PRACTICAL -4

AIM: Write a program on an unloaded cluster for several different numbers of nodes and record the time taken in each case. Draw a graph of execution time against the number of nodes

□ WHAT IS HPC CLUSTER?

High-Performance Computing (HPC) clusters are powerful computing systems that consist of multiple interconnected computers, or nodes, working together to solve complex computational problems. These clusters are designed to provide significantly higher computational power and speed compared to individual computers.

□ HOW TO BUILD AN HPC CLUSTER?

Building an HPC (High-Performance Computing) cluster involves several steps, and the process can be complex. Here's a general overview of the steps you might take:

Define Requirements: Identify the specific computational requirements of your applications or simulations. Determine the desired level of performance, scalability, and parallel processing capability.

Select Hardware: Choose high-performance hardware components, including processors, memory, accelerators (such as GPUs or FPGAs), interconnects, and storage.Consider factors like power consumption, cooling, and physical space constraints.

Networking: Select a high-speed interconnect technology (e.g., InfiniBand, Ethernet) for efficient communication between nodes. Plan the network topology to minimize latency and maximize bandwidth.

Cluster Architecture: Decide on the cluster architecture, such as symmetric or asymmetric, and the arrangement of nodes. Consider the use of a master node for managing the cluster and coordinating tasks.

Operating System and Software:

- Choose a suitable operating system for the cluster (common choices include Linux distributions tailored for HPC, such as CentOS, Ubuntu, or specialized HPC-centric OS).

- Install necessary software tools, libraries, and middleware for parallel processing and job scheduling (e.g., MPI for message passing, Slurm or Torque for job scheduling).

File System: Implement a high-performance parallel file system that can handle the I/O demands of your applications. Consider distributed or parallel file systems like Lustre or GPFS.

Cluster Management: Install cluster management software for monitoring, resource allocation, and job scheduling. Popular choices include OpenHPC, Bright Cluster Manager, or custom configurations using tools like Puppet or Ansible.

Power and Cooling: Ensure adequate power and cooling infrastructure to support the cluster, especially considering the high power consumption of HPC systems.

Testing: Perform thorough testing to validate the functionality and performance of the HPC cluster. Test parallel processing capabilities, network communication, and overall system stability.

Optimization: Fine-tune the cluster configuration based on performance benchmarks and application requirements. Optimize the software stack, parallelization strategies, and resource allocation.

Documentation: Create comprehensive documentation for the cluster setup, configuration, and maintenance procedures.

Training: Provide training for administrators and users on how to effectively use and manage the HPC cluster.

Scaling: Plan for future scalability by designing the cluster with expansion capabilities, enabling the addition of more nodes as needed.

“Building an HPC cluster requires expertise in hardware, networking, system administration, and parallel programming. Depending on your specific needs and resources, you may also consider seeking assistance from HPC specialists, consultants, or vendors with experience in cluster deployment.”

KEY COMPONENT OF HPC CLUSTER

Computer Hardware:

- Includes servers, storage, and a dedicated network.
- Provision at least three servers for primary, worker, and client nodes.
- High-end servers with ample processors and storage are essential.

- Networking infrastructure requires high-bandwidth TCP/IP equipment like Gigabit Ethernet

Software:

- Comprises tools for monitoring, provisioning, and managing the cluster.
- Software stacks include libraries, compilers, debuggers, and file systems.
- HPC frameworks like Hadoop offer fault-tolerance and automatic system redirection

Facilities:

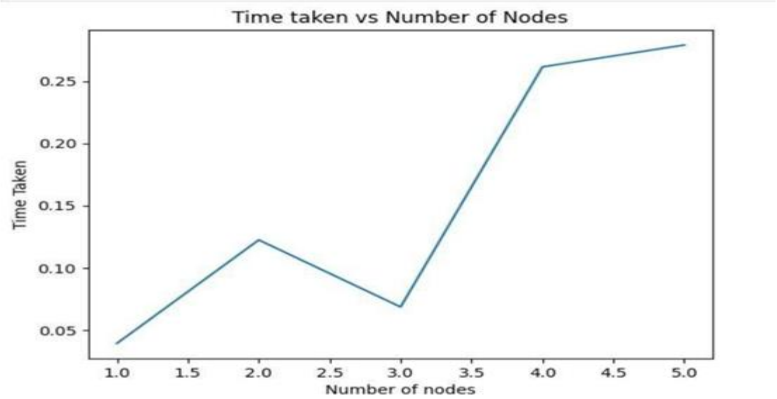
- Physical space to hold racks of servers.
- Power capacity up to 43 kW to operate and cool the servers.

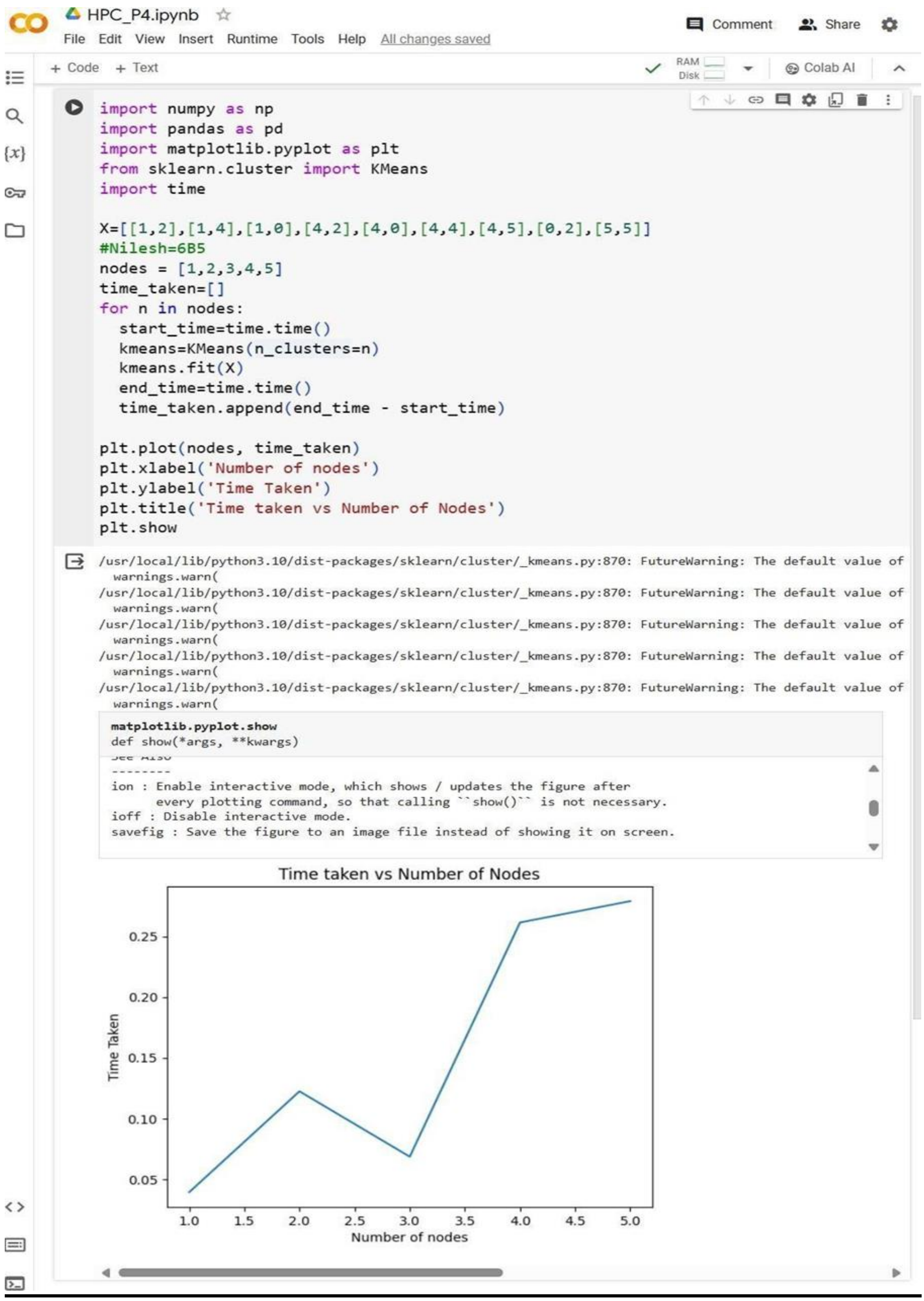
- Networking gear such as NICs and switches are crucial for the cluster

CODE: Draw a graph of execution time against the number of nodes

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt from sklearn.cluster import KMeans
import time
X=[[1,2],[1,4],[1,0],[4,2],[4,0],[4,4],[4,5],[0,2],[5,5]]
#Nilesh=6B5
nodes = [1,2,3,4,5]
time_taken=[] for n in nodes:
start_time=time.time() kmeans=KMeans(n_clusters=n) kmeans.fit(X) end_time=time.time()
time_taken.append(end_time - start_time)
plt.plot(nodes, time_taken)
plt.xlabel('Number of nodes')
plt.ylabel('Time Taken')
plt.title('Time taken vs Number of Nodes')
plt.show
```

OUTPUT:





```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import time

X=[[1,2],[1,4],[1,0],[4,2],[4,0],[4,4],[4,5],[0,2],[5,5]]
#Nilesh=6B5
nodes = [1,2,3,4,5]
time_taken=[]
for n in nodes:
  start_time=time.time()
  kmeans=KMeans(n_clusters=n)
  kmeans.fit(X)
  end_time=time.time()
  time_taken.append(end_time - start_time)

plt.plot(nodes, time_taken)
plt.xlabel('Number of nodes')
plt.ylabel('Time Taken')
plt.title('Time taken vs Number of Nodes')
plt.show
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of
  warnings.warn(
```

```
matplotlib.pyplot.show
def show(*args, **kwargs)
See Also
--------
ion : Enable interactive mode, which shows / updates the figure after
      every plotting command, so that calling ``show()`` is not necessary.
ioff : Disable interactive mode.
savefig : Save the figure to an image file instead of showing it on screen.
```