying infrastructure of the service to meet Service Level Agreement (SLA). For example, when physical disk breaks, then the provider seamlessly migrates the data that was on the disk and replaces it with new one.
- *usage-based billing* - cloud resources are billed for actual usage, for example for the time a Virtual Machine was running or the number of queries issued. This increases cost predictability as the consumer knows upfront what the cost of the service is.

Service models:

- *Software-as-a-Service (SaaS)* - consumer buys software that runs on cloud infrastructure and is managed by the cloud provider. The software is usually accessed by thin clients such as web browsers or application programming interfaces. Consumers do not control deployment of the software, they can only configure it through the means of administration console that is usually provided. Notable examples of SaaS offerings are Google Apps, Salesforce or Concur.
- *Platform-as-a-Service (PaaS)* - cloud provider gives consumers the capability to deploy their applications on a platform. The platform defines execution mode, programming interfaces, services and tools that can be used when creating applications. The consumer is not aware of the underlying infrastructure such as Virtual Machines or servers. Notable example of PaaS offering is Google App Engine where consumers can, for example, deploy Java applications that seamlessly scale up and down according to demand.
- *Infrastructure-as-a-Service (IaaS)* - consumers can provision low level compute, storage and network resources that can be used as building blocks for higher level services. Consumers have fine grained level of control over provisioned resources. For example, they can run arbitrary software on provisioned Virtual Machines. Higher flexibility does come with additional cost - resources have to be manually managed by consumers.

Deployment models:

- *Public cloud* - The cloud infrastructure is owned and managed by a cloud provider and is available for use by general public. The cloud provider is responsible for capacity planning for all shared resources and distributing them to clients. On the other hand, clients see public cloud as infinite pool of resources that can be provisioned on demand.

- *Private cloud* - The cloud infrastructure is deployed for exclusive use of a single customer. Resources in private clouds are not shared among different customers. Private clouds give customers the advantage of using cloud programming stack and job execution systems. This leads to better utilization of resources and higher maintainability of software.

- *Hybrid cloud* - The cloud infrastructure consists of both private and public resources. The two parts are separate, but are bound together by a software that allows their integration. Such model is often used to handle sensitive data on private cloud and non-sensitive on public cloud, or to move peak load of a web application to a public cloud.

1.4. Problem statement

This thesis builds upon the execution model, simulation procedure, and scheduling algorithms that were proposed in base research [41], which addresses the problem of workflow ensemble scheduling in Infrastructure-as-a-Service clouds with budget and deadline constraints. In that work, static (offline) and dynamic (online) scheduling and resource provisioning algorithms are proposed and analyzed with regard to various environment parameters, such as task runtime variance or Virtual Machine provisioning delay. However, there is an assumption that file transfer time between tasks is either negligible or included in task runtimes. While that assumption may be correct for some types of workflows, for data-intensive workflows it may lead to incorrect or overly-optimistic schedules.

In this thesis, we explore the area of workflow ensemble scheduling algorithms that are aware of the underlying storage architecture and can consider data transfers between tasks when scheduling ensembles. We want to determine how data transfers influence ensemble execution under budget and deadline constraints and how execution systems should handle data-intensive ensembles. In this thesis we develop and implement a global storage model for Cloud Workflow Simulator, which is used for transferring files between tasks. The model features the ability to dynamically calculate bandwidth and supports a configurable number of replicas, thus allowing us to simulate various levels of congestion in the system. Based on this model, we modify the original scheduling algorithms to take file transfers into account. We introduce a new File Locality-Aware scheduling algorithm that takes advantage of file caching to speed up ensemble execution.

1.5. Goals of the thesis

The main goals of this thesis are to develop a simulation model for storing and transferring files on Infrastructure-as-a-Service clouds and to develop a novel workflow scheduling algorithm that minimizes time spent on file transfers. The goals are accomplished by the following scientific contributions of this thesis:

- we provide a study on state of the art on the topic of cloud storage models and workflow scheduling algorithms,
- we define a simulation model for file storage and transfer on Infrastructure-as-a-Service clouds,
- we implement the file storage model in Cloud Workflow Simulator,
- we extend existing workflow scheduling algorithms to take advantage of our storage models,
- we develop a novel workflow scheduling algorithm that optimizes time spent on transferring files,
- we evaluate scheduling algorithms and asses their performance using various scientific application workflows,
- we discuss how various storage systems affect the execution of workflow applications on clouds.

1.6. Organization of the Thesis

The thesis is organized as follows. Chapter 2 presents related work and state of the art research. Chapter 3 describes the problem and introduces the aforementioned storage model. In Chapter 4 we introduce alterations to the algorithms presented in the base research. Chapter 5 contains a description of the evaluation procedure we employed and Chapter 6 discusses the results of our evaluation. Finally, Chapter 7 outlines general conclusions and explores possibilities for future work.

1.7. Summary

In this chapter we discussed motivation of the thesis, introduced problem statement and described our research goals. We introduced cloud computing and workflow concepts that are utilized in next chapters. We also described organization of the thesis.

2. State of the art

In this chapter we review current state of research in the topic of scheduling scientific workflows and ensembles in different environments, for example, on clouds or grids. We also review the problem of data transfers between workflow tasks and data-aware scheduling algorithms.

2.1. State of the art review

Many scientific applications are represented as workflows of tasks. As a result, workflow scheduling has become an important and popular topic of research. Workflow scheduling algorithms have been widely studied and there are numerous works on algorithms for scheduling single workflows onto generic execution units. This includes algorithms like Heterogeneous Earliest Finish Time (HEFT) [50], Predict Earliest Finish Time (PEFT) [9], Lookahead [16], and many others. While these algorithms provide good results for single workflows, they do not directly apply to the execution environment we consider in this thesis (Infrastructure-as-a-Service clouds), where compute resources can be provisioned and deprovisioned on demand.

There are also works that focus on the problem of workflow scheduling with only one constraint, for example, cost, deadline or storage space. Chen et al. [20] address the problem of scheduling large workflows onto execution sites while staying with storage constraints. The algorithm they propose minimizes runtime and does not consider resource cost. The authors of [46] introduce an algorithm for static cost minimization and deadline constrained scheduling. Their solution is very refined because they target realistic clouds with heterogeneous Virtual Machines and provisioning/deprovisioning delays. They also model file transfers between tasks as Peer-to-Peer messages. This is different from our approach where we employ global storage system for file transfers. We believe that global storage model better applies to Infrastructure-as-a-Service clouds, where providers offer storage services that are cheap and reliable. The Hybrid Cloud Optimized Cost (HCOC) scheduling algorithm [15] minimizes execution cost given deadline constraints. It supports clouds by provisioning resources and can handle multi-core Virtual Machines. This algorithm and its variations are different from our work because we consider scheduling of workflow ensembles with two constraints. There are other works that focus on scheduling ensembles with multiple constraints [24, 49], but their execution model differs from ours, where we schedule workflows according to their priorities.

The problem of data transfers between workflow tasks and storage, in general, has also been researched previously. Most of the time scheduling algorithms take generic communication cost into consideration [50]. Such algorithms do not directly apply to our execution model, where tasks do not communicate with each other, but stage files to and from global storage system. There are algorithms [44] that schedule workflows while minimizing storage usage. This is an interesting problem, but we do not consider it in this work. Juve et al. [33] evaluate data sharing options on Infrastructure-as-a-Service clouds. They use a very similar scheme for transferring files between workflow tasks, and their approach is implemented using a global storage system, however, they do not evaluate any data-aware scheduling algorithms. Nevertheless it reinforces that our execution model is valid and used in real-world applications.

Stork [37] is a data-aware scheduler that has been developed for efficient management of data transfers and storage space, but it does not address workflows or clouds. In [53] data clustering techniques are used for distributing datasets of workflow applications between cloud storage sites. In [19] an integrated task and data placement for workflows on clouds based on graph partitioning is derived, with the goal of minimizing data transfers. The approach to use data locality for efficient task scheduling is also widely applied to MapReduce, where various improvements over default Hadoop scheduling are proposed [29]. Bharathi et al. [14] analyze the impact of data staging strategies on workflow execution on clouds. They present decoupled, loosely-coupled and tightly-coupled models for transferring data between tasks, based on their interaction with workflow manager. They observe that decoupling file transfer tasks from computations can result in significant makespan reduction for data-intensive workflows. Ranganathan et al. [45] present data management techniques for computational jobs in grids. They use simulation procedures to analyze algorithms that place computational tasks on sites that already have input files for the tasks.

2.2. Summary

None of the aforementioned related work considers scheduling algorithms for workflow ensembles on Infrastructure-as-a-Service clouds that optimize data transfers and include a flexible data storage model. In this thesis we tackle this interesting research problem.

3. Proposed storage and execution model

In this chapter we introduce main assumptions of our application and environment model, provide details of the proposed storage model, and define the main performance metric that is used for evaluating and comparing workflow scheduling algorithms.

3.1. Execution model

The execution model used to evaluate our scheduling and provisioning algorithms is an extension of the model proposed in [41]. We introduce additions to support modelling file storage and transfers between workflow tasks.

A cloud consists of an unlimited number of homogeneous single-core virtual machines (Virtual Machines) that can be provisioned and deprovisioned at any time. The reason for such cloud model is the fact that studies say that there is typically one Virtual Machine type that is best suited for a particular workflow application [33]. There is a delay between the time a Virtual Machine provisioning (deprovisioning) request is sent and the Virtual Machine is available for execution. This is to account for typical startup and shutdown delays that are present in real-world public clouds [42, 30].

Virtual Machines can execute only one task at a time. A Virtual Machine not executing any task is called idle one. When a task is submitted to a non-idle Virtual Machine, it is enqueued for execution. When a task successfully finishes execution on a Virtual Machine, a queued task (if any) is dequeued and executed according to the FIFO rule. A Virtual Machine is charged $1 for each N-minute interval it has been running. Partial usage within a billing interval is rounded up. Unless specified otherwise, N is assumed to be 60 (one hour). Again, this is for simplicity and to reflect typical billing practises of cloud providers [32].

Target applications are ensembles of workflows (sets of workflows). Workflow tasks have runtime estimates that indicate how long it takes to execute them on a given Virtual Machine type. Unlike in the original model, the estimates do not include file transfers or any other kind of communication costs. Runtime estimates can often be obtained from preliminary runs of workflows [33, 51]. Some workflow applications, like Periodograms [13], even include an analytical performance model that estimates task runtimes. To account for the fact that actual task runtime depends on the dynamic state of the system it runs on (e.g., CPU cache, disk or network latencies), and often differs from predictions, we randomize runtimes under a uniform distribution of $\pm p\%$. Workflows have integral, non-negative priorities that indicate their importance for completion. The priority system is exponential, meaning that it is better to complete workflow with priority p than any number of workflows with lower priorities. This assumption reflects the reality of many scientific applications, for example CyberShake [39, 28] where it is better to compute seismic hazard at the location of a single tall building than for an entire unpopulated desert.

A task has zero or more input files that have to be staged entirely to a Virtual Machine before it can start executing the task. Similarly, the task has zero or more output files, which may be used as inputs to other task or be the end result of a workflow. File names are unique within a workflow, meaning that two files with the same names but referenced in different workflows are considered different. These assumptions are in line with real-world execution environments like Pegasus and its DAX file format for workflow representation [23]. Similar to runtimes, we assume file sizes can be estimated based on historical data. Files are assumed to be write-once, read-many (WORM), meaning that a file is never updated once it has been written.

The goal of scheduling and resource provisioning is to complete as many workflows as possible within the given deadline and budget constraints, respecting workflow priorities. A workflow is considered complete if all its jobs and file transfers have completed successfully within budget and deadline constraints.

The simulator has an event loop with a global timer. At the beginning of a simulation the event loop calls the initial scheduling procedures, sets a global timer to zero and then waits for incoming events. The simulation is considered completed when no more events are in the event loop. Each event is a quartet of: type, payload, destination and delay time t_d. When an event is sent at time t, it is received at the destination on time $t + t_d$. With such architecture we are able to simulate file transfer times, delays and varying task runtimes.

3.2. Storage and file transfer model

We propose a global storage model to be used for transferring input and output files between tasks, as well as for storing the final results of the application for retrieval. Each Virtual Machine has a local disk cache of files that were generated on the Virtual Machine or transferred to the Virtual Machine from the global storage system. To stage in (out) a file to (from) a Virtual Machine, a request has to be sent to the global storage management system. The system then transfers the file to (from) the Virtual Machine according to the dynamic state, configuration parameters, file size and file presence in the per-Virtual Machine local cache. We have chosen global storage model for file transfers instead of peer-to-peer transfers because such paradigm is widely used in grid environments in the form of simple NFS folders or shared distributed filesystems [47, 27]. What is more, Agarwal et al. in their research show that peer-to-peer data sharing in workflow applications does not perform well on typical public clouds [8]. Cloud providers offer globally accessible APIs for storing and retrieving files, for example Google Cloud Storage (GCS) [7]. Our model supports all aforementioned storage models, from the case of a single shared-disk to highly scalable distributed storage systems.

We assume that transfers between tasks always require uploading or downloading entire files. This is because most cloud storage systems are not POSIX compliant and they have little support for reading parts of files [54]. With this assumption we additionally offload part of the burden of file transfer optimization to workflow application

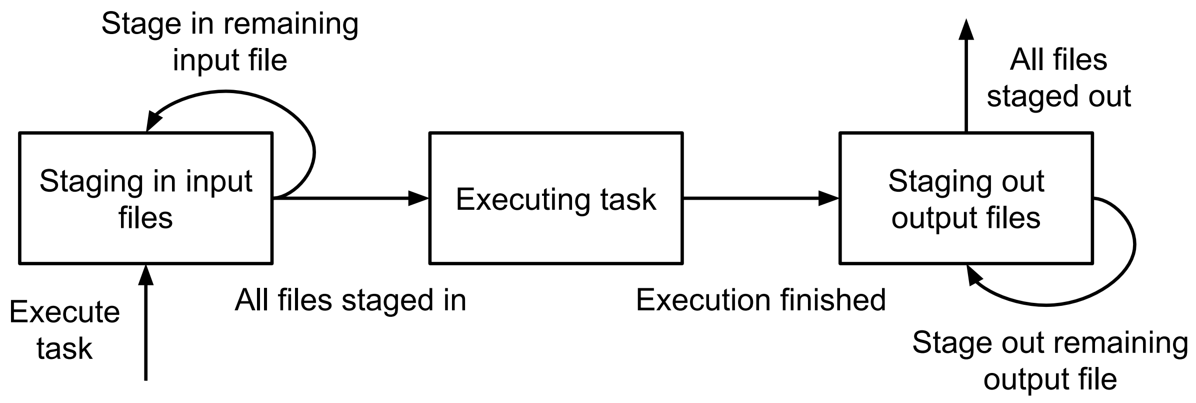

Figure 3.1: State diagram of a Virtual Machine that is requested to execute a task.

developers, who should design their applications so that input files are fully utilized to minimize unnecessary data transfers.

Although we acknowledge that cloud storage services are not free, we consider storage usage charges to be outside the scope of this thesis. This means that the amount of bytes stored and transferred does not affect the total cost of ensemble execution. The rationale behind such assumption is the fact that in many cases *all* input and output files are preserved for further analysis after workflow completion. What is more, the typical price for storing all of a workflow's files in the cloud for the duration of its execution is much smaller than the price for computing. For example, the Workflow Gallery [48] provides a sample Montage application consisting of 1000 tasks that executes in 3 hours 10 minutes and generates files with a total size of 4.2GiB. According to current Google Cloud pricing [6], it costs at least 460 times more to rent standard Virtual Machines for the computation (approx \$0.1995) than to store all files for the duration of the workflow's execution (approx \$0.00043). A study by Berriman et al. [12] provides similar numbers, showing that short-term storage is much cheaper than computation.

In the model we assume that before executing any task all its input files have to be fully staged in from the global storage to a Virtual Machine. Staging of input files occurs in sequence, i.e. input file transfer I starts only after transfer $I-1$ has successfully finished. This is done intentionally; in our congestion model parallel transfers would not provide any advantages for a single task, since the bandwidth of a link to a Virtual Machine is a limiting factor. More importantly, for transferring files to multiple tasks, starting simultaneous transfers could delay execution of those transfers that are almost complete. If a file requested for staging in has not been an output of an already completed task, it is assumed to be workflow input file that has been prepared and saved to the storage prior to the execution. Task output files are handled in a similar fashion. This process is outlined in the Figure 3.1. Similar approach has already been employed in real-world workflow applications, such as the one described in [33].

To model dynamic state of the global storage system we introduce the concept of *replicas* responsible for handling file operations. A replica is a storage device that contains copies of all files and fulfills file staging requests. The system is characterized by maximum read b_r and write b_w bandwidths that never change. The bandwidths are assigned to all the endpoints, so the congestion can occur on both Virtual Machine and replica endpoints, and they are identical for all replicas across the system. Read and write bandwidths are independent, meaning that reading does not affect writing and is processed separately. The storage system consists of r replicas. Simple NFS folders and RAID arrays are modeled by a low number of replicas, whereas scalable distributed storage services like Google Cloud Storage or Amazon Simple Storage Service are modeled by a larger number of replicas. The number of replicas is constant through the execution, meaning that only one storage type can be modelled at a time.

At any given point in time, all currently running staging requests for a given file are handled equally by all replicas. The replicas assign a fair share of their bandwidth to each request. A file cannot be transferred faster than a given maximum bandwidth, which is identical for replicas and Virtual Machines. The bandwidth accounts for both Virtual Machine limitations (e.g., network connectivity speed) and storage device characteristics (e.g., physical disk speed). The process of assigning dynamic bandwidth is outlined in the Algorithm 1 and Figure 3.2 shows sample states of the system.

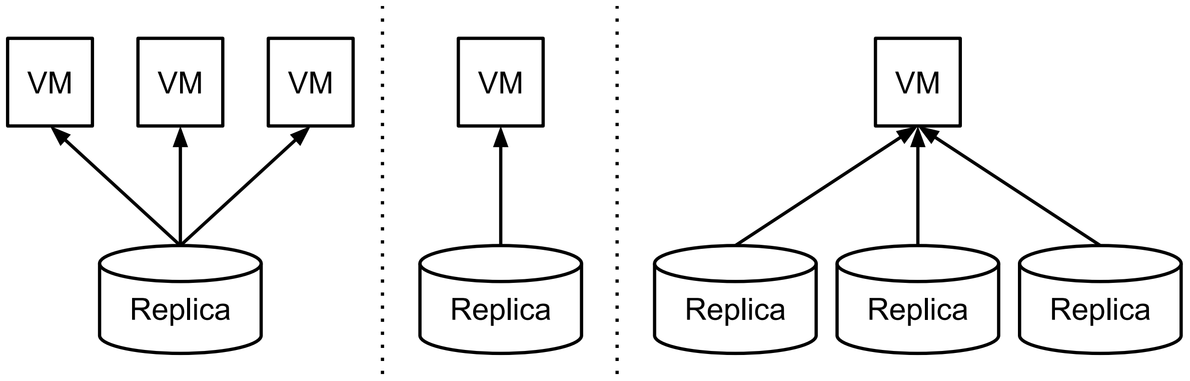

Figure 3.2: Virtual Machines staging in files from storage systems with different number of replicas. On the left diagram each Virtual Machine is assigned $\frac{1}{3}b_r$ bandwidth and on the center diagram the only one Virtual Machine is assigned bandwidth of b_r. The Virtual Machine on the right diagram is assigned bandwidth of b_r, as it cannot be higher, since the bottleneck is the Virtual Machine endpoint.

Each request to the storage system is handled with latency of l in milliseconds to account for any network and system delays. The latency parameter has the highest impact on execution of ensembles with lots of small files. Recent studies confirm that distributed storage systems have non-negligible latencies, even in the order of hundreds of milliseconds [35]. We do not include additional factors such as metadata overheads, because the storage is modelled to support only writing and reading files and we believe these factors would not influence the results significantly.

Read transfer finish time t_e of a file with size s that has started at the time t_s can be computed by solving Equation 3.1, where B_r is bandwidth assigned to the transfer, changing over time.

$$s = \int_{t_s}^{t_e} B_r \,\mathrm{d}t \tag{3.1}$$

Thanks to the fact that $B_r(t)$ is a discontinuous function with finite time intervals where it is defined by constant functions, we can precisely compute the time it takes to transfer a file. The computation algorithm works as follows. When a file transfer is started, its last modification time t_m is set to the current time and the bytes remaining to be

Algorithm 1 Dynamic bandwidth calculation in the model

```
Input: $b_r, b_w$: maximum read and write bandwidths; $r$: number of storage replicas
Output: $B_r, B_w$: dynamic read and write bandwidths
  procedure CALCULATE BANDWIDTH($b_r, b_w, r$)
      $T_r \leftarrow$ set of running read transfers
      $T_w \leftarrow$ set of running write transfers
      if $\overline{T_r} > 0$ then
          $B_r \leftarrow \text{MIN}(b_r r / \overline{T_r}, b_r)$
      else
          $B_r \leftarrow \emptyset$
      end if
      if $\overline{T_w} > 0$ then
          $B_w \leftarrow \text{MIN}(b_w r / \overline{T_w}, b_w)$
      else
          $B_w \leftarrow \emptyset$
      end if
  end procedure
```

transferred, s_d, is set to the file size. Then, for each file transfer the s_d variable is decreased by the number of bytes transferred since the last modification time, t_m, and t_m is set to the current time. Any transfer that has zero bytes remaining is removed. Next, compute lowest expected remaining transfer time, t_l, based on the current bandwidth and schedule the algorithm to be invoked again no later that t_l from the current time. The transfer update procedure is formally described in Algorithm 2.

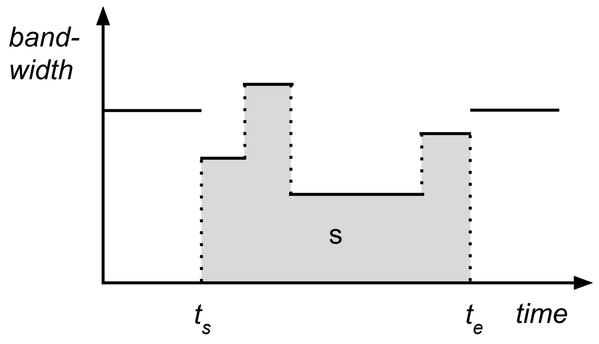

Figure 3.3: Sample storage bandwidth function with file transfer start t_s, finish t_e and its size s marked.

A Virtual Machine may have a local disk of size c bytes to serve as a file cache. We acknowledge that local disks are often billed separately from Virtual Machines, but for the scope of this thesis we assume that the price of the disk is included in the Virtual Machine cost. All the input and output files of tasks executed on the Virtual Machine are stored in the cache according to its policy. The files are cached in the order they were staged in to or out from the Virtual Machine. We have modelled the cache using a First In First Out (FIFO) policy [22]. The cache discards the least recently stored files first. When a file is required for task execution as an input and is available in the Virtual Machine cache, no staging request is issued and the file can be used immediately. The file caching system, if enabled, works passively during ensemble execution, i.e. no action is required from scheduling and provisioning algorithms to take advantage of it.

Algorithm 2 Updating file transfers in the simulator

```
Input: $B$: current read (write) bandwidth; $t$: current time
  procedure UPDATE_TRANSFERS($B$ ,$t$)
      $G \leftarrow$ set of running read (write) transfers
      for $g$ in $G$ do
          $t_m \leftarrow$ MODIFICATION_TIME($g$)
          $s_d \leftarrow$ REMAINING_BYTES($g$)
          REMAINING_BYTES($g$) $\leftarrow s_d - (t - t_m) * B$
          MODIFICATION_TIME($g$) $\leftarrow t$
      end for
      $t_n \leftarrow \inf$                                        ▷ Next scheduled update time
      for $g$ in $G$ do
          $s_d \leftarrow$ REMAINING_BYTES($g$)
          if $s_d = 0$ then
              $G \leftarrow G \setminus \{g\}$                    ▷ Remove finished transfer
          end if
          $t_p \leftarrow s_d / B$                                 ▷ Predicted remaining time
          $t_n \leftarrow$ MIN($t_n, t_p$)
      end for
      SCHEDULE_UPDATE($t_n$)              ▷ Schedule next update no later than $t_n$ from now
  end procedure
```

3.3. Performance metric

Scheduling algorithms are evaluated using the performance metric defined in previous research by Malawski et al [41]. The metric is: for a given ensemble e executed under budget b and deadline d, get all successfully completed workflows and sum their partial priority-based scores. The formal definition of the priority-based exponential score is:

$$Score(e, b, d) = \sum_{w \in Completed(e,b,d)} 2^{-Priority(w)} \tag{3.2}$$

The higher $Score(e, b, d)$ the higher rank of given ensemble execution. The performance metric is defined in this way to be consistent with our assumption that it is better to complete a workflow with priority p than any number of workflows with lower priority.

3.4. Summary

In this chapter we introduced the assumptions about the application and environment model, which is Infrastructure-as-a-Service cloud with homogeneous single core Virtual Machines. We explained the format of workflows that are used during simulation procedures. We defined storage and file transfer model that can simulate various cloud storage types (e.g., memory storage, distributed global storage or NFS folder). Finally, we outlined performance metric that is used to compare and evaluate scheduling algorithms.

4. Storage and locality-aware algorithms

In this chapter we start by describing our modifications to the base algorithms presented in Malawski's et al. work [41]. These modifications are designed to account for an execution environment in which file transfers take non-zero time. Such modified versions of the algorithms that are storage-aware are called Storage-Aware DPDS (SA-DPDS), Storage- and Workflow-Aware DPDS (SWA-DPDS) and Storage-Aware SPSS (SA-SPSS).

In addition, we describe a new scheduling algorithm that takes advantage of caches and file locality to improve performance. The new File Locality-Aware Scheduling Algorithm can be then combined with dynamic provisioning algorithms of DPDS and WA-DPDS. The resulting algorithms are called Dynamic Provisioning Locality-Aware Scheduling (DPLS) and Storage- and Workflow-Aware DPLS (SWA-DPLS).

4.1. Storage aware scheduling algorithms

The algorithms introduced in Malawski's et al. work [41] are responsible for two aspects: resource provisioning (creating instances of Virtual Machines) and scheduling (assigning tasks to Virtual Machines). The earlier developed algorithms include: Dynamic Provisioning Dynamic Scheduling (DPDS), Workflow-Aware DPDS (WA-DPDS) and Static Provisioning Static Scheduling (SPSS).

All the base algorithms use a runtime prediction function $R(t)$ to estimate how long it will take to execute a task t. The function is used, for example, to calculate the critical path length in static algorithms. This function is defined as follows:

$$R(t) = UniformDistribution(Runtime(t), p) \tag{4.1}$$

where $Runtime(t)$ is task's runtime defined in the workflow and p is uniform distribution radius in percent. We define a new prediction function, $R_s(t)$, that includes input and output file transfer times estimations as:

$$R_s(t) = R(t) + \sum_{f \,\in\, Files(t)} T(f) \tag{4.2}$$

where $Files(t)$ is a set of task input and output files and $T(f)$ is an optimistic transfer time prediction function. $T(f)$ is defined as:

$$T(f) = \begin{cases} \frac{Size(f)}{b_r}, & \text{if f is an input file} \\ \frac{Size(f)}{b_w}, & \text{otherwise.} \end{cases} \tag{4.3}$$

where b_r and b_w are maximum read and write bandwidths as defined in Section 3.2 and $Size(f)$ is the size of the file in bytes. We call this modified prediction function optimistic because it uses the maximum available bandwidth for estimating the transfer time. We acknowledge that this may be inaccurate because of congestion effects that may occur during ensemble execution. However, this is intentionally not optimized further, because we only need a lower bound on the transfer time. The modified versions of the algorithms that use the prediction function $R_s(t)$ are called Storage-Aware DPDS (SA-DPDS), Storage- and Workflow-Aware DPDS (SWA-DPDS) and Storage-Aware SPSS (SA-SPSS).

4.2. File locality-aware scheduling algorithm

The original dynamic (online) scheduling algorithms schedule workflows onto randomly selected, idle Virtual Machines. They do not exploit information about files that are already present in the Virtual Machine's local cache. As a result, when a task is submitted to run on a Virtual Machine there may exist another Virtual Machine where it would finish earlier by using cached data. Based on this observation, we have developed a novel dynamic scheduling algorithm that takes advantage of file locality to produce better schedules. The algorithm examines the Virtual Machines' caches at the time of task submission and chooses the Virtual Machine on which the task is predicted to finish earliest, according to runtime and file transfer time estimates.

The algorithm uses a modified task runtime prediction function, based on the one from Equation 4.2. The function ignores transfer time estimates for files that are already present on a Virtual Machine and it adds remaining runtime estimates for all tasks that are queued or currently running on the Virtual Machine. Queued tasks are considered because they are already scheduled for execution on the Virtual Machine.

$$R_{ncf}(t, vm) = R(t) + \sum_{f \,\in\, NCF(t,vm)} T(f) \tag{4.4}$$

$$R_{vm}(t, vm) = R_{ncf}(t, vm) + \sum_{s \,\in\, S(vm)} R_{ncf}(s, vm) - A(s) \tag{4.5}$$

where vm is a Virtual Machine and $NCF(t, vm)$ is the set of all input files of task t that are not present in the local file cache of vm (the set of not cached files). $S(vm)$ is a set of currently queued and running tasks on the Virtual Machine and $A(s)$ is current runtime of a task s, or none if not running. Such prediction function was designed to determine at what point in time task t is predicted to finish on, possibly non-idle, virtual machine vm. The predicted speedup S of a task t executing on vm is defined as the difference between the predicted runtimes from Equations 4.2 and 4.5.

$$S(t, vm) = R_s(t) - R_{vm}(t, vm) \tag{4.6}$$

This function allows us to estimate what is the speedup of running a task on a selected Virtual Machine that may contain cached input files. The speedup may, of course, be negative, meaning that the task will finish earlier on an idle Virtual Machine with empty cache than on the selected one. This may happen for example when the selected Virtual Machine executes a long-running task or its cache is empty.

Algorithm 3 File Locality-Aware Dynamic Scheduling

```
Input: $p_{max}$: maximal priority in the ensemble; $e$: ensemble to schedule
  procedure SCHEDULE($p_{max}$)
    for $0 \le i \le p_{max}$ do
      $P_i \leftarrow \emptyset$                                    ▷ List of ready tasks with priority $i$
    end for
    for root task $t$ in WORKFLOWS($e$) do
      $P_{\text{PRIORITY}(t)} \leftarrow P_{\text{PRIORITY}(t)} \cup \{t\}$
    end for
    while deadline not reached do
      $IdleVirtualMachines \leftarrow$ set of idle Virtual Machines
      $P_{highest} \leftarrow$ non-empty $P_i$ with lowest $i$, or $\emptyset$ it none
      if $P_{highest} \neq \emptyset$ and $IdleVirtualMachines \neq \emptyset$ then
        $VirtualMachines \leftarrow$ set of running Virtual Machines
        $b_s \leftarrow -\inf$                                      ▷ Highest speedup
        $b_t \leftarrow null$                                       ▷ Task of the highest speedup
        $b_{vm} \leftarrow null$                                    ▷ Virtual Machine of the highest speedup
        for $t$ in $P_{highest}$ do
          for $VirtualMachine$ in $VirtualMachines$ do
            $s \leftarrow S(t, VirtualMachine)$                     ▷ Equation 4.6
            if $s > b_s$ then
              $b_s \leftarrow s$
              $b_t \leftarrow t$
              $b_{vm} \leftarrow VirtualMachine$
            end if
          end for
        end for
        SUBMIT($b_t, b_{vm}$)
        Update $P_i$ sets once $b_t$ finishes
      end if
    end while
  end procedure
```

The scheduling procedure is shown in the Algorithm 3. At the beginning, a family of sets P_i containing ready tasks with priority i is created. A task is called ready when all its input dependant tasks have successfully completed. Root tasks of a workflow are tasks that have no input dependencies. Root tasks, by definition, are always ready and are initially added to their respective P_i sets. The algorithm operates until the deadline is reached and schedules tasks when there is at least one idle Virtual Machine and there is at least one ready task. The condition about existence of at least one idle Virtual Machine was introduced to have at least one *good* candidate Virtual Machine, thereby avoiding queuing many tasks onto a single Virtual Machine. Without this condition all ready tasks would be immediately scheduled onto, most likely non-idle, Virtual Machines, making the algorithm essentially a static one. In the task submission block, the algorithm computes the predicted speedup for each pair of: ready task with highest priority, and possibly non-idle Virtual Machine. Then the the pair of task and Virtual Machine with highest speedup is selected for submission. Once the task finishes, its ready children are added to the P_i sets, according to their priorities.

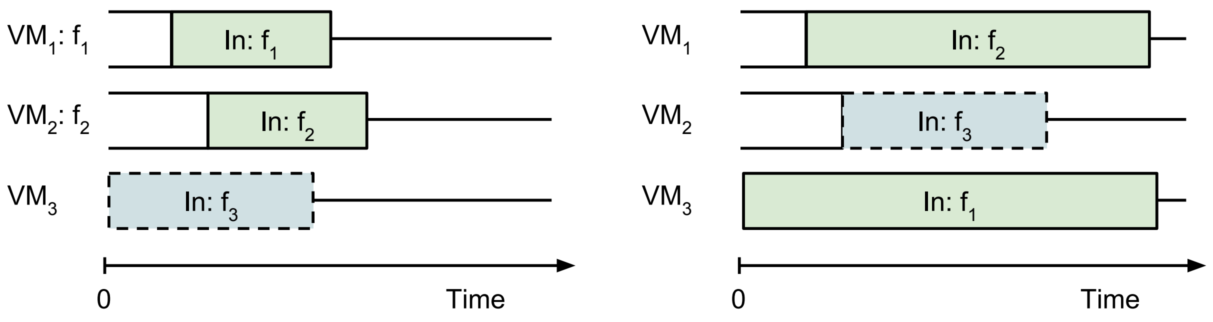

Figure 4.1: Schedules produced by locality-aware algorithms with cache (left) and with no cache (right). At time 0 $VirtualMachine_1$ and $VirtualMachine_2$ are busy. There are also three ready tasks, the one with dashed border has lower priority. On the left, $VirtualMachine_1$ and $VirtualMachine_2$ contain respectively f_1 and f_2 files in their cache. Higher priority tasks are submitted to non-idle Virtual Machines, which makes them start later than lower priority task. However, at the end they finish earlier than without cache as shown on the right.

Locality-aware scheduling algorithm ensure that ready tasks with highest priority are submitted to Virtual Machines in the order of their predicted speedup ranking. Lower priority tasks are always deferred when there is at least one higher priority ready task waiting for execution. A lower priority task submitted to an idle Virtual Machine, however, may start execution earlier than higher priority task that was submitted to a non-idle Virtual Machine. This is schematically explained in the Figure 4.1. Additionally, with Virtual Machine caching disabled, idle Virtual Machines are always chosen for task submission, meaning that this procedure reduces to the dynamic scheduling algorithm defined in [41]. Finally, the dynamic provisioning algorithms from [41] can be used together with the locality-aware scheduling procedure. The resulting algorithms are called Dynamic Provisioning Locality-Aware Scheduling (DPLS) and Storage- and Workflow-Aware DPLS (SWA-DPLS).

4.3. Summary

In this chapter we presented our modifications to base algorithms from Malawski's et al. work [41]. We also showed a novel workflow scheduling algorithm that optimizes file transfer to Virtual Machines to produce schedules that minimize budget consumed and execution makespan. The summary of algorithms and their characteristics is presented for convenience in Table 4.1.

	Scheduling	Provisioning	Workflow-Aware	Storage-Aware	File Locality-Aware
SPSS	static	static	+	–	–
DPDS	dynamic	dynamic	-	–	–
WA-DPDS	dynamic	dynamic	+	–	–
SA-SPSS	static	static	+	+	–
SA-DPDS	dynamic	dynamic	-	+	–
SWA-DPDS	dynamic	dynamic	+	+	–
DPLS	dynamic	dynamic	-	+	+
SWA-DPLS	dynamic	dynamic	+	+	+

Table 4.1: Summary of the algorithms. Scheduling refers to the process of assigning tasks to Virtual Machines, while provisioning is the process of creating and terminating Virtual Machine instances. Workflow-aware algorithms use the information on the workflow structure in their decisions. Storage-aware algorithms take into account the estimated data transfer times when scheduling tasks. File locality-aware algorithms use the new scheduling algorithms that takes the advantage of caches and file locality to improve performance.

5. Evaluation procedure

In this chapter we describe in details the evaluation procedure we used to validate algorithms and simulator implementation. We specify default parameters that were used during simulation runs and give an overview of workflow applications used during experiments.

5.1. Experiment setup

The algorithms were evaluated using the Cloud Workflow Simulator [4]. This simulator supports all of the assumptions that were stated in the problem description in Chapter 3. The simulator was extended to support the storage model introduced in this thesis. New scheduling algorithms were developed and the original algorithms were extended, as described in Chapter 4. Our intent was to focus thoroughly on simulation studies. We simulated hundreds of thousands of ensemble executions with various parameters, which would be infeasible to run on real testbed.

We evaluated the algorithms using ensembles consisting of synthetic workflows from the Workflow Gallery [48]. We have selected workflows representing several different classes of applications [34]. The selected applications include: CyberShake [39], a data-intensive application used by the Southern California Earthquake Center to calculate seismic hazards, Montage [31], an I/O-bound workflow used by astronomers to generate mosaics of the stars, and Epigenomics and SIPHT (sRNA Identification Protocol using High-throughput Technology) [38], two CPU-bound bioinformatics workflows.

The ensembles used in the experiments were created from randomly selected workflows from one application. Their sizes (number of tasks) were chosen according to a modified Pareto distribution described in [41]. The distribution selects large workflows (of size greater than 900 tasks) with a slightly higher probability than the standard Pareto distribution. The priorities are assigned to workflows according to their size: the larger the workflow is, the higher priority it has. This strategy aims to reflect the typical scientific ensemble structure, where there are a few large and important workflows and many small workflows of lower importance. Unless specified otherwise, we used ensembles with 20 workflows.

For each experiment run, the maximum and minimum viable budget and deadline were computed. The exact calculation procedure is described in [41]. 10 evenly chosen points in the interval between the minimum budget and them maximum budget, and 10 evenly distributed points in the interval between the minimum deadline and

	No storage	In-memory	Distributed	NFS
Read/write bandwidth	∞	100 MiB/s	10 MiB/s	20 MiB/s
Operation latency	0	1 ms	50 ms	200 ms
Local Virtual Machine cache size	0	50 GiB		
Number of replicas	∞	1, 2, 5, 10, 50	∞	5
Virtual Machine provisioning time	120 s			
Virtual Machine deprovisioning time	60 s			
Runtime variance	$\pm 5\%$			
Ensemble size	20, 50	20		
Budgets/deadlines	10			

Table 5.1: Summary of input parameters for all experiments run. Columns headers depict storage type modeled in an experiment while row headers present input parameter name. The rationale and sources for input parameter values is described in detail in the experiment chapter.

the maximum deadline, were used to run 100 simulations with different pairs of deadline and budget.

Unless specified otherwise, the simulations were run with task runtime estimation uncertainties of $\pm 5\%$. This number was chosen to reflect the fact that estimates in the real-world are not perfect. The Virtual Machine provisioning delay parameter was set to 120 seconds, which is typical of what can be observed in public clouds [42]. The Virtual Machine deprovisioning delay was set to 60 seconds to take into account any cleanup and shutdown operations.

5.2. Summary

In this chapter we defined simulation procedure that was used during this research. Table 5.1 summarizes the input parameters for all of the experiments to follow. The experiments model cloud environments with four different storage systems: infinitely fast storage, in-memory storage, distributed storage and an NFS-like file system.

6. Discussion of results

In this chapter we analyze the relative performance of our proposed scheduling algorithms on clouds with different storage system configurations. This is, from fast, in-memory storage systems, through distributed file systems, to low-performance NFS servers. The analysis consists of running simulation experiments and then interpreting aggregate results. Additionally, we investigate how those storage systems affect the execution of ensembles.

6.1. No storage system

In the first experiment we evaluated the performance of our algorithms in an environment where there is actually no storage. This is equivalent to having storage system with infinite read and write bandwidths and zero latency. In such a system, all file transfers finish instantaneously. This setup allows us to compare Storage- and File Locality-Aware versions of the algorithms with their unaware counterparts and determine whether there were any performance regressions introduced with our modifications.

The experiment consists of 100 simulations (10 budgets and 10 deadlines) for each application ensemble consisting of 50 workflows (5 applications), each scheduling algorithm variant (7 algorithms). The experiment was repeated 10 times with different randomly-chosen ensembles and all results were aggregated into one dataset. This represents a total of 35,000 simulation runs.

Figure 6.1 shows the average number of completed workflows by normalized deadline. Deadlines are normalized to the range from 0 to 1 for each experiment run. Only the results for CyberShake are included here because they are similar for the other applications.

The performance of the Storage- and File Locality-Aware scheduling algorithms was identical to their unaware counterparts, with very minor differences resulting from randomization. This is visible in Figure 6.1, where the lines for all the algorithms overlap almost perfectly. This is consistent with our expectations, given that the storage-aware algorithms are designed to reduce to the storage-unaware procedures when the storage is infinitely fast. It is also worth noting that the results presented here are similar to the ones from [41], which confirms that our experiment setup is correct.

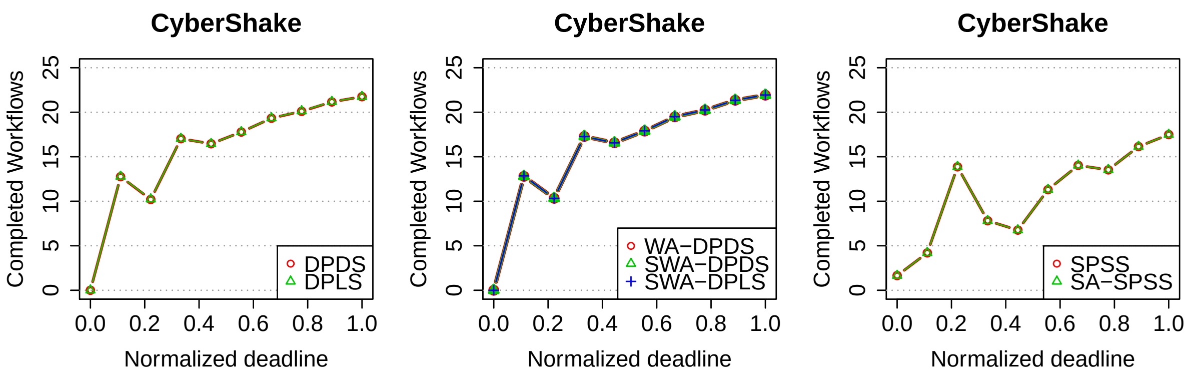

Figure 6.1: Average number of completed workflows within budget and deadline constraints for scheduling algorithm families. The underlying storage system is infinitely fast.

6.2. In-memory storage

In-memory storage storage systems have been known for their performance and high throughput for a long time [26]. They are widely used in response time-critical applications, e.g., in telecommunications applications. Recently, they have become popular for caching files in high traffic web applications[52], for example using the memcached distributed cache system [25]. Such systems can be also used for storing and transferring workflow's input and output files. In our experiment, the in-memory storage system is modelled with maximum read and write bandwidths of 100 MiB/s. This number reflects the approximate upper limit of throughput for the Gigabit Ethernet networks that are common in commercial clouds. Latency is set to 1ms, because memory storage systems are often simple key-value arrays that are bounded primarily by network delays. The experiments were run with 1, 2, 5, 10 and 50 as the number of replicas. Again, this is to model typical in-memory storage systems that can easily scale by adding nodes that replicate data. The local Virtual Machine cache size was set to to 50 GiB, as this is a typical local disk size of a public cloud Virtual Machine, for example of m3 or c3 instance types on Amazon EC2 [2].

Figure 6.2 shows the results of 70,000 simulation runs (10 deadlines x 10 budgets x 4 applications x 7 algorithms x 5 replicas x 5 experiments). The Y-axis represents the average exponential score from definition 3.2, while the X-axis represents normalized deadline as introduced Section 6.1. The first observation is that the score is always zero for minimal normalized deadline. This is expected, because deadlines and budgets are computed using estimates of task computation time only, and are set to barely allow for the execution of the smallest workflow with low safety margins. Knowing that file transfers take non-zero time it is expected that hardly any workflow can successfully complete within the minimal deadline.

The performance of the dynamic algorithms varies slightly, depending on whether they are aware of the existence of underlying storage or not, and on the type of application. This is because they are, by design, able to adapt to a changing environment and are resilient to uncertainties in tasks runtimes. The two applications shown in Figure 6.2 are both data-intensive applications. For CyberShake the results are very consistent across algorithms because it contains a low number of potentially cacheable files, resulting in less advantage for the locality-aware

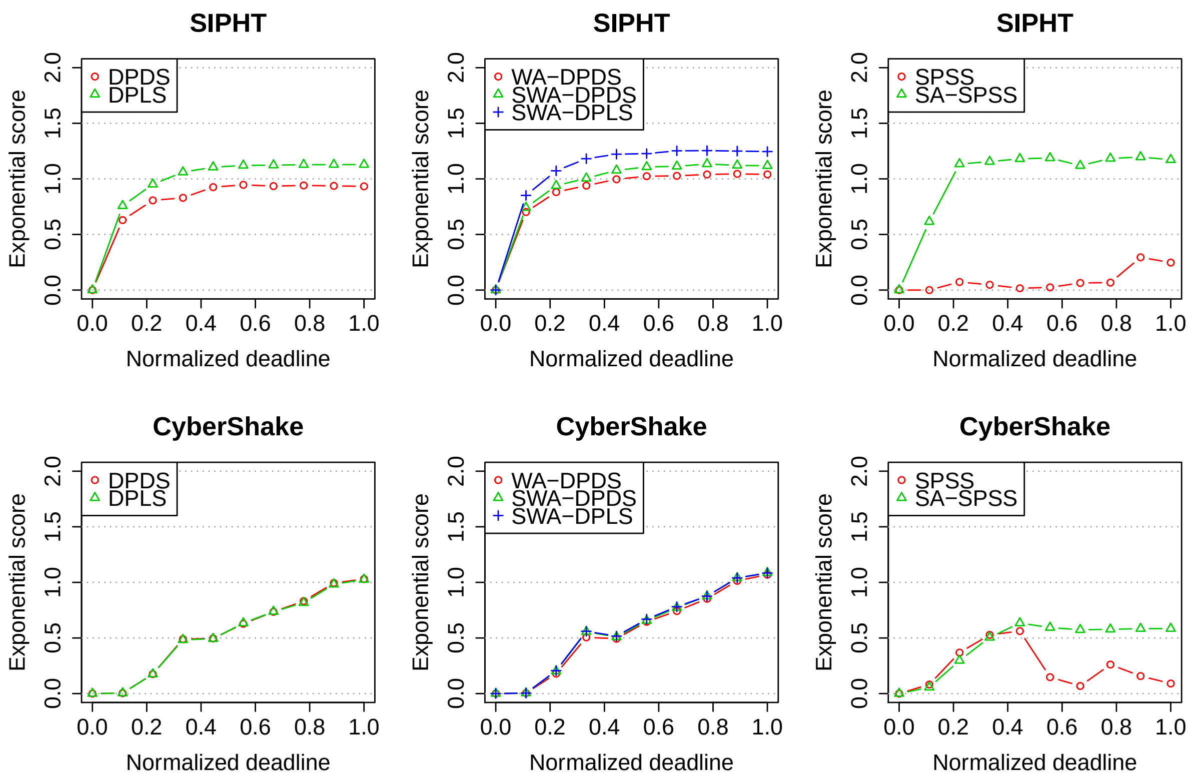

Figure 6.2: Average algorithm exponential score for SIPHT and CyberShake applications running in an environment with an in-memory storage system.

algorithms. Other data-intensive applications, such as SIPHT, often exhibit larger differences between the algorithms. With growing deadline, DPLS performs noticeably better than DPDS, because, as the deadline grows, there is more opportunity to leverage the local Virtual Machine cache and thus improve the performance. The family of Workflow-Aware algorithms produce similar results. SWA-DPLS outperforms other algorithms for the SIPHT application, while for CyberShake it is only slightly better. Also, SWA-DPDS is better than WA-DPDS because it is able to admit or reject workflows more accurately.

The static algorithms, on the other hand, show large differences in performance between storage-aware and unaware variants. Figure 6.2 shows that for CyberShake application the performance of SA-SPSS is better than SPSS. This is even more visible for SIPHT where SA-SPSS always produces better schedules than SPSS. Fragility with respect to runtime uncertainties is the main reason behind this behavior. Static algorithms rely on accurate task runtime estimates, and when transfer times are not included, the schedule plan degrades considerably. During the planning phase the static algorithm tries to squeeze as many workflows as possible within the budget and deadline constraints, leaving no safety margins. Therefore, when the runtimes are underestimated (i.e. no transfers included) most workflows that would normally finish just before the deadline are considered failures. Finally, one can observe that, as the deadline for CyberShake application grows, the score for SPSS gets lower. The reason is that SPSS begins to admit the highest priority workflow (which is very large) for execution at around a normalized deadline of 0.45. However, due to underestimates of task runtime, the execution of that workflow fails and the score becomes significantly lower.

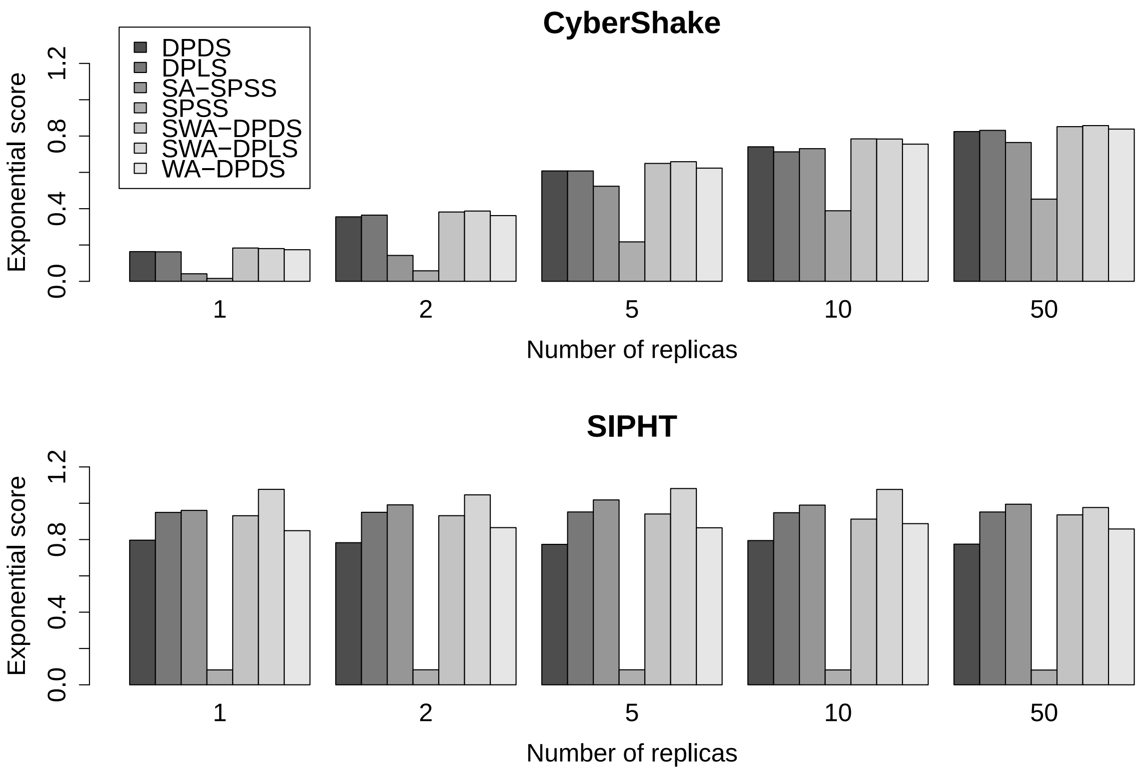

Figure 6.3: Average algorithm exponential score in the function of the number of replicas of in-memory storage system.

Figure 6.3 shows algorithm performance as a function of the number of replicas. For SIPHT, all algorithms perform approximately the same, regardless of the number of replicas. This application exhibits low parallelism, therefore increasing the number of storage replicas affects its execution only a little. CyberShake yields the opposite result. There is high correlation between the number of replicas and average score for all algorithms. The root cause behind this effect is that CyberShake workflows start with tasks that require very large input files, with size in the order of 20GB. Scheduling algorithms often start many workflows at a time, which results in many parallel transfers. This situation is illustrated in Figure 6.4. Finally, parallel transfers take less time when there are more replicas, which is consistent with the results.

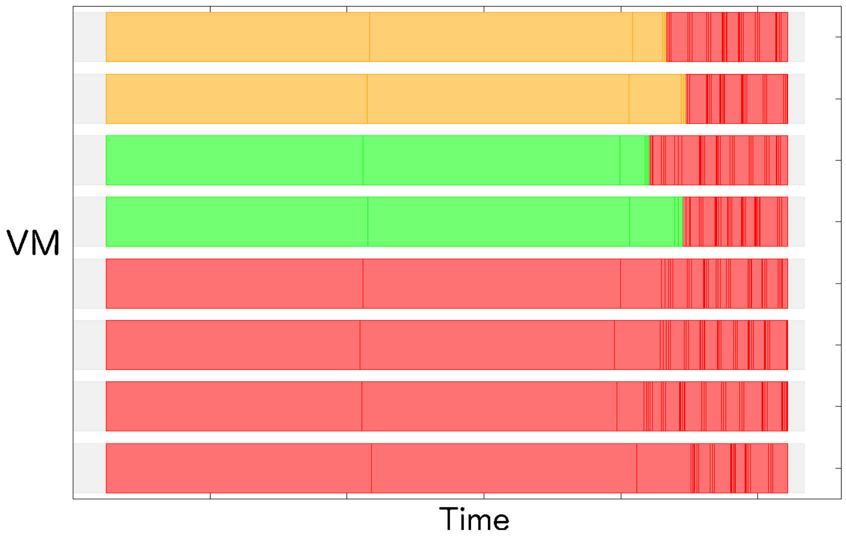

Figure 6.4: Example CyberShake ensemble execution generated by the DPDS algorithm on a cloud with distributed storage system. Horizontal bars are Virtual Machines and colored ranges are task executions which include including file transfers. Tasks from different workflows have separate colors. It is worth noticing that initial tasks of each workflow take long time, which is caused by large input files that need to be staged in to a Virtual Machine.

6.3. Distributed storage system

Distributed storage systems, such as Amazon Simple Storage Service, are among the most popular solutions for storing and transferring data on clouds. This is because most cloud providers offer massively scalable, managed services for storing objects. To model a distributed object storage system we have set the number of replicas to infinity. With this, we are able to simulate behavior of popular cloud storage systems [27], where application developers have only access to a storage API and the cloud provider manages the system to scale it on demand to keep its parameters (e.g., throughput) at constant levels. Latency of 50ms was used and a local Virtual Machine cache of size 50GiB. The total number of simulations run was 21,000 (10 deadlines x 10 budgets x 3 applications x 7 algorithms x 10 experiments).

Figure 6.5 shows the average score of the algorithm as a function of normalized deadline. There is little difference in the performance of the dynamic algorithms for the Montage application. This is because the exponential score metric hides the long tail of low-priority workflows that have successfully completed (e.g., completing a workflow with priority 10 adds 2^{-10} to the score). Figure 6.6 shows that the file locality-aware dynamic algorithms are able to complete more workflows within the same deadline. Knowing that the algorithms also have a slightly higher exponential score we conclude that the algorithm produces better schedules. One notable case where this is not the case is the CyberShake application with larger deadlines (Figure 6.5), where the simplest, unmodified WA-DPDS algorithm performs best. This is a result of the fact that workflow runtimes are overestimated in the presence of a cache. Therefore, WA-DPLS rejects workflows from execution when they could actually finish within budget and deadline. When the Virtual Machine cache is disabled, however, Storage-Aware algorithms always have the best scores, as Figure 6.7 shows.

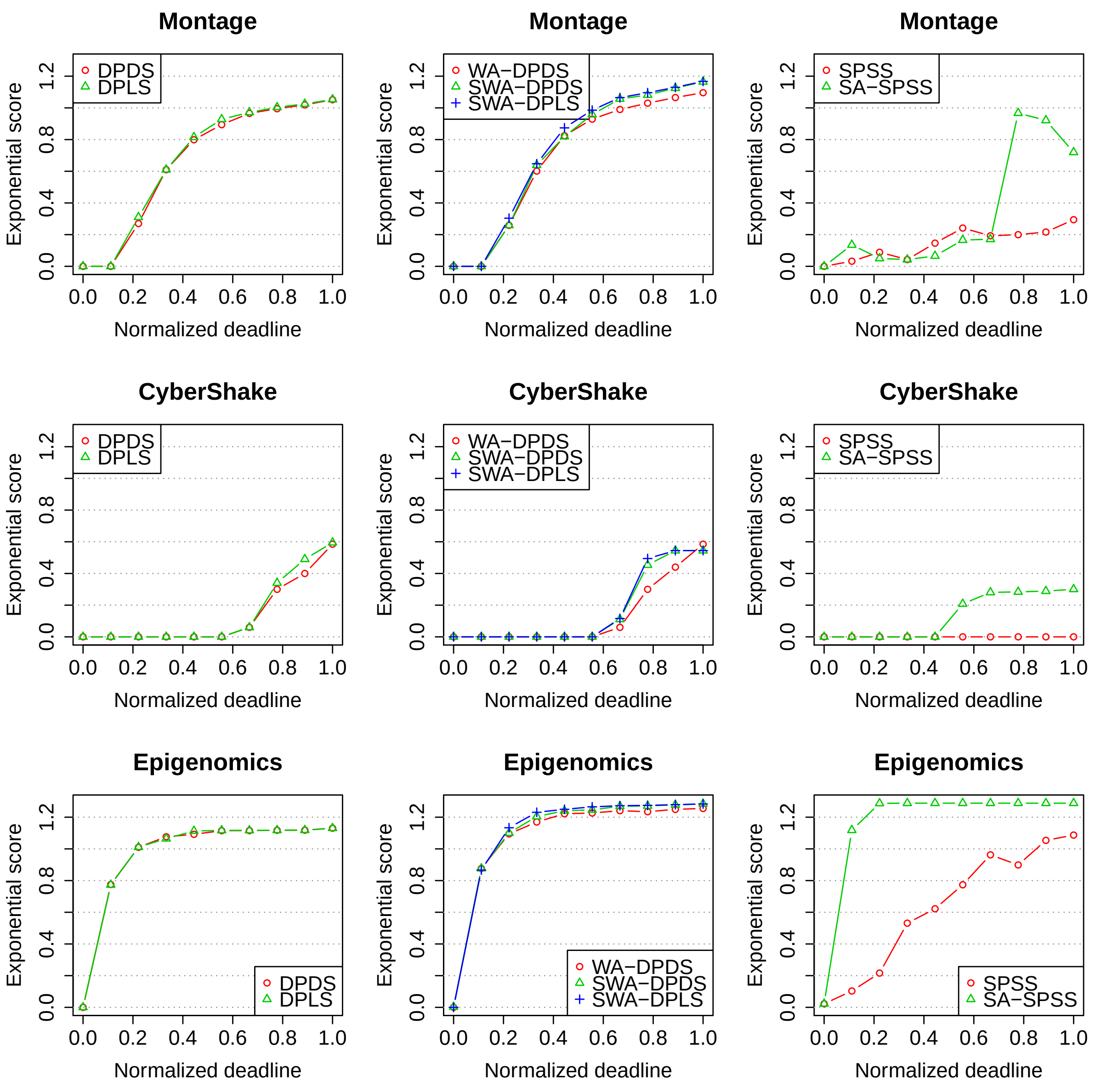

Figure 6.5: Average exponential score for applications running in an environment with a distributed storage system.

The static, storage-aware algorithm outperforms its unaware counterpart. It is superior both in terms of score and the number of completed workflows. For Montage application it sometimes performs worse. This is, again, caused by overestimates of transfer times. Investigating how to improve the estimation function is a non-trivial task, because it requires simulating the state of the entire execution environment. Potential improvements have been left for future work.

Figure 6.8 shows the ratio of file transfer time to total ensemble execution time averaged for all algorithms for this experiment. The CyberShake application spends vast majority (94%) of its total execution time on file transfers. This is caused by large input files of root tasks that have to be staged in to Virtual Machines. In comparison, Montage and Epigenomics spend 14% and 5% of their time on transfers.

Finally, Figure 6.9 shows the average cache bytes hit ratio per application and algorithm. This value is defined

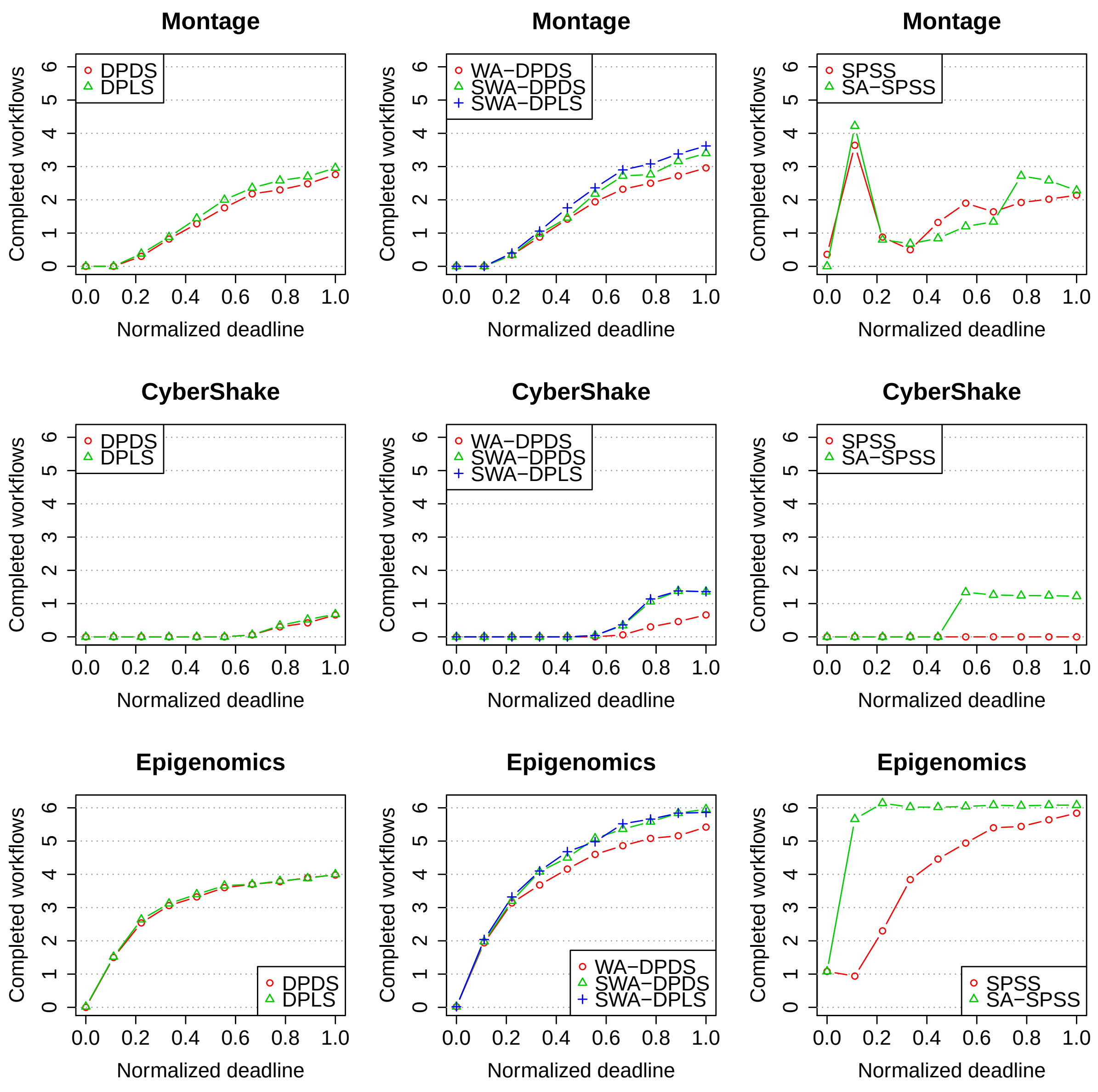

Figure 6.6: Average number of completed workflows in an environment with a distributed storage system.

as the ratio of the total number of bytes retrieved from the cache to the total size of of all files requested to be transferred. One thing to note is that even unmodified algorithms produce schedules with significant cache hit ratios. Some applications, such as Epigenomics, are naturally cache-friendly and have high cache hit ratios regardless of schedule. This is caused by the fact that many of the tasks in these workflows share the same input files. The increase in hit ratio for the file locality-aware algorithms varies by application. It is as low as 1.4 percentage points for CyberShake with DPDS and DPLS algorithms, and as high as 15.5 percentage points for Montage with WA-DPDS and SWA-DPLS algorithms. This confirms that file locality-aware scheduling procedures can reduce the time spent on file transfers for certain application types.

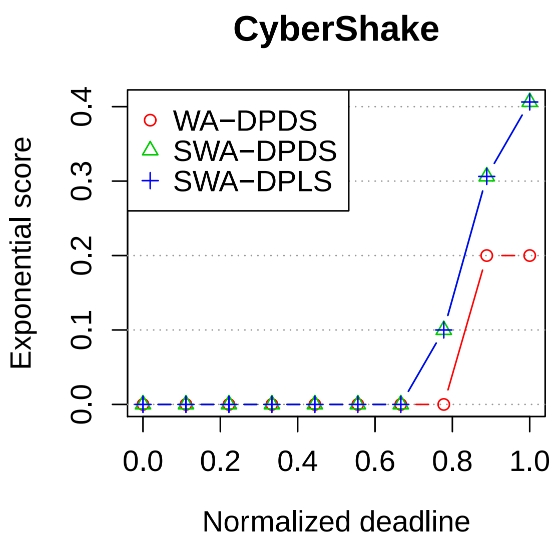

Figure 6.7: Exponential score in environment with distributed storage system and disabled Virtual Machine cache.

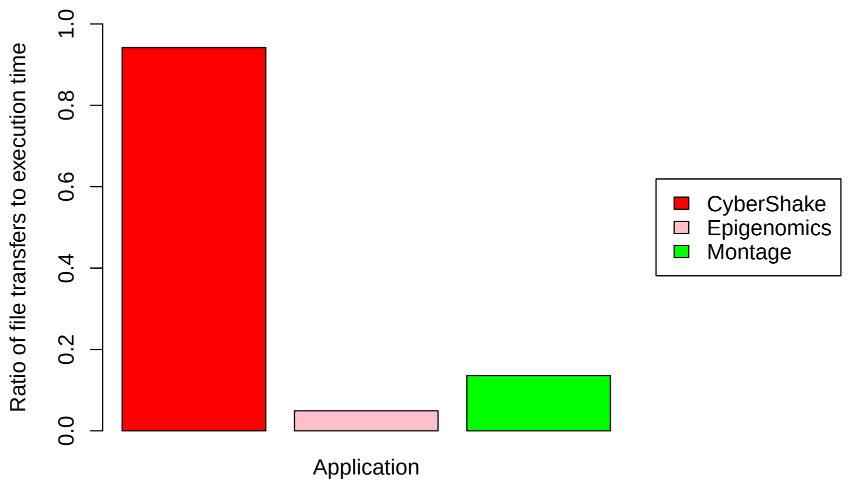

Figure 6.8: Average ratio of total time spent on transfers to total ensemble execution time, per application for distributed storage system.

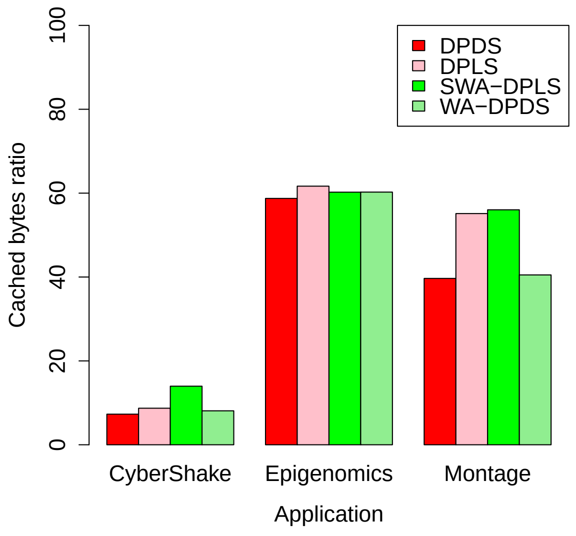

Figure 6.9: Average cache bytes hit ratio per application for distributed storage system.

6.4. NFS storage

Our final experiment was designed to simulate an NFS server that is connected to all Virtual Machines. The NFS share is backed by an array of disks in RAID 5 configuration [10, Chapter RAID]. The replica count was set to 5, the maximum read and write bandwidths were 20MiB/s, the latency was 200 ms, and the local Virtual Machine cache size was 50GiB. The experiment consisted of running 2,100 simulations for CyberShake, Montage and SIPHT. Table 6.1 shows the average exponential score for each pair of application and algorithm, averaged over all simulation runs. For comparison, the table includes the results for the environment where storage is infinitely fast from Section 6.1. The results for NFS storage are similar to what we have presented in previous sections in that the storage and file locality-aware algorithms outperform their original variants.

		DPDS	DPLS	SA-SPSS	SPSS	SWA-DPDS	WA-DPDS	SWA-DPLS
A	CyberShake	0.2388	0.2563	0.2593	0.0000	0.3756	0.2638	0.3810
	Montage	0.6224	0.6282	0.3369	0.1321	0.6981	0.6640	0.7083
	SIPHT	1.0315	1.0371	1.1791	0.2810	1.1390	1.1026	1.1443
B	CyberShake	1.1117	1.1111	0.8983	0.8983	1.1799	1.1799	1.1780
	Montage	0.8664	0.8697	0.7513	0.7513	0.9182	0.9182	0.9232
	SIPHT	1.0925	1.1617	1.2202	1.2202	1.1655	1.1655	1.1947

Table 6.1: Average exponential score of all application and algorithm pairs in the NFS server storage experiment (A), compared to environment where storage is infinitely fast (B).

When comparing performance metrics for algorithms with storage indefinitely fast and NFS storage we can see that for certain applications the average score is substantially lower for all algorithms when storage is enabled. This is particularly visible for the data-intensive applications. For example, the score for DPLS and CyberShake is reduced from 1.1111 to 0.2563 or from 0.7513 to 0.1321 for SPSS and Montage. This is important observation, because it shows that file operations may have substantial effects on ensemble execution.

6.5. Summary

In this chapter we analyzed in four experiment setups performance of scheduling algorithms and impact of data transfers on workflow application execution. The results show that algorithms we developed perform better for most data-intensive applications. Furthermore, we show that data transfers play are non-negligible for most applications.

7. Conclusions and future work

In this chapter we summarize the thesis and draw final conclusions on the algorithms we developed and workflow applications evaluated. Additionally we describe possible future extensions to this work, in the context of the results of this thesis.

7.1. Conclusions

As the popularity of running scientific workflow applications on clouds grows, it is important to optimize the performance of ensemble scheduling algorithms in such environments. In this thesis, we proposed a model for simulating various storage systems that can be used on Infrastructure-as-a-Service clouds. We proposed the modifications to the original algorithms to take into account the data transfers. Most importantly, we developed and evaluated a novel dynamic scheduling algorithm that is aware of the underlying storage system. This algorithm optimizes the schedule by taking advantage of file locality to reduce the number of file transfers.

We showed that for most data-intensive applications locality-aware algorithms perform better than other evaluated algorithms. It is able to successfully complete more, higher priority workflows within budget and deadline constraints. We also conclude that our modifications to the original algorithms are essential. Without them, the performance of the static algorithm (SPSS) is prohibitively low. Also, the workflow admission procedure used in WA-DPDS performs worse without our modifications.

We evaluated scientific applications in the context of how file transfers affect their execution. We showed that some applications may spend as much as 90% of their execution time on file transfers when operating on top of a distributed storage system such as Google Cloud Storage. What is more, we observe that caching files on local Virtual Machine storage is of great significance, as some applications have a cached bytes hit ratio higher than 50%.

We achieved all goals of the thesis. We researched current state of the art on the topic of the thesis. Then we proposed simulation model for file storage and transfer on Infrastructure-as-a-Service clouds. The storage model has been successfully implemented in Cloud Workflow Simulator. On top of that, we developed a novel workflow scheduling algorithm that optimizes file transfers across tasks. The algorithm was evaluated using real world workflow applications and assessed the results in comparison with existing scheduling algorithm. Finally, we analyzed how storage systems affect the execution of different workflow application types on clouds.

7.2. Future work

Improvements to the file transfer estimation procedure are left for future work. We acknowledge that the current procedure often overestimates transfer times and there is significant room for improvement. Additionally, we reason that the SPSS algorithm is particularly vulnerable to dynamic changes in the environment that result in missed deadlines. This may be improved by introducing safety margins to the algorithm in the future. Introducing a billing model for storage is also a possible future research direction.

The results of this thesis can be used by scientists running their applications on clouds to determine the best ensemble scheduling algorithm. Also, the impact of a chosen storage system on workflow execution can be assessed. This information may be used to evaluate storage systems and workflow applications themselves.

The work presented here can be extended to the scope of heterogeneous or hybrid clouds. It would be interesting to address the storage and data placement problem in a multi-cloud environment. We consider this to be a global optimization problem that is different from the non-trivial challenge of selecting the appropriate storage system and scheduling workflows within a single cloud. In the future it would be valuable to combine these local and global approaches, using the methods presented in this thesis and in our related research [40].

A. Source Code

This chapter describes application, namely Cloud Workflow Simulator, that was developed to conduct this thesis. It presents requirements of the simulator and points to documentation sources for further use.

Cloud Workflow Simulator is the tool that was developed during this research. It is an open source project, publicly avaliable for redistribution under Apache 2.0 License [3]. It is a standalone Java application that can be run in variety of environments. To run Cloud Workflow Simulator the following requirements must be met:

- Java Runtime Environment, version 1.6 or higher - for running and developing Cloud Workflow Simulator
- Python runtime, version 2.7 or higher - for simulation visualization
- Ruby runtime, version 1.9 or higher - for simulation visualization
- gnuplot and gnuplot ruby gems - for simulation visualization
- ant - for building Cloud Workflow Simulator
- git, version 1.9 or higher - for contributing changes to Cloud Workflow Simulator

Source code of the project is available at malawski/cloudworkflowsimulator public repository on GitHub service [4]. There is no custom installation procedure; everything that has to be done in order to run Cloud Workflow Simulator is to compile it. Documentation pages of the project describe how to configure Cloud Workflow Simulator, run it and analyze simulations.

B. Publications

The following paper was created during the course of work on this thesis. The publication uses the simulation model and scheduling algorithms defined in this thesis, as well as, shares insights from results analysis. The author of this thesis is co-author of the publication.

- Piotr Bryk, Maciej Malawski, Gideon Juve, Ewa Deelman, *Storage-aware Algorithms for Scheduling of Workflow Ensembles in Clouds*, Journal of Grid Computing, 2015 (submitted)

Bibliography

[1] Amazon elastic compute cloud. `http://aws.amazon.com/ec2/`. Accessed: 2015-06-12.

[2] Amazon elastic compute cloud instance types. `http://aws.amazon.com/ec2/instance-types/`. Accessed: 2015-06-01.

[3] Apache 2.0 license. `http://www.apache.org/licenses/LICENSE-2.0`. Accessed: 2015-06-01.

[4] Cloud workflow simulator github repository. `https://github.com/malawski/cloudworkflowsimulator`. Accessed: 2015-05-02.

[5] Google cloud. `https://cloud.google.com/`. Accessed: 2015-06-12.

[6] Google cloud pricing. `https://cloud.google.com/pricing/`. Accessed: 2015-05-02.

[7] Google cloud storage. `https://cloud.google.com/storage/`. Accessed: 2015-05-01.

[8] R. Agarwal, G. Juve, and E. Deelman. Peer-to-peer data sharing for scientific workflows on amazon ec2. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 82–89. IEEE, 2012.

[9] H. Arabnejad and J. G. Barbosa. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *Parallel and Distributed Systems, IEEE Transactions on*, 25(3):682–694, 2014.

[10] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau. *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books, 2014.

[11] B. Balis. Increasing scientific workflow programming productivity with hyperflow. In *Proceedings of the 9th Workshop on Workflows in Support of Large-Scale Science*, pages 59–69. IEEE Press, 2014.

[12] G. B. Berriman, E. Deelman, G. Juve, M. Rynge, and J.-S. Vöckler. The application of cloud computing to scientific workflows: a study of cost and performance. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371(1983):20120066, 2013.

[13] G. B. Berriman, G. Juve, E. Deelman, M. Regelson, and P. Plavchan. The application of cloud computing to astronomy: A study of cost and performance. In *e-Science Workshops, 2010 Sixth IEEE International Conference on*, pages 1–7. IEEE, 2010.

[14] S. Bharathi and A. Chervenak. Data staging strategies and their impact on the execution of scientific workflows. In *Proceedings of the second international workshop on Data-aware distributed computing*, page 5. ACM, 2009.

[15] L. F. Bittencourt and E. R. M. Madeira. Hcoc: a cost optimization algorithm for workflow scheduling in hybrid clouds. *Journal of Internet Services and Applications*, 2(3):207–227, 2011.

[16] L. F. Bittencourt, R. Sakellariou, and E. R. Madeira. Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm. In *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, pages 27–34. IEEE, 2010.

[17] E. Bocchi, M. Mellia, and S. Sarni. Cloud storage service benchmarking: Methodologies and experimentations. In *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, pages 395–400. IEEE, 2014.

[18] S. Callaghan et al. Metrics for heterogeneous scientific workflows: A case study of an earthquake science application. *International Journal of High Performance Computing Applications*, pages 274–285, 2011.

[19] U. V. Çatalyürek, K. Kaya, and B. Uçar. Integrated data placement and task assignment for scientific workflows in clouds. In *DIDC '11 Proceedings of the fourth international workshop on Data-intensive distributed computing*, DIDC '11, pages 45–54, San Jose, California, USA, 2011. ACM.

[20] W. Chen and E. Deelman. Partitioning and scheduling workflows across multiple sites with storage constraints. In *Parallel Processing and Applied Mathematics, LNCS 7204*, pages 11–20. Springer, 2012.

[21] L. B. Costa, H. Yang, E. Vairavanathan, A. Barros, K. Maheshwari, G. Fedak, D. Katz, M. Wilde, M. Ripeanu, and S. Al-Kiswany. The Case for Workflow-Aware Storage: An Opportunity Study. *Journal of Grid Computing*, July 2014.

[22] A. Dan and D. Towsley. An approximate analysis of the lru and fifo buffer replacement schemes. *SIGMETRICS Perform. Eval. Rev.*, 18(1):143–152, Apr. 1990.

[23] E. Deelman et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[24] R. Duan, R. Prodan, and X. Li. Multi-objective game theoretic scheduling of bag-of-tasks workflows on hybrid clouds. *Cloud Computing, IEEE Transactions on*, 2(1):29–42, Jan 2014.

[25] B. Fitzpatrick. Distributed caching with memcached. *Linux journal*, 2004(124):5, 2004.

[26] H. Garcia-Molina and K. Salem. Main memory database systems: An overview. *Knowledge and Data Engineering, IEEE Transactions on*, 4(6):509–516, 1992.

[27] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 29–43. ACM, 2003.

[28] R. Graves et al. Cybershake: A physics-based seismic hazard model for southern california. *Pure and Applied Geophysics*, 168(3-4):367–381, 2011.

[29] T. Gunarathne, B. Zhang, T.-L. Wu, and J. Qiu. Scalable parallel computing on clouds using Twister4Azure iterative MapReduce. *Future Generation Computer Systems*, 29(4):1035–1048, 2013.

[30] Z. Hill, J. Li, M. Mao, A. Ruiz-Alvarez, and M. Humphrey. Early observations on the performance of windows azure. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 367–376. ACM, 2010.

[31] J. C. Jacob et al. Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking. *International Journal of Computational Science and Engineering*, 4(2):73–87, 2009.

[32] R. Jellinek, Y. Zhai, T. Ristenpart, and M. Swift. A day late and a dollar short: the case for research on cloud billing systems. In *Proceedings of the 6th USENIX conference on Hot Topics in Cloud Computing*, pages 21–21. USENIX Association, 2014.

[33] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. P. Berman, and P. Maechling. Data sharing options for scientific workflows on amazon ec2. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–9. IEEE Computer Society, 2010.

[34] G. Juve et al. Characterizing and profiling scientific workflows. *Future Generation Computer Systems*, 29(3):682–692, 2013.

[35] G. Kaur, U. Moghariya, and J. Reed. Understanding and taming the variability of cloud storage latency. Technical report, 2013.

[36] J. Kitowski, M. Turała, K. Wiatr, and Ł. Dutka. Pl-grid: foundations and perspectives of national computing infrastructure. In *Building a National Distributed e-Infrastructure–PL-Grid*, pages 1–14. Springer, 2012.

[37] T. Kosar and M. Balman. A new paradigm: Data-aware scheduling in grid computing. *Future Generation Computer Systems*, 25(4):406–413, Apr. 2009.

[38] J. Livny, H. Teonadi, M. Livny, and M. K. Waldor. High-throughput, kingdom-wide prediction and annotation of bacterial non-coding rnas. *PloS one*, 3(9):e3197, 2008.

[39] P. Maechling et al. Scec cybershake workflows—automating probabilistic seismic hazard analysis calculations. In *Workflows for e-Science*, pages 143–163. Springer, 2007.

[40] M. Malawski, K. Figiela, M. Bubak, E. Deelman, and J. Nabrzyski. Scheduling multi-level deadline-constrained scientific workflows on clouds based on cost optimization. *Scientific Programming*, 2014.

[41] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski. Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in infrastructure-as-a-service clouds. *Future Generation Computer Systems*, 48:1–18, 2015.

[42] M. Mao and M. Humphrey. A performance study on the vm startup time in the cloud. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 423–430. IEEE, 2012.

[43] P. Mell and T. Grance. The nist definition of cloud computing. 2011.

[44] A. Ramakrishnan, G. Singh, H. Zhao, E. Deelman, R. Sakellariou, K. Vahi, K. Blackburn, D. Meyers, and M. Samidi. Scheduling data-intensive workflows onto storage-constrained distributed resources. In *CCGRID 2007. Seventh IEEE International Symposium on*, pages 401–409. IEEE, 2007.

[45] K. Ranganathan and I. Foster. Simulation studies of computation and data scheduling algorithms for data grids. *Journal of Grid computing*, 1(1):53–62, 2003.

[46] M. Rodriguez and R. Buyya. Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds. *Cloud Computing, IEEE Transactions on*, 2(2):222–235, April 2014.

[47] F. B. Schmuck and R. L. Haskin. Gpfs: A shared-disk file system for large computing clusters. In *FAST*, volume 2, page 19, 2002.

[48] R. F. d. Silva, W. Chen, G. Juve, K. Vahi, and E. Deelman. Community resources for enabling research in distributed scientific workflows. In *e-Science (e-Science), 2014 IEEE 10th International Conference on*, volume 1, pages 177–184. IEEE, 2014.

[49] R. Tolosana-Calasanz, J. Á. BañAres, C. Pham, and O. F. Rana. Enforcing qos in scientific workflow systems enacted over cloud infrastructures. *Journal of Computer and System Sciences*, 78(5):1300–1315, 2012.

[50] H. Topcuoglu, S. Hariri, and M.-y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *Parallel and Distributed Systems, IEEE Transactions on*, 13(3):260–274, 2002.

[51] J.-S. Vöckler, G. Juve, E. Deelman, M. Rynge, and B. Berriman. Experiences using cloud computing for a scientific workflow application. In *Proceedings of the 2nd international workshop on Scientific cloud computing*, pages 15–24. ACM, 2011.

[52] Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny. Characterizing facebook's memcached workload. *IEEE Internet Computing*, 18(2):41–49, 2014.

[53] D. Yuan, Y. Yang, X. Liu, and J. Chen. A cost-effective strategy for intermediate data storage in scientific cloud workflow systems. In *IPDPS Proceedings*, pages 1–12, Atlanta, GA, USA, 2010. IEEE.

[54] S. Zhang, S. Zhang, X. Chen, and X. Huo. Cloud computing research and development trend. In *Future Networks, 2010. ICFN'10. Second International Conference on*, pages 93–97. IEEE, 2010.

[55] Z. Zhang, D. Katz, M. Wilde, J. Wozniak, and I. Foster. MTC envelope: defining the capability of large scale computers in the context of parallel scripting applications. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing - HPDC '13*, pages 37–48, New York, NY, USA, 2013. ACM.