An Automated Tool Profiling Service for the Cloud

Ryan Chard*, Kyle Chard†, Bryan Ng*, Kris Bubendorfer*, Alex Rodriguez†, Ravi Madduri† and Ian Foster†
*School of Engineering and Computer Science, Victoria University of Wellington
†Computation Institute, University of Chicago & Argonne National Laboratory

***Abstract*—Cloud providers offer a diverse set of instance types with varying resource capacities, designed to meet the needs of a broad range of user requirements. While this flexibility is a major benefit of the cloud computing model, it also creates challenges when selecting the most suitable instance type for a given application. Sub-optimal instance selection can result in poor performance and/or increased cost, with significant impacts when applications are executed repeatedly. Yet selecting an optimal instance type is challenging, as each instance type can be configured differently; application performance is dependent on input data and configuration; and instance types and applications are frequently updated. We present a service that supports automatic profiling of application performance on different instance types to create rich application profiles that can be used for comparison, provisioning, and scheduling. This service can dynamically provision cloud instances, automatically deploy and contextualize applications, transfer input datasets, monitor execution performance, and create a composite profile with fine grained resource usage information. We use real usage data from four production genomics gateways and estimate the use of profiles in autonomic provisioning systems can decrease execution time by up to 15.7% and cost by up to 86.6%.**

I. Introduction

Cloud computing has transformed how computing capabilities are acquired and used. Cutting-edge and large-scale computing capabilities previously accessible only to those with large computing centers are now available at the click of a button. Furthermore, these capabilities can be configured to meet the precise requirements of a particular application or workload. Commercial cloud providers, such as Amazon Web Services (AWS) Elastic Compute Cloud (EC2), offer a range of instance types designed to meet specific user needs. EC2 alone offers 38 instance types (as of November 2015), each of which can be customized with varying disk, memory, and CPU capabilities. Users may also provision optimized storage, network connections, and specialized hardware (e.g., cluster or GPU instances).

This near limitless choice offers unparalleled flexibility but also introduces new challenges. The same application may perform quite differently on different cloud configurations. Different configurations can have quite different costs. Thus, effective use of cloud infrastructure requires that the user understand how an application performs, such that the most suitable instance type can be used and configured. Good choices become more important when large amounts of computing are to be performed, as is the case with so-called science gateways, web portals that offer access to a set of predefined tools. (We use the term "tool" here to denote any application or script.) For example, the Globus Galaxies platform [1], which uses autonomic cloud provisioning to dynamically execute various tools on behalf of users, consumed more than half a million EC2 instance hours in 2015, for a total cost of more than $150,000. As we will show below, the performance and cost of a single tool can vary by an order of magnitude or more on different instance types, making it easy to make expensive mistakes when choosing cloud configurations.

It is such concerns that motivate the work described here, which focuses on the design, implementation, and evaluation of a novel cloud profiling service. The purpose of this service is to automate the creation of tool profiles, concise descriptions of the performance and CPU, memory, network, and disk requirements of a supplied tool under different environments and scenarios. These profiles can then be used to optimize instance selection, enable tool comparison, and improve provisioning and scheduling algorithms. The profiling service is self-contained and can be deployed externally, and configured to automatically profile any supplied tools on different cloud instances and with different configurations. The service can automatically provision test instances, deploy and configure tools on provisioned instances, stage input datasets, monitor tool execution, and record fine-grained tool profiles.

To demonstrate the value of our profiling approach we have analyzed five genomics tools commonly used within Globus Genomics, a popular gateway built on the Globus Galaxies platform. For these tools, we compute profiles for representative input data and settings (derived from production usage) and evaluate the extent to which profiles can be used to improve performance and decrease cost for Globus Genomics users. We find that selecting instance types based on tool profiles can reduce execution time by up to 15.7% and cloud computing costs by up to 86.6% relative to simpler heuristics that do not consider profiles.

The rest of this paper is organized as follows. We present related work in Section II and the Globus Galaxies platform in Section III. In Section IV, we describe the profiling service and in Section V discuss the results of employing the service to profile a set of tools. In Section VI we investigate how tool profiles can be used by improve provisioning decisions. Finally, we conclude and present future work.

II. Related Work

Numerous studies have explored the viability of performing scientific analyses on commercial clouds [2], [3], compared cloud and HPC infrastructure [4], and investigated the practicality of hosting scientific gateways on clouds [5], [6]. These studies focus on using a specific tool or benchmark suite to

evaluate the performance of specific cloud infrastructure. In contrast, we provide a general-purpose service for evaluating a range of tools on a variety of cloud instance types.

Most profiling work focuses on the ability to predict job execution and completion times. These predictions are important for scheduling algorithms such as shortest job first and backfilling. Statistical analysis is one common basis for prediction and has been shown to significantly improve job run time estimates based on large supercomputing logs [7]. Approaches such as ARIA (Automatic Resource Inference and Allocation) [8], leverage mathematical models to estimate job completion times. Most often these approaches are based on historical analysis of execution logs, however small scale benchmarks in test environments can also be used to produce run time estimates [9]. In general, most profiling work focuses on prediction in homogeneous environments, however it has been shown that predictions can be inferred across platforms based on analysis of tools executed on each platform [10].

The HPC community has long recognized the importance of understanding and forecasting resource requirements for efficiently managing and utilizing resources [11]. One approach for gathering this information is via microbenchmarks [12]. One limitation of this approach is that microbenchmarks do not monitor a tool for the duration of its execution, it is therefore difficult to accurately capture the behavior of tools with varying degrees of resource utilization. Canonical correlation analysis provides an alternative method for predicting execution performance based on identifying the relationship between resource utilization and performance [11]. Performance information can also be derived by analyzing the structured patterns in application performance logs [13] or by using machine learning algorithms to classify applications [14]. Virtualized environments offer the ability to gather fine grained resource consumption at the visualization layer. Virtualization-level profiling can also be used to support real-time profile and resource adaption based on allocated resources [15]. These efforts focus primarily on offline profiling, in which profiles are established before execution. Similar techniques have also been applied in online profiling as a way of adapting resource usage of long running workflows [16] and responding to event-based monitoring of cloud services [17].

While our work leverages some of this related research, our approach is differentiated by its goal of providing a general-purpose service for creating tool profiles over a range of scenarios. This goal leads us to focus on the use of offline profiling to construct rich profiles that other services can then use to guide provisioning decisions. This work also neatly dovetails into our prior results on cloud provisioning, in which we used cloud tomography to characterize and monitor the opaque AWS network [18] and extended earlier work [19] to construct fine grained real-time health monitoring for AWS instances; and developed a cost-aware provisioner that reduces the cost of executing scientific workflows [20]. That prior work provided two major facets needed for cost and time effective cloud provisioning in the cloud; this paper adds the final component, tool profiling.

III. Globus Galaxies platform

The Globus Galaxies platform provides a mechanism for creating SaaS-based scientific gateways. Each gateway is hosted on the cloud and uses on-demand and elastic, compute resources to dynamically execute jobs as they are submitted by users. The gateways leverage the Galaxy workflow engine [21], enabling the creation, execution, and management of parallel workflows composed of various tools; Globus [22] to provide reliable and secure big data transfers between users and the gateways; Globus Nexus [23] for identity management, authentication, and access control; Swift [24] for parallelizing individual workflow components; HTCondor [25] for job management; and a specialized cost-aware cloud provisioner [20], [26] for autonomically provisioning cloud infrastructure. The Globus Galaxies platform has been used to create gateways for genomics [1], climate and economic policy [27], medical imaging [28], and cosmology [29].

TABLE I: Globus Genomics Tool Execution Statistics

Tool	Times Run	Exec Freq Rank	Exec %	Avg Exec Time
BWA MEM	1473	4	5.9%	00:38:22
FastQC	1471	5	5.9%	00:29:13
MarkDups	1341	7	5.3%	00:35:43
Bowtie	85	39	0.3%	01:23:57
BWA ALN	83	40	0.3%	00:20:40

A. Globus Genomics

Globus Genomics, the largest instantiation of the Globus Galaxies platform, provides state-of-the-art genomics analysis capabilities. It currently encompasses 20 separate gateways that collectively serve over 300 researchers working on various cancers, neurodegenerative disorders, and other disorders. While these gateways provide similar capabilities, and in most cases, the same tools, usage can vary significantly between gateways depending on researcher behavior. TABLE I summarizes characteristics of five tools used by Globus Genomics researchers. These characteristics are calculated from analysis of production logs of four Globus Genomics gateways over a four month period. These measurements include the number of times the tool has been used, the ranking when compared with other Globus Genomics tools, the percentage of total execution time spent on this tool, and the average execution time. This summary highlights the different characteristics of these tools ranging in execution time from 20 minutes to more than an hour, and being executed rarely and frequently.

B. Autonomic Provisioning

The Globus Galaxies platform uses an autonomic cost-aware provisioner [20] to provision cloud infrastructure on-demand. The provisioner monitors the gateway's job execution queue (HTCondor) and based upon assessment of real-time cloud markets (i.e., on-demand and spot prices) provisions the cheapest instance at that point in time.

The provisioner is typically pre-configured with a set of suitable instance types that satisfy the needs of every tool

available in the gateway. It then selects the cheapest instance type at the time of execution. We have recently extended the provisioner to use pre-defined (manually created) tool profiles for several commonly used tools. These profiles essentially map a particular tool to its most suitable instance type. The provisioner uses these profiles to increase the search scope when provisioning instances, by selecting an instance type that is at least as powerful as that required by the profile, while still selecting the cheapest instance type. We showed that even rudimentary profiles can significantly decrease total cost and execution time. However, while these profiles have proven useful, they require intimate and expert knowledge of a given tool and trial-and-error analysis of instance types. Those limitations spurred us to develop the method that we describe here for automatically and systematically deriving rich profiles without calling on human expertise.

IV. Profiling Service

The information included in a tool profile can range from a binary measure of whether a tool can execute on a specific instance through to fine-grained representation of resource usage over time. We focus on the latter as it can enable prediction of job execution time (and cost) on different instance types and support resource aware multi-tenanting of tools.

However, manually constructing a profile for a new tool is time-consuming due to the numerous permutations that must be considered, as well as the frequency with which tools and instance types change. For example, tool performance is dependent on instance type, specific settings of that instance (e.g., optimized storage and network), input datasets, and invocation parameters. Having identified the range of configurations to be profiled, users must then painstakingly embark on a trial-and-error analysis of instance types by deploying suitable infrastructure, configuring and installing the tool and its dependencies, and measuring tool performance and resource usage.

To address this, our profiling service is designed to enable the automated execution and monitoring of arbitrary tools over any instance type. A tool can be provided either as a self-contained directory with the application executable and a collection of input data files, or as a contextualization script that can be used to install the tool and its dependencies on demand. The user specifies the instance types and configurations to be evaluated and the profiling service then executes the application on all specified combinations and collects performance metrics periodically. We leverage Performance Co-Pilot (PCP) to collect performance metrics as it is lightweight and has little overhead on application execution.

A. Architecture

The profiling service is implemented as a web application. As shown in Fig. 1, it includes the profiling web service, a reliable database (hosted on AWS RDS), and a dynamic pool of provisioned worker nodes, each with a management web service for control and monitoring tool execution.

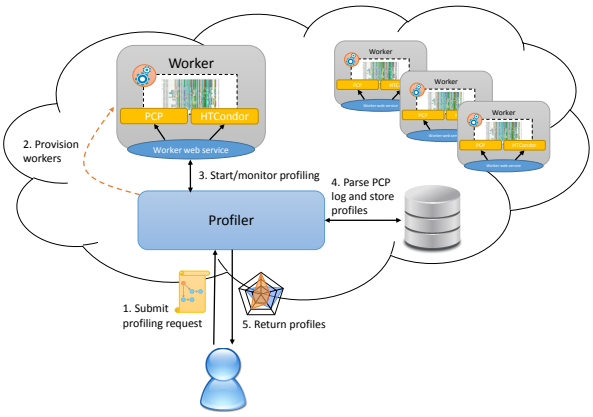


Fig. 1: Profiling service architecture

The profiling service exposes a RESTful web service interface for requesting and managing profiles. The service can be deployed on any cloud instance configured with access to the profile database (local or RDS) and a mounted NFS shared file system. The profiling service uses the shared file system for staging data and tools to worker nodes.

The profiling service is implemented as a multi-threaded Python application. When a profiling job is requested the service creates a new thread that is responsible for overseeing the execution and monitoring of the tool. The thread will provision EC2 instances, stage/deploy the workload and tool, monitor execution, parse monitoring logs, and store the profile in the database.

Each worker node is deployed with a dynamic web service that allows the provisioning service to control and monitor the executed tool. It is also configured with a local HTCondor client to manage the execution of the tool locally. We use Cloudinit to dynamically contextualize the worker instance after it is provisioned. This contextualization process installs and configures the worker web service, HTCondor, and PCP, as well as mounting the shared file system.

B. Profiling Process

Fig. 2 shows the profiling process. A user requests a new profile by submitting a JSON profile request (Listing 1) specifying the tool, instance types, and instance and tool configurations. We provide a light-weight client to simplify interactions with the profiling service.

Listing 1: A partial JSON profile request.

```
{
 name: 'my profile',
 executable: 'my_app.sh',
 workloads: {
  workload_1: ['file1', 'file2'],
 }, configurations: {
  config_1: '--config 1 --config 2',
  config_2: '--config 1 --config 3',
 }, instance_types : [{
   type: 'r3.xlarge',
   override: '$threads=8',
   acquisition: 'spot',
```

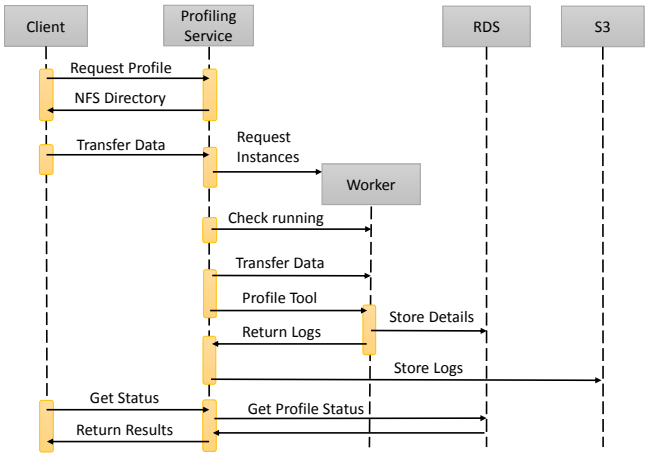


Fig. 2: A sequence diagram showing the steps involved when the profiling service profiles a tool.

```
    bid: 6
  },{
    type: 'c3.8xlarge',
          override: '$threads=32'
  }]
}
```

In response to a profile request the profiling service creates a unique working directory for each profiling job and returns this path to the client. The client can then optionally upload the tool and any input files used for execution. Alternatively, the client may choose to upload a contextualization script for deploying the tool automatically. A profiling service thread is started for each instance type and configuration to manage provisioning and tool execution. The job request may optionally specify other properties such as target availability zones and cloud acquisition model: either on-demand or spot pricing. In the case of spot pricing, the request may also optionally specify a bid price. If no bid price is set the maximum bid is used.

Once a cloud instance is created, cloudinit is used to contextualize the worker. This process includes the installation and configuration of the worker web service and the mounting of the shared file system. Once the web service becomes responsive, the profiler thread initiates a request to the worker's web service indicating the executable to be profiled and the location of the workload. The worker service then copies the input data to its local drive from the shared file system. This is done to eliminate overhead incurred from accessing the shared filesystem during profiling. The worker service then processes the executable and appends a command to isolate execution within the local working directory. In addition, a string template is used to swap any overriding configurable parameters based on the instance type. For example, a client may specify the number of threads to use on each instance type to make optimal use of available cores. Immediately prior to executing the workload, a PCP logger is initiated to collect performance statistics on the host machine at one second intervals. The workload is then submitted to HTCondor to be executed on a local slot. During execution, PCP appends resource usage statistics to a pre-defined log file.

The profiling service probes each worker periodically to determine the status of the job being profiled, and records these progress markers in the database. Once the workload completes the PCP logger is halted and the exit code and execution time are updated in the database. The PCP logs are then transferred to the shared filesystem and the profiler thread parses them into CSV format. The resulting CSV file is uploaded to an S3 bucket for long-term storage. Throughout profiling the client can request status updates. Once the profiling job is complete the client can download the resulting profiles. Listing 2 shows an example profile, which includes high level summary information for each instance type as well as the location of the stored CSV logs for each configuration.

Listing 2: A partial JSON tool profile.

```
{
 name: 'my profile',
 executable: 'my_app.sh',
 log_files: 's3/output_bucket',
 results: [  {
    workload: ['file1', 'file2'],
    configuration: '--config 1 --config 2',
    instance_type: {
      type: 'c3.8xlarge',
      override: '$threads=32',
      exit_status: 'Completed'
      execution_time: 839912
      performance: {
        memory: { ... },
        cpu: { ... },
        disk: { ... },
        network: { ... }
      }
    }
  }, ... ]
}
```

V. Creating Genomics Tool Profiles

We use the following five genomics analysis tools to evaluate the capabilities of our profiling service. These tools are among the most frequently used tools on Globus Genomics, accounting for 17.7% of compute time consumed on four production gateways (summary statistics are included in TABLE I). They also have varied resource requirements, ranging in execution time from ten minutes to more than an hour.

1) **FastQC** [30] provides quality control on raw sequence data.
2) **PICARD MarkDuplicates (MarkDups)** [31] flags or removes duplicate reads in a SAM or BAM file.
3) **BWA ALN** [32] aligns short sequences to a reference sequence (e.g., the human genome).
4) **Bowtie** [33] is a short read aligner designed for high performance and memory efficiency.
5) **BWA MEM** [34] maps low-divergent sequences against large reference genomes. It chooses automatically between local and end-to-end alignments, supports paired-end reads, and performs chimeric alignment.

TABLE II: AWS Instance Types

Type	Family	vCPU	Memory (GB)	Storage (GB)	Network
c3.2xlarge	Compute optimized	8	15	2 x 80	High
c3.4xlarge	Compute optimized	16	30	2 x 160	High
c3.8xlarge	Compute optimized	32	60	2 x 320	High
g2.2xlarge	GPU instances	8	15	1 x 60	High
g2.8xlarge	GPU instances	32	60	2 x 120	10 Gigabit
r3.xlarge	Memory optimized	4	30.5	1 x 80	Moderate
r3.2xlarge	Memory optimized	8	61	1 x 160	High
r3.4xlarge	Memory optimized	16	122	1 x 320	High
r3.8xlarge	Memory optimized	32	244	2 x 320	10 Gigabit
m3.2xlarge	General purpose	8	30	2 x 80	High

TABLE III: Genomics Tool Profiles Over R3 Instances. Note: * indicates tool failure.

Instance	Tool	Time (H)	Memory (MB)	Memory %	Idle CPU %	User CPU %	Sys CPU %	Disk-R (MB)	Disk-W (MB)
r3.xlarge	Bowtie	2:23	6920.54	22.56	0.35	86.62	12.97	7.46	9526.87
r3.2xlarge	Bowtie	0:52	6011.29	9.78	0.08	76.52	23.37	0.35	10334.18
r3.4xlarge	Bowtie	0:29	6120.88	4.98	0.33	65.94	33.59	0.12	7673.24
r3.8xlarge	Bowtie	0:22	6474.33	2.63	2.51	49.73	47.48	0.42	7123.70
*r3.xlarge	BWA MEM	2:37	24314.41	79.25	0.65	98.53	0.39	11812.97	2545.70
r3.2xlarge	BWA MEM	1:22	10100.33	16.44	4.18	94.79	0.57	3700.75	8785.78
r3.4xlarge	BWA MEM	1:19	37935.04	30.85	13.77	85.30	0.49	0.35	8482.01
r3.8xlarge	BWA MEM	0:43	41279.97	16.78	47.10	49.02	2.92	0.28	9030.93
r3.xlarge	FastQC	0:15	54.23	0.18	75.05	24.17	0.53	29026.68	15.97
r3.2xlarge	FastQC	0:15	31.07	0.05	87.37	12.33	0.13	5.45	583.09
r3.4xlarge	FastQC	0:15	233.66	0.19	93.69	6.18	0.08	0.02	962.15
r3.8xlarge	FastQC	0:14	235.66	0.10	96.84	3.10	0.04	0.02	27.75
*r3.xlarge	BWA ALN	5:24	2311.53	7.53	2.76	96.98	0.20	29.04	9009.79
r3.2xlarge	BWA ALN	2:43	32760.49	53.32	22.59	76.57	0.60	3.99	31631.29
r3.4xlarge	BWA ALN	1:40	36748.67	29.89	37.68	61.68	0.52	1.18	32273.80
r3.8xlarge	BWA ALN	1:15	34278.09	13.93	53.47	45.80	0.64	1.18	31886.82
r3.xlarge	MarkDups	0:24	11512.12	37.52	70.85	27.18	0.50	4927.71	6928.91
r3.2xlarge	MarkDups	0:23	41639.11	67.77	83.46	15.93	0.24	0.22	6681.06
r3.4xlarge	MarkDups	0:22	69226.44	56.30	90.80	8.95	0.15	0.45	6989.04
r3.8xlarge	MarkDups	0:24	52198.18	21.22	93.12	6.69	0.11	0.45	7215.32

A. AWS Testbed

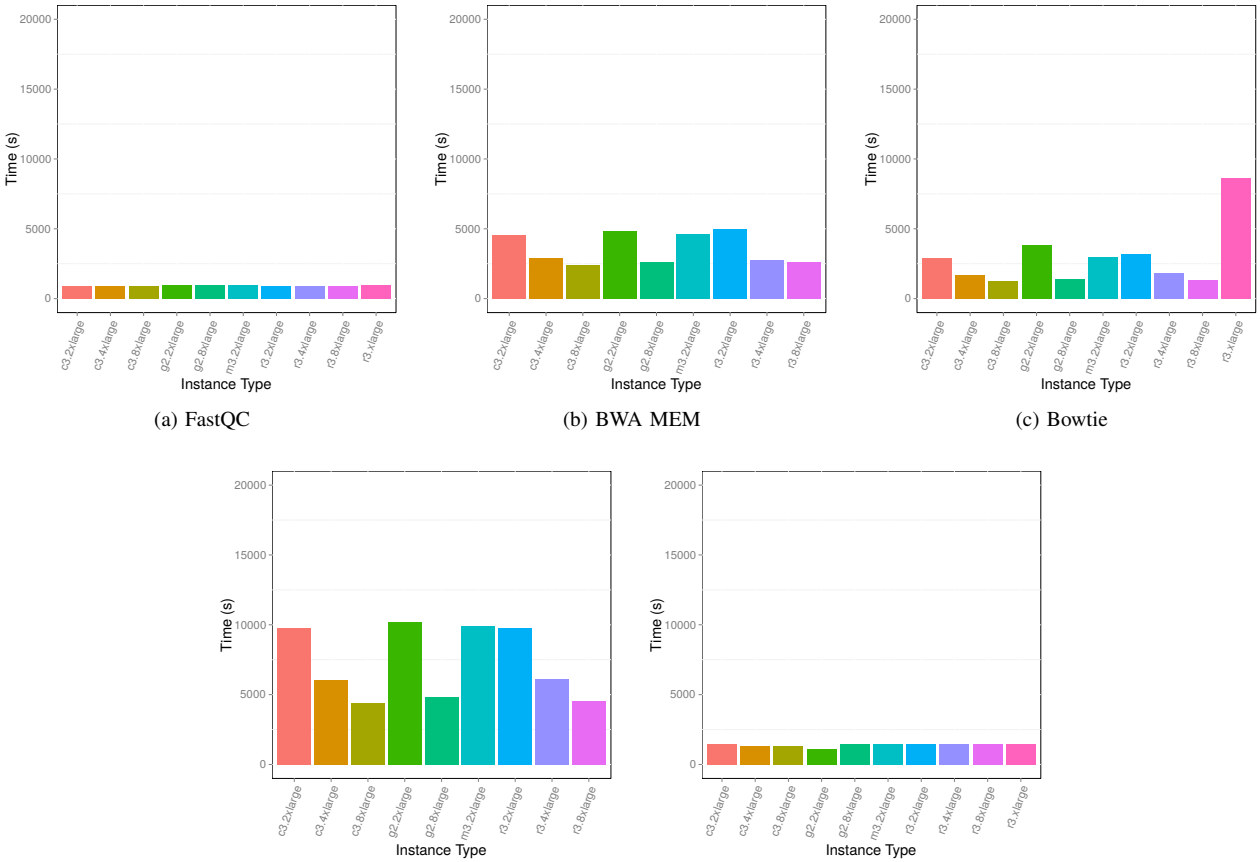
We selected 10 different AWS instance types on which to create tool profiles. For consistency, we selected instance types that support *instance storage*. That is, we do not consider instance types that use EBS storage, as this would likely have a significant effect on application performance. We leave analysis of advanced instance configurations to future work. We investigate performance on the M3 (general purpose), G2 (GPU enabled), C3 (compute optimized) and R3 (memory optimized) instance families. All profiling is performed in the us-east-1 AWS region, using spot instances in the cheapest availability zone at the time of profiling. The selected instance types are described in TABLE II.

B. Profiles

TABLE III summarizes profile data for the five genomics tools on four R3 instance types. These instance types range in capacity from r3.xlarge, with 4 vCPUs and 30.5GB of memory, through to r3.8xlarge, which has 32 vCPUs and 244GB of memory. Where possible, each profiled tool is dynamically configured to use all available vCPUs and memory during execution. TABLE III highlights the richness of the profile information collected. It also shows fairly consistent usage across instance types, indicating that execution time and resource requirements could be forecast for other instance types. In future work, we will analyze a wider set of instance types and tools with the aim of determining a representative set of instance types that reduce the profiling search space.

Our results show that tool performance is not entirely dependent on instance type. For example, *Bowtie* consumes roughly the same amount of memory regardless of the amount of available memory, number of threads, and vCPUs. *FastQC* executes for roughly the same amount of time on all instances. This is because it uses only a single vCPU rather than all available vCPUs. *BWA MEM* and *BWA ALN* both fail when using the r3.xlarge instance type as there is insufficient memory for execution. These findings highlight the potential improvements when provisioning and scheduling execution with complete knowledge of tool performance. For example, provisioning small instances or concurrently executing several tools on a single large instance can result in cheaper execution and improved performance.

Disk and network usage are minimal for each of the five tools profiled. Network usage is low as our profiling service attempts to minimize network overhead by staging data to the local file system. In addition, the tools themselves are executed on a single instance and require no network communication. Disk usage is (near) identical between instance types because the tools are executed with the same input datasets and settings, thus, the resulting data written to disk is also the same. The exception is small instance types which show increased disk usage as the disk is used for swap space when



(a) FastQC (b) BWA MEM (c) Bowtie

(d) BWA ALN (e) MarkDups

Fig. 3: The execution time of the tools over various instance types.

memory usage exceeds capacity.

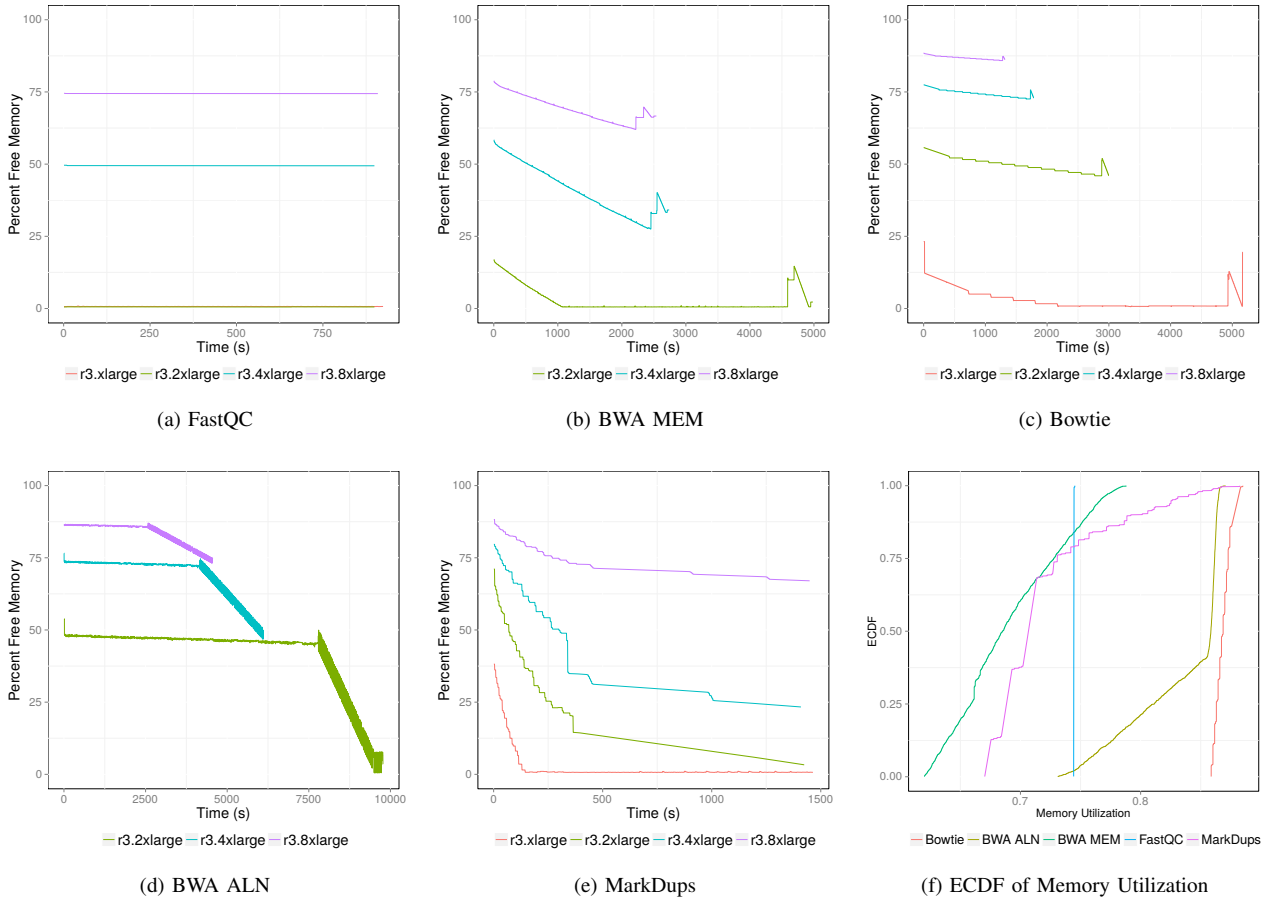
C. Execution Performance

We show in Fig. 3 the execution time for each tool on each instance type. As expected, the tools perform differently on different instance types. For example, *Bowtie* (Fig. 3c) and *BWA ALN* (Fig. 3d) exhibit a strong correlation between the number of vCPUs and execution time. In contrast, *FastQC* (Fig. 3a), which uses only a single vCPU, performs similarly regardless of instance type. In all cases, the tools execute fastest on c3.4xlarge, c3.8xlarge, g2.8xlarge, r3.4xlarge, and r3.8xlarge. These instances have at least 16 cores and 60 GB of memory each, with the exception of c3.4xlarge with only 30 GB of memory. In fact, the largest difference in execution time across these five instance types is 31%, with an average difference across all tools of 19.48% (*Bowtie*: 31%, *BWA ALN*: 28.4%, *BWA MEM*: 21.7%, *FastQC*: 8.8%, *MarkDups*: 7.5%), indicating that these instance types can be used somewhat interchangeably for these tools. Conversely, the largest executions of the *BWA ALN* and *BWA MEM* tools fail on the smallest r3.xlarge instance type due to memory limitations. It is important to note that tool performance is heavily dependent on the input dataset used.

D. Resource Usage

The profiling service enables capture of fine-grained execution characteristics. By default, we have configured the profiling service to monitor resource usage every second. We use these data to investigate fine-grained resource usage profiles for the R3 family of instances.

Figs. 4a–4e show temporal memory profiles and Fig. 4f the empirical cumulative distribution function (ECDF) of free memory for each tool on r3.8xlarge instances. The memory profile gives the amount of free memory, as a percentage of total instance memory, for each second of execution. Fig. 4b shows that *BWA MEM* has two distinct phases of execution: first, memory usage gradually decreases over the first phase of execution, before it is re-acquired towards the end of execution for a secondary phase. This is because the *BWA MEM* tool loads the entire input dataset into memory and pipes output to a sorting phase (which reduces memory usage). Once the initial phase ends, memory is released and the sorted output is written to disk. *BWA ALN*'s memory use (Fig. 4d) also presents an interesting pattern. This tool's memory profile is governed by two phases, with a substantial change in memory usage approximately half way through the execution. All of the tools use only a fraction of the available memory



(a) FastQC (b) BWA MEM (c) Bowtie

(d) BWA ALN (e) MarkDups (f) ECDF of Memory Utilization

Fig. 4: The percentage of free memory for each instance type during tool execution (a) - (e). Empirical cumulative distribution function of free memory for each workload over the r3.8xlarge instance type (f).

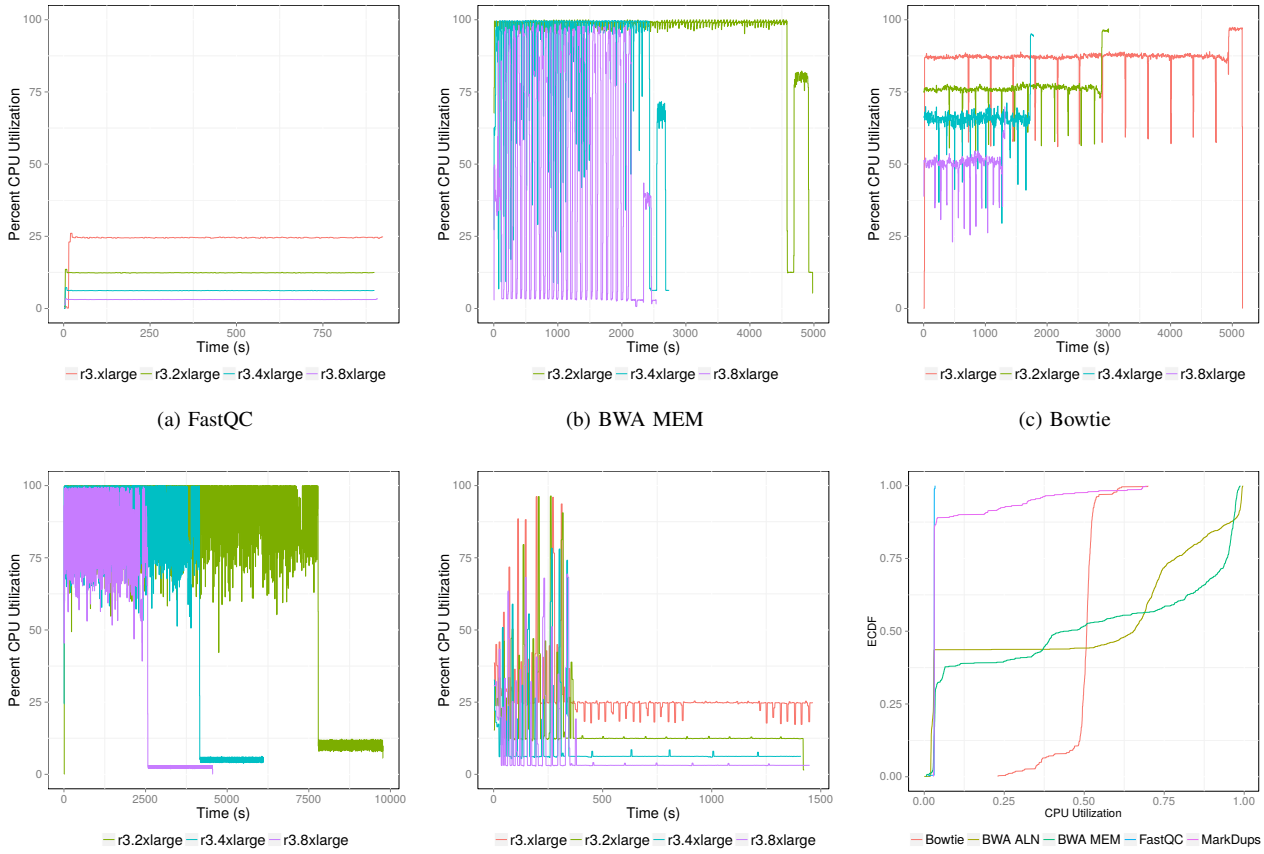
when executed on large instance types, indicating that, from a memory perspective, each tool could be executed concurrently on these instance types.

The ECDFs in Fig. 4f show the proportion of time spent with various amounts of free memory throughout execution. Each of the five profiled tools exhibits distinct and identifiable ECDF curves. For example, the *Bowtie* tool exhibits bimodality which manifests as memory utilization occurring with the greatest probability in two distinct values while *FastQC* has constant memory utilization and *MarkDups* exhibits high kurtosis (a measure of whether data is peaked or flat relative to a normal distribution).

Figs. 5a–5e plot the temporal CPU profile while Fig. 5f shows the ECDF of CPU usage for each tool on r3.8xlarge instances. *FastQC* CPU usage appears as flat lines in Fig. 5a and a vertical line in the ECDF Fig. 5f with very little CPU utilization because only a single core (out of thirty two available cores) is used. *BWA MEM* (Fig. 5b) and *BWA ALN* (Fig. 5d) consume most of the instance's CPU resources for approximately half of their execution time. Both tools exhibit symmetrical ECDF curves with low kurtosis for CPU utilization, suggesting that the computational intensity of these tools (Fig. 5f) is due mostly to many moderate fluctuations. *MarkDups* (Fig. 5e) exhibits volatile, CPU usage for the first third of its execution, from that point, CPU usage is constant and below 25%. Finally, *Bowtie* (Fig. 5c) uses a constant amount of CPU throughout execution. Interestingly, this tool does not use all available CPU resources: it uses only 50% on the 32 vCPU r3.8xlarge but 80% on the two vCPUs. This result implies that CPU usage is constrained by another factor.

E. Discussion

The profiles identify the practical implications of selecting appropriate instance types for tools. While any one tool may execute on a range of different instance types, its resource consumption may vary across different types, as may its response to resource limits. For example, *BWA ALN* and *BWA MEM* perform significantly worse on smaller instances and fail when using the r3.xlarge instance type. Thus, from the above results, *BWA ALN* should be executed on instances with at least 60 GB of memory. The tool also executes at least 38% faster on 16 core machines than on 8 core machines. Conversely, tools such as *FastQC* cannot take advantage of enhanced capabilities (e.g., more than one vCPU) and in fact



(a) FastQC (b) BWA MEM (c) Bowtie

(d) BWA ALN (e) MarkDups (f) ECDF of CPU Utilization

Fig. 5: The percentage of CPU utilization for each instance type during tool execution (a) - (e). Empirical cumulative distribution function of CPU utilization for each workload over the r3.8xlarge instance type (f).

should be deployed on the smallest available instance type, with its performance varying by only 10% across all instance types. Similarly *Bowtie* uses a maximum of 8GB of memory and therefore does not need instance types with high degrees of memory.

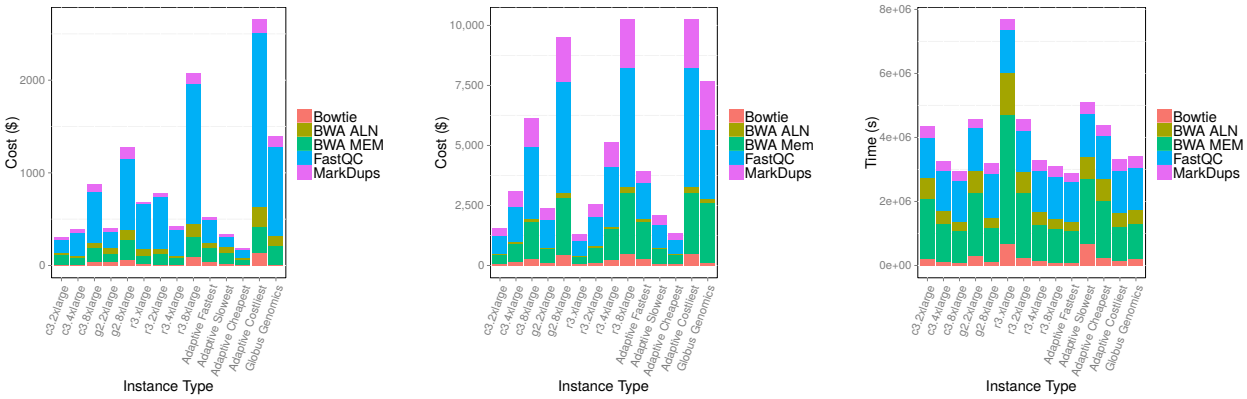
The profiling service has two pertinent limitations. Firstly, the profiler is not currently capable of capturing the utilization of specialized hardwares such as GPUs. However, the profiler has been designed for extensibility to enable the incorporation of further monitoring components. Secondly, the profiler is unable to forecast the performance of a tool when executed with drastically different dataset sizes. To avoid this problem our current research employs autonomic computing techniques and acquire online feedback via execution traces of production usage to dynamically update execution forecasts.

VI. Applying Profiles to Globus Genomics

We are extending the Globus Galaxies platform to leverage tool profiles, both to select the "best" instance type for a tool given current economic conditions and to schedule concurrent execution of tools that individually would not make full use of an instance's resources. In the later case, we leverage HTCondor to divide instances into multiple *slots*—units of specific computational capacity. We will use this model in combination with profiles obtained from the profiling service to dynamically assign slots to tools based on requirements derived from their profile.

When executing workloads on the cloud, execution time is only one dimension by which to assess tool performance. Often, users are equally concerned with cost. To analyze both the cost and execution improvements that can be achieved by using profiles, we surveyed four production Globus Genomics gateways to derive a representative workload. This workload contains over 2,000 executions of the five profiled tools over a period of 90 days. We calculate the cost of executing this workload by analyzing workload logs to determine the time at which each job was submitted and use our profiles to calculate the execution time on each instance type. We then compute cost based on historical AWS on-demand and spot pricing information at submission time for every instance type.

Fig. 6 shows the total cost and execution time for the workload using each instance type. The figure also shows the execution time and cost that are obtained when using the Globus Genomics provisioning approach and several adaptive



(a) Spot Price (b) On-demand Price (c) Execution Time

Fig. 6: The cost and time of using each instance type. The four adaptive strategies (fastest, slowest, cheapest, costliest) model autonomic provisioning approaches in which an instance type is selected based on the price or projected tool execution time.

strategies that use profiles. This approach applies a single static instance type for each gateway (m3.2xlarge and r3.8xlarge). These adaptive measures are designed to demonstrate the tradeoffs involved in selecting the fastest, slowest, cheapest, and most expensive instances. We account for the failure of the smallest instance types to execute the large *BWA MEM* and *BWA ALN* tools by using sufficiently large instance types.

Fig. 6a presents the cost of executing the tools over each spot instances. Counterintuitively, the adaptive fastest solution (in which the fastest instance type for each tool is selected) is only marginally more expensive than the adaptive slowest approach—a result that we attribute to the frequently executed *FastQC* tool which has, on average, a slightly faster execution time on the pre-selected c3.4xlarge instances than on the more expensive 8xlarge instance type. The adaptive costliest approach presents the worst-case scenario for executing a workload based on spot instance price. Its overall cost is greater than that of any individual instance type due to it selecting the most expensive instance type, regardless of tool, at the time of a job's submission. These findings demonstrate the savings that can be achieved when tool profiles are considered in the provisioning process. The cheapest solution uses multiple instance types (each the most suitable for a specific tool), as the projected cost is less than with any single instance type. Fig. 6b shows the cost when using on-demand instances for each instance type and strategy. We see a substantial increase in costs across all approaches relative to those for spot prices.

Fig. 6c shows the time required to compute the entire workload when using specific instance types and each adaptive solution. We see clear differences in performance across instance types. For example, the execution time when using r3.xlarge is almost double that of other, larger instances. In contrast, execution time with standard Globus Genomics provisioning is not significantly larger than any single instance type; this is because individual gateways are often preconfigured with relatively large instance types (r3.8xlarge), minimizing execution time over cost. The cheapest solution reduces overall cost by 86.6% while increasing execution time by 29.2% over the Globus Genomics approach. The fastest solution reduces the monetary cost and execution time by 62.6% and 15.7%, respectively. In future work we aim to incorporate these tradeoffs in our provisioning approach to optimally select instances based on real-time pricing information, predicted execution time, and resource usage.

VII. Conclusion

There are many technical challenges involved in deploying tools across a wide variety of instances types. This flexibility is both a blessing and a curse, on one hand flexibility enables tools to be executed on near-optimal instance types, however, the use of sub-optimal instance types may incur additional cost and hinder performance.

To address the needs of users in understanding the requirements of their tools we have presented an automated profiling service capable of creating rich tool profiles. The profiling service manages the deployment and execution of tools across a dynamic pool of provisioned and configured instances. It captures fine-grained performance statistics from the tool's execution and uses them to create profiles which can be used for instance provisioning, tool deployment, and scheduling. We have applied this approach to several genomics tools and shown that the resulting profiles can significantly improve performance and cost in real-world cloud provisioning scenarios.

We plan next to use the profiling service to survey a range of tools used in Globus Galaxies deployments. We will integrate the profiling service into the platform in order to transparently collect performance statistics and create profiles of jobs deployed through the gateways. We will also investigate methods for evaluating more complex configurations, such as those involving EBS storage, and integrate support for delegated IAM roles to enable use of the service without the need for user AWS credentials. Finally, we will investigate methods for more accurately determining failure. Some tools terminate

without any indication of failure; potential solutions include user-defined assertions on results and consensus methods for evaluating results from different instance types.

The source code for the profiling service is available at https://github.com/ryanchard/cloud-profiler.git.

ACKNOWLEDGMENTS

This work was supported in part by the U.S. Department of Energy under contract DE-AC02-06CH11357, the NIH under contracts 1U54EB020406-01 Big Data for Discovery Science Center, R24HL085343 Cardiovascular Research Grid, and Victoria University of Wellington Research Grant 207069. We acknowledge generous research credits provided by Amazon Web Services that helped support our work.

REFERENCES

[1] R. Madduri, K. Chard, R. Chard, L. Lacinski, A. Rodriguez, D. Sulakhe, D. Kelly, U. Dave, and I. Foster, "The Globus Galaxies platform: delivering science gateways as a service," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 16, pp. 4344–4360, 2015.

[2] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. H. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 6, pp. 931–945, 2011.

[3] R. Chard, R. Madduri, N. Karonis, K. Chard, K. Duffin, C. Ordonez, T. Uram, J. Fleischauer, I. Foster, M. Papka, and J. Winans, "Scalable pCT image reconstruction delivered as a cloud service," *IEEE Transactions on Cloud Computing*, vol. Preprint, no. 99, pp. 1–1, 2015.

[4] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright, "Performance analysis of high performance computing applications on the Amazon Web Services cloud," in *Proceedings of the 2nd International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2010, pp. 159–168.

[5] L. K. Zentner, S. M. Clark, P. M. Smith, S. Shivarajapura, V. Farnsworth, K. P. Madhavan, and G. Klimeck, "Practical considerations in cloud utilization for the science gateway nanoHUB.org," in *Procceedings of the 4th IEEE International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2011, pp. 287–292.

[6] E. Skidmore, S.-j. Kim, S. Kuchimanchi, S. Singaram, N. Merchant, and D. Stanzione, "iPlant atmosphere: A gateway to cloud infrastructure for the plant sciences," in *Proceedings of the 2011 ACM Workshop on Gateway Computing Environments (GCE)*. ACM, 2011, pp. 59–64.

[7] W. Tang, N. Desai, D. Buettner, and Z. Lan, "Job scheduling with adjusted runtime estimates on production supercomputers," *Journal of Parallel and Distributed Computing*, vol. 73, no. 7, pp. 926–938, 2013.

[8] A. Verma, L. Cherkasova, and R. H. Campbell, "ARIA: Automatic resource inference and allocation for mapreduce environments," in *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC)*. ACM, 2011, pp. 235–244.

[9] E. Elmroth and J. Tordsson, "Grid resource brokering algorithms enabling advance reservations and resource selection based on performance predictions," *Future Generation Computer Systems*, vol. 24, no. 6, pp. 585–593, 2008.

[10] L. Yang, X. Ma, and F. Mueller, "Cross-platform performance prediction of parallel applications using partial execution," in *Proceedings of the ACM/IEEE Supercomputing Conference*, Nov 2005, pp. 40–40.

[11] A. V. Do, J. Chen, C. Wang, Y. C. Lee, A. Zomaya, and B. B. Zhou, "Profiling applications for virtual machine placement in clouds," in *Proceedings of the IEEE International Conference on Cloud Computing (CLOUD)*, July 2011, pp. 660–667.

[12] T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy, "Profiling and modeling resource usage of virtualized applications," in *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, 2008, pp. 366–387.

[13] O. DeMasi, T. Samak, and D. H. Bailey, "Identifying HPC codes via performance logs and machine learning," in *Proceedings of the first workshop on Changing landscapes in HPC security*. ACM, 2013, pp. 23–30.

[14] J. Zhang and R. Figueiredo, "Application classification through monitoring and learning of resource consumption patterns," in *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS)*, April 2006, p. 10 pp.

[15] Y. Becerra, D. Carrera, and E. Ayguade, "Batch job profiling and adaptive profile enforcement for virtualized environments," in *Proceedings of hte 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, Feb 2009, pp. 414–418.

[16] H.-L. Truong, T. Fahringer, and S. Dustdar, "Dynamic instrumentation, performance monitoring and analysis of grid scientific workflows," *Journal of Grid Computing*, vol. 3, no. 1-2, pp. 1–18, 2005.

[17] P. Leitner, C. Inzinger, W. Hummer, B. Satzger, and S. Dustdar, "Application-level performance monitoring of cloud services based on the complex event processing paradigm," in *Proceedings of the 5th International Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2012, pp. 1–8.

[18] R. Chard, K. Bubendorfer, and B. Ng, "Network health and e-Science in commercial clouds," *Future Generation Computer Systems*, vol. Preprint, 2015.

[19] C. Reich, K. Bubendorfer, M. Banholzer, and R. Buyya, "A SLA-oriented management of containers for hosting stateful web services," in *Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing*, December 2007.

[20] R. Chard, K. Chard, K. Bubendorfer, L. Lacinski, R. Madduri, and I. Foster, "Cost-aware cloud provisioning," in *Proceedings of the 11th International Conference on e-Science*. IEEE, August 2015, pp. 136–144.

[21] J. Goecks, A. Nekrutenko, J. Taylor *et al.*, "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences," *Genome Biol*, vol. 11, no. 8, p. R86, 2010.

[22] I. Foster, "Globus Online: Accelerating and democratizing science through cloud-based services," *Internet Computing, IEEE*, vol. 15, no. 3, pp. 70–73, May 2011.

[23] K. Chard, M. Lidman, B. McCollam, J. Bryan, R. Ananthakrishnan, S. Tuecke, and I. Foster, "Globus Nexus: A platform-as-a-service provider of research identity, profile, and group management," *Future Generation Computer Systems*, pp. –, 2015.

[24] M. Wilde, I. Foster, K. Iskra, P. Beckman, Z. Zhang, A. Espinosa, M. Hategan, B. Clifford, and I. Raicu, "Parallel scripting for applications at the petascale and beyond," *Computer*, vol. 42, no. 11, pp. 50–60, 2009.

[25] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor–a hunter of idle workstations," in *Proceedings of the 8th International Conference on Distributed Computing Systems*. IEEE, 1988, pp. 104–111.

[26] R. Chard, K. Chard, K. Bubendorfer, L. Lacinski, R. Madduri, and I. Foster, "Cost-aware elastic cloud provisioning for scientific workloads," in *Proceedings of the 8th International Conference on Cloud Computing (CLOUD)*. IEEE, June 2015, pp. 971–974.

[27] R. Montella, D. Kelly, W. Xiong, A. Brizius, J. Elliott, R. Madduri, K. Maheshwari, C. Porter, P. Vilter, M. Wilde, M. Zhang, and I. Foster, "FACE-IT: A science gateway for food security research," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 16, pp. 4423–4436, 2015.

[28] K. Chard, R. Madduri, X. Jiang, F. Dahi, M. W. Vannier, and I. Foster, "A cloud-based image analysis gateway for traumatic brain injury research," in *Proceedings of the 9th Gateway Computing Environments Workshop*. IEEE, 2014, pp. 13–16.

[29] R. Chard, S. Sehrish, A. Rodriguez, R. Madduri, T. D. Uram, M. Paterno, K. Heitmann, S. Cholia, J. Kowalkowski, and S. Habib, "PDACS: a portal for data analysis services for cosmological simulations," in *Proceedings of the 9th Gateway Computing Environments Workshop*. IEEE, 2014, pp. 30–33.

[30] S. Andrews *et al.*, "FastQC: A quality control tool for high throughput sequence data," *Reference Source*, 2010.

[31] "PICARD mark duplicates," http://broadinstitute.github.io/picard/, accessed: 2015-11-13.

[32] H. Li and R. Durbin, "Fast and accurate short read alignment with Burrows–Wheeler transform," *Bioinformatics*, vol. 25, no. 14, pp. 1754–1760, 2009.

[33] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with Bowtie 2," *Nature methods*, vol. 9, no. 4, pp. 357–359, 2012.

[34] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM," *arXiv preprint arXiv:1303.3997*, 2013.