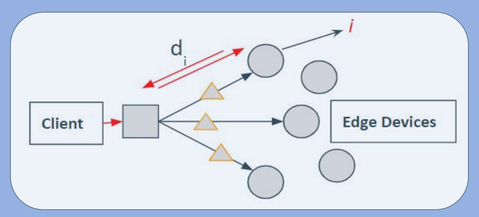
Probabilistic Task Assignment on Edge Computing

TEXAS STATE UNIVERSITY
San Marcos, TX 78666
Computer Science • NSF REU
Edge Computing Summer 2022

Seth Brodzik • Michael Paulson
Contact: Sethmb83@gmail.com • MichaelPaulson91@gmail.com
Faculty Advisor : Dr. Xiao Chen

Overview

- **Overview:**
  - Recently, edge computing and its effective ability to allocate processing resources has attracted a lot of attention from academia and industry
  - Task offloading is one of the larger problems to be solved in edge computing and can be defined as the transfer of resource-intensive computational tasks to an external, resource-rich platform
  - Being an optimization problem, which involves many uncertain parameters, we decide to adopt a Chance Constrained Method as a powerful paradigm to model our uncertainty
- **The Problem:** *How can we select the best combination of available edge devices to guarantee, with a specific level of confidence and minimum performance output, the shortest possible delay in processing our tasks?*



Task Assignment Optimization Problem

Task Offloading Problem

- We assume that a client needs to choose out of ***n*** edge devices to process tasks, each device with their own associated network delay d_i, work output before battery depletion w_i, and chosen factor (represented by 0 or 1) x_i.
- We aim to minimize delay while also meeting a minimum work threshold ***W*** and a probability ***p***.
- We implement a Brute Force solution to find our ground truth set, and a Binary Sort implementation to benchmark our performance.
  - Brute Force rapidly decreases in performance due to its computational complexity *O(2ⁿ)*, hence the need for a quick and efficient algorithm.
  - Binary Sort is more efficient - *O(n · log(n))* - but does not make use of all the parameters given.

$$\begin{aligned} \underset{X}{\text{minimize}} \quad & D \\ \text{s. t.} \quad & P(\sum_{i=1}^{n} d_i x_i < D) \geq p \\ & \sum_{i=1}^{n} w_i x_i \geq W \\ & x_i \in \{0,1\} \end{aligned}$$

Original Probabilistic Problem

Proposed Algorithm

For the sake of finding a more quantifiable solution, we transform our previously stated problem into a more closed form equation called *P2.* The network delays d_i are rewritten in terms of $d_i = \mu_i$ (average delay) + **A** (probabilistic error constant) • σ_i (standard deviation of delay).

Additionally, a set of ***n*** chosen and unchosen (x_i) edge devices called ***X*** can also be represented as the value ***h***, a weight corresponding to the probabilistic error constant **A** divided by the set's standard deviation σ.

We then find an auxiliary problem to problem P2 which we will call P3. The goal of P3 is similar to that of P2, but it is a linear function of (μ, σ^2) with a coefficient **k** (where k ≥ 0), instead of (μ, σ) with a constant **A**. Here **k** can take on different values and plays an important role in solution searching.

- Our proposed algorithm, *Finding the Minimum Solution* ***(FMS)***, has three critical steps:
  1. Call sub-algorithm *Reduce Search Space* ***(RSS)*** to reduce the search range of device combination possibilities from [0 ∞] to [0 h].
  2. Call Algorithm FEB to obtain all the extreme points in the search range [0 h] in P3
  3. Compare all the X values of the extreme points using the objective function $\mu + A\sigma$ *to get the minimum solution to P2*

Algorithm FMS: Finding the Minimum Solution to $\mathcal{P}2$

Require: Input: μ_i, σ_i, w_i for $i = \{1, 2, \cdots, n\}$, A, W
Output: the minimum solution to $\mathcal{P}2$

1: Call Algorithm RSS to get the search range of $[0\ h]$ in $\mathcal{P}3$
2: Call Algorithm FEP to obtain all the extreme points in the search range $[0\ h]$ in $\mathcal{P}3$
3: Compare all the X values of the extreme points using the objective function $\mu + A\sigma$ to get the minimum solution to $\mathcal{P}2$

Algorithm FMS

$$c = -\frac{\mu_a - \mu_b}{\sigma_a^2 - \sigma_b^2}$$

Cross Point Equation

Results

After running our simulations over 720 times, in our longest test, we returned a 100% match rate when compared to the ground truth provided by the brute force algorithm.

When benchmarking the computational and time complexity against brute force and binary search we often see significant improvement when using the FMS algorithm. Occasionally Binary search will show a larger speedup, but for the more computationally intensive solutions, FMS provides a solution far more efficiently.

Devices	BF(seconds)	BS(seconds)	FMS(seconds)	BS Speedup	FMS Speedup
10	0.02	0.06	0.37	0	0
12	0.02	0.02	0.10	1	0
14	0.06	0.02	0.02	3	3
16	0.24	0.01	0.02	24	12
18	1.02	0.74	0.06	1	17
20	4.46	0.52	0.02	9	223
24	130.73	0.30	0.02	436	6537
30	12762.24	0.01	0.02	1276224	638112
2000	N/A	2132.49	1.53	N/A	N/A

Speed up table for various device numbers

Conclusion

- The Proposed FMS Algorithm, along with P2 and P3, prove to efficiently solve our Task Offloading Problem by providing the optimal solution with minimal effort. Even under extreme circumstances, such as **n** = 2000 devices and a 0.99 confidence interval, our result is always found in seconds.
- Our 100% match rate may be too good to be true. No flaws were found upon further analysis of our implementation, however, if given more time we would continue to investigate the potential issue.
- Further research may involve deducing potential areas of oversight, altering the simulation parameters, optimizing the Finding Extreme Points Algorithm, or including other edge device parameters such as network upload speed.