Costless: Optimizing Cost of Serverless Computing through Function Fusion and Placement

Alina Saduova, Brian Wolk,
Richard Bankhead

Costless: Optimizing **Cost** of Server**less** Computing through Function Fusion and Placement

Tarek Elgamal[†], Atul Sandur[‡], Klara Nahrstedt[¶], Gul Agha[‖]
Department of Computer Science, University of Illinois, Urbana-Champaign
Email: [†]telgama2@illinois.edu, [‡]sandur2@illinois.edu, [¶]klara@illinois.edu, [‖]agha@illinois.edu

- Presents algorithm to optimize price of serverless applications in AWS Lambda
- Three factors affecting price are described
- Threshold held on latency
- Algorithm allows for exploration of different function fusion-placement solutions
- Achieved cost optimization of over 37-57% with latency increases of 5-15%
- Link to paper: https://arxiv.org/pdf/1811.09721.pdf

How prices are calculated in AWS Lambda

1) Number of times function is executed per month
2) Memory allocated to the function
3) The runtime of the function
4) The price per 1 Gig of memory and 1 Second (Gb-s)

Factors Affecting price in AWS Lambda

1) Fusing a sequence of functions
2) Splitting functions across edge and cloud resources
3) Allocating the memory for each function

Fusing a sequence of functions

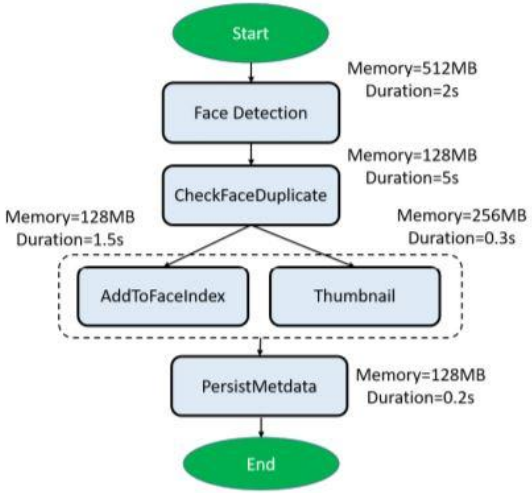


Fig. 1: Example AWS workflow (state machine)

Number of state transitions

- A state machine can be created to let one Lambda function do the work of two or more while avoiding the cost of transitioning between the different Lambdas
- Example: Face Detection and CheckFaceDuplicate can be fused in example workflow
- However because of the need to memory allocate 512 MB across both functions, the cost saved on state transitions is outweighed by the additional memory allocation in this case
- Conclusion: It is not trivial to decide which functions to fuse, Could have implications on price and latency of the fused functions

Splitting functions across edge and cloud resources

Edge vs. cloud computation

- Computing functions on edge devices can be cost effective because charge is per-device, no matter how many functions are run on the device
- Edge device usually communicates with cloud through Amazon S3 - additional cost
- Latency increase for transition from edge device to cloud
- Desirable to place the functions that reduce transmission time on edge devices
- In model, computation and transmission times considered and best placements chosen with respect to price and latency

MODELS

Resource model:	Data model	Workflow model
• **Edge - device close to the data source *(price for connecting is p_e/mo)***	• **JSON format data**	**Directed acyclic graph $G_f = (V_f, E_f)$, where $Vf = \{ fi \mid i = 1...n\}$ and $Ef = \{ fi \rightarrow fj \mid i = j , 1 \leq i, j \leq n\}$**
• **Cloud - set of user-defined functions $\{fi \mid i = 1...n\}$ with memory allocation of $m_{i,c}$**	• **If binary (i.e. compressed image) → persistent storage + JSON encoding**	$f_i \rightarrow f_j$ means fi is executed before f_j and the output of fi is the input of f_j

MODELS cont.

Price model:

$$P(G_f, X_{i=1,..,n}) = \sum_{i=1}^{i=n} X_i \cdot r \cdot e_{i,C} \cdot m_{i,C} \cdot p_{m_{i,C}} + r \cdot (n+1) \cdot p_s + p_E$$

- X_i is 1 for Cloud and 0 for Edge
- Entire workflow of functions is executed for r times
- The number of transitions for n functions is n+1

Execution Time Model

Total execution time:

$$T(G_f, X_{i=1,..,n}) = t_n(G_f, X_{i=1,..,n}) - t_0$$

Execution time of function f_i:

$$t_i(G_f, X_{i=1,..,n})) = \underbrace{t_{i-1}(G_f, X_{i=1,..,n})}_{\text{completion time of prev. function}} + \underbrace{(1-X_i) \cdot e_{i,E}}_{\text{Execution time on edge}} + \underbrace{|X_i - X_{i-1}| \cdot tr(E \xrightarrow{D_{f_{i-1}}} C)}_{\text{Transmission time}} + \underbrace{X_i \cdot (e_{i,C} + s_{i,C})}_{\text{Execution time on cloud}} \quad (5)$$

is a transmission time of intermediate data

Function Profile

Profile of function includes:

1) Execution cost $e_{i,E}$ *or* $e_{i,C}$
2) Transmission time:

$$tr(E \xrightarrow{D_{f_{i-1}}} C)$$

where D_{fi} is size of daata and B is bandwidth from E to C

3) Allocated memory $m_{i,C}$
4) Scheduling delay $s_{i,C}$

Problem definition

Let G f = (V f ,E f) be the new function graph after function fusion, where

$V_f = \{ f_i | \ i = 1...m\}$ and $f_i = f_1 \mid f_2 \mid f_3 \mid \ldots$

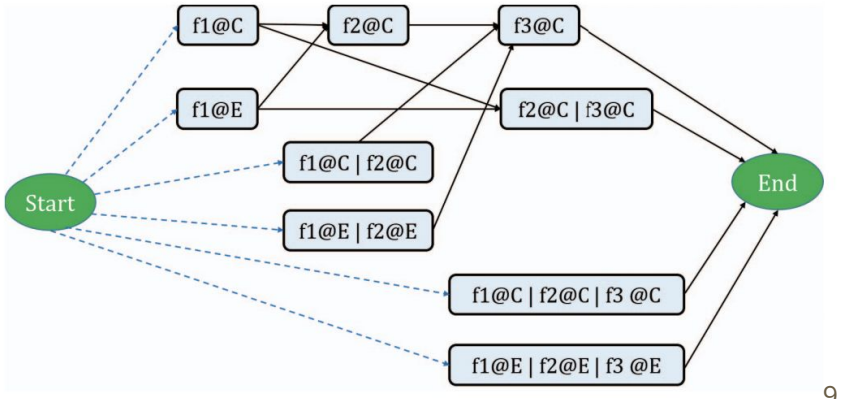
Let X i be the placement variable
$\{X_i \mid i = 1...m\}$

We define the cost optimization problem as **finding the fused graph G_f and the placement variables $\{X_i \mid i = 1...n\}$ such that the price P is minimized and the execution time does not exceed a certain threshold T_{thresh}**

Proposed approach

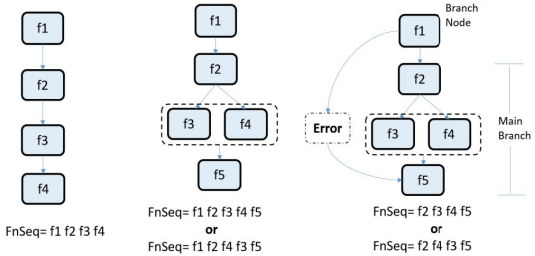
Feasible Solutions: Each solution requires deciding which functions to fuse if any (Function fusion) and assigning each fused function to E or C (Function placement).

Cost Graph Representation:



Algorithm

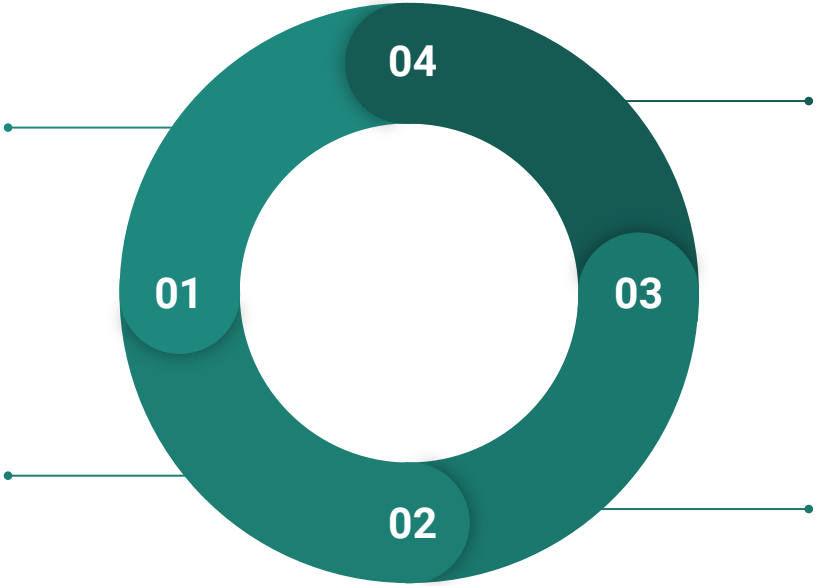
Create an intermediate form of different workflow types:



Construct cost graph:

We define a function L(f') that denotes the set of possible placements of function f' .

$$L(f' = (f_1\ f_2)) = \{(f_1@E \mid f_2@E), (f_1@C_{m_1} \mid f_2@C_{m_1}), (f_1@C_{m_2} \mid f_2@C_{m_2})\}$$



Solve the CSP problem:

The main idea behind LARAC algorithm is to apply Dijkstra's shortest path algorithm on an aggregated cost $c_{uv}/c^*+\lambda d_{uv}/d^*$ that includes both the price and the delay values

Add cost graph links:

We add links for all nodes:
1 - From START to first functions
2 - last function in FnSeq to the END node
3 - between intermediate nodes, where j is the successor of i in FnSeq

Evaluation

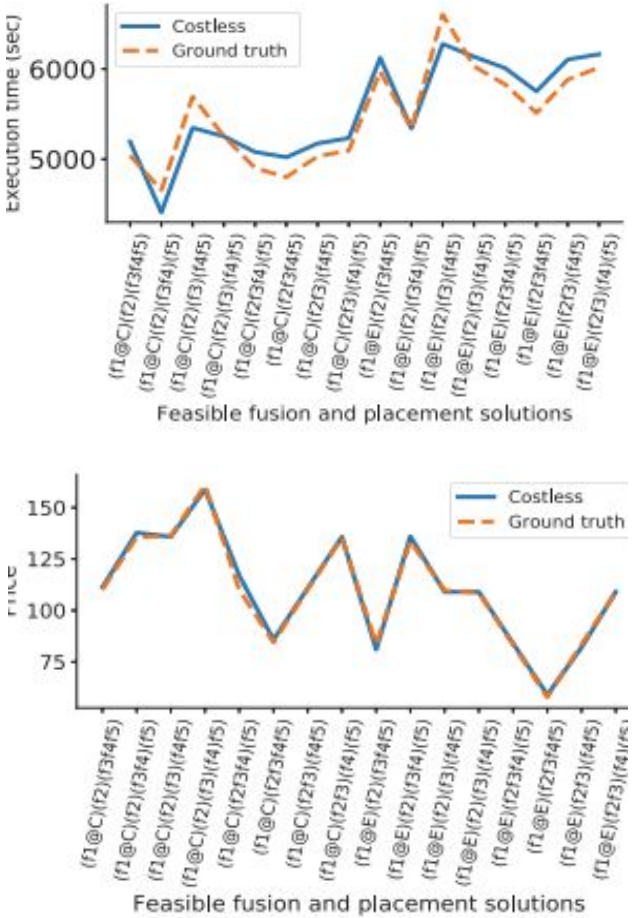
- Goal: Show that Costless is an effective means of choosing configuration for function placement, fusion and memory to reduce FaaS costs
- Method: compare Costless results against against real-world data
- Used small image processing app to explore the effects of function fusion
  - 5 functions, two of which are run in parallel - same structure as displayed in slide (X)
- Table shows data for each function when run without any fusion:
  - Averages are calculated over 20 runs

Function	Avg. exec. time [128 MB / 256 MB / Edge]	Avg. scheduling delay	Max Memory used	Avg. billed duration [128 MB / 256 MB]
f1	893 ms / 772 ms / 1870 ms	61 ms	42 MB	955 ms / 822 ms
f2	970 ms / 743 ms	52 ms	38 MB	1016 ms / 800 ms
f3	2063 ms / 1080 ms	172 ms	83 MB	2116 ms/1144 ms
f4	844 ms / 735 ms	153 ms	37 MB	883 ms/788 ms
f5	153 ms / 101 ms	67 ms	38 MB	211 ms/144 ms

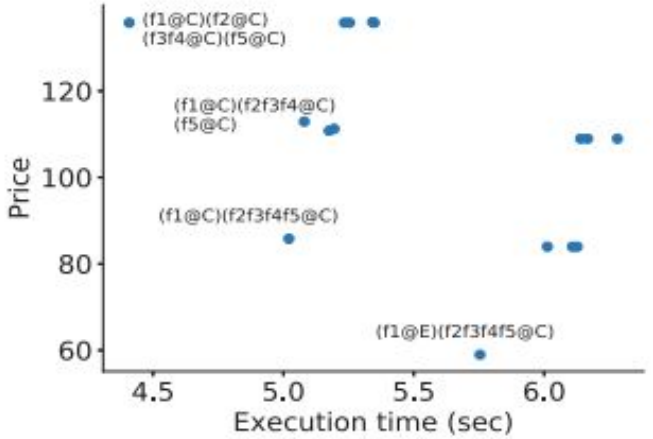
Evaluation - Model Accuracy

- Compared model estimates (Costless) against live data for each possible combination of fused/non-fused functions (Ground truth)

- Average error for model compared to actual shows that Costless is very accurate
  - Considered all possible combinations of fused and non-fused functions
  - Results within 1.2% of model for price and 4% execution time!



Evaluation - Results



- Most expensive option is the fastest
  - Original configuration! Fast because it leverages parallelism in functions 3 and 4 but expensive due to many function transitions
- Slightly cheaper option fuses one additional function from previous config
  - Limiting the number of fusions keeps the code modular and maintainable
- Inexpensive option which is still quite fast
  - Same as cheapest option but does not use edge device
- Least expensive option is less than half the price of the most expensive
  - Places f1 on edge device and fuses all of f2, 3, 4 and 5 to reduce function transition cost

Evaluation: Conclusion

- Function fusion is an effective way to reduce cost
  - Especially in situations where FaaS pricing is dominated by transition cost
- Choosing which functions to fuse together in a FaaS application is a non-trivial problem
- The Costless algorithm was developed to solve this problem by transforming the problem into a constrained shortest path problem
- Algorithm was able to find a function configuration that reduced cost by 37% while increasing latency only 5% on an image processing app
  - Price reduction can be increased to 57% when functions are placed on edge devices

Critique: Strengths

- Model produces very accurate results compared to the real-world tests
- Costless is a useful way to demystify the billing model of FaaS

Critique: Weaknesses

- Modularity of code is an important factor in program maintenance which is mostly ignored by the authors in favor of more quantifiable statistics

Critique: Evaluation

Pros:

- Authors make a strong case for the importance of fusion
- Clear benefits to examining configuration options for cost:performance analysis
- Algorithm runtime is a huge improvement over a brute-force alternative

Cons:

- Scalability listed as a benefit to Costless but the evaluation only covers a small 5 function example
- Perhaps outside the scope of the paper, it would be interesting to learn how well Costless would translate to cloud platforms outside of AWS

Questions

Slide for letting the audience ask questions