Scaling Shifts Seriously Smashes Serverless Systems

Computational Spectroscopy with funcX

Eric Jonas
Assistant Professor, Department of Computer Science
Physical Sciences Division, University of Chicago
ericj@uchicago.edu | @stochastician | jonaslab.uchicago.edu

Who am I ?

Occupy the Cloud: Distributed Computing for the 99%

Eric Jonas, Qifan Pu, Shivaram Venkataraman, Ion Stoica, Benjamin Recht
University of California, Berkeley
{jonas, qifan, shivaram, istoica, brecht}@eecs.berkeley.edu

ABSTRACT

Distributed computing remains inaccessible to a large number of users, in spite of many open source platforms and extensive commercial offerings. While distributed computation frameworks have moved beyond a simple map-reduce model, many users are still left to struggle with complex cluster management and configuration tools, even for running simple embarrassingly parallel jobs. We argue that stateless functions represent a viable platform for these users, eliminating cluster management overhead, fulfilling the promise of elasticity. Furthermore, using our prototype implementation, PyWren, we show that this model is general enough to implement a number of distributed computing models, such as BSP, efficiently. Extrapolating from recent trends in network bandwidth and the advent of disaggregated storage, we suggest that stateless functions are a natural fit for data processing in future computing environments.

CCS CONCEPTS

• **Computer systems organization** → **Cloud computing**; • **Computing methodologies** → *Distributed programming languages*;

KEYWORDS

Serverless, Distributed Computing, AWS Lambda, PyWren

ACM Reference Format:
Eric Jonas, Qifan Pu, Shivaram Venkataraman, Ion Stoica, Benjamin Recht University of California, Berkeley {jonas, qifan, shivaram, istoica, brecht}@eecs.berkeley.edu. 2017. Occupy the Cloud: Distributed Computing for the 99%. In *Proceedings of SoCC '17, Santa Clara, CA, USA, September 24–27, 2017,* 7 pages.
https://doi.org/10.1145/3127479.3128601

1 INTRODUCTION

Despite a decade of availability, the twin promises of scale and elasticity [2] remain out of reach for a large number of cloud computing users. Academic and commercially-successful platforms (Apache Hadoop, Apache Spark) with tremendous corporate backing (Amazon, Microsoft, Google) still present high barriers to entry for the average data scientist or scientific computing user. In fact, taking advantage of elasticity remains challenging for even sophisticated users, as the majority of these frameworks were designed to first target on-premise installations at large scale. On commercial cloud platforms, a novice user confronts a dizzying array of potential decisions: one must ahead of time decide on instance type, cluster size, pricing model, programming model, and task granularity.

Such challenges are particularly surprising considering that the vast number of data analytic and scientific computing workloads remain embarrassingly parallel. Hyperparameter tuning for machine learning, Monte Carlo simulation for computational physics, and featurization for data science all fit well into a traditional map-reduce framework. Yet even at UC Berkeley, we have found via informal surveys that the majority of machine learning graduate students have never written a cluster computing job due to complexity of setting up cloud platforms.

In this paper we argue that a *serverless* execution model with *stateless* functions can enable radically-simpler, fundamentally elastic, and more user-friendly distributed data processing systems. In this model, we have one simple primitive: users submit functions that are executed in a remote container; the functions are stateless as all the state for the function, including input, output is accessed from shared remote storage. Surprisingly, we find that the performance degradation from using such an approach is negligible for many workloads and thus, our simple primitive is in fact general enough to implement a number of higher-level data processing abstractions, including MapReduce and parameter servers.

Recently cloud providers (e.g., AWS Lambda, Google Cloud Functions) and open source projects (e.g., OpenLambda [16], OpenWhisk [31]) have developed infrastructure to run event-driven, stateless functions as micro-services. In this model, a function is deployed once and is invoked repeatedly whenever new inputs arrive and elastically scales with input size. Our key insight is that we can dynamically inject code into these functions, which combined with remote storage, allows us to build a data processing system that inherits the elasticity of the serverless model while addressing the simplicity for end users.

We describe a prototype system, PyWren[1], developed in Python with AWS Lambda. By employing only stateless functions, PyWren helps users avoid the significant developer and management overhead that has until now been a necessary prerequisite. The complexity of state management can instead be captured by a global scheduler and fast remote storage. With PyWren, we seek to under-

Cloud Programming Simplified: A Berkeley View on Serverless Computing

Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai
Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira
Karl Krauth, Neeraja Yadwadkar, Joseph E. Gonzalez, Raluca Ada Popa
Ion Stoica, David A. Patterson

UC Berkeley

serverlessview@berkeley.edu

Abstract

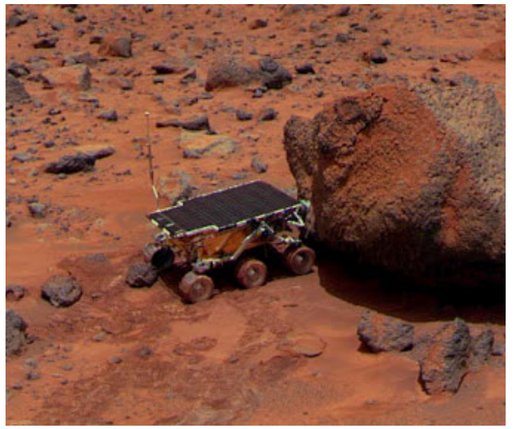
Serverless cloud computing handles virtually all the system administration operations needed to make it easier for programmers to use the cloud. It provides an interface that greatly simplifies cloud programming, and represents an evolution that parallels the transition from assembly language to high-level programming languages. This paper gives a quick history of cloud computing, including an accounting of the predictions of the 2009 Berkeley View of Cloud Computing paper, explains the motivation for serverless computing, describes applications that stretch the current limits of serverless, and then lists obstacles and research opportunities required for serverless computing to fulfill its full potential. Just as the 2009 paper identified challenges for the cloud and predicted they would be addressed and that cloud use would accelerate, we predict these issues are solvable and that serverless computing will grow to dominate the future of cloud computing.

Contents

But what do I do now?

My research group's goal?

Rapid understanding of every unknown small molecule



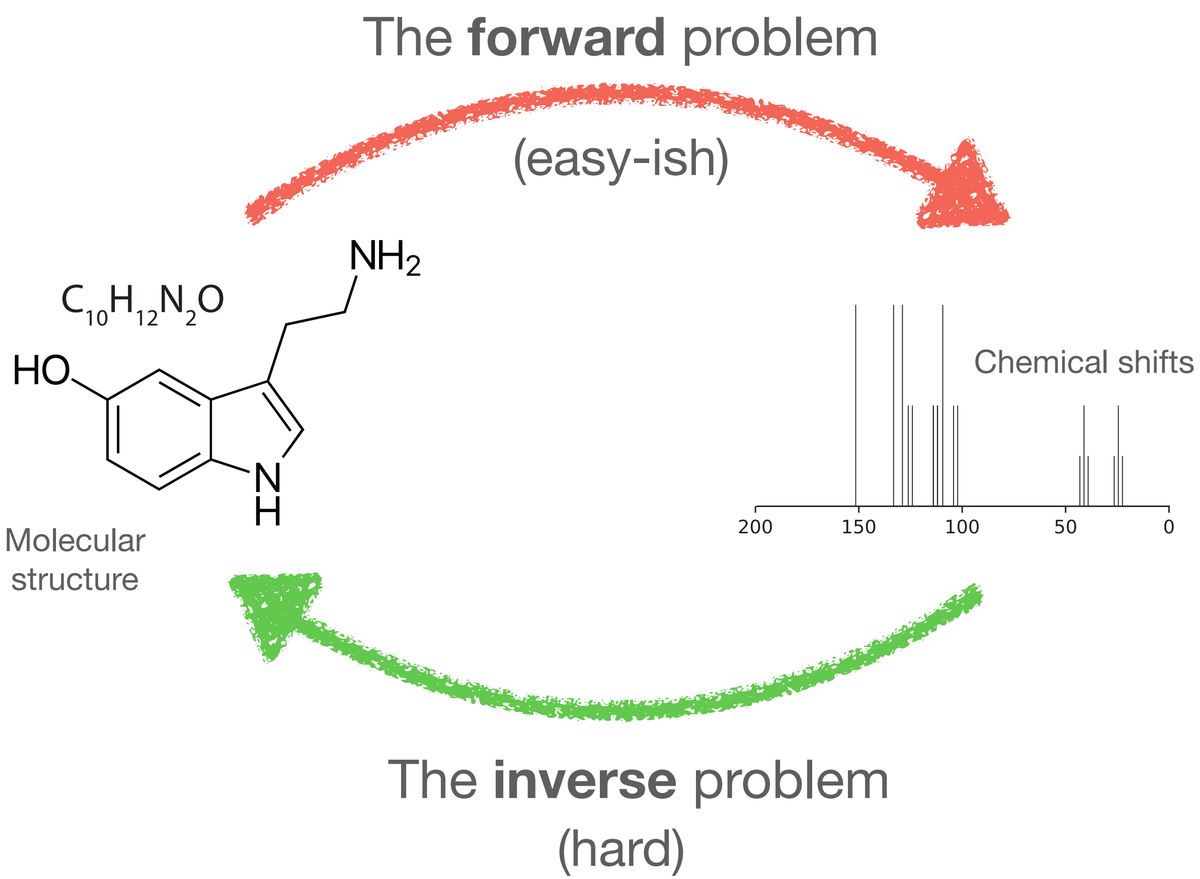
Via automated analysis and experimentation

What do I work on?

Inverse problems for spectroscopy with machine learning

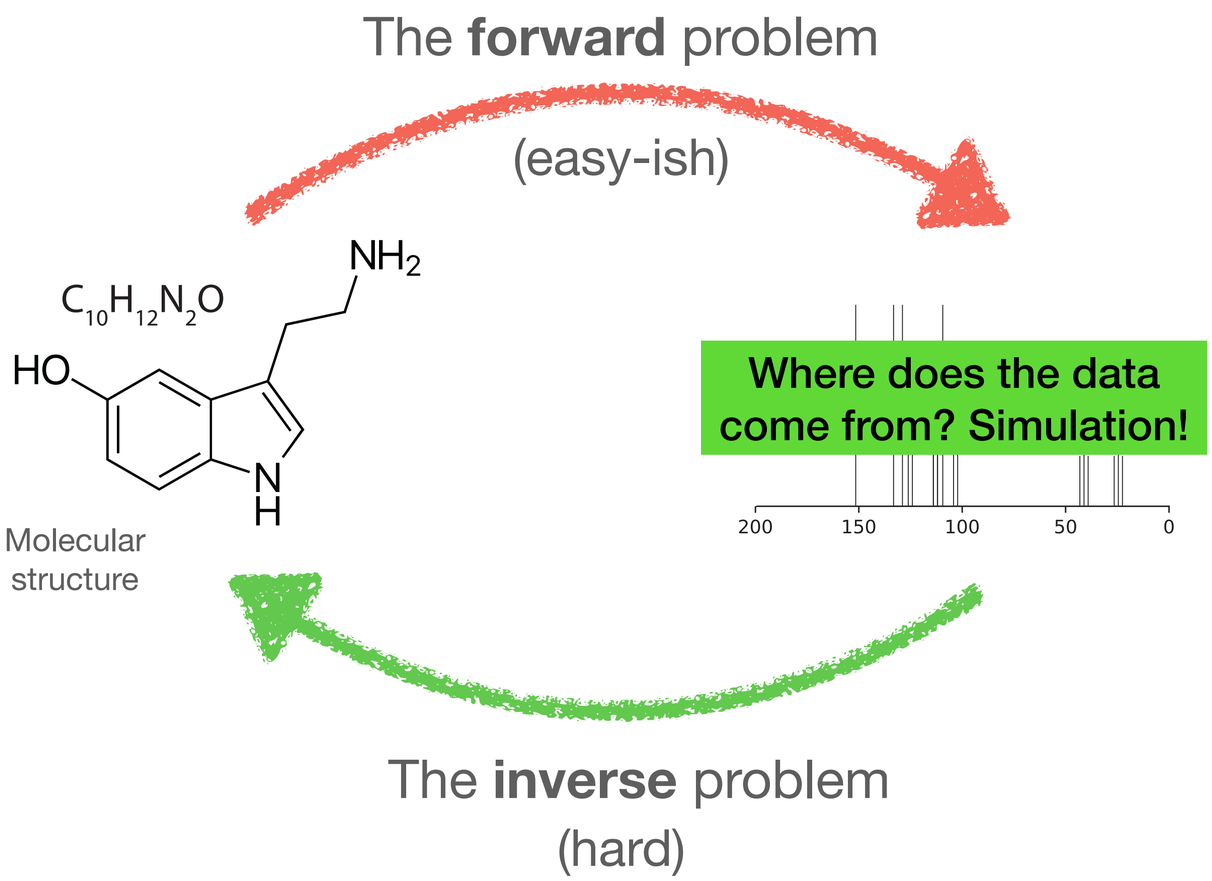
What do I work on?

Inverse problems for spectroscopy with machine learning



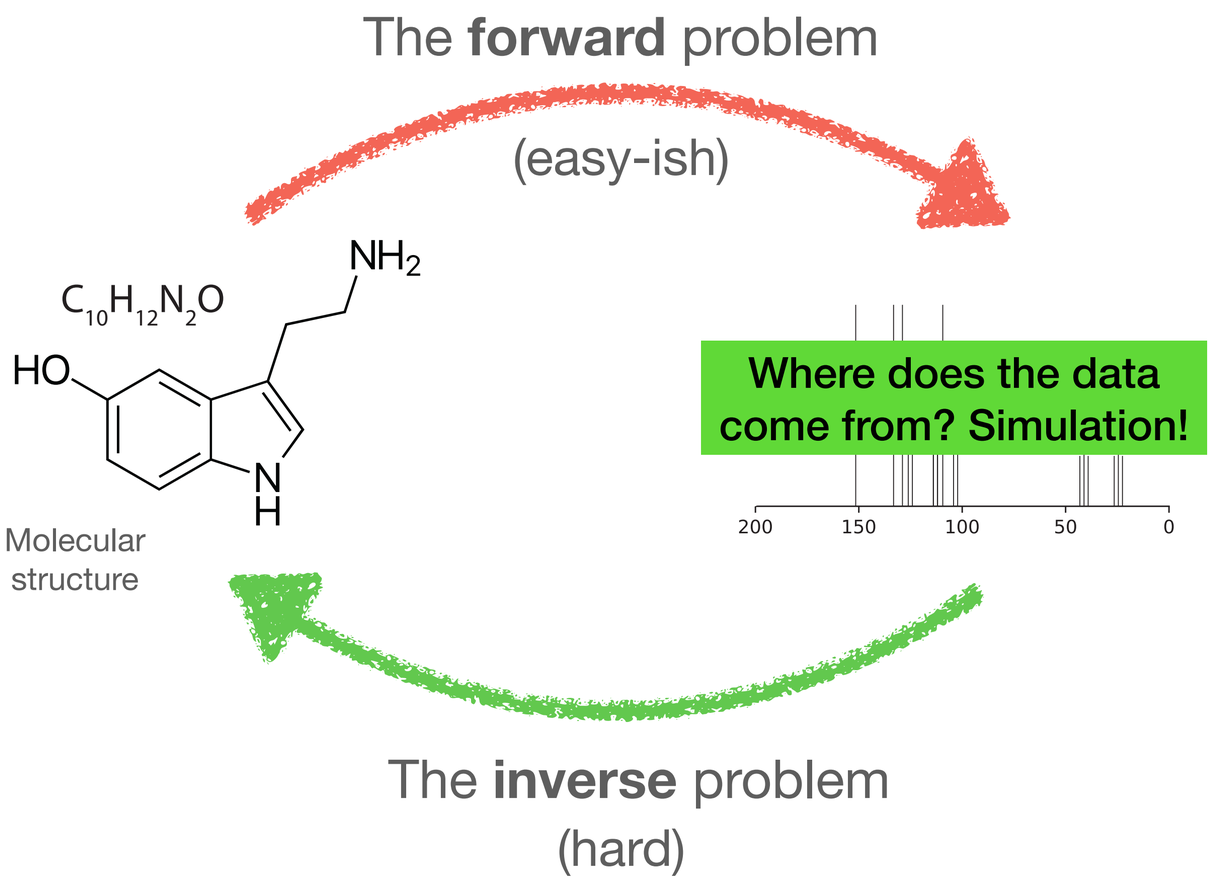
What do I work on?

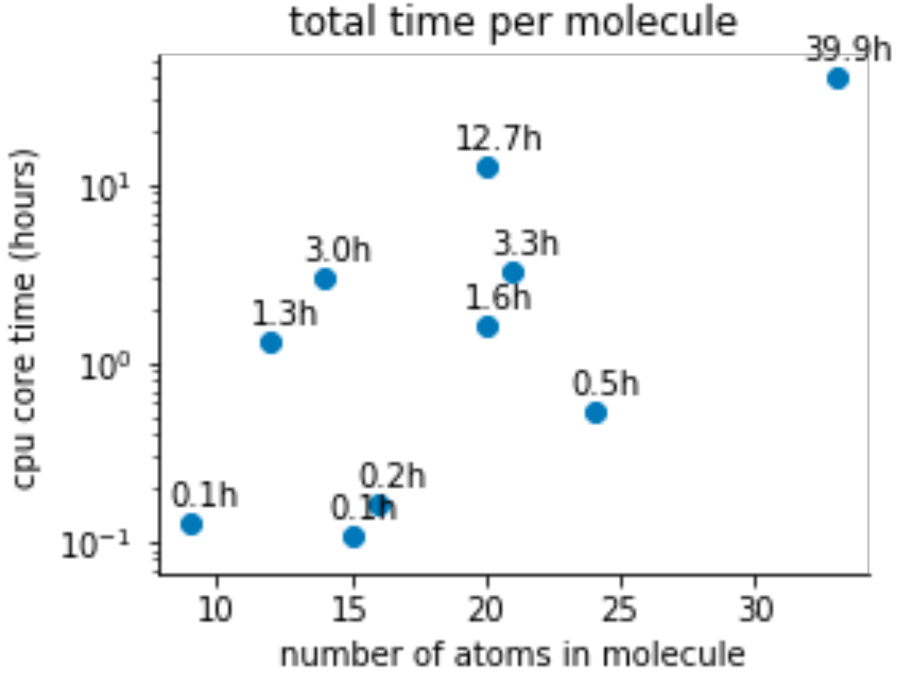
Inverse problems for spectroscopy with machine learning



What do I work on?

Inverse problems for spectroscopy with machine learning



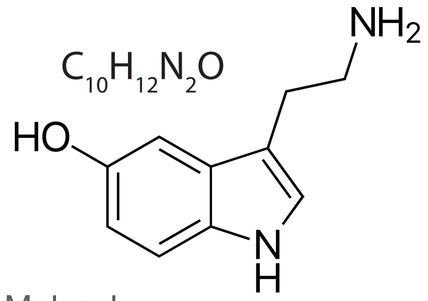


What do I work on?

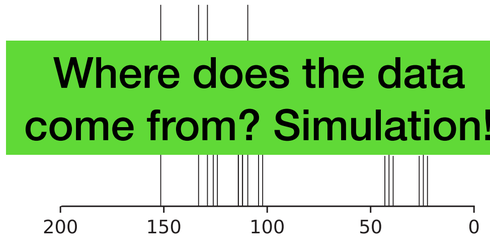
Inverse problems for spectroscopy with machine learning

The **forward** problem

(easy-ish)



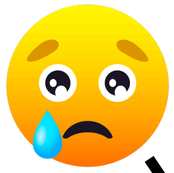
Molecular structure

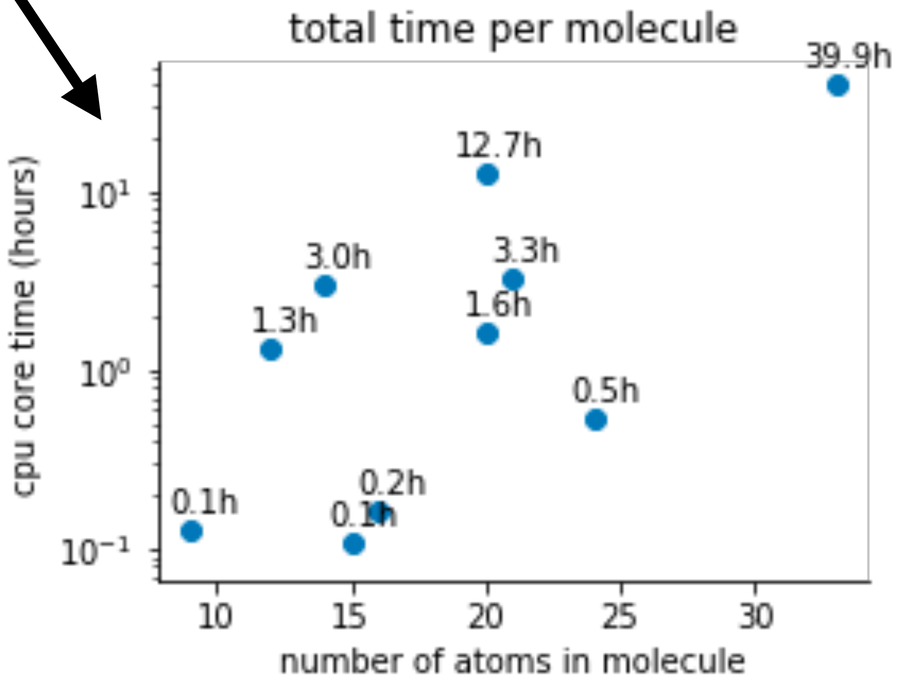


Where does the data come from? Simulation!

The **inverse** problem

(hard)





funcX to the rescue!

Compute requirements : simulating chemical shifts

funcX to the rescue!

Compute requirements : simulating chemical shifts

Each task takes ~10-60 core-h
(Say ~40 core-h on avg)
(0.1-1 wall-clock on a modern node)

funcX to the rescue!

Compute requirements : simulating chemical shifts

Each task takes ~10-60 core-h
(Say ~40 core-h on avg)
(0.1-1 wall-clock on a modern node)

I have 100,000 of them

~4M core-h total

funcX to the rescue!

Compute requirements : simulating chemical shifts

Each task takes ~10-60 core-h
(Say ~40 core-h on avg)
(0.1-1 wall-clock on a modern node)

I have 100,000 of them
~4M core-h total

Input size: ~10kB

Outputs size: ~1 MB

funcX to the rescue!

Compute requirements : simulating chemical shifts

Each task takes ~10-60 core-h
(Say ~40 core-h on avg)
(0.1-1 wall-clock on a modern node)

I have 100,000 of them

~4M core-h total

Input size: ~10kB

Outputs size: ~1 MB

Embarrassingly parallel

funcX to the rescue!

Compute requirements : simulating chemical shifts

Each task takes ~10-60 core-h
(Say ~40 core-h on avg)
(0.1-1 wall-clock on a modern node)

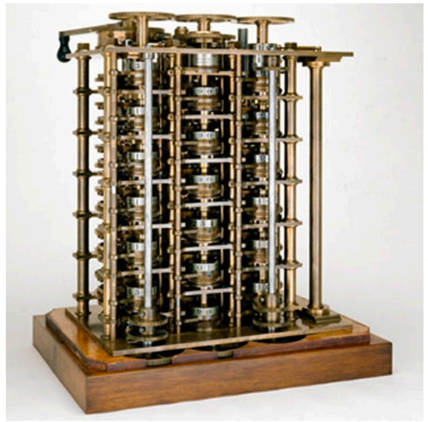
I have 100,000 of them

~4M core-h total

Input size: ~10kB

Outputs size: ~1 MB

How long does this take?



~7 years on
a workstation

~1 month at TACC thanks to funcX

Embarrassingly parallel

How do I (mis-)use funcX

How do I (mis-)use funcX

- Much of FaaS work has focused on real-time use, but that’s not my primary use case — I want fire-and-forget

How do I (mis-)use funcX

- Much of FaaS work has focused on real-time use, but that’s not my primary use case — I want fire-and-forget
- My tasks are generally idempotent — retries can be fine (in fact they cache into $SCRATCH on the endpoint)

How do I (mis-)use funcX

- Much of FaaS work has focused on real-time use, but that’s not my primary use case — I want fire-and-forget
- My tasks are generally idempotent — retries can be fine (in fact they cache into $SCRATCH on the endpoint)
- I maintain this is a common use case!

How do I (mis-)use funcX

- Much of FaaS work has focused on real-time use, but that’s not my primary use case — I want fire-and-forget
- My tasks are generally idempotent — retries can be fine (in fact they cache into $SCRATCH on the endpoint)
- I maintain this is a common use case!
- I don’t want to think about the endpoint too much

How do I (mis-)use funcX

- Much of FaaS work has focused on real-time use, but that's not my primary use case — I want fire-and-forget
- My tasks are generally idempotent — retries can be fine (in fact they cache into $SCRATCH on the endpoint)
- I maintain this is a common use case!
- I don't want to think about the endpoint too much
- So I fire off 100k tasks and don't worry about it…

How do I (mis-)use funcX

- Much of FaaS work has focused on real-time use, but that's not my primary use case — I want fire-and-forget
- My tasks are generally idempotent — retries can be fine (in fact they cache into $SCRATCH on the endpoint)
- I maintain this is a common use case!
- I don't want to think about the endpoint too much
- So I fire off 100k tasks and don't worry about it…

Actual photo of funcX devops team

Solutions

Solutions

- Substantially reduce size of return values (1MB -> 5kB) by staging them to S3

Solutions

- Substantially reduce size of return values (1MB -> 5kB) by staging them to S3
- FuncX team has developed an Executor interface (similar to Python's AsyncIO) that makes this scale of tasks easy

Solutions

- Substantially reduce size of return values (1MB -> 5kB) by staging them to S3
- FuncX team has developed an Executor interface (similar to Python's AsyncIO) that makes this scale of tasks easy
- Everything works swimmingly now!

The future

Feature requests

The future

Feature requests

- Visibility into the myriad of queues used

The future

Feature requests

- Visibility into the myriad of queues used
- Flush the queue completely (reset to zero)

The future

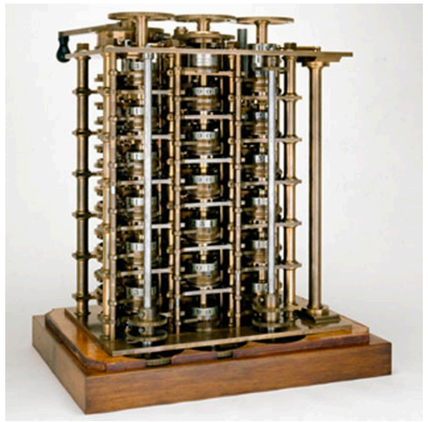
Feature requests

- Visibility into the myriad of queues used
- Flush the queue completely (reset to zero)
- Hosted endpoints?

The future

Feature requests

- Visibility into the myriad of queues used
- Flush the queue completely (reset to zero)
- Hosted endpoints?

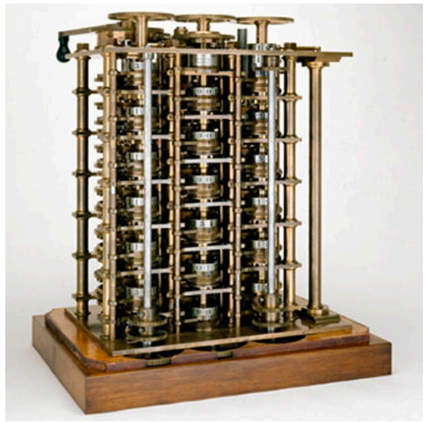


~7 years on
a workstation

The future

Feature requests

- Visibility into the myriad of queues used
- Flush the queue completely (reset to zero)
- Hosted endpoints?



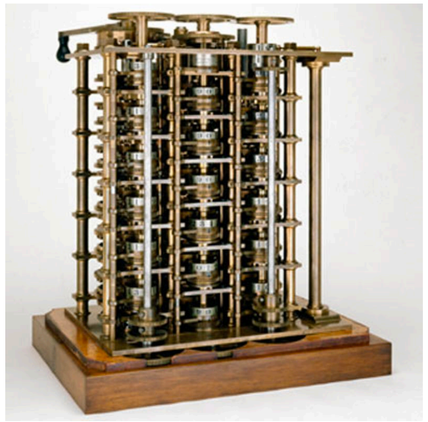
~7 years on
a workstation

~1 month at TACC
thanks to funcX

The future

Feature requests

- Visibility into the myriad of queues used
- Flush the queue completely (reset to zero)
- Hosted endpoints?



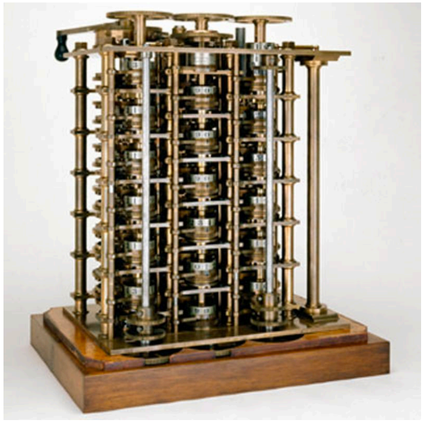
~7 years on
a workstation

~1 month at TACC
thanks to funcX

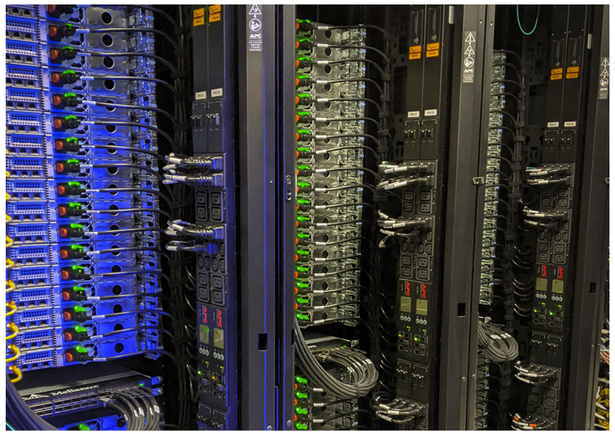
This should take an hour!

The future

Feature requests



~7 years on
a workstation

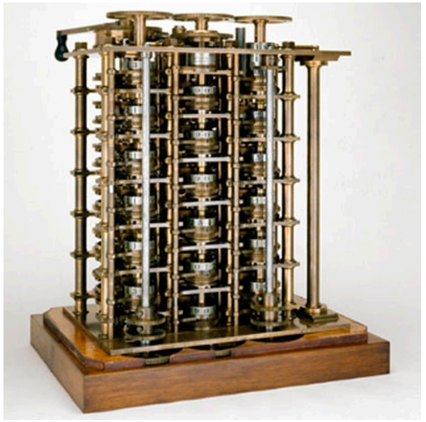


~1 month at TACC
thanks to funcX

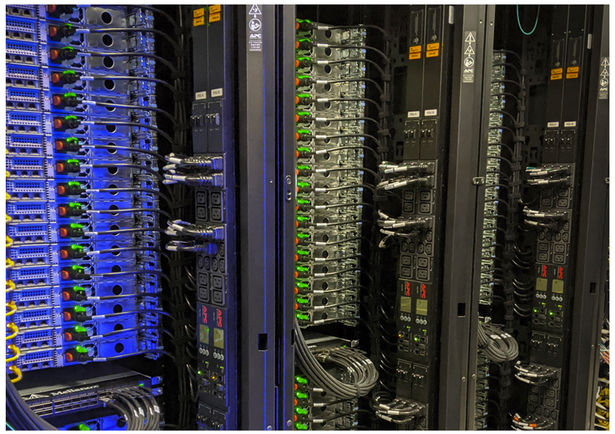
This should take an hour!

The future

Feature requests



~7 years on
a workstation



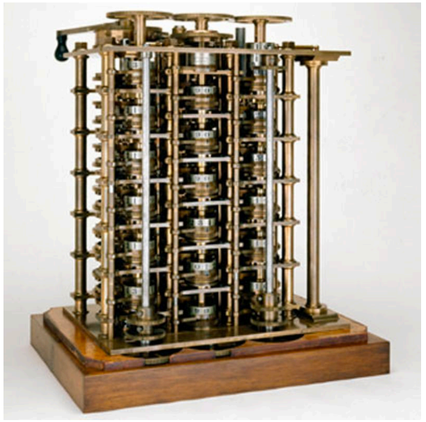
~1 month at TACC
thanks to funcX

This should take an hour!

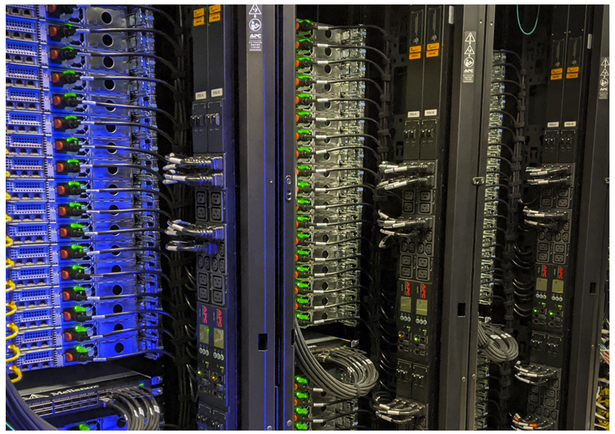
FuncX + national supercomputing infrastructure: the thermodynamic limit of computation!

The future

Feature requests



~7 years on
a workstation



~1 month at TACC
thanks to funcX

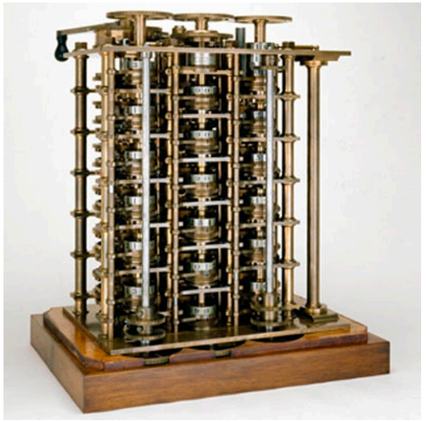
This should take an hour!

FuncX + national supercomputing infrastructure: the thermodynamic limit of computation!

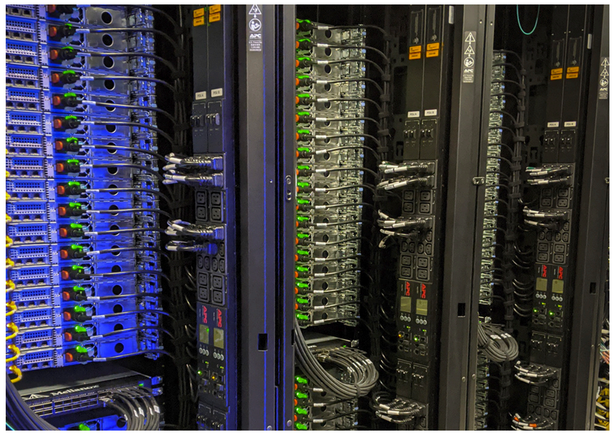
Endpoint-agnostic computation

The future

Feature requests



~7 years on
a workstation



~1 month at TACC
thanks to funcX

This should take an hour!

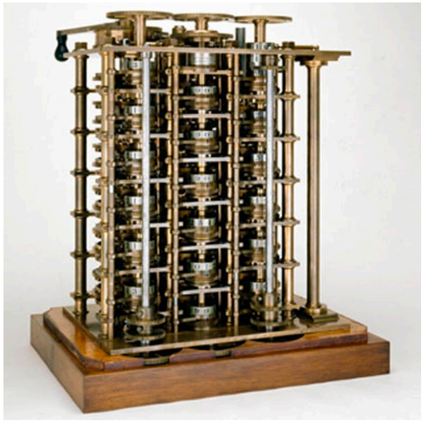
FuncX + national supercomputing infrastructure: the thermodynamic limit of computation!

Endpoint-agnostic computation

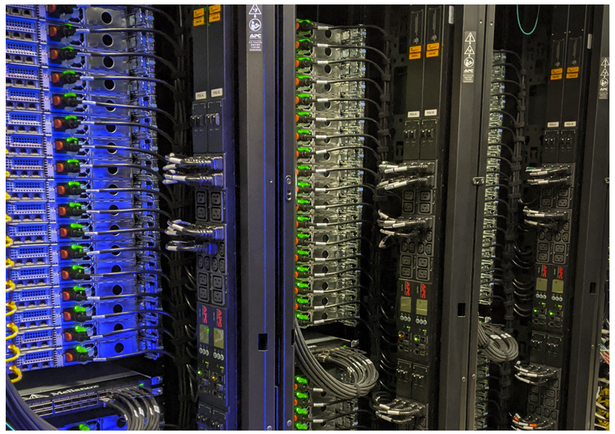
Embedded into language significantly expands accessibility!

The future

Feature requests



~7 years on
a workstation



~1 month at TACC
thanks to funcX

This should take an hour!

FuncX + national supercomputing infrastructure: the thermodynamic limit of computation!

Endpoint-agnostic computation

Embedded into language significantly expands accessibility!

funcX lets us all spend more time on science and less on infrastructure!