CS466: Cloud Computing Assignment Report

Title: A Matching Theory Framework for Tasks Offloading in Fog Computing for IoT Systems

---

Submitted by

Feyaz Baker
181CO119

Aditya Chirania
181CO104

Anusha P Das
181CO108

Table of Contents

Important Links:

1. The proposed algorithm has been implemented in Google Colab and can be viewed through this link.
2. The source code and readme had been uploaded in GitHub and can be accessed here.
3. Link to the assigned research paper.

Paper Summary

Our paper "A Matching Theory Framework for Tasks Offloading in Fog Computing for IoT Systems" details a matching theory framework to send tasks from different edge devices to Fog Nodes for processing. Fog Computing is the constraint where you have limited geographical area, with two classes of IoT devices. You have Edge Devices, which are thought of as interfaces that generate problems which require computation to solve. Since Edge Devices are usually lightweight computing devices, they rely on connections to the second class of device, which is Fog Nodes.

Fog Nodes are heavy systems capable of receiving a problem that an Edge device transmits. Fog Nodes are used to perform the computation required, and then they send back the result to the Edge Device. The constraints in this scenario are as follows:

- A Fog Node can have a preference towards a particular task: for example some Fog Nodes are good at Floating Point Operations and are thus suited for heavy convolutions or neural network training, but some other devices are fast at Integer Operations which make them good for fast image transforms. There could be dedicated hardware architectures that make some classes of tasks easier than others.

- There are a fixed number of task types, and each task that an Edge Device sends will be one of these task types.

- An edge device can have a preference towards a Fog Node. Some Edge Devices may just want the computation done quickly, so they opt for the best system for the job, or other Edge Devices may have slow transmitting rates, so they prefer a Fog Node closer to it than one far away.

- These Fog Nodes and Edge devices may be placed randomly at any point in the constrained area. This is done to accurately simulate real life situations where optimal placement for ensuring closeness to Fog Nodes cannot be guaranteed.

- Each Edge Device can send only one task. It is assumed that while each Edge Device can compile its task into a single packet of information that needs to be processed, as well as that the packet of information will be only one specific task type.

- Each Edge Device task has a fixed number of instructions, which is known to both Edge Device and Fog Nodes. With this information, and the specific

architecture of the Fog Node, it is possible to find the time it would take to execute the specific task at a Fog Node.

- Each Fog Node has a hard limit on the number of tasks it can process in parallel. This limit doesn't specify whether we can hot-swap tasks or if we have to wait until a whole batch is processed before giving it more tasks.

- Both Fog Nodes and Edge Devices can send requests that are nearly instantaneous, but the time to send data for processing and after processing; is a function of their distance of separation. These request can be from the Edge Device to the Fog Node, querying how much more time the Fog Node will be busy for, or from the Fog Node to query how many instructions the Edge Device task has.

This creates a matching theory problem, where each Edge Device has a preference order of Fog Nodes as a function of data transit time and the time when the Fog Node is next free, and each Fog Node has a preference for every Edge Device (and by extension, their tasks). This intentionally generic model was used so we could work around the complexities of proprietary data in Fog Nodes and Edge Devices.

Algorithms

1. Proposed Matching Theory Algorithm

The Matching Theory Algorithm that Chiti et al have proposed in this paper can be summarised as follows:

1. For each task, construct a preference ordering of Fog Nodes according to the inverse sum of waiting time before that Fog Node can accept tasks, and the cost of communication link between that Edge Device and that Fog Node. For communication link costs, the concept in the paper seems to be taken from Chen et. al, but since Chen et. al provides a method of comparing across communication media and we have taken all connections to be the same, device to device via cable link, we can convert this cost as an analogue of time taken to cover this distance in a CAT5 cable.

2. For each Fog Node, construct a preference ordering of Edge Devices according to the inverse sum of waiting time before

that Fog Node can accept this task, the cost of communication link between that Edge Device and that Fog Node, and the time it would take to execute that task. The paper specifies the use of a Cache hit or miss time for each Fog Node, for each Task Type. But it also doesn't specify any hard values, anywhere. The paper doesn't define the total time for executing a Task at a Fog Node, only that it is known. We have extrapolated from the existence of a fixed number of instructions, and different processor capacities of different Fog Nodes, that the intended method was to simulate the time for executing a task as a function of number of instructions in that task and the time per instruction of that Fog Node. We implicitly assume that each instruction takes only one clock cycle, and that other time corrections would come from the cache hits and misses.

3. Then, each Edge Device proposes its task to its preferred Fog Node.

4. Each Fog Node accepts proposals from Edge Devices until its capacity for parallel tasks is filled. Once this capacity is filled, we reject tasks according to the Deferred Acceptance Algorithm.

5. Once the DAA completes processing, each Fog Node will start executing tasks, and can tell each Edge Device about when the Fog Node will be free to accept new tasks. This information is used to create the next batch of preference orderings of Fog Nodes for each Edge Device.

6. With the new preference orderings, we execute the previous two steps for each unallocated Task, until all tasks are allocated.

However, there are some areas where the algorithm wasn't explained completely, and we have made adaptations where needed. The full details are in the Implementation Details section.

2. Greedy Algorithm

The Greedy Algorithm referenced in the paper isn't cited from anywhere, and it only mentions that it prioritises the nearest Fog Node. As such, we have implemented the following algorithm:

1. First find the preference of each Fog Node for each Task, by inverse of distance between them.

2. Find the preference ordering of each Task for each Fog Node, by inverse of distance between them.

3. Then, each Edge Device proposes its task to its preferred Fog Node.

4. Each Fog Node accepts proposals from Edge Devices until its capacity for parallel tasks is filled. Once this capacity is filled, we reject tasks according to the Deferred Acceptance Algorithm.

5. Once the DAA completes processing, each Fog Node will start executing tasks, and can tell each Edge Device about when the Fog Node will be free to accept new tasks. Every time the Fog Node is free to accept tasks, it does so according to DAA and the preference orders generated earlier.

6. With the new preference orderings, we execute the previous two steps for each unallocated Task, until all tasks are allocated.

3. Randomised Algorithm

Once again, a specific paper hasn't been cited in reference to this algorithm. They also don't have a repo for us to examine and evaluate, so we implemented this algorithm:

1. Each Fog Node randomly generates a preference ordering of Tasks.

2. Each Edge Device randomly generates a preference ordering of Fog Nodes.

3. Each Fog Node receives computation requests from Edge Devices according to their preference orderings. And each Fog Node accepts these requests to fill their capacity according to the Deferred Acceptance Algorithm.

4. Every time a Fog Node finishes executing a task, Edge Devices resubmit their requests to it, and Fog Nodes choose their highest preference task. This step repeats until all Tasks have been executed.

Implementation Details

1. Proposed Matching Theory Algorithm

The paper mentions the algorithm vaguely, we had to track other papers that they cited to even understand the terms. The paper mentions using Cache hit and Cache miss times for each type of task, but we haven't implemented this because we would need to simulate the processing of several random instructions to get an accurate measure of the cache hits and misses.

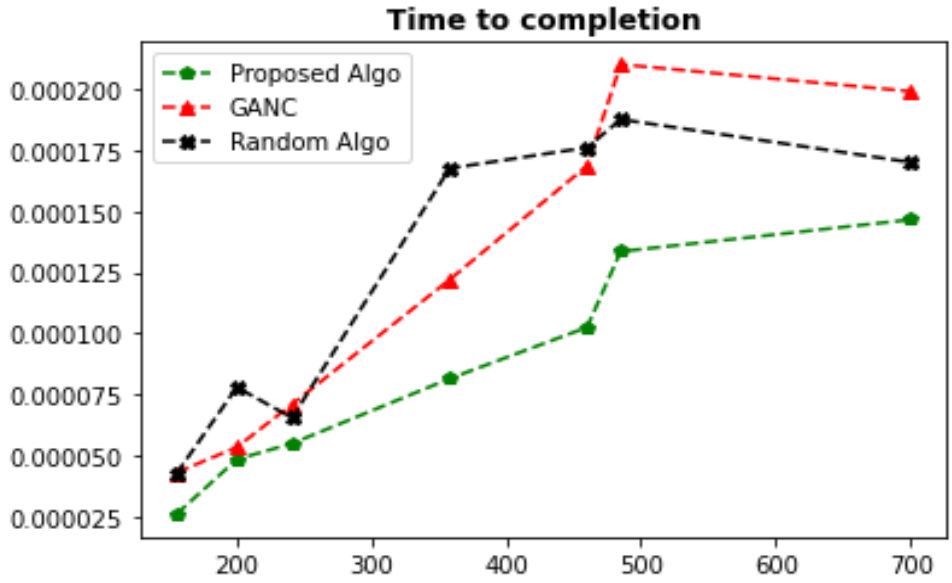
2. Greedy Algorithm

For lack of better sources, we made the algorithm decide the preference lists purely based on the inverse of distance. This way, it technically doesn't count the time needed for execution, but keeps the rest of the process similar so we can compare them more effectively without other optimizations. Given the nature of the algorithm, we didn't have to re-calculate the distances at any point.

3. Randomised Algorithm

The randomised algorithm was implemented with a random shuffle of all Edge Devices and Fog Nodes as applicable. This shuffle was repeated at every point that the preference orders were re-calculated.

Results



Our implementation shows results comparable to those from the paper. Our random algorithm's results actually fluctuate between the GANC and the proposed algorithm, but then again it's supposed to be random.
The proposed algorithm has the best results throughout.

Resources

Original Source Paper

Chen et. al. - Communication costs across media
https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8014294

Deferred Acceptance Algorithm -
https://en.wikipedia.org/wiki/Gale%E2%80%93Shapley_algorithm

Jain's Fairness Index - https://en.wikipedia.org/wiki/Fairness_measure