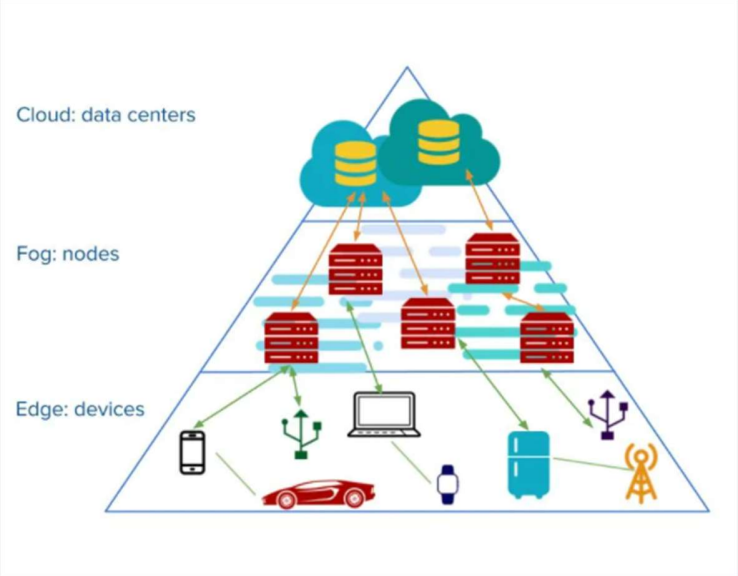
CS528 Course Project

P2: Trust Aware Scheduling of Online Tasks in FOG Nodes
Date: 16 April 2024

Ridhiman Dhindsa, 210101088
Aditya Gupta, 210101009

Introduction

FOG computing is a decentralized computing infrastructure in which data, compute, storage and applications are located somewhere between the data source and the cloud. Like edge computing, FOG computing brings the advantages and power of the cloud closer to where data is created and acted upon. This is often done to improve efficiency, though it might also be done for security and compliance reasons. The FOG metaphor comes from the meteorological term for a cloud close to the ground, just as FOG concentrates on the edge of the network.



Task scheduling in FOG computing involves allocating the tasks in FOG nodes based on factors such as node availability, reliability or shared trust, cost of execution, processing power and network connectivity. Despite having various bio-inspired algorithms for task scheduling in FOG nodes, they still face latency issues since it is an NP-hard problem. Existing task scheduling algorithms modeled by metaheuristic, nature inspired and machine learning techniques address various scheduling parameters such as trust, fault tolerance, cost, energy consumption, execution time etc. Trust and reliability are important factors and hence trust aware task scheduling is necessary in FOG computing systems. Reliability ensures that the system carries out the assigned task successfully with errors or failures. A balance of trust, reliability and fault tolerance produces good quality of service and efficient task scheduling. Therefore, it is important to devise methods to simulate and study trust aware scheduling in online scenarios in FOG systems.

In essence, the challenge is to schedule tasks efficiently across nodes, considering trust levels, cost implications, and reliability requirements, while ensuring tasks meet their deadlines.

Problem Statement

We are given an online stream of tasks associated with different users, and are required to devise an approach to schedule tasks such that it **minimizes cost**, **maximizes reliability** and ensures that **deadlines are met**. Each task has an arrival time, execution time, owner (user who sent the task to the FOG system), deadline and failure probability (FP). The failure probability is modeled as an exponential function of failure rate (f) and execution time of the task.
The FOG system consists of a fixed number of nodes which process each task accordingly. Each FOG node has a direct trust (DT) value from each user and a shared trust (ST) value from the set of all users. ST is the average of all DT values for that node. These trust values represent the probability of successful execution of any task on that node, and are a measure

of reliability of the node. The cost of execution per unit time on a particular node is directly dependent on its ST value. The higher the ST value, higher is the cost of execution per unit time. This means that *executing a task on a more reliable node might ensure its successful completion, but will also cost more*.
After every task execution, DT values must be updated, i.e. increased if it was a successful execution and decreased if it failed. Changes in direct trust will subsequently cause changes in shared trust.
Unsuccessful execution of a task could be due to 2 reasons-

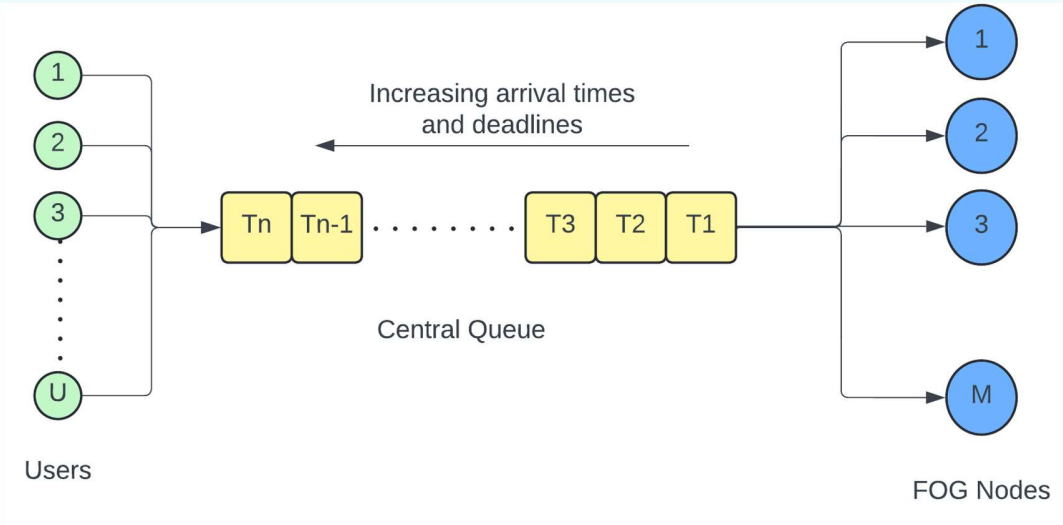
1. Missing the deadline.
2. Failure due to FP. This probabilistic event could occur due to several reasons such as errors, resource timeout, preemption etc.

The aim is to schedule tasks in an online setup, i.e. schedule tasks as and when they arrive. This is different from an offline setup where prior knowledge of the order of tasks might have affected the solution.
Three objectives for the problem:

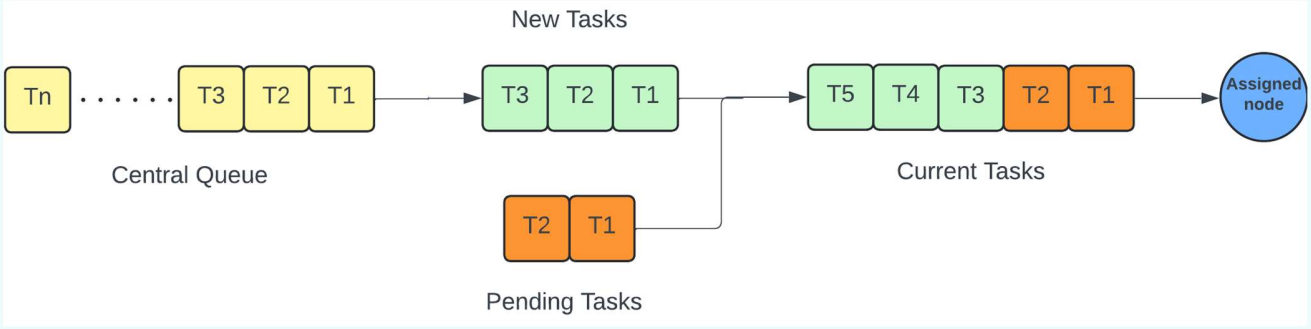
1. Maximize reliability (trust value) of node allocated.
2. Minimize overall cost of execution.
3. Ensure that deadlines are met.

Solution

We can model the above problem as follows. The set consisting of U users who generate tasks in an online manner and these get pushed into a **central queue**. This ensures that tasks are ordered by their arrival times and are served accordingly. This FCFS policy ensures that tasks with earlier deadlines are served first, since each task has the same relative deadline K. Then the absolute deadline will be: $d_j = a_j + K$. Here K is greater than processing/execution times of all tasks.



These tasks need to be assigned to a set of M nodes each having individual trust values. All tasks with the same arrival time are popped from the front of queue and stored in the set of **new tasks**. The set **current tasks** denotes those tasks that must be assigned nodes if any of them are free. If the number of current tasks is more than the number of free nodes, some tasks will be left pending. These are added to the set of current tasks in the next iteration. Therefore, at the beginning of every iteration, **current tasks = pending tasks + new tasks**.



Approach-1: Worst Task to Best Node Matching

Assign task with highest failure probability to most reliable/trusted node.

For assignment of tasks to nodes, we assign the task with the highest FP value to the node with highest reliability. This means we sort the current tasks in decreasing order of failure probability and assign as many as possible to free nodes.
For this, we check all nodes which are currently free, and choose the node with the lowest cost ratio R_c, which is defined as: $\mathbf{R_c = C/ST[m]}$. Here C = cost of execution per unit time, $\mathbf{C = Base + BR*(ST[m])^2}$, m = machine/node index, ST = shared trust of node m. Since we are trying to minimize cost and maximize reliability/trust, we need to minimize R_c. Hence, we choose the available node with lowest R_c.
This should produce a more balanced approach where on average a task fails less often. This gives a balanced distribution of trust values and costs. However, its optimality shall be compared in the testcases.

Approach-2: Best Task to Best Node Matching

Assign task with lowest failure probability to most reliable/trusted node.

For assignment of tasks to nodes, we assign the task with the lowest FP value to the node with highest reliability. This means we sort the current tasks in increasing order of failure probability and assign as many as possible to free nodes.
For this, we check all nodes which are currently free, and choose the node with the lowest cost ratio R_c, which is defined as: $\mathbf{R_c = C/ST[m]}$. Here C = cost of execution per unit time, $\mathbf{C = Base + BR*(ST[m])^2}$, m = machine/node index, ST = shared trust of node m. Since we are trying to minimize cost and maximize reliability/trust, we need to minimize R_c. Hence, we choose the available node with lowest R_c.
This might result in a few node's shared trust increasing continuously but at the same time its cost also increases to very high values. Simultaneously, other less reliable nodes get tasks that more prone to failure and their shared trust keeps decreasing. Overall it represents a skewed distribution whose effect on overall execution cost needs to be studied.

Approach-3: Best Task to Worst Node Matching

Assign task with lowest failure probability to least reliable/trusted node.

For assignment of tasks to nodes, we assign the task with the lowest FP value to the node with highest reliability. This means we sort the current tasks in increasing order of failure probability and assign as many as possible to free nodes.
For this, we check all nodes which are currently free, and choose the node with the highest cost ratio R_c, which is defined as: $\mathbf{R_c = C/ST[m]}$. Here C = cost of execution per unit time, **C**

= Base + BR*(ST[m])² , m = machine/node index, ST = shared trust of node m. Since we are trying to minimize cost and maximize reliability/trust, we need to minimize R_c. Hence, we choose the available node with highest R_c as the worst node.
This represents a scenario similar to approach 1, which strives for a balanced distribution of shared trust and costs.

Testcases and Observations

The parameters to be considered are:

1. U, number of users
2. M, number of nodes
3. f, failure rate
4. K, relative deadline
5. BR, brand constant
6. Total Time ~ 1000s

The attributes checked for are:

7. Net cost of executing all tasks
8. Number of tasks which crossed their deadlines
9. Number of failed tasks due to FP
10. Idle time of the FOG system & percentage utilization
11. Final shared trust of each node
12. Cost and reliability of each task-node matching

Testcases (approach 1):

Case-1: original case (U=M)

U=5, M=5, K=5, f=0.2, BR=10

```
------------------FoG System Report--------------------
Total uptime= 1010.00s
Idle time= 684s
System utilization= 65.80%
Total no. of tasks= 1286
No. of tasks that missed deadline= 18
No. of tasks that failed due to FP= 70
Net cost of executing all tasks= 20647.35

---------------------Final DT and ST values-------------------------------
ST: 0.89 0.80 0.91 0.92 0.93

DT:
0.81 0.66 0.88 0.99 0.96
0.99 0.80 0.96 0.89 0.96
0.81 0.81 0.90 0.98 0.87
0.87 0.81 0.91 0.86 0.99
0.98 0.89 0.89 0.87 0.87
```

Here, the number of users and nodes are kept same. It can be observed that the system utilization in this case as well as all subsequent cases is ~65%, since this quantity depends on the arrival rate of tasks, which corresponds to the mean of the arrival time Poisson distribution.

Case-2: (U>M)

U=12, M=5, K=5, f=0.2, BR=10

```
------------------FoG System Report--------------------
Total uptime= 1010.00s
Idle time= 682s
System utilization= 65.90%
Total no. of tasks= 1342
No. of tasks that missed deadline= 21
No. of tasks that failed due to FP= 71
Net cost of executing all tasks= 17475.47

---------------------Final DT and ST values------------------------------
ST: 0.81 0.84 0.82 0.83 0.78

DT:
0.80 0.92 0.91 0.80 0.89
0.57 0.79 0.78 0.82 0.88
0.87 0.68 0.84 0.87 0.79
0.74 0.86 0.75 0.81 0.80
0.74 0.87 0.85 0.81 0.65
0.90 0.84 0.72 0.80 0.81
0.81 0.92 0.86 0.92 0.79
0.83 0.90 0.85 0.80 0.79
0.93 0.84 0.89 0.85 0.79
0.73 0.81 0.83 0.83 0.85
0.81 0.69 0.73 0.90 0.68
0.94 0.91 0.78 0.80 0.68
```

Here, the number of users is more than number of nodes. In this case, it can be observed that the share trust (ST) values (range: 0.78-0.84) are lower than case 1 (range: 0.80-0.93). This is because there are more users, hence the net direct trust (DT) gets more widely distributed. Also since the number of users are more, the total number of tasks increased from 1286 to 1342. Net cost is lower than case 1 because ST values are lower, and cost depends on ST^2.

Case-3: (U<M)

U=2, M=5, K=5, f=0.2, BR=10

```
------------------FoG System Report--------------------
Total uptime= 1010.00s
Idle time= 687s
System utilization= 65.65%
Total no. of tasks= 1335
No. of tasks that missed deadline= 15
No. of tasks that failed due to FP= 26
Net cost of executing all tasks= 12567.73

---------------------Final DT and ST values------------------------------
ST: 0.79 0.84 0.50 0.50 0.50

DT:
0.77 0.87 0.50 0.50 0.50
0.80 0.81 0.50 0.50 0.50
```

Here, the number of users is less than the number of nodes. In this case, users are less than number of machines, so it can be observed that ST values of 2 machines are high, but the remaining 3 machines remain unused, hence their ST values are 0.5. This phenomenon is observed because in the first iteration the user's tasks are assigned to 2 arbitrary nodes. On successful completion the ST values of these 2 nodes increases. Now in subsequent iterations,

tasks are always assigned to these 2 nodes only due to their high reliability. As a result their trust values keep getting higher and higher while the other nodes remain unused.

Case-4: decrease f

U=5, M=5, K=5, f=0.02, BR=10

```
------------------FoG System Report--------------------
Total uptime= 1010.00s
Idle time= 679s
System utilization= 66.05%
Total no. of tasks= 1323
No. of tasks that missed deadline= 20
No. of tasks that failed due to FP= 111
Net cost of executing all tasks= 19498.87

---------------------Final DT and ST values------------------------------
ST: 0.85 0.81 0.91 0.86 0.82

DT:
0.91 0.80 0.96 0.97 0.90
0.87 0.80 0.87 0.85 0.83
0.85 0.71 0.96 0.77 0.96
0.91 0.93 0.85 0.74 0.75
0.73 0.82 0.92 0.97 0.64
```

Here the failure rate f is decreased. When f is decreased, FP = exp(-ft) will increase. Due to high failure probability, tasks failed due to this will increase, as can be observed by comparing this case with case-4. Here we have 111 (8.4%) tasks failing due to FP, in comparison to 70 (5.4%) in case 1 (total tasks nearly same).

Case-5: decrease K

U=5, M=5, K=2, f=0.2, BR=10

```
------------------FoG System Report--------------------
Total uptime= 1010.00s
Idle time= 677s
System utilization= 66.15%
Total no. of tasks= 1355
No. of tasks that missed deadline= 412
No. of tasks that failed due to FP= 48
Net cost of executing all tasks= 10498.89

---------------------Final DT and ST values------------------------------
ST: 0.40 0.41 0.43 0.39 0.39

DT:
0.52 0.37 0.63 0.32 0.67
0.49 0.55 0.42 0.44 0.46
0.57 0.59 0.30 0.52 0.31
0.27 0.34 0.33 0.25 0.31
0.14 0.22 0.47 0.44 0.21
```

Here the relative deadline K is decreased. When relative deadline K is decreased, more tasks fail due to crossing their deadlines. This represents more stringent conditions. This case has a much higher value 412/1355 (30.4%) compared to 18/1286 (1.4%) in case-1.

Testcases (approach 2):

Case-1: original case (U=M)

U=5, M=5, K=5, f=0.2, BR=10

```
------------------FoG System Report--------------------
Total uptime= 1010.00s
Idle time= 693s
System utilization= 65.35%
Total no. of tasks= 1379
No. of tasks that missed deadline= 14
No. of tasks that failed due to FP= 78
Net cost of executing all tasks= 17348.25

---------------------Final DT and ST values------------------------------
ST: 0.88 0.94 0.83 0.92 0.90

DT:
0.97 0.95 0.75 0.98 0.89
0.87 0.98 0.84 0.97 0.96
0.89 0.88 0.83 0.90 0.83
0.72 0.99 0.87 0.94 0.90
0.96 0.91 0.89 0.79 0.92
```

Here, we keep the number of users and nodes same. It can be compared to approach-1, case-1, where the cost is higher but number of tasks failing is less. In this case the net cost of execution is less. This is because we are mapping tasks with high failure probability to machines with lesser reliability. This increases the risk of a task failing as poor tasks are mapped to poorly functioning nodes. This drives down costs but increases failures. But at the same time, tasks with lowest FP are mapped to nodes with best reliability. This inherently represents a biased distribution since the best performing tasks get the best facilities.

Case-2: (U>M)

U=12, M=5, K=5, f=0.2, BR=10

```
------------------FoG System Report--------------------
Total uptime= 1010.00s
Idle time= 682s
System utilization= 65.90%
Total no. of tasks= 1333
No. of tasks that missed deadline= 9
No. of tasks that failed due to FP= 83
Net cost of executing all tasks= 15656.06

---------------------Final DT and ST values------------------------------
ST: 0.83 0.84 0.85 0.81 0.85

DT:
0.81 0.86 0.85 0.64 0.92
0.74 0.90 0.84 0.90 0.81
0.92 0.90 0.90 0.70 0.90
0.84 0.93 0.86 0.84 0.76
0.81 0.86 0.76 0.85 0.92
0.87 0.85 0.74 0.89 0.88
0.92 0.91 0.89 0.83 0.90
0.80 0.79 0.86 0.73 0.76
0.85 0.80 0.87 0.80 0.91
0.92 0.85 0.84 0.86 0.82
0.79 0.71 0.93 0.80 0.78
0.70 0.78 0.87 0.87 0.84
```

As observed previously, increasing users lowers the average ST values due to wider DT distribution.

Case-3: (U<M)

U=2, M=5, K=5, f=0.2, BR=10

```
------------------FoG System Report--------------------
Total uptime= 1010.00s
Idle time= 684s
System utilization= 65.80%
Total no. of tasks= 1393
No. of tasks that missed deadline= 8
No. of tasks that failed due to FP= 46
Net cost of executing all tasks= 7129.24

---------------------Final DT and ST values-------------------------------
ST: 0.91 0.88 0.50 0.50 0.50

DT:
0.90 0.95 0.50 0.50 0.50
0.92 0.80 0.50 0.50 0.50
```

As observed previously, ST values of only 2 machines become higher and higher while the remaining nodes are unused.

Case-4: decrease f

U=5, M=5, K=5, f=0.02, BR=10

```
------------------FoG System Report--------------------
Total uptime= 1010.00s
Idle time= 683s
System utilization= 65.85%
Total no. of tasks= 1331
No. of tasks that missed deadline= 17
No. of tasks that failed due to FP= 119
Net cost of executing all tasks= 16884.81

---------------------Final DT and ST values-------------------------------
ST: 0.84 0.83 0.89 0.89 0.90

DT:
0.92 0.89 0.99 0.82 0.90
0.67 0.86 0.94 0.98 0.98
0.80 0.78 0.93 0.97 0.86
0.96 0.72 0.71 0.79 0.82
0.84 0.90 0.91 0.90 0.96
```

The number of tasks failed due to FP increase in comparison to case-1.

Case-5: decrease K

U=5, M=5, K=2, f=0.2, BR=10

Here too, the number of tasks that miss their deadline are more.

```
------------------FoG System Report--------------------
Total uptime= 1010.00s
Idle time= 677s
System utilization= 66.15%
Total no. of tasks= 1340
No. of tasks that missed deadline= 261
No. of tasks that failed due to FP= 60
Net cost of executing all tasks= 10884.22

---------------------Final DT and ST values-------------------------------
ST: 0.61 0.33 0.61 0.57 0.61

DT:
0.44 0.36 0.39 0.75 0.51
0.71 0.57 0.80 0.39 0.57
0.70 0.16 0.49 0.76 0.75
0.66 0.33 0.57 0.35 0.48
0.55 0.24 0.79 0.60 0.71
```

Conclusion

From the testcases of both the approaches used above, we can conclude that approach-1 yields a schedule whose overall execution cost is higher. But the number of tasks failed due to FP are lesser in approach-1. Approach-2 produces an overall lower execution cost but the number of tasks failed due to FP are higher.
This inherently represents **a tradeoff between cost of execution and number of task failures**. If the user wants lesser task failures, he'll have to pay more in terms of cost of execution. On the other hand, if the user wishes to lower the cost of execution, he'll have to compromise on the reliability of task execution.
Approach-1 is suitable if the user wishes to maximize reliability and has no cost constraints. This works since we are scheduling the worst performing task on the most reliable node, striving for a more equitable distribution of trust. **Approach-2** is recommended if the user wishes to lower the cost as much as possible. This works on the principle of scheduling the best performing task on the most reliable node. It aims at provisioning a few nodes with the very best reliability, but at the cost of other nodes performing poorly. Hence it represents a skewed distribution of trust. **Approach-3** can be expected to perform similar to approach-1 since it schedules the best performing task on the least reliable node. This is equivalent to applying the order imposed by approach-1 in reverse. This approach also must produce an equitable distribution of trust and consequently, higher costs. But there is a possibility of lesser task failures as observed in approach-1.
The **uses cases** for this kind of trust aware scheduling in an online setting are numerous. For instance, it can be used in dynamically provisioned HPC systems and by cloud computing systems such as AWS, Microsoft Azure, Google Cloud etc. wherein 2 types of pricing plans can be provided to customers. One, a higher pricing model, which assures higher reliability and lesser task failures, ideal for safety critical workflows. The other, a more economically priced model which costs relatively lesser, but might compromise on the success of task execution in large samples. Of course, the parameters can be dynamically changed as per the user's requirements. For e.g. the brand constant BR can be increased/decreased appropriately to control the pricing model for cloud computing resources lent to customers.