Multiprocessor Scheduling, Hu's Algorithm, and Application

ABDULLAH AL FORKAN
Electronic Engineering
Hochschule Hamm Lippstadt
Lippstadt, Germany
Abdullah-al.forkan@stud.hshl.de

***Abstract*—Multiprocessor scheduling is a great innovation in terms of processing tasks. It has revolutionized task processing by completing multiple tasks at the same time, which saves us time and energy. This paper initially explains the terms of multiprocessor scheduling and then focuses on Hu's algorithm which is simple but a powerful approach to solve specific scheduling scenarios. Then a Python implementation on the physical internet has been simulated to understand Hu's algorithm and multiprocessor scheduling. Finally, the results are discussed and finished with a conclusion.**

***Index Terms*—Multiprocessor scheduling, Hu's algorithm, applications, Physical internet**

I. Introduction

The art and science of multiprocessor scheduling in modern task management systems has created a multidimensional world, where each dimension is focused on processing individual tasks. Starting from the 1950s this art has been implemented in all systems efficiently allocating tasks to multiple processors. The main goal of multiprocessor scheduling is to decrease the total execution time, which is called makespan. Besides it also maximizes processor utilization and ensures weather task dependencies and deadlines are met.

Various scheduling methods are implemented under multiprocessor scheduling to handle different types of challenges. Static scheduling pre-assigns tasks and dynamic scheduling makes real-time task allocation. Preemptive scheduling interrupts the running tasks to set higher priority to a specific task. In contrast, non-preemptive scheduling ensures simultaneous task execution. Besides, each of these scheduling has some challenging factors like task Dependencies, Communication Overhead, Load Balancing, and Heterogeneity. To overcome those challenges various algorithms and heuristics have been introduced over time.

In this paper, I focused on Hu's algorithm, a powerful approach designed for specific multiprocessor scheduling scenarios. I provided a detailed overview of the algorithm including how it works and theorem analysis. Hu's algorithm makes the multiprocessor scheduling schedule faster and more efficient. It breaks all the complex tasks into smaller sub-tasks and maintains the dependencies creating a tree-like graph.

After all the theoretical discussion, an interesting implementation of the physical internet is performed that represents Hu's algorithm functionalities on multiprocessor scheduling systems. It is developed step by step following all the algorithmic steps and finally gets the desired output that makes the physical internet delivery system efficient.

II. Multiprocessor Scheduling

Multiprocessor scheduling assigns tasks to multiple processors in a computing system to optimize resource utilization and meet task deadlines. The main goal is to minimize the total execution time (makespan) and maximize processor efficiency simultaneously. Here Multiple CPUs are installed onto a single chip that makes the load sharing possible. Multiprocessor systems are fast and they can handle heavy workloads. However proper scheduling is required to achieve a balanced task processing in all processors, otherwise adding more CPUs does not necessarily make the system run faster [1].

A. Approaches to Multiprocessor Scheduling

There are two approaches to Multiprocessor Scheduling.

1) Symmetric Multiprocessing: In symmetric or single queue multiprocessing, all processors are the same and have equal access to shared resources like memory. Each CPU gets tasks from a globally shared queue or can have its private queue. Individual CPU capabilities are not considered when Scheduling decisions are made. It has some disadvantages like cache affinity and lack of scalability. Symmetric systems are easy to design but may not be the most efficient for diverse task requirements.

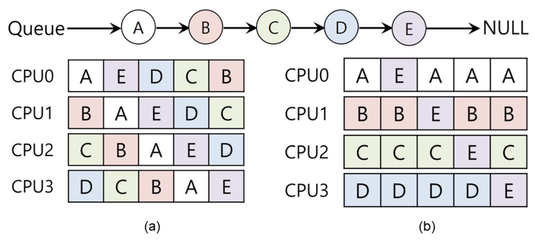


Fig. 1. Symmetric Multiprocessing

In Fig:1, we can see the global queue and in part (a), a regular Symmetric system with a cache affinity problem can be observed where different kinds of tasks are executed on different CPUs. Keeping the same task on a specific CPU would be faster as some of its state is already present in the cache of that CPU. It is called preserving affinity for most tasks, which can be seen in part (b) of the figure. [1]

2) Asymmetric Multiprocessing: There is a master processor in asymmetric or multi-queue multiprocessing and other processors work as slave processors. The master is responsible for scheduling and managing tasks for other slave processors. It has multiple queues, and each queue follows a specific rule. Processing tasks become simplified here. Fig:2 shows how it

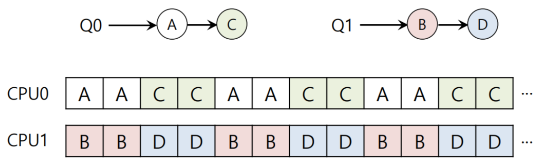


Fig. 2. Asymmetric Multiprocessing

creates different queues and how CPUs are scheduled according to the queues. It solves the problem of synchronization and data sharing since the master can decide for others. Therefore it provides more scalability and cache affinity. [1]

B. Types of Multiprocessor Scheduling:

1) Static Scheduling: In static scheduling, tasks are assigned to the processors before the system starts running. It is simpler and deterministic, and resources can be allocated in advance as the task characteristics are known beforehand. Tasks are assigned based on their types and resources. The schedule remains constant during execution. Due to this nature, any changes in requirements can lead to inefficiency. [6]

2) Dynamic Scheduling: Dynamic scheduling happens online where the schedule is generated during the execution of the processes. Tasks are assigned based on their availability and current workload. Changes in the requirements during execution do not affect the processing which makes it good for diverse task characteristics. The task execution starts as soon as they arrive in the system. As the processing happens in real-time, it is complex, can introduce more overhead, and is non-deterministic. [6]

3) Preemptive Scheduling: Preemptive Scheduling is where a running task can be preempted by a higher-priority task. Round-robin is a preemptive scheduling algorithm.

4) Non-Preemptive Scheduling: In non-preemptive Scheduling, a process assigned to a processor cannot be preempted until completion. An example would be the First-Come-First-Serve scheduling algorithm.

C. Challenges

1) Cache coherence: There is a cache memory included in each CPU. It is a very small and fast memory that remembers temporary information and can be accessed in a short time comparing the main memory. More cache memory can make the system with slow main memory appear faster. The problem with cache memory in multiprocessor scheduling is that when CPU0 changes data in its cache, it is not updated on another CPU cache. Therefore, CPU1 gets the old data instead of the updated value. This problem is solved by bus snooping, where each cache pays attention to memory updates on the bus. [1]

2) Load Balancing: Load imbalance occurs when the system experiences bottleneck issues because of the CPU overload. This mainly happens in asymmetric multiprocessing scheduling. According to the fig:2, CPU0 is executing A and C from Q0. CPU1 is executing B and D from Q1. Load imbalance happens when there is nothing to execute in CPU0 anymore as shown in fig:3 (a). To solve this, task-D or B can be migrated to Q0 as shown in part (b) of the figure. [1] Another

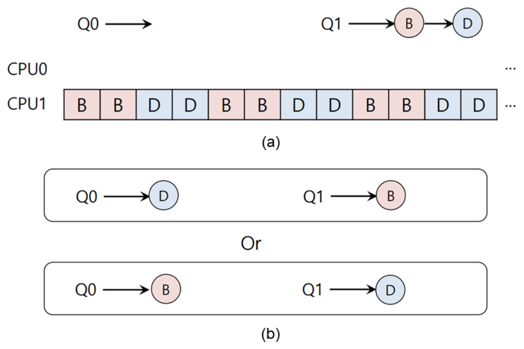


Fig. 3. Load Balancing

way to solve this is called work stealing. A source queue that has lower tasks occasionally steals tasks from another fuller queue.

3) Task Dependency: Tasks are dependent on each other sometimes. Therefore some tasks cannot start scheduling until the dependent tasks are completed. Hu's algorithm can manage this kind of task very efficiently, illustrated by a directed acyclic graph.

III. Hu's Algorithm

Hu's algorithm is designed for specific kinds of tasks that have dependencies and can be divided into multiple subprocesses in multiprocessor scheduling. It can resemble a recipe that has to be cocked to maintain an order. The main goal of this algorithm is to find out the optimal way to schedule the sub-processes to reduce the total time needed by the processors. [3] [2]

A. How It Works

1) Task Graph and Assumptions: Hu's algorithm creates a task graph where each sub-process is presented as a node and the nodes are connected through arrows according to their dependencies. It assumes that the graph is a forest. All processes here have the same type and a unit delay. [1]

2) Assign Priorities on greedy strategy: Tasks have dependencies and dependent tasks have to be executed first. Therefore a priority level of the sub-processes has to be introduced. Hu's algorithm follows a greedy strategy to choose vertices with all predecessors scheduled and maximum labels. [5]

3) Schedule: According to priority, sub-processes have to be scheduled considering the execution time, starting with the highest priority. It schedules processes at each step and increments one by one until all are scheduled.

4) Exactness Theorem: Hu's algorithm follows this formula with operations of the same type:

$$\bar{a} = \max_{\gamma} \left\lceil \frac{\sum_{j=1}^{\gamma} p(\alpha + 1 - j)}{\gamma + \lambda - \alpha} \right\rceil$$

Here, $\bar{a}$ is a lower bound on the latency λ and γ is a positive integer. Hu's algorithm on a tree with unit-cycle resources achieves latency λ with $\bar{a}$ resources. As $\bar{a}$ is lower bound, λ is minimum. [3]

IV. Applying Multiprocessor Scheduling and Hu's Algorithm in Physical Internet

A. Physical Internet

Similar to the digital internet where data packets are exchanged between senders and receivers through various network hubs, the physical internet also exchanges packages through a worldwide network that is hyper-connected physically, digitally, and operationally. It improves supply chains through standard interfaces and protocols, channel synchronization, and advanced digital containers. [4]

B. Task and Dependencies

A physical internet network among three towns is considered for this implementation. The towns are city-A, city-B, and city-C and three network hubs surround them. The hubs are hub-1, hub-2, and hub-3. Fig:4 shows the overall structure of the area.

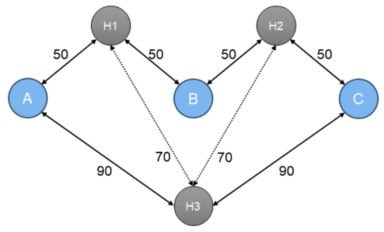
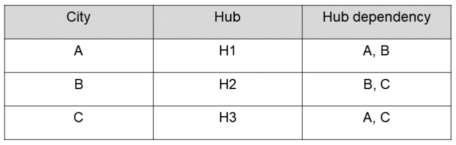


Fig. 4. Physical internet area

The delivery vehicles would be used very efficiently, ensuring maximum utilization of the space. Each vehicle delivers the packages to a destination hub and the hub always reassigns a new delivery task to the vehicle. Fig:5 shows all the tasks and dependencies.



City	Hub	Hub dependency
A	H1	A, B
B	H2	B, C
C	H3	A, C

Fig. 5. Task and dependencies

C. Algorithm Steps

1) Createing a Task Graph: Hu's algorithm starts with a task graph that represents all the possible processes as a tree. The task graph for the physical internet scenario is shown in the fig:6.

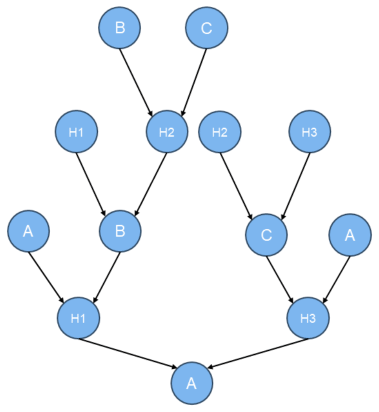


Fig. 6. Task graph

2) Assigning Priorities: Priorities are assigned based on available tasks and the required time to complete the task. If ten containers are generated randomly, then Hu's algorithm schedules them in an optimal way that requires minimum execution time and is easily accessible.

3) Scheduleing with Multiprocessor Consideration:

V. Python Implimentation

A. First Step

Networkx library is imported to create a task graph and cities, and hubs are specified. Then each city and hub is a node. After that, all the distances between cities and hubs are assigned as follows:

```
import networkx as nx
import random

cities = ["A", "B", "C"]
hubs = ["H1", "H2", "H3"]
G = nx.DiGraph()

for node in cities + hubs:
    G.add_node(node)
G.add_edge("A", "H1", distance=50)
G.add_edge("B", "H1", distance=50)
G.add_edge("C", "H1", distance=150)
G.add_edge("H1", "H2", distance=100)
```

B. Second Step

To start the task execution or delivery process, ten containers were created randomly with the origin and destination city. As the system knows where to send the containers, it calculates the priorities for each of them. So a priority calculation function has been implemented that finds the shortest path using the distance.

```
calculate_priority(origin, destination):
 distance = nx.shortest_path_length(G,
 origin, destination, weight=’distance’)
```

C. Third Step

Using the distance, Hu’s algorithm searches for the shortest weighted path and the actual path nodes that need to be processed first using Dijkstra.

```
distance, path = nx.single_source_dijkstra
(G, origin,destination, weight=’distance’)
```

D. Fourth Step

Each hub has a limited capacity and it needs to be checked before starting the process.

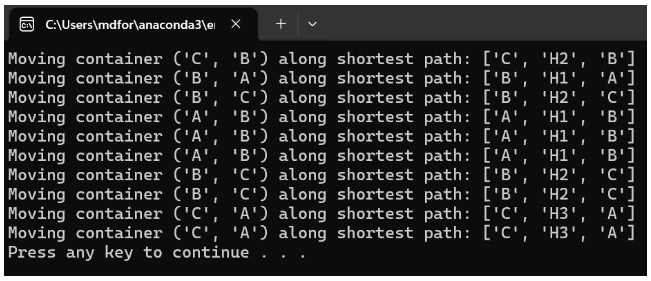
```
if current_hub_loads[path[i]] >=
hub_capacities[path[I]]:
   distance = float(’inf’)
   break
else:
    current_hub_loads[path[i]] += 1
```

E. Fifth Step

Finally, all the tasks are listed in the scheduler based on their priorities and printed.

```
schedule = []
for container, path, _ in schedule:
  print(f"Moving container {container}
  along shortest path: {path}")
```

VI. Results and Analysis



```
Moving container ('C', 'B') along shortest path: ['C', 'H2', 'B']
Moving container ('B', 'A') along shortest path: ['B', 'H1', 'A']
Moving container ('B', 'C') along shortest path: ['B', 'H2', 'C']
Moving container ('A', 'B') along shortest path: ['A', 'H1', 'B']
Moving container ('A', 'B') along shortest path: ['A', 'H1', 'B']
Moving container ('A', 'B') along shortest path: ['A', 'H1', 'B']
Moving container ('B', 'C') along shortest path: ['B', 'H2', 'C']
Moving container ('B', 'C') along shortest path: ['B', 'H2', 'C']
Moving container ('C', 'A') along shortest path: ['C', 'H3', 'A']
Moving container ('C', 'A') along shortest path: ['C', 'H3', 'A']
Press any key to continue . . .
```

Fig. 7. Results

fig:7 presents the results of this physical internet Python implementation. All the containers are labeled with their origin and destination location. Hu’s algorithm calculates the optimal path for that location and calculates the path, which is printed after the container label. here, the first container is received from city-C and it has to be delivered to city-B. The shortest path for this container is [C, H2, B] which means it has to go through hub-2. Similarly, all the containers have been assigned to their nearest hub to be delivered or processed faster.

VII. Conclusion

In conclusion, this paper has discussed the most basic terms of multiprocessor scheduling and Hu’s algorithm. Then a real-life Python application of the physical internet has been implemented in the context of Hu’s algorithm and multiprocessing. Most of the systems run on multiprocessor scheduling now, due to its powerful processing. It can be customized and improved according to the needs by applying various techniques. Asymmetric multiprocessor scheduling gives us a master-slave system and symmetric is independent. They are used based on the required system needs and can introduce various challenges like cache coherence, cache affinity, load imbalance, communication overhead, task dependency, and heterogeneity. Disciplines like bus snooping, task migration, work-stealing, and various algorithms like Hu’s can solve these problems.

Bigger problems are solved by breaking it into smaller problems. That is what Hu’s algorithm does. It breaks the tasks into sub-tasks and starts scheduling and maintaining an order. It provides an optimal way for scheduling and reduces the total time. A task graph is created at first and priorities are assigned. A Python implementation of this algorithm is executed to find out the optimal path of a physical internet scenario. The physical internet is a great example of Hu’s algorithm on multiprocessor scheduling. It is about reducing the global distance and making an advanced logistic network. Though this is not yet applied fully in real life, researchers showed the potential benefits of this network, such as great cost reduction, efficient use of vehicles, fewer vehicles, digital pi containers, less time, less travel time for the drivers, etc.

References

[1] Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. *Operating systems: Three easy pieces: Ch-10: Multi-CPU Scheduling*. Arpaci-Dusseau Books, Madison, 2018.

[2] Alexey S. Chernigovskiy, Roman Yu. Tsarev, and Alexey N. Knyazkov. Hu’s algorithm application for task scheduling in n-version software for satellite communications control systems. In *2015 International Siberian Conference on Control and Communications (SIBCON)*, pages 1–4. IEEE, 2015.

[3] G. DeMicheli. *Synthesis of Digital Circuits: (202-207)*. McGraw-Hill series in electrical and computer engineering Electronics and VLSI circuits. McGraw, New York, NY, internat. ed. edition, 1994.

[4] dhl.com. Physical internet.

[5] Giovanni De Micheli. 0002.scheduling2: Hu’s algorithm: Stanford university. 2004.

[6] S. Amarjeet Singh Sahi Government Polytechnic College ,Talwara. Static vs dynamic scheduling, 2023.