



Profiling, Predicting, and Provisioning: Enabling Cost-aware Computing for the Cloud and Modern Heterogeneous Environments

Matt Baughman

Committee: Kyle Chard, Ian Foster, Hank Hoffman

Master's Thesis Defense

Aug. 17, 2021



The State of Heterogeneity...

	HPC	Cloud	Edge
Description	<ul style="list-style-type: none">• System-level homogeneity• Enterprise parts• Increasingly accelerators	<ul style="list-style-type: none">• Configurable• Enterprise parts• Extensive resources• accelerators	<ul style="list-style-type: none">• Integrated systems• Varying infrastructure• Low-power, consumer components
Related Metrics (time and...)	<ul style="list-style-type: none">• Allocation units• Queueing overheads	<ul style="list-style-type: none">• Allocation units• \$\$\$ (money)	<ul style="list-style-type: none">• Resource capability• Overhead costs
Current Examples	<ul style="list-style-type: none">• DOE Computing Infrastructure• University Clusters	<ul style="list-style-type: none">• AWS• Google Cloud• Chameleon	<ul style="list-style-type: none">• Raspberry Pi• Amazon Greengrass• Particle Cloud

The State of Heterogeneity... Applications

❖ Instrument Control

- Real-time
- Time-sensitive
- Compute offloading

❖ HPC workloads

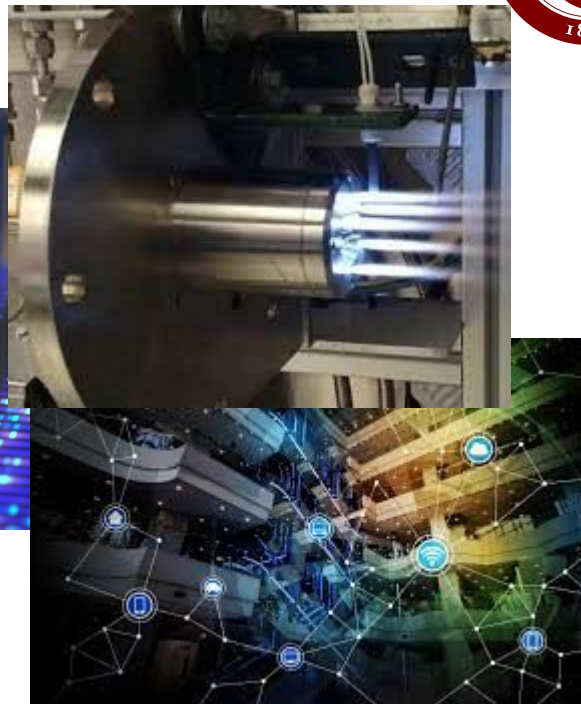
- Configuration dependent
- Resource selection
- Deployment and scaling

❖ Edge and sensor networks

- Wide ranging information
- Low power → needs execution environment

❖ Machine Learning/ Artificial Intelligence

- Utilization of accelerators/ ASICs
- Data and location dependent



The Problem...

- ❖ An ever growing number of tasks...
- ❖ An ever growing number of resources to use...
- ❖ How do we enable effective matching?

```
def control_stuff(time_sensitive_data):  
    return(doing_stuff)
```

```
def definitely_recursion(recursive_data):  
    return(definitely_recursion(recursive_data))
```

```
def sent_stuff(uncertainable_stuff):
```

```
def pass_masters(questionable_stuff):  
    return(pass if random((0,100))>50 else do_industry())
```

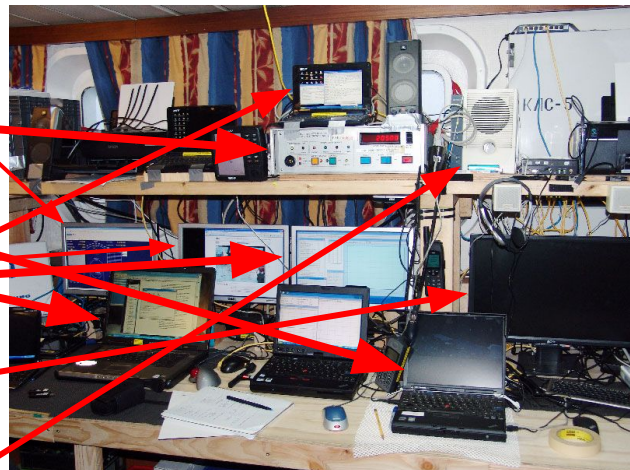
```
def do_stuff(big_data):  
    return(stuff_done)
```

```
while True:  
    1+1  
    return(unlikely)
```

```
def solve_collatz(start_value):  
    if start_value>0 and start_value <= np.inf:  
        return(1)  
    else:  
        return(1)
```

```
def parallel_stuff(1):  
    return([1]*np.inf)
```

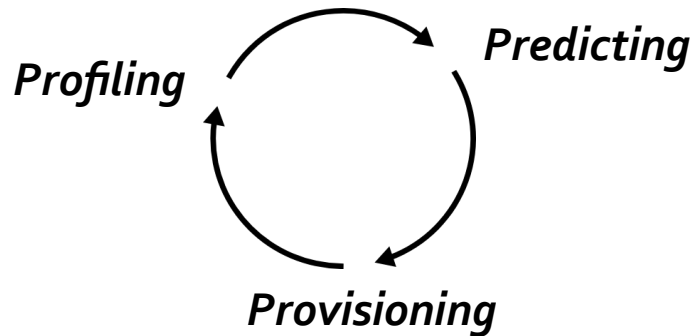
```
def p_equals_np(math):  
    return(True)
```



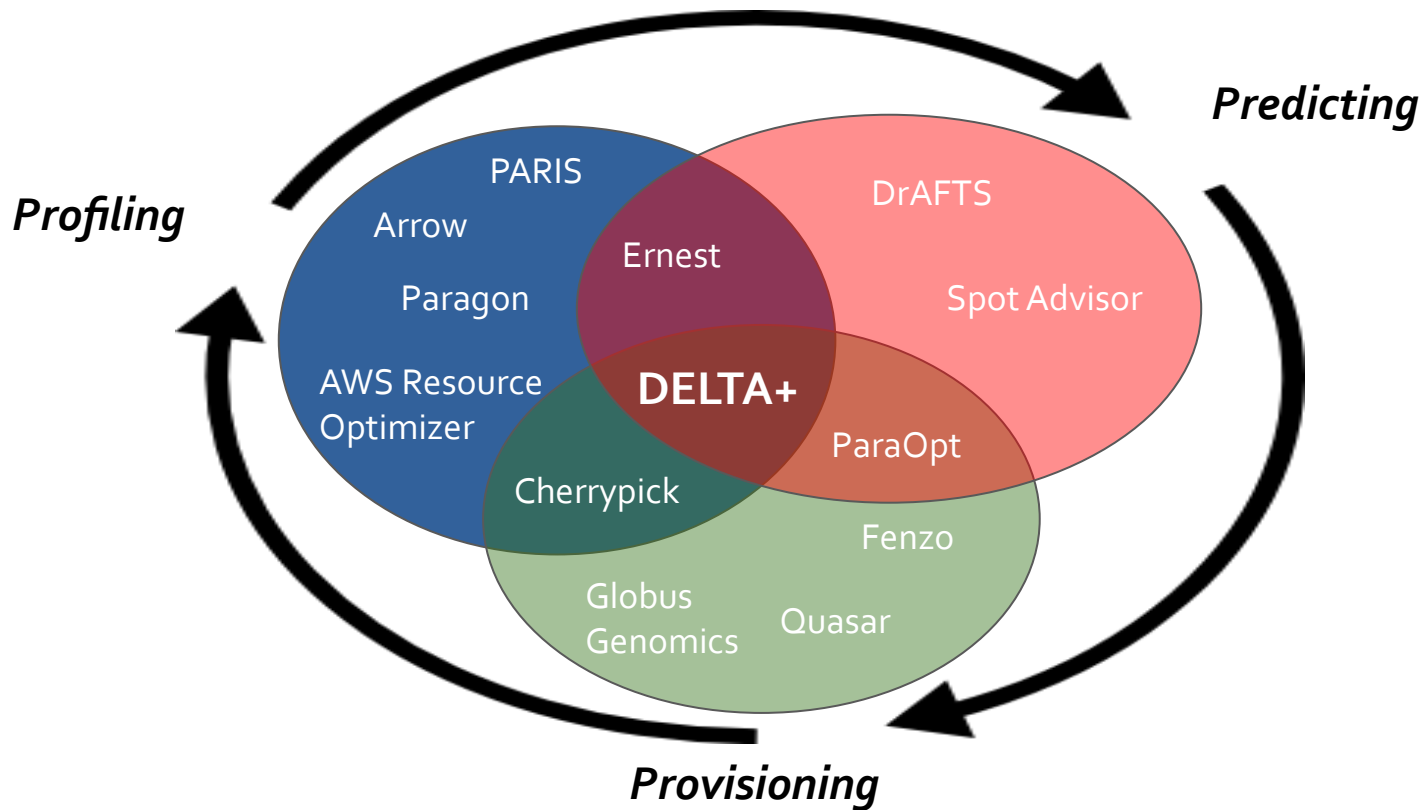
(“modern” distributed system)

Problem Formalization

- ❖ Given execution characteristics of an application on one resource, can we predict those characteristics on another resource?
- ❖ Given a cloud instance type and an application's runtime, can we predict how much it will cost to run using spot instances?
- ❖ How do we match workloads to resources using cost and time constraints?

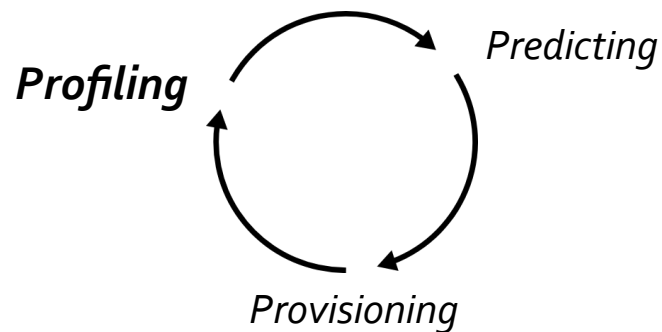


Where to find a solution?





Estimating Execution Characteristics on the Cloud



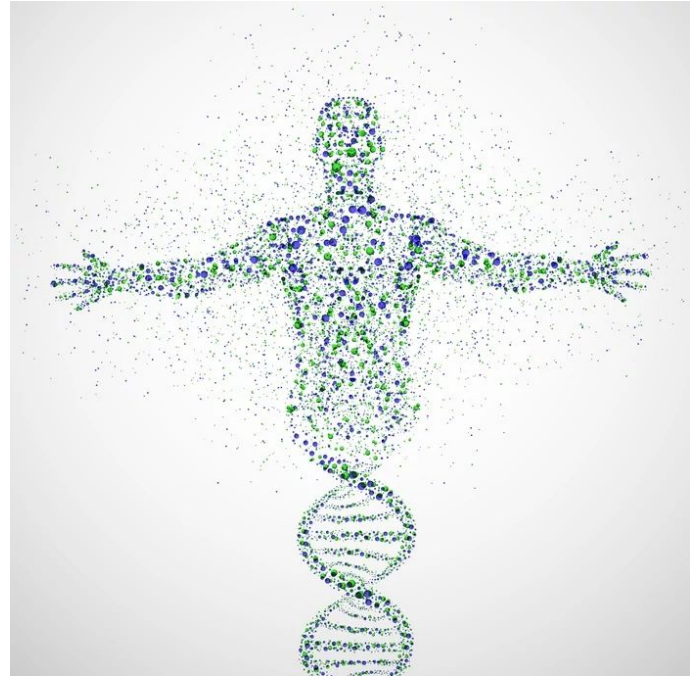
The Profiling Problem

- ❖ Matching workloads to one of nearly 400 instance types on the cloud
 - Human selection is generally not that good and definitely not scalable
 - The number of instances is up three fold from three years ago
 - Not all portions of an ensemble of applications may be equally suited to a resource
- ❖ Existing HPC application models focus on more homogenous environments
- ❖ Manual profiling is expensive



Application domain: Genomics

- ❖ Next generation genomics relies on both HPC and cloud infrastructures
 - A great example of big data *and* big compute
 - Increasing use of cloud due to innovative and changing nature of field
- ❖ Thousands of tools developed for specialized analyses
 - Wrapped into unique pipelines
- ❖ Small overheads compound when analyzing millions of genomes



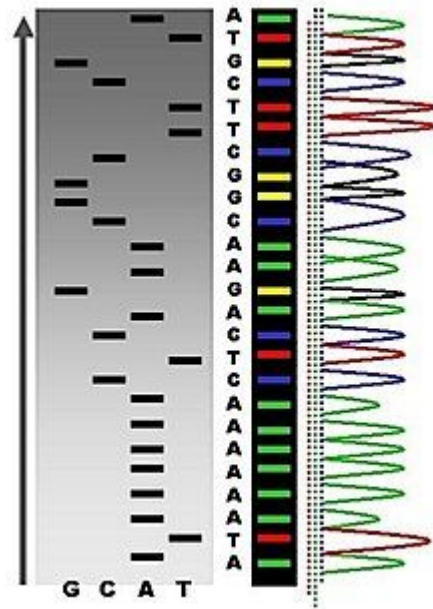


The Profiling Solution

- ❖ Modular, lightweight predictive solution
- ❖ Learn from disparate and sparsely overlapping data
- ❖ Utilize this multisource information to enable efficient resource selection
- ❖ Use an intentional design process to allow the predictive task to be broken into sub-tasks
 - Task 1: Predicting runtime based on resource
 - Task 2: Predicting runtime based on input data
 - Task 3: Combine these models into a composite

Execution characteristics across cloud instance types

- ❖ Application: SwiftSeq workflow (Pitt, 2018)
 - High performance genomics pipeline
 - Modularly composable with multiple tools
 - Identify genomic “variants”
- ❖ ~600 execution traces of 9-tool pipelines
 - Two different pipelines based on primary variant caller
- ❖ Data from 14 different cloud instance types
 - 2 vCPUs to 72 vCPUs
- ❖ Known but variable quantities re: instance configuration
 - Number of CPUs
 - Quantity of RAM
 - Optimization type
- ❖ Constant input data





Resource Model and Results

Using Platypus variant caller

Application	MAPE	SD	Baseline MAPE
BwaMem	8.0	6.8	33.7
RgMergeSort	10.6	6.3	10.6
PicardMarkDuplicates	10.3	7.9	68.2
IndexBam	34.7	21.3	57.5
PlatypusGerm	33.3	26.2	76.0
ContigMergeSort	13.5	16.1	56.5
ConcatVcf	30.5	34.9	51.2
SamtoolsFlagStat	8.8	3.2	9.4

Using Haplotype variant caller

Application	MAPE	SD	Baseline MAPE
BwaMem	12.0	16.1	40.8
RgMergeSort	8.4	6.6	8.4
PicardMarkDuplicates	11.8	9.9	64.9
IndexBam	44.0	50.0	142.4
ContigMergeSort	26.1	21.9	31.0
ConcatVcf	8.2	5.8	10.4
SamtoolsFlagStat	9.8	8.5	14.5
HaplotypeCaller	5.7	4.8	144.1

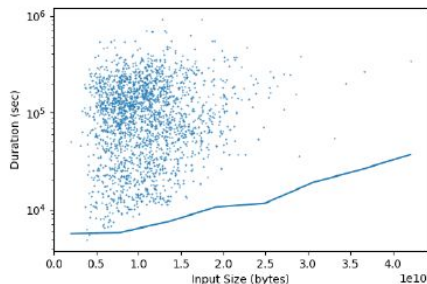
$$f(vCPU_s, RAM) = \frac{1}{a * vCPU_s + b * RAM} + c$$

Execution characteristics dependent on input data

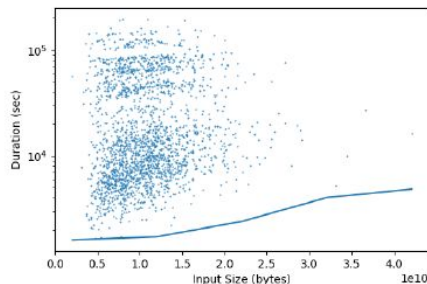
- ❖ ~50,000 SwiftSeq traces from Beagle2 supercomputer
- ❖ 14-tool pipelines leading to nearly ~1mil tasks
 - Focused on tumor-normal pair comparison
- ❖ Homogeneous hardware
 - Autoscaling
- ❖ Noisy data due to scaling



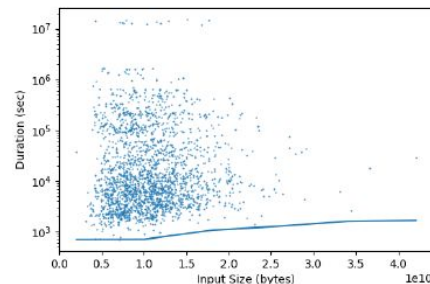
Fitting Lower Bounds



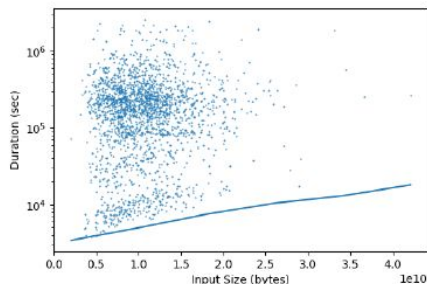
(a) BwaMem



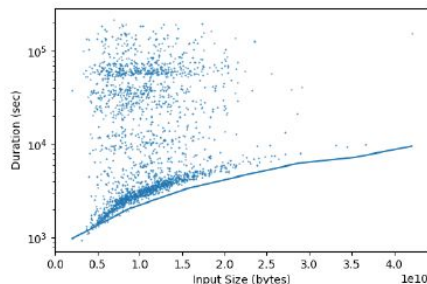
(b) ContigMergeSort



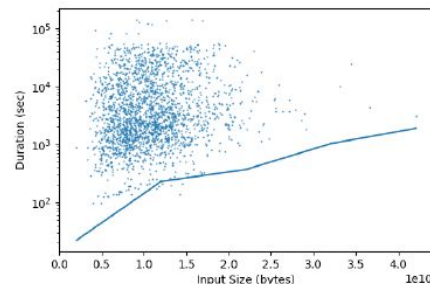
(c) IndexBam



(d) PicardMarkDuplicates



(e) RgMergeSort



(f) SamtoolsFlagstat



Input data model

- ❖ Model
 - Similar to resource model
 - Fit along lower bound
- ❖ Results
 - <15% error across all tools
 - Increased accuracy correlated with highest cluster density

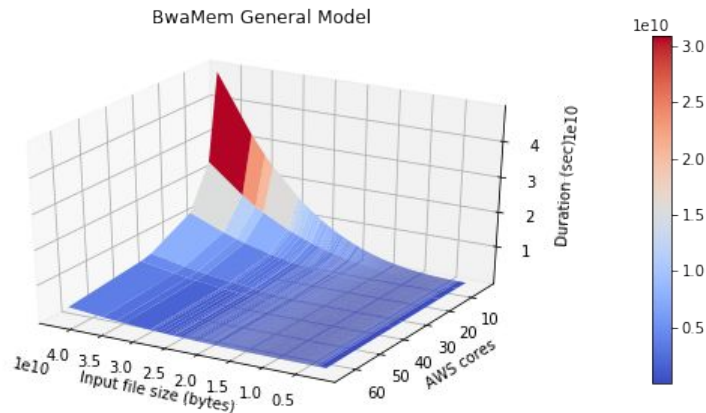
$$\begin{aligned}g(\text{size}) &= \frac{k}{\text{size}^l} + m \\ &= k \times \text{size}^{-l} + m\end{aligned}$$

Application	MAPE	SD
BwaMem	5.4	5.1
RgMergeSort	1.4	1.0
PicardMarkDuplicates	8.0	5.4
IndexBam	14.5	8.7
ContigMergeSort	4.3	4.0
SamtoolsFlagStat	12.6	10.0

Model Combination and Retraining

- ❖ Creation of composite regression
 - Similar to combining CPU and RAM model for AWS data
- ❖ Accuracy greatly improved by retraining composite model with single new point outside existing axis

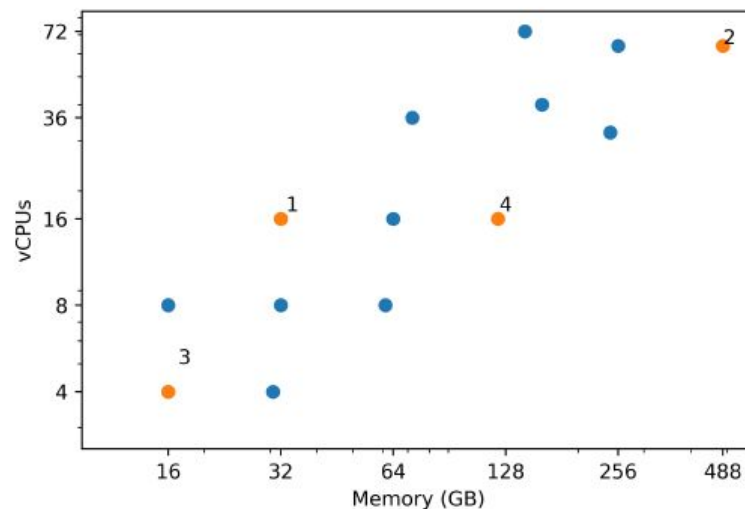
$$G(\text{size}, v\text{CPU}s, \text{RAM}) = f(v\text{CPU}s, \text{RAM}) \times g(\text{size}) \times r$$



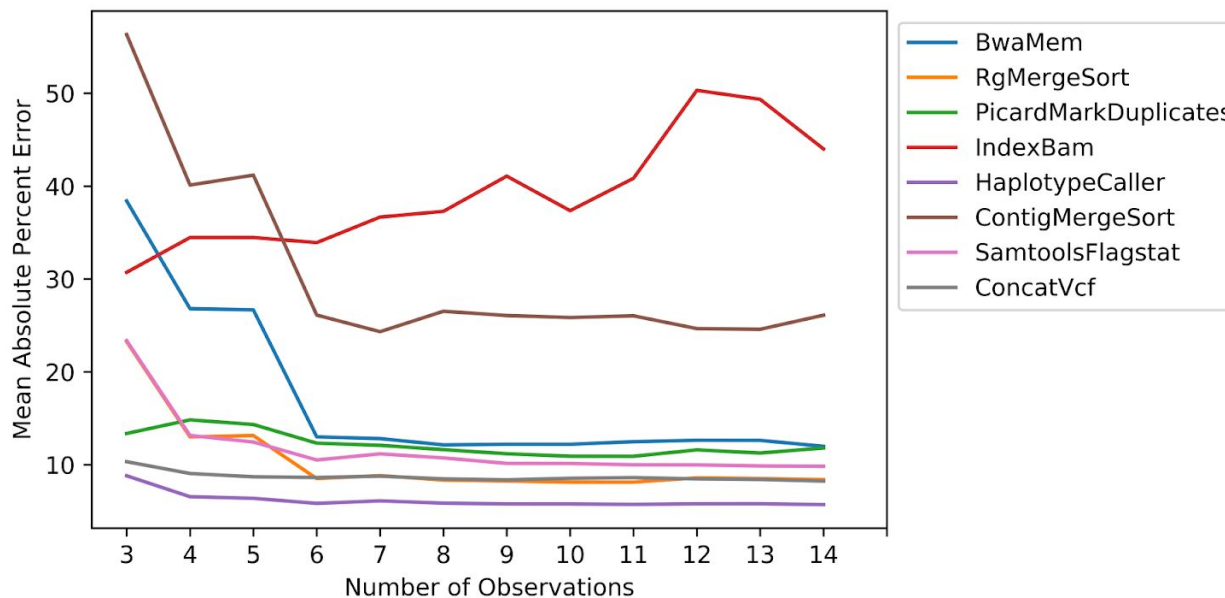


Experiment Design and Variance Reduction

- ❖ Progression of experimental instance selection
 - Identify areas of knowledge sparsity
 - Predict point where knowledge at that point would maximize reduction in uncertainty after a single additional experiment
 - Maximization of marginal uncertainty reduction

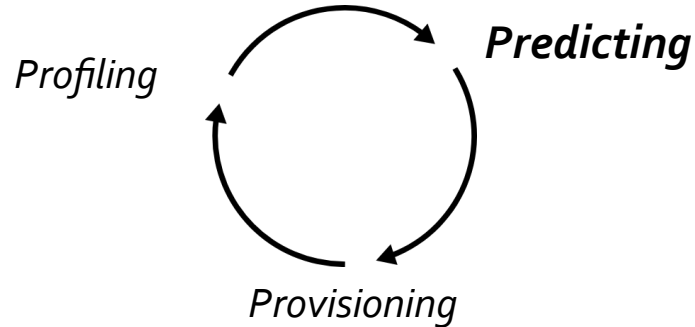


Experiment Design Results



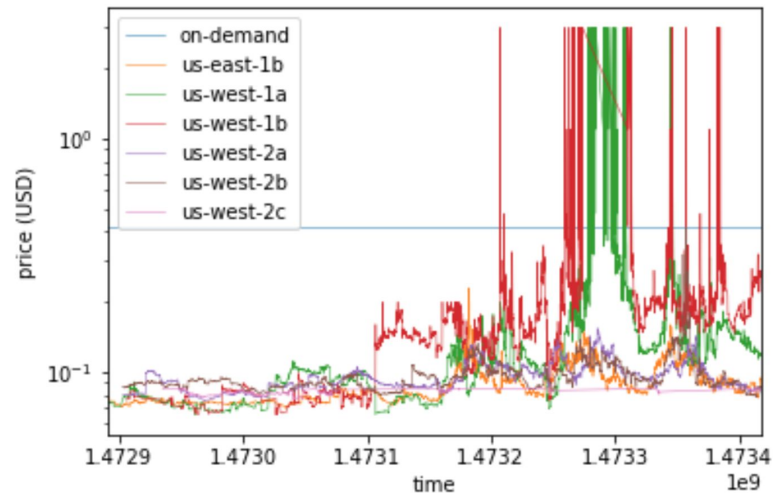


Predicting Resource Costs Given Configuration



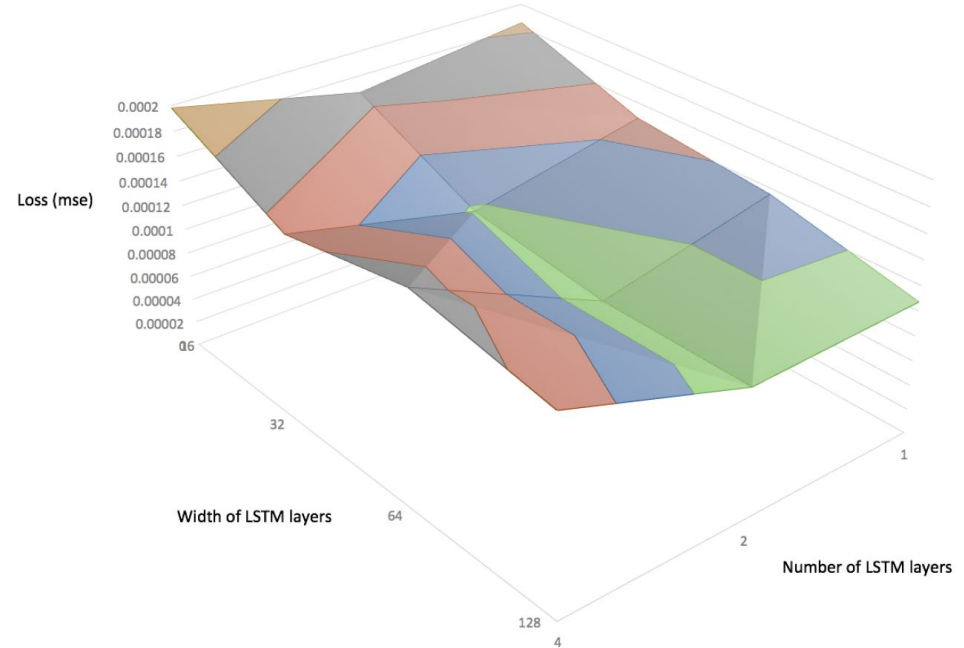
AWS Spot Market

- ❖ Auctioning off overhead resources
- ❖ Preemptible instances
 - Terminations with 2 minutes warning
 - Due to price or market pressures
- ❖ Can vary in durability, price, and availability
- ❖ Offers savings up to 90%
 - Excellent for short tasks
 - Can be exploited for ultra-cheap function execution



Spot Price Prediction

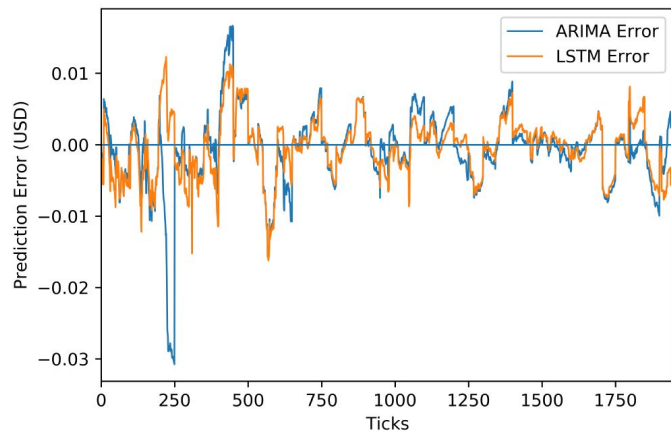
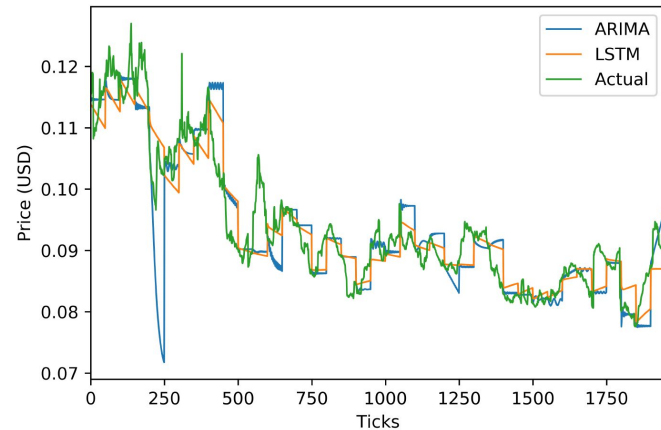
- ❖ Preliminary grid search
- ❖ LSTM model for time series prediction
 - Two layers of LSTM blocks
 - 32 x 2 primary model
 - Single dense layer to consolidate output
 - 250 epochs
- ❖ Expected value vs durability





Price Prediction Results

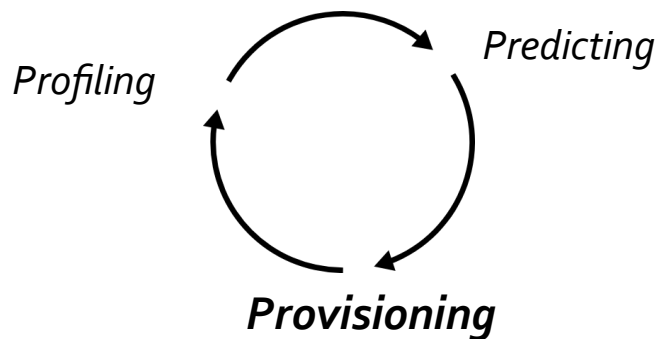
- Up to 90% reduction in error for some specific time series for some specific time series
- Consistent 15% reduction in validation error over ARIMA
- Shows statistically significant improvement over ARIMA in all training and validation metrics.



	ARIMA	LSTM
Root Mean Square Error (RMSE)	0.00557	0.00423
Mean Absolute Error (MAE)	0.00362	0.00320



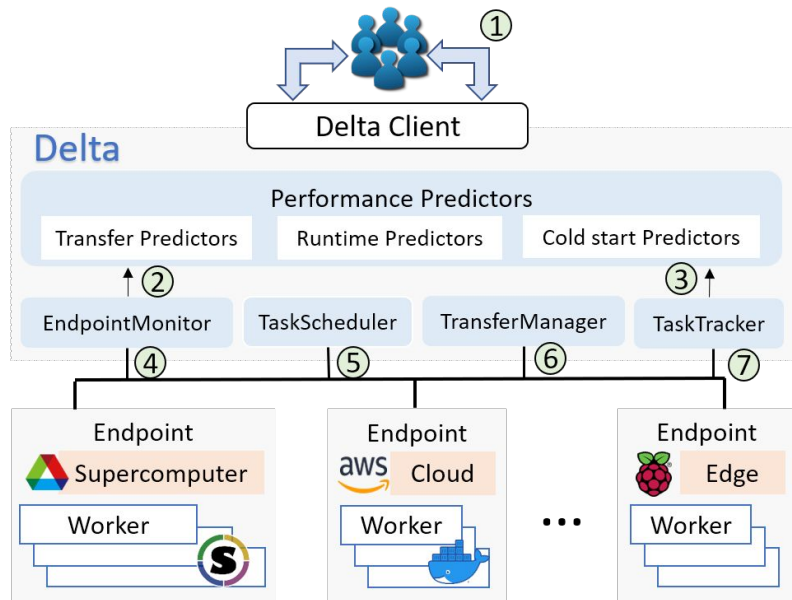
Cost-Aware Execution in Automated Serverless





Delta Overview

- ❖ Automated coordination of FaaS ecosystems
- ❖ Unifying compute across the edge, HPC, and cloud
- ❖ Design components
 - Remove endpoint selection from end-users
 - Allow for tunable optimization metrics
 - Enable intelligent use of all available resources
 - Handle dynamic ecosystems
- ❖ Areas for improvement
 - Does not take advantage of resource providers
 - Optimization for single dimension---time



Kumar, Baughman, et al. "Coding the computing continuum: Fluid function execution in heterogeneous computing environments." In *Heterogeneity in Computing Workshop at IPDPS* (IPDPSW) pp. 66-75. 2021.

DELTA+: Adding multi-dimensional costs

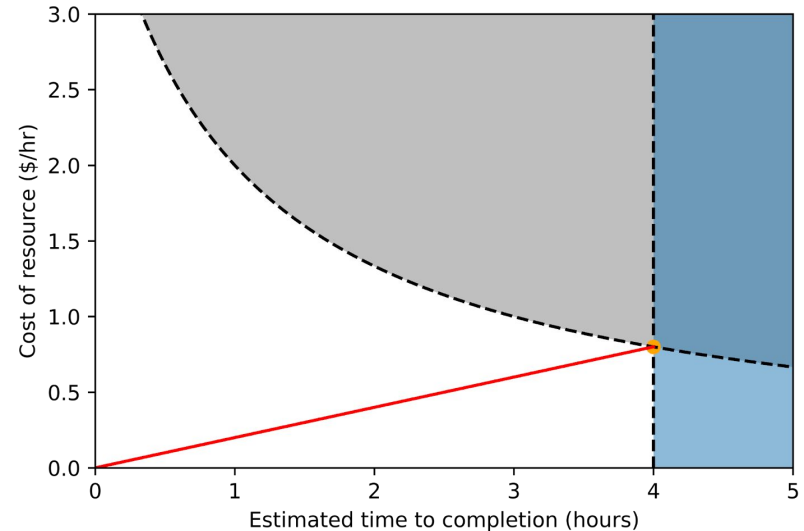
- ❖ Distribute tasks to different resources with different costs
- ❖ Relating cost vs. time
 - Different resources have different optimization criteria
- ❖ User definition
 - Allows for expression of tradeoffs and weighting
 - E.g., \$1 ≈ 10 minutes
 - Costs at different levels
- ❖ Constraint satisfaction
 - Cost budgets
 - Time budgets
 - HPC Allocations
- ❖ How do we optimize under constraints?





DELTA+: Constraints and optimization

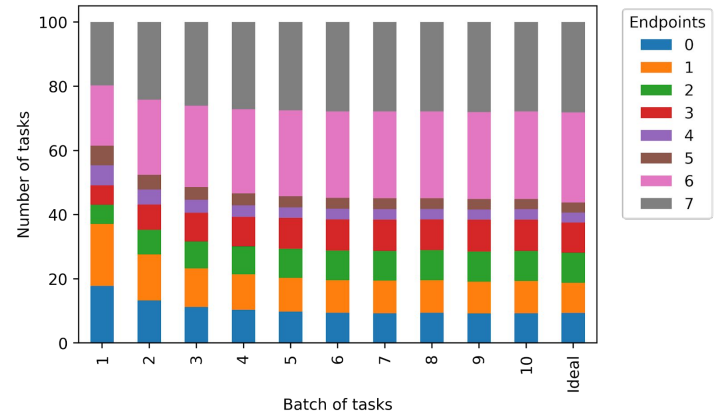
- ❖ Deadlines
 - Allows for prioritization of faster or more time-effective resources for task allocation
 - When do you need the compute done?
- ❖ Cost constraints
 - Absolute budgets
 - Prioritizes cost-effective resources
- ❖ Automated optimization definition
 - Multiple constraints make resource selection difficult
 - Solution: select resources to maximize the likelihood of workload completion under constraint





DELTA+: Experimental Results

- ❖ Incorporate probabilistic task placement
- ❖ Experimental setup
 - Simulated 8 endpoints of various costs and performance levels
 - Costs initially known, performance hidden
 - Launched 10 batches of 100 tasks
 - Optimized for performance per dollar
- ❖ Results
 - **Learned relative performance to within 10% of optimal in 5 batches**





Future Work

- ❖ Incorporating the full 3-P cycle into Delta+
 - Use existing capacity for multidimensional costs
 - Develop control mechanisms to automate resource providers
 - Integrate model-based cost prediction for cloud and HPC resources
 - Coordinate these tasks with an ML-based approach
- ❖ Extending our work to the edge
 - Port compute request functionality to extreme LP devices
 - Managed function execution across a hierarchical ecosystem
- ❖ Large scale deployments
 - Using resource providers, test Delta+'s capabilities at 1,000+ endpoint scale
- ❖ Automate installation and configuration



Key Takeaways

- ❖ Built a lightweight profiling framework that incorporates cross-platform integration and experiment design.
 - **Achieved ~30% mean error** in an unexplored experiment space
- ❖ Developed a predictive model for AWS spot instance pricing to enable expected cost calculations.
 - **Achieved <5% mean test error** for commonly used instance types
- ❖ Incorporated these elements into a unified framework we call **DELTA+**
 - Demonstrated the ability to learn execution characteristics about a novel computing environment and distribute tasks according to a specified cost-time tradeoff



Relevant Publications

[1] **Baughman, M.**, Haas, C., Wolski, R., Foster, I., & Chard, K. (2018). Predicting Amazon spot prices with LSTM networks. In Proceedings of the 9th Workshop on Scientific Cloud Computing (pp. 1-7).

[2] **Baughman, M.**, Chard, R., Ward, L., Pitt, J., Chard, K., & Foster, I. (2018). Profiling and predicting application performance on the cloud. In 11th IEEE/ACM International Conference on Utility and Cloud Computing (UCC).

[3] **Baughman, M.**, Caton, S., Haas, C., Chard, R., Wolski, R., Foster, I., & Chard, K. (2019). Deconstructing the 2017 changes to AWS spot market pricing. In Proceedings of the 10th Workshop on Scientific Cloud Computing (pp. 19-26).



Relevant Publications

[4] **Baughman, M.**, Chakubaji, N., Truong, H. L., Kreics, K., Chard, K., & Foster, I. (2019). Measuring, quantifying, and predicting the cost-accuracy tradeoff. In 2019 IEEE International Conference on Big Data (Big Data) (pp. 3616-3622). IEEE.

[5] Kumar, R., **Baughman, M.**, Chard, R., Li, Z., Babuji, Y., Foster, I., & Chard, K. (2021). Coding the computing continuum: Fluid function execution in heterogeneous computing environments. In 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) (pp. 66-75). IEEE.

[6] **Baughman, M.**, Kumar, R., Foster, I., & Chard, K. (2021). Expanding Cost-Aware Function Execution with Multidimensional Notions of Cost. In Proceedings of the 1st Workshop on High Performance Serverless Computing (pp. 9-12).



Acknowledgements



Contact—mbaumhman@uchicago.edu

Advisors



Ian Foster



Kyle Chard



Rohan Kumar

Committee



Hank Hoffman



Anna Woodard

Contributors



Ryan Chard



Zhuozhao Li



Yadu Babuji



Logan Ward



Summary

- ❖ Using distributed compute resources is increasingly complex
- ❖ Any attempt at effective and efficient scaling requires automation
- ❖ This process can be distilled into a profiling, prediction, and provisioning cycle
- ❖ Serverless infrastructure enables this model seamlessly and at scale
- ❖ Cost-aware execution increases in importance with diversity of resources





Related Projects

- ❖ **CSPOT (Wolski, 2019)**
 - “Serverless Platform of Things in C”
 - A lightweight serverless framework focusing on extreme low-power, wireless devices
- ❖ **Daleel (Samreen, 2016)**
 - Machine learning techniques for cloud configuration
 - Adaptive deployment and reconfiguration based on user specifications
- ❖ **Cherrypick (Alipourfard, 2017)**
 - Instance type selection and cloud configuration
 - Focus on big data and HPC-style workloads
- ❖ **DELTA (Kumar, 2021)**
 - Minimizes execution time through intelligent task balancing and distribution
 - Automating task distribution using serverless
- ❖ **Scrimp (Chard, 2017)**
 - Profiling, predicting, and provisioning for the AWS spot market
 - Focus on large scale science on the cloud
- ❖ **Amazon Resource Optimization (AWS, 2019)**
 - Recommends resources based on historic executions



The Importance

- ❖ Open science outfits generally have a very limited budget for compute
- ❖ The **effective** and **efficient** use of resources can increase overall compute-per-dollar by an order of magnitude.
- ❖ Effective refers to the ability of a system to show practical usefulness for a wide variety of cloud computing
- ❖ Efficiency refers to a system's ability to provision resources dynamically, according to the duration and time sensitivity as specified by the user.
- ❖ Adequately determining both effective methods of optimization and the efficient near-optimum thereof can extend the budget of a limited resources “science in the cloud” outfit significantly