Energy Prediction for I/O Intensive Workflow Applications

by

Hao Yang

B.E., Huazhong University of Science and Technology, 2011

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

The Faculty of Graduate and Postdoctoral Studies

(Electrical and Computer Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

September 2014

Abstract

As workflow-based data-intensive applications have become increasingly popular, the lack of support tools to aid resource provisioning decisions, to estimate the energy cost of running such applications, or simply to support configuration choices has become increasingly evident. The goal of this thesis is to design techniques and tools to predict the energy consumption of these workflow-based applications, evaluate different optimization techniques from an energy perspective, and explore energy/performance tradeoffs.

This thesis proposes a methodology to predict the energy consumption for workflow applications. More concretely, it makes three key contributions: First, it proposes a simple analytical energy consumption model that enables adequately accurate energy consumption predictions. This makes it possible not only to estimate energy consumption but also to reason about the relative benefits different system configuration and provisioning decisions offer. Second, an empirical evaluation of energy consumption is carried out using synthetic benchmarks and real workflow applications. This evaluation quantifies the energy savings of performance optimizations for the distributed storage system as well as the energy and performance impact of power-centric tuning techniques. Third, it demonstrates the predictors ability to expose energy performance tradeoffs for the synthetic benchmarks and workflow applications by evaluating the accuracy of the energy consumption predictions. Overall, the prediction obtained an average accuracy of more than 85% and a median of 90% across different scenarios, while using less than 200x less resources than running than

actual applications.

Preface

This dissertation is based on the energy prediction work that I led during my M.A.Sc studies, which also had the collaboration with Lauro Beltrão Costa and Matei Ripeanu. I was the main author of the work, and responsible for coming up with the research methodology, evaluating it, analyzing the results as well as writing academic papers. The research projects that are the building blocks for this work are described in Chapter 1. Based on the research presented in this dissertation, part of the materials have been submitted for the following publication.

> **Energy Prediction for I/O Intensive Workflow Applications** *Hao Yang*, Lauro Beltrão Costa, and Matei Ripeanu [56].

Table of Contents

List of Tables

List of Figures

List of Acronyms

BLAST The Basic Local Alignment Search Tool

DSS Default Storage System

EDP Energy Delay Product

FITS Flexible Image Transport System

FIFO First In, First Out

FUSE Filesystem in Userspace

HPC High Performance Computing

MPI Message Passing Interface

MTC Many Task Computing

POSIX Portable Operating System Interface

SAI System Access Interface

WOSS Workflow-Optimized Storage System

Acknowledgements

First of all, I would like to thank my advisor Dr. Matei Ripeanu for his support during my graduate study. His insightful advices, patience and continuous support have carried me throughout this journey. What I accomplished during my MASc study, as well as the improvement of my reasoning, critical thinking would not have been possible without his guidance.

I also want to thank my collaborators Lauro Beltrão Costa, Samer Al-Kiswany, E-malayan Vairavanathan, Ketan Maheshwari, Abmar Barros for their advices, suggestions and efforts on the projects we worked together. The projects would never been accomplished without their contribution. I want to thank Prof. Karthik Pattabiraman, Prof. Sathish Gopalakrishnan and other members of the Computer Systems Reading Group (C-SRG) for the weekly paper discussion and giving me the opportunity to have a glimpse of different research areas. I want to extend my thanks as well to my labmates Bo Fang, Elizeu Santos-Neto, Abdullah Gharaibeh and others for creating a wonderful lab environment and for the enjoyable discussions we had.

I am grateful to my friends in Canada, China and other places in the world. I appreciate the encouragement, friendship they provide.

Last but not least, special thanks to my family. To me they are the foremost and everlasting source of support. I would never be whom I am today without my parents. I am also grateful to my girlfriend. I deeply appreciate their unconditional love and understanding of my decisions (including going to Canada to pursue my master degree).

Dedication

To my family.

Chapter 1

Introduction

1.1 Motivation

Scientific investigation relies increasingly on workflow-based data-intensive applications. These workflow applications are typically assembled using different standalone binaries, which communicate via temporary files stored on a distributed storage system [44]. A workflow management system schedules the tasks resulted from the execution of these binaries based on completion of dependent tasks [54].

In this setup, the performance of the storage system plays a key role in the overall workflow performance [27, 51]. In fact, the storage systems have evolved to incorporate advanced techniques that enable trade-offs over interrelated performance metrics such as throughput, reliability, and generated I/O overhead, to best match the application/deployment scenario at hand [11, 14]. At the same time, user/administrator decisions involve allocating resources (e.g., total number of nodes) and configuring the storage system (e.g., chunk size, data placement policy, and replication level).

Consequently, configuring and provisioning the system entails searching a complex multi-dimensional configuration space. In this space, generally the user's goal is "to optimize a multi-objective problem, in at least two dimensions: maximize performance (e.g., reduce application execution time) while minimizing cost (e.g., reduce the total number of CPU hours or dollar amount spent)" [27].

While the trade-off space over these traditional performance metrics has been extensively studied over the past decades, performance with regard to energy efficiency is relatively new. Moreover, this aspect has grown in importance due its impact on cost or even the feasibility of building large (exascale) data-centers/supercomputers [19]. There have been increasing initiatives in both industry and academia to drive energy-efficient optimizations, system design, modeling approaches and other energy-oriented aspects [5, 10, 18, 23, 31, 33].

This context presents three main questions: First, *in what scenarios, if any, do existing optimization techniques lead to energy savings?* Second, *What is the performance and energy sensitivity, in terms of time-to-solution and energy consumption, of power-centric tuning?* Third, *How can users balance time-to-solution and energy consumption when given a target application and its inputs?* The goal of this thesis is to answer these questions and additionally build tools to support the user/administrator in the complex and relatively unexplored task of determining the desired balance between application's time-to-solution and its energy bill.

1.2 Proposed Approach

The focus of this thesis is to devise an energy prediction mechanism that is able to predict the application energy consumption given the resource allocation and storage system configuration. To support users make system configuration, resource allocation, performance and energy balance decisions, the mechanism should be time efficient (e.g., short runtime to search the configuration space) and easy to use (e.g., it should not require complex system instrumentation or seeding measurements. Section 3.1 discusses the detailed requirements.

This work follows a two-pronged approach to build the mechanism: First, a coarse-grained energy consumption model is built to link the energy characteristics of the underlying computing platform and those of the workflow application. The energy model captures different coarse-grained application states: idle state, application processing, storage operations, network transfers. The relatively high-level model enables time efficient predictions for different configuration scenarios while still guaranteeing reasonable accuracy.

Second, this work augments an application performance predictor [27] our group has built (and I have contributed to) with the energy model to enable energy consumption predictions. The predictor uses a queue-based model to simulate the time each participating machine of the distributed platform spends in the aforementioned power states. Synthetic benchmarks are used to gather the system service times of the platform and power profiles of according application states. To seed the energy estimation model, the energy predictor takes the underlying computing platform's performance and power characteristics, the application traces logged by the storage system as inputs. Based on the workload trace the predictor generates the I/O events to be processed and drives the prediction. It predicts the time-to-solution and energy consumption of a real application execution given the resource allocation setting and storage system configuration.

The proposed approach uses a simple analytical model, seeding process, utilizes the simulation model from a performance predictor and obtains sufficient accuracy. Compared to an only analytical model based approach that focuses a single node setup, the approach targets a distributed environment. Compared to a detailed simulation approach that considers low-level system events (e.g., simulating each network packet), this approach is more lightweight and does not require kernel/storage systems changes to enable prediction. Also, it still achieves sufficient accuracy to reason about different configuration decisions, and performance energy tradeoffs.

1.3 Research Projects as Building Blocks for This Work

1.3.1 MosaStore Storage System

MosaStore[1] is a research project that builds a versatile storage system that "harnesses resources from network-connected machines to build a high-performance, yet low cost, storage system" [14].

I have collaborated with Samer Al-Kiswany, Lauro Beltrão Costa, Emalayan Vairavanathan, Abmar Barros, and Matei Ripeanu on this project. I was one of the main designers and developers involved in the project. I implemented key modules, functional tests, unit tests, synthetic benchmarks for the system, and I was involved in peer code review, contributed to the automated testing framework improvement, application integration and various debugging issues. The direct contribution of this project is an open source working prototype, and more importantly, this prototype serves as the experimental vehicle for this thesis.

1.3.2 Research Projects as Building Blocks for This Work

This thesis proposes an energy consumption model that uses power states and time estimates in each state to predict the overall energy consumption. The proposed methodology in this thesis requires several essential building blocks: a performance prediction mechanism that can give time estimates to the energy prediction model, a evaluation platform that can run workflow applications efficiently, and support versatile configurations in which the accuracy of the proposed energy prediction mechanism is evaluated. This section describes my contributions to these building blocks that form together the experimental framework for this thesis. The following text presents the description of these projects, the relationship to my thesis, my contributions, and my publications related to these projects.

[1]`http://www.mosastore.net`

Intermediate Storage Prediction and Provisioning for Workflow Applications. Users have to make system provisioning, resource allocation, and configuration decisions for I/O-intensive workflow applications. To enable selecting a good choice in a reasonable time, this project proposes an approach that accelerates the exploration of the configuration space based on a low cost performance predictor that estimates total execution time of a workflow application in a given setup.

Relationship to my thesis: The performance predictor built by this project gives the essential inputs for the energy model proposed in my thesis. My thesis augments the performance predictor with fine-grained time estimates of different power states. Also my thesis evaluates the configurations visited in this project from an energy perspective.

My contributions were: proposing evaluation scenarios, designing system improvements that support this project, validation for the performance predictor.

The results of this research have been published by Costa et al. [26], [27], [24].

A Workflow-Optimized Storage System using Cross-Layer Optimization. This projects proposes using file system custom metadata as a bidirectional communication channel between applications and the storage middleware. This channel can be used to pass hints that enable cross-layer optimizations.

Relationship to my thesis: This project built an intermediate storage prototype that my thesis uses as the main evaluation platform. The experience gained in this project helped me evaluate the proposed energy prediction mechanism using synthetic benchmarks and real applications in my thesis.

My contributions were: architecting and implementing system modules (e.g., read module optimizations), validation for the proposed workflow-optimized storage system, debugging the system integration with applications, proposing evaluation scenarios.

The results of this work were published by Al-Kiswany et al. [13], [12],[16], and Costa et al. [28].

Evaluating Storage Systems for Scientific Data in the Cloud. Infrastructure-as-a-Service (IaaS) clouds are an appealing resource for scientific computing. This projects investigates the capabilities of a variety of POSIX-accessible distributed storage systems to manage data access patterns resulting from workflow application executions in the cloud.

Relationship to my thesis: This project improved my understanding of the storage system that is used by my thesis, and it helped the integration of the storage system evaluated in my thesis and the workflow applications my thesis targets.

My contributions were: packaging MosaStore system as a reusable cloud image, comparing and analyzing different storage systems' performance.

The result of this work was published by Maheshwari et al. [41].

1.4 Contributions

This thesis makes the following contributions:

1. Proposes a simple analytical energy consumption model that enables adequately accurate energy consumption predictions. This makes it possible not only to estimate energy consumption but also to reason about the relative benefits different system configuration and provisioning decisions offer.

2. Carries out an empirical evaluation of energy consumption using synthetic benchmarks and real workflow applications. This evaluation quantifies the energy savings of performance optimizations for the distributed storage system as well as the energy and performance impact of power-centric tuning techniques.

3. Demonstrates the predictor's ability to expose energy performance tradeoffs for the synthetic benchmarks and workflow applications by evaluating the accuracy of the

energy consumption predictions. Overall, the prediction obtained an average accuracy of more than 85% and a median of 90% across different scenarios, while using less than 200x less resources than running actual applications.

1.5 Dissertation Structure

The rest of this dissertation is organized as follows: Chapter 2 discusses the target applications, supporting middleware, and the performance predictor that is used as the starting point of this thesis. Chapter 3 details the design of the energy predictor. Chapter 4 evaluates the energy predictor using synthetic benchmarks and real-world applications under different configuration choices. Chapter 6 describes previous research related to this work, while Chapter 5 discusses prediction inaccuracies and performance optimization vs. energy optimization. Finally Chapter 7 concludes this dissertation and proposes some future directions.

Chapter 2

Background

This chapter discusses the application domain this work targets: workflow applications, and the underlying intermediate storage layer for workflow application execution. Then it explains the rationale for using intermediate storage systems and demonstrates an example of such systems that our group built. Finally it discusses the performance predictor that predicts the workflow application's execution time.

2.1 Many-Task Applications

Scientists from various fields like astronomy, astrophysics, chemistry, pharmaceutical domain are dealing with an increasing amount of data. Based on the input data size and number of tasks, the problem space can be partitioned into four main categories [45](Figure 2.1).

To process a small number of tasks and small input size, tightly coupled Message Passing Interface (MPI) applications are often used. On the other hand, data analytics like data mining that handle a large amount of data often adopt MapReduce [29]. When dealing with large number of tasks and increasingly large input size, another widely adopted approach to support scientific workflow applications is the *many-task* approach [44]. Many-Task Computing (MTC) keeps the data size of an individual task modest, however, it handles large amount of tasks and large datasets. Some MTC applications can have

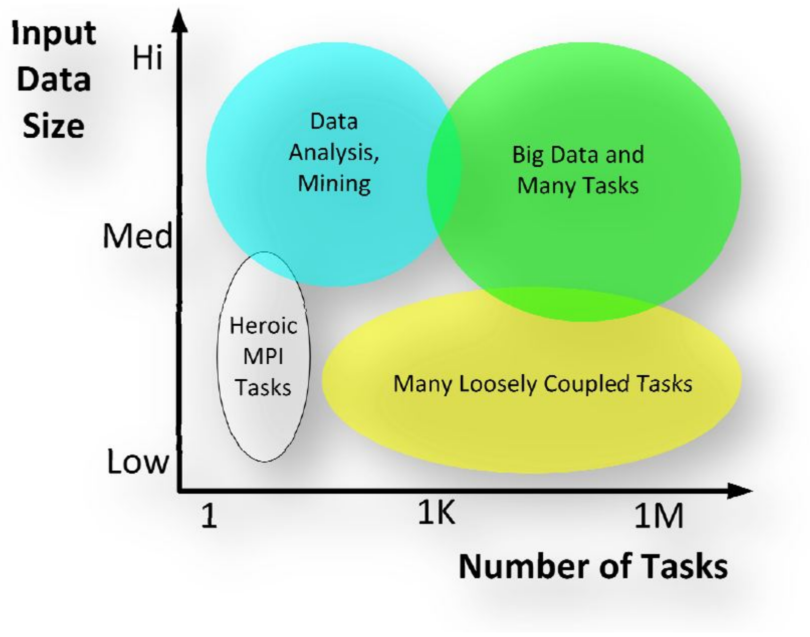


Figure 2.1: Problem types with respect to data size and number of tasks [45]

simple workflows (e.g., BLAST, The Basic Local Alignment Search Tool [17]), while others have multiple workflow stages and various data access patterns (e.g., Montage [40] - an astronomy application that assembles Flexible Image Transport System (FITS) images into custom mosaics).

Many-task workflows use a loose task-coupling model: the standalone executables that compose the workflow communicate through temporary files stored in a shared storage system. This model offers several advantages when running large-scale computing equipments: (1) *Rapid development.* Researchers can use legacy application binaries and write in high-level scripting language such Swift [54] to produce highly parallel executions. (2) *Easy to deploy.* Because of the loosely task-coupled model, the tasks can be scheduled in a distributed system easily as long as the storage is shared. Typically the independent tasks of a workflow can be scheduled arbitrarily on the allocated machines. (3) *Fault tolerance.* Since MTC tasks communicate via file system operations (e.g, producing an intermediate result file, reading a file written by a previous stage), it is natural and easy to resume the workflow using the intermediate files on the shared storage in the face of failure. However, there are several drawbacks related to the execution model. For instance, the workflow tasks that exhibit different patterns (e.g., metadata operation rate, I/O volume and frequency) are mapped uniformly to the underlying shared file systems. The generality of the approach leads to poor performance in some cases.

There are research works that propose different approaches and try to improve the performance of executing many-task applications. Some past work benchmarks key metrics of the performance of shared file systems in order to understand and alleviate the performance bottleneck [59], while some past work proposes using POSIX extended file attributes in the support storage to enable per-file optimized operations for application performance improvement [12].

2.2 Supporting Middleware and Usage Scenario

2.2.1 Backend Storage and Intermediate Storage

To efficiently run HPC workloads, scientists use traditional scientific environments such as clusters, grids and supercomputers (e.g., IBM Blue Gene/P supercomputers [3]), but also emerging computing platforms such as cloud environments (e.g., Amazon EC2 [1]). Traditionally scientific applications are supported on generic distributed storage systems including GPFS [8] and Lustre [9]. These systems provide reliable and secure support to most of the applications, however, these generic systems offer limited performance to scientific workflows without application-specific optimizations. As shown in Figure 2.2, in the Blue Gene/P supercomputer when a compute node (storage client) tries to write/retrieve files from the file system, it sends I/O requests to I/O nodes, then the I/O nodes forward the requests to file server nodes. Additionally these generic distributed storage systems provide strong consistency semantics and thus sacrifice performance (e.g., GPFS performs poorly when files are under the same directory [58]).

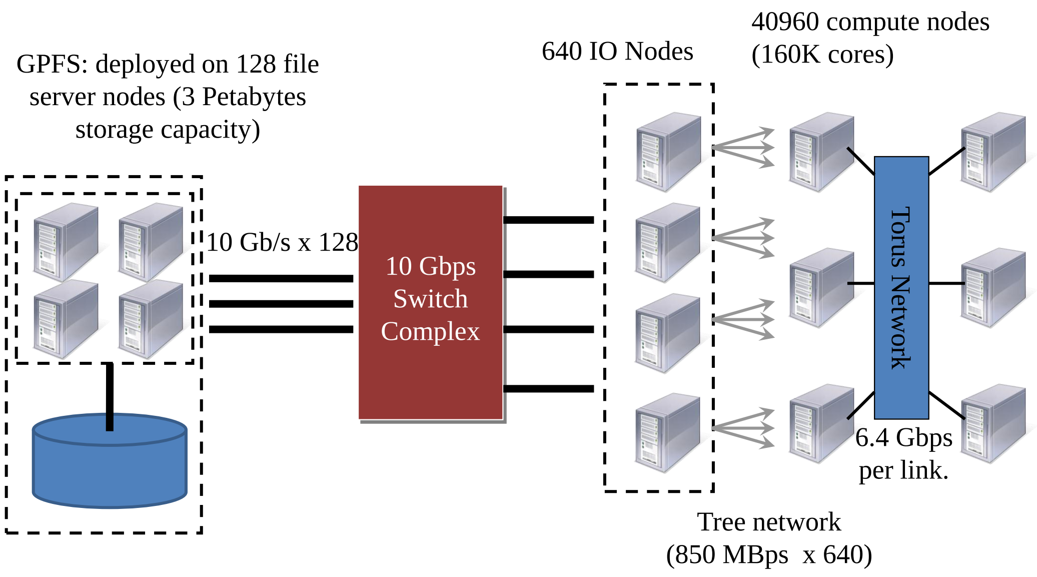

GPFS: deployed on 128 file server nodes (3 Petabytes storage capacity)
10 Gb/s x 128
10 Gbps Switch Complex
640 IO Nodes
40960 compute nodes (160K cores)
Torus Network
6.4 Gbps per link.
Tree network (850 MBps x 640)

Figure 2.2: Architecture of Blue Gene/P Supercomputer [49]. This tiered setup shows the bottleneck due to the limited bandwidth between the compute nodes and the storage nodes.

To avoid the latency to access the backend storage system and also to trade consistency for performance, recent work [14, 20, 55] proposes using an in-memory shared storage layer as an *intermediate* storage system among compute nodes themselves for inter-task communication (Figure 2.3). Each compute node that participates in the workflow execution contributes its local storage to form a shared intermediate storage. At the beginning of the execution, the input files are staged-in from the file server nodes to the compute nodes. During the execution, the workflow runtime engine schedules workflow tasks to the compute nodes that produce intermediate files to the shared storage, and the files are consumed by later workflow tasks. When the execution finishes, the final output files are staged-out from the intermediate storage back to the backend storage system. Therefore, the intermediate storage provides a high-performance storage abstraction to support the workflow execution.

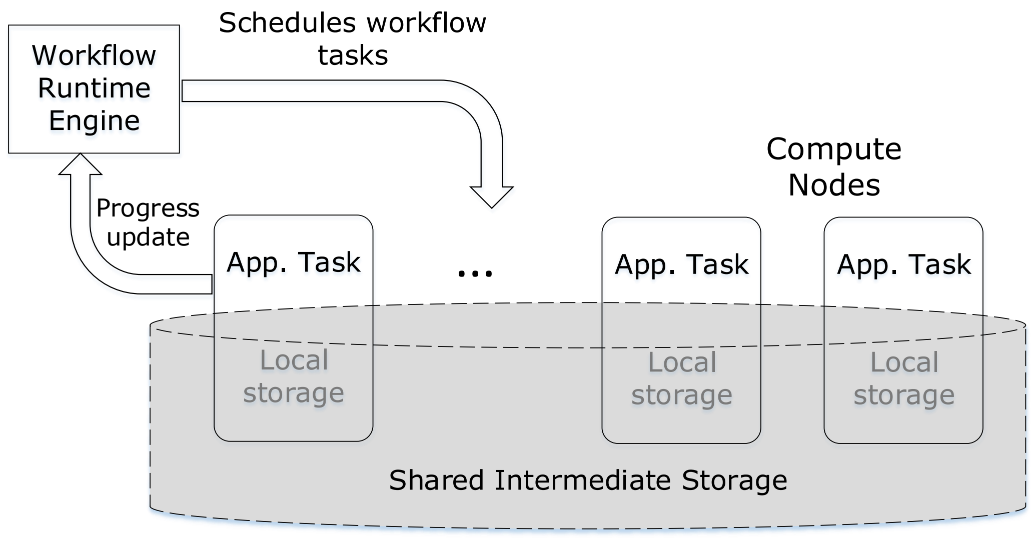


Figure 2.3: **High level architecture of a workflow system.** The shared intermediate storage harnesses the storage space of the participating compute nodes and provides a low latency shared storage space. The input/output data is staged in/out from the backend storage. Depending on configuration, each node may run also storage service that contributes space to the intermediate storage (i.e., storage services and computing tasks may be collocated) [12].

This relatively simple execution model has allowed assembling complex workflow applications and executing them on large shared-nothing infrastructures. For example, Montage is an image processing workflow that assembles together tens of different standalone executables, generates tens of thousands of independent tasks, processes hundreds of GBs of data, and routinely uses hundreds of cluster nodes [40].

2.2.2 MosaStore Intermediate Storage System[2]

This section describes MosaStore, a highly configurable intermediate storage system that is designed to harness storage space from connected machines and build a high performance

[2]Chapter 1 presents my contribution to MosaStore and to the resulting publications.

and scalable storage layer for different application environments.

The direct result of the MosaStore project is a working prototype. Also the prototype serves as the vehicle for this thesis and other research projects. It has been used by multiple institutions in different projects. Besides the workflow application domain this thesis targets, MosaStore is also used for a number of other domains: configurable security [38], checkpointing for desktop grid computing [15], and data deduplication [25].

I was actively involved in *Intermediate Storage Prediction and Provisioning for Workflow Applications* project [24, 26, 27], *Using Cross-Layer Optimization in Workflow Optimized Storage* project [12, 13, 16, 28], *Evaluating Storage Systems for Scientific Data in the Cloud* project [41], and the *energy consumption prediction* project presented in this thesis [56]. MosaStore is the main research and evaluation platform for the work presented in this thesis.

MosaStore Architecture

MosaStore [14, 51] is an object-based distributed storage system that implements the aforementioned execution model and uses cross-layer optimization between the workflow scheduler and storage system to further improve the application's performance. Figure 2.4 shows the architecture of MosaStore. It has three major components: the centralized metadata server, the donor nodes (storage nodes) that store data chunks, and the System Access Interface (SAI) at the client side which offers the I/O interface. The following text describes the three components. A more complete description can be found in Al-Kiswany et al. [50].

- **The metadata manager**. The manager maintains the persistent metadata information about the files and directories in the system. It is stateless (e.g., it does not keep track of the cached data at the client side, or maintain the list of open files).

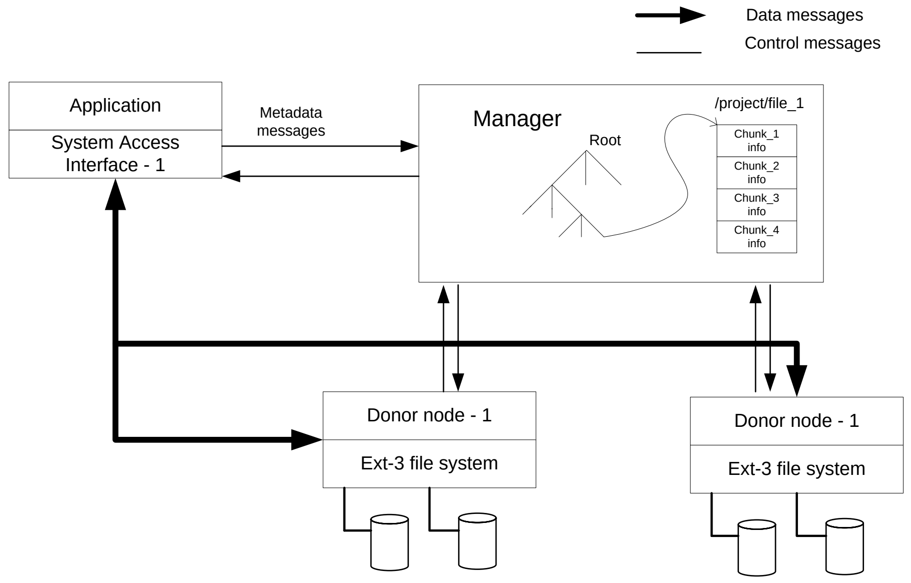


Figure 2.4: **MosaStore Storage System Architecture**. There are three high-level components presented: the System Access Interface (SAI) performing the storage client; the storage nodes that store file chunks; the manager that stores file metadata [50].

After the clients retrieve the metata information about the files needed, the clients can directly initiate parallel I/O requests with the storage nodes. This design of the metadata service greatly simplifies the development complexity, while improves the system scalability. The manager uses the NDBM library [4] to store the metadata information.

- **The storage node**. The storage nodes contribute the storage space (can be memory or disk based) to the shared distributed storage. They publish their status to the manager using a soft-state registration process. Also it involves in serving the I/O requests of the clients and participate in the garbage collection mechanism based on an epidemic protocol.

- **The System Access Interface (SAI)**. The SAI is a user-level file system implementation file system using FUSE kernel module [2]. It provides a Portable Operating

System Interface (POSIX) API. The design principle is to improve the performance of the application execution and not necessarily offer strong consistency semantics. Thus it relaxes some of the POSIX system calls (e.g., fsync, fstatfs) while optimizing the major system calls (e.g., open, read, write).

File-Level Configuration

MosaStore enables multiple global configuration parameters, such as deployment media (e.g., RAM, disks), chunk size (files are split into chunks for high performance I/O), maximum allowed per-node storage space. More importantly, MosaStore enables per-file configuration, which leads to substantial performance benefits when an application has multiple file access patterns. For instance, when a large file is read multiple times by the same client during the execution, users can increase the cache size for this file to avoid remote fetches from storage nodes.

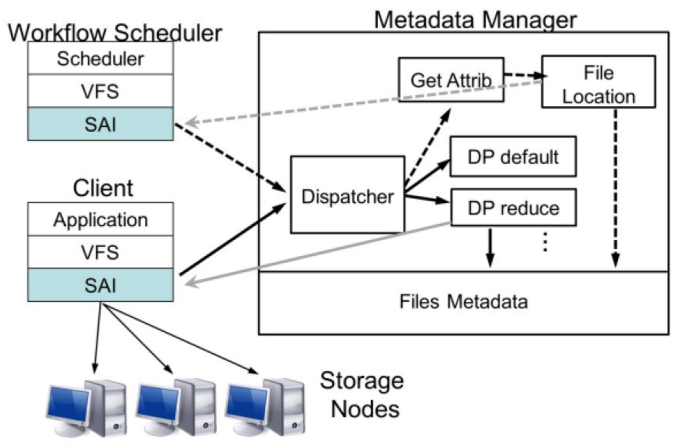


Figure 2.5: Cross-layer communication as proposed by Al-Kiswany [16].(i) the solid lines show the chunk allocation initiated by the client, and later processed by the pattern-specific data placement modules. (ii) the dashed lines show the file location requests made by workflow scheduler.

To enable per-file configuration, MosaStore uses POSIX extended attributes of files to keep their access patterns. Figure 2.5 shows the design of cross-layer communication in MosaStore. From the client's perspective, it can give hints about the data access patterns of the files that it will later consume to MosaStore and set the hints using the extended attributes. After receiving the attribute set call, MosaStore uses the corresponding data placement policies for the files instead of a default policy. From the runtime scheduler's perspective, it can retrieve the file location and other information stored in the extended attributes from MosaStore, thus make optimized scheduling decisions (e.g., putting the coming computation to the node which stores the input files). The cross-layer communication improves the workflow performance as it enables data access optimizations at file-level granularity. Previous work [16, 51] has shown that MosaStore as an optimized intermediate storage can significantly reduce the execution time of complex workflow applications over the default backend file systems.

2.2.3 Configuration Decisions

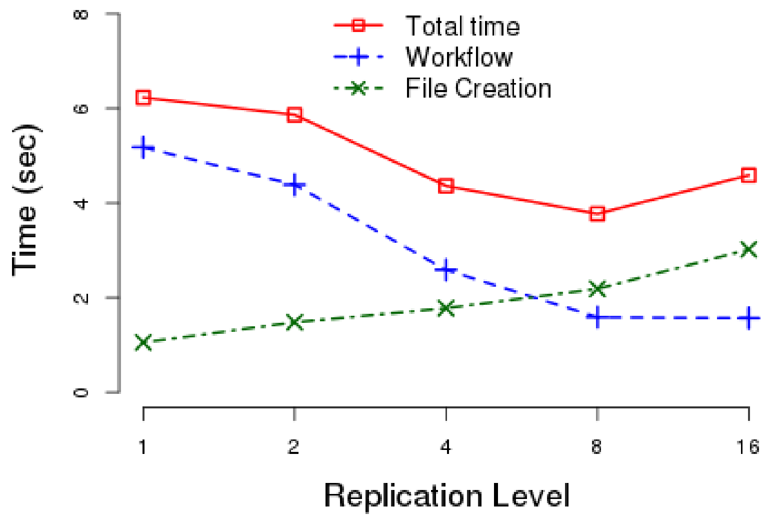


Figure 2.6: Breakdown of broadcast benchmark presented by Costa et al. [28]. In this benchmark an input file is staged-in to the intermediate storage, then in the first stage one client reads it and produces an intermediate file. In the second stage, the intermediate file is read by processes running in parallel on different clients. Each of these processes writes its output independently on the intermediate storage and later stages-out the output. The figure shows the time to create replicas, to execute the actual workflow and the total time.

Despite the advantages brought by this workflow application execution model, there are many configuration choices (e.g., data placement policies, file chunk size, number of nodes to allocate to storage) [28] to be made for the application execution. Different workflow applications, however, achieve optimal performance with different configuration choices [11, 14]. Figure 2.6 highlights the problem using a broadcast benchmark. As one increases the number of replicas of the intermediate file, the time to create all the replicas increases, however, the workflow time deceases due to the fact that there are more data access points when there are more replicas in the system. The figure shows the total execution time reaches a minimum when the storage system sets the number of replicas to 8, which is not known beforehand. There exists a configuration space where system configuration parameters can be tuned and various parameter setup can lead to substantial performance differences.

Exploring the configuration space via application runs is time consuming and costs substantial computational resources. As a result, there has been increasing demand for optimized configuration and resource provisioning decisions via time-efficient and lightweight approaches. To this end, our group built a performance predictor to efficiently predict the application performance given a certain resource and storage configuration. The next section presents the performance predictor.

2.3 The Discrete-event Performance Predictor[3]

As shown in previous sections, the time-to-solution of a workflow application can vary significantly depending on the configuration choices. However, manually exhausting all the configuration scenarios is both time and resource consuming. Costa at el. [27] build a performance predictor in the context of workflow applications that addresses the problem of

[3]Chapter 1 presents my contribution to the performance predictor and to the resulting publications.

configuration space exploration and supporting decisions. Since this performance predictor is the starting point for this work on predicting energy consumption of workflow application execution, the rest of this subsection presents it in more detail. Additional details can be found in Costa et al. [27].

The performance predictor uses a queue-based model to represent the distributed storage system. It takes as inputs a description of the overall workflow composition, a characterization of the workload generated by each stage of the workflow application, the system configuration (e.g., the replication level, system-wide chunk size used by the shared storage system), and the performance characteristics of the hardware platform (summarized in Figure 3.1). The predictor uses the system configuration and performance characteristics to instantiate the intermediate storage system model, and uses the workload description to drive a discrete-event simulation to obtain runtime estimates for each stage of the workflow and for the aggregate runtime. The remaining of this section explains the key building blocks of the performance predictor.

2.3.1 The System Model

The predictor models the participating system components of an intermediate storage. The system components are modeled similarly: each system component is composed of a service module with its in- and out- queues (shown in Figure 2.7). An *application driver* emulates the workflow scheduler and replays the application traces for all workflow stages. Then, the storage layer is modeled by the following components: the *manager* component is responsible for storage metadata operations. The *storage* component stores and replicates file chunks. The *client* component provides an I/O interface to the application by communicating with storage components (via read/write operations) and metadata manager. Each module runs as a service that handles various request types. A service has configurable

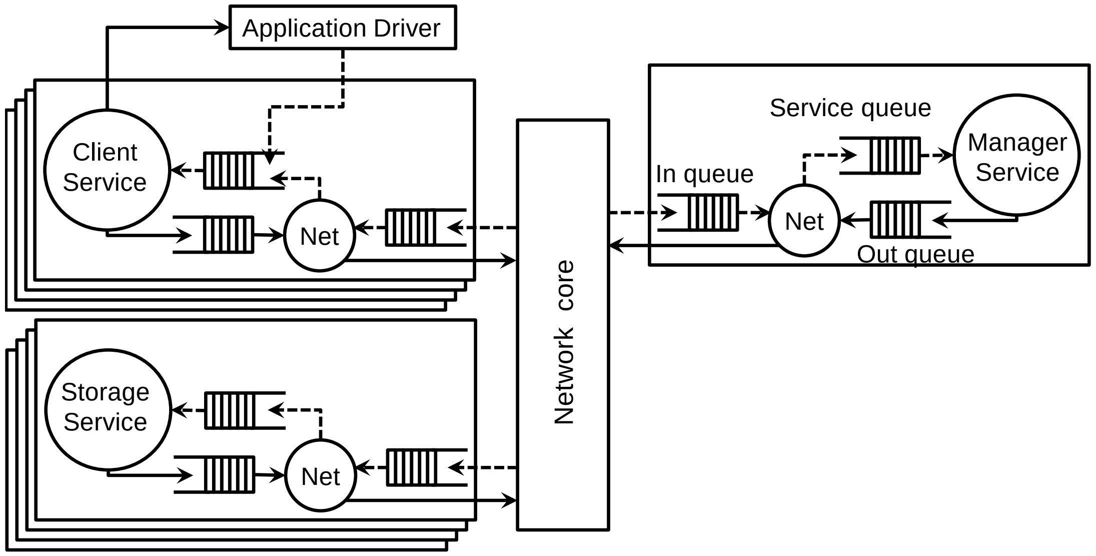


Figure 2.7: The queue-based model: The application driver replays the workflow trace to emulate the workflow execution.

service times for different request types. Each request is placed in the corresponding FIFO queue and later removed from the queue after it is fulfilled.

The rationale behind the prediction mechanism development is increasing accuracy of the modeling until it reaches a adequate level (correctly predicting the relative performance of different configuration choices does not require perfect accuracy). To avoid over engineering, the predictor captures core I/O events and models file operations at chunk-level granularity. However, it simplifies the modeling of metadata operations and captures control paths at coarser granularity as the accuracy of modeling metadata operations hardly impact the total execution time.

2.3.2 Model Seeding: System Identification

The aforementioned system model is a generic abstraction of actual hardware platforms. To determine the performance characteristics of an actual platform, the predictor uses a lightweight identification process for its key performance-related parameters (Table 2.1).

Table 2.1: Platform Performance Parameters

local network service time	μ^{local}
remote network service time	μ^{remote}
manager service time	μ^{ma}
storage node service time	μ^{sm}
client processing time	μ^{cli}

The parameters include the service time of the three system components (manager - μ^{ma}, storage - μ^{sm}, client - μ^{cli}). Each of the three modules has a network component that handles the incoming and outgoing requests. Since the client can request the data from a collocated or remote storage module, the network service time is different in the two cases. Thus, the key parameters for system identification also include remote network service time $\mu(^{remote})$ and local network service time (μ^{local}).

A non-intrusive and low-cost procedure at client-level (no changes required to the kernel module or underlying system) identifies the value to seed storage system's parameters. The seeding process deploys one client, one storage module and one manager on the actual machines, and measures the time to read/write files of different sizes. Also a script runs a network utility tool (e.g., iperf) measures the network service times in scenarios where the client and the storage module are collocated and non-collocated.

The predictor is seeded with an empirical distribution of obtained values for each performance parameter. Use a distribution instead of a fix value reflects the actual applications runs on the computing platform.

2.3.3 Workload Description

The workload description is an application trace logged by the distributed storage. The trace reveals two important pieces of information of the workflow application : (i) it contains per client I/O operations (e.g., open, read, write, flush) with timestamps, operation size, offset and type as well as application compute times between these operations. (ii) it shows the files' dependency graph and the predictor can use this graph to infer the workflow stages and tasks.

After the workload description is obtained, the predictor processes the trace to infer the operations' runtime, compute times of every workflow task, generates I/O events to be simulated, and later simulates the application's I/O execution by driving the I/O events.

Chapter 3

The Design of the Energy Consumption Predictor

The previous chapter discusses the application domain, the middleware and usage scenarios, and highlights the performance configuration problem. The main focus of this dissertation is to address the energy-based analogue of the configuration and provisioning problem focused on application turnaround time in the same execution context. The proposed approach is to design energy prediction tools that enable exploring the system configuration and provisioning space, and evaluating the performance-energy tradeoffs. This chapter discusses the requirements for the energy predictor (Section 3.1), the analytical energy model that estimates the application execution's energy consumption (Section 3.2), how the energy model is seeded with the actual power profiles obtained from the computing platform (Section 3.3), and finally the implementation of the energy predictor (Section 3.4).

3.1 Requirements

To reach the goal of designing energy prediction tools to find optimal configuration instead of running actual experiments, predicting energy consumption should not be time-consuming (so that the tool can be used to explore multiple configurations) and should provide adequate energy consumption accuracy (so that the relative estimates of energy performance tradeoffs are valid among different configurations, thus making it possible to evaluate the tradeoffs among different configuration choices). Additionally, the prediction tools should be simple to use: it should not require complex system instrumentation or seeding measurements.

These considerations make the performance predictor described in Section 2.3 a good starting point for a energy prediction tool. It provides a breakdown of time spent by each system component, which can be leveraged as input to the energy predictor while satisfying a common set of requirements: (i) simple model and seeding mechanism, (ii) effective identification of the desired system configuration, (iii) scalability to predict a workflow application run on an entire cluster while using much less resources than running the actual application [27].

3.2 Energy Model Description

A typical stage of a workflow progresses as follows: (i) each node brings the input data from the intermediate storage to memory (likely through multiple I/O operations), (ii) the processor loads the data from the memory and processes it, (iii) the output is pushed back to the intermediate storage.

Thus different phases of the workload can be associated with different power profiles: (1) Idle state - part of the power is spent simply to keep the node on; (2) Application processing state - the node runs application task binaries on the CPU with the data already in memory (once it has already been fetched from storage); (3) Storage servicing state - serving read/write requests; and (4) Network IO state - performing inter-node data transfers. The energy usage modeling is guided according to the power consumption profile of each of these states: idle (P^{idle}), CPU processing (P^{App}), storage operations ($P^{storage}$), and network transfers (P^{net}). As RAMDisks are used for the shared storage[4], the I/O operations are mainly operations over RAM. This idea of profiling different power states is in the same principle as the coarse-grained energy consumption models used by Costa et al. [25] and Ibtesham et al. [35] in different contexts. Also these power states are sufficient to represent major execution states of the workflow runs that this work targets.

With this mindset, the total energy spent during a workflow application execution can be expressed as the sum of the energy consumption of individual nodes:

$$E_{cluster} = \sum_{i}^{N} E_i^{total} \tag{3.1}$$

where N is the total number of nodes, and E_i^{total} is the total energy consumption of node i. For each node the energy usage during a workflow application execution is:

$$E_i^{total} = E_i^{base} + E_i^{App} + E_i^{WS} \tag{3.2}$$

The *base energy* is the energy spent to maintain the node active:

$$E_i^{base} = P_i^{idle} * T^{total} \tag{3.3}$$

where P_i^{idle} is the node's idle power and T^{total} is the predicted application runtime. This base portion of the total energy consumption accounts mainly for the non energy-proportionality of the hardware platform. As platforms become increasingly energy proportional, it is expected that the share of the idle power in the total energy envelope to decrease.

The *application energy* is the additional energy the application consumes once the data

[4]This is a common setup for running workflow applications sometimes often imposed by the infrastructure itself (e.g., IBM BG/P nodes are not equipped with hard drives.)

needed has been fetched. It is modeled by

$$E^{App} = (P^{App} - P^{idle}) * T^{App} \tag{3.4}$$

The *workflow system energy* (E_i^{WS}) is the energy spent by the underlying workflow system that performs data reads and writes. It is modeled by

$$E_i^{WS} = E_i^{storage} + E_i^{net} \tag{3.5}$$

E_i^{WS} is the sum of the energy spent on reading/writing from/to the local storage ($E_i^{storage}$) and sending/receiving data to/from other compute node over the network (E_i^{net}). Further, $E_i^{storage}$ is modeled by

$$E_i^{storage} = (P^{storage} - P^{idle}) * T_i^{storage} \tag{3.6}$$

where $P^{storage}$ is the node power consumption performing storage operations and $T_i^{storage}$ is the time spent on these operations.

Similarly, the following is the estimation for the energy spent on network transfers:

$$E_i^{net} = (P^{net} - P^{idle}) * T_i^{net} \tag{3.7}$$

which is the product of node power when doing network transfers and the time spent doing this. As the performance predictor tracks the network events, it is feasible to estimate the time spent on each read/write data from/to the network.

The high-level energy consumption analytical model captures key parts of the total consumption during the workflow execution: E_i^{base} is the idle consumption due to non energy-proportionality; E_i^{App} is the energy spent by the application doing necessary computation; E_i^{WS} is the energy spent on performing storage I/O and network I/O. Architectural design improvement can reduce the energy cost of E_i^{base}, while algorithmic optimizations could reduce E_i^{App}. And workflow scheduling and intermediate storage optimizations can contribute to less energy overhead of I/O (E_i^{WS}).

The model estimates main ingredients of the energy cost using the average value of different power profiles and the associate time spent in those profiles. When the intermediate storage is initiated with different configuration decisions, the model consumes different time inputs. When power tuning techniques (e.g., Dynamic Voltage and Frequency Scaling (DVFS)) are used, the model consumes different power states. Thus, this model is able to capture the energy cost of various configuration and technique decisions this work targets. Admittedly, the coarse grained model will not achieve perfect accuracy for each configuration point. Since the requirement of the model is to guide good configuration choices, the model meets the requirement as long as it estimates the correct relative values of different configurations (Chapter 4 evaluates the predictor using various configurations).

This linear model requires both power and time input. Section 3.3 explains how the power parameters are gathered, and Section 3.4 explains the changes made to augment the performance predictor [27] to generate the time input for the energy model.

3.3 Energy Model Seeding

To seed the parameters in the energy model, one needs to get both the power characteristics of the nodes[5] and the corresponding time spent on each power profile.

Synthetic workloads that resemble different phases of the workflow application execution are used to obtain an estimate of the power consumption of different power states:

[5]Since this work considers homogenous compute nodes that have similar performance and power, one node is used to perform power identification. In the case of a heterogenous platform, the seeding process should be performed for each node type.

Table 3.1: Platform Power Parameters

idle node power	P_i^{idle}
node power when stressing CPU only	P_i^{App}
node power when performing storage operations	$P_i^{storage}$
node power when doing network transfer	P_i^{net}

(i) P_i^{idle}, the power samples are retrieved when the nodes are idle over a period; (ii) P_i^{App}, *stress* [6] is used to impose load on CPU and measure power; (iii) $P_i^{storage}$, local write and read throughput tests are used to get this profile; and (iv) P_i^{net}, remote writes from a client to a storage service are performed and the power is measured at the client side. Table 3.1 shows the gathered power parameters.

Accordingly, the energy model is also seeded with the time spent on each power states: (i) T^{total}, the total execution time is estimated by the performance predictor; (ii) T^{App}, the time spent on CPU processing is inferred by the predictor using the logged application trace by the storage system; (iii) $T_i^{storage}$, the time spent on performing local I/O operations is estimated by the predictor; (iv) T_i^{net}, the time spent on doing network transfers is also estimated by the predictor which keeps track of the network events during simulation and gathers time estimates.

3.4 Implementation

Figure 3.1 presents the energy-related augments to the performance predictor presented in Section 2.3. The predictor is augmented to track the time each node spends on the different phases of a workflow task: executing the compute intensive part of the application, writing to/reading from storage, and receiving/sending network data. As explained in Section 2.3.3, a workflow scheduler executes the application on a minimum setup, and the intermediate storage system logs the client side trace. Then the predictor preprocesses the trace to infer

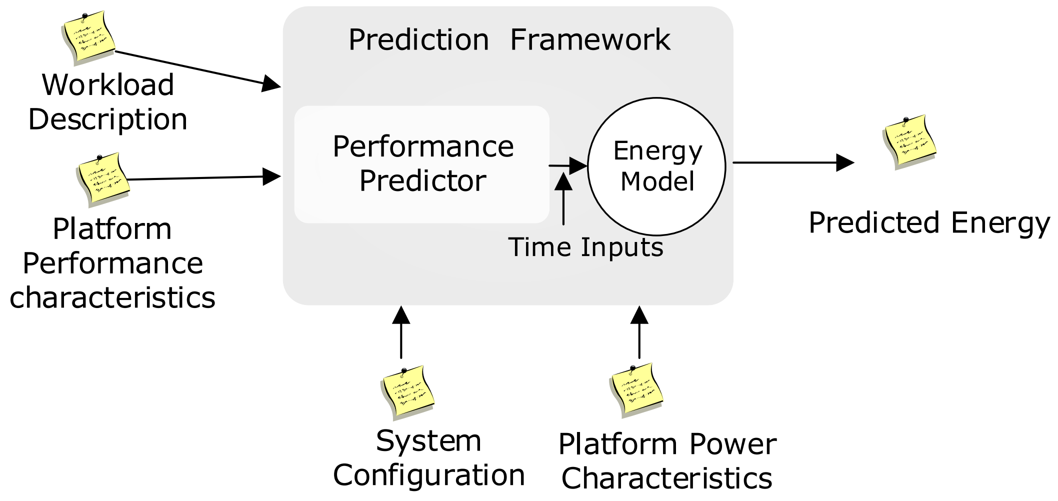


Figure 3.1: The predictor receives the application description, platform performance and power characteristics. The performance predictor estimates the time for several events in the system and passes this information to a module that uses power characteristics of the platform and uses the energy model to estimate the energy consumption.

the time spent on doing computation. To track the time spend on network, each network request is instrumented to track the traffic sent to each machine. The storage entity is instrumented to track the time spent on each read/write request. The performance predictor consumes an empirical distribution of numbers for each performance parameter. When one entity processes a network request/store request, the predictor increments the service time based on the requested data size and performance parameter inputs.

When the simulation completes, the predictor obtains detailed statistics of each machine's storage I/O time and network transfer time. The detailed statistics also enables future opportunity to predict energy consumption on a heterogenous cluster where each machine has different power profiles.

The aforementioned energy model is implemented as modules that augment the discrete-event performance simulator in Java. The statistics module keeps track of the time spent on each power state. The energy module estimates the energy consumption. This module implements the model described in Section 3.2, receives the parameters that describes the power consumption of the platform (Section 3.3) in each different phase of a workflow task, and it uses the same system configuration as the performance predictor (e.g., the number of storage nodes, the number of clients). Finally, the module obtains the estimated time that is spent on different power states from the performance predictor and estimates the predicted energy.

To make the picture clearer, consider predicting the energy consumption of a single workflow task. The client contacts the manager regarding the location of the input files. The manager replies with a set of storage entities that store the files. The client retrieves the data chunks and does computation based on the fetched data. After the computation is done, the client contacts the manager to store output files to the storage entities. In this process the storage entities keep track of their read/write service time. Also each entity tracks the time spent in network transfers. The energy module harnesses the gathered power characteristics and the time estimates of power states to predict the overall consumption. The consumption includes reading input files, writing output files, computation and network transfers.

Chapter 4

Evaluation

This chapter uses synthetic benchmarks that represent common data access patterns that exist in workflow applications and real applications to evaluate the accuracy of the proposed energy consumption predictor. The two representative applications (BLAST [17], Montage [40]) incorporate multiple patterns to demonstrate the accuracy of the predictor. Additionally, the predictor is evaluated using different configuration choices, power tuning techniques (Section 4.4) and analyze energy-performance tradeoffs when one varies the size of the resource allocation (i.e., the number of allocated nodes) (Section 4.5).

Summary of results: For synthetic benchmarks the predictor achieves 89% accuracy in average. For real workflow applications the predictor achieves 91% accuracy in average. Overall, the median accuracy is 90% across different scenarios. Additionally, the predictor is able to capture the energy impact of CPU throttling when applied to applications with different characteristics. Finally, the predictor accurately makes the resource allocation decisions on different platforms when considering energy-performance tradeoff.

4.1 Experimental Setup and Platform

4.1.1 Storage System

The evaluation uses MosaStore [14, 16] as the intermediate storage system because it has multiple configuration knobs (e.g., replication level, chunk size, data placement policy) and one can evaluate their impact on energy consumption. The storage services use RAMDisks as the storage media as RAMDisks are commonly used to support workflow applications and they are the only option in some supercomputers (e.g., IBM BG/P machines).

4.1.2 Testbed and Power Meters

The evaluation uses 11 nodes from Grid5000 'Taurus' cluster at 'Lyon' site [22]. Each node has two 2.3GHz Intel Xeon E5-2630 CPUs (each with six cores), 32GB memory and 10 Gbps NIC. A dedicated node runs the metadata manager and workflow scheduler, while other nodes run the storage service, the I/O client service, and application processes. Each node is connected to a SME Omegawatt power-meter, which provides 0.01W power resolution at 1Hz sampling rate. The power consumption is aggregated, for each node, for the duration of the workflow execution to measure the total energy consumption. The evaluation does not consider the energy consumed by the node that runs metadata service and workflow scheduler as these are fixed and not subject to the configuration changes that the prediction mechanism targets (e.g., replication level, number of nodes).

Table 4.1: Platform power parameters for Taurus cluster. These values change when power tuning techniques are applied or other clusters are considered.

idle power	P_i^{idle}	91.6W
power when stressing CPU only	P_i^{App}	125.2W
power when performing storage operations	$P_i^{storage}$	129.0W
power when doing network transfer	P_i^{net}	127.7W
peak power	P_i^{peak}	225.0W

The Machine Power Profile. The energy model seeding procedure is executed as described in Section 3.3 to identify the power consumption of a node in the different power states: *Idle*, *App*, *storage*, *net*, *peak*. Table 4.1 shows the identified values of the default testbed configuration ('Taurus' cluster). These values change when power tuning techniques are applied (Section 4.4) or other clusters are considered (Section 4.5).

4.1.3 Evaluation Metrics

The evaluation focuses on prediction accuracy by comparing the energy consumption and execution time of actual runs and predictions. Prediction inaccuracy is reported as defined by $I(E) = |1 - E_{pred}/E_{actual}|$ for energy, and $I(T) = |1 - T_{pred}/T_{actual}|$ for time.

Plots report average of 10 trials with error bars showing the standard deviation.

4.2 Synthetic Benchmarks: Workflow Patterns

Previous work has identified common data access patterns in real applications [21, 48, 55, 57]. The evaluation uses synthetic benchmarks that mimic these patterns and evaluates the predictor's accuracy for each pattern independently before evaluating real applications that have multiple patterns.

The workflow patterns used to evaluate are: pipeline, reduce and broadcast. The synthetic benchmarks involve multiple concurrent clients and storage nodes, and each client performs intensive 'read-process-write' procedures mimicking workflow stages.

Pipeline benchmark models a set of compute tasks assembled in a number of parallel sequences in such a way that the output of a previous task is the input of a next task in a chain (Figure 4.1). In this experiment, 10 application pipelines run concurrently on 10 compute nodes and perform three processing stages that read/write files from/to the distributed shared storage, and also stress CPU.

Reduce benchmark represents a single task that consumes the outputs produced by multiple computations. In the experiments, 10 processes run in parallel on different nodes, and each produces an intermediate result. A following reduce task consumes those intermediate files, and produces the final output.

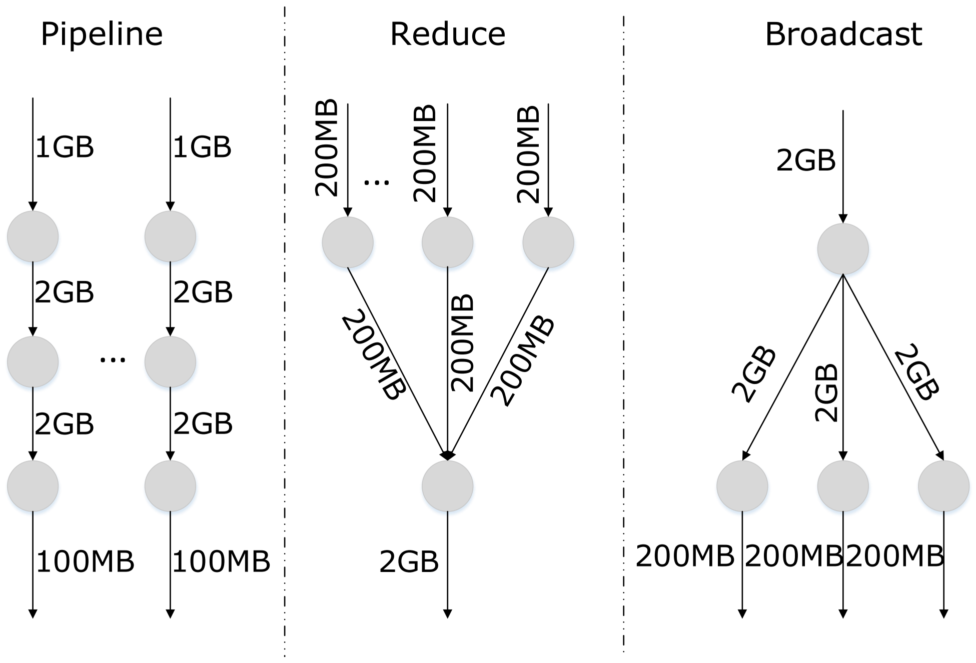


Figure 4.1: Pipeline, Reduce and Broadcast benchmarks. Circles represent a workflow task performing CPU processing using *stress* [6] and arrows represent data transfers among stages. The labels on the arrows represent the file sizes used in the benchmarks.

Broadcast benchmark has a single task producing an output file that is consumed by multiple concurrent tasks. 10 processes run in parallel and consumes the file produced in an earlier stage.

Label *DSS* (Default Storage System) is used for experiments running a default system configuration: data chunks are striped across storage nodes in a round-robin fashion and no optimization is provided for any workflow pattern. Label *WOSS* (Workflow Optimized Storage System) is used for the experiments where the system configuration is optimized for a specific workflow pattern (including location aware scheduling, data placement or replication) [16, 51]. The goal of showing results for these two configurations is two-fold: (i) demonstrate the predictors accuracy in a default configuration setting (Section 4.2.1), and (ii) show its ability to predict energy savings when performance optimized configurations are used (Section 4.2.2).

4.2.1 Evaluating Energy Prediction Accuracy on DSS

The predictor's accuracy is firstly evaluated when the workflow patterns are running on a default storage system (DSS). DSS uses default global configuration for all the applications (i.e., no replication, files are striped round-robin across all storage nodes). Figure 4.2 presents the predicted and actual energy consumption for the synthetic patterns on DSS. The pipeline benchmark exhibits the best accuracy (only 5.2% inaccuracy). For reduce the average inaccuracy is 16.4%, while for broadcast it is 15.9%. Overall, the energy consumption predictions have an average of 12.5% inaccuracy and typically close to one standard deviation interval. Importantly, the predictor response time is 20-30x times faster than running the actual benchmark, resulting in the usage of 200x-300x less resources (machines $\times$ time) and showing that the results satisfy the objectives presented in Section 3.1.

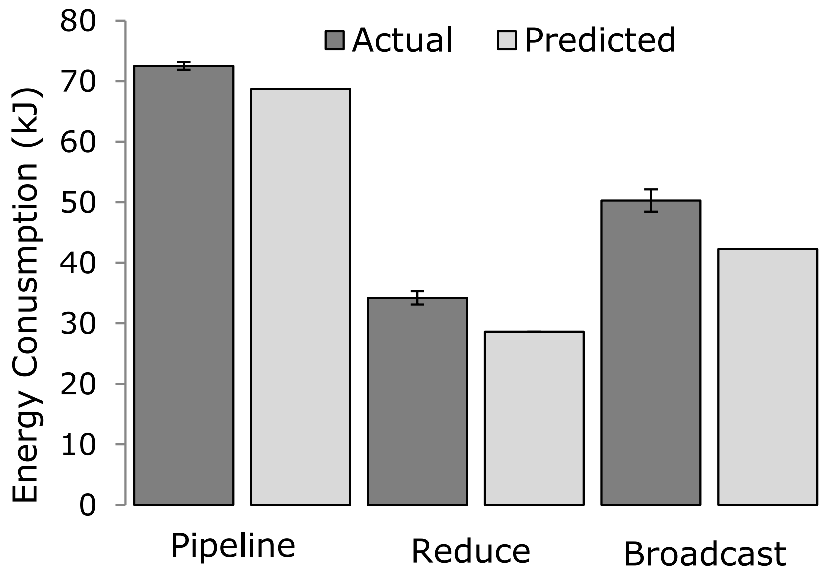


Figure 4.2: Actual and predicted average energy consumption for pipeline, reduce, broadcast benchmarks on DSS.

4.2.2 Evaluating Energy Prediction Accuracy on WOSS

In the default storage system configuration (DSS), a round-robin data placement policy is used: the files produced by workflow tasks are striped across the nodes of the shared storage. Thus, when one task consumes the input files, it needs to connect to other compute nodes and receive file chunks, which generates high network contention and results in suboptimal performance.

As discussed in Section 2.2.2, users can use POSIX file extended attributes to inform the storage system about specific data access patterns. File extended attributes enable workflow optimizations including moving computation near data, and location aware scheduling. Hence, this section evaluates the predictor's ability to capture the energy savings when using a workflow optimized storage system (WOSS). Table 4.2 shows the metadata attributes and the corresponding optimization used in different patterns. The hints about different patterns are set in key-value pairs. The key is the name of the attribute, while the latter

Table 4.2: Metadata attributes and the corresponding optimizations

Pipeline pattern	set (DP, local)	Indicates preference to allocate the file blocks on the local storage node.
Reduce pattern	set(DP, collocation | <group-name>)	Preference to allocate the blocks for all files within the same <group-name> on one node.
Broadcast pattern	set(Replication, <repNum>)	Replicate the blocks of the file <repNum> times.

is the concrete value of the attribute.

In the pipeline scenario, unlike DSS that stores the data chunks of each file across all storage nodes, WOSS stores the intermediate pipeline files on the storage node co-located with the application. The workflow scheduler later places the task that consumes the file on the same node to improve performance via data locality. In the reduce pattern scenario, WOSS co-places the output files on a single node and exposes their location, so that the scheduler can schedule the reduce task on the machine. In the broadcast pattern scenario, parallel tasks consume the same file concurrently, which creates an access bottleneck. WOSS creates multiple replicas of the bottleneck file so that the parallel tasks have multiple data access points.

Figure 4.3, Figure 4.4 and Figure 4.5 show the actual and predicted average energy consumption for the three benchmarks on DSS and WOSS.

The right plot in Figure 4.3 shows the actual and predicted energy for the pipeline benchmark: the predictor achieves 13.4% inaccuracy. The predictor achieves 12.2% inaccuracy for reduce. Figure 4.5 shows the results using 4 replicas: the predictor achieves

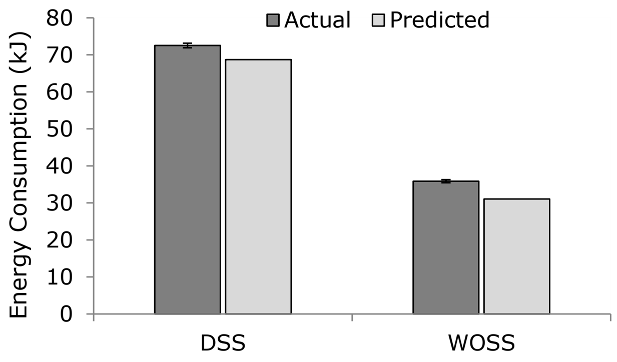


Figure 4.3: Actual and predicted average energy consumption for the pipeline benchmark.

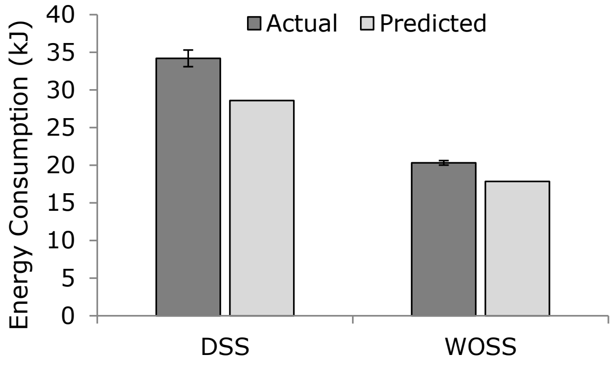


Figure 4.4: Actual and predicted average energy consumption for the reduce benchmark.

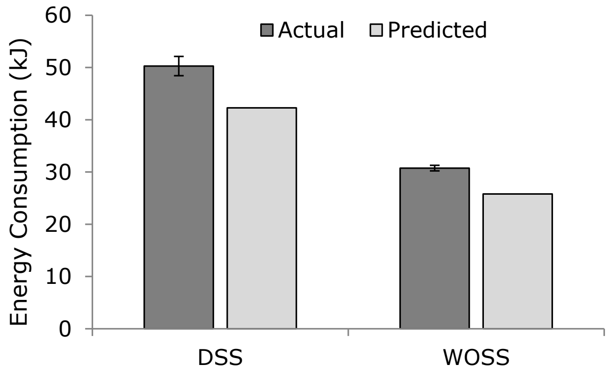


Figure 4.5: Actual and predicted average energy consumption for the broadcast benchmark.

16% inaccuracy for broadcast. WOSS exploits data locality and location-aware scheduling, thus it reduce the energy spent on data movement as well as idle energy since it accelerate the application execution. The predictor accurately predicts the energy savings compared with DSS. It has 2.7% inaccuracy for the energy savings of replacing DSS with WOSS in the pipeline scenario, 22.6% inaccuracy for predicting energy savings for reduce, 15.6% inaccuracy for broadcast.

4.2.3 Summary

The predictor captures the energy consumption for both DSS and WOSS configurations with adequate accuracy and, more importantly, accurately predicts the energy savings brought by WOSS. As a result, the predictor can help users to make storage system configuration decisions based on the energy consumption metric.

4.3 Predicting the Energy Envelope of Real Applications

This section evaluates the framework's prediction availability when it is used for real workflow applications. It uses two applications: BLAST [17] and Montage [40] with representative workloads to evaluate the proposed predictor.

4.3.1 BLAST Results

BLAST [17] is a DNA search tool. Each node receives 8 DNA sequence queries as input (a file for each node) and all nodes search the same database file (i.e., BLAST has the broadcast pattern) (Figure 4.6). The database *refseq_rna* has 18 files and they are stored on the intermediate storage. The input files are staged in the intermediate storage and each node produces one output file that is staged out to the backend storage.

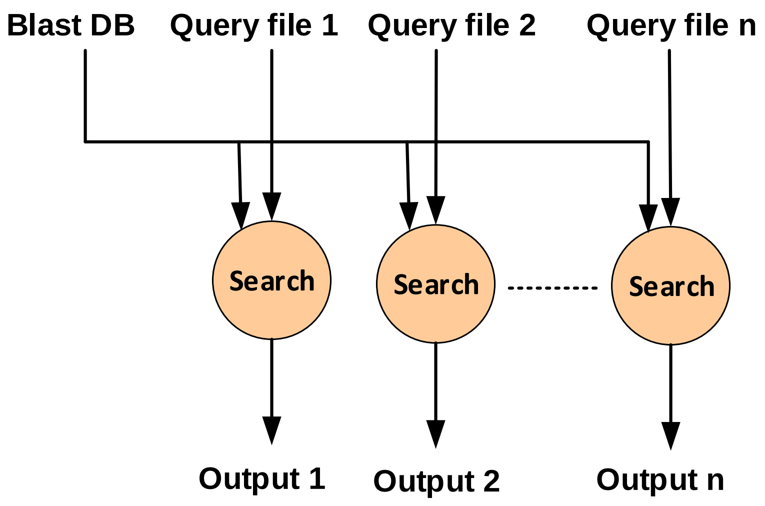
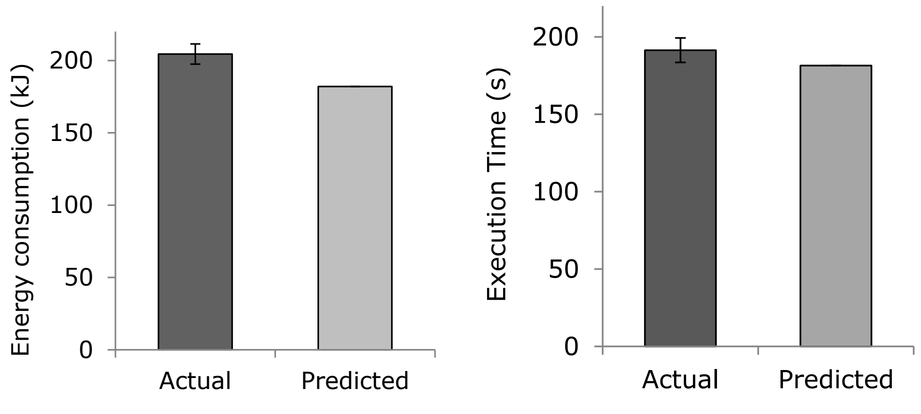


Figure 4.6: BLAST workflow. All nodes search BLAST database (1.8GB) in parallel.



(a) Energy

(b) Time

Figure 4.7: Actual and predicted average energy consumption and execution time for BLAST.

As shown in Figure 4.7, the predictor achieves 11% inaccuracy for energy prediction and 5.2% inaccuracy for time prediction. Compared with the previously discussed broadcast benchmark, the real application prediction achieves better results.

4.3.2 Increasing the Workflow Complexity: Montage Results

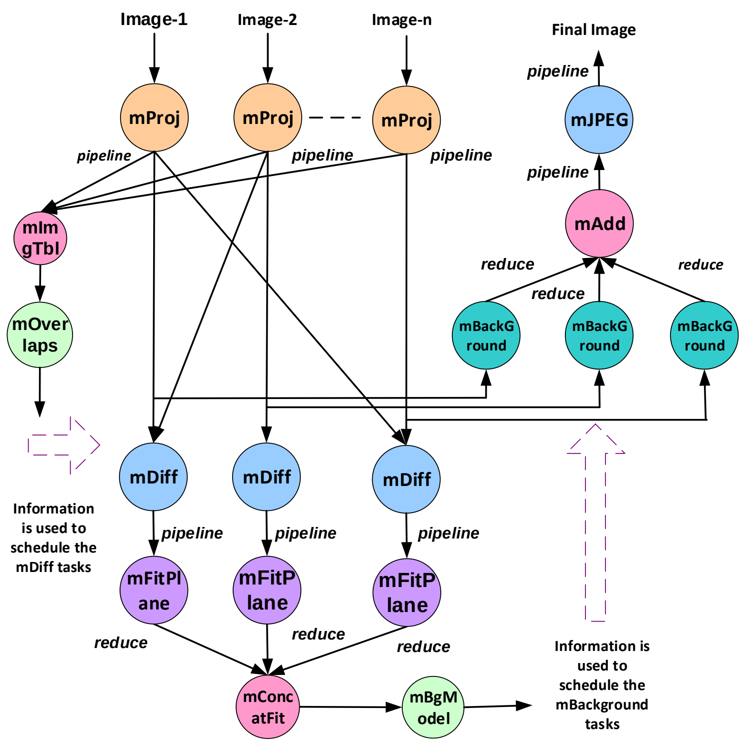
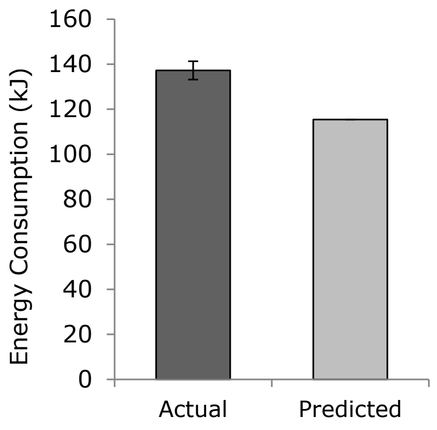


Figure 4.8: Montage workflow

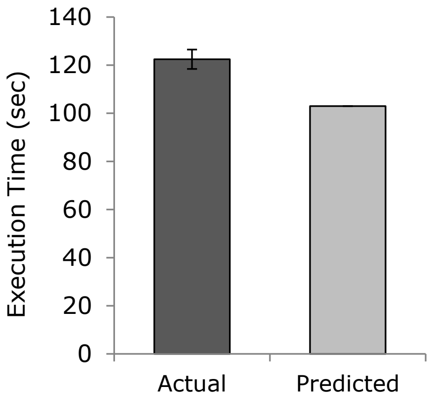
Montage [40] is a complex astronomy workflow composed of 10 different stages (Figure 4.8), and a highly variable I/O communication intensity among the workflow stages (Table 4.3 shows a small workload and Table 4.4 shows a large workload). Additionally, the application has a number of distinct workflow patterns (e.g., mProject, mDiff and m-Background have pipeline pattern; mConcatFit and mAdd have reduce pattern). In total

Table 4.3: Characteristics of small Montage workload

Stage	Data	#Files	File Size
stageIn	320MB	163	1.7MB-2.1MB
mProject	1.3GB	324	3.3MB-4.2MB
mImgTbl	50KB	1	50KB
mOverlaps	54KB	1	54KB
mDiff	409MB	895	100KB - 3MB
mFitPlane	1.8MB	449	4KB
mConcatFit	21KB	1	21KB
mBgModel	8.3KB	1	8.3KB
mBackground	1.3GB	325	3.3MB - 4.2MB
mAdd	1.3GB	2	503MB
mJPEG	15MB	1	15MB
stageOut	518MB	2	15MB-503MB



(a) Energy



(b) Time

Figure 4.9: Actual and predicted average energy consumption and execution time for small Montage workload.

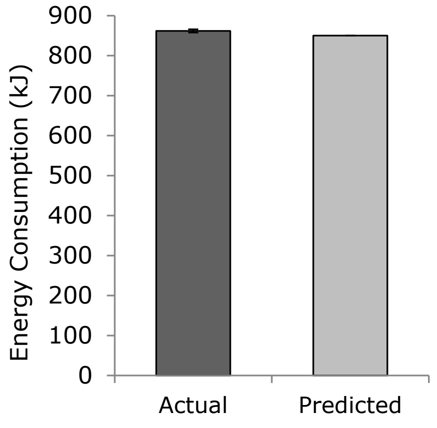
the small workload contains around 2000 tasks, while the large workload contains around 13200 tasks.

Table 4.3 shows the small Montage workload used for evaluation. Although it is the smaller of two Montage workloads, it still has complex workflows and a large amount of tasks. Figure 4.9 shows the prediction results. The predictor achieves 13.8% inaccuracy in time and 15.9% inaccuracy in energy for Montage.

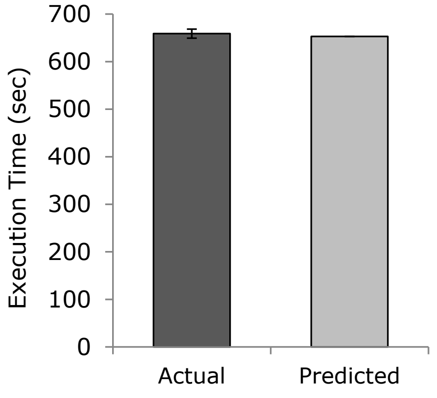
Table 4.4: Characteristics of large Montage workload

Stage	Data	#Files	File Size
stageIn	1.9GB	955	1.7MB-2.1MB
mProject	8.0GB	1910	3.3MB-4.2MB
mImgTbl	284KB	1	284KB
mOverlaps	336KB	1	336KB
mDiff	2.6GB	5654	100KB - 3MB
mFitPlane	12MB	2833	12MB
mConcatFit	575KB	1	575KB
mBgModel	49KB	1	49KB
mBackground	8.0GB	1910	3.3MB - 4.2MB
mAdd	6.0GB	2	3.0GB
mJPEG	46MB	1	46MB
stageOut	3.05GB	2	46MB-3GB

Table 4.4 shows the large Montage workload used for evaluation. Figure 4.10 shows the prediction results, while Table 4.5 shows the per-stage prediction results of the large workload (StageIn and StageOut are writing raw inputs to the intermediate storage and writing final outputs to the backend stoage. Thus they are not considered in the prediction. mImgTbl is included as part of mProject stage). In terms of total energy cost and time to solution, the predictor achieves 1.5% inaccuracy for energy consumption and 1% inaccuracy for time. However, as the per-stage results show, some stages are under-predicted (e.g., mProject, mAdd), while some are over-predicted (e.g., mConcatFit, mBackground).



(a) Energy



(b) Time

Figure 4.10: Actual and predicted average energy consumption and execution time for large Montage workload.

Table 4.5: Per-stage results of large Montage workflow workload

Stage	Actual	Predicted	Inaccuracy
mProject	148.7kJ	131.6kJ	11.6%
mOverlaps	2514J	2625J	4.4%
mDiff	145.5kJ	160.5kJ	10.4%
mFitPlane	58.1kJ	57.8kJ	0.6%
mConcatFit	25.2kJ	29.2kJ	15.8%
mBgModel	35.7kJ	34.4kJ	3.8%
mBackground	50.5kJ	58.7kJ	16.2%
mAdd	310.2kJ	283.3kJ	8.7%
mJPEG	85.4kJ	92.0kJ	7.8%

Improving the per-stage accuracy is an ongoing work.

4.3.3 Summary

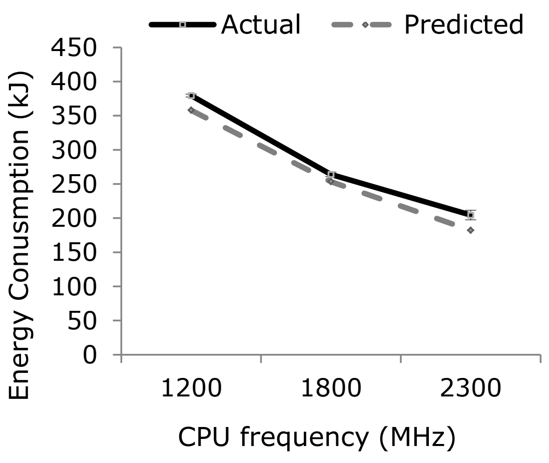
Overall, the energy predictions are more accurate for the real applications than for synthetic benchmarks. This happens because the synthetic benchmarks are designed to produce a high stress on the I/O subsystem, which results in contention and higher variance that is harder to capture when modeling the storage system.

4.4 Predicting the Energy Impact of Power-centric Tuning

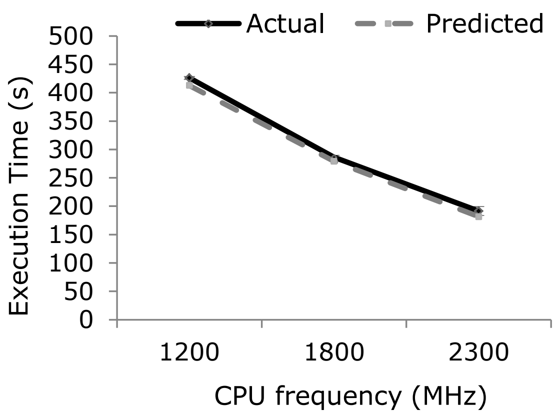
CPU frequency scaling (a.k.a. CPU throttling) is an important technique where processors run at less-than-maximum frequency to conserve power. Frequency scaling, however, limits the number of instructions a processor can issue in a given amount of time; thus this technique can prolong the execution time while conserving instantaneous power. Therefore, it is not clear whether frequency scaling can reduce the energy cost for workflow applications.

To evaluate the predictor's ability to predict the energy impact of CPU frequency scaling, two types of representative applications are used: (i) BLAST, with same workload as in the previous section, representing a mix of I/O and CPU intensive applications; (ii) the pipeline benchmark, performing just I/O operations (the benchmark is modified to only have a minimum CPU stressing stage), representing an I/O intensive application. The processors are set at different frequencies (1200MHz, 1800MHz and 2300MHz), and for each frequency independent seeding is performed.

Figures 4.11 and 4.12 show the actual and predicted energy consumption for BLAST and, respectively for the pipeline benchmark, for different frequencies. Since BLAST is more CPU intensive, using the minimum frequency (1200MHz) just prolongs the runtime and leads to 85.5% more energy consumed than when using the maximum frequency. The

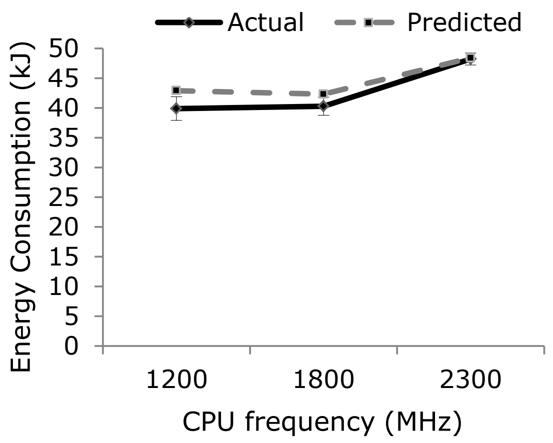


(a) Energy

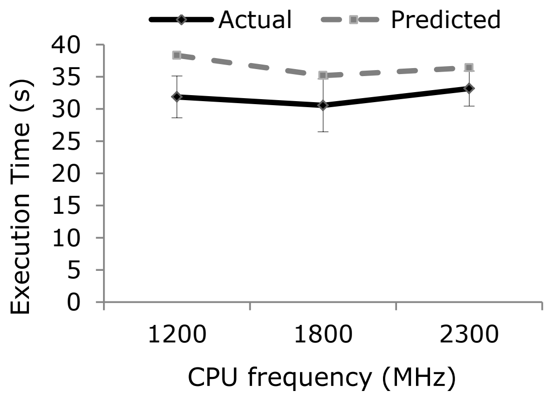


(b) Time

Figure 4.11: Actual and predicted average energy consumption and execution time for BLAST for various CPU frequencies.



(a) Energy



(b) Time

Figure 4.12: Actual and predicted average energy consumption and execution time for the pipeline benchmark for various CPU frequencies.

predictor accurately estimates the much higher (96.5%) energy cost. For pipeline, using minimum frequency does not increase runtime. In fact, since the instantaneous power is reduced, CPU throttling actually brings energy savings, which is partially captured by the predictions. The actual runs show 17% energy savings, while the predictor estimates 11% savings.

Summary: The results for the two workloads highlight that, depending on the computational and I/O characteristics of the workflow application, CPU throttling can bring energy savings or lead to additional energy costs. The predictor provides an effective mechanism to predict the energy consumption when the platform enables power tuning techniques like frequency scaling and can be used in practice to make configuration decisions.

4.5 Predicting Energy-Performance Tradeoffs

Another important decision available to users/administrators is the allocation size. As one allocates more compute and storage nodes for executing workflow applications, the performance should improve because of the extra computing resources. However, due to scale overheads and non-energy proportionality the total energy cost will likely increase. For instance, the Montage workload evaluated has the lowest energy footprint when using only one node (yet in this case it displays the highest time to solution). A popular hybrid metric that finds a compromise between these two metrics is the energy-delay product (EDP). This section evaluates the predictor's ability to accurately estimate this hybrid metric for various setups.

Energy-delay product is estimated while varying the number of allocated nodes. Due to platform size limit, the same Montage workload used in previous sections is executed on up to 10 nodes and evaluate the accuracy of the predictor. Figure 4.13 shows the EDP results,

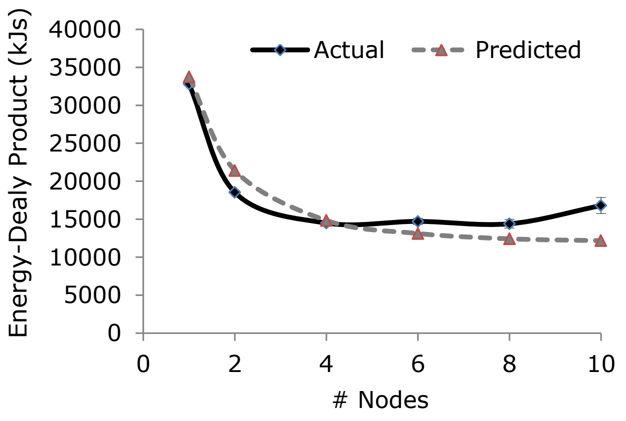


Figure 4.13: Actual and predicted Montage energy-delay product (EDP) at the various scales the experiments can be executed in ‘Taurus’ cluster.

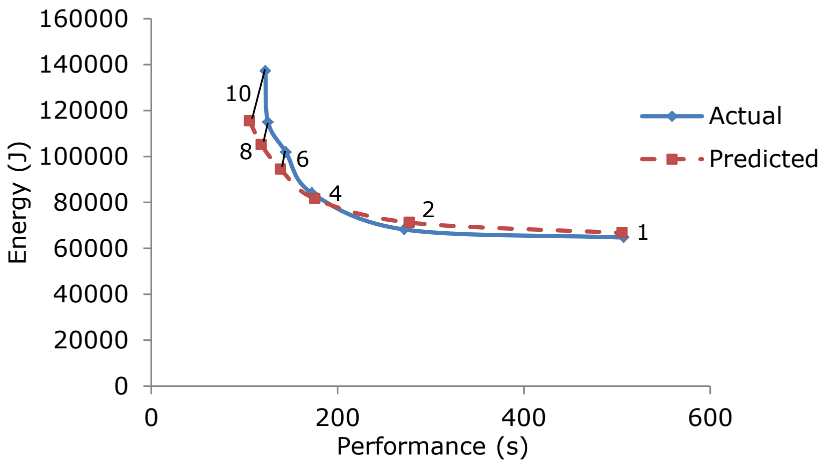


Figure 4.14: Actual and predicted Montage energy consumption and performance at the various scales the experiments can be executed in ‘Taurus’ cluster. The numbers in the plot represent the number of allocated nodes in the executed scenarios.

and Figure 4.14 shows the predicted and actual energy consumption and performance. The experiments suggest that the predictor can be used to make resource allocation decisions in 'Taurus' cluster: the actual runs indicate that using 8 nodes gives the best EDP for the workload evaluated, and the predictor suggests that 8 - 10 nodes are good choices.

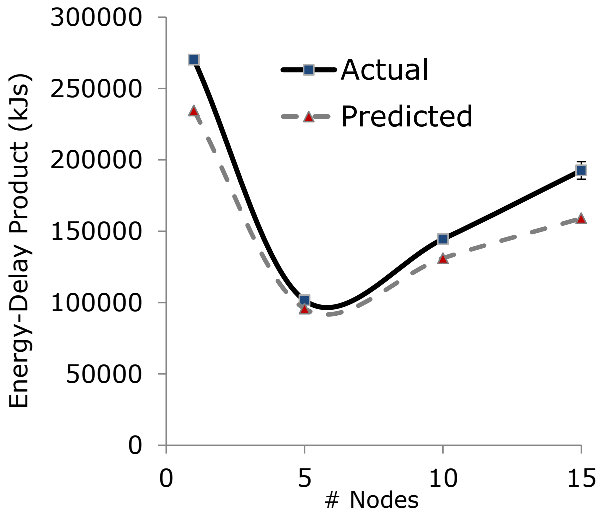


Figure 4.15: Actual and predicted Montage energy-delay product (EDP) on up to 15 nodes in 'Sagittaire' cluster.

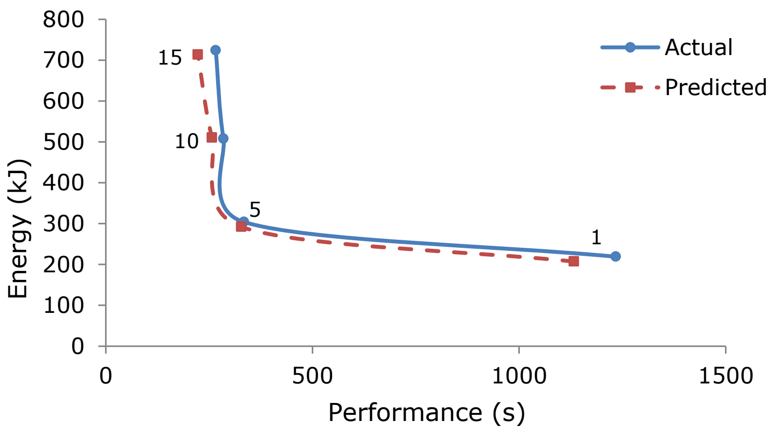


Figure 4.16: Actual and predicted Montage energy consumption and performance at the various scales in 'Sagittaire' cluster. The numbers in the plot represent the number of allocated nodes in the executed scenarios.

The predictor is also evaluated using 15 nodes (each of which has two 2.4GHz AMD Opteron CPUs (each with one core), 2GB RAM and 1 Gbps NIC) from Grid5000 'Sagittare' cluster. 'Sagittaire' has a larger number of machines, but it is less energy proportional than 'Taurus' cluster (servers in 'Sagittaire' are circa 2006 generation, while servers in 'Taurus' are circa 2012 generation). Figure 4.15 shows the predicted and average EDP on 15 nodes using the same Montage workload, while Figure 4.16 shows the predicted and actual energy consumption and performance. The predictor accurately shows that using 5 nodes for the workload is the best decision compared with other executed scenarios.

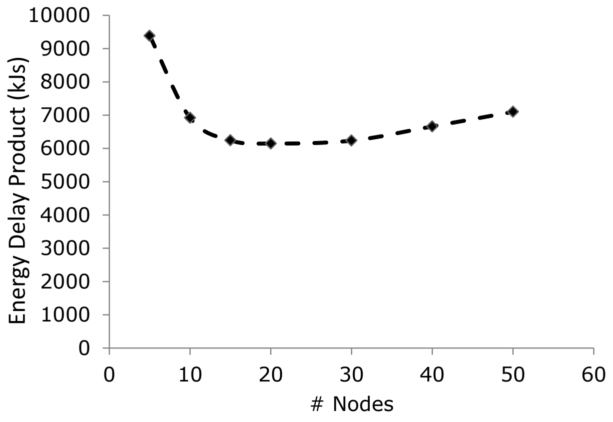


Figure 4.17: Predicted Montage energy-delay product (EDP) on up to 50 nodes in a hypothetical cluster in which idle power is 10% of the peak.

Figure 4.17 shows the Montage EDP results when the predictor is evaluated in a hypothetical cluster where the value of power states are the same as 'Taurus' except that the idle power is 10% of the peak power (in 'Taurus' the idle power is 40% of the peak). This hypothetical cluster represents a more energy proportional cluster than the clusters in Grid5000. The results suggest using 20 nodes is the best decision.

Summary: This section evaluates the predictor's ability to make resource allocation decisions when the users consider hybrid energy and performance metric. The accuracy is evaluated on different platforms. The best decision on the least energy-proportional platform ('Sagittaire') is choosing 5 nodes for the target workload. The best on the most energy-proportional platform (the hypothetical cluster) is using 20 nodes. The predictor accurately demonstrates that as the non-peak efficiency of the platform improves, the users can allocate more resources to optimize for energy-delay-product.

Chapter 5

Discussion

5.1 What Are the Causes of Inaccuracies?

Although some inaccuracy is expected from the simplicity of the model and its seeding mechanism as mentioned in §3, it is important to discuss in more depth the sources of inaccuracies. They fall in three main categories: first, the cluster used in the evaluation shares the same networking switch with two other clusters, thus interference can impact the accuracy of the seeding measurements: the platform characteristic gathered during seeding could be different from the one during experiments. This factor can be addressed by having more exclusive reservations to limit network interference.

Second, despite that the nodes in the cluster used for evaluation are homogenous machines, they can have different performance and power profiles. For instance, the peak power of machine can vary around 3%, and the idle power can vary around 5%. Thus, the performance and power parameters gathered from one node might not accurately reflect the characteristics of other nodes in the same cluster.

The third source, and more important in this context, is attributing inaccuracies precisely to time or to energy modelling. One approach to validate the inaccuracy source is to compare the energy prediction results between giving real time inputs to the energy predictor (in this case accurate breakdown of the time spent in each power profile is provided to energy model) and giving predicted times to the predictor. Experiments are conducted to log the full I/O trace of the synthetic benchmarks we used in Chapter 4. Consider Figure 4.2 that shows actual and predicted energy consumption for the synthetic benchmarks on DSS, the energy predictor achieves 5% inaccuracy for the pipeline benchmark, while for the reduce benchmark the inaccuracy is 16%. For the pipeline scenario, the predicted times and actual times are close (within 1%), thus the final 5% inaccuracy can be attributed to the energy model. For the reduce benchmark, the overall predicted time is underestimated by 6%, which leads to energy underprediction. When giving the actual time inputs to the energy model, the prediction inaccuracy is reduced to 9%. Thus, out of the 16% inaccuracy, 9% can be attributed to the energy model and 7% can be attributed to the time prediction.

5.2 What to Do to Improve Accuracy?

As discussed in the previous section, the nodes in a cluster can have different characteristics. One approach to increase accuracy is to perform performance and power seeding process on multiple nodes and obtain the average power value per state.

The energy model captures the major execution states, however, it does not include the energy spent on metadata path and workflow scheduling. Implementation the energy cost of those operations can improve the prediction accuracy.

Since the energy predictor requires time estimates from the performance predictor, and good time estimates lead to accurate energy prediction. As suggested by Costa et al. [26], currently the performance predictor does not capture workflow scheduler overheads, the workflow task launch overheads. Modeling these overheads can improve the accuracy of the time to solution, as well as the spent energy.

5.3 What Is the Advantage of Using the Proposed Energy Model Compared With Others?

The power states based model proposed in this thesis highlights the major execution states during the workflow run. The model treats the activities within a power state as a black box. Thus, it does not require modeling low-level activities. One can reason about the energy spent by non-energy proportionality, by the application's algorithmic operations, and by the underlying workflow supporting storage. Additionally, the inputs to this model are empirically average power per state, which are easy to obtain and does not require heavy system instrumentation.

As it is shown in Chapter 4, the predictor which implements the proposed energy model is sufficient for decision making in different scenarios (e.g., performance optimization, power-tuning techniques, resource allocation). Other energy models could increase the accuracy of the prediction at the expense of increasing complexity, but it is not clear the practical benefits brought by a more complex model.

5.4 Optimizing for Time VS. Optimizing for Energy

To optimize for time, one can use optimized storage configuration or add more resources (i.e., add more nodes). The former approach usually exploits data locality and location-aware scheduling, which generally reduces the amount of data transfers and, thus, it reduces the energy costs as well. Due to non-energy proportionality of the state-of-the-art platforms, idle power, however, remains a large portion of the total power consumption (for the cluster used in the evaluation the idle power is 40% of the peak power). Increasing the allocation size of a workload could improve performance at the cost of spending more energy. The experiments demonstrate that (Chapter 4.4), for a subclass of applications, it is additionally possible to optimize for energy only by using power-tuning techniques like CPU throttling. However, as demonstrate in the previous chapter, these techniques need to be carefully considered, as they can bring energy savings or lead to additional costs depending on the specific application patterns. The proposed energy prediction tool is particularly useful to support this type of decisions.

Chapter 6

Related Work

This chapter explains an overview of related work in the areas of energy consumption modeling and how this work differs.

Power state based modeling. Previous work use coarse grained power state based energy modeling in different contexts. Costa et al. [25] proposed using machine's idle power, peak CPU load power and peak I/O power, extra energy spent on hash computation to model data deduplication tradeoff. Ibtesham et al. [35] presents a coarse grained energy consumption model for rollback and recovery mechanisms. It uses three power profiles including application running, checkpointing compression, and checkpoint commit. While this thesis targets a distributed setup and enables exploration in a richer configuration space.

Resource utilization based modeling. Economou et al. [30] presents a non-intrusive method for modeling full-system power consumption based on the idle power and utilization metrics: CPU utilization, off-chip memory access count, hard disk I/O rate, network I/O rate. Fan at el. [32] uses CPU utilization as the main indicator of estimating power usage of individual machines. For the application domain this work targets, it is hard to obtain the resource utilization in a minimum system setup and predict the values in a large scale.

System events based modeling. SimplePower [53], Soft-Watt [39], and Mambo [47] provide low level analytical models tied to architectural events and uses simulations to predict power consumption. These more granular models, however, typically lead to a longer prediction time and are often highly coupled to the underlying architecture. Additionally, some parameters of the model are hard to obtain unless one executes the application in every possible configuration.

Simulation-based modeling. Similar to the proposed approach in this thesis, past work uses simulation instead of simply analytical modeling. For instance, OMNeT++ [52] provides general and accurate network modeling, while DiskSim [7] can model the storage devices at the hardware level. These tools could be modified and integrated to build a detailed simulator. However, due to the low component level simulation, they often lack fast time to solution. The proposed approach in this dissertation has achieved reasonable accuracy while remains lightweight and provides a simple and effective way to seed the model [27].

Time-series data based statistical modeling. Samak et al. [46] present an approach to analyze large power consumption datasets from computing infrastructures. They use Pig data processing on Hadoop for computing statistics and aggregating data. Then R framework is used to consume the time-series data and derive hourly and daily power consumption prediction models. The main goal of their work is to provide a large datasets processing pipeline and predict the consumption of whole infrastructures at a coarse level, while this paper aims at providing a framework for accurate consumption prediction of specific workloads and also evaluate the configuration choices, energy-performance trade-offs.

Analytical modeling of distributed applications. Some work use analytical models to evaluate the energy efficiency of distributed scientific applications. Feng et al. [34] developed a predictor for profiling power and energy characteristics of parallel scientific applications. Their results suggested for fixed problem size of some workloads, increasing the number of nodes always increases energy consumption but does not always improve performance, which is also observed in our evaluation. Ge et al. [37] have similar scope as this paper and uses two steps approach: (1) it develops analytical models for both energy and performance profiles of parallel scientific workload, and estimate the performance and energy costs for varying configurations (e.g., number of cores, CPU frequencies), (2) it explores the configuration space to find the optimal setting. The main difference from this paper is that Ge et al. [37] focus on parallel applications while this work focuses on the distributed storage layer of workflow applications that have various patterns and much more I/O operations. For the performance modeling it extended Amdahl's to derive speedups, while this work uses discrete event based simulation to obtain the runtime and obtain higher accuracies. Pakin et al. [42] focus on evaluating the energy savings when DVFS is enabled. It presents normalized energy and performance models using compute-boundedness and normalized frequencies, and evaluated energy prediction with various CPU frequencies. This work considers modeling the I/O operations in the memory and network to get better accuracy. Unlike Pakin et al. [42] that use compute-boundness in developing the models, Freehy et al. [36] use CPU criticality to evaluate the energy and performance tradeoff.

Distributed storage modeling. Some works focus on evaluating the energy efficiency in the distributed storage layer. EEffSim [43] presents a configurable energy simulator for modeling multi-server storage systems. It can model the the energy consumption of multiple energy-saving techniques, including write offloading, opportunistic spin-down and heterogeneous storage devices and others. However, the accuracy of the simulator for real world applications is not evaluated.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This thesis presents an energy consumption predictor for estimating workflow-based applications' energy usage. The accuracy of the predictor is evaluated using synthetic benchmarks that represent common data access patterns and two real world workflow applications. Additionally, the ability of the proposed predictor is evaluated to support choosing between different storage configurations (a default configuration - DSS, and a workflow optimized configuration - WOSS), to support configuration decisions for processor frequency scaling, and resource provisioning decisions. Overall, the experiments demonstrate that the predictor is a low cost, time-efficient tool for evaluating power-tuning techniques that target a multitude of scenarios and success metrics (e.g., energy, energy-delay product).

As described in the Preface, the energy prediction work presented in this thesis is one part of my contributions to the MosaStore research project. I also contributed to the general system development, performance prediction and provisioning project, cross-layer optimization project and supporting scientific data in the cloud project.

7.2 Future Work

Future work will improve the proposed energy prediction mechanism in multiple directions: (i) improve the accuracy of the energy prediction by addressing the points described in

Chapter 5, (ii) explore a richer space of different configuration choices (e.g., replication level, chunk size, other power tuning techniques), (iii) explore platforms that have different energy proportionality, (iv) apply different optimization criteria (e.g., dollar cost, joules per task).

Bibliography

[1] Amazon Elastic Compute Cloud (Amazon EC2). `http://http://aws.amazon.com/ec2/`.

[2] Fuse: Filesystem in userspace. `http://fuse.sourceforge.net/`.

[3] IBM BlueGene/P (BG/P). `http://www.research.ibm.com/bluegene/`, 2008.

[4] New Database Manager (NDBM) library, Berkeley.

[5] SPEC, The SPEC Power Benchmark. `http://www.spec.org/power_ssj2008/`.

[6] Stress. `http://people.seas.harvard.edu/~apw/stress/`.

[7] DiskSim. `http://www.pdl.cmu.edu/DiskSim/`.

[8] General Parallel File System. `http://www-03.ibm.com/systems/software/gpfs/`.

[9] Lustre file system. `http://lustre.opensfs.org/`.

[10] The Green Grid. `http://www.thegreengrid.org`.

[11] Michael Abd-El-Malek, William V. Courtright II, Chuck Cranor, Gregory R. Ganger, James Hendricks, Andrew J. Klosterman, Michael P. Mesnier, Manish Prasad, Brandon Salmon, Raja R. Sambasivan, Shafeeq Sinnamohideen, John D. Strunk, Eno

Thereska, Matthew Wachs, and Jay J. Wylie. Ursa minor: Versatile cluster-based storage. In *Proc. of the Conf. on File and Storage Technologies*, Dec. 2005.

[12] Samer Al-Kiswany, Lauro Beltrão Costa, Hao Yang, Emalayan Vairavanathan, and Matei Ripeanu. A Cross-Layer Optimized Storage System for Workflow Applications. In *submission to IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2014.

[13] Samer Al-Kiswany, Lauro Beltrão Costa, Hao Yang, Emalayan Vairavanathan, and Matei Ripeanu. A Software Defined Storage for Scientific Workflow Applications. In *Preparation*, 2014.

[14] Samer Al-Kiswany, Abdullah Gharaibeh, and Matei Ripeanu. The Case for a Versatile Storage System. *SIGOPS Oper. Syst. Rev.*, 44:10–14, March 2010.

[15] Samer Al-Kiswany, Matei Ripeanu, and Sudharshan S. Vazhkudai. A checkpoint storage system for desktop grid computing. *CoRR*, abs/0706.3546, 2007.

[16] Samer Al-Kiswany, Emalayan Vairavanathan, Lauro B. Costa, Hao Yang, and Matei Ripeanu. The case for cross-layer optimizations in storage: A workflow-optimized storage system. *CoRR*, abs/1301.6195, 2013.

[17] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, Oct. 1990.

[18] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, December 2007.

[19] Christian L. Belady. In the Data Center, Power and Cooling Costs more than the IT Equipment it Supports. February 2010.

[20] John Bent, Douglas Thain, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Miron Livny. Explicit control a batch-aware distributed file system. In *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1*, NSDI'04, pages 27–27, Berkeley, CA, USA, 2004. USENIX Association.

[21] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, Mei-Hui Su, and K. Vahi. Characterization of scientific workflows. In *Workflows in Support of Large-Scale Science, 2008. WORKS 2008. 3rd Workshop on*, pages 1–10, 2008.

[22] Franck Cappello, Eddy Caron, Michel Daydé, Frédéric Desprez, Yvon Jégou, Pascale Primet, Emmanuel Jeannot, Stephane Lanteri, Julien Leduc, Noredine Melab, Guillaume Mornet, Raymond Namyst, Benjamin Quetier, and Olivier Richard. Grid'5000: a large scale and highly reconfigurable grid experimental testbed. In *Grid Computing, 2005. The 6th IEEE/ACM Intl. Workshop on*, pages 8 pp.+, 2005.

[23] Wu chun Feng, Xizhou Feng, and Rong Ge. Green supercomputing comes of age. *IT Professional*, 10(1):17–23, 2008.

[24] Lauro Beltrão Costa, Samer Al-Kiswany, Abmar Barros, Hao Yang, and Matei Ripeanu. Predicting intermediate storage performance for workflow applications. In *Proceedings of the 8th Parallel Data Storage Workshop*, pages 33–38. ACM, 2013.

[25] Lauro Beltrão Costa, Samer Al-Kiswany, Raquel Vigolvino Lopes, and Matei Ripeanu. Assessing data deduplication trade-offs from an energy and performance perspective. In *2011 Intl. Green Computing Conf. and Workshops*, 2011.

[26] Lauro Beltrão Costa, Samer Al-Kiswany, Hao Yang, and Matei Ripeanu. Supporting Storage Configuration and Provisioning for I/O Intensive Workflows. In *submission to IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2014.

[27] Lauro Beltrão Costa, Samer Al-Kiswany, Hao Yang, and Matei Ripeanu. Supporting Storage Configuration for I/O Intensive Workflows. In *28th International Conference on Supercomputing (ICS2014)*, 2014.

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

[29] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[30] Dimitris Economou, Suzanne Rivoire, and Christos Kozyrakis. Full-system power analysis and modeling for server environments. In *In Workshop on Modeling Benchmarking and Simulation (MOBS*, 2006.

[31] EPA. EPA Report to Congress on Server and Data Center Energy Efficiency. Technical report, U.S. Environmental Protection Agency, 2007.

[32] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andr Barroso. Power provisioning for a warehouse-sized computer. In *The 34th ACM International Symposium on Computer Architecture*, 2007.

[33] Wu-chun Feng and Kirk Cameron. The green500 list: Encouraging sustainable supercomputing. *Computer*, 40(12):50–55, December 2007.

[34] Xizhou Feng, Rong Ge, and Kirk W. Cameron. Power and energy profiling of scientific applications on distributed systems. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers - Volume 01*, IPDPS '05, pages 34–, Washington, DC, USA, 2005. IEEE Computer Society.

[35] Kurt Brian Ferreira, Dewan Ibtesham, David DeBonis, and Dorian Arnold. *Coarse-grained Energy Modeling of Rollback/Recovery Mechanisms.* Mar 2014.

[36] Vincent W. Freeh, David K. Lowenthal, Feng Pan, Nandini Kappiah, Robert Springer, Barry Rountree, and Mark E. Femal. Analyzing the energy-time trade-off in high-performance computing applications. *IEEE Trans. Parallel Distrib. Syst.*, 18(6):835–848, 2007.

[37] Rong Ge, Xizhou Feng, and Kirk W. Cameron. Modeling and evaluating energy-performance efficiency of parallel processing on multicore based power aware systems. In *IPDPS*, pages 1–8. IEEE, 2009.

[38] Abdullah Gharaibeh, Samer Al-Kiswany, and Matei Ripeanu. Configurable security for scavenged storage systems. In *Proceedings of the 4th ACM International Workshop on Storage Security and Survivability*, StorageSS '08, pages 55–62, New York, NY, USA, 2008. ACM.

[39] S. Gurumurthi, A. Sivasubramaniam, M.J. Irwin, N. Vijaykrishnan, and M. Kandemir. Using complete machine simulation for software power estimation: the softwatt approach. pages 141–150, Feb. 2002.

[40] A. C. Laity, N. Anagnostou, G. B. Berriman, J. C. Good, J. C. Jacob, D. S. Katz, and T. Prince. Montage: An Astronomical Image Mosaic Service for the NVO. In P. Shopbell, M. Britton, and R. Ebert, editors, *Astronomical Data Analysis Software and Systems XIV*, volume 347 of *Astronomical Society of the Pacific Conf. Series*, page 34, Dec 2005.

[41] Ketan Maheshwari, Justin Wozniak, Hao Yang, Daniel S. Katz, Matei Ripeanu, Victor

Zavala, and Michael Wilde. Evaluating storage systems for scientific data in the cloud. In *5th Workshop on Scientific Cloud Computing (ScienceCloud)*, 2014.

[42] Scott Pakin and Michael Lang. Energy Modeling of Supercomputers and Large-Scale Scientific Applications. *IEEE*, 2013. In Proceedings for the IGCC 2013 : International Green Computing Conference.

[43] Ramya Prabhakar, Erik Kruus, Guanlin Lu, and Cristian Ungureanu. Eeffsim: A discrete event simulator for energy efficiency in large-scale storage systems. In *Energy Aware Computing (ICEAC), 2011 International Conference on*, pages 1–6. IEEE, 2011.

[44] Ioan Raicu, Ian T. Foster, and Yong Zhao. Many-Task Computing for Grids and Supercomputers. In *IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS08) 2008.*

[45] Ioan Raicu, Zhao Zhang, Mike Wilde, Ian Foster, Pete Beckman, Kamil Iskra, and Ben Clifford. Toward loosely coupled programming on petascale systems. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, pages 22:1–22:12, Piscataway, NJ, USA, 2008. IEEE Press.

[46] Taghrid Samak, Christine Morin, and H. Bailey, David. Energy consumption models and predictions for large-scale systems. In *The Ninth Workshop on High-Performance, Power-Aware Computing*, Boston, États-Unis, 2013. IEEE, IEEE.

[47] H. Shafi, P. Bohrer, J. Phelan, C. Rusu, and J. Peterson. Design and validation of a performance and power simulator for PowerPC systems. *IBM Journal of Research and Development*, 47(5):641–651, 2003.

[48] Takeshi Shibata, SungJun Choi, and Kenjiro Taura. File-Access Patterns of Data-Intensive Workflow Applications and their Implications to Distributed Filesystems. In *Proc. of the 19th ACM Intl. Symp. on High Performance Distributed Computing*, HPDC '10, pages 746–755, 2010.

[49] Emalayan Vairavanathan. Towards a high-performance scalable storage system for workflow applications. Master's thesis, The University of British Columbia, September 2014.

[50] Emalayan Vairavanathan, Samer Al-Kiswany, Lauro Beltrão Costa, Hao Yang, and Matei Ripeanu. MosaStore functional and design specification. 2012.

[51] Emalayan Vairavanathan, Samer Al-Kiswany, Lauro Beltrão Costa, Zhao Zhang, Daniel S. Katz, Michael Wilde, and Matei Ripeanu. A Workflow-Aware Storage System: An Opportunity Study. In *Cluster Computing and the Grid, IEEE Intl. Symp. on*, pages 326–334, 2012.

[52] A. Varga. Using the OMNeT++ Discrete Event Simulation System in Education. *Education, IEEE Trans. on*, 42(4), 1999.

[53] N. Vijaykrishnan, M. J. Irwin, H. S. Kim, and W. Ye. Energy-driven integrated hardware-software optimizations using simplepower. pages 95–106, 2000.

[54] Michael Wilde, Mihael Hategan, Justin M. Wozniak, Ben Clifford, Daniel S. Katz, and Ian T. Foster. Swift: A language for distributed parallel scripting. *Parallel Computing*, 37(9):633–652, 2011.

[55] Justin M. Wozniak and Michael Wilde. Case Studies in Storage Access by Loosely Coupled Petascale Applications. In *Proc. of the 4th Annual Workshop on Petascale Data Storage*, PDSW '09, pages 16–20, 2009.

[56] Hao Yang, Lauro Beltrão Costa, and Matei Ripeanu. Energy Prediction for I/O Intensive Workflows. In *submisson to 7th Workshop on Many-Task Computing on Clouds, Grids, and Supercomputers (MTAGS) 2014 (Co-located with Supercomputing/SC 2014)*.

[57] Ustun Yildiz, Adnene Guabtni, and Anne H. H. Ngu. Towards Scientific Workflow Patterns. In *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science*, WORKS '09, pages 13:1–13:10, New York, NY, USA, 2009. ACM.

[58] Zhao Zhang, Allan Espinosa, Kamil Iskra, Ioan Raicu, Ian T. Foster, and Michael Wilde. Design and evaluation of a collective io model for loosely coupled petascale programming. *CoRR*, abs/0901.0134, 2009.

[59] Zhao Zhang, Daniel S. Katz, Michael Wilde, Justin M. Wozniak, and Ian 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.