Cypress: Input size—Sensitive Container Provisioning and Request Scheduling for Serverless Platforms

Vivek M. Bhasi, Jashwant Raj Gunasekaran, Aakash Sharma,
Mahmut Taylan Kandemir, Chita Das

PennState

ACM SoCC '22
November 8-10, 2022

Emerging Apps for Serverless

The growing popularity of Serverless is being accompanied by an increased variety of apps being deployed on them

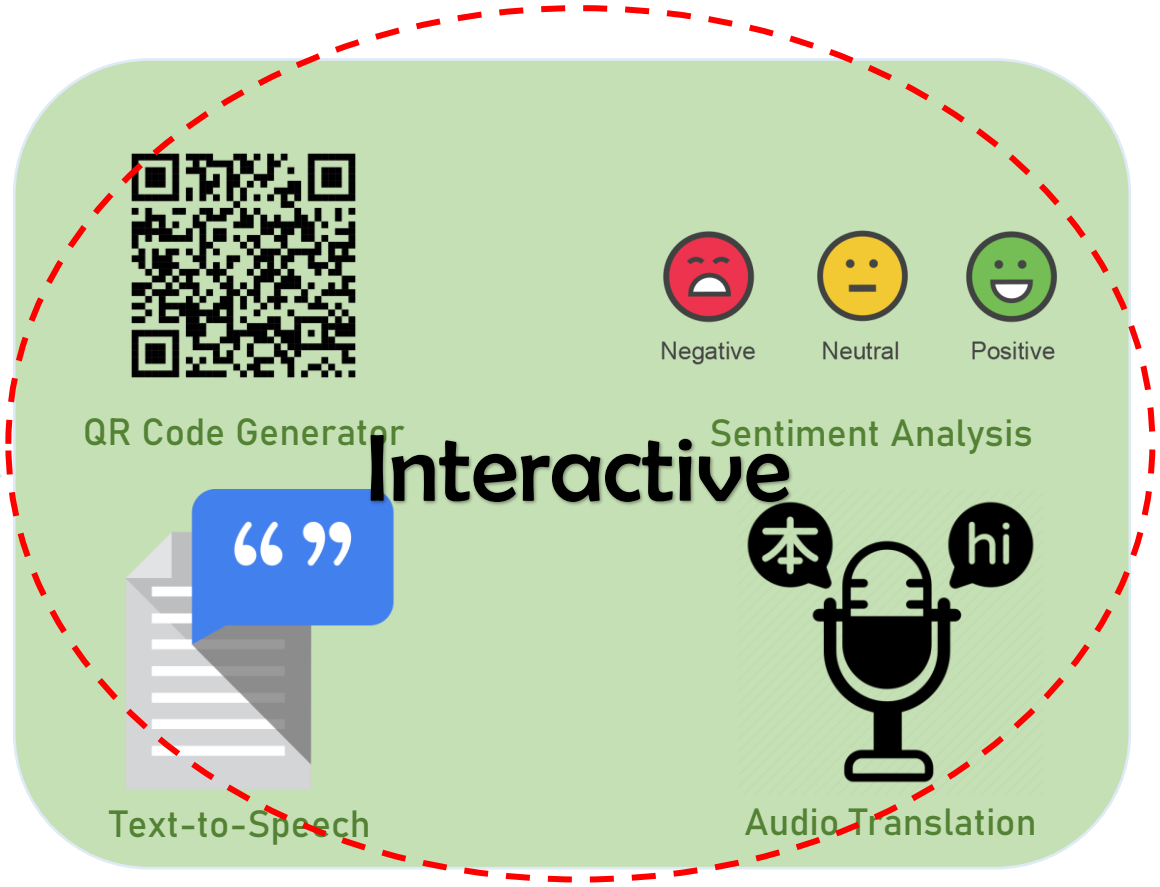
Short-running

Instant Scalability

Fine-Grained Billing

Easy-to-Manage

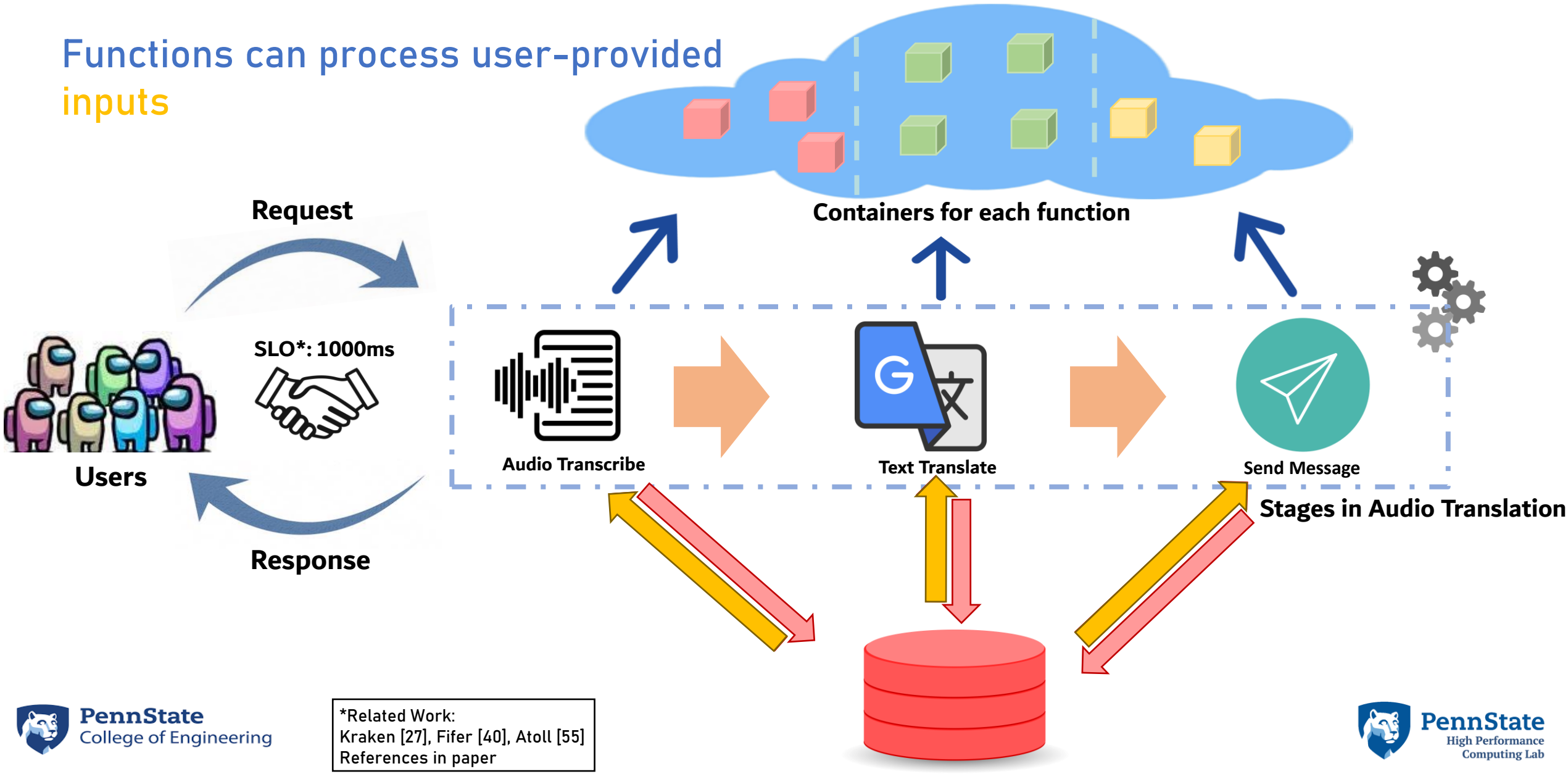
Serverless Characteristics



Target Applications

Serverless 101

Functions can process user-provided inputs



PennState College of Engineering

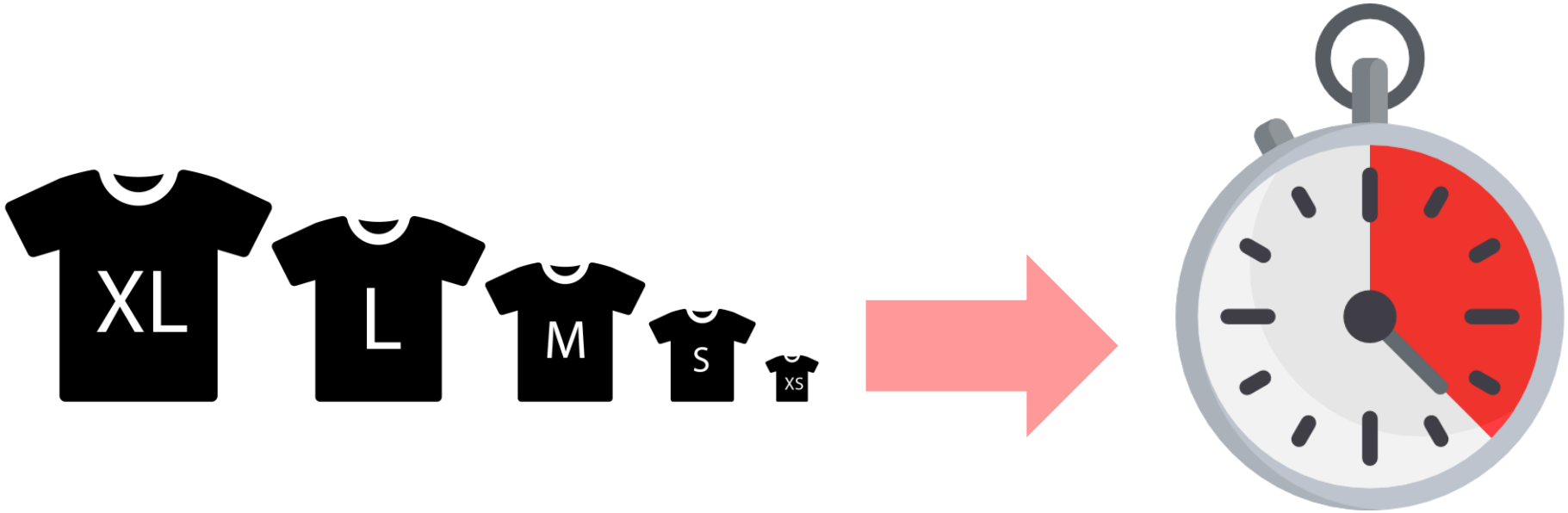
*Related Work:
Kraken [27], Fifer [40], Atoll [55]
References in paper

PennState High Performance Computing Lab

Input Size—Sensitive Functions

- Input-sensitive profiling ➔ exec time depends on input size for such functions

- Such functions: Input size—Sensitive functions (IS functions)

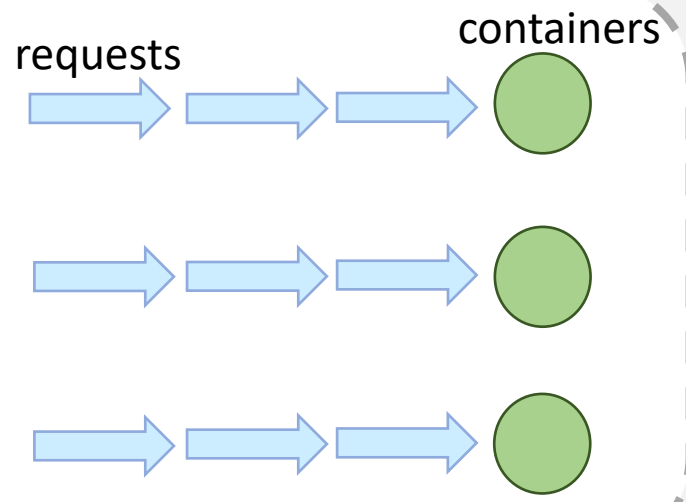


Challenges due to IS Functions

Challenge 1:
Input size—Sensitive Container Allocation

Assumption*: Each request triggers the same execution time for a function;
Hence requests batched uniformly for the same function

$$Batch\ Size\ (f) = \left\lfloor \frac{Function\ SLO\ (f)}{Average\ Execution\ Time\ (f)} \right\rfloor$$

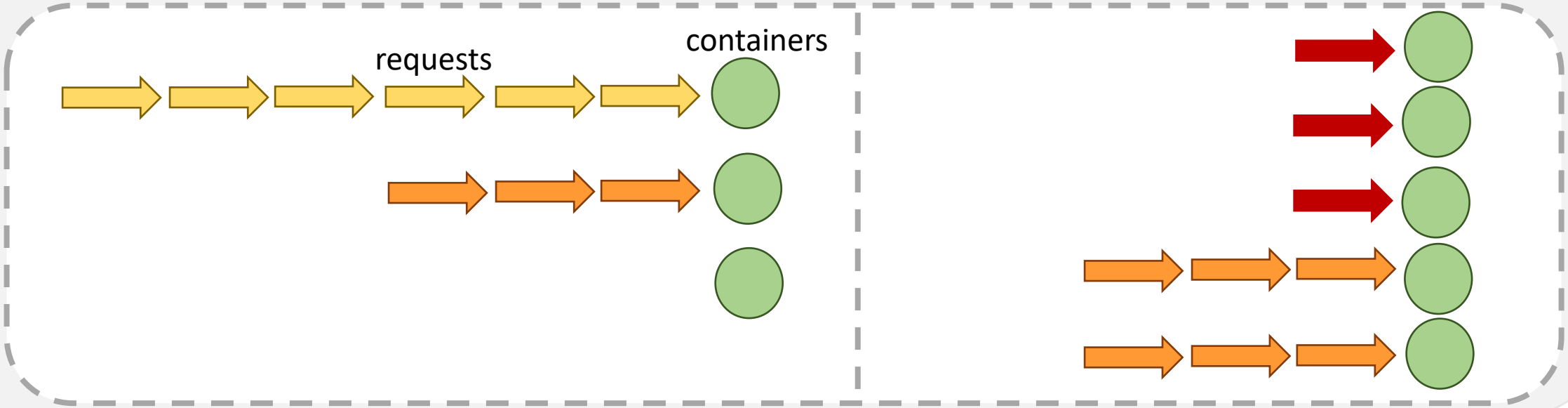


*Related Work:
Kraken [27], Fifer [40]
References in paper

Challenges due to IS Functions

Challenge 1:
Input size—Sensitive Container Allocation

Reality: Each request triggers an execution time dependent on the input size to the function;
Hence requests have to be batched according to actual execution times



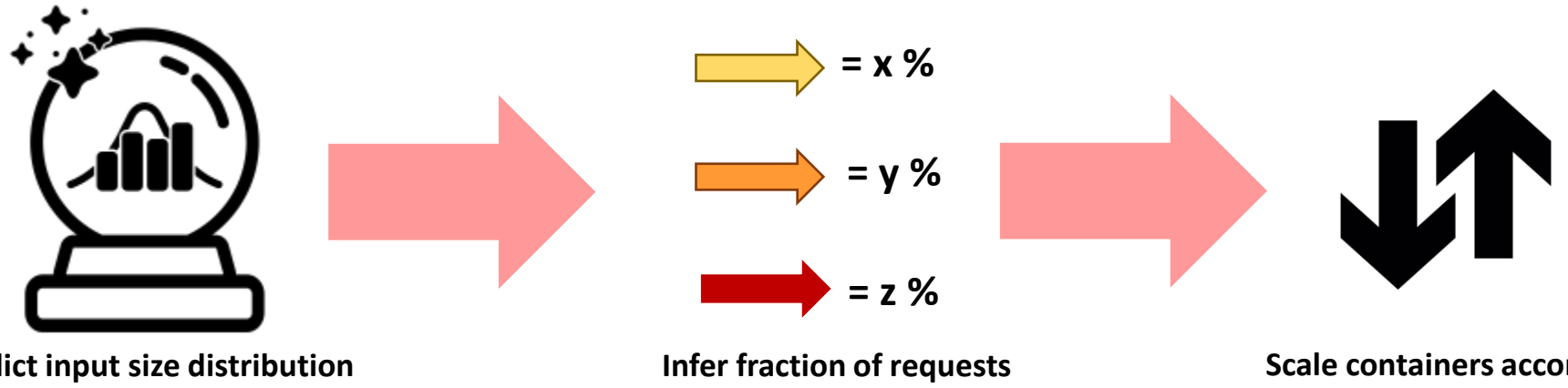
light* medial* heavy*

* w.r.t the relative input sizes associated with the majority of requests

Challenges due to IS Functions

Challenge 1:
Input size—Sensitive Container Allocation

Moreover, input size distribution of future requests has to be predicted as well to provision containers in advance to hide cold starts

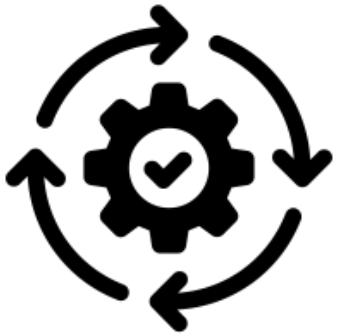


Challenges due to IS Functions

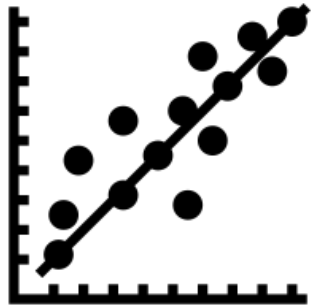
Challenge 1:
Input size—Sensitive Container Allocation

Opportunity 1: Input size—Sensitive policies for predictive scaling and request batching

Input size—Sensitive Profiling (IS Profiling)* can be used to estimate the execution times of functions, given their input sizes



Observe execution time vs input size characteristics of functions over multiple runs for various input sizes



Use linear/powerlaw regression to build a model that predicts the function execution time using input size

PennState College of Engineering

*For further details about IS profiling, please refer the paper

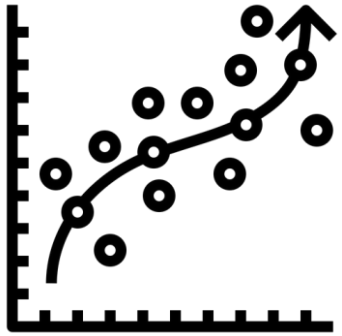
PennState High Performance Computing Lab

Challenges due to IS Functions

Challenge 1:
Input size—Sensitive Container Allocation

Opportunity 1: Input size—Sensitive policies for predictive scaling and request batching

IS profiling can be integrated into policies as necessary to enable input size distribution prediction and Input size—Sensitive Batching (IS Batching) for proactive container allocation



Use a statistical model for input size distribution prediction

+

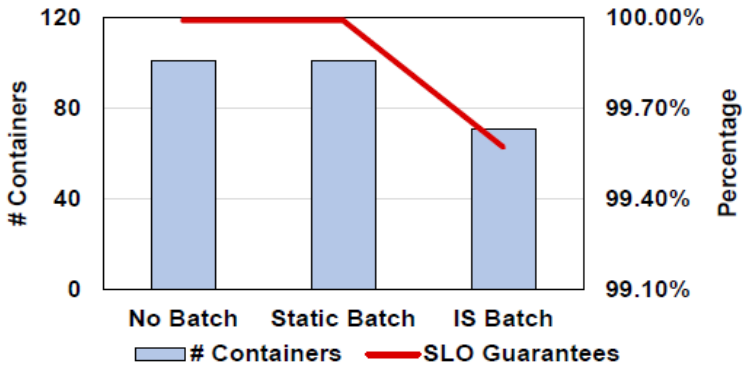
Perform IS Batching akin to First Fit Bin Packing

Challenges due to IS Functions

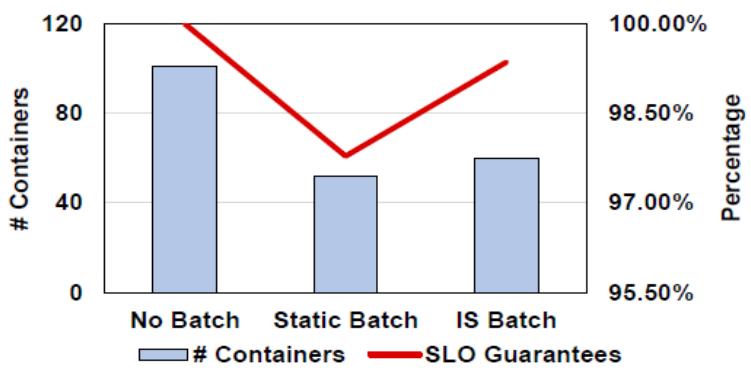
Challenge 1:
Input size—Sensitive Container Allocation

Opportunity 1: Input size—Sensitive policies for predictive scaling and request batching

For motivation, we conduct a mini-experiment comparing IS Batch (our policies so far) vs policies with no request batching and statically-computed batch sizes



(a) Heavy.



(b) Light.

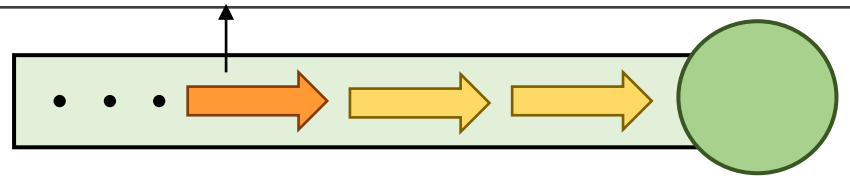
IS Batch is seen to provision the appropriate # containers for both input size distributions for the Sentiment Analysis app under Poisson trace (mean = 100rps)

Challenges due to IS Functions

Challenge 2:
Improving SLO compliance

Although IS Batch allocates the appropriate # containers, the SLO compliance can be improved

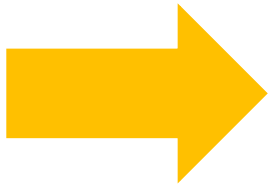
Despite having lesser buffer time to execute, a heavier request is scheduled after a light one in this example



Main Problem: Containers allocated to meet a request's SLO without considering its position in the execution queue

Challenges due to IS Functions

Challenge 2:
Improving SLO compliance



Opportunity 2: Input size—
Sensitive Request Reordering

Now, incorporating IS Reordering into the existing IS Batch scheme to prioritize requests with lower slack, we get the IS (Batch + RR) scheme

IS (Batch + RR) is observed to improve SLO compliance considerably while maintaining the same, minimal # containers. Below is a table comparing SLO compliance for the previous experiment for the light distribution

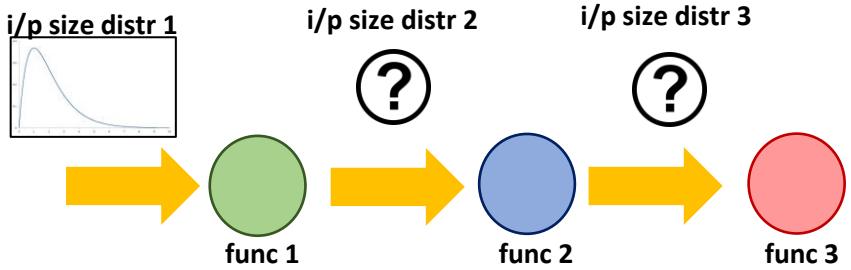
No Batch	Static Batch	IS Batch	IS (Batch + RR)
99.99%	97.78%	99.35%	99.98%

Challenges due to IS Functions

Challenge 3: Multi-function apps

While the policies mentioned so far can be effective for single-function apps, they may not suffice for multi-function apps (composed of 'function chains')

Problem 1: The input size distribution to each function will be different as each function processes its inputs before sending it to subsequent functions. Thus, proactive scaling becomes challenging.



Problem 2: The prediction errors for execution times, request load and input size distributions can accumulate and reactive scaling may not be sufficient to counteract this while ensuring SLO compliance.

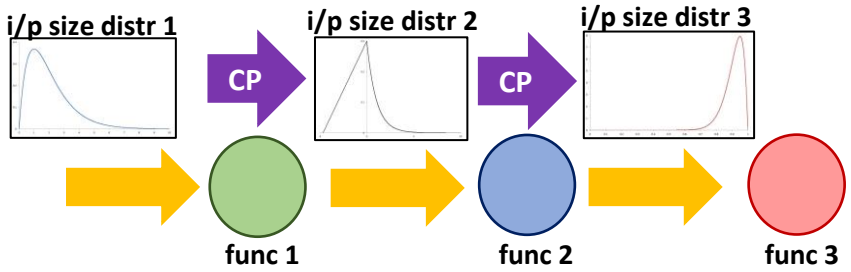
Challenges due to IS Functions

Challenge 3:
Multi-function apps

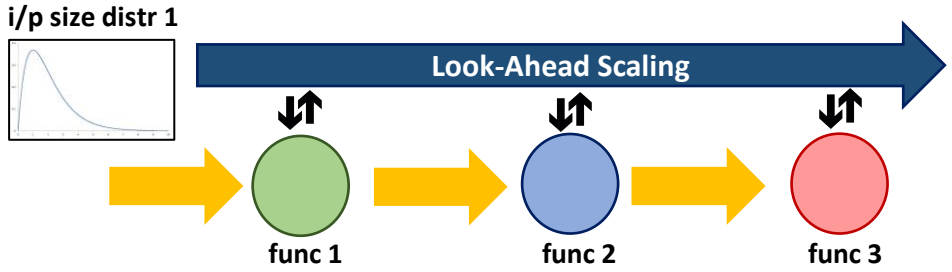
Opportunity 3: Chained Prediction + Look-Ahead Scaling

Introduce policies that are cognizant of particular aspects of function chains

Chained Prediction (CP): Use a variation of IS Profiling to predict the output size distribution of each function given its input. This enables proactive provisioning for multi-function apps.



Look-Ahead Scaling (LAS): Scale containers appropriately for descendant functions as requests arrive at the initial function(s). Thus, enough buffer time for scaling containers in response to the actual i/p size distribution



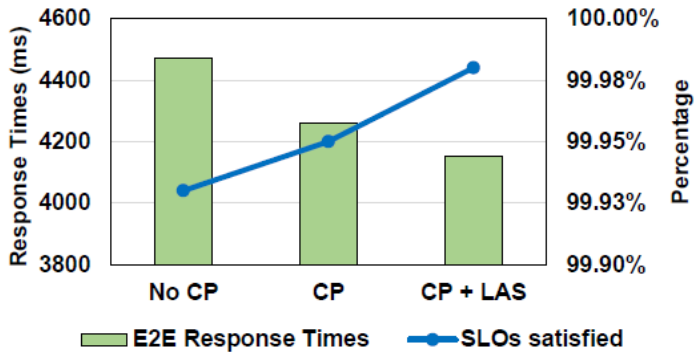
Challenges due to IS Functions

Challenge 3: Multi-function apps

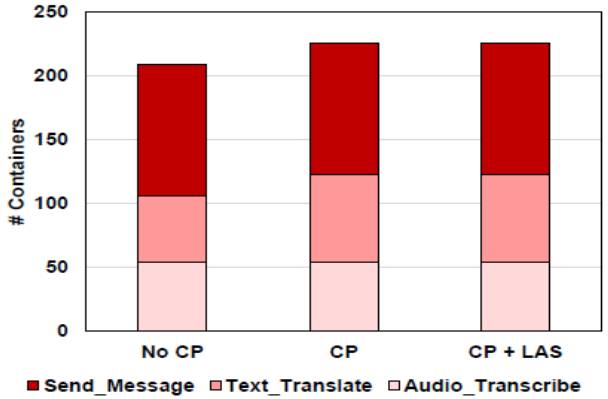
Opportunity 3: Chained Prediction + Look-Ahead Scaling

To demonstrate the benefits of CP and LAS, we conduct a mini-experiment

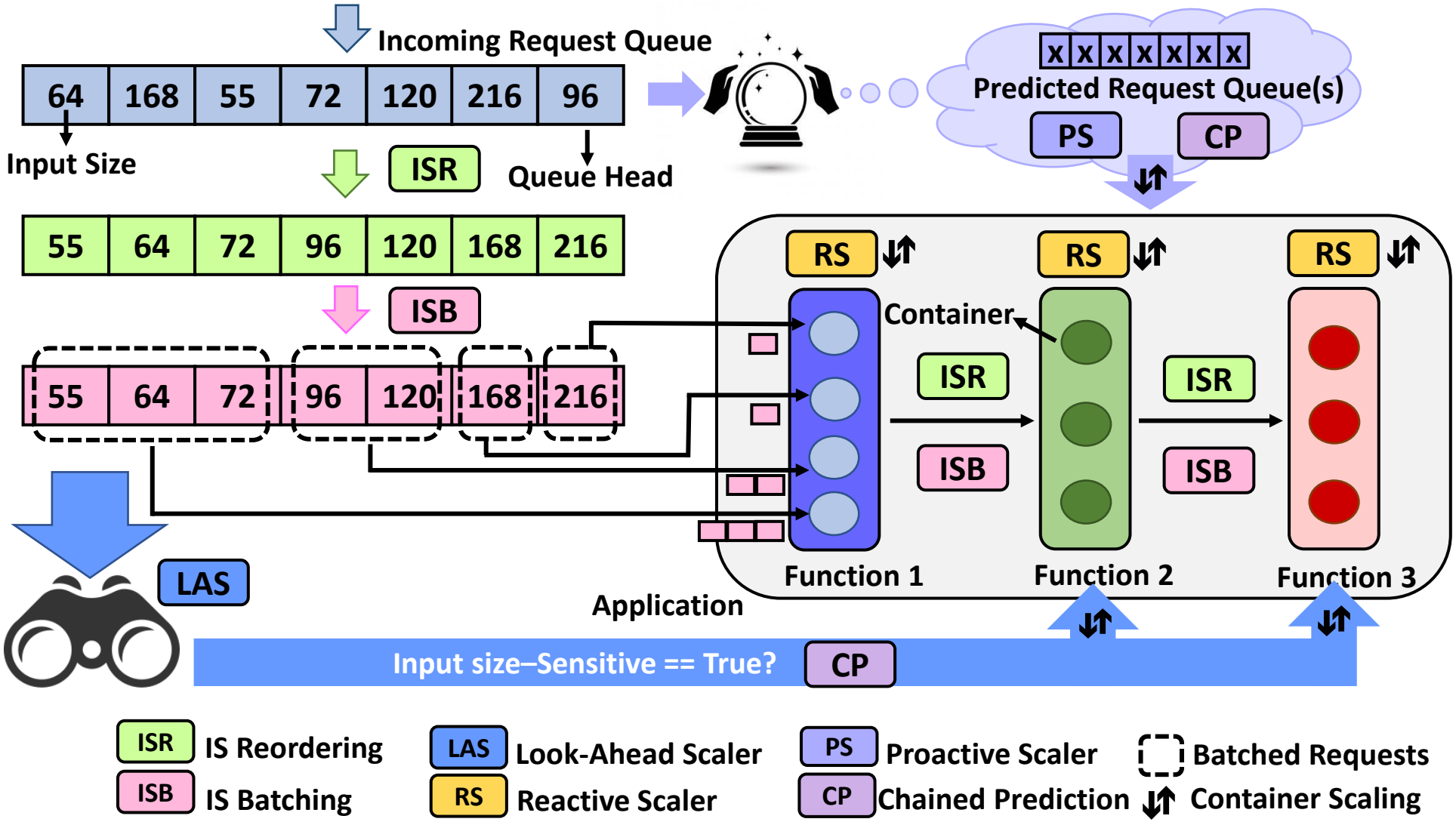
Adding CP and LAS on top of existing policies are shown to improve the E2E response time as well as SLO compliance



Enabling CP allows the appropriate # containers to be spawned for each function. LAS spawns them as requests arrive (not reactively) ➔ improves response time and SLO compliance



Putting it all together: Cypress



Experimental Setup

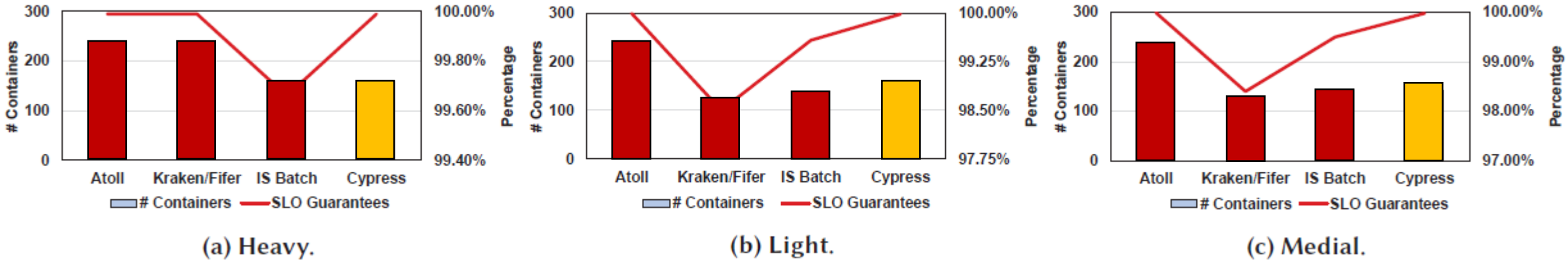
Platform: OpenFaaS + Kubernetes

Hardware: 6 node cluster (Intel CascadeLake with 48 cores, 192 GB RAM and 1 TB storage each) with 10 Gigabit Ethernet

Evaluation Methodology:

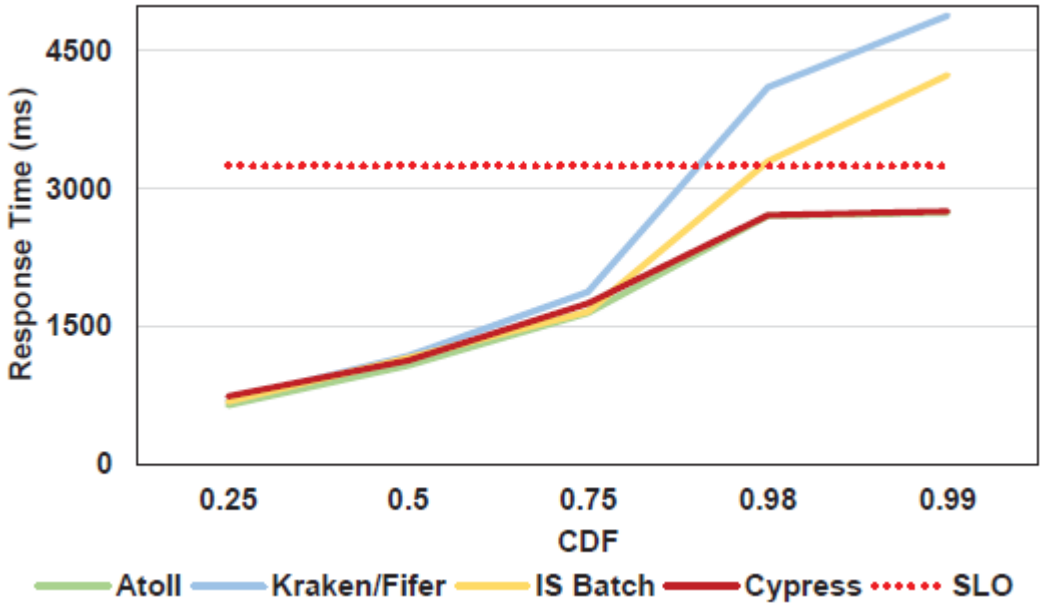
- Traces: Poisson (mean rps = 250 rps) , Wiki, Twitter (peak rps up to 250)
- Apps: Sentiment Analysis, QR Code, Image Compression, Email Categorization, Audio Translation
- SLOs: Maximum execution time (function) + 20%
- Schemes: Atoll, Kraken, Fifer, IS Batch

Evaluation: # Containers vs SLO compliance



- Atoll spawns the most containers regardless of i/p size distribution due to the lack of request batching
- Kraken and Fifer either over-provision or under-provision depending on the i/p size distribution due to request batching using avg. function execution times
- IS Batch spawns the same # containers as Cypress but has lesser SLO compliance due to it not having IS Reordering, LA Scaling
- Similar patterns are seen for multi-function apps as well

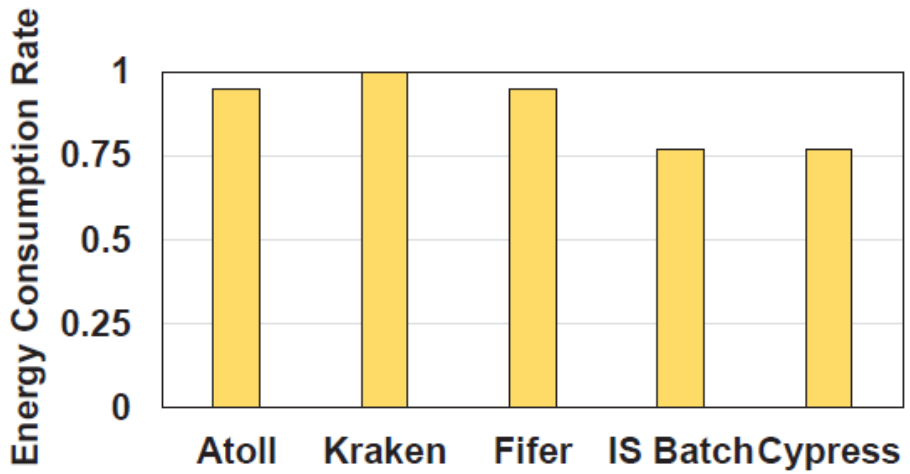
Evaluation: Response Time Distribution and Resilience to Erratic Traces



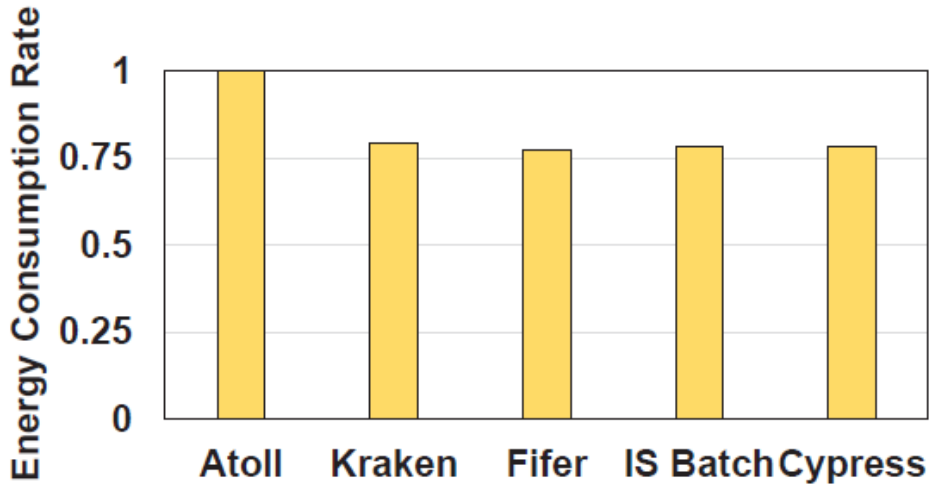
Distribution	Scheme				SLO
	Atoll	*Kraken/Fifer*	*IS Batch*	*Cypress*	
Heavy	5256	5260	5189	2989	3340
Light	5059	6262	5022	2889	

- Cypress remains well within the SLO for the majority of application-trace-input distribution combinations at the tail (P99)
- In particular, for the erratic trace (Wiki), we see that Cypress performs better than all other schemes. For this example, it is likely due to IS Reordering

Evaluation: Energy Consumption



(a) Heavy.



(b) Light.

- Here, Cypress uses 19-23% lesser energy than other Atoll, Kraken and Fifer in the shown example for the heavy distribution
- Note that for the light distribution, although Kraken, Fifer may spawn lesser containers than Cypress, the energy difference between them is 1-2% only

Concluding Remarks

- Adopting the serverless platform for input size–sensitive apps introduces critical request scheduling and resource management challenges for the provider
- To address these, we introduce Cypress, an Input size—Sensitive RM framework
- Cypress uses various scaling services that leverage its policies to spawn up to 66% fewer containers, thereby improving cluster-wide energy savings up to 2.95x while remaining highly SLO compliant

Thank You!