Mashup: Making Serverless Computing Useful for HPC Workflows via Hybrid Execution

Rohan Basu Roy
Northeastern University

Tirthak Patel
Northeastern University

Vijay Gadepally
MIT Lincoln Laboratory

Devesh Tiwari
Northeastern University

Abstract

This work introduces Mashup, a novel strategy to leverage serverless computing model for executing scientific workflows in a hybrid fashion by taking advantage of both the traditional VM-based cloud computing platform and the emerging serverless platform. Mashup outperforms the state-of-the-art workflow execution engines by an average of 34% and 43% in terms of execution time reduction and cost reduction, respectively, for widely-used HPC workflows on the Amazon Cloud platform (EC2 and Lambda).

CCS Concepts: • **Computer systems organization** → **Cloud computing**; *Heterogeneous (hybrid) systems*; • **Computing methodologies** → *Distributed computing methodologies*; Massively parallel and high-performance simulations.

Keywords: Serverless Computing, Cloud Computing, HPC Workflows, Hybrid Execution

ACM Reference Format:
Rohan Basu Roy, Tirthak Patel, Vijay Gadepally, and Devesh Tiwari. 2022. Mashup: Making Serverless Computing Useful for HPC Workflows via Hybrid Execution. In *27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '22), April 2–6, 2022, Seoul, Republic of Korea.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3503221.3508407

1 Introduction

HPC users are increasingly chaining multiple applications together to form a scientific "workflow". This is because with increasing complexity, maintaining and modularly growing a single monolithic application can become challenging [64, 70, 81]. However, these scientific workflows themselves can become very complex — each workflow can have tens to thousands of tasks, and each task can have very different computational resource requirements and the degree of parallelism can change over time [23, 25, 87].

Due to the complex structure and varying parallelism degree in different phases of a workflow, a user needs to over-provision the compute resources to meet the demand at the peak of parallelism [30, 53]. In response, domain users have developed multiple workflow managers to make workflow execution efficient [5, 17–19, 31, 67]. This is especially popular on cloud computing platforms where resources can be reserved and released more quickly than a typical on-premise cluster [11, 40, 53, 84]. While these approaches have been popular, they still suffer from resource under-utilization, over-provisioning challenges, and high expenses [20, 46, 52, 62]. Motivated by these trends, this paper explores: "*how can scientific workflows take advantage of the next emerging trend in cloud computing – serverless computing*".

Unfortunately, the serverless computing model is not traditionally designed for scientific applications [12, 24, 34]. Serverless computing platforms pose multiple design and implementation challenges that make it difficult for scientific workflow execution. As our study shows, even if one could engineer a system to leverage the serverless platform for scientific workflows, the resulting execution would be sub-optimal. This work describes the design and implementation of a system that overcomes these challenges and makes serverless attractive for scientific workflows.

Contributions. In particular, this work makes the following contributions.

- We propose Mashup, a novel solution to leverage the serverless computing model for executing HPC workflows. The key idea behind Mashup is to execute HPC workflows in a hybrid fashion – taking advantage of both traditional VM-based cloud computing platforms and emerging serverless platforms.

- Mashup designs novel techniques to mitigate the challenges of executing HPC tasks on serverless computing platforms (e.g., the challenge of cold-start, stateless nature, and scalability). Mashup's prototype implementation demonstrates the advantage of the

PPoPP '22, April 2–6, 2022, Seoul, Republic of Korea

ACM ISBN 978-1-4503-9204-4/22/04…$15.00
https://doi.org/10.1145/3503221.3508407

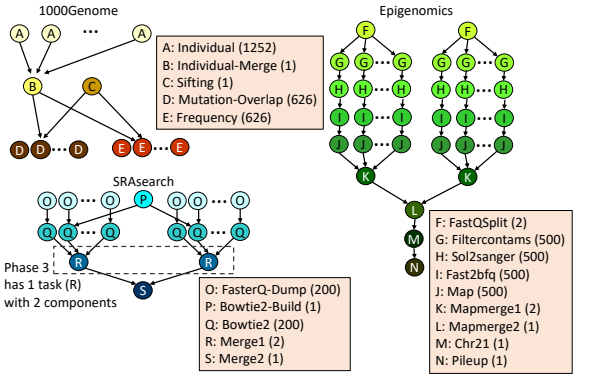

Figure 1. DAG structures of the 1000Genome, SRAsearch, and Epigenomics workflows (the numbers inside brackets denote the number of components of the corresponding task).

approach and its novel techniques by speeding up the execution of complex, real-world HPC workflows and saving the incurred expense on the cloud computing platform.

- Our extensive experimental evaluation confirms that Mashup is effective in reducing executing time and cost for widely-used HPC workflows (e.g., 1000Genome, SRAsearch, Epigenomics [9, 57, 66]) on Amazon Cloud Computing platform (EC2 and Lambda). Mashup outperforms the state-of-the-art approach for workflow execution engines (e.g., Pegasus and Kepler [2, 18]) by an average of 34% and 43% in terms of execution time reduction and cost reduction, respectively.

2 Background and Motivation

HPC workflows of various domains such as biology, chemistry, and astrophysics consist of a large number of loosely-coupled components [4, 7, 26, 29, 59, 61]. Hence, these workflows are commonly expressed as *Directed Acyclic Graphs* (DAGs) that establish the dependencies of the individual components. Fig. 1 shows the DAGs of three workflows: 1000Genome, SRAsearch, and Epigenomics, which are widely used by the HPC community [43, 47, 76, 77, 85, 92]. The number of components in each of these workflows can range from a few hundred to a few thousand. Some of these components do not have data or state dependency between them and are inherently parallel. Other components with dependencies are expressed according to their precedence pattern in the DAG. Before we motivate the problem, we briefly review some definitions and terms.

Components. A component is the most basic unit of the execution in a scientific workflow. Different components can run the same code but use different input data (e.g., all the circles in Fig. 1 correspond to a unique component; circles with the same color-coding execute the same code/logic).

Phase. The workflow components that can run in parallel, with no precedence dependency among each other, are jointly referred to as a *phase* (e.g., Epigenomics, shown in Fig. 1, has 9 phases).

Task. Different components within a phase that execute different code/logic are referred to as different *tasks*. A phase consists of one or more tasks (e.g, phase one of the workflow 1000Genome in Fig. 1 has one task named Individual, while phase two of the same workflow has two tasks named Individual-Merge and Sifting).

Note that a task can have one or more components. These components run the same piece of code with different inputs (e.g., Fig. 1 shows that the task Individual in 1000Genome workflow has 1,252 components, while Individual-Merge and Sifting have one component each). A phase can have multiple tasks. Tasks and their corresponding components within a phase can run concurrently (e.g., phase three of 1000Genome contains two tasks, Mutation-Overlap and Sifting, each with 626 components. All of these components can execute in parallel). However, tasks across phases can have dependencies. These dependencies can be at the component level. Some components may be dependent on multiple components of the previous phase, or a single component being the producer for multiple components in the next phase. For example, the single component of the task Individual-Merge of 1000Genome is dependent on all the 1,252 components of the task Individual.

To orchestrate the execution of these workflows, several workflow management platforms like Pegasus, Kepler, and Polyphony have been developed [2, 18, 67]. They control the execution by appropriately spawning workflows based on their DAG precedence. Their smallest unit of computation is either a physical server or a virtual machine (VM) in a cloud. These individual execution units are called nodes. To exploit the code parallelism, multiple nodes are reserved for the execution of a workflow, forming a cluster.

.

What are the sources of inefficiency when executing scientific workflows on traditional clusters? Domain scientists run workflows using on-premise parallel compute clusters, or using cloud computing clusters [11, 25, 40, 80]. As illustrated above, scientific workflows can have complex structures and varying degrees of parallelism during different phases. This means that a user needs to over-provision the compute resources to meet the demand at the peak of parallelism. But, at other times, this may lead to resource wastage as allocated computing resources might be idle.

A solution to this problem is the use of cloud computing platforms [16, 40, 46, 52, 61, 89]. Using VM-based cluster mitigates the problem of resource under-utilization, but only to a certain extent. This is because provisioning and dynamically changing the VM-based cluster size has long latency (up to 40 minutes in some cases [15, 42, 54]).

While this approach has been popular, it still suffers from resource under-utilization and over-provisioning challenges [20, 46, 52, 62]. A consequential challenge is the cost/expense incurred by the end-user. To achieve better performance, end-users tend to over-provision resources and pay high bills to the service provider [20, 55]. Over-provisioned idle VMs still cost the end-users.

What is the serverless computing model and how can it alleviate the above challenges? Serverless computing, also known as *Function-as-a-Service* (FaaS), is rapidly being adopted as a computing paradigm in public clouds, especially with the advent of AWS Lambda, Google Cloud Functions, and Microsoft Azure Functions. In serverless computing, *serverless functions* or execution units are microVMs (e.g., Firecracker [1]) or containers (e.g., OpenWhisk [45]) with limited computing resources and fixed execution time limits. Serverless functions spawn up with pre-configured file systems and runtime environments, thus relieving the application developers of the additional overhead of setting up the environment. The users only need to upload their application logic to the cloud provider, in the form of user-defined functions along with the dependencies.

Functionally, the difference between microVMs with traditional cloud VMs is that they are lightweight and can spawn up relatively quickly. Unlike VM-based clusters, serverless functions can scale up and down elastically depending upon the resource requirements. This helps in solving the problem of over- and under-utilization of resources. Also, users are charged for the exact amount of computing resources and only for the period that they use the resources. This helps to reduce the recurring cost due to resource wastage, which is frequently incurred in traditional VM-based cloud clusters. However, it also poses new challenges that need to be solved for efficient execution of scientific workflows on serverless platforms.

What new challenges does serverless computing pose for HPC workflows? Despite its advantages, serverless computing poses several challenges, namely: **(1) stateless execution, (2) execution timeout, (3) cold-start and scalability overhead at high concurrency.**

By design, serverless functions are stateless and cannot directly communicate with other concurrent functions. This characteristic is unsuitable for HPC workflows which inherently require tasks to communicate with each other. Second, serverless functions have caps on resource usage (e.g., execution timeout). For example, AWS Lambdas have a strict execution time limit of 15 minutes and 512 MB of disk space with up to a maximum of 10 GB of memory. Many tasks in a scientific workflow cannot finish their execution under such strict time limits.

Third, cold-start overhead is incurred because a task code needs to be loaded in a container before execution. This

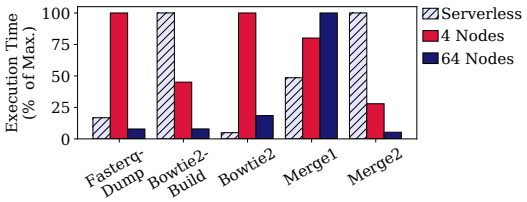

Figure 2. The preferable choice of execution environment for the tasks in a workflow varies depending upon the resource requirements.

is particularly undesirable for tasks in a workflow that is short-running. This is further compounded by the scalability bottleneck at high concurrency observed in a serverless environment (discussed in Sec. 3). Although a high level of concurrency may not be a common and frequent characteristic for enterprise applications [79, 83], high concurrency is a requirement for many scientific workflows (e.g., the task Individual in 1000Genome has 1,252 concurrent components).

Besides the above hurdles, a major challenge is determining which platform is optimal for executing a particular task in a workflow. **The optimal execution environments (serverless vs. VM-based large clusters) for different tasks within a workflow can be different**. We use SRAsearch workflow as an illustrative example to demonstrate this. For this experiment, we utilize the widely-used *r5.large* instance type on AWS EC2 because it incurs a similar expense per unit time as that of Amazon Lambda (the serverless platform). The number of nodes in the VM-based cluster is varied from four nodes to 64 nodes to illustrate how increasing the VM-based resources (more parallelism) may affect the preferred execution location of a task. On a serverless platform, the different components of a task are executed by separate serverless functions. Recall that a task in a workflow can have multiple components and these components can be executed concurrently. SRAsearch has five tasks: FasterQ-Dump, Bowtie2-Build, Bowtie2, Merge1, and Merge2, as shown in Fig. 1; each task has a different number of components.

Fig. 2 shows that different tasks of SRAsearch achieve lower execution times on different types of execution environments, depending upon the size of the VM-based cluster. For example, the task Fasterq-Dump achieves faster execution on a serverless platform when the VM-based cluster is smaller, but the trend reverses when the number of nodes is increased to 64. This is because a four-node cluster does not provide sufficient parallelism to Fasterq-Dump to execute fast enough (multiple components contend for the same resources). However, at a higher node count, the resource contention problem on the VM-based cluster is alleviated, and it outperforms the serverless execution.

Serverless functions may not be as powerful as VM clusters and they have other associated overheads (Sec. 3). Due

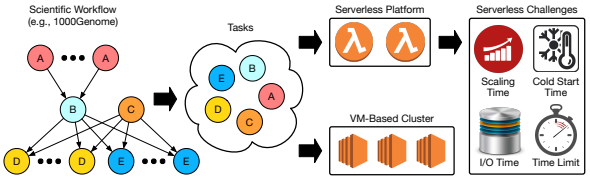

Figure 3. Mashup's hybrid execution model for scientific workflows.

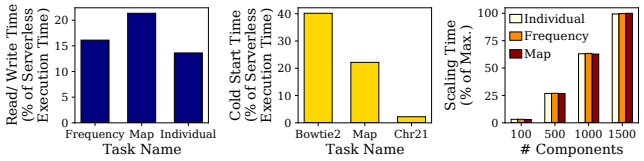

Figure 4. (a) I/O time, (b) cold start time, and (c) scaling time significantly impact the performance of serverless functions.

to these challenges, the VM-based clusters outperform but provisioning large VM-based resources result in higher cost and resource under-utilization. Overall, the optimal execution platform depends on the concurrency level and the computation/memory intensity of the task; it is non-trivial to determine apriori. *Therefore, Mashup designs and implements a novel, hybrid execution strategy to find the optimal execution platform and mitigate other previously discussed serverless-specific challenges to minimize the execution time of HPC workflows.*

3 Mashup: Design and Implementation

The basic design principle behind Mashup is to take advantage of serverless platforms for executing concurrent components of a scientific workflow. The base design is to place all tasks with more components than the number of available cluster nodes, on the serverless platform. *We implement and refer to this basic design as Mashup (without Placement Decision Controller or PDC), or simply* ***Mashup w/o PDC****.* Although promising, it suffers from certain serverless-specific inefficiencies. Mashup improves its design by reducing the impact of those inefficiencies. Mashup designs and implements a placement decision controller (PDC) that determines the task placement: serverless vs. VM cluster (Fig. 3).

Mashup's PDC takes a two-step approach to determine the optimal platform for a task. First, Mashup executes all the tasks and their components in the VM-based cluster of a given size and records the execution time. Then, in the second step, Mashup executes all the tasks and their associated components on the serverless platform. Mashup compares the execution time for each task and chooses the platform where it achieves lower execution time. This is related to our discussion in the previous section, where we showed that some tasks are more suitable for execution on VM-based clusters, while others may achieve lower execution time on a serverless platform. However, a simple comparison of execution time might not be possible or sufficient since the serverless execution suffers from various challenges (e.g., time cap, I/O overhead, statelessness).

Getting around the problem of execution time cap. Mashup mitigates this challenge by using checkpointing as a tool to store the state of the functions in remote storage. This checkpointing is performed 30 seconds before the time limit is reached. Then, the next set of serverless functions that start the task from its stored state is spawned. Note that, checkpointing is not required for running tasks on a VM cluster. All placement decisions and execution/cost analysis of Mashup includes these checkpoint/restart overheads (i.e., I/O latency related to checkpoint/restart toward execution time and storage cost toward expenses).

Mitigating the I/O overhead of stateless execution. Upon the completion of the execution of a serverless function, all the data generated and stored in the local storage of the function's microVM is erased. Thus, the only way that the serverless functions can share data among multiple phases is to communicate via external remote storage that acts as a mediator. The state needs to be transferred via a storage medium across two phases of a scientific workflow. This extra layer of communication increases the latency of scientific workflow execution because remote storage bandwidth now impacts the performance of a task. This I/O overhead (read and write) via the external storage can be significant as characterized in detail in [6], especially for tasks with multiple components, where all the components do I/O via external storage to communicate their states across phases.

The overhead is not uniform across all tasks – some tasks incur more overhead than others. For example, Fig. 4(a) shows the task Map has higher I/O overhead compared to other tasks such as Frequency and Individual. This is because Map performs both read and write, while the others perform only read or write. This I/O overhead affects a task's optimal execution platform (serverless vs. VM-based cluster) – this is accounted for by Mashup's PDC controller. For example, the task Frequency, which has 626 components, appears ideal for serverless execution due to its high degree of parallelism, but on a 64-node VM-based cluster, it executes 2× faster than a serverless execution. This performance degradation on the serverless platform is due to I/O overhead.

Though the computing time of a task might be lesser on serverless than on a VM cluster, the I/O time can increase the overall execution time of the task. With the help of its PDC, Mashup identifies such tasks and executes them in a VM cluster. Mashup's decision of an appropriate execution environment for such a task changes according to the cluster size or the network I/O bandwidth of the cluster.

Addressing the effect of cold start times on short-running tasks. Cold start overhead can be significant (up to a few seconds) for some tasks in scientific workflows. For example, Fig. 4(b) shows that for the task Bowtie2 (belonging to SRAsearch workflow, with 200 components), the cold start time is almost 40% of the task execution time. However, for long-running tasks with one or few components like Chr21 (belonging to Epigenomics workflow, with 1 component), the cold start time has little impact on its execution time.

Mashup mitigates this challenge in two ways: (1) Mashup chooses to not place very short-running tasks (less than one second) on serverless platforms because the cold start cost can be up to a few seconds in the worst case [41, 79, 83]. A conservative two-second cold start overhead is always added to serverless execution time during PDC decision-making. (2) Mashup makes an exception for the previous decision only if such short-running tasks have high concurrency and re-appear frequently in the workflow (e.g., Mapmerge in Epigenomics workflow). Cloud providers typically keep microVMs and their memory contents alive for 5-10 minutes after a serverless microVM has completed its execution, to avoid future cold-starts. Mashup takes advantage of this policy for frequently re-appearing short tasks. Also, Mashup actively pre-warms the task by prefetching to avoid the cold start overhead for tasks with high concurrency to take advantage of serverless parallelism.

Achieving high concurrency in a scalable manner. During our experiments, we discovered that serverless platforms are not as efficient at high levels of concurrency (> 100 components). Fig. 4(c) shows that as the concurrency increases, the scaling time also increases (i.e., high scaling time at higher concurrency level). Scaling time is defined as the time difference between the start time of the first serverless function and the start time of the last serverless function, where all the functions belong to the same task (i.e., multiple components of the same task).

Scaling issues are an artifact of multiple challenges. When a serverless microVM (or a container) is initialized, the task code along with its dependencies, which reside in a remote server, is read into the storage of the microVM. When a task has multiple components and each of them requires separate serverless functions, the overhead of performing all the aforementioned steps scales up linearly. The scheduler takes more time to decide due to increased load and the microVMs take longer to start up as the likelihood of the same server hosting multiple microVMs increases.

For serverless platforms to be attractive for scientific workflows, the scaling time has to be low because, in real-world workflows, multiple tasks have a high number of components that need to be executed concurrently. Mashup makes an interesting observation that scaling time is largely independent of the code executed by individual components. This allows Mashup to estimate the execution time of a task using serverless functions without actually executing all of its components. Mashup executes only one component using a serverless function and estimates the cost of placing the full task on serverless via a simple analytical model that captures the linear behavior of scaling time. Mashup compares this estimated time with the execution time on the VM cluster and determines the optimal platform.

Algorithm 1 Mashup's Placement Decision Controller

```
Input: The DAG of a workflow
Output: Placement decision for all the tasks in the workflow
for all the tasks in the workflow do
    Measure serverless execution time of the task (T^func)
        Scale up the components of the task
        Start-up microVMs for the execution
        Read input from the remote storage to the microVMs
        Execute in each microVM
        Checkpoint for storage and re-start, if required
        Write output to storage
    Measure VM cluster execution time of the task (T^VM)
        Distribute input from master node to all workers
        Execute task in worker nodes
        Write output to the master node

    if T^func < T^VM then
        Placement decision for the task is serverless
    else
        Placement decision for the task is traditional VM cluster
    end if
end for
```

Next, we describe the optimization formulation of Mashup that integrates all previously described mechanisms to determine the optimal execution platform for tasks in a workflow (also summarized in Algorithm 1).

Mashup optimization formulation. Mashup solves the optimization problem of finding the preferred execution environment for different tasks in a workflow to minimize the execution time of the complete workflow. A workflow (W) has phases (P_i, with $1 \leq i \leq P_{max}$), with each phase having tasks (t_j, with $1 \leq j \leq t_{max}$), where each of the tasks have $C_{i,j}$ number of components. For tasks that are executed on a serverless platform, the execution time consists of serverless specific overheads like scaling time, cold start time, and I/O time, along with the time consumed in running the task by concurrent serverless functions. The serverless specific overhead part of the execution time is dependent on the number of components of the task. All of these components run in parallel, and their combined runtime is equal to the runtime of one component running in a serverless function. Thus, the execution time of a task t_j in phase P_i with $C_{i,j}$ components, each executing on separate serverless functions is

$$T_{i,j}^{func} = \alpha \times C_{i,j} + R^{func}(t_j) + \beta \quad (1)$$

α and β are experimentally-derived constant factors. α captures the proportionality by which the serverless specific overhead scales with the number of components of a task.

$R^{func}(t_j)$ is the runtime of a component of the task t_j in phase P_i in a serverless function.

When a task runs on a VM-based cluster, all of its components run on the nodes of the cluster. Depending on the resource utilization of the tasks, the components can contend for resources in the cluster, thus increasing the execution time of the task. Unlike the serverless platform, task execution does not incur any additional overhead like scaling time or cold start time. The execution time of a task t_j in phase P_i with $C_{i,j}$ components, executing on a VM cluster is

$$T_{i,j}^{VM} = (R^{VM}(t_j))^{\gamma \times C_{i,j}} \quad (2)$$

Here γ (> 1) is a constant of proportionality by which the runtime of a task, $R^{VM}(t_j)$, increases with the number of components. Mashup optimizes the following expression:

$$\underset{m_{i,j} \in \{0,1\}}{\text{argmin}} \left[\sum_{i=1}^{P_{max}} \sum_{j=1}^{t_{max}} m_{i,j} \times (T_{i,j}^{func}) + (1 - m_{i,j}) \times (T_{i,j}^{VM}) \right] \quad (3)$$

Here $m_{i,j}$ can attain two values, 0 and 1. When $m_{i,j} = 1$, then the task t_j of phase P_i undergoes a serverless execution, else it is executed on a VM cluster. Note that Mashup's PDC experimentally determines the value of the factors (α, β, and γ) since these depend on the workflow and the cloud platform. Mashup's PDC autonomously determines all the factors and solves the optimization problem without requiring user engagement. As our evaluation confirms, Mashup's solution is aligns well with the optimal execution environment determined via exhaustive search (Sec. 5, Fig. 9).

Miscellaneous design considerations. Finally, we discuss a few generic design issues of Mashup.

Execution overheads. We note that Mashup requires executing the workflow once on the VM-based cluster which may appear as additional overhead. However, most current scientific workflow resource managers also require executing the full workflow once to extract parallelism within the workflow. Mashup leverages the same run to make placement decisions. Mashup requires executing certain tasks on the serverless platform, but as our design describes these executions are minimized via various optimizations. Also, typically the same scientific workflow is run multiple times in production, amortizing the one-time cost.

Selection of VM instance type. The execution time of a task on a VM-based cluster is affected by the chosen VM instance type. For fairness and consistency, our default VM-based cluster per node expense is approximately the same as that of the serverless platform per unit time. However, as our evaluation confirms (Sec. 5), for both cheap and expensive VM instances, Mashup continues to provide benefits relative to its chosen baseline VM-only execution mode. This is because Mashup's design is targeting generic major sources of inefficiencies instead of focusing on only outperforming a specific cluster composition.

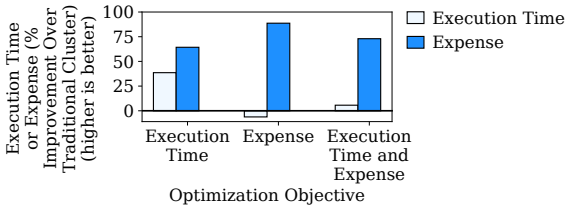

Figure 5. Reducing execution time also reduces expense (SRAsearch).

Choice of execution time as the primary objective metric. One design consideration of Mashup could have been to minimize both execution time and expense simultaneously. However, by minimizing only execution time, the expense is reduced due to an overall reduction of time for which computing resources remain active for running a workflow (Fig. 5 as an example for SRAsearch; other workflows yield similar trends). Whereas, if expense reduction is chosen as an objective or expense reduction and execution time reduction are given equal priority, the optimization tends to choose the serverless platform for the execution of most tasks as serverless functions are much cheaper than VM clusters. Thus, even though the expense is reduced, the execution time increases by a greater extent as many tasks are not suitable for serverless execution. Mashup, with its focus on execution time only, yields roughly similar cost savings as putting equal weight on both objectives. We were also driven by the preference of application users who prefer simple objective criteria and simple optimization frameworks where reasoning about the execution time is easier.

Rationale for task-level placement decisions. Mashup performs task-level placement decisions instead of component-level placement decisions (i.e., it places the different components belonging to the same task in the same execution platform). This is because component-level placement decisions can exponentially increase the overhead associated with the placement decision controller as all different combinations of component placements have to be individually profiled for all the tasks. A component-level approach also complicates the data movement and exchange patterns among the different components as they might be executed on different platforms, leading to an overall increase in the execution expense and time.

Optimal VM configuration. Mashup's placement decision controller ensures that the optimal VM-cluster configuration is used for the execution of the workflow. For example, a workflow (e.g., SRAsearch) can benefit from having a cluster divided into two sub-clusters with two separate master

nodes, so that different components running in parallel do not contend for the same computational resource. Mashup recognizes the most optimal VM configuration and uses that as a baseline for the VM cluster to ensure that the VM cluster does not suffer from apparent resource bottlenecks.

Mashup: Implementation. Overall, Mashup has been designed and implemented with the focus on keeping the barrier to adoption for scientific workflows very limited. The user only needs to provide the executables of the tasks with the workflow DAG structure to Mashup. Mashup takes charge of the execution of the workflow – the placement decision controller decides which task should be executed on the serverless platform.
Mashup leverages the command-line interface (CLI) of the cloud provider for installation and configuration with the user account credentials. Mashup sets up a VM cluster according to the user's chosen VM family and node count. Then, Mashup sets up the tasks of the workflow in the cluster as well as in the cloud provider's serverless platform. Then, Mashup configures remote storage to be used for exchanging data among multiple tasks running in different platforms and loads the initial input data (e.g., AWS Simple Storage Service (S3) bucket).

For a serverless platform, Mashup uses AWS Lambda. It packages and sends all the necessary task executables directly to the Lambda service controller. Mashup resides on a local machine and through the help of the cloud provider's CLI commands, it executes the tasks in different platforms of the cloud (serverless or VM cluster). Mashup offers the checkpointing capability and automatically saves a task state in S3 to perform checkpointing, once it approaches the execution time limit of the serverless platform. Mashup maintains multiple copies of remote storage with frequent consistency checks to recover from failures. It checkpoints the states of the executed tasks to account for the chance of serverless platform failures.

4 Experimental Methodology

Evaluated Workflows. Mashup's evaluation is driven by the following workflows: **1000Genome, SRAsearch,** and **Epigenomics**. These workflows are chosen for multiple reasons. They have been extensively used by the HPC community as standard benchmarks to compare the performance of workflow management systems and schedulers [43, 47, 76, 77, 85, 92]. The workflows consist of multiple possible precedences and connection dynamics in a DAG: that is, fan-out (multiple components dependent on the output of a single component from a different application executed in the previous phase), fan-in (a component dependent on multiple components executed in the previous phase), and strong connection (all components in a phase are connected to all components in the next phase). The DAG structure of these workflows are diverse (Fig. 1), and their resource requirements are also significant in terms of resources like compute power, memory, and network bandwidth [39, 58, 65].

These benchmarks are also a part of multiple scientific research projects and solve critical problems. The 1000Genome (5 tasks with a total of 2,506 components working on 600 GB of data) benchmark studies human genetic variation and works on the most detailed catalog of gene sequences. SRAsearch (5 tasks with a total of 404 components working on 6 TB of data) uses several search optimizations to scan through biological data sequences to unveil patterns of proteins and DNA. Epigenomics (9 tasks with a total of 2007 components working on 5 TB of data) is a compute-intensive workflow for automating various operations in scientific sequence processing.

Competing Techniques. We compare the performance of Mashup against the following methods:

Traditional VM-based Cluster (Traditional Cluster) and Serverless-only execution. VM-based cluster execution corresponds to the traditional and most common way of executing HPC workflows in the cloud computing setup. A cluster of VMs on multiple nodes is reserved, with one node as the master node. Tasks in each of the phases are spawned in parallel, and consecutive phases are spawned sequentially according to the precedence order defined in the DAG of the workflow. Since the whole computation occurs within a cluster, there is no need to maintain external storage, and hence, it does not incur any storage expenses.

We observed that using a single cluster can be inefficient for certain workflows due to resource contention (*e.g.,* Merge1 in SRAsearch), and two clusters each of half-size might yield better execution time results. While this information is not available apriori, we utilized this information to make the "traditional VM-based cluster" approach more competitive and provide higher baseline performance for Mashup. In the serverless-only execution strategy, all the tasks are executed by serverless functions and no VM clusters are involved. Checkpointing is used for components that exceed the run-time limit of serverless functions, and hence, remote storage effects on execution time and cost are accounted for.

State-of-the-art Approaches. We also compare the performance of Mashup with the *state-of-the-art* workflow management systems; *Pegasus* and *Kepler*. We use the most recent release (October-2018 for Kepler, and September-2019 for Pegasus) of the state-of-the-art HPC workflow managers. These are the most widely used workflow managers in the HPC community [38, 69, 75, 88], although these approaches are agnostic to serverless computing platforms. Given a

workflow DAG and a VM cluster, they perform a profiling run to learn the structure and introduce several optimizations to decide how to schedule different applications to exploit parallelism. Since the entire workflow runs on a VM cluster, they do not need any external storage. Note that, Mashup does not leverage these optimizations, but still outperforms them. We also note that both Pegasus and Kepler workflow managers require pre-profiling a workflow, incurring an overhead similar to Mashup.

Evaluation Platform. We use different services of the Amazon Web Services (AWS) to run the workflows. As a serverless platform, we use AWS Lambdas, which is currently one of the most widely used commercial *Function-as-a-service (FaaS)* platforms. Each of the serverless functions executing a component of an application in a workflow corresponds to one AWS Lambda with 3GB of memory (expense is $0.12/hr/function). Compute nodes are AWS Elastic Compute Cloud (EC2) VMs. To make each of the VMs similar to an AWS Lambda, we choose `r5.large` instances for our main set of experiments as they have similar per second execution expense ($0.12/hr/node) as AWS Lambdas.

Furthermore, to demonstrate that our findings/trends are not sensitive to the choice of VM-type, we evaluated Mashup's performance with other EC2 VM families that have relatively lesser and more per second execution expense than that of an AWS Lambda (`m5.large`, referred to as cheap VM family, has $0.08/hr/node expense; `r5b.large`, referred to as expensive VM family, has $0.15/hr/node expense).

We evaluate Mashup with a varying range of cluster sizes from 2 nodes to 96 nodes. We choose this range of cluster size as the execution time of the workflows varies considerably from a 2 node cluster to a 96 node cluster (85% reduction in execution time from a 2 node cluster to a 96 node cluster), but on further increasing the cluster size, the performance in terms of execution time does not improve considerably. As an external storage service required in a hybrid execution environment, we use AWS Simple Storage Service (S3) buckets, which are accessible both from Lambdas as well as EC2s. While for the experimental demonstration, Mashup is evaluated on AWS Lambda, its design principles are generic and can be ported to other platforms. For statistical significance and accounting for cloud variability, we performed our experiments ten times and averaged the execution time. We confirm that our experiments and benefits are repeatable and statistically significant.

Mashup is configurable for end-users to specify different types of EC2 instances. The algorithm and optimization formulation extends naturally, but as expected, determination of optimal placement may require running the components of workflows on different types of EC2 instances.

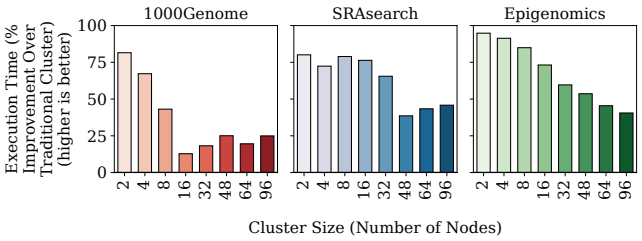

Figure 6. Mashup consistently reduces execution time of a workflow.

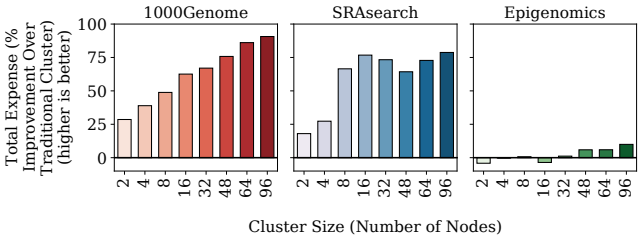

Figure 7. Mashup can also reduce expense due to a reduction in the overall workflow execution time.

Evaluation Metrics. All performances are expressed as a percentage improvement over a traditional VM cluster execution. Hence, Mashup's improvement is $(1 - \frac{\text{Mashup performance}}{\text{VM cluster performance}}) \times 100\%$. We evaluate the workflow execution time and the execution expense, which includes the combined expense of running all the VM nodes in a cluster, the expense of running each of the serverless functions (AWS Lambdas), and the expense of maintaining an S3 bucket during execution (applicable only for Mashup).

5 Mashup: Evaluation and Analysis

In this section, we analyze the performance of Mashup.

Does Mashup reduce workflow execution times and the overall expense? Fig. 6 confirms that Mashup is effective in reducing the workflow execution time. On an average, Mashup improves the workflow execution time of 1000Genome, SRAsearch and Epigenomics by 37%, 63% and 68% respectively over traditional cluster (Fig. 6). The improvement in Epigenomics is more than the other two workflows because Epigenomics has more percentage of tasks that are suitable for serverless execution. For example, the task FastQSplit in Epigenomics, which consists of more than 35% of the workflow execution time, is greatly benefited by execution on serverless functions. A larger cluster has more capacity in terms of computing resources and hence the performance benefits from running certain tasks in serverless reduce with an increase in cluster size.

Further, Fig. 7 shows that Mashup is effective in reducing the expense in running the workflows. Mashup reduces expense by an average of more than 62% for 1000Genome and

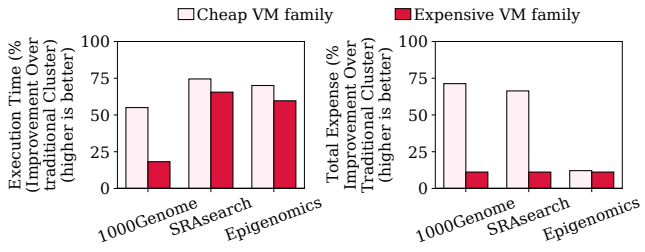

Figure 8. Mashup performs well with different family of VM instances in its hybrid environment (results for 48 node cluster).

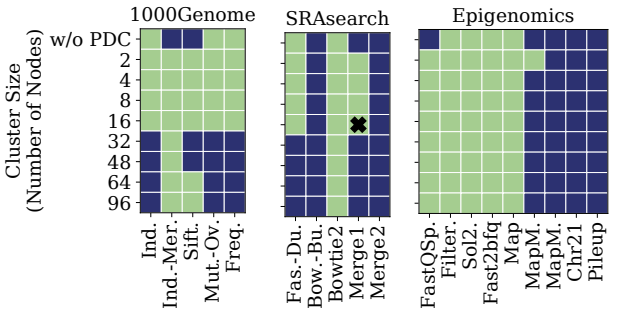

Figure 9. Mashup chooses the optimal execution environment for tasks in a workflow (green denotes execution by serverless functions and blue denotes execution in VM cluster). Only in one case Mashup is incorrect in the estimation of the optimal execution environment for a task (shown by a cross).

59% for SRAsearch (Fig. 7). Reduction in expense is primarily because of the reduction in the execution time. The careful design of Mashup ensures that serverless-specific expenses (e.g., storage and data movement) do become significant enough to outweigh the savings in cost achieved via the reduction in execution time. Note that the improvement in expense is much lower for the workflow Epigenomics. This is because this workflow has a lot of tasks that benefit from a serverless execution; these tasks are long-running. Since in addition to the serverless functions, the VM cluster also stays active in such periods, it adds up to the expense and hence, reduction in overall expense is limited. But, Mashup is able to improve execution time by more than 90% for Epigenomics (Fig. 6).

Though the hybrid execution environment of Mashup has both active VM cluster and serverless functions (in contrast to only VMs in a traditional VM-based execution), still the expense incurred to the cloud provider is much lower because of the overall reduction in workflow execution time. Using much less computing resources than a traditional cluster execution, Mashup can achieve lower execution time and cost. For example, even with an active VM cluster of 48 nodes, Mashup achieves 41% lower execution time and 81% lower expense than a 96 node traditional VM cluster execution for SRAsearch. We found that in all cases, using just half of VM cluster nodes than a traditional cluster execution, Mashup lowers both execution time and cost compared to the double-sized traditional VM cluster execution.

Impact of workflow size. We confirmed that Mashup provides benefits for all representative input sizes and types specified with the workflow distribution considered in the study. For example, on an average, Mashup improves the workflow execution time of SRAsearch by 60%, 63%, 66%, and 65% over a traditional cluster for four different representative input data sets [57]. It reduces expense by 58%, 59%, 63%, and 60% respectively for these inputs. These inputs represent different protein sequences, with data size varying from 5 TB to 8.4 TB.

Does Mashup provide execution and cost-effectiveness across different types of VM nodes? Recall that, for a fair comparison, in the default case, we keep the price of VM nodes approximately the same as the Lambda workers (price per unit time). To check Mashup's sensitivity across different VM price points, we evaluate Mashup on other different types of VM nodes; one with per second execution expense lower than serverless functions (cheap VM family) and another with the expense higher than serverless functions (expensive VM family).

From Fig. 8, we observe that Mashup continues to be beneficial across VM families, both in terms of workflow execution time and execution cost over a traditional VM cluster execution. Note that when the VM family becomes more expensive, the improvement reduces both in terms of expense and execution time. This is because an expensive VM family already has significantly more compute, memory, and network capacity than a cheap VM family, and hence the scope of improvement using a hybrid execution environment reduces. However, in terms of absolute performance, Mashup with the cheap VM family can reduce the execution time of a workflow by more than 52% and expense by 57% over a traditional VM cluster execution on an expensive VM family.

Why does Mashup work effectively? Mashup works effectively because the PDC phase of Mashup helps it to understand the interaction of tasks in a workflow with the underlying system hardware. This assists Mashup to make an informed decision about the suitable choice of an execution environment. In almost all of our experiments, Mashup has been successful in determining the optimal execution environment for all the tasks with one exception (Fig. 9). Note that the decision of the optimal execution environment for the tasks in a workflow changes both with cluster size as well as VM family type. Following a naive strategy (Mashup

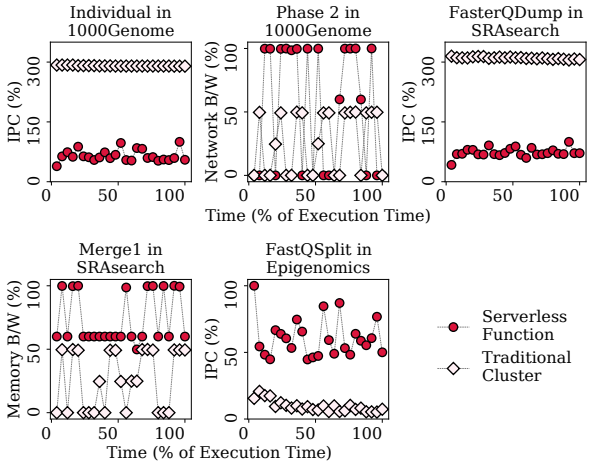

Figure 10. Underlying system metrics like instructions per cycle (IPC), network and memory bandwidth (% of maximum in a serverless function) helps to understand why profiling by Mashup helps to reduce workflow execution time.

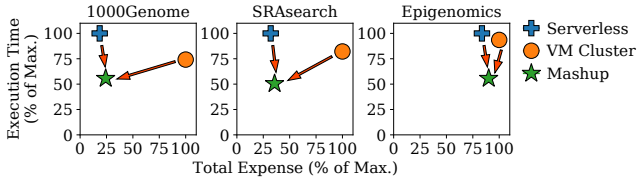

Figure 11. Mashup's hybrid execution strategy achieves best of the both worlds (smaller is better).

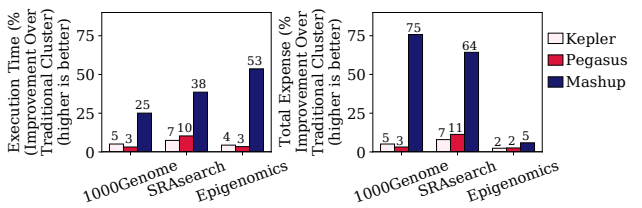

Figure 12. Mashup performs better than other state-of-the-art HPC workflow schedulers both in terms of reducing execution time and expense (results for 48 node cluster). The numbers over the bars denote percentage improvement over VM cluster.

without PDC) would result in making the same choice of execution environment (depending upon the number of components of each of the tasks; the first row in Fig. 9).

Through this simple strategy without PDC can be effective to some extent in certain cases, but it does not provide any guarantee to find the most optimal execution environment, and also the performance can deteriorate in some cases compared to a traditional VM execution. Next, we discuss some of the cases where a task execution environment decision without PDC proves to be sub-optimal.

To illustrate, we consider and discuss the execution of the task Individual in 1000Genome workflow. It has 1,253 components, so Mashup (without PDC) will spawn its components on serverless functions. However, we observe that especially when the cluster size increases, a VM cluster execution has much less execution time. This is because the instructions per second (IPC) of this task is higher in a VM cluster compared to a serverless execution, as seen in Fig. 10. The components of this task do not contend against one another for resources and hence a VM cluster with more aggregated compute power than serverless functions turns out to be beneficial. A similar behavior is observed in the task FasterQ-Dump of SRAsearch. It has 200 components, but with an increase in cluster size, its performance improves in a VM cluster compared to a serverless execution.

Similarly, there are other cases where Mashup's PDC helps it to achieve better placement of tasks. For example, the second phase of the workflow 1000Genome has two tasks, Individual-Merge and Sifting and hence Mashup (without PDC) executes them on a VM cluster. However, both of these tasks run in parallel, with processes spawned across the entire cluster, and they contend for network bandwidth to communicate with the master node. They are I/O intensive and hence bandwidth contention becomes a major bottleneck. Through its PDC phase, Mashup identifies this issue and spawns these tasks on separate serverless functions. As a result, we observe a higher achieved IPC in a serverless environment (Fig. 10).

The other key source of effectiveness is Mashup's ability to reduce serverless computing specific overheads such as cold start time, I/O time, and scaling time. Mashup's placement decision controller helps Mashup to reduce these overheads. For example, on average, Mashup reduces cold start time, I/O time, and scaling time by 90%, 61%, and 94% compared to Mashup without PDC. A serverless-only execution incurs almost 1.3× worse than Mashup without PDC in all dimensions (cold start time, I/O time, and scaling time). Hence, serverless-only execution is not a compelling design point; it incurs more than 10% execution time degradation than the VM cluster-based execution. Mashup without PDC improves the situation, and Mashup provides further improvement over Mashup without PDC in mitigating these serverless specific overheads. Finally, Fig. 11 shows that Mashup's hybrid execution strategy achieves the best of both worlds (serverless-only execution and VM-based cluster execution) – reducing execution time and expense desirably.

In practice, Mashup's estimation of the execution time of a task using PDC is more than 95% accurate. We found that variability during profiling mostly affected only short-running tasks (a few milliseconds to seconds). Mashup conservatively place those tasks on a VM-based cluster. We performed the same task placements across different inputs provided in the workflow suite and found similar performance-cost savings (32% performance and 41% cost on average).

How does Mashup perform compared to other state-of-the-art workflow managers. Here we compare Mashup against Pegasus and Kepler. They execute workflows only on VM clusters and performs several optimizations in terms of scheduling tasks as well as controlling data movement pattern. To the best of our knowledge, Mashup is the first workflow manager that systematically uses a hybrid execution environment comprising of VM cluster and serverless functions to execute HPC workflows. This hybrid approach of workflow execution is beneficial than traditional ways of executing a workflow in a VM cluster, as can be seen from Fig. 12. On average, Mashup executes workflows with a 34% less execution time compared to execution via both Pegasus and Kepler workflow managers on the same VM cluster. At the same time, Mashup also reduces the execution expense by an average of 43% due to an overall reduction in execution time.

The source of improvement is Mashup's ability to take advantage of serverless platforms for mitigating resource contention among components of a single task on VM-based execution. Unlike Mashup, Pegasus and Kepler do not take into consideration the contention for resources among multiple components of a single task. However, despite significant improvements over the state-of-the-art, we note that Mashup is complementary to Pegasus and Kepler's efforts since the optimizations embedded in Pegasus and Kepler (e.g., data reuse, redundant computation elimination, task grouping) can still be applied over Mashup.

Is Mashup portable across different cloud platforms? Mashup's design and benefits are portable beyond AWS. For example, on Google Cloud, we observed similar trends (e.g., 33% and 68% improvement in performance and cost, respectively for 1000Genome; 43% and 88% improvement in performance and cost, respectively for SRAsearch, using 16-node configuration). We noticed that the placement decision of all components remained the same. Even without the profiling, Mashup offers similar cost and performance improvement on Google Cloud (e.g., 10%, and 56% cost improvement for 1000Genome and SRAsearch, respectively without profiling versus 68% and 88% improvement with profiling; 22%, and 38% execution time improvements versus 33% and 48% for Mashup with profiling).

6 Related Works

Over the past several decades, scientific workflows have been used across multiple domains to represent the complex precedence pattern of different tasks and components [33, 43, 49, 63, 64, 84]. They have been behind several major scientific discoveries [27, 56, 70, 81]. Since the number of tasks and components in a workflow can be overwhelmingly large (mostly in 1000s), it is not possible to manage each of such components individually – hence the concept of workflow management systems like Pegasus, Kepler, DAGman, Parsl, and others have become popular [2, 3, 5, 10, 14, 17–19, 31, 44, 64, 67, 68, 71, 82]. Workflow managers frequently use different optimizations to schedule the tasks, control I/O pattern, and optimally share network resources [10, 14, 16, 18, 22, 46, 48, 51, 78, 89]. These optimizations frequently make the workflow managers suitable only for certain specific kinds of workflows, having specific computational and I/O characteristics [2, 3, 13, 21, 50, 56, 60, 64, 71, 82, 86]. But, prior efforts have not devised strategies to leverage serverless computing for reducing the execution time and cost of scientific workflows. It requires developing new strategies to mitigate serverless-specific challenges since the serverless computing model was not originally designed to serve HPC workflows as a primary objective – Mashup attempts to work in this conventional landscape.

When workflows execute tasks in a cluster, different components of it can contend for similar resources, thus increasing the overhead [11, 25, 35]. To mitigate this, some workflow managers execute workflows in a distributed manner, in multiple clusters [36, 51, 72, 73, 80, 91]. But, this approach increases the expense and also leads to resource under-utilization [8, 32]. Also, traditional workflow managers exhibit inefficiencies to schedule a large number of short-running tasks among multiple nodes in a cluster [28, 37, 74, 90]. To bridge these gaps, we design Mashup, which uses a hybrid execution model and demonstrates that it is possible to exploit a serverless execution model for speeding up the execution of HPC workflows.

7 Conclusion

To the best of our knowledge, Mashup is the first work done that systematically uses serverless computing and traditional VM cluster hybrid environment for the execution of HPC workflows. Typically HPC workflows have tasks that can be accelerated by a serverless execution. However, not all types of tasks are suitable for serverless platforms, due to several serverless computing-specific challenges. Mashup exploits the benefits of serverless computing and improves HPC workflow execution time by 34% and reduces the expense by 43%, on average over the state-of-the-art HPC workflow managers. *The framework of Mashup is open-sourced for community adoption and enabling future research. It is available at https://zenodo.org/record/5733395.*

Acknowledgments

We thank the anonymous reviewers for their valuable feedback. This work is supported by NSF Award 1910601, 1920020, and 2124897. The research was sponsored by the United States Air Force Research Laboratory and the United States

Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

[1] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. 2020. Firecracker: Lightweight Virtualization for Serverless Applications. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*. 419–434.

[2] Ilkay Altintas, Chad Berkley, Efrat Jaeger, Matthew Jones, Bertram Ludascher, and Steve Mock. 2004. Kepler: an extensible system for design and execution of scientific workflows. In *Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004.* IEEE, 423–424.

[3] Kaizar Amin, Gregor Von Laszewski, Mihael Hategan, Nestor J Zaluzec, Shawn Hampton, and Albert Rossi. 2004. Gridant: A client-controllable grid workflow system. In *37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the.* IEEE, 10–pp.

[4] Pau Andrio, Adam Hospital, Javier Conejero, Luis Jordá, Marc Del Pino, Laia Codo, Stian Soiland-Reyes, Carole Goble, Daniele Lezzi, Rosa M Badia, et al. 2019. BioExcel Building Blocks, a software library for interoperable biomolecular simulation workflows. *Scientific data* 6, 1 (2019), 1–8.

[5] Yadu Babuji, Anna Woodard, Zhuozhao Li, Daniel S Katz, Ben Clifford, Rohan Kumar, Lukasz Lacinski, Ryan Chard, Justin M Wozniak, Ian Foster, et al. 2019. Parsl: Pervasive parallel programming in python. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*. 25–36.

[6] Rohan Basu Roy, Tirthak Patel, and Devesh Tiwari. 2021. Characterizing and Mitigating the I/O Scalability Challenges for Serverless Applications. In *2021 IEEE International Symposium on Workload Characterization (IISWC)*.

[7] Andrea R Beccari, Carlo Cavazzoni, Claudia Beato, and Gabriele Costantino. 2013. LiGen: a high performance workflow for chemistry driven de novo design.

[8] Jakub Beránek, Stanislav Böhm, and Vojtěch Cima. 2019. ESTEE: A simulation toolkit for distributed workflow execution. In *The International Conference for High Performance Computing, Networking, Storage, and Analysis*.

[9] Shishir Bharathi, Ann Chervenak, Ewa Deelman, Gaurang Mehta, Mei-Hui Su, and Karan Vahi. 2008. Characterization of scientific workflows. In *2008 third workshop on workflows in support of large-scale science*. IEEE, 1–10.

[10] Junwei Cao, Stephen A Jarvis, Subhash Saini, and Graham R Nudd. 2003. Gridflow: Workflow management for grid computing. In *CCGrid 2003. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2003. Proceedings.* IEEE, 198–205.

[11] Adam G Carlyle, Stephen L Harrell, and Preston M Smith. 2010. Cost-effective HPC: The community or the cloud?. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*. IEEE, 169–176.

[12] Paul Castro, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. 2019. The rise of serverless computing. *Commun. ACM* 62, 12 (2019), 44–54.

[13] Jieun Choi, Theodora Adufu, and Yoonhee Kim. 2017. Data-locality aware scientific workflow scheduling methods in HPC cloud environments. *International Journal of Parallel Programming* 45, 5 (2017), 1128–1141.

[14] Rafael Ferreira da Silva, Rosa Filgueira, Ilia Pietri, Ming Jiang, Rizos Sakellariou, and Ewa Deelman. 2017. A characterization of workflow management systems for extreme-scale applications. *Future Generation Computer Systems* 75 (2017), 228–238.

[15] Marcos Dias De Assunção, Alexandre Di Costanzo, and Rajkumar Buyya. 2009. Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters. In *Proceedings of the 18th ACM international symposium on High performance distributed computing*. 141–150.

[16] Daniel De Oliveira, Eduardo Ogasawara, Fernanda Baião, and Marta Mattoso. 2010. Scicumulus: A lightweight cloud middleware to explore many task computing paradigm in scientific workflows. In *2010 IEEE 3rd International Conference on Cloud Computing*. IEEE, 378–385.

[17] Ewa Deelman, Dennis Gannon, Matthew Shields, and Ian Taylor. 2009. Workflows and e-Science: An overview of workflow system features and capabilities. *Future generation computer systems* 25, 5 (2009), 528–540.

[18] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira Da Silva, Miron Livny, et al. 2015. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems* 46 (2015), 17–35.

[19] Ewa Deelman, Karan Vahi, Mats Rynge, Rajiv Mayani, Rafael Ferreira da Silva, George Papadimitriou, and Miron Livny. 2019. The evolution of the pegasus workflow management software. *Computing in Science & Engineering* 21, 4 (2019), 22–36.

[20] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-efficient and qos-aware cluster management. *ACM SIGPLAN Notices* 49, 4 (2014), 127–144.

[21] P. Donnelly, N. Hazekamp, and D. Thain. 2015. Confuga: Scalable Data Intensive Computing for POSIX Workflows. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. 392–401.

[22] Rubing Duan, Radu Prodan, and Thomas Fahringer. 2007. Performance and cost optimization for multiple large-scale grid workflow applications. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*. 1–12.

[23] Stefan Eilemann, Fabien Delalondre, Jon Bernard, Judit Planas, Felix Schuermann, John Biddiscombe, Costas Bekas, Alessandro Curioni, Bernard Metzler, Peter Kaltstein, et al. 2016. Key/value-enabled flash memory for complex scientific workflows with on-line analysis and visualization. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. Ieee, 608–617.

[24] Simon Eismann, Joel Scheuner, Erwin van Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina L Abad, and Alexandru Iosup. 2020. Serverless Applications: Why, When, and How? *IEEE Software* 38, 1 (2020), 32–39.

[25] Pradeep Fernando, Ada Gavrilovska, Sudarsun Kannan, and Greg Eisenhauer. 2018. Nvstream: Accelerating hpc workflows with nvram-based transport for streaming objects. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*. 231–242.

[26] W Freudling, M Romaniello, DM Bramich, P Ballester, V Forchi, CE García-Dabló, S Moehler, and MJ Neeser. 2013. Automated data reduction workflows for astronomy-The ESO Reflex environment. *Astronomy & Astrophysics* 559 (2013), A96.

[27] Richard Gerber, William Allcock, Chris Beggio, Stuart Campbell, Andrew Cherry, Shreyas Cholia, Eli Dart, Clay England, Tim Fahey, Fernanda Foertter, et al. 2014. DOE High Performance Computing Operational Review (HPCOR): Enabling Data-Driven Scientific Discovery at

HPC Facilities. (2014).

[28] Wolfgang Gerlach, Wei Tang, Kevin Keegan, Travis Harrison, Andreas Wilke, Jared Bischof, Mark DSouza, Scott Devoid, Daniel Murphy-Olson, Narayan Desai, et al. 2014. Skyport-container-based execution environment management for multi-cloud scientific workflows. In *2014 5th International Workshop on Data-Intensive Computing in the Clouds*. IEEE, 25–32.

[29] Samik Ghosh, Yukiko Matsuoka, Yoshiyuki Asai, Kun-Yi Hsin, and Hiroaki Kitano. 2011. Software for systems biology: from tools to integrated platforms. *Nature Reviews Genetics* 12, 12 (2011), 821–832.

[30] Devarshi Ghoshal and Lavanya Ramakrishnan. 2017. Madats: Managing data on tiered storage for scientific workflows. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*. 41–52.

[31] Jeremy Goecks, Anton Nekrutenko, and James Taylor. 2010. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome biology* 11, 8 (2010), 1–13.

[32] N. Hazekamp, N. Kremer-Herman, B. Tovar, H. Meng, O. Choudhury, S. Emrich, and D. Thain. 2018. Combining Static and Dynamic Storage Management for Data Intensive Scientific Workflows. *IEEE Transactions on Parallel and Distributed Systems* 29, 2 (2018), 338–350.

[33] Nicholas Hazekamp, Joseph Sarro, Olivia Choudhury, Sandra Gesing, Scott Emrich, and Douglas Thain. 2015. Scaling up bioinformatics workflows with dynamic job expansion: A case study using galaxy and makeflow. In *2015 IEEE 11th International Conference on e-Science*. IEEE, 332–341.

[34] Joseph M Hellerstein, Jose Faleiro, Joseph E Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and Chenggang Wu. 2018. Serverless computing: One step forward, two steps back. *arXiv preprint arXiv:1812.03651* (2018).

[35] Valerie Hendrix, James Fox, Devarshi Ghoshal, and Lavanya Ramakrishnan. 2016. Tigres workflow library: Supporting scientific pipelines on hpc systems. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 146–155.

[36] Muhammad H Hilman, Maria A Rodriguez, and Rajkumar Buyya. 2020. Multiple workflows scheduling in multi-tenant distributed systems: A taxonomy and future directions. *ACM Computing Surveys (CSUR)* 53, 1 (2020), 1–39.

[37] Christina Hoffa, Gaurang Mehta, Tim Freeman, Ewa Deelman, Kate Keahey, Bruce Berriman, and John Good. 2008. On the use of cloud computing for scientific workflows. In *2008 IEEE fourth international conference on eScience*. IEEE, 640–645.

[38] Zahaf Houssam-Eddine, Nicola Capodieci, Roberto Cavicchioli, Giuseppe Lipari, and Marko Bertogna. 2020. The HPC-DAG task model for heterogeneous real-time systems. *IEEE Trans. Comput.* (2020).

[39] Zhuoyi Huang, Jin Yu, and Fuli Yu. 2013. Cloud processing of 1000 genomes sequencing data using Amazon Web Service. In *2013 IEEE Global Conference on Signal and Information Processing*. IEEE, 49–52.

[40] Qingye Jiang, Young Choon Lee, and Albert Y Zomaya. 2015. Executing large scale scientific workflow ensembles in public clouds. In *2015 44th International Conference on Parallel Processing*. IEEE, 520–529.

[41] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Yadwadkar, et al. 2019. Cloud programming simplified: A berkeley view on serverless computing. *arXiv preprint arXiv:1902.03383* (2019).

[42] Gideon Juve and Ewa Deelman. 2011. Wrangler: Virtual cluster provisioning for the cloud. In *Proceedings of the 20th international symposium on High performance distributed computing*. 277–278.

[43] Gideon Juve, Ewa Deelman, Karan Vahi, Gaurang Mehta, Bruce Berriman, Benjamin P Berman, and Phil Maechling. 2009. Scientific workflow applications on Amazon EC2. In *2009 5th IEEE international conference on e-science workshops*. IEEE, 59–66.

[44] Johannes Köster and Sven Rahmann. 2012. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics* 28, 19 (2012), 2520–2522.

[45] Aleksandr Kuntsevich, Pezhman Nasirifard, and Hans-Arno Jacobsen. 2018. A distributed analysis and benchmarking framework for apache openwhisk serverless platform. In *Proceedings of the 19th International Middleware Conference (Posters)*. 3–4.

[46] Young Choon Lee and Albert Y Zomaya. 2013. Stretch out and compact: Workflow scheduling with resource abundance. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. IEEE, 219–226.

[47] Weiling Li, Yunni Xia, Mengchu Zhou, Xiaoning Sun, and Qingsheng Zhu. 2018. Fluctuation-aware and predictive workflow scheduling in cost-effective infrastructure-as-a-service clouds. *IEEE Access* 6 (2018), 61488–61502.

[48] Cui Lin, Shiyong Lu, Xubo Fei, Artem Chebotko, Darshan Pai, Zhaoqiang Lai, Farshad Fotouhi, and Jing Hua. 2009. A reference architecture for scientific workflow management systems and the VIEW SOA solution. *IEEE Transactions on Services Computing* 2, 1 (2009), 79–92.

[49] Xiangyu Lin and Chase Qishi Wu. 2013. On scientific workflow scheduling in clouds under budget constraint. In *2013 42nd International Conference on Parallel Processing*. IEEE, 90–99.

[50] Eric Lyons, Michael Zink, Anirban Mandal, Cong Wang, Paul Ruth, Chandrasekar Radhakrishnan, George Papadimitriou, Ewa Deelman, Komal Thareja, and Ivan Rodero. 2020. DyNamo: Scalable Weather Workflow Processing in the Academic Multicloud. In *100th American Meteorological Society Annual Meeting*. AMS.

[51] Ketan Maheshwari, Eun-Sung Jung, Jiayuan Meng, Venkatram Vishwanath, and Rajkumar Kettimuthu. 2014. Improving multisite workflow performance using model-based scheduling. In *2014 43rd International Conference on Parallel Processing*. IEEE, 131–140.

[52] Maciej Malawski, Gideon Juve, Ewa Deelman, and Jarek Nabrzyski. 2015. Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds. *Future Generation Computer Systems* 48 (2015), 1–18.

[53] Ming Mao and Marty Humphrey. 2011. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *SC'11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–12.

[54] Ming Mao and Marty Humphrey. 2012. A performance study on the vm startup time in the cloud. In *2012 IEEE Fifth International Conference on Cloud Computing*. IEEE, 423–430.

[55] Paul Marshall, Kate Keahey, and Tim Freeman. 2011. Improving utilization of infrastructure clouds. In *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 205–214.

[56] Marta Mattoso, Kary Ocana, Felipe Horta, Jonas Dias, Eduardo Ogasawara, Vitor Silva, Daniel de Oliveira, Flavio Costa, and Igor Araújo. 2013. User-steering of HPC workflows: state-of-the-art and future directions. In *Proceedings of the 2nd ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*. 1–6.

[57] Aaron McKenna, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis, Andrew Kernytsky, Kiran Garimella, David Altshuler, Stacey Gabriel, Mark Daly, et al. 2010. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome research* 20, 9 (2010), 1297–1303.

[58] Anton Nekrutenko and James Taylor. 2012. Next-generation sequencing data interpretation: enhancing reproducibility and accessibility. *Nature Reviews Genetics* 13, 9 (2012), 667–672.

[59] Panayiotis Neophytou, Roxana Gheorghiu, Rebecca Hachey, Timothy Luciani, Di Bao, Alexandros Labrinidis, Elisabeta G Marai, and Panos K Chrysanthis. 2012. Astroshelf: understanding the universe through scalable navigation of a galaxy of annotations. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. 713–716.

[60] Marco AS Netto, Rodrigo N Calheiros, Eduardo R Rodrigues, Renato LF Cunha, and Rajkumar Buyya. 2018. HPC cloud for scientific and business applications: taxonomy, vision, and research challenges. *ACM Computing Surveys (CSUR)* 51, 1 (2018), 1–29.

[61] Kary Ocaña, Silvia Benza, Daniel De Oliveira, Jonas Dias, and Marta Mattoso. 2014. Exploring large scale receptor-ligand pairs in molecular docking workflows in HPC clouds. In *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*. IEEE, 536–545.

[62] Gagandeep Panwar, Da Zhang, Yihan Pang, Mai Dahshan, Nathan DeBardeleben, Binoy Ravindran, and Xun Jian. 2019. Quantifying memory underutilization in hpc systems and using it to improve performance via architecture support. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 821–835.

[63] Ioan Raicu, Ian T Foster, and Yong Zhao. 2008. Many-task computing for grids and supercomputers. In *2008 workshop on many-task computing on grids and supercomputers*. IEEE, 1–11.

[64] Gonzalo P Rodrigo, Erik Elmroth, Per-Olov Östberg, and Lavanya Ramakrishnan. 2017. Enabling workflow-aware scheduling on hpc systems. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*. 3–14.

[65] Eric E Schadt, Michael D Linderman, Jon Sorenson, Lawrence Lee, and Garry P Nolan. 2010. Computational solutions to large-scale data management and analysis. *Nature reviews genetics* 11, 9 (2010), 647–657.

[66] Greg Schulz. 2011. *Cloud and virtual data storage networking*. CRC Press.

[67] Khawaja S Shams, Mark W Powell, Tom M Crockett, Jeffrey S Norris, Ryan Rossi, and Tom Soderstrom. 2010. Polyphony: A workflow orchestration framework for cloud computing. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE, 606–611.

[68] Srinath Shankar and David J DeWitt. 2007. Data driven workflow planning in cluster management systems. In *Proceedings of the 16th international symposium on High performance distributed computing*. 127–136.

[69] Julian Shun. 2020. Practical parallel hypergraph algorithms. In *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 232–249.

[70] Ola Spjuth, Erik Bongcam-Rudloff, Guillermo Carrasco Hernández, Lukas Forer, Mario Giovacchini, Roman Valls Guimera, Aleksi Kallio, Eija Korpelainen, Maciej M Kańduła, Milko Krachunov, et al. 2015. Experiences with workflows for automating data-intensive bioinformatics. *Biology direct* 10, 1 (2015), 1–12.

[71] Young-Kyoon Suh, Hoon Ryu, Hangi Kim, and Kum Won Cho. 2016. EDISON: a web-based HPC simulation execution framework for large-scale scientific computing software. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 608–612.

[72] Kyle MD Sweeney and Douglas Thain. 2018. Early Experience Using Amazon Batch for Scientific Workflows. In *Proceedings of the 9th Workshop on Scientific Cloud Computing*. 1–8.

[73] Kyle MD Sweeney and Douglas Thain. 2018. Efficient integration of containers into scientific workflows. In *Proceedings of the 9th Workshop on Scientific Cloud Computing*. 1–6.

[74] Claudia Szabo, Quan Z Sheng, Trent Kroeger, Yihong Zhang, and Jian Yu. 2014. Science in the cloud: Allocation and execution of data-intensive scientific workflows. *Journal of Grid Computing* 12, 2 (2014), 245–264.

[75] Matteo Turilli, Mark Santcroos, and Shantenu Jha. 2018. A comprehensive perspective on pilot-job systems. *ACM Computing Surveys (CSUR)* 51, 2 (2018), 1–32.

[76] Amandeep Verma and Sakshi Kaushal. 2017. A hybrid multi-objective particle swarm optimization for scientific workflow scheduling. *Parallel Comput.* 62 (2017), 1–19.

[77] Laurens Versluis, Erwin Van Eyk, and Alexandru Iosup. 2018. An analysis of workflow formalisms for workflows with complex non-functional requirements. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*. 107–112.

[78] Cong Wang, George Papadimitriou, Mariam Kiran, Anirban Mandal, and Ewa Deelman. 2020. Identifying Execution Anomalies for Data Intensive Workflows Using Lightweight ML Techniques. In *2020 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–7.

[79] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2018. Peeking behind the curtains of serverless platforms. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*. 133–146.

[80] Teng Wang, Suren Byna, Bin Dong, and Houjun Tang. 2018. UniviStor: Integrated hierarchical and distributed storage for HPC. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 134–144.

[81] David Woollard, Nenad Medvidovic, Yolanda Gil, and Chris A Mattmann. 2008. Scientific software as workflows: From discovery to distribution. *IEEE software* 25, 4 (2008), 37–43.

[82] Jia Yu and Rajkumar Buyya. 2004. A novel architecture for realizing grid workflow using tuple spaces. In *Fifth IEEE/ACM International Workshop on Grid Computing*. IEEE, 119–128.

[83] Tianyi Yu, Qingyuan Liu, Dong Du, Yubin Xia, Binyu Zang, Ziqian Lu, Pingchao Yang, Chenggang Qin, and Haibo Chen. 2020. Characterizing serverless platforms with serverlessbench. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 30–44.

[84] Dong Yuan, Yun Yang, Xiao Liu, and Jinjun Chen. 2010. A cost-effective strategy for intermediate data storage in scientific cloud workflow systems. In *2010 IEEE international symposium on parallel & distributed processing (IPDPS)*. IEEE, 1–12.

[85] Shuai Zeng, Zhen Lyu, Siva Ratna Kumari Narisetti, Dong Xu, and Trupti Joshi. 2018. Knowledge Base Commons (KBCommons) v1. 0: A multi OMICS'web-based data integration framework for biological discoveries. In *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 589–594.

[86] Qimin Zhang, Nathaniel Kremer-Herman, Benjamin Tovar, and Douglas Thain. 2018. Reduction of Workflow Resource Consumption Using a Density-based Clustering Model. In *2018 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*. IEEE, 1–9.

[87] Dongfang Zhao, Chen Shou, Tanu Maliky, and Ioan Raicu. 2013. Distributed data provenance for large-scale data-intensive computing. In *2013 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 1–8.

[88] Qi Zhao, Zhengyi Qiu, and Guoliang Jin. 2019. Semantics-aware scheduling policies for synchronization determinism. In *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*. 242–256.

[89] Yong Zhao, Youfu Li, Ioan Raicu, Shiyong Lu, Cui Lin, Yanzhe Zhang, Wenhong Tian, and Ruini Xue. 2014. A service framework for scientific workflow management in the cloud. *IEEE Transactions on Services Computing* 8, 6 (2014), 930–944.

[90] Yong Zhao, Youfu Li, Ioan Raicu, Shiyong Lu, Wenhong Tian, and Heng Liu. 2015. Enabling scalable scientific workflow management in the Cloud. *Future Generation Computer Systems* 46 (2015), 3–16.

[91] C. Zheng, B. Tovar, and D. Thain. 2017. Deploying High Throughput Scientific Workflows on Container Schedulers with Makeflow and Mesos. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. 130–139.

[92] Zhaomeng Zhu, Gongxuan Zhang, Miqing Li, and Xiaohui Liu. 2015. Evolutionary multi-objective workflow scheduling in cloud. *IEEE Transactions on parallel and distributed Systems* 27, 5 (2015), 1344–1357.

Artifact Appendix

Abstract

Mashup executes HPC/scientific workflows in a hybrid execution environment consisting of a VM-based cluster and a serverless platform. For every task in the Directed Acyclic Graph (DAG) of a workflow, Mashup decides which execution environment (VM-based cluster versus serverless platform) is suitable. Accordingly, it executes the tasks to reduce execution time and expense incurred to the end user for running the workflow. Our artifact packages the scripts and data that can be used to reproduce all the results in the paper. Additionally, it also contains the scripts to launch the benchmarks and carry out the experiments on AWS Lambda serverless and EC2 VM platforms. The artifact is available at the following link:

https://zenodo.org/record/5733395

It includes the following:

- The workflows 1000Genome, Epigenomics, and SRAsearch.
- Scripts to set up the workflows in AWS Lambda serverless environment.
- Scripts to set up the workflows in AWS EC2 VM environment.
- Scripts and data for hybrid execution of workflows using 3 different types of VM families.
- Scripts and data for hybrid execution of workflows on different VM cluster sizes ranging from 2 to 96.
- Scripts and data of profiling individual components of the workflows.

Multiple runs are performed for each of the experiments. Along with hybrid execution of HPC workflows, execution results and scripts of only EC2 cluster execution and only serverless Lambda execution are also provided.

Artifact check-list (meta-information)

- **Algorithm:** Algorithm: A hybrid serverless and VM cluster execution algorithm for HPC workflows.
- **Program:** HPC workflows 1000Genome, Epigenomics, and SRAsearch (included in the artifact).
- **Data set:** The data set required to set-up the individual tasks of the workflows are included. Data generated through experiments, containing I/O time, compute time, and execution time of workflow components in serverless and VM-based cluster are included in the artifact.
- **Run-time environment:** Python3.6 with *boto3* and *awscli*. AWS is invoked for spawning serverless functions and setting up EC2 VMs from a Ubuntu 18.04.4 LTS server.
- **Hardware:** AWS Lambda serverless platform, AWS S3, and AWS EC2 VMs of m5, r5, and r5b families (node count varying from 2 to 96).
- **Metrics:** Compute time, read time, write time of individual components of a workflow, along with the total execution time of the workflow and the expense of the end-user.
- **Output:** Execution time, compute time, I/O time, and cost.
- **Experiments:** Primarily of two different types – (1) Mashup with PDC, where in the profiling phase, Mashup decides the correct execution environment for each of the tasks of a workflow. (2) Mashup without PDC where the tasks with a large number of components are run on serverless and the rest of the tasks execute on a VM-based cluster. Along with these hybrid environments, experimental data of executing all tasks in serverless and all tasks in a VM-based cluster are also provided.
- **How much disk space required (approximately)?:** 2 GB
- **Publicly available?:** Yes
- **Archived (DOI)?:** https://zenodo.org/record/5733395

Description

This artifact provides the framework of Mashup, which executes HPC workflows in a hybrid serverless and serverful (VM-based cluster) environment. As an input, Mashup requires a workflow DAG, the binaries of the individual tasks of a workflow, and the input data of the tasks. Mashup then sets up the tasks in both serverless environment and a VM-based cluster. With the help of Mashup's PDC, it performs profiling of the tasks to determine the most optimal execution environment. This task-level optimal choice helps in the reduction of execution time of the entire workflow and the cloud execution cost incurred by the end-user.

Methodology

Mashup is implemented in Python3.6 and is easily portable to use with multiple cloud providers. As a dependency, it only requires the command line interface (CLI) of the respective cloud provider to be installed and configured with the user account credentials. Mashup sets up a VM nodes according to the user's chosen VM family and number of nodes. Then, Mashup sets up the tasks of the workflow in the cluster as well as in the cloud provider's serverless platform. It then configures a remote storage to be used for exchanging data among multiple tasks running on different platforms and loads the initial input data in it. The Placement Decision Controller (PDC) of Mashup spawns the tasks of the workflow once on the serverless platform and once on a VM-based cluster and decides the optimal execution environment for each of the tasks. Then accordingly, Mashup spawns the tasks of the workflow in the most optimal execution environment, following the precedence pattern of the workflow DAG. Mashup also takes certain steps like having redundant remote storages and VMs which can control the execution of the workflow, in the event of a failure of the master node.

Installation

To set up the workflows and the framework of Mashup, *boto3* must be installed and *awscli* should be configured with the user's AWS account credentials. This will allow the user to set up and directly export the tasks of a workflow to the EC2 VMs and the serverless deployment packets to AWS via the *aws lambda create-function* command. More details on installation is provided in the *README.md* file in the artifact.

Evaluation and expected results

All results from Fig.6 – Fig.12 are expected to be reproduced from the data in the artifact.