This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2415780, IEEE Transactions on Cloud Computing

Designing Self-Tuning Split-Map-Merge Applications for High Cost-Efficiency in the Cloud

Dinesh Rajan and Douglas Thain
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, Indiana 46556
Email:{dpandiar, dthain}@nd.edu

Abstract—Cloud platforms are attractive for executing large concurrent applications that require access to a pool of resources for concurrently executing the partitions of their workloads. Historically, application designers have tuned concurrent applications for specific hardware and platforms. But such approaches are not viable in cloud platforms as applications can be deployed on a variety of platforms and the operating environments can vary in each deployment. In this work, we argue and demonstrate that concurrent applications in cloud platforms must be self-tuning. First, we show that applications must incorporate a model of the overheads of operation. Second, we show that applications must determine their resource requirements and tune their operation to the operating conditions using estimations from the model. We build two self-tuning applications, E-Sort and E-MAKER, and demonstrate their ability to achieve high cost-efficiency by determining the right scale of partitions and resources to use for operation and adapting their behavior according to the characteristics of the deployed environment.

Index Terms—Cloud Computing, Distributed Execution, Scientific Applications, Data-intensive Applications, Data Partitioning, Workload Decomposition, Resource Provisioning.

1 Introduction

Resource- and data-intensive applications, such as data processing, bioinformatics, and molecular simulations typically run by partitioning their workloads into independent tasks, distributing the tasks for concurrent execution, and gathering the outputs to produce the final results [1]–[3]. The execution of such applications requires access to dedicated infrastructure, often at large scales, to achieve reasonable completion times. Previously, such modes of execution were only available in restricted, proprietary, and expensive setups, such as high-performance clusters and supercomputers.

Cloud platforms have alleviated these limitations by offering public and on-demand access to resources at scale. As a result, several resource- and data-intensive applications have been built or ported to run on cloud platforms [4]–[6]. These applications are deployed and run by the users or operators interested in harnessing their functions and capabilities. The operators deploy the applications to operate on the desired inputs using the resources they provision from cloud platforms.

The deployment of applications on cloud platforms incurs monetary costs and demands cost-efficient operation. In this work, we consider cost-efficiency in terms of the time and monetary cost of operation. Further, we consider the deployment and execution of applications using resources exclusively dedicated to each instance of the applications. Such deployments are common in research and scientific communities where the operators independently and directly run the applications using the on-demand access to resources [5], [7]. The applications can be deployed and executed in any of the cloud platforms accessible to the operators. As a result, the characteristics of the target operating environment (such as execution speed and network bandwidth) are unknown and unpredictable prior to deployment.

In the past, application designers would tune a concurrent application for specific hardware and operating environment (such as supercomputers and high performance clusters) and require the application to be executed in those environments. But if the target hardware is unknown or variable when the application is designed, the application must be self-tuning at runtime. In other words, **concurrent applications in cloud environments must be self-tuning to be cost-efficient**.

The performance and costs of concurrent applications is determined by the (logical) expression and (physical) realization of concurrency during their execution. The expression of concurrency pertains to the number of partitions of the workload while its realization involves the simultaneous execution of the partitions. In this work, we argue that applications must be self-modeling in order to determine and tune their logical and physical concurrency at runtime. The applications must incorporate a model of their execution formulating the gains and overheads of concurrency. We also argue that cost-efficient operation requires applications to explicitly exert control of the partitioning of the workload to tasks, the binding of data to tasks, and the submission of tasks for simultaneous execution.

Using the model and control of the parameters of concurrent execution, the applications tune and adapt their execution according to the characteristics of the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2415780, IEEE Transactions on Cloud Computing

deployed environment. The applications first measure the characteristics of the environment that influence their transfer overheads, processing overheads, and I/O overheads, and thereby, their cost-efficiency. The measurements are applied in the model to estimate the overheads of executing the defined workload under the current operating conditions. Using these estimates, the applications determine and adapt their logical and physical concurrency during runtime to achieve cost-efficient operation in the deployed environment.

We demonstrate the application-level modeling, control, and adaptations of concurrent execution in two applications - Elastic (E-Sort) and Elastic Maker (E-MAKER). E-Sort is a representative data processing application while E-Maker is a bioinformatics tool for annotating genomes. We evaluate the effectiveness of the techniques by considering their impact in minimizing the time and monetary cost of execution under various operating conditions. We demonstrate the application-level adaptations based on estimations derived from the model using measurements of the operating environment achieve higher cost-efficiency when compared to approaches that assume certain operating conditions or sample the environment once before execution.

The paper is organized as follows: Section 2 describes the architecture and construction of concurrent applications. Section 3 discusses the deployment of applications in the cloud and the challenges in achieving cost-efficient operation. Section 4 presents the application-level modeling, control, and adaptation techniques to overcome the challenges. Section 5 demonstrates and studies the techniques in E-Sort and E-MAKER. Section 6 reviews related work and Section 7 presents the conclusion.

2 Programming Model

In this work, we consider workloads that are built and executed using the paradigm of *split-map-merge*. This paradigm encompasses the bag-of-tasks [8], bulk synchronous parallel [9], scatter-gather [2], and the popular Map-Reduce [1] (where the split is done during the initial upload of the data to the filesystem of the execution framework) models of concurrent programming.

In the split-map-merge paradigm, the workload of the application is run by splitting the input into multiple partitions. Each partition is applied with a map function to produce their output. The individual outputs are then merged to produce the final result or output. A number of large-scale workloads are expressed and executed using the split-map-merge paradigm [10]–[12]. We provide a formal expression for the paradigm below.

$$Workload : f(N) \rightarrow O, \tag{1}$$

$$Split(N, k) : N \rightarrow \{s_1, s_2, \ldots, s_k\}, \tag{2}$$

$$Map(k) : f(s_i) \rightarrow O_i \;\; for\ i = 1, 2, \ldots, k, \tag{3}$$

$$Merge(k) : \{O_1, O_2, \ldots, O_k\} \rightarrow O. \tag{4}$$

Equation 1 describes the overall execution of the workload which performs a transformation on input N to produce output O. The split step in Equation 2 takes a parameter k and splits N into k partitions. The map step runs the transformation on each of the k partitions as shown in Equation 3. Finally, the merge step in Equation 4 aggregates the outputs of the individual map functions to produce the final output O. The merge function could be a simple concatenation of the outputs or a sophisticated function (e.g., merge of values in sorted order) depending on the workload. It is important to note the value of O is not affected by the choice of k. The concurrency in split-map-merge workloads results from the simultaneous execution of the map operations.

2.1 Architecture

We define applications that adapt their execution to the characteristics of the deployed environment as *elastic applications* [13], [14]. These applications are characterized by their flexibility in the partitioning of the workload, tolerance to failures, adaptability to the resources available for operation, and portability across different platforms. In this work, we implement the split-map-merge workloads as elastic applications. The workload of elastic applications can consist of a single, multiple, or iterative split-map-merge phases[1].

Elastic applications are typically implemented as a master-coordinator that describes and coordinates the concurrent execution of the workload. The coordinator partitions the defined workload into self-contained tasks that are described by their execution command, input files, and output files. The coordinator then submits the tasks for execution. The submitted tasks are dispatched to the provisioned instances for concurrent execution and retrieved on their completion. The dispatch of tasks includes the transfer of their input files and execution commands. Similarly, the retrieval of completed tasks includes the transfer of output data and files.

In summary, the execution of split-map-merge workloads in elastic applications has four components: *partitioning of workload into tasks, transfer of input and output data of tasks, execution of tasks, and merging of task outputs*.

Several middleware and execution frameworks are available for building and deploying elastic applications [15]–[17]. The challenges in determining the number of partitions of the workload and resources to provision exist in deployments using any of these frameworks. In this work, we chose the Work Queue framework [18] since it allows applications to select the partitioning of their workload and direct their concurrent execution, unlike other frameworks.

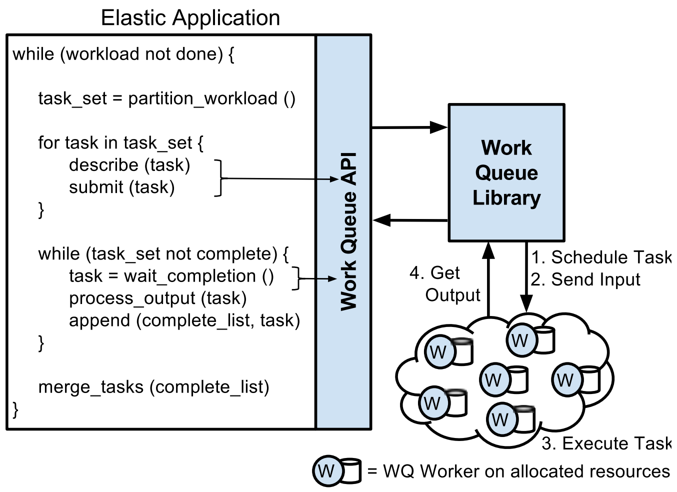
1. The presence of multiple or iterative split-map-merge phases does not impact the function or utility of the application-level techniques presented in this work. Rather, as we show in Section 4.3, they can be helpful in enabling the application-level techniques to measure the operating environment in each phase and apply the measurements to achieve efficient execution in the next phase.

2168-7161 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2415780, IEEE Transactions on Cloud Computing

2.2 Construction and deployment

We use the Work Queue [18] framework to implement the master-coordinator of elastic applications. Work Queue provides interfaces for describing the tasks of a workload, submitting the tasks for execution, and retrieving the results of the executions. Work Queue has been used to build a number of elastic applications in fields such as molecular dynamics, data mining, bioinformatics, and fluid dynamics [11]–[14], [19].


```
while (workload not done) {

    task_set = partition_workload ()

    for task in task_set {
        describe (task)
        submit (task)
    }

    while (task_set not complete) {
        task = wait_completion ()
        process_output (task)
        append (complete_list, task)
    }

    merge_tasks (complete_list)
}
```

Fig. 1: Overview of the construction and operation of elastic applications using the Work Queue framework. The code on the left is an outline of an elastic application.

The Work Queue framework is based on a master-worker execution model. The applications use its API - in C, Perl, or Python - to build the master-coordinator that describes, submits, and aggregates tasks. The Work Queue library handles the scheduling and mapping of tasks onto workers, transfer of data to workers, and rescheduling of failed tasks. Figure 1 presents the outline of applications typically constructed using Work Queue.

The submitted tasks are executed by the Work Queue workers. The workers are lightweight standalone processes run on the provisioned instances. They connect to a specified master and perform actions dictated by the master, such as transferring data and executing tasks.

Work Queue requires applications to explicitly specify the software and data dependencies for the tasks so the environment needed for execution can be created at the workers without concern for the native execution environment. This also enables Work Queue to provide data management facilities, such as caching and scheduling policies that favor workers with cached data.

3 Challenges in Cloud Environments

The time and cost of operation of elastic applications are determined by the gains achieved in the concurrent execution of tasks and the overheads incurred in the split, merge, and data transfer phases. The gains and overheads are determined by the number of partitions and resources chosen for operation. They are also influenced by the characteristics of the operating environment. For instance, the network bandwidth influences the transfer overheads while the size of the RAM at the provisioned instances influences the overheads of executing the map functions on the partitions.

In this work, we focus on operation using the on-demand instances in cloud platforms. These instances can be provisioned and terminated at convenience and are metered to incur charges only for the duration of use. To simplify exposition, we assume the instances cost \$1 per hour and incur \$0.01 for every gigabyte of data transferred to and from the instances[2]. Like many cloud platforms, we compute the operating costs by rounding the operating times to the nearest hour.

Figure 2 illustrates the impact of the partitions and the characteristics of the operating environment on the time and costs of running E-MAKER on the Anopheles Agambiae genome. The estimations plotted in the figure are derived using the model described in Section 4.1. The plots assume the number of partitions and the instances provisioned for operation are equivalent. From Figure 2, we observe the running time and operating costs exhibit varying trends for different network bandwidth. Further, Figure 2b shows the operating costs for various partitions exhibit irregular patterns due to the effects from the use of the hourly boundaries for calculating costs. The operating costs drop at partitions where the time of operation falls to the next lowest hourly boundary. In summary, Figure 2 shows that the choice of the number of partitions and instances to provision are critical to the cost-efficient operation of applications.

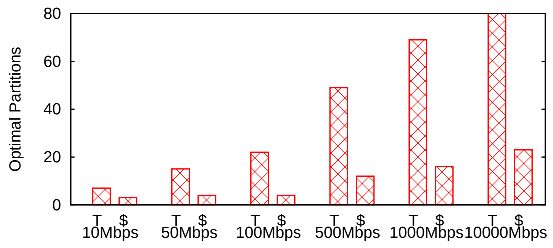


Fig. 3: Illustration of the optimal partitions that achieve minimum running time (denoted by T) and operating costs (denoted by \$) for annotating the Anopheles Agambiae genome under different network bandwidth values.

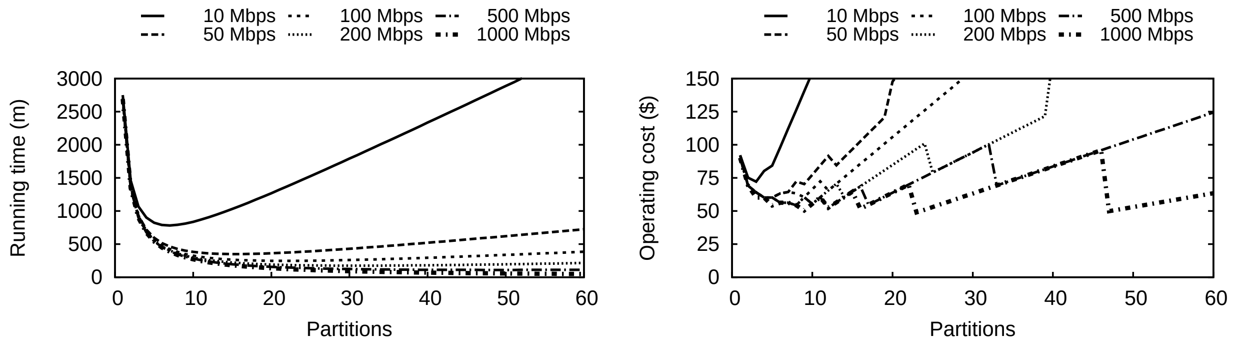
Figure 3 plots the partitions that achieve the minimal operating time and the lowest operating costs in Figure 2. It shows the optimal partitions to vary with network bandwidth. This demonstrates that performance and cost cannot always be optimized simultaneously, and so the partitioning must take into account the differing objectives of each user.

3.1 Common approaches

A common approach to run concurrent applications is to delegate the partitioning and submission of the

2. Our observations remain the same with the prices in commercial platforms such as Amazon EC2.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2415780, IEEE Transactions on Cloud Computing



(a) Running time (T)

(b) Operating costs ($)

Fig. 2: Estimated running time (in minutes) and operating costs (in $) of E-MAKER for annotating the Anopheles Agambiae genome. The estimations are shown for different network bandwidth. The operating costs exhibit irregular patterns due to the use of hourly boundaries in determining the cost of use.

partitions for execution to the underlying middleware. Consider Hadoop [15], a widely adopted middleware for executing concurrent and data-intensive workloads expressed using the MapReduce paradigm [1]. Hadoop relies on a distributed file system, such as Hadoop Distributed File System (HDFS) [20], for managing data during operation. HDFS partitions the input data and stores the partitions across the nodes provisioned for operation. HDFS arbitrarily partitions the data into blocks (default size of 128 MB) regardless of the workload and the concurrency feasible during its operation. HDFS leaves the optimal tuning of the block sizes to the operators (and not the users) of the cluster and does not provide guidance on an optimal partitioning strategy for executing the defined workload.

In summary, the partitioning strategy is fixed and globally enforced on every application executed in Hadoop. Such strategies are common in shared environments where resources are provisioned and maintained to support a single instance of the application that runs multiple workloads and serves the needs of multiple users over a prolonged duration. The pool of resources in such environments is centrally controlled and administered by the middleware, and the costs of maintaining the resources are shared among the operators and users.

The use of arbitrary and global policies for operation results in cost-inefficiencies in environments where each instance of the application is deployed on resources exclusively dedicated for their operation. This effect can be inferred from Figure 3 where the partitions that achieve minimal operating costs vary depending on the operating environment. To achieve cost-efficiency, the workloads must be partitioned and the resource requirements must be estimated according to the characteristics of the deployed operating environment. These decisions must be made in every deployment of the application since the operating environment can vary between deployments. Further, these decisions must be revised and adapted during operation since the characteristics of the operating environment, such as network bandwidth, are liable to dynamically change.

3.2 Solutions

The cost-efficient operation of concurrent applications demands well-informed choices on the number of partitions and simultaneous executions of the partitions. Previously, these decisions were made during application design by benchmarking performance on specific hardware and operating environments [21], [22]. These techniques proved successful when the operation of the applications were restricted to the hardware and operating environments considered during design.

The access to a variety of cloud platforms and types of resources offered in these platforms implies that applications cannot be built and tuned for specific operating environments. In addition, the use of commodity hardware and the sharing of the hardware among multiple users often leads to variations in the characteristics of the operating environments in cloud platforms [23]. In other words, the operating environment is unknown and unpredictable when the application is designed. Therefore, the operating parameters, such as the number of partitions and instances to provision, that achieve cost-efficient operation must be determined at runtime.

To correctly determine the parameters for cost-efficient operation, we argue and demonstrate the following principles as solutions:

1. **Applications must be self-modeling** by formulating and incorporating a model of the performance and overheads of their runtime components.
2. **Applications must be self-tuning** by exerting control over the partitioning and concurrent execution of the workload and dynamically adapting the execution according to the observed operating environment.

4 Techniques

We now describe the techniques to realize self-modeling and self-tuning applications. The techniques model, control, and adapt the logical and physical concurrency of the applications during operation.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2415780, IEEE Transactions on Cloud Computing

4.1 Application-level modeling

The runtime performance of elastic applications is determined by the partitioning, task execution, data transfer, and merge components described in Section 2.1. Accordingly, we model the operating time of elastic applications as follows:

$$T_{operation} = T_{partition} + T_{tasks} + T_{data} + T_{merge}. \quad (5)$$

Note this model of the running time differs from Amdahl's law [24] in that $T_{partition}$ and/or T_{merge} at the coordinator increases with the number of partitions.

The overheads of the partition $T_{partition}$ depend on the size of the input data N and the number of partitions K. We model this as a linear relationship:

$$T_{partition} = (a * N) + (b * K), \quad (6)$$

where a and b are constants that reflect the costs of reading (input) data and creating a partition respectively.

As we noted in Section 2, the merge overheads T_{merge} can vary based on the implementation and characteristics of the workload. Therefore, these overheads are formulated and discussed individually in Section 5.2.

The execution time of the tasks T_{tasks} is determined by the size of the input and the partitions. If input N is partitioned into K tasks, the execution time of a task is

$$T_{task} = T(\frac{N}{K}). \quad (7)$$

If R is the number of instances provisioned for operation, the execution of K tasks is prolonged by a factor of $\lceil K/R \rceil$ (as only R tasks can be executed simultaneously). Thus, the total execution time of tasks is

$$T_{tasks} = T_{task} * \lceil \frac{K}{R} \rceil. \quad (8)$$

The data overheads T_{data} collectively represents the input T_{inputs} and output $T_{outputs}$ transfer overheads. The inputs consists of the software and unique data dependencies of the tasks. The software dependencies include executables, scripts, and libraries are required for execution and are common across tasks. These dependencies can be transfered once and cached for subsequent tasks. In contrast, the unique data dependencies are specific to each task and must be transferred for every execution. These dependencies specify the configurations and data for operation in each task.

$$T_{data} = (Data_{in} + Data_{out})/BW, \quad (9)$$

$$Data_{in} = size(N) + R * size(software), \quad (10)$$

$$Data_{out} = size(N), \quad (11)$$

where BW represents the available network bandwidth.

The model in Equation 5 is applied to determine the number of partitions K and the number of instances R to provision for executing the defined workload. Further, the estimations from the model enable the applications to tune and adapt the partitions according to the characteristics of the deployed environment, such as network bandwidth, physical memory allocated at the resources, and the processing capacity for operation.

4.1.1 Assumptions

The model in Equation 5 assumes a scheduling strategy that operates in the following order: dispatch the tasks submitted for execution to the provisioned instances (this includes transfer of the inputs), wait for the tasks to finish execution, retrieve the outputs and results of the executions, and so forth. It is also assumed that all the instances for operation are provisioned at the same time and the workers running on them are connected to the master before the dispatch of tasks begins.

The model for task executions in Equation 8 assumes the instances provisioned for operation are homogeneous in their hardware and processing capabilities. In cloud platforms, this assumption is satisfied by provisioning instances of the same size. Further, it assumes that each task consumes a single CPU core.

Finally, the model of the data overheads in Equation 9 assumes single-threaded communication where data is transferred to one worker at a time. A multi-threaded mode is helpful when communicating with heterogeneous instances with wide differences in their processing capabilities. Otherwise, we expect the impact from multi-threaded communication on the estimations to be minimal since the bandwidth remains the same while being shared across multiple threads.

4.1.2 Cost-efficiency metrics

We formulate the operating cost $C_{\$}$ of the applications using the time of operation modeled in Equation 5, the instances provisioned for operation R, and the data transfered during operation.

$$C_{\$} = \$_{IH} * R * H_{operation} + \$_{GB} * (Data_{in} + Data_{out}), \quad (12)$$

where $H_{operation}$ is $T_{operation}$ (defined in Equation 5) rounded to the nearest hour. $\$_{IH}$ represents the cost incurred per instance per hour of use while $\$_{GB}$ represents the cost charged per gigabyte of data transfer to and from the instances. To simplify analysis and provide a general context, we assume $\$_{IH}$ to be \$1 and $\$_{GB}$ to be \$0.01.

We note that favorable trade-offs often exist between the time and cost of operation. For example, in Figure 2b, the lowest operating cost (\$60.16) under a bandwidth of 100Mbps is achieved when running with 4 partitions. However, accommodating a 0.33% increase in the operating cost (\$60.36) for operation with 9 partitions leads to a 50% decrease in the operating time. This is because operation with 9 partitions increases the gains from concurrent executions and lowers the time of operation such that it matches the gains in cost when running with 4 partitions. The small increase in cost for the operation with 9 partitions results from the increase in the transfer costs for the software dependencies.

In our evaluations, we find it useful to compute and use a metric called *Cost-Time product* to consider these trade-offs and assign equal importance to the time and

2168-7161 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2415780, IEEE Transactions on Cloud Computing

cost of operation.

$$Cost\text{-}Time\ product = C_{\$} * T_{operation}. \tag{13}$$

We note this metric is only one of several ways of expressing cost-efficiency since different weights may be assigned to the time and cost of operation based on the preferences of the operators. At the same time, we note the model can be easily extended to provide estimations on the cost-efficiency metrics preferred by the operators.

4.2 Application-level control

The application-level model in Section 4.1 provides estimations on the performance and overheads of the runtime components. However, to regulate the overheads and achieve cost-efficiency, the application must explicitly direct and control the following actions using estimates from the model.

Partitioning workload into tasks: The applications must define the partitioning of the workload into tasks using estimates from the model. The number of tasks created for operation also dictates the overheads associated with the task executions, data transfers, and merge operations. As a result, control over the number of tasks enables applications to lower the running time and minimize the incurred overheads in their deployed environments.

Binding data to tasks: The applications must explicitly bind the data dependencies to the created tasks. This control enables the application to manage the data transfer overheads of the tasks. The control over the binding of data is also necessary for the adaptations during operation that adjust the partitioning of the workload according to the observed conditions.

Merging outputs of tasks: The application-level control of the partitioning requires similar control over the merge operations so the outputs of tasks created from the partitions are correctly aggregated to produce the final results. This control also helps correctly estimate the overheads associated with the merge phase.

Submitting tasks for execution: The applications must direct the submission of tasks for execution and thereby, the number of simultaneous executions. This control enables applications to manage the overheads associated with transferring the common software dependencies. For example, when the transfer overheads associated with the software dependencies are large, the application can regulate the number of simultaneous executions so the transfer of these dependencies is minimized. This is because the common dependencies are cached for subsequent tasks after their initial transfer.

4.3 Application-level adaptation

Elastic applications cannot assume, predict, or control the characteristics of the operating environment in which they are deployed. Therefore, the applications must adapt their operation to the characteristics of the deployed environment. We focus on adaptations of the two parameters that dictate the time and cost of operation: the number of partitioned tasks and the number of instances used for operation.

A simple approach for determining the number of tasks and resources to provision involves the global enforcement of default values, or requiring the operators or users to manually determine them. This approach is employed in Hadoop [15] and illustrated in Figure 4a. While this approach provides operators with the ability to define and control the runtime behavior, it requires detailed knowledge of the characteristics of the workload and the overheads of operation in the deployed environment. In addition, the effectiveness of this approach requires tight control over the operating environment to provide consistent characteristics throughout operation.

We present two techniques to determine the number of partitions and resources for running the defined workload in the deployed environment without operator intervention. The first technique performs an initial assessment of the operating environment using a sample execution and resource allocation. It applies the measurements in the model to determine the operating parameters that achieve cost-efficient operation. This enables the optimal operation of applications in any deployed environment and operating conditions. This technique is also similar to the approach suggested by cloud providers for determining the right type of hardware and instances for running a workload [25]. Figure 4b illustrates this technique for adapting the operation according to the characteristics of the deployed environment. The effectiveness of this technique requires the operating conditions that impact the performance of the applications to remain unchanged during operation.

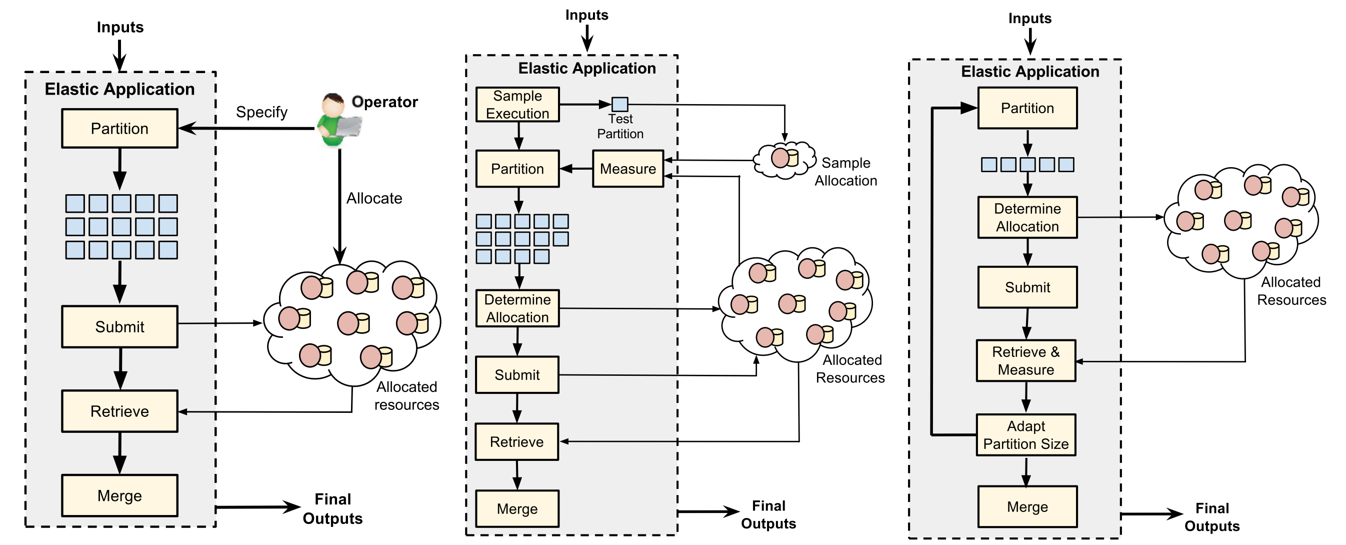
The operating conditions in cloud platforms, especially network bandwidth, are prone to vary during operation due to multi-tenant effects such as congestion, varying load on the shared network links, and oversubscription of the networking hardware. This impacts the cost-efficient operation of applications whose overheads of operation are influenced by these conditions. In this work, we focus on the adaptations to changes in the network bandwidth during operation.

To handle changes in the operating conditions during runtime, the second technique periodically measures the operating conditions and dynamically adapts the number of partitions and instances chosen for operation. This technique progressively partitions the workload, measures the conditions during operation of the submitted partitions, and applies the measurements in determining the number of partitions and instances for operating the remainder of the workload under the current conditions. Figure 4c illustrates this approach. In this work, the adaptations do not attempt to repartition already dispatched partitions since this requires complex feedback mechanisms (such as the ability to continously monitor the global state and cost of operation on the partitions) which is beyond the scope of this paper.

The operating conditions measured in both adaptation

2168-7161 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2415780, IEEE Transactions on Cloud Computing



(a) The manual approach to partitioning using user-specified partition sizes. (b) The sample execution-based approach that performs an assessment of the operating environment before operation. (c) The adaptive approach that continually measures and adapts to the characteristics of the operating environment.

Fig. 4: Illustration of the different strategies for partitioning the defined workload in elastic applications.

techniques include the execution overheads of the tasks, the network bandwidth, the local overheads of partition and merge, and the size of physical memory allocated at the provisioned resources. The measurements are incorporated in the model expressed in Equations 5, 12, and 13 to determine the current cost of operation and estimate the cost for running the remainder of the workload. The estimations are made considering the hourly boundaries in the metering of the provisioned instances to maximize their usage. The partitions and the instances used for operation are provisioned, maintained, and terminated based on the estimations of the current costs and the costs for operating the remainder of the workload. The adaptations in the presented technique is regulated by the size of the partitions since the measurements are performed at the dispatch and completion of partitions. Further, the adaptations are terminated when it is determined that the remainder of the workload can be completed within the upcoming hourly boundary in the metering of the provisioned instances.

5 Experimental Analysis

We apply the presented techniques in building two self-tuning applications - Elastic Sort (E-Sort) and Elastic MAKER (E-MAKER). E-Sort performs sorting by partitioning the data and concurrently sorting the partitions on provisioned instances. The sorted partitions are then merged at the master-coordinator to produce the final sorted sequence. E-Sort uses the GNU Sort tool as its kernel for sorting the partitions. We use E-Sort as a representative application whose workload resembles many data analysis, mining, and processing workloads.

The second application, E-MAKER, is a bioinformatics program for annotating genome sequences [12], [13]. The annotations enable biologists to identify the presence of various cellular elements and their contributions to the functions of the genome. The annotations are performed by comparing the subject genome against a set of reference sequences and identifying similarities. E-Maker partitions and dispatches the sequences for concurrent annotation. It uses the MAKER tool [26] as the kernel for annotating the partitioned sequences.

Our goal in this study is not to build optimized implementations of the applications but to use the applications to demonstrate and evaluate the application-level techniques in effect. Besides, the techniques are necessary irrespective of the optimizations in the implementations.

As we noted earlier, the merge overheads for E-Sort and E-MAKER differ due to the workload and the implementation of the merge algorithm. In E-Sort, the partitions are merged using a k-way merge algorithm that iteratively compares the records in the partitions and aggregates them in sorted order. The asymptotic running time of the algorithm is $O(N * K)$. The merge overheads of E-Sort are modeled as

$$T_{merge} = (c * N * K) + (d * N), \tag{14}$$

where c represents the cost of the comparisons in the merge algorithm and d is the cost associated with reading the sorted records in the partitions. On the other hand, the merge in E-MAKER is trivial since the results of the tasks - the annotated sequences - are simply concatenated in a output directory. Hence, we model the overheads of merge to be constant and negligible.

Organization: We begin our analysis of self-tuning applications by observing the effects of the number of partitions on the operating time and cost. We observe differences in the effects of the number of partitions on the overheads of partition, merge, data transfer, and task executions in E-Sort and E-MAKER. The application-

2168-7161 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2415780, IEEE Transactions on Cloud Computing

level models in E-Sort and E-MAKER provide estimates on the time and cost of operation considering these effects. We experimentally validate these estimations with the goal of not showing estimations that perfectly match with the observed values, but to show the effectiveness of the model in providing information about the overheads of operation and their impact on time and cost.

The validations enable us to utilize the model in studying the effects of the characteristics of the workload and operating environment on the time and cost of operation using different partitions. The study establishes that decisions on the number of partitions and instances for cost-efficient must be made considering the characteristics of the workload and operating environment. We then experimentally demonstrate the self-tuning capabilities of E-Sort and E-MAKER in determining the number of partitions and instances to provision for cost-efficient operation. The applications utilize the application-level model and control to determine these parameters.

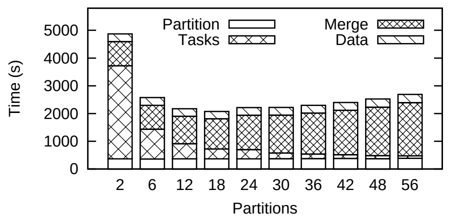
Our analysis considers the number of instances provisioned for operation to be equivalent to the number of partitions chosen for operation as it enables the simultaneous execution of the partitions. However, it may be useful to create smaller partitions to achieve faster failure recovery, manage resource consumption, and obtain measurements on the operation at smaller intervals. Therefore, we study the effects of *over-partitioning* where the number of partitions is greater than the instances determined for cost-efficient operation. We observe the impact of over-partitioning to be negligible when the partition and merge overheads are minimal. In such cases, over-partitioning can be useful for adapting to changing operating conditions without incurring additional overheads. We evaluate and demonstrate the dynamic adaptation technique that progressively over-partitions and runs the workload when the characteristics of the deployed environment, such as network bandwidth, vary during operation.

Experimental platforms and inputs: Our evaluations are done using on-demand instances from Microsoft Azure [27] - a commercial cloud platform, FutureGrid [28] - a national infrastructure that provides an IaaS testbed for building cloud applications, and Notre Dame CRC - a campus-wide infrastructure at the University of Notre Dame that offers access to IaaS instances.

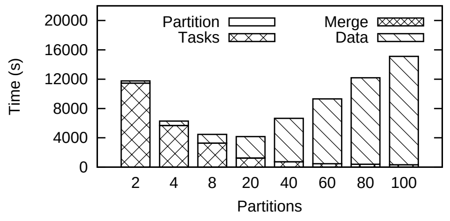
The E-Sort runs in our evaluations operate on 2 billion records that total 11GB in size. The E-MAKER runs operate on 800 contiguous sequences of the Anopheles Agambiae PEST strain which amount to 7.5MB. The software overheads for E-Sort consist of the transfer of the GNU Sort executable which is 100KB. In E-MAKER, the software overheads include the reference dataset and the libraries required for the execution of the MAKER tool and are 4GB in size.

5.1 Effects of the number of partitions

We experimentally observe the impact from the number of partitions on performance and break down the impact on each of the runtime components. Figure 5 plots the individual runtimes of the partition, task execution, merge, and data transfer components of E-Sort and E-MAKER. In this figure, the number of instances provisioned for operation is equivalent to the number of partitions.



(a) E-Sort



(b) E-MAKER

Fig. 5: Comparison of the overheads of the partition, task execution, data transfer, and merge operations during operation of E-Sort and E-MAKER.

Figure 5a shows the operating time of E-Sort for different partitions is dictated by the task execution times and merge overheads. The task execution times decrease exponentially while the merge overheads increase linearly as the number of partitions increases. The opposing trends in the task execution and merge overheads results in a running time that decreases with increasing partitions until the merge overheads outweigh the decrease in the task execution times.

In contrast, the operating time of E-MAKER, plotted in Figure 5b, is determined by the data transfer overheads and the task execution times. In E-MAKER, the data overheads increase linearly due to the software dependencies being transferred to each compute instance where the tasks are executed. As a result, the increase in the data overheads offset the gains in the concurrent executions as the partitions increase.

In summary, we observe two distinct patterns in the impact of the partitions on the operation of the two applications. The concurrency in operation is counteracted by the partition and merge overheads in E-Sort and the data overheads in E-MAKER. The model formulated in Section 4.1 provides estimations on these overheads and their impact on the time and cost of operation.

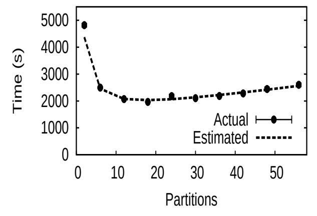
5.2 Validation of the estimations from the model

We validate the application-level model by comparing the estimations from Equations 5, 12, and 13 against the values observed in operation. Figure 6 presents the comparison of the estimated and actual time, cost, and cost-time product of operation for E-Sort and E-MAKER. The observed values for E-Sort were recorded

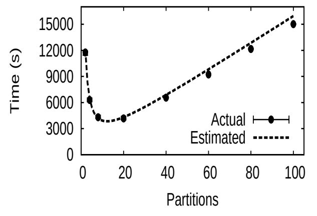
2168-7161 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2415780, IEEE Transactions on Cloud Computing

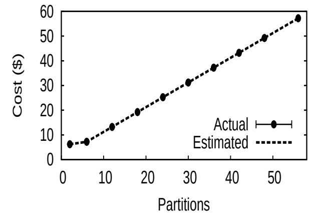
during operation on Microsoft Azure instances when the network bandwidth was measured at 800 Mbps. On the other hand, E-MAKER was observed in operation on FutureGrid instances when the network bandwidth was 200 Mbps. As we described earlier, the cost of operation on these platforms is computed after rounding the time of operation to the nearest hour.



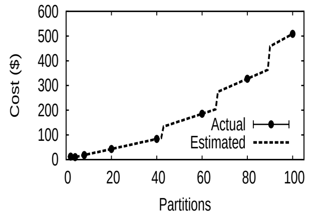
(a) $T_{operation}$: E-Sort



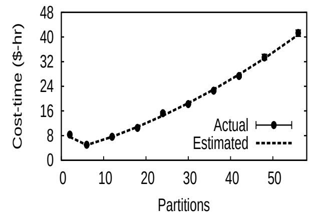
(b) $T_{operation}$: E-MAKER



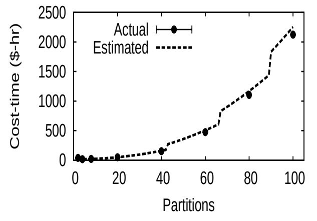
(c) $C_{\$}$: E-Sort



(d) $C_{\$}$: E-MAKER



(e) *Cost-time product*: E-Sort



(f) *Cost-time product*: E-MAKER

Fig. 6: Comparison of the observed values during operation with the estimations from the model for E-Sort and E-MAKER. The run times were averaged over 3 runs and the error bars describe the observed minimum and maximum values.

The estimations from the model use the same values for the execution time of a task in Equation 7 and the constants in Equations 6 and 14 as the measurements made from the execution of the sample partitions by the respective applications. As before, we maintain the instances (*R*) for operation to be equivalent to the number of partitions (*K*) in Figure 6.

The model correctly estimates the overheads of concurrent operation and their impact on the time and cost of operation. Figure 6 shows the estimations from the model on the time, cost, and cost-time product of operation reflect the values observed during operation. The validation of the model enables us to use the estimations from the model in analyzing the operation of applications with different characteristics of the workload (such as higher task execution times), operating parameters (such as number of partitions), and operating environments (such as network bandwidth). Further, the estimations from the model can be applied in correctly identifying the optimal number of partitions and instances to provision for operation.

5.3 Effects of the characteristics of the workload and operating environment

We study the choice of the number of partitions by considering the impact of the characteristics of the workload and operating environment on cost-efficient operation. In this study, we compare workloads with different task execution overheads that incur the same partition, merge, and data overheads. That is, we vary the gains in concurrency by increasing the execution times of the tasks relative to the other overheads of operation. These configurations are analogous to applications with similar but complex workloads where the task execution overheads are dominant in the overheads of operation.

We also consider operation under different network bandwidth (which impacts the transfer overheads) as it can vary between deployments due to differences in the platform configurations and hardware. Further, the network resources are often shared among multiple tenants of IaaS and PaaS platforms resulting in variations of the bandwidth from traffic patterns, congestion, and demand for resources. Finally, we study the effects of the size of physical memory (which impacts the I/O overheads in task executions) allocated at the instances.

Figures 7 and 8 illustrate the effects of the characteristics of the workload and network bandwidth on the operating time, costs, and cost-time product for various partitions in E-Sort and E-MAKER respectively. In these figures, each row plots the operation for the task execution times (T_{task}) that are multiples (1x, 2x, and 5x) of the value observed in Figure 5. We artificially introduced delays in the execution of the tasks to inflate their execution times to the desired proportion. The overheads of partition and merge operations are the same as plotted in Figure 5.

The increase in the execution time of the tasks relative to the partition, merge, and data overheads leads to an increase in the gains realized from concurrent operation. Therefore, the number of partitions that achieve lower operating times increases with higher task execution times as observed in Figures 7 and 8. Similarly, an increase in network bandwidth lowers the data transfer overheads and increases the gains realized from operation with higher number of partitions. This effect is seen in Figure 8 for E-MAKER due to the large transfer overheads from software dependencies.

As we noted in Section 3, the operating costs exhibit irregular trends due to the rounding of the operating time to the nearest hour. That is, the cost and cost-time product of operation with increasing partitions in Figures 7 and 8 are influenced by the magnitude of the decrease in the operating time and if the decrease results in a drop to the next lowest hourly boundary.

Finally, Figure 9 plots the estimated and observed task execution times for sorting different sizes of data on an

2168-7161 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2415780, IEEE Transactions on Cloud Computing

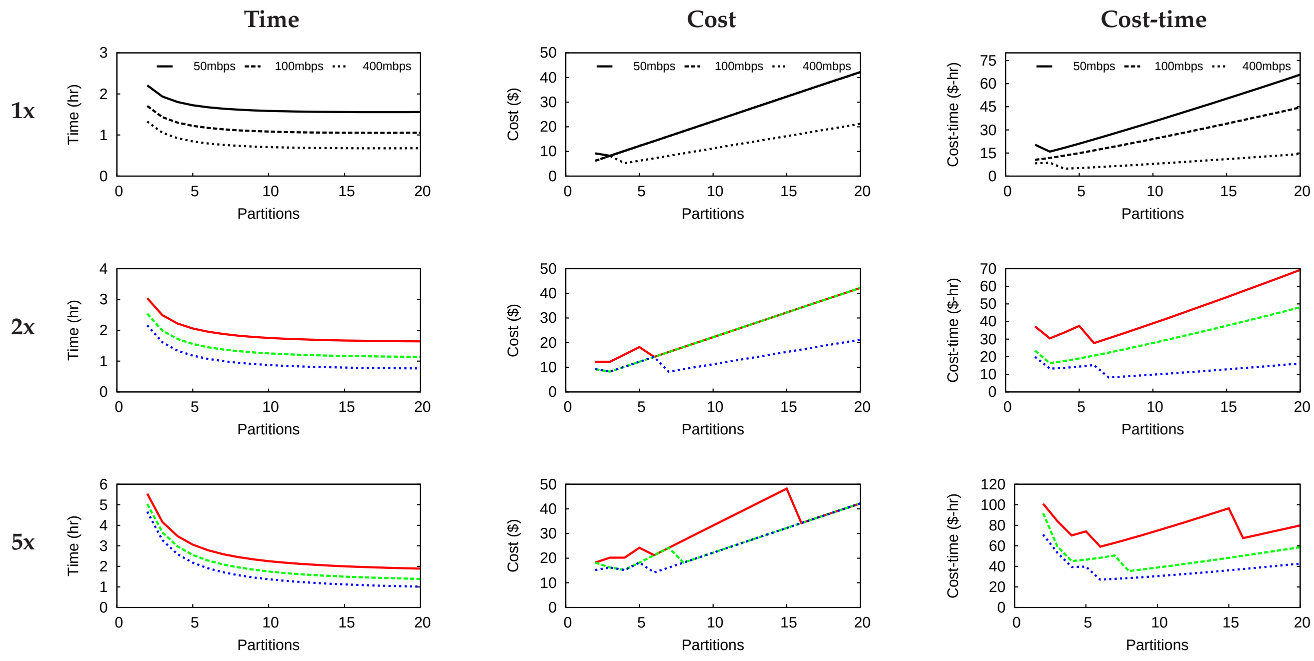


Fig. 7: Estimated operating time, operating costs, and cost-time product of E-Sort for sorting 2 billion records totaling 11GB under various characteristics of the operating environment.

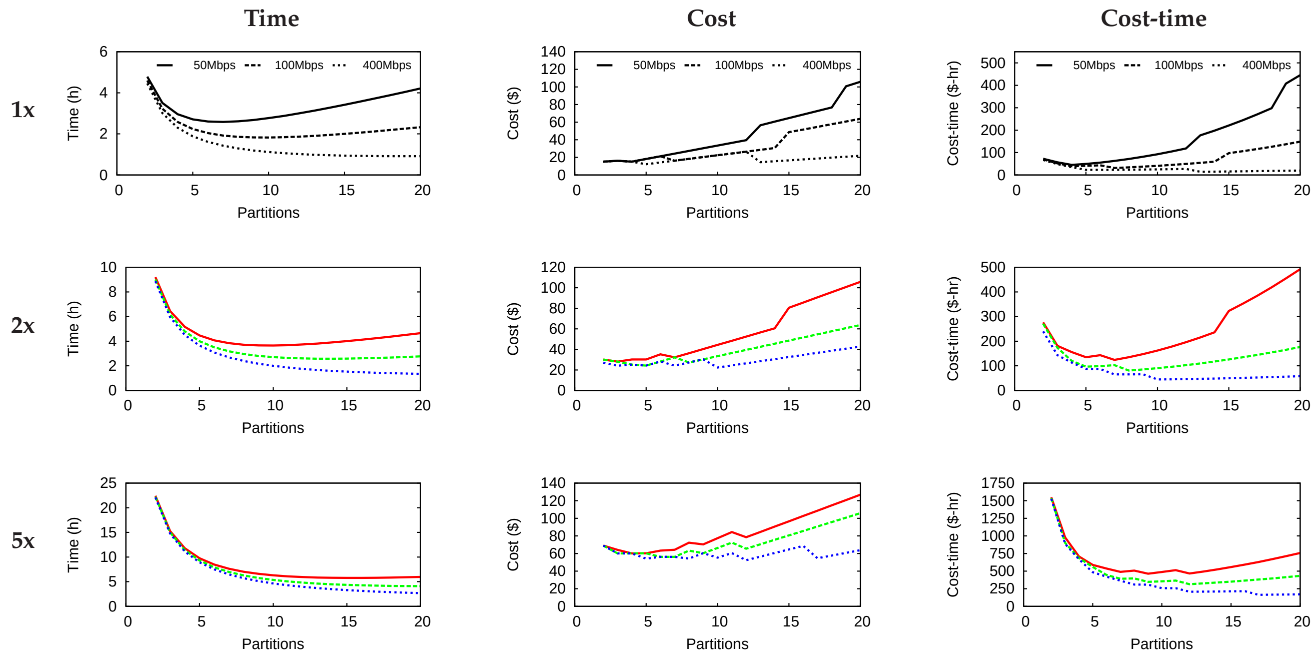


Fig. 8: Estimated operating time, operating costs, and cost-time product of E-MAKER for annotating 800 contiguous sequences of the Anopheles Agambiae genome for various characteristics of the operating environment.

instance with 12GB of memory. We find the observed times deviate from the estimations when the data size exceeds 12GB. However, we do not notice a similar impact on E-MAKER since the genome sequences loaded in memory for annotation are often less than 10MB. In this work, we limit the partitions in E-Sort to the size of the memory at the provisioned instances to minimize the unpredictable impact on the task execution overheads.

In summary, the determination of the number of partitions and instances to provision must be made in conjunction with measurements of the characteristics of the workload and operating environment.

5.4 Adaptations to the characteristics of the workload and operating environment

In this section, we show the application-level adaptations of the number of partitions and instances used in operation based on the initial assessment of the overheads of running the workload in the deployed environment. The adaptations measure the characteristics of the workload (such as task execution times) and operating environment (such as network bandwidth) by operating

2168-7161 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2415780, IEEE Transactions on Cloud Computing

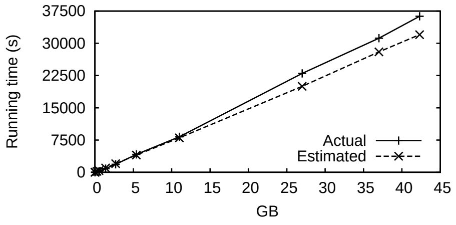


Fig. 9: Estimated and observed task execution times T_{task} for sorting various data sizes at an instance with 12 GB of RAM.

a sample partition on a sample allocation. The sample partition comprises about 1% of the defined workload in the applications. The sample allocation consists of a single instance in the same environment in which the application will be deployed. This sample allocation is further used in operating the remainder of the workload to prevent wastage as instances in cloud platforms incur charges to the nearest hour.

The measurements using the sample partition and allocation provide information on the network bandwidth and the overheads of task execution, partition, and merge. Based on these measurements, the application can use the model to determine the number of partitions and instances to provision.

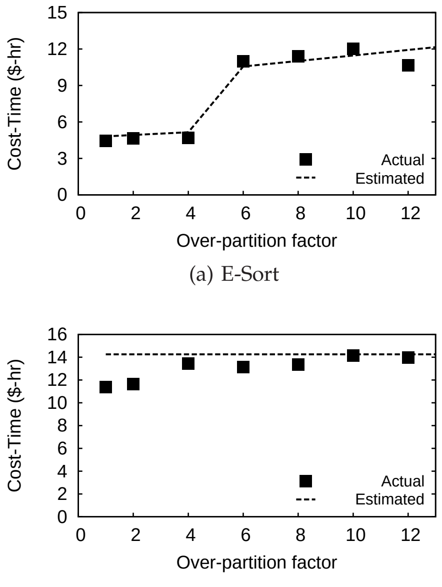
Figures 10 and 11 show the chosen partitions along with the actual cost-time product observed when running with those partitions. The overheads of operating the sample partitions are minimal compared to the overall time of operation. This can be observed in Figures 10 and 11 where the actual cost-time product closely tracks the cost-time product estimated by the model. In summary, these figures show that the applications achieve cost-efficiency by tuning their operation to the operating conditions in the deployed environment.

5.5 Over-partitioning of workload

Our analysis so far considers the number of partitions and instances provisioned for operation to be equivalent. The over-provisioning of instances relative to the number of partitions chosen for operation results in resource wastage and high costs. In this section, we consider the over-partitioning of the workload relative to the number of instances provisioned for operation. The over-partitioning is useful when smaller partitions or tasks are desired for faster detection and re-execution of failed tasks, limiting the consumption of resources by tasks, and quickly adapting the operation to varying operating conditions measured from the execution of tasks.

Figure 12 describes the effects of creating partitions greater than the number of instances determined for cost-efficient operation. The over-partition factor represents the multiplicative factor applied on the number of instances determined for cost-efficient operation.

In E-Sort, over-partitioning incurs higher partition and merge overheads without recording any increase in the gains from the increased concurrency. This is because the number of partitions that can be simultaneously executed is limited by the number of instances available for operation. As a result, the time and cost of operation of E-Sort increases with over-partitioning as seen in Figure 12a (at over-partition factor of 6, the time of operation increases to the next hourly boundary resulting in a sharp increase in the cost-time product).



(a) E-Sort

(b) E-MAKER

Fig. 12: Illustration of the effects of over-partitioning in E-Sort and E-MAKER. The actual values were observed with the same experimental setup and inputs as Figure 6.

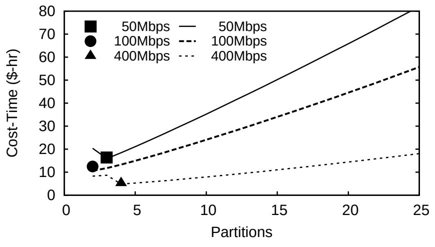
In E-MAKER, the effects of over-partitioning on the time and cost of operation are marginal. This is because the partition and merge overheads are negligible and the data transfer overheads only increase with the number of instances used for execution of the tasks. In the next section, we utilize over-partitioning in E-MAKER to enable the measurement of varying operating conditions at shorter intervals and the adaptation of the operating parameters to the measured conditions.

5.6 Adaptations to varying operating conditions

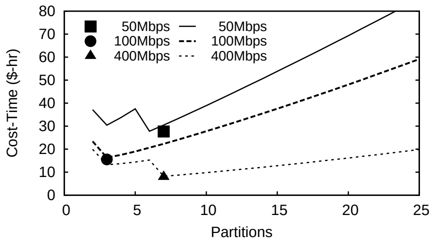
In this section, we demonstrate the dynamic adaptations when the characteristics of the operating environment that impact the overheads and performance of the applications vary during operation. We consider changes in the network bandwidth since it is prone to vary due to multi-tenant effects. We show the dynamic adaptations in E-MAKER where the impact from changes in the bandwidth are pronounced due to the large common data transfer overheads.

Figure 13 shows the dynamic adaptations by E-MAKER to the operating conditions observed during runtime. It plots the number of partitions and instances chosen for operation based on the observed bandwidth. The adaptations in E-MAKER function by progressively partitioning and allocating the instances for operation based on the observed conditions. It also over-partitions the workload by a factor of 2 to enable measurement

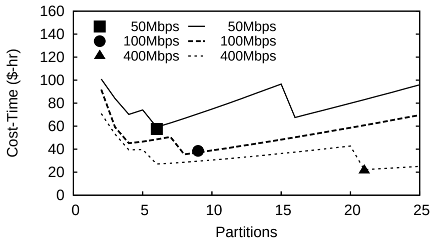
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2415780, IEEE Transactions on Cloud Computing



(a) T_{task}: 1x

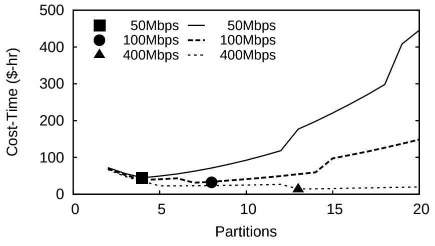


(b) T_{task}: 2x

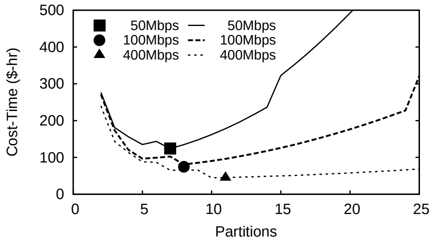


(c) T_{task}: 5x

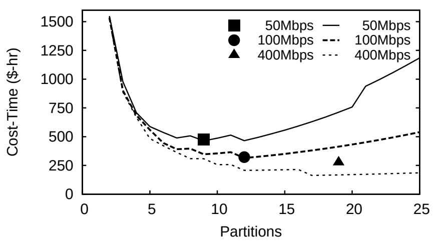
Fig. 10: The lines represent the estimated values for sorting 2 billion records totaling 11GB under different bandwidth. The individual points plot the partitions dynamically chosen during operation by measuring the operating environment.



(a) T_{task}: 1x



(b) T_{task}: 2x



(c) T_{task}: 5x

Fig. 11: The lines represent the estimated values for annotating 800 sequences of Anopheles Agambiae under different bandwidth. The individual points plot the partitions dynamically chosen during operation by measuring the operating environment.

and adaptation at shorter intervals without incurring additional overheads. In our setup, E-MAKER measures the operating environment after the dispatch of every task and recomputes the operating parameters.

We observe in Figure 13 that when the bandwidth drops after 300 seconds of operation, E-MAKER recomputes its operating parameters and lowers the number of partitions and instances it uses for operation. This minimizes the transfer overheads which become pronounced at low bandwidth. When the bandwidth increases again at 2100 seconds, E-MAKER determines that it can achieve cost-efficiency by continuing operation with the current scale of instances rather than increasing the instances in the deployment. In the experiment show in Figure 13, the overheads of progressive partitioning and re-computation of the operating parameters was less than 5% of the time of operation.

For comparison, Figure 13 also plots the operation of E-MAKER using the measurements from the operation of sample partitions in the same operating environment. Figure 13 shows a sharp increment in the operating cost of the sample partitioning approach at 3600 seconds when the time of operation exceeds the hourly boundary. At this point, the instances are provisioned for another hour of use and thus the operating costs increase. From the comparisons of the two techniques, we note the dynamic adaptations are better positioned to handle variations in the operating environment while incurring low overheads. However, the dynamic adaptations can incur large overheads and prove disadvantageous without a properly designed control system [29] in the presence of spurious and frequent variations.

6 Related Work

Previous efforts have built and studied solutions for the efficient deployment and operation of computational processes. We review these efforts and summarize the differences with the techniques presented in this work.

Techniques for service-oriented environments: Several efforts have studied techniques for the optimal operation and provisioning of resources for service-oriented and multi-tenant environments such as web applications [30], e-commerce systems [31], and databases [32]. The techniques for service-oriented environments include load prediction and estimation [33], [34], analysis of previous deployments and loads [31], and monitoring and adapting the provisioned resources according to the needs of the services [30], [35]. The techniques advocate the adaptation of the operating environment to the demands of the services.

In these environments, the workload is unknown, unpredictable, and determined by external factors such as the current demands of users. Further, these environments have different economies of operation since the operating costs are incurred in serving multiple users and guaranteeing the negotiated service level agreements. Our work solves a different problem, where the end-user presents a finite workload and must bear the cost of the chosen configuration.

Resource provisioning mechanisms: The provisioning and allocation of resources for large-scale and resource-intensive applications have been extensively studied in the context of distributed computing [36]–[38]. Recently, several efforts have considered the cost-efficient deployment of scientific and data-intensive workloads in cloud

2168-7161 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2415780, IEEE Transactions on Cloud Computing

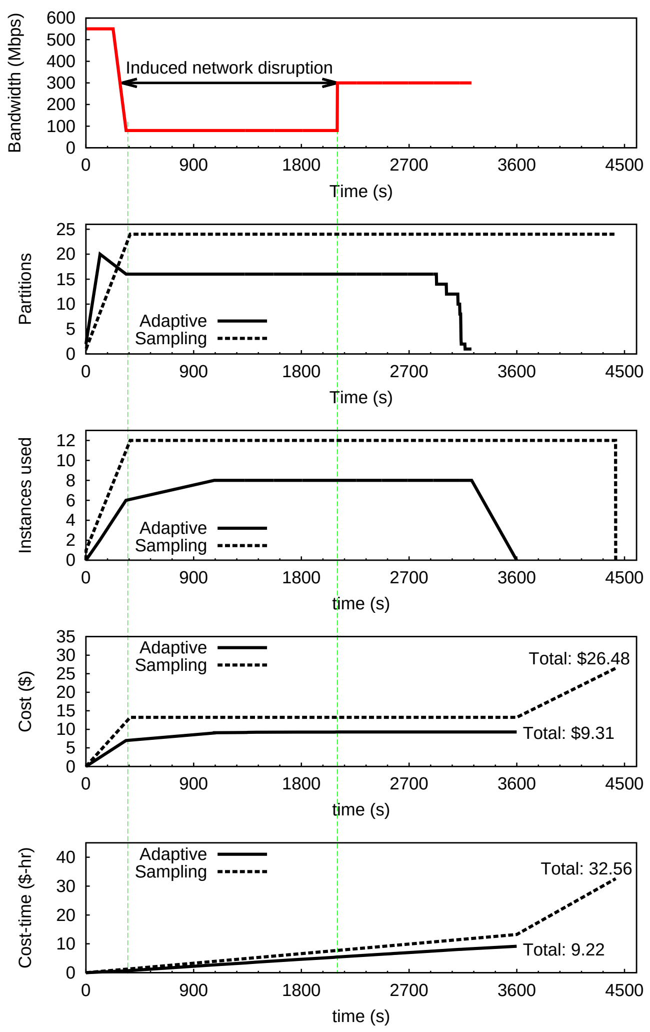


Fig. 13: Dynamic adaptations of the operating parameters in E-MAKER according to the observed network bandwidth. For comparison, the operation using the parameters chosen from the initial sampling of the environment is also plotted.

platforms [39]–[41]. The effort in [42] considers a class of elastic applications with flexibility in the quality of the results computed and presents solutions for their efficient resource allocation. The authors in [39] and [43] present heuristics for the optimal allocation of resources for an arbitrary batch of independent jobs. The efforts in [40], [41] present scheduling techniques in the middleware for multi-user environments running on cloud resources. These scheduling techniques argue for coordination between cloud providers and users to maximize the resource utilization of the provisioned resources.

We extend the work in this area by considering deployments where operators or users independently and directly provision resources from cloud platforms for operation. The provisioned resources are exclusively dedicated and maintained for the operation of an instance of the application. We present techniques for determining the scale of resources that achieve cost-efficient operation in these deployments.

Workload partitioning: The partitioning and decomposition of workloads have previously been studied in shared execution environments, such as grids and clusters [44], [45]. Recently, Agarwal et al. [46] considered a system with a large proportion of recurring jobs and utilized information from prior executions to determine the optimal degree of parallelism to enable during operation. The work in [47] describes the importance of identifying the optimal number of data partitions for MapReduce applications. It presents preliminary insights from an approach combining code and data analysis with optimization techniques. In our work, we argue that applications using the split-map-merge paradigms, such as MapReduce, can benefit by internally incorporating a model that can harness the code and data analysis to determine cost-efficient operation.

Finally, the efforts in [48], [49] are closely related by their use of a model describing the time and cost of operation of applications. The framework in [48] is targeted at deployments in a hybrid environment comprised of resources drawn from a local cluster and a commercial cloud. The framework schedules jobs on the local cluster and provisions resources in the cloud when the capacity at the cluster cannot satisfy the time constraints of the applications. The Conductor framework in [49] presents an abstraction for efficiently deploying MapReduce applications. It includes a model describing the costs, capabilities, and the computation and storage capacities of various instances offered in a cloud platform. The models are applied to select the services that achieve the lowest cost in running the MapReduce application. The applicability of the framework is restricted to the cloud services and instances that are modeled.

7 Conclusion and Future Work

The cost-efficient operation of concurrent applications in cloud platforms is determined by the number of partitions and compute instances chosen for operation. We showed the scale of partitions and instances that achieve cost-efficient operation varied depending on the characteristics of the workload and the environment in which they are deployed. Further, these operating parameters have to determined in diverse, unknown, and often unpredictable operating environments. In order to determine the number of partitions and instances for cost-efficient operation in the deployed environment, we argue that applications must be self-modeling and self-tuning. In this work, we considered the class of applications executed using the split-map-merge paradigm and presented application-level techniques for realizing self-modeling and self-tuning applications. We showed these applications achieved high cost-efficiency by determining the resources needed for execution and adapting their execution according to the measured characteristics of their deployed environments.

We regard the techniques in this paper as an illustration of the principles of building concurrent applications in the cloud. For these principles to be effective, we

2168-7161 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2415780, IEEE Transactions on Cloud Computing

believe the techniques must be tailored and tuned to the properties and runtime components of the class of applications (e.g., directed acyclic graphs) being considered. An useful extension of the presented techniques will consider the effects of the size of instances to determine if cost-efficiency can be achieved by switching to a different instance size or service when the operating conditions vary. Another direction would consider workloads that can be partitioned into tasks which utilize multiple cores for execution. Our current work proceeds in this direction by studying the trade-offs between the capabilities and costs of different instances sizes in cloud platforms for running multi-core applications.

Acknowledgements

This work was supported in part by National Science Foundation grants CNS-0643229, CNS-855047, and OCI-1148330.

References

[1] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” in *Symposium on Operating System Design and Implementation (OSDI)*, 2004, pp. 137–150.

[2] D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey, “Scatter/gather: A cluster-based approach to browsing large document collections,” in *Proceedings of the 15th International ACM SIGIR conference on Research and development in information retrieval*, 1992, pp. 318–329.

[3] J.-P. Goux, S. Kulkarni, J. Linderoth, and M. Yoder, “An enabling framework for master-worker applications on the computational grid,” in *High-Performance Distributed Computing, 2000. Proceedings. The Ninth International Symposium on*, 2000, pp. 43–50.

[4] W. Lu, J. Jackson, and R. Barga, “AzureBlast: a case study of developing science applications on the cloud,” in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. ACM, 2010, pp. 413–420.

[5] E. Roloff, M. Diener, A. Carissimi, and P. Navaux, “High performance computing in the cloud: Deployment, performance and cost efficiency,” in *Cloud Computing Technology and Science, 4th IEEE International Conference on*, 2012, pp. 371–378.

[6] C. Wang, K. Ren, J. Wang, and Q. Wang, “Harnessing the cloud for securely outsourcing large-scale systems of linear equations,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1172–1181, Jun. 2013.

[7] R. Barga, D. Gannon, and D. Reed, “The client and the cloud: Democratizing research computing,” *IEEE Internet Computing*, vol. 15, no. 1, pp. 72–75, Jan. 2011.

[8] W. Cirne, D. Paranhos, L. Costa, E. Santos-Neto, F. Brasileiro, J. Sauve, F. A. B. Silva, C. Barros, and C. Silveira, “Running bag-of-tasks applications on computational grids: the mygrid approach,” in *Parallel Processing, 2003. Proceedings. 2003 International Conference on*, Oct 2003, pp. 407–416.

[9] A. V. Gerbessiotis and L. G. Valiant, “Direct bulk-synchronous parallel algorithms,” *Journal of parallel and distributed computing*, vol. 22, no. 2, pp. 251–267, 1994.

[10] A. Luckow and et al., “Distributed replica-exchange simulations on production environments using saga and migol,” in *IEEE Fourth International Conference on eScience*, 2008, pp. 253–260.

[11] B. Abdul-Wahid, L. Yu, D. Rajan, H. Feng, E. Darve, D. Thain, and J. A. Izaguirre, “Folding Proteins at 500 ns/hour with Work Queue,” in *8th IEEE International Conference on eScience (eScience 2012)*, 2012.

[12] A. Thrasher, Z. Musgrave, D. Thain, and S. Emrich, “Shifting the Bioinformatics Computing Paradigm: A Case Study in Parallelizing Genome Annotation Using Maker and Work Queue,” in *IEEE International Conference on Computational Advances in Bio and Medical Sciences*, 2012.

[13] D. Rajan, A. Thrasher, B. Abdul-Wahid, J. A. Izaguirre, S. Emrich, and D. Thain, “Case Studies in Designing Elastic Applications,” in *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2013.

[14] D. Rajan, A. Canino, J. A. Izaguirre, and D. Thain, “Converting a High Performance Application to an Elastic Cloud Application,” in *The 3rd IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2011)*, 2011.

[15] Hadoop, http://hadoop.apache.org/, 2007.

[16] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, “Dryad: Distributed data parallel programs from sequential building blocks,” in *Proceedings of EuroSys*, March 2007.

[17] S. Jha, Y. E. Khamra, and J. Kim, “Developing Scientific Applications with Loosely-Coupled Sub-tasks,” in *Proceedings of the 9th International Conference on Computational Science: Part I*, ser. ICCS ’09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 641–650.

[18] P. Bui, D. Rajan, B. Abdul-Wahid, J. Izaguirre, and D. Thain, “Work Queue + Python: A Framework For Scalable Scientific Ensemble Applications,” in *Workshop on Python for High Performance and Scientific Computing (PyHPC) at the ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (Supercomputing)* , 2011.

[19] C. Moretti, A. Thrasher, L. Yu, M. Olson, S. Emrich, and D. Thain, “A Framework for Scalable Genome Assembly on Clusters, Clouds, and Grids,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 12, 2012.

[20] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The hadoop distributed file system,” in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE, 2010, pp. 1–10.

[21] M. A. Suleman, M. K. Qureshi, and Y. N. Patt, “Feedback-driven threading: Power-efficient and high-performance execution of multi-threaded workloads on cmps,” in *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2008, pp. 277–286.

[22] R. Rabenseifner, G. Hager, and G. Jost, “Hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes,” in *Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on*, Feb 2009, pp. 427–436.

[23] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, “Runtime measurements in the cloud: Observing, analyzing, and reducing variance,” *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 460–471, Sep. 2010.

[24] G. M. Amdahl, “Validity of the single processor approach to achieving large scale computing capabilities,” in *Proceedings of the Spring Joint Computer Conference*, ser. AFIPS ’67 (Spring). ACM, 1967, pp. 483–485.

[25] “How do I select the right instance type?” http://aws.amazon.com/ec2/faqs, accessed: 2014-07-21.

[26] C. Holt and M. Yandell, “MAKER2: an annotation pipeline and genome-database management tool for second-generation genome projects,” *BMC Bioinformatics*, no. 12, p. 491, 2011.

[27] “Windows Azure Cloud Platform,” http://www.windowsazure.com, accessed: 2013-12-21.

[28] “FutureGrid,” https://portal.futuregrid.org, accessed: 2013-12-21.

[29] G. F. Franklin, D. J. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*, 4th ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[30] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood, “Agile dynamic provisioning of multi-tier internet applications,” *ACM Trans. Auton. Adapt. Syst.*, vol. 3, no. 1, pp. 1–39, Mar. 2008.

[31] D. Villela, P. Pradhan, and D. Rubenstein, “Provisioning servers in the application tier for e-commerce systems,” *ACM Transactions on Internet Technology*, vol. 7, no. 1, Feb. 2007.

[32] S. Sakr and A. Liu, “Sla-based and consumer-centric dynamic provisioning for cloud databases,” in *Proceedings of the IEEE Fifth International Conference on Cloud Computing*, 2012, pp. 360–367.

[33] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, “Cloudscale: Elastic resource scaling for multi-tenant cloud systems,” in *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, ser. SOCC ’11. New York, NY, USA: ACM, 2011, pp. 5:1–5:14.

[34] Q. Zhang, L. Cherkasova, and E. Smirni, “A regression-based analytic model for dynamic resource provisioning of multi-tier applications,” in *Proceedings of the Fourth International Conference on Autonomic Computing*, 2007, pp. 27–.

[35] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem, “Adaptive control of virtualized resources in utility computing environments,” *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3, pp. 289–302, 2007.

2168-7161 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2415780, IEEE Transactions on Cloud Computing

[36] G. Juve and E. Deelman, "Resource Provisioning Options for Large-Scale Scientific Workflows," in *2008 IEEE Fourth International Conference on eScience*. IEEE, Dec. 2008, pp. 608–613.

[37] M. D. de Assunçao and R. Buyya, "Performance analysis of allocation policies for interGrid resource provisioning," *Inf. Softw. Technol.*, vol. 51, no. 1, pp. 42–55, Jan. 2009.

[38] T. Sandholm, J. A. Ortiz, J. Odeberg, and K. Lai, "Market-Based Resource Allocation using Price Prediction in a High Performance Computing Grid for Scientific Applications," in *2006 15th IEEE International Conference on High Performance Distributed Computing*. IEEE, 2006, pp. 132–143.

[39] S. Genaud and J. Gossa, "Cost-Wait Trade-Offs in Client-Side Resource Provisioning with Elastic Clouds," in *IEEE 4th International Conference on Cloud Computing*, Jul. 2011, pp. 1–8.

[40] C. Vecchiola, S. Pandey, and R. Buyya, "High-Performance Cloud Computing: A View of Scientific Applications," in *2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks*. IEEE, Dec. 2009, pp. 4–16.

[41] T. A. Henzinger, A. V. Singh, V. Singh, T. Wies, and D. Zufferey, "FlexPRICE: Flexible Provisioning of Resources in a Cloud Environment," in *2010 IEEE 3rd International Conference on Cloud Computing*. IEEE, Jul. 2010, pp. 83–90.

[42] Q. Zhu and G. Agrawal, "Resource Provisioning with Budget Constraints for Adaptive Applications in Cloud Environments," *IEEE Transactions on Services Computing*, 2012.

[43] F. Chang, J. Ren, and R. Viswanathan, "Optimal Resource Allocation in Clouds," in *IEEE 3rd International Conference on Cloud Computing*. IEEE, 2010, pp. 418–425.

[44] W. Chen and E. Deelman, "Integration of Workflow Partitioning and Resource Provisioning," in *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2012, pp. 764–768.

[45] M. K. Hedayat, W. Cai, S. J. Turner, and S. Shahand, "Distributed Execution of Workflow Using Parallel Partitioning," in *2009 IEEE International Symposium on Parallel and Distributed Processing with Applications*. IEEE, 2009, pp. 106–112.

[46] S. Agarwal, S. Kandula, N. Bruno, M.-C. Wu, I. Stoica, and J. Zhou, "Re-optimizing data-parallel computing," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, Berkeley, CA, USA, 2012, pp. 21–21.

[47] Q. Ke, V. Prabhakaran, Y. Xie, Y. Yu, J. Wu, and J. Yang, "Optimizing data partitioning for data-parallel computing," in *Proceedings of the 13th USENIX conference on Hot topics in operating systems*, ser. HotOS'13. Berkeley, CA, USA: USENIX Association, 2011, p. 13.

[48] T. Bicer, D. Chiu, and G. Agrawal, "Time and Cost Sensitive Data-Intensive Computing on Hybrid Clouds," in *Cluster, Cloud and Grid Computing, 12th IEEE/ACM International Symposium on*, May 2012, pp. 636–643.

[49] A. Wieder, P. Bhatotia, A. Post, and R. Rodrigues, "Orchestrating the deployment of computations in the cloud with conductor," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, 2012, p. 27.

Dinesh Rajan received his M.S in Computer Science and Engineering from the University of Notre Dame in 2008. He is currently a Ph.D student in Computer Science and Engineering at the University of Notre Dame. His research studies the design of concurrent applications deployed and operated on distributed systems.

Douglas Thain received the B.S. in Physics in 1997 from the University of Minnesota and the M.S. and Ph.D. in Computer Sciences in 1999 and 2004 from the University of Wisconsin. He is currently an Associate Professor of Computer Science and Engineering at the University of Notre Dame, where his research focuses on scientific applications of distributed computing systems.

2168-7161 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.