Serverless Science for Simple, Scalable, and Shareable Scholarship

Kyle Chard and Ian Foster
Department of Computer Science, University of Chicago, Chicago, IL 60615, USA
Data Science and Learning Division, Argonne National Laboratory, Lemont, IL 60439, USA
Email: {chard, foster}@uchicago.edu

***Abstract*—The adoption of computation- and data-intensive science, or eScience, makes research progress increasingly dependent on the availability, management, and use of sophisticated cyberinfrastructure. An unfortunate consequence is that researchers face increasingly burdensome demands for managing and maintaining cyberinfrastructure. The advent of virtualization and cloud computing has helped, by allowing outsourcing of some such tasks to reliable and scalable cloud providers. But much more progress is needed before we can create a research cyberinfrastructure that allows researchers to focus on creative thought rather than systems management. We examine here how the emerging paradigm of serverless computing, in which arbitrary functions can be dispatched seamlessly to scalable, secure, and reliable service providers, can move us in that direction. To demonstrate how serverless computing can transform scientific computing, we describe three serverless computing models: service-oriented computing, research automation, and function as a service, presenting illustrative case studies for each.**

***Index Terms*—Serverless science, serverless computing, function as a service, Globus**

I. Introduction

As bandwidth increases and wireless communication protocols become yet faster and more ubiquitous, there is a rapid blurring between local, remote, and mobile computing. As George Gilder observed in 2000, "When the network is as fast as the computer's internal links, the machine disintegrates across the net into a set of special-purpose appliances" [1]. As transport costs are reduced, it becomes increasingly attractive to build appliances that are specialized for one function, for which they can exploit economies of scale to reduce costs or energy consumption, or to increase speed and reliability.

The transformation envisaged by Gilder has underpinned the adoption of cloud computing—a form of remote computing in which a cloud provider operates resources on behalf of users. Researchers can now access enormous computing capacity at the click of a button, without regard to where that computing capacity is located or on what physical resources it is delivered. Cloud computing has enabled many new applications, reduced time to delivery, and improved reliability and performance, often at a fraction of the cost of local deployments.

In 2005, we presented a vision for *service-oriented science* [2]: a model in which scientific data and capabilities are outsourced and abstracted behind internet-accessible services, in many ways moving towards the specialized appliances envisaged by Gilder. Over the past decade, we have seen adoption of service-oriented science and an evolution in research cyberinfrastructure. Scientists now exploit many specialized services in their daily work, such as collaborative document editing, web-hosted email, video conferencing tools, online lab notebooks, and research data management services. Each of these services streamlines research by eliminating the need to manage local cyberinfrastructure. Each facilitates sharing of data and effort. Yet many opportunities remain to further disintegrate scientific computing infrastructure in ways that simplify additional research processes, enable scaling in terms of the number of activities that can be performed automatically, and reduce barriers to sharing of research artifacts. For example, at many experimental facilities around the world, data collection and analysis processes engage only underpowered local computers and are managed manually in ways that preclude sharing and scaling. We show here how such challenges can be overcome by allowing management and analysis tasks to be dispatched to remote providers.

The advent of serverless computing provides a potential path towards breaking down remaining barriers to effective automation and outsourcing. In serverless computing the cloud provider operates the physical servers as well as the virtual resources, such as the virtual machines, containers, operating systems, and software stacks. The best-known serverless model is function-as-a-service (FaaS) as offered by Amazon Lambda [3], Google Cloud Functions [4], and Azure Functions [5]. FaaS allows users to define functions in a high level programming language. They, or others, may then invoke that function one or more times relying on the cloud provider to elastically provision infrastructure. While FaaS is the most common serverless model, and may be viewed as the foundation on which other serverless offerings are built, we adopt the definition proposed by researchers at the University of California, Berkeley: a serverless service must "scale automatically with no need for explicit provisioning, and be billed based on usage" [6].

We present here a vision for serverless science. We focus specifically on three main areas that exemplify the application of serverless computing in science: 1) the doption of service-oriented science as illustrated by Globus for research data management; 2) the ability to automate multi-step research processes that comprise both human and machine activities and span locations and timescales; and 3) the adoption of FaaS models that enable execution of computational workloads comprised of functions on distributed research infrastructure.

II. Software as a Service

The increase in network bandwidth and availability of low-cost, elastic, and reliable infrastructure as a service (IaaS) platforms has underpinned the adoption of software as a service (SaaS). Both in industry and research, there is an evolving ecosystem of services that are accessible via well-defined web interfaces. In science, such services, including Globus [7], have quickly become crucial tools for researchers. The reason for such rapid adoption is that SaaS is more than simply a mode of accessing scientific capabilities it is also a business model in which an available, scalable, and reliable software appliance is leased, rather than purchased, by users.

The SaaS-approach has proven to be particularly advantageous in science as it provides benefits to both consumers and providers of services. For example, consumers can access powerful, scalable, and reliable capabilities through well-defined interfaces without needing to deploy or manage software themselves; providers benefit from supporting only a single copy of the software and from the economies of scale possible. Importantly, costs can be low, as only a single instance of the software is operated by the provider and the increased cost by adding a new user is typically small.

A. State of the art: Globus

To illustrate the benefits of SaaS in science we briefly describe our experiences developing and operating Globus [7]. Having developed and supported the Globus toolkit via a traditional software distribution model for more than a decade, we adopted the SaaS model in 2011 with the release of what we initially called Globus Online and now is referred to simply as Globus. Globus initially provided data transfer services. It now provides a range of research data management capabilities including data sharing, discovery, automation, authentication, and groups management. In the nearly 10 years since Globus was created, adoption has been compelling, as summarized in Figure 1. We briefly describe two popular Globus capabilities: data management and identity and access management.

1) Data management: Globus Transfer [8] facilitates reliable management, transfer, synchronization, and sharing of data irrespective of where it is stored. Operated as a cloud-hosted service and with a client-side agent, called Globus Connect, Globus provides access to more than 20 000 storage endpoints around the world. The Globus Connect software can be deployed as an interface to a range of storage backends, from laptops and parallel file systems through to tape and cloud storage. Users can, via the Globus service, access data directly using HTTP or they can transfer it between storage systems using high performance third-party GridFTP transfers. The Globus service manages the secure and reliable movement of data. It reduces the complexity of accessing and moving data including authenticating with each endpoint, tuning transfer parameters to optimize performance, monitoring the transfer to ensure that errors are identified and resolved, and finally validating the integrity of the transferred data.

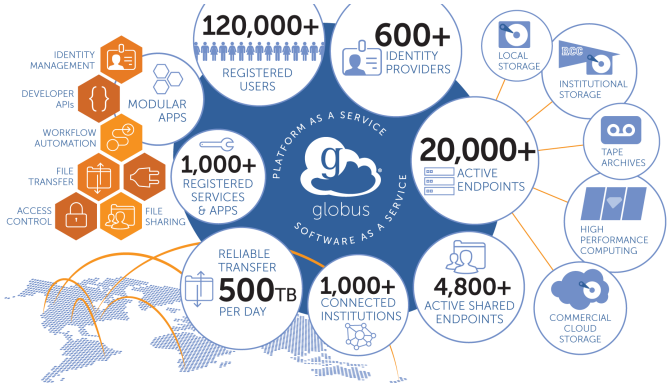

Fig. 1. A perspective on the Globus universe. From left to right: major elements of the platform, relevant statistics, and supported storage systems.

2) Identity and access management: Globus Auth [9] provides an authentication and authorization platform to more than 1000 unique Globus and third party applications and services. It solves several key challenges for scientific computing most notably providing a standards-compliant trust fabric that pervades individual computing centers and resources. It simplifies the task of integrating with external identity providers (e.g., Google) and identity federations (e.g., InCommon and EduGain) for authentication with apps and services; it implements a federated authorization model in which users may combine various identities into a set which can be used to make authorization decisions based on the collective set of permissions granted to those identities; it provides a general and extensible security model for REST APIs, enabling services to implement strong, standard OAuth 2 security; it implements a user-managed and auditable model for delegating access to services and applications; and it securely stores short-term tokens for accessing cyberinfrastructure services.

B. The future of services in science

We, and many others, have demonstrated not only the technical feasibility of SaaS for science, but also the unique benefits and economic advantages of the model. Over the last decade there has been a general movement towards service oriented methods as a mode of delivering scientific capabilities to researchers. From services designed for sharing data and specialized capabilities [10, 11] through to services that offer lower-level capabilities for managing data and computations [12]–[14]. As a result there is a growing ecosystem of service-based capabilities available to researchers.

As the number of scientific services increases, there is inevitable duplication of capabilities across services. For example, most services manage their own users, groups, and data. In industry, these shared requirements gave rise to platform as a service, in which middleware-like services are provided to satisfy these shared needs so that different service developers need not implement and manage equivalent capabilities. Similar platforms for science seem likely to be needed to reduce the cost and complexity of creating and managing scientific services. Some platform-like building blocks are

already available to scientific service developers, such as the identity and access management services provided by Globus. It is crucial for the future of SaaS in science that these services evolve into a cohesive and interoperable ecosystem. The adoption of standard web interfaces and protocols, such as OAuth 2, is important for simplifying interoperability.

The development and support of any software requires funding, whether via volunteer effort, grants, or direct payments. Service-based approaches can reduce, but do not eliminate, funding needs. Thus, a major obstacle to sustainability is the mechanics of collecting subscription or pay-per-use revenue, negotiating contracts, accepting payments, marketing new capabilities, etc. Such capabilities are unlikely to be provided by a single group, instead a general marketplace seems desirable, to serve as a single point of contact for users, developers, institutions, projects, and funding agencies wanting to use scientific services. Core marketplace capabilities must include directory/information services to keep track of available offerings; payment models that service providers can leverage; and a support system for addressing the needs of users.

III. Automation and event-based computing

The increasing volume and variety of data, coupled with new specialized (and distributed) cyberinfrastructure and various scientific services, has led to increasingly complex research processes. It is now less common for research to be conducted by a single researcher working with just their own data on a laptop. Instead, research requires teams of collaborators, data obtained from diverse and distributed sources, and analyses conducted on a heterogeneous set of computers. Furthermore, research processes are not reliant solely on data and computational resources, they often require varying degrees of human participation to, for example, label data for training, check images for errors, and provide metadata for discovery and publication.

Outsourcing research processes to web-based services addresses only part of this problem: it makes it simple to perform individual research activities; however it does not allow for the reliable, secure, and efficient orchestration of a diverse set of research processes that span resources, locations, and time periods. Consider, for example, an experiment to image a biological specimen using a scanning electron microscope. The researcher will repeatedly perform the same operations for each sample: check images for errors and quality, move data to a remote computer for storage, execute reconstruction and analysis algorithms on a cluster, register data and provenance information into an electronic lab notebook, and share resulting data with their colleagues. The monotony of such operations place unreasonable overhead on researchers, which in turn may lead to potential mistakes and lost productivity. If we look to the lessons learned from the introduction of the assembly line, where similar repeated activities are completely automated, it would significantly reduce the time to science and increase research productivity if we were to automate such research processes.

An effective research automation system would essentially eliminate the need for researchers to perform mundane and repeated activities. Like home automation systems, automated research processes should be scheduled or started as a result of a particular event, for example, in the example above, the acquisition of sample images, but potentially any other type of event such as publication of a new method or paper, sharing of new data, or availability of new compute resources. It should then be able to reliably orchestrate a series of steps such as data movement, analysis, and human-in-the-loop actions, requiring input from users only when strictly necessary.

A. State of the art: Globus Automate

To explore these ideas we are developing Globus Automate [15], a platform that aims to (1) make it easy to create robust, secure distributed research automation flows, comprising both automated and human-in-the-loop actions, that can be automatically triggered by events; and (2) allow anybody to integrate additional services so that automation flows can include their actions and events. We look towards the success of online automation services, such as if-this-then-that (https://ifttt.com/), to inspire our design with the aim to make it easy for researchers to define and manage research flows. While such systems provide an example of what is possible, they are not currently designed for the scale or complexity of scientific use cases. For example, research requires multistep automation flows, rather than just simple trigger-action behaviors; asynchronous, long-lived actions, in addition to synchronous tasks; and robust end-to-end security that integrates with existing research cyberinfrastructure.

Globus Automate relies on a simple, declarative, JSON-based, state machine language for defining flows, based on the Amazon States Language [16]. This language allows for concise definition of flows comprising multiple actions, with control logic ranging from simple sequential actions to complex branching and iteration, with simple but powerful timeout, retry, and recovery capabilities. As the system evolves we aim to develop a simple web-based interface to support creation and management of automation flows without requiring significant technical expertise.

An automation flow consists of a trigger (associated with an event source or schedule) followed by one or more steps that invoke pre-defined actions. We define a simple REST interface by which external services can be integrated with Globus Automate to generate triggers or perform actions. Services implementing these interfaces may be registered with Globus Automate and integrated into flows by any user, subject to service access rights. Globus Automate constructs Amazon Lambda functions for each step. The Lambda function is responsible for calling the appropriate external service and, in the case of asynchronous invocation, monitoring state and retrieving results. Globus Automate flows are encoded into a series of Amazon Step Functions [17] using shared state to pass input/output between steps alongside necessary access tokens to securely execute remote actions.

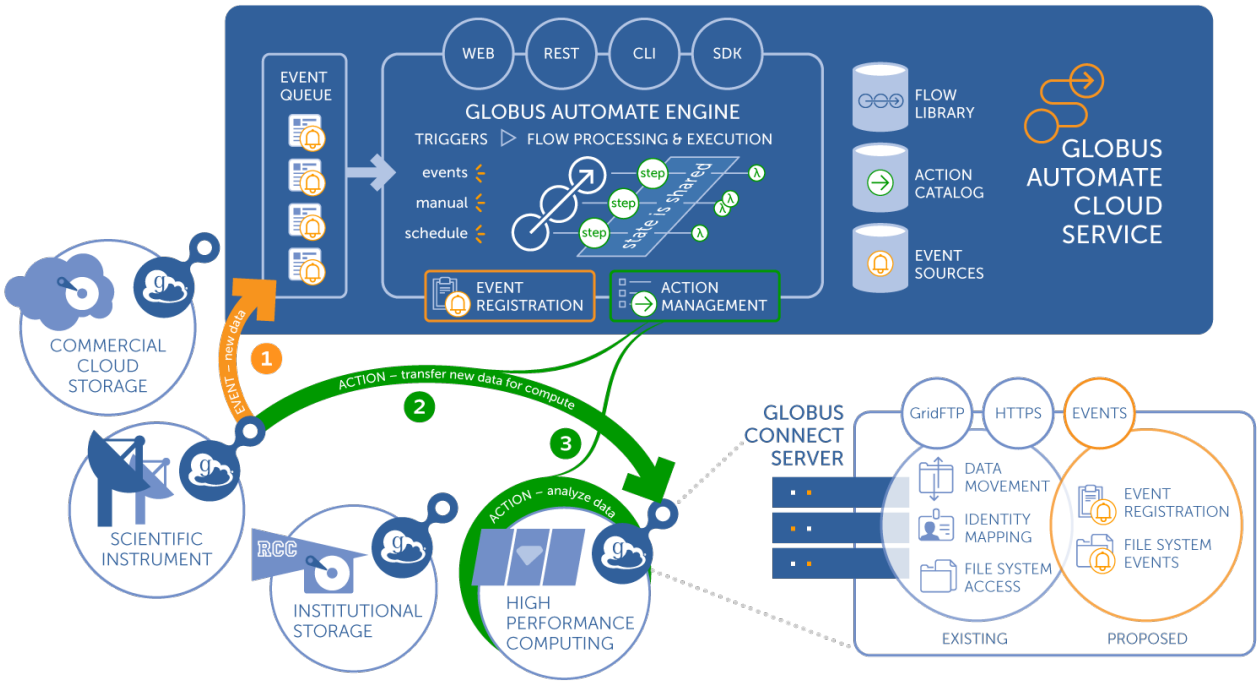

Fig. 2. Globus Automate, showing the cloud-hosted automation engine and an example application in which (1) generation of data at a scientific instrument produces an event that is received by Automate; (2) the event triggers a data transfer action to move the new data to a HPC system; and (3) delivery of data to the HPC system triggers a computational data analysis task. The lower right shows the internals of the Globus Connect Server enhanced with event detection capabilities.

B. *The future of automation in science*

While scientific workflow systems, such as Pegasus [18] and Swift [19], have a long history of representing and orchestrating computational processes, our focus here is on a related, but tangential problem, of securely and reliably automating, for many thousands of scientists, sequences of tasks that may span locations, storage systems, administrative domains, and timescales, and integrating both computational and human inputs.

Several key challenges must be solved before such automated approaches can be relied upon by scientists. First, there is a need to develop new mechanisms for capturing events from various file systems and other sources. Systems such as Ripple [20] illustrate how events can be captured from large-scale parallel file systems. Second, home automation systems and online automation services have been widely adopted not because they can represent complex automation rules, but due to their rich ecosystem of connected services. A similar set of automation-compliant research services is necessary in order to support the rich automation workflows required by researchers. Finally, the complexity of automating activities is primarily due to the need to provide reliability and fault-tolerance, it is therefore crucial that automation systems are able to overcome problems, such as when resources are offline, data transfers are delayed due to networking errors, or researchers are unavailable to perform their task, without requiring significant human intervention.

IV. Function as a Service

FaaS removes the need to manage physical and virtual cyberinfrastructure allowing users to define and execute high-level programming functions. Users simply register a function ($f(x)$), invoke registered functions ($y = f(x), y' = f(x')$), and wait for results ($[y, y']$). Adoption in industry has been overwhelming. Large companies such as Netflix and Thomson Reuters use FaaS to implement rule-based self-managing infrastructure and analytics systems capable of responding to thousands of events per second.

There are significant opportunities for similar abstractions in scientific computing. Researchers are frequently overwhelmed by the need to understand the complexities of research cyberinfrastructure such as the esoteric batch queuing systems, vanguard authentication models, customized software modules, inflexible HPC allocation policies, and complexity of efficiently mapping long-tail workloads across provisioned nodes. As scientific workloads become increasingly granular, for example as a result of machine learning, event-based computing, and a desire to decompose monolithic code into smaller computational components, these challenges are likely to become more significant.

The scientific ecosystem brings with it unique challenges. For example, unlike the near-homogeneous environments offered by cloud data centers, the computational resources available to researchers span administrative domains and thus come with unique authentication and provisioning ecosystems

(e.g., with diverse interfaces and container technology). The functions to be executed may be long running, rely on complex software environments, and require access to large and distributed data. Scientific instruments may also pose particularly strict quality of service requirements. Finally, while cloud providers operate using standard currencies, the allocation models used in science are often based on coarse virtual credits granted to researchers.

A. State of the art: funcX

We are developing funcX [21] a distributed FaaS system for science. Unlike existing FaaS solutions, which assume deployment in homogeneous data centers or rely on container orchestration systems, funcX aims to federate a loosely coupled set of function serving *endpoints* deployed on globally distributed and heterogeneous computing systems, from clouds to clusters and laptops. funcX implements a SaaS-based interface via which endpoints are registered and access is shared with permitted users. The funcX service allows users to define and register functions centrally and to subsequently execute those functions on remote endpoints to which users are authorized to access.

When registering a function, funcX inspects the function dependencies (e.g., system and Python packages) and dynamically constructs a Docker container using repo2docker [22]. When the function is executed on a remote endpoint, funcX manages the deployment of that container onto a compute node, Kubernetes pod, cloud instance, or local container for execution. Depending on the deployment environment funcX will make use of the available provisioning interface (e.g., Cloud API or batch scheduler) to acquire compute resources for execution. On clusters, for example, funcX uses a pilot job model by submitting a request for a set of nodes from the batch scheduler and deploying specialized funcX manager software on those nodes. When the nodes are acquired the funcX manager will register with the endpoint and receive function requests for execution. Each function is executed in a container on the node. The manager is responsible for ensuring that the correct container is instantiated for each function and for removing unused containers.

B. Parallel programming with FaaS

While FaaS models represent an intuitive and powerful model for executing code, they are not inherently designed for implementing complex applications. Most often, scientific applications are assembled by gluing together a diverse set of software components. While the components themselves may be written in various languages (e.g., Fortran, C/C++, and) the orchestration layer is typically written in higher level scripting languages such as Python. This approach has obvious benefits to users: it is easy to create sophisticated programs, it avoids the need to re-write existing components, and it makes it easy to understand the program. Furthermore, it can be easily mapped to parallel and distributed computers as the individual components can be executed concurrently.

Parsl [23] is one example of a parallel programming model designed to enable creation of parallel scientific programs comprised of functions. Parsl is a Python library that augments Python by introducing decorators that can be used to wrap both Python and external components. Functions that are wrapped with the decorator (called Apps) indicate to Parsl an opportunity for parallelism as well as explicitly stating input and output semantics. When the program is executed Parsl forms a dependency graph with nodes representing App invocations (i.e., tasks) and edges representing the dependencies established between input and output data. When all task dependencies are met, Parsl sends tasks for execution on connected resources. Depending on the connected resources, these tasks may be executed concurrently such as when using multiple threads on a single machine or multiple nodes on a cluster. When calling an App, Parsl will return a *future* in lieu of the App result and thus the program can continue executing until reaching a barrier. When the task completes, Parsl fills in the result of the future and marks dependent nodes in the dependency graph as ready to execute.

The advantage of the Parsl model is that it offers a simple function-based model for executing tasks—each App is represented by a function that can be executed remotely. It is also powerful as it allows developers to implement other functionality in native Python code. Thus it allows developers to specify the logic of the program without regard for the specific mechanism by which the components are executed. This approach is unlike other common techniques, such as scientific workflow systems and domain specific languages, that require users to learn new syntax and may limit the set of capabilities exposed to developers.

While Parsl provides an easy to use model for writing parallel programs, it relies on a modular runtime layer on which to execute these functions. At a high level, the Parsl library assembles a queue of tasks for execution on connected resources. We aim in the near future to support the use of funcX for execution, but at present Parsl can execute tasks on clouds, clusters, and supercomputers using a range of executors, including external executors such as IPyParallel [24] and WorkQueue [25] as well as three specialized Parsl executors: High Throughput Executor (HTEX), Extreme Scale Executor (EXEX), and Low Latency Executor (LLEX).

C. The future of FaaS in science

As scientific workloads become more granular, interactive, and event-based, FaaS-based execution models will become more attractive due to the potential to improve efficiency and reduce costs [26]. There are however several challenges that must be addressed before FaaS can be adopted generally in science such as the need to support distributed and remote execution, support heterogeneous architectures and accelerators, and integrate with data.

As container technology becomes more pervasive, and container orchestration systems become readily deployed in research environments, it will become less difficult to move scientific applications between locations. However container

technology is not yet able to completely abstract different execution environments. This is especially the case in HPC environments where, for example, different MPI libraries are optimized for different systems. Other challenges include the lack of a common container implementation, heterogeneous container orchestration interfaces, and a lack of interoperability between different container types.

As FaaS-based models become more popular and the ecosystem of execution endpoints develops, there is likely to be inefficiencies with respect to how functions are mapped to heterogeneous infrastructure. We expect that there will be opportunities to develop new scheduling algorithms that operate at a fine-grain level, take into account a fluid environment of local, cloud, and edge environments, and exploit tradeoffs among execution time, latency, data locality, and other analysis-specific criteria.

V. Related work

Scientific gateways have long embraced web-based models for providing access to scientific capabilities [27]. For example, CyberGIS [10] and CIPRES [11] provide environments for geospatial analyses and for inference of phylogenetic trees, respectively. As gateways have proliferated, general-purpose frameworks have emerged, including Agave [12], Apache Airavata [13], and HubZero [14]. These frameworks provide broad functionality, from data management to job execution, and allow developers to deploy gateways without needing to re-implement such functionality. However, unlike the SaaS model, in which a single service is operated for everyone, these frameworks are offered as deployable services that are deployed for each gateway.

Automation and event-based computing is most common in home automation scenarios. In science, a number of workflow systems, such as Galaxy [28], Pegasus [18], Swift [19], and Taverna [29], have been developed to enable reliable and reproducible execution of scientific applications, although they do not support the same type level of automation: typically they do not support trigger-based execution, long-running asynchronous workflows, or human-in-the-loop activities.

Cloud-hosted FaaS systems such as Amazon Lambda [3], Google Cloud Functions [4], and Azure Functions [5] are widely used in industry. However, these services are tightly integrated with their respective cloud platforms and cannot be deployed on external cyberinfrastructure. Hybrid cloud-edge models, such as Amazon Greengrass [30], enable functions to be deployed on edge devices, but are designed for small-scale, single device execution. Open FaaS solutions, including OpenWhisk [31], Fn [32], Kubeless [33], Open-FaaS [34], and Abaco [35], can be installed locally; however, all rely on container orchestration infrastructure (e.g., Kubernetes) and cannot be deployed on the broad range of scientific cyberinfrastructure.

Various function-based parallel programming libraries have been developed to support intuitive implementation of scientific applications. PyWren [36] provides a simple Python programming model for executing stateless Python functions on Amazon Lambda. It has been shown to scale to thousands of concurrent functions. Other models such as Dask [37] provide transparent parallelization of existing Python libraries, such as Pandas dataframes. Thus, developers can convert non-parallel programs into parallel programs by replacing calls to existing libraries. Researchers are increasingly exploring even more granular parallel execution in models such as those implemented by Swift/T [38] and Ray [39]. Both implement a decentralized execution model capable of scaling to support execution of millions of short duration tasks.

VI. Summary

Cloud platforms provide a reliable, scalable, and often cost-effective basis for outsourcing computational and data-intensive science. Part of the success of cloud computing in science has been the fact that researchers can access enormous and elastic computing capacity without needing to manage low-level cyberinfrastructure. Thus enabling them to instead focus on higher level science objectives. Serverless computing promises a similarly transformative impact, enabling abstraction of virtual as well as physical resources, and ushering in a new era of serverless science. Adoption of serverless approaches will reduce burdens on researchers by enabling the automation and outsourcing of various scientific tasks to an even greater degree than is possible today.

We have described how serverless computing models can be adapted to scientific scenarios by embracing the distributed nature of research cyberinfrastructure. We focused on three examples of serverless computing in science. *Service-oriented science*, as exemplified by Globus and other scientific services, enables researchers to perform a range of activities through standard APIs and web browsers. *Research automation*, illustrated here by Globus Automate, allows researchers to specify complex research processes by linking both human and computer actions. *Function as a service* execution models simplify task execution on distributed and heterogeneous infrastructure through a simple programming abstraction. We argue that serverless computing models are central to the future of eScience and that they will ultimately reduce overheads on researchers, enable more effective use of research cyberinfrastructure, decrease the cost of compute- and data-intensive science, and increase the quantity and quality of research.

Acknowledgements

This work was supported in part by Laboratory Directed Research and Development funding from Argonne National Laboratory under US Department of Energy under Contract DE-AC02-06CH11357 and US National Science Foundation under awards OAC-1550588 and OAC-1835890. We thank the Globus and Parsl teams for developing the software described in this paper and for the many discussions that have contributed to this work. We specifically thank Rachana Ananthakrishnan, Yadu Babuji, Ben Blaiszik, Ryan Chard, Dan Katz, Zhuozhao Li, Jim Pryune, Tyler Skluzacek Steve Tuecke, Mike Wilde, and Anna Woodard.

REFERENCES

[1] G. Gilder, *Gilder Technology Report*, June 2000.

[2] I. Foster, "Service-oriented science," *Science*, vol. 308, no. 5723, pp. 814–817, 2005.

[3] "Amazon Lambda," https://aws.amazon.com/lambda. Accessed August 30, 2019.

[4] Google Cloud Functions. https://cloud.google.com/functions/. Accessed August 30, 2019.

[5] "Azure Functions," https://azure.microsoft.com/en-us/services/functions/. Visited August 30, 2019.

[6] E. Jonas, J. Schleier-Smith, V. Sreekanti, C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. J. Yadwadkar, J. E. Gonzalez, R. A. Popa, I. Stoica, and D. A. Patterson, "Cloud programming simplified: A Berkeley view on serverless computing," *CoRR*, vol. abs/1902.03383, 2019.

[7] K. Chard, S. Tuecke, and I. Foster, "Efficient and secure transfer, synchronization, and sharing of big data," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 46–55, Sep. 2014.

[8] B. Allen, J. Bresnahan, L. Childers, I. Foster, G. Kandaswamy, R. Kettimuthu, J. Kordas, M. Link, S. Martin, K. Pickett, and S. Tuecke, "Software as a service for data scientists," *Commun. ACM*, vol. 55, no. 2, pp. 81–88, Feb. 2012.

[9] S. Tuecke, R. Ananthakrishnan, K. Chard, M. Lidman, B. McCollam, S. Rosen, and I. Foster, "Globus Auth: A research identity and access management platform," in *12th IEEE International Conference on e-Science (e-Science)*, Oct 2016, pp. 203–212.

[10] S. Wang, "A CyberGIS framework for the synthesis of cyberinfrastructure, GIS, and spatial analysis," *Annals of the Association of American Geographers*, vol. 100, no. 3, pp. 535–557, 2010.

[11] M. A. Miller, W. Pfeiffer, and T. Schwartz, "Creating the CIPRES science gateway for inference of large phylogenetic trees," in *Gateway Computing Environments Workshop (GCE)*, Nov 2010, pp. 1–8.

[12] R. Dooley, S. R. Brandt, and J. Fonner, "The Agave platform: An open, science-as-a-service platform for digital science," in *Practice and Experience on Advanced Research Computing*. ACM, 2018, p. 28.

[13] S. Marru, L. Gunathilake, C. Herath, P. Tangchaisin, M. Pierce, C. Mattmann, R. Singh, T. Gunarathne, E. Chinthaka, R. Gardler, A. Slominski, A. Douma, S. Perera, and S. Weerawarana, "Apache Airavata: A framework for distributed applications and computational workflows," in *ACM Workshop on Gateway Computing Environments (GCE)*. New York, NY, USA: ACM, 2011, pp. 21–28.

[14] M. McLennan and R. Kennell, "HUBzero: A platform for dissemination and collaboration in computational science and engineering," *Computing in Science Engineering*, vol. 12, no. 2, pp. 48–53, March 2010.

[15] R. Ananthakrishnan, B. Blaiszik, K. Chard, R. Chard, B. McCollam, J. Pruyne, S. Rosen, S. Tuecke, and I. Foster, "Globus platform services for data publication," in *Proceedings of the Practice and Experience on Advanced Research Computing (PEARC)*. ACM, 2018, pp. 14:1–14:7.

[16] "Amazon States Language," https://docs.aws.amazon.com/step-functions/latest/dg/concepts-amazon-states-language.html. Accessed August 30, 2019.

[17] "Amazon Step Functions," https://aws.amazon.com/step-functions. Accessed August 30, 2019.

[18] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. F. da Silva, M. Livny, and K. Wenger, "Pegasus, a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015.

[19] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster, "Swift: A language for distributed parallel scripting," *Parallel Computing*, vol. 37, no. 9, pp. 633–652, 2011.

[23] Y. Babuji, A. Woodard, Z. Li, D. S. Katz, B. Clifford, R. Kumar, L. Lacinski, R. Chard, J. M. Wozniak, I. Foster, M. Wilde, and K. Chard, "Parsl: Pervasive parallel programming in python," in *28th International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*. ACM, 2019, pp. 25–36.

[20] R. Chard, K. Chard, J. Alt, D. Y. Parkinson, S. Tuecke, and I. Foster, "Ripple: Home automation for research data management," in *37th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW)*, June 2017, pp. 389–394.

[21] R. Chard, T. J. Skluzacek, Z. Li, Y. Babuji, A. Woodard, B. Blaiszik, S. Tuecke, I. Foster, and K. Chard, "Serverless supercomputing: High performance function as a service for science," *CoRR*, vol. abs/1908.04907, 2019.

[22] repo2docker. https://repo2docker.readthedocs.io. Accessed August 30, 2019.

[24] IPyParallel. https://ipyparallel.readthedocs.io/en/latest/. Accessed August 30, 2019.

[25] P. Bui, D. Rajan, B. Abdul-Wahid, J. Izaguirre, and D. Thain, "Work Queue + Python: A framework for scalable scientific ensemble applications," in *Workshop on Python for High Performance and Scientific Computing (PyHPC) at the ACM/IEEE International Conference for High Performance Computing*, 2011.

[26] B. Varghese, P. Leitner, S. Ray, K. Chard, A. Barker, Y. Elkhatib, H. Herry, C. Hong, J. Singer, F. P. Tso, E. Yoneki, and M. Zhani, "Cloud futurology," *Computer*, vol. 52, no. 9, pp. 68–77, Sep. 2019.

[27] K. A. Lawrence, M. Zentner, N. Wilkins-Diehr, J. A. Wernert, M. Pierce, S. Marru, and S. Michael, "Science gateways today and tomorrow: positive perspectives of nearly 5000 members of the research community," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 16, pp. 4252–4268, 2015.

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

[29] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, and T. Oinn, "Taverna: A tool for building and running workflows of services," *Nucleic Acids Research*, vol. 34, pp. W729–W732, 2006.

[30] AWS Greengrass. https://aws.amazon.com/greengrass/. Accessed August 30, 2019.

[31] I. Baldini, P. Castro, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, and P. Suter, "Cloud-native, event-based programming for mobile applications," in *International Conference on Mobile Software Engineering and Systems*. ACM, 2016, pp. 287–288.

[32] Fn project. Urlhttps://fnproject.io. Accessed August 30, 2019.

[33] Kubeless. https://kubeless.io. Accessed August 30, 2019.

[34] A. Ellis, "Introducing functions as a service (Open-FaaS)," 2017, https://blog.alexellis.io/introducing-functions-as-a-service.

[35] J. Stubbs, R. Dooley, and M. Vaughn, "Containers-as-a-service via the actor model," in *11th Gateway Computing Environments Conference*, 2017.

[36] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the cloud: Distributed computing for the 99%," in *Symposium on Cloud Computing (SoCC)*. ACM, 2017, pp. 445–451.

[37] Dask. http://docs.dask.org/en/latest/. Accessed August 30, 2019.

[38] J. M. Wozniak, T. G. Armstrong, M. Wilde, D. S. Katz, E. Lusk, and I. T. Foster, "Swift/T: Large-scale application composition via distributed-memory dataflow processing," in *13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, May 2013, pp. 95–102.

[39] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul *et al.*, "Ray: A distributed framework for emerging AI applications," in *13th USENIX Conf. on Operating Systems Design and Implementation*, 2018, pp. 561–577.