Project Title:
Fedsort: An Optimized Federated Scheduling Strategy for Workloads with Inter-task Dependencies

PRESENTED BY

Suhas Gowda Harish - PES1UG21CS633

Sparsh BK - PES1UG21CS610

Shresht V G - PES1UG21CS566

UNDER THE GUIDANCE OF

Dr. Prafullata Kiran Auradkar

DEPARTMENT OF CSE

PES UNIVERSITY

Problem Statement

Developing an Innovative Federated Scheduling Strategy to Enhance Overall Job Response Time in Data Center Workloads with Inter-Task Dependencies.

Motivation

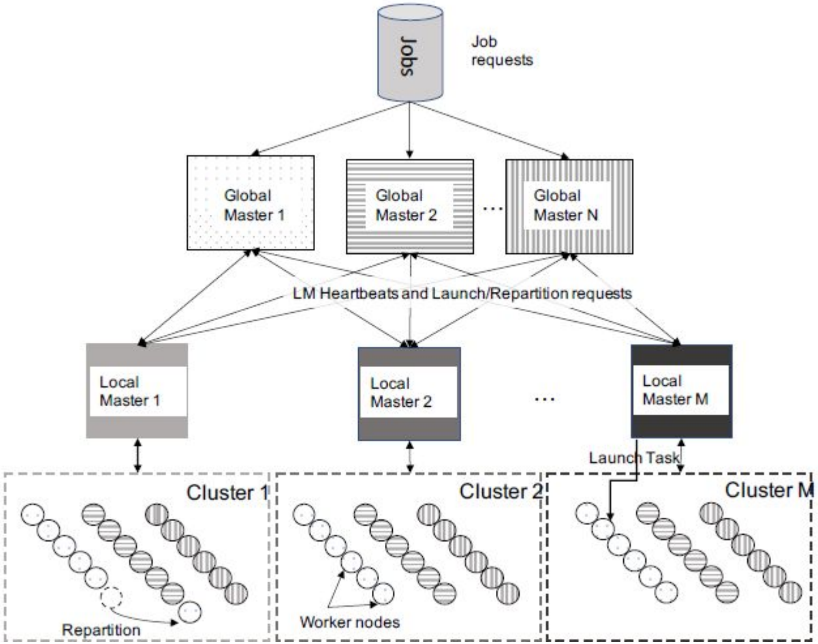
Challenges in Cloud Scheduling :

- Modern cloud workloads are becoming increasingly **heterogeneous**, with diverse resource needs and execution characteristics.
- Scheduling these workloads efficiently is critical for performance and resource utilization.

	HADOOP: YARN	SPARROW	HAWK
Type of scheduling	Centralized Scheduling	Distributed Scheduling	Hybrid Scheduling
Latency (Time taken to execute tasks after submission to the scheduler)	High Latency	Low Latency	Moderate Latency (better compared to centralized not as good as distributed)
Placement (Strategic placing of tasks into clusters based on resource constraints)	Good Placement i.e has **complete global view** of the cluster	Bad Placement i.e schedules tasks based on random probing	Has **partial global view** of the cluster (better compared to distributed not as good as centralised).
Limitations (Research gap)	**Scalability issues** and at higher cluster loads(greater than 4000 nodes) the performance deteriorates drastically.	Performance reduces at high cluster load due to random probing.	At high cluster load long jobs force the **short jobs** to run **only on reserved nodes** thus reducing performance. **Head of line blocking**(bottleneck)

Megha

Federated Scheduling



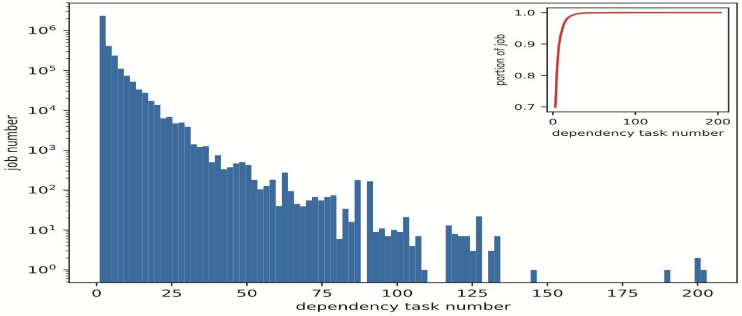
- **Megha**, a **decentralized global scheduler**
- Uses **flexible partitioning** to overcome the limitations of existing schedulers.
- Top level Global masters(GM) use global view of the system resources to make scheduling decisions.
- The LM will validate the decisions made by GM and then deploys tasks to the worker nodes of the clusters.
- Megha achieves low allocation times and ensures against head-of-line blocking and starvation of any class of jobs

Drawbacks of Federated Scheduling

- **No Support for Inter-Task Dependencies**
    - Megha was originally designed assuming tasks within jobs are independent.
    - It could not handle Directed Acyclic Graph (DAG)-based jobs with internal task dependencies, which are common in real-world workflows like data processing pipelines.

Introduction to traces

1. Batch jobs are composed of multiple tasks, with dependencies, that can be modelled as a directed-acyclic graph.[1][2]
2. Both the Google cluster trace (2019) and the alibaba cluster trace (2018) provide details reflecting this inter-task dependencies in real-world data center workloads



What is a Trace?

- This dataset includes information on resource usage (CPU, memory, disk) and job characteristics (duration, failures) for workloads running on Google's production cloud, Alibaba's cloud platform,etc.
- Considering this information we only take the parameters which we require for our project.
- The parameters being considered for Alibaba Cluster Trace

JobID

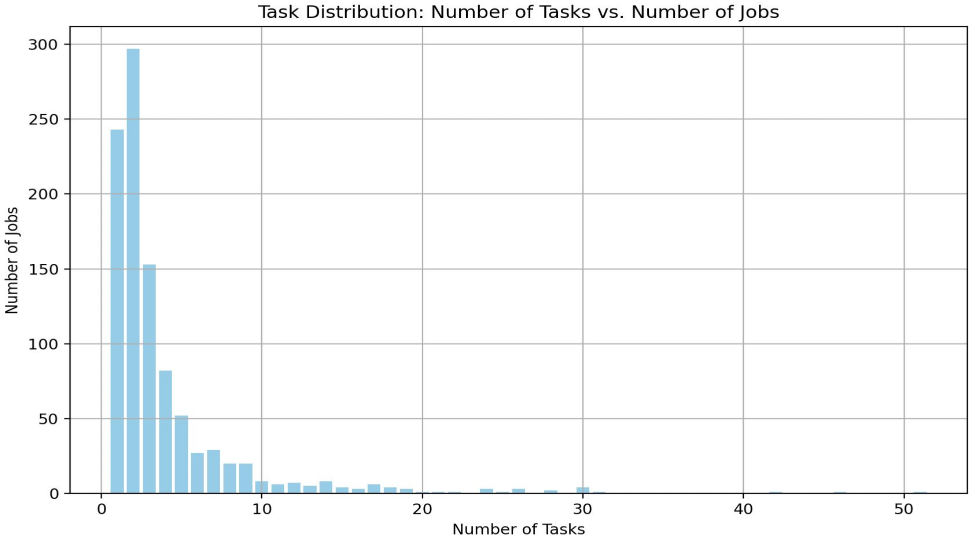
Number_of_tasks

Job-completion_time (in sec)

Individual_task_durtion_time (in sec)

Current trace drawback

- **Smaller Task Size**: Many jobs consist of very few tasks , limiting the complexity of job flows.
- For a quality experiment analysis we would need a better task distribution



Synthetic DAG creation

Motivation:

- Original dataset had limited DAGs, reducing complexity in task dependencies.
- Goal: Enhance dataset with realistic dependency structures for federated scheduling evaluation.

Approach:

- **Algorithm**: Adapted from "Characterizing and Synthesizing Task Dependencies of Data-Parallel Jobs in Alibaba Cloud" to synthesize realistic DAGs based on Critical Path (CP) and Level (LV) constraints.
- **Parameters Used**: Jobs with s=10 and s=12 to ensure a variety of task dependency structures.

Methods applied:

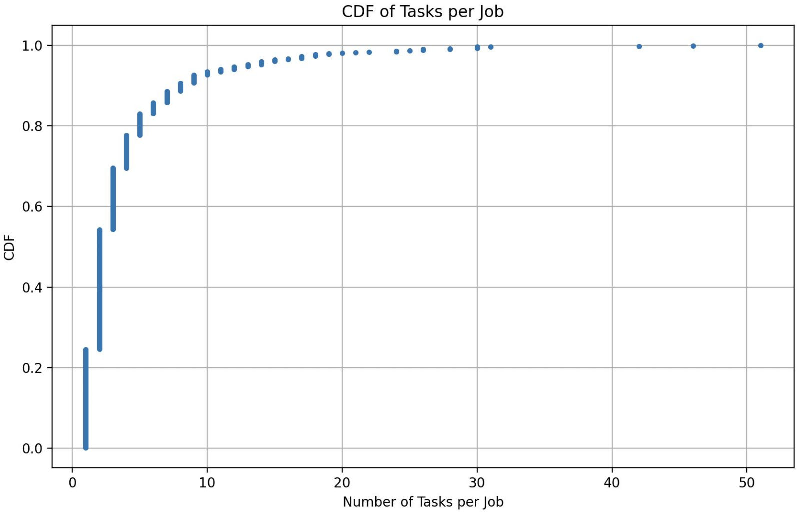
Method 1: Unconditional DAG Generation for Small Jobs

- **Process**: For jobs with fewer than 3 tasks, generate DAGs unconditionally.
- **Outcome**: Randomized DAG generation can sometimes lead to independent tasks only. Added new tasks with durations randomized within median time range.

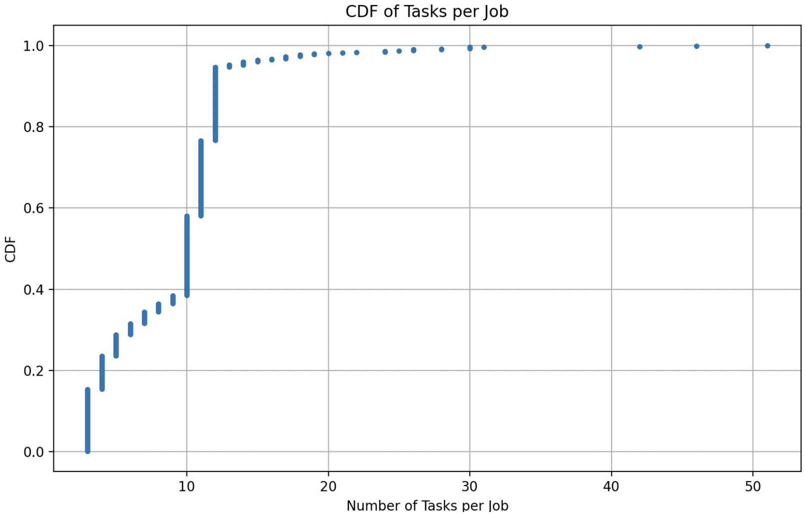
Method 2: Conditional DAG Retention to Maximize Dependencies

- **Process**: For jobs with fewer than 3 tasks, apply DAG generation only if it results in dependent tasks.
- **Fallback**: Retain original structure if the DAG is independent.
- **Advantage**: Increases dependent task distribution, enhancing dataset complexity.

Comparison

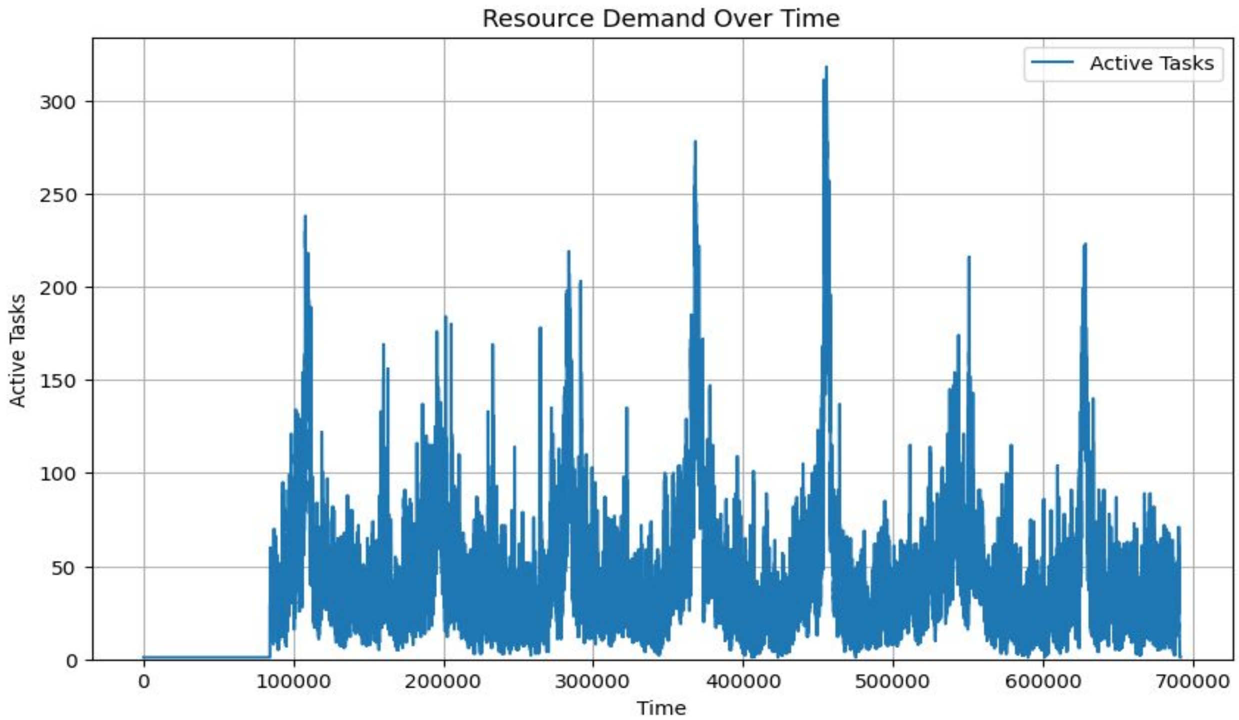


Original trace



Synthesised trace

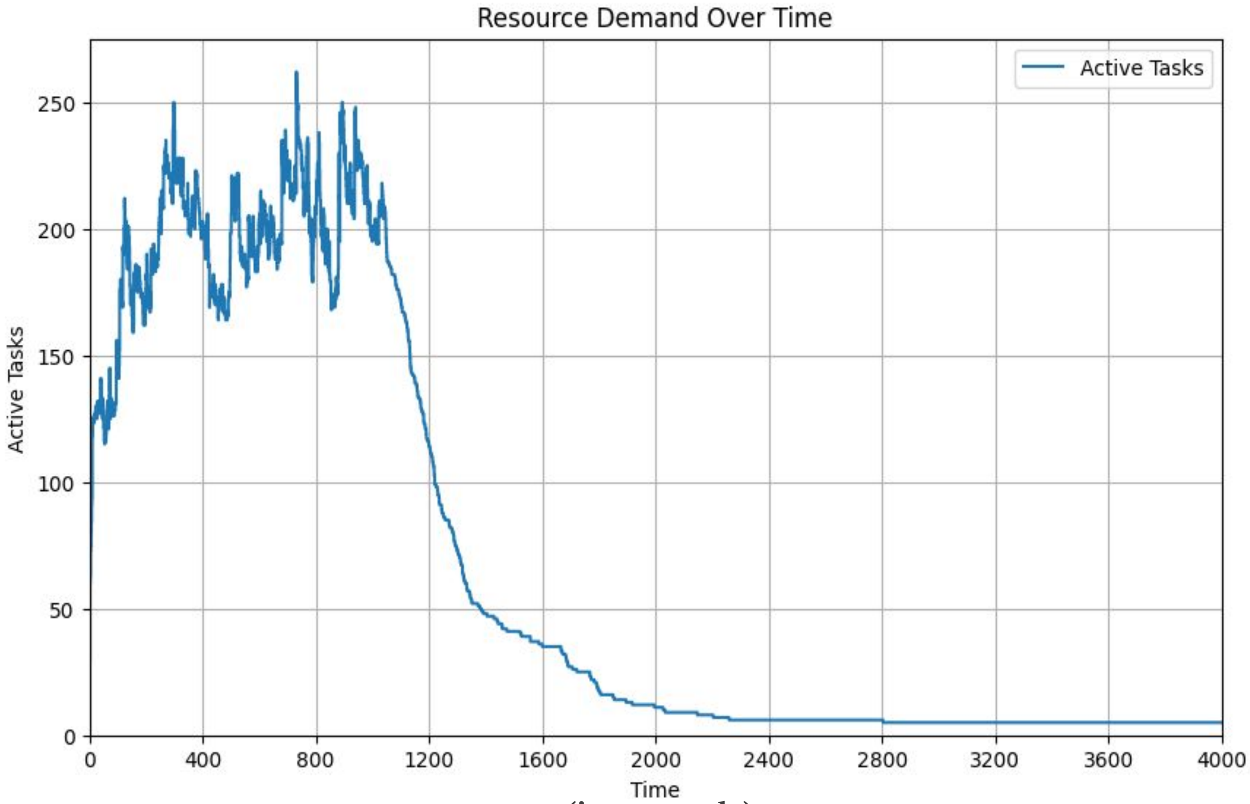
Resource Demand Graph (Full Trace)



(in seconds)

Jobs - 75,473
Tasks - 2,94,124
Peak nodes - **318**
Ideal Average Job Completion Time - **231.7633 s**

Resource Demand Graph (1K Jobs Trace)



(in seconds)

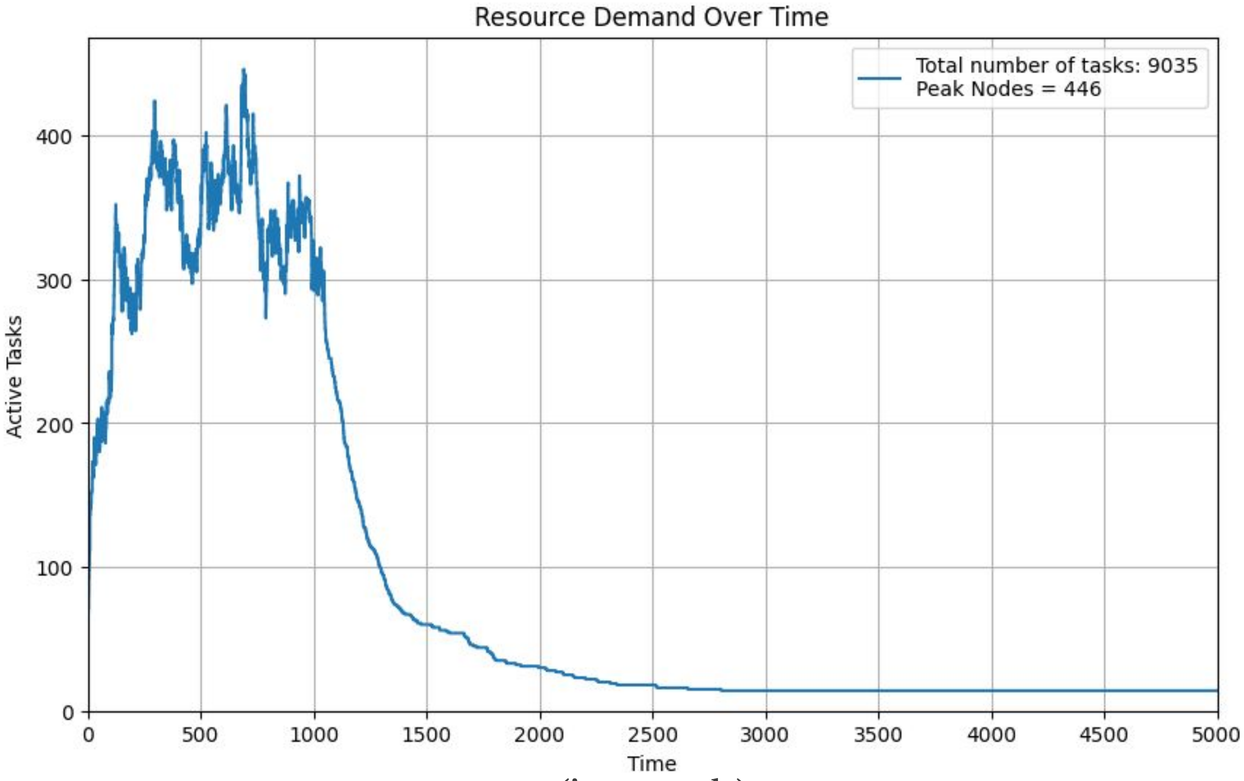
Jobs - 1000

Tasks - 3926

Peak nodes - **262**

Ideal Average Job Completion Time - **847.8325 s**

Resource Demand Graph (1K Jobs Synthetic DAG Trace)



(in seconds)

Jobs - 1000

Tasks - 9035

Peak nodes - **446**

Ideal Average Job Completion Time - **3133.018 s**

What is Inconsistency?

- When a Global Master can't find an available worker node in its partition, it checks other partitions.
- If a node is available, it requests the local master to confirm and temporarily assigns the node.
- However, the Global master to which this node belongs to might also try to use the same node, causing an **Inconsistency Event.**

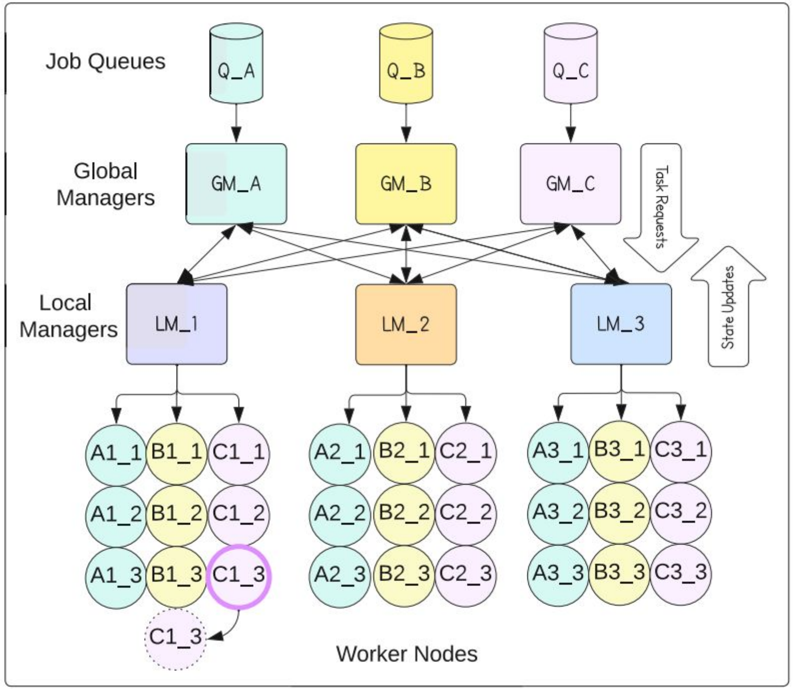


Fig. 1. Megha's Architecture.

FedSort: Our approach for scheduling dependent tasks

Task Sorting (Dependency-Aware)

- Tasks are initially **sorted by job and dependency order**:
  - **Independent tasks** are scheduled first parallely among the worker nodes
  - **Dependent tasks** are appended in order, preserving prerequisites
    (Sorted using task IDs before the underscore)

Dependency-Aware Queuing

- A queue is maintained for tasks **waiting on unresolved dependencies**
- On each task completion, the system checks if **blocked tasks** can now be scheduled

Batch Scheduling

- Tasks ready for execution are **batched** and sent together to reduce:
  - **Network overhead**
  - **Mapping inconsistencies**
- Prioritizes assigning tasks within a **Global Master's internal partition**, then tries **external partitions** if needed

> The motivation for batching is to reduce the number of LM status updates. Updating the global state of a GM is a costly operation both in terms of processing and communication overheads. During our experiments, we found that without batching, multiple invalid requests

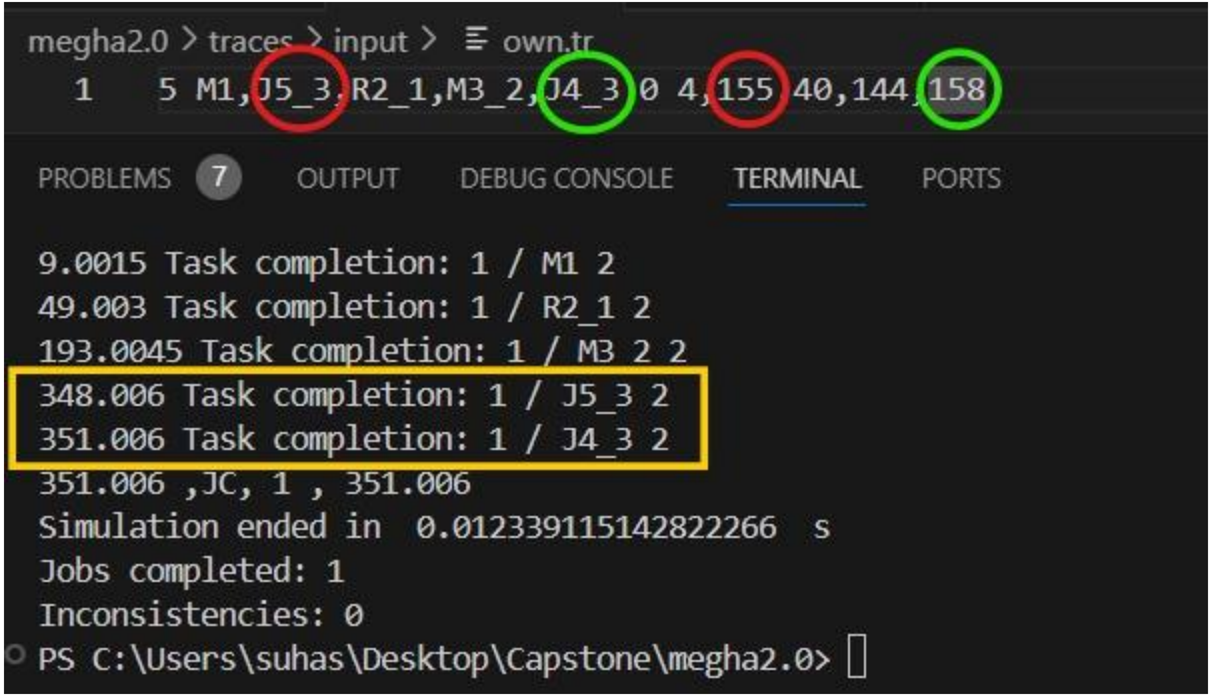
FIFO-Based Priority

- Tasks from **earlier-arriving jobs** are given **higher priority**
- Preserves fairness while managing dependent workflows

Fedsort: Scheduling Workflow

As you can see both task number 4 and 5 are dependent on task 3 and they are being completed at 351 and 348 seconds which is a difference off 3 seconds .

Their individual task duration is 158 and 155 seconds respectively and their difference is also 3 seconds , hence they are being scheduled at the same time in parallel once task 3 is finished (As task 3 ended at 193 seconds and 193+155=348)



Workload Modeling with Poisson Arrival Processes

- **Job Arrival Modeling:** Jobs arrive following a **Poisson** process, **simulating random, independent** job arrivals over time.

- **Inter-Arrival Time (IAT):** Time between consecutive jobs is modeled with an **exponential distribution**.

- **Load Control:** The **mean** of the exponential distribution **determines system load**, with a **lower mean** indicating **higher job arrival rates**.

- **Purpose:** Varying IAT allows for **testing system performance** under **different load conditions**.

Workload Distribution for Means - 1s, 2s, 3s

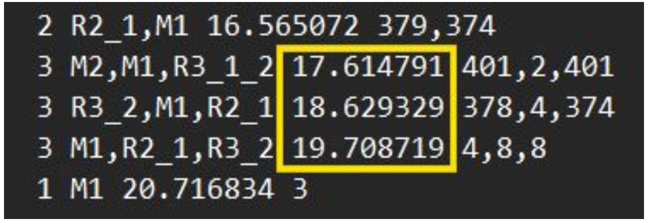
There are 1000 jobs in this workload and each **job arrives** in a **time difference** of around **1 second**.

In the below picture each row is a job
1st attribute - No. of Tasks in Job

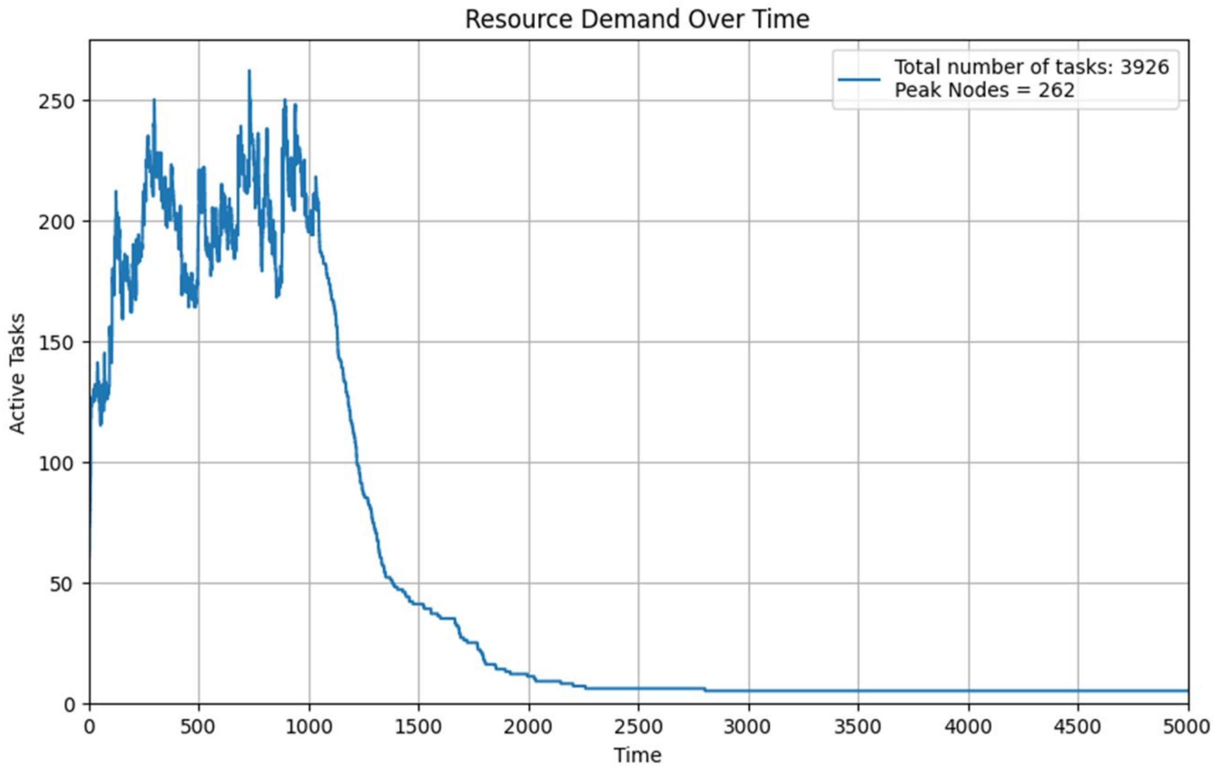
2nd attribute - Task names with dependency

3rd attribute - Job arrival time

4th attribute - Individual Task durations



```
2 R2_1,M1 16.565072 379,374
3 M2,M1,R3_1_2 17.614791 401,2,401
3 R3_2,M1,R2_1 18.629329 378,4,374
3 M1,R2_1,R3_2 19.708719 4,8,8
1 M1 20.716834 3
```



Workload Distribution for Means - 1s, 2s, 3s

There are 1000 jobs in this workload and each **job arrives** in a **time difference** of around **2 second**.

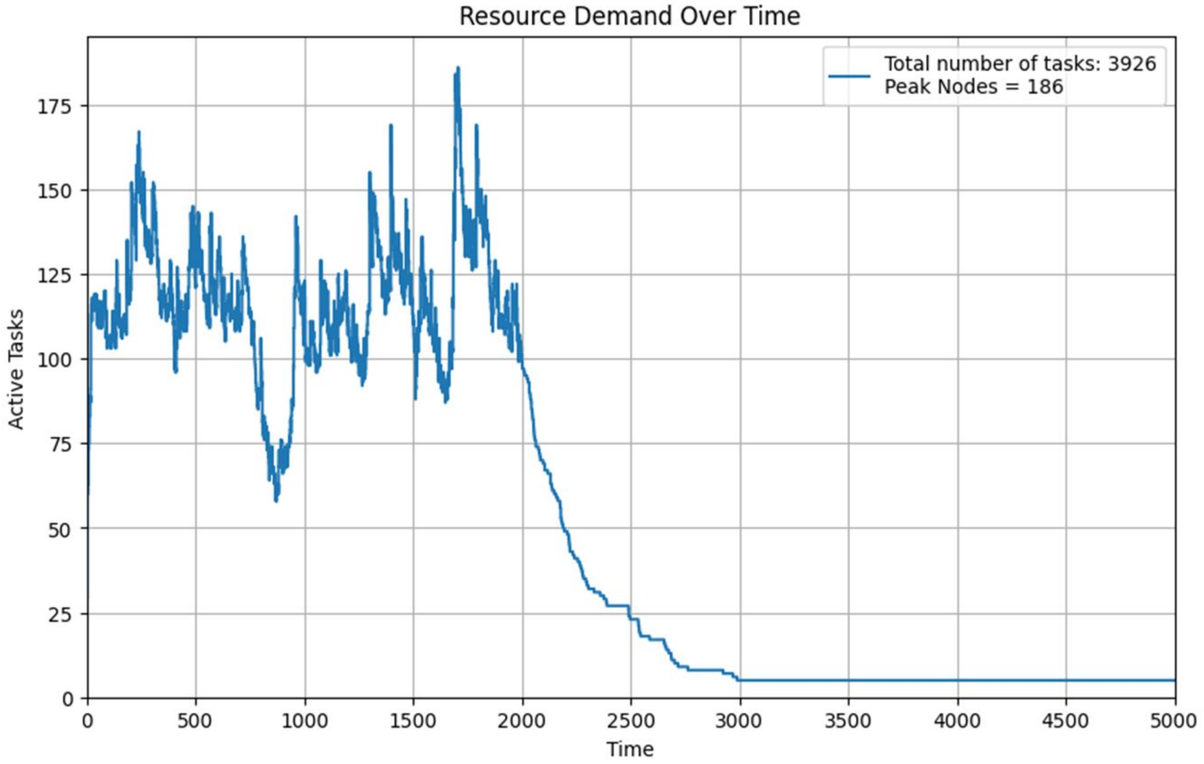
In the below picture each row is a job
1st attribute - No. of Tasks in Job

2nd attribute - Task names with dependency

3rd attribute - Job arrival time

4th attribute - Individual Task durations

```
2 R2_1,M1 32.02976 379,374
3 M2,M1,R3_1_2 33.94153 401,2,401
3 R3_2,M1,R2_1 36.034288 378,4,374
3 M1,R2_1,R3_2 38.017429 4,8,8
1 M1 40.085564 3
```



Workload Distribution for Means - 1s, 2s, 3s

There are 1000 jobs in this workload and each **job arrives** in a **time difference** of around **3 second**.

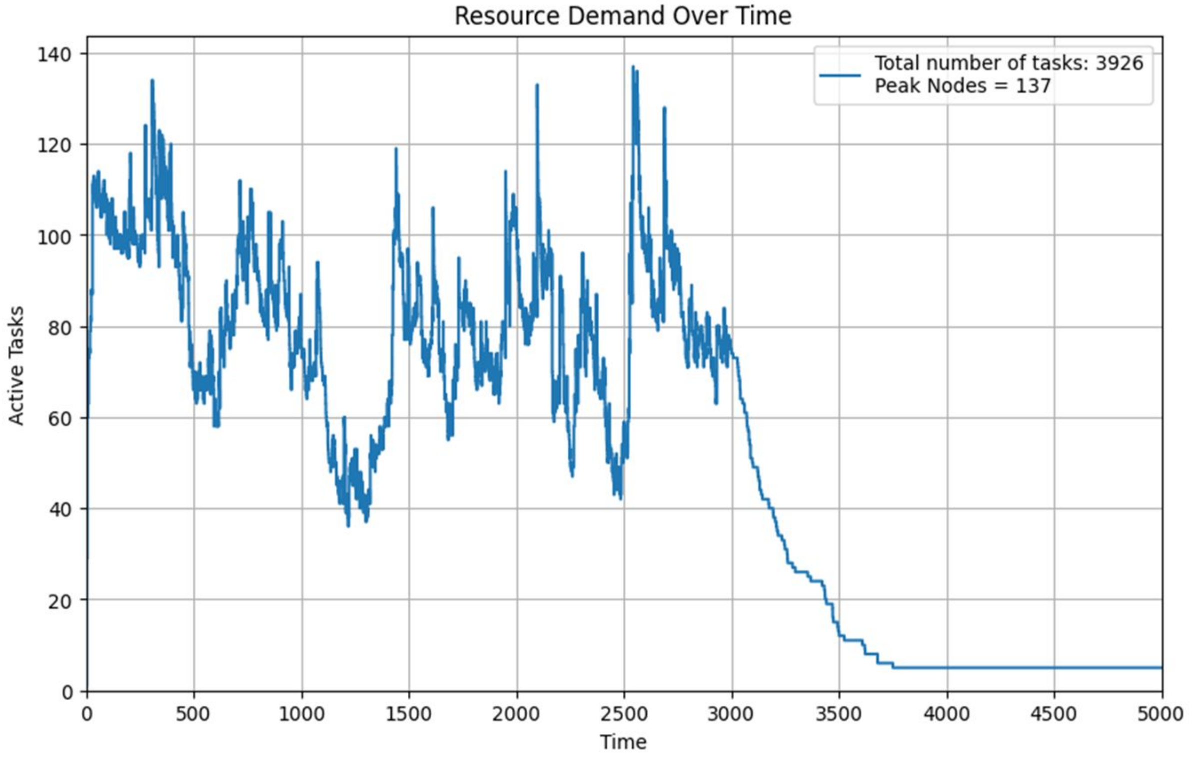
In the below picture each row is a job
1st attribute - No. of Tasks in Job

2nd attribute - Task names with dependency

3rd attribute - Job arrival time

4th attribute - Individual Task durations

```
2 R2_1,M1 47.99316 379,374
3 M2,M1,R3_1_2 51.006314 401,2,401
3 R3_2,M1,R2_1 54.038955 378,4,374
3 M1,R2_1,R3_2 57.063156 4,8,8
1 M1 59.976353 3
```



Average Delay in JCT & Inconsistencies Comparison

JOB INTER-ARRIVAL TIME – MEAN OF 1 SECON

	Without sorting		Our sorting	
	Average Delay in Job Completion Time (in seconds)	Inconsistencies	Average Delay in Job Completion Time (in seconds)	Inconsistencies
G2L2S20	5741.053	4	4515.958	4
G2L5S50	1499.847	105	932.182	316
G5L5S50	1216.403	61	880.1872	45
G2L5S100	336.812	63	115.1711	55
G5L5S100	310.102	93	129.1464	70
G2L5S270	0.24839	75	0.22305	65
G5L5S275	0.321144	63	0.354637	67
G5L5S1000	0.030672	15	0.038642	18

Average Delay in JCT & Inconsistencies Comparison

JOB INTER-ARRIVAL TIME – MEAN OF 2 SECONI

	Without sorting		Our sorting	
	Average Delay in Job Completion Time (in seconds)	Inconsistencies	Average Delay in Job Completion Time (in seconds)	Inconsistencies
G2L2S20	4963.111	10	3143.188	10
G2L5S50	1093.116	489	518.617	248
G5L5S50	1086.103	31	448.382	44
G2L5S100	25.567	175	13.330	178
G5L5S100	17.861	277	12.317	262
G2L5S270	0.326	188	0.268	154
G5L5S275	0.348	86	0.519	86
G5L5S1000	0.113	29	0.238	41

Average Delay in JCT & Inconsistencies Comparison

JOB INTER-ARRIVAL TIME – MEAN OF 3 SECONI

	Without sorting		Our sorting	
	Average Delay in Job Completion Time (in seconds)	Inconsistencies	Average Delay in Job Completion Time (in seconds)	Inconsistencies
G2L2S20	4444.409	6	2538.148	9
G2L5S50	774.004	44	185.132	76
G5L5S50	500.684	39	261.484	36
G2L5S100	0.712	158	0.606	146
G5L5S100	0.531	105	1.113	127
G2L5S270	0.833	168	0.407	164
G5L5S275	0.682	61	0.585	57
G5L5S1000	0.188	27	0.031	14

THANK YOU