Project Report ON

AI-AUGMENTED HPC MANAGEMENT: AUTONOMOUS SCALING AND INTELLIGENT DIAGNOSTICS FOR SLURM CLUSTERS

Submitted

In partial fulfilment

For the award of the Degree of

Post Graduate Diploma in High Performance Computing

System Administration

(C-DAC, ACTS (Pune))

Guided by:	**Submitted by:**	
Mr. Ashutosh Das	Ajinkya Zine	PRN: 250840127002
	Anjali Hongekar	PRN: 250840127003
	Harshada Rathor	PRN: 250840127008
	Mohit Pawaskar	PRN: 250840127012
	Soham Bhamare	PRN: 250840127021

Centre of Development of Advanced Computing (C-DAC)

ACTS (Pune – 411008)

CERTIFICATE

TO WHOMSOEVER IT MAY CONCERN

This is to certify that

Ajinkya Zine	PRN: 250840127002
Anjali Hongekar	PRN: 250840127003
Harshada Rathor	PRN: 250840127008
Mohit Pawaskar	PRN: 250840127012
Soham Bhamare	PRN: 250840127021

Have successfully completed their project on

"AI-Augmented HPC Management: Autonomous Scaling and Intelligent Diagnostics for Slurm Clusters"

Under the guidance of

Mr. Ashutosh Das

Project Guide

Mr. Ashutosh Das

Course Coordinator

Ms. Divya Patel

HOD ACTS

Mr. Gaur Sunder

ACKNOWLEDGEMENT

This is to acknowledge our indebtedness to our Project Guide, **Mr. Ashutosh Das,** CDAC ACTS, Pune for his constant guidance and helpful suggestions for preparing this project **AI-Augmented HPC Management: Autonomous Scaling and Intelligent Diagnostics for Slurm Clusters.**

We express our deep gratitude towards his inspiration, personal involvement, constructive criticism that he provided us along with technical guidance during the course of this project.

We take the opportunity to thank head of department **Mr. Gaur Sunder** for providing us such a great infrastructure and environment of our overall development.

It is our great pleasure in expressing sincere and deep gratitude towards **Mrs. Risha P.R.** (Program Head) and **Ms. Divya Patel** (Course Coordinator, PG-DHPCSA) for their valuable guidance and constant support throughout this work and help to pursue additional studies.

Also, our warm thanks to **C-DAC ACTS** Pune, which provided us this opportunity to carry out this prestigious Project and enhance our learning in various technical fields.

ABSTRACT

Traditional High-Performance Computing (HPC) management is often plagued by rigid resource allocation and the high operational overhead of manual troubleshooting. As clusters scale, the frequency of node failures—ranging from service drifts in Munge to mount issues in NFS—creates significant downtime. This project, "AI-Augmented HPC Management: Autonomous Scaling and Intelligent Diagnostics for Slurm Clusters," introduces a paradigm shift by implementing a self- governing management layer. By integrating the Gemini API with the Slurm Workload Manager, we have developed an intelligent framework that can perceive cluster states, diagnose root causes of failures, and execute remedial actions autonomously.

The infrastructure is built on a high-availability foundation using Ubuntu, featuring centralized identity management via LDAP and flexible storage orchestration through LVM and NFS. To ensure secure and seamless communication across distributed environments, Tailscale is utilized to establish a Zero-Config VPN mesh. This provides a stable, encrypted management backplane, allowing the Slurm controller to maintain consistent connectivity with compute nodes regardless of network complexity. Services such as Bind9 DNS and NTP are meticulously configured to provide the sub-millisecond clock synchronization and name resolution required for stable cluster operations and authenticated job scheduling.

The innovative core of the project is the Python-based AI Agent, The agent monitors the cluster's health and queue depth; when it detects a DRAIN state or a bottlenecked job queue, it leverages the Gemini API to analyze system logs and determine the most efficient path for recovery or expansion. Through Ansible automation, the agent can dynamically provision new nodes or fix existing ones by restarting failed daemons or correcting configuration errors. This results in an elastic, self-healing HPC environment that maximizes hardware utilization and significantly reduces the need for manual administrative intervention.

TABLE OF CONTENTS

Sr.No.	Title	Page No.
1	**Introduction and Overview of Project**	6
1.1	Purpose	
1.2	Aims and Objectives	
2	**Prerequisites**	7&8
2.1	Hardware and Software Requirements	
3	**Overall Description**	9-18
3.1	Introduction	
3.2	Data Flow: Job Queuing to Cloud Provisioning	
3.3	System Architecture	
4	**Cluster Setup and Configuration**	19-23
4.1	Base Layer: DNS, NTP, and SSH Passwordless Communication	
4.2	Identity Management: LDAP Centralized Authentication	
4.3	Storage: LVM Elastic Management and NFS Shared Filesystems	
4.4	Security: Munge Authentication for Slurm Nodes	
5	**Working and Component Analysis**	24-28
5.1	Slurm Workload Manager: Local Queue Management	
5.2	AI Agent: LLM-Powered Root-Cause Analysis	
5.3	Ansible: Automated Cloud Node Configuration	
5.4	AWS Integration: Dynamic Cloud Bursting Logic	
5.5	Log Analysis: Converting Cryptic Errors to Plain English	
6	**Implementation and Output**	29-34
6.1	Deployment of the 5-PC Physical Cluster	
6.2	AI Diagnostic Results: Real-time Log Remediation	
7	**Conclusion and Future Scope**	35-36
8	**References**	37

1. INTRODUCTION AND OVERVIEW OF PROJECT

1.1 PURPOSE

The purpose of the project titled **"AI-Augmented HPC Management: Autonomous Scaling and Intelligent Diagnostics for Slurm Clusters**" to develop a **self-optimizing hybrid HPC infrastructure** that combines AI-driven diagnostics with automated cloud scalability to eliminate the limitations of traditional on-premise clusters. By enabling intelligent fault analysis, zero-touch automation, and seamless cloud bursting, the system minimizes downtime, reduces operational complexity, and ensures continuous job execution while delivering a **scalable, resilient, and cost-efficient Slurm-based computing environment**.

1.2 AIM AND OBJECTIVE

- **Aim**:
  1. To build a **centralized Slurm cluster** with unified identity and shared storage
  2. To implement **AI-based real-time diagnostics** for automatic root cause analysis.
  3. To enable **autonomous cloud bursting** using AWS for elastic compute scaling.
  4. To achieve **zero-touch automation** using Infrastructure as Code (IaC).
  5. To reduce **job wait time and system downtime**.
  6. To optimize **operational cost** through intelligent scaling and automation

- **Objective**: The project seeks to achieve the following objectives:

  1. **Centralized Infrastructure**: To build a functional Slurm cluster using LDAP for unified identity and NFS and LVM for shared, consistent storage across all nodes.
  2. **AI-Powered Diagnostics**: To implement an AI agent that performs real-time Root Cause Analysis on system logs, converting cryptic errors into plain-English troubleshooting steps
  3. **Autonomous Cloud Bursting**: To enable "Elasticity" by using Ansible and Boto3 to automatically provision AWS instances when local compute resources are exhausted.
  4. **Zero-Touch Automation:** To eliminate manual configuration errors by using Infrastructure as Code (IaC) to synchronize environments between local hardware and cloud nodes
  5. **Operational Optimization**: To reduce system downtime and hardware costs by leveraging intelligent troubleshooting and "pay-as-you-go" cloud scaling

2. PREREQUISITES

2.1 HARDWARE AND SOFTWARE REQUIREMENTS

➢ **Hardware Requirements**

- **Login Node (Slurm Controller)**
  - Multi-core CPU (8 cores or higher)
  - **16–32 GB RAM**
  - High-speed SSD storage
  - Runs Slurm controller, LDAP, DNS, NTP, NFS, LVM management
  - Reliable high-speed network interface (1 Gbps / 10 Gbps)
- **Master Node**
  - Multi-core CPU (4–8 cores)
  - **8–16 GB RAM**
  - SSD storage
  - Used for user access, job submission, and compilation
  - Integrated with LDAP and shared NFS storage
- **Compute Nodes (Local Cluster[1&3])**
  - Multi-core CPUs (8+ cores per node)
  - **16 GB RAM or more per node**
  - Local SSD/HDD for OS and scratch space
  - Connected via high-speed LAN
- **Storage Node[Compute 2]**
  - High-capacity disks
  - RAID support for fault tolerance
  - Configured with **LVM** and exported via **NFS**
  - Provides shared and elastic storage to all nodes
- **Networking Infrastructure**
  - Managed switch (1 Gbps / 10 Gbps)
  - Low-latency, reliable internal network
- **Cloud Infrastructure (AWS)**
  - EC2 compute instances (on-demand / spot)
  - Elastic Block Store (EBS)
  - VPC with secure networking

➢ Software Requirements

- **Operating System**
  - Linux (Ubuntu Server)
- **Cluster & Resource Management**
  - **SLURM Workload Manager**
  - **MUNGE** (authentication)
- **Core Infrastructure Services**
  - **DNS**, **NTP**, **SSH**
  - **LDAP** for centralized identity
  - **NFS** for shared storage
  - **LVM** for flexible storage management
- **Automation & Configuration**
  - **Ansible** (IaC and node provisioning)
  - **Boto3** (AWS automation via Python)
- **AI & Monitoring**
  - **Python 3.x**
  - **Large Language Models (LLMs)** for log analysis
  - System log tools (journald, rsyslog)
  - Monitoring scripts / exporters
- **Cloud & Networking**
  - **AWS CLI**
  - IAM roles and security groups

Version Control & DevOps Tools

- **Git**
- Shell scripting (Bash)

3. Overall Description

3.1 Introduction

High Performance Computing (HPC) clusters form the computational backbone of modern scientific research, engineering simulations, data analytics, and artificial intelligence workloads. These environments are designed to deliver massive parallel processing capabilities through tightly coupled compute nodes coordinated by workload schedulers such as SLURM. Despite significant advances in scheduling efficiency and resource allocation policies, the operational management of HPC clusters remains largely manual, reactive, and administrator-driven.

Traditional HPC administration relies heavily on predefined rules, static thresholds, and post- failure troubleshooting. When jobs fail, administrators are required to manually inspect scheduler logs, system logs, and application outputs to determine the cause of failure. This process is time-consuming, error-prone, and highly dependent on administrator expertise. Furthermore, conventional schedulers lack semantic understanding of failures; they can detect that a job failed, but not why it failed or how to remediate it. As cluster scale and workload heterogeneity increase, this manual approach becomes a critical bottleneck, leading to reduced system availability, longer job turnaround times, and inefficient resource utilization.

In parallel, modern HPC workloads increasingly exhibit dynamic and bursty resource demands, particularly in domains such as machine learning, genomics, and large-scale simulations. Static on-premise clusters are often unable to accommodate sudden spikes in resource requirements, resulting in prolonged job queuing and underutilization during off-peak periods. While cloud computing offers elastic scalability, its integration into traditional HPC environments remains largely manual and policy-driven, lacking intelligent automation.

To address these limitations, this project titled “AI-Assisted HPC Cluster Administration: Smart Log Analysis and Auto Infrastructure Provisioning” proposes an intelligent, self-managing HPC framework that augments conventional cluster management with artificial intelligence–driven decision-making. The proposed system introduces a multi-agent AI management layer that operates alongside the SLURM scheduler, continuously observing job states, queue dynamics, node health metrics, and system logs in real time.

A key contribution of this framework is its ability to perform automated log interpretation and root cause analysis. Instead of presenting raw error messages to administrators, the AI agent analyzes job failure logs and generates structured diagnostic insights that explicitly answer:-

- What went wrong during job execution
- Why the failure occurred, including resource, configuration, or system-level causes
- How the issue can be resolved, through actionable remediation suggestions

- This significantly reduces mean time to diagnosis (MTTD) and minimizes dependence on manual log inspection.

In addition to intelligent diagnostics, the framework addresses the challenge of resource starvation through autonomous infrastructure scaling. When the AI agent detects prolonged job wait times such as jobs remaining in the SLURM PENDING (PD) state beyond acceptable thresholds it interprets this condition as insufficient computational capacity. In response, the system automatically provisions cloud-based compute nodes, integrates them into the existing SLURM cluster, and enables seamless job execution without user or administrator intervention.

By combining AI-driven observability, automated root cause analysis, and cloud-based elasticity, the proposed system transforms a traditionally static HPC environment into a self- healing, elastic, and adaptive hybrid cluster. This paradigm shifts HPC administration from a reactive, manual process to a proactive and autonomous one.

Key Innovation:

The fundamental innovation of this work lies in replacing manual troubleshooting and static infrastructure scaling with AI-driven autonomous decision-making, thereby improving cluster resilience, scalability, and operational efficiency.

3.2 Data Flow: Job Queuing to Cloud Provisioning

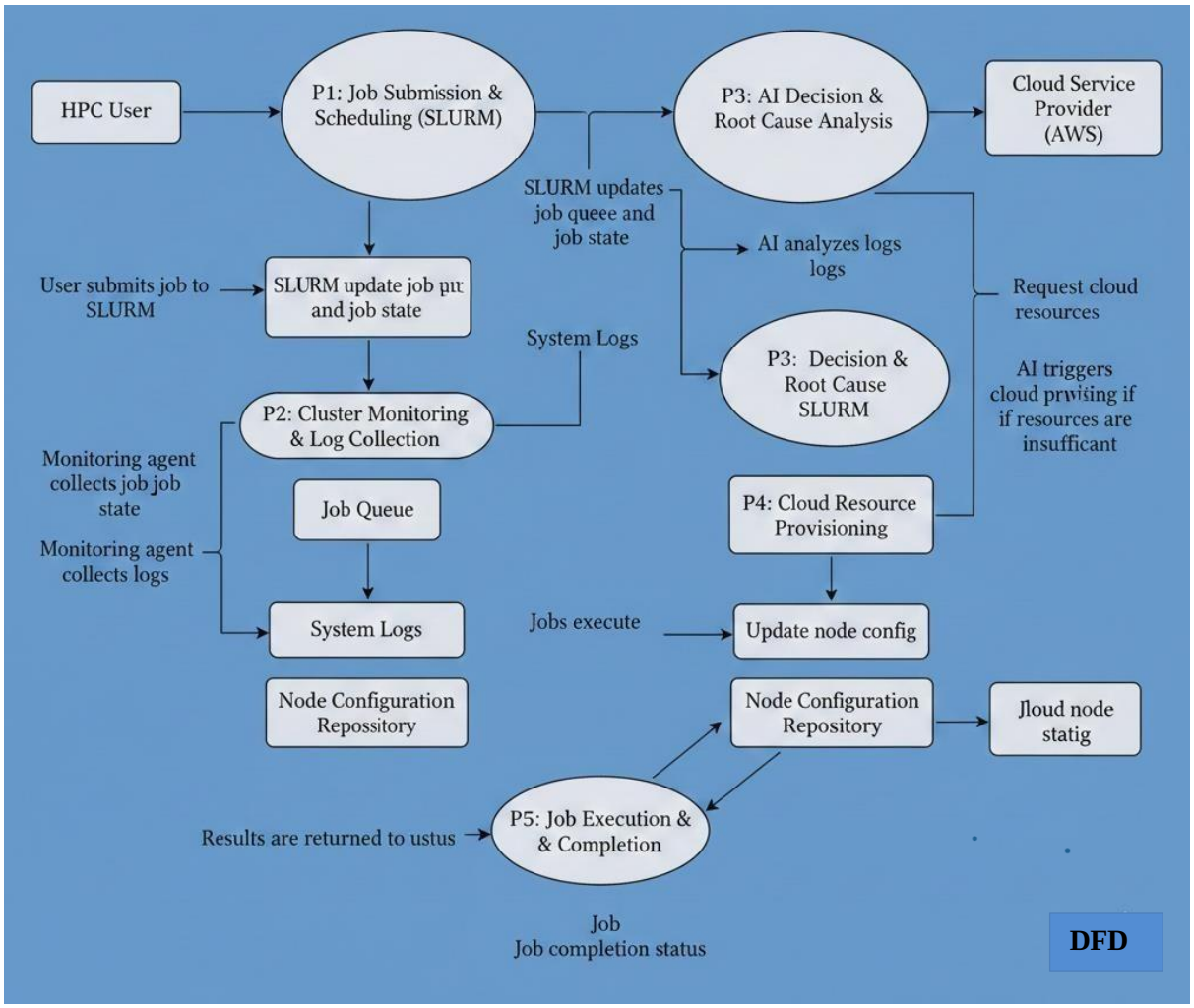


Fig. Dataflow Diagram

The Data Flow Diagram (DFD) illustrates the functional workflow of the proposed **AI-Assisted HPC Cluster with Cloud Bursting**, highlighting how computational jobs, monitoring data, and control decisions flow across different system components.

The data flow begins with the **HPC User**, who submits a batch job to the system using the SLURM workload manager. The job submission includes execution scripts and resource specifications such as CPU cores, memory, and wall-time. This input is processed by **Process P1: Job Submission and Scheduling (SLURM)**, where the scheduler validates the request and places the job into the scheduling queue with an initial job state.

SLURM continuously maintains job status information in the **Job Queue** data store. These job states, along with scheduler metadata, are forwarded to **Process P2: Cluster Monitoring and Log Collection**. The monitoring component collects real-time information related to job execution, queue wait times, node availability, and scheduler behavior. Simultaneously, system-generated logs from compute nodes, SLURM daemons, and operating system services are stored in the **System Logs** data store.

The collected job state data and system logs are then supplied to **Process P3: AI Decision and Root Cause Analysis**, which represents the intelligent decision-making layer of the system. This process analyzes scheduling delays, abnormal job behavior, and log patterns to identify conditions such as prolonged queue wait times, repeated job failures, or resource bottlenecks. Based on this analysis, the AI module determines whether the cluster is experiencing resource insufficiency or operational anomalies.

When resource starvation is detected, the AI decision module generates a **cloud resource provisioning request**, which is sent to the **Cloud Service Provider (AWS)**. This interaction is managed by **Process P4: Cloud Resource Provisioning**, where additional compute instances are dynamically launched. Once provisioned, the configuration details of these nodes are updated in the **Node Configuration Repository**, ensuring consistency in authentication, scheduling, and cluster integration.

The updated node configuration information is then utilized by **Process P5: Job Execution and Completion**. SLURM automatically schedules pending jobs onto the newly added cloud- based or existing on-premise compute nodes. Jobs are executed, and execution outputs along with job completion status are generated.

Finally, the job execution results and completion status are returned to the **HPC User**, completing the data flow cycle. This closed-loop workflow enables automated monitoring, intelligent decision-making, elastic resource scaling, and efficient job execution without manual administrative intervention.

Overall, the DFD demonstrates how the proposed system integrates workload scheduling, AI-driven analysis, and cloud bursting to transform a traditional static HPC cluster into a **self-managing, scalable, and intelligent computing environment**.

3.3 System Architecture

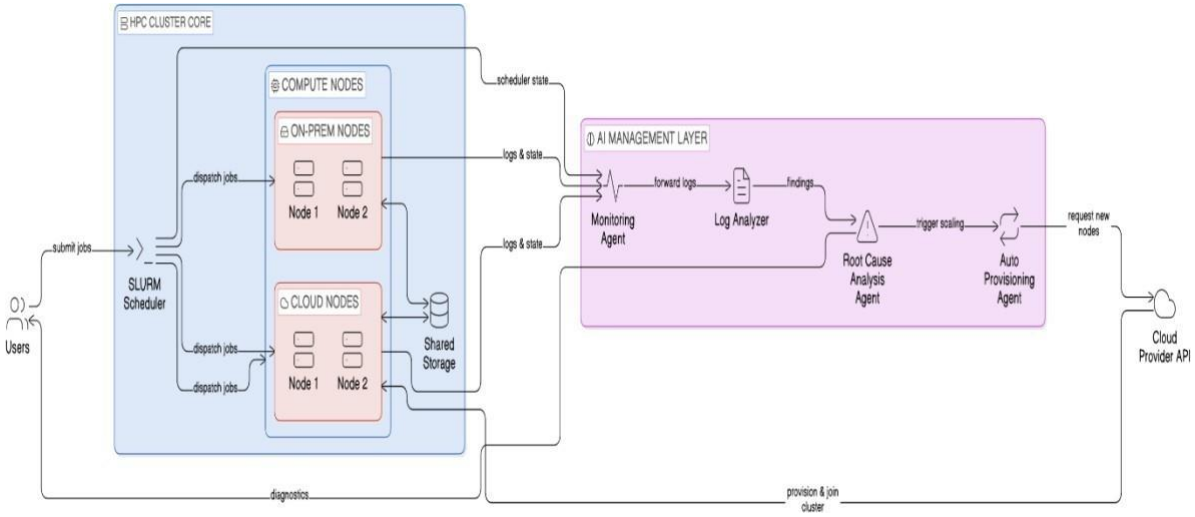


Fig. System Architecture

The proposed system architecture represents an AI-assisted, hybrid High Performance Computing (HPC) cluster management framework that integrates traditional on-premise compute infrastructure with elastic cloud resources under a unified scheduling and intelligent decision-making layer.

Conventional HPC clusters are primarily designed around fixed hardware capacity and rule-based administration. While such systems are effective for predictable workloads, they struggle to adapt to dynamic job patterns, bursty demand, and complex failure scenarios. As a result, operational efficiency degrades as cluster size and workload diversity increase.
The architecture is specifically designed to address critical limitations of conventional HPC clusters, including:
Static resource provisioning, where compute capacity remains fixed regardless of workload demand, Long job queue wait times, caused by resource contention during peak usage Manual fault diagnosis, which relies on administrator expertise and reactive troubleshooting Reactive infrastructure scaling, where scaling decisions are delayed or inefficient
By combining SLURM-based workload management, AI-driven observability and root-cause analysis, and automated cloud provisioning, the system enables autonomous, scalable, and fault-aware cluster operation.

At a high level, the architecture is composed of three major layers:

- **HPC Cluster Core**
- **AI Management Layer**
- **Cloud Provider Interface**

Each layer performs a well-defined role while maintaining loose coupling through structured control and telemetry interfaces, ensuring modularity, extensibility, and safe automation.

3.2.1 HPC Cluster Core

The HPC Cluster Core constitutes the execution backbone of the system and encapsulates all components responsible for job submission, scheduling, execution, and data access. This layer preserves the traditional HPC operational model, ensuring compatibility with existing user workflows and administrative practices.

i) User Interaction Layer

Users interact with the cluster through standard HPC submission workflows using SLURM commands such as sbatch, srun, and salloc. From the user's perspective, the system behaves like a conventional HPC environment, ensuring backward compatibility and zero disruption to existing workflows.
Once a job is submitted:

- Job metadata is registered with the SLURM Scheduler
- Resource requirements such as CPU cores, memory, wall-time, and node constraints are captured
- The job is placed into the scheduling queue

At no point are users required to be aware of AI-driven automation or cloud integration. This abstraction ensures that intelligent cluster management remains transparent, preserving the simplicity and predictability expected in HPC environments.

ii) SLURM Scheduler

The SLURM Scheduler functions as the central control plane for workload management and remains the authoritative scheduling authority within the system.
Its responsibilities include:

- Maintaining the global cluster state
- Managing job queues, priorities, and fair-share policies
- Dispatching jobs to available compute nodes
- Tracking job execution states such as PENDING, RUNNING, COMPLETED, and FAILED

Crucially, SLURM is not replaced or modified by the AI layer. Instead, the AI system operates as an intelligent supervisory layer that observes scheduler behavior and acts only through well- defined administrative interfaces.
Scheduler state information including queue depth, job wait times, node availability, and resource utilization is continuously exposed to the AI Management Layer for analysis and decision-making.

iii) Compute Nodes Subsystem
The compute subsystem forms a hybrid execution fabric consisting of both on-premise and cloud-based compute nodes. This design allows the cluster to maintain a stable baseline capacity while dynamically extending resources when required.

iv) On-Premise Compute Nodes
On-premise compute nodes represent the baseline computational capacity of the cluster. These nodes:

- Are statically provisioned
- Operate within the institutional or enterprise data center
- Are tightly integrated with shared storage, authentication, and networking services

They primarily handle:

- Regular and predictable workloads
- Long-running batch jobs
- Cost-sensitive or data-local computations

During execution, these nodes continuously generate:

- Job execution logs
- Resource utilization metrics (CPU, memory, I/O)
- Health and status signals

This telemetry is forwarded to the AI Management Layer, enabling real-time monitoring, anomaly detection, and historical trend analysis.

v) Cloud Compute Nodes
Cloud compute nodes act as elastic, on-demand extensions of the cluster. They are dynamically provisioned when local resources are insufficient to meet workload demand or quality-of-service constraints.
Key characteristics include:

- Temporary instantiation as virtual machines
- Automated configuration to match the cluster runtime environment
- Seamless registration as SLURM compute nodes

Once provisioned, cloud nodes are indistinguishable from on-premise nodes from the scheduler's perspective, allowing SLURM to dispatch jobs uniformly without special handling or reconfiguration.

vi) Shared Storage Subsystem

A centralized shared storage subsystem provides a unified data namespace across all compute nodes, both on-premise and cloud-based.

This ensures that:

- Input datasets are accessible from any execution node
- Output results are centrally persisted
- Job migration across heterogeneous nodes remains transparent

The shared storage layer plays a critical role in maintaining data consistency, reproducibility, and execution correctness across hybrid environments.

3.2.2 AI Management Layer

The AI Management Layer is the defining innovation of the proposed architecture. It operates as an intelligent supervisory control system that continuously observes cluster behavior, diagnoses inefficiencies, and triggers corrective actions.

This layer is logically decoupled from the scheduler, ensuring safe, explainable, and auditable automation without interfering with core scheduling semantics.

i) Monitoring Agent

The Monitoring Agent acts as the sensory system of the architecture.

It continuously collects:

- SLURM scheduler state
- Job queue statistics
- Node health and utilization metrics
- System and application logs

Data is ingested in near real-time, allowing the AI system to maintain a time-synchronized, global view of cluster behavior.

ii) Log Analyzer

The Log Analyzer performs semantic and statistical analysis on collected logs to extract meaningful patterns such as:

- Repeated job failures
- Resource starvation indicators
- Node-specific error signatures

iii) Root Cause Analysis (RCA) Agent

The Root Cause Analysis Agent applies AI/ML-based reasoning to determine the underlying causes of detected anomalies or performance degradation.
Typical inferences include:

- Insufficient compute capacity leading to prolonged queue times
- Node misconfiguration or partial hardware failures
- Memory under-allocation causing job termination
- I/O contention on shared storage systems

By correlating scheduler state, node telemetry, and historical patterns, the RCA Agent produces explainable, actionable diagnostics rather than opaque decisions.

iv) Auto-Provisioning Agent

The Auto-Provisioning Agent converts analytical insights into infrastructure-level actions. When the RCA Agent identifies resource scarcity as the primary bottleneck:

- A scaling trigger is generated
- Provisioning requests are sent to the Cloud Provider API
- New cloud compute nodes are instantiated and integrated

This closed-loop feedback mechanism enables self-adaptive cluster scaling driven by workload behavior rather than static thresholds.

3.2 3.Cloud Provider Interface

The Cloud Provider Interface exposes elastic execution capacity through secure API interactions.

Through this interface, the system can:

- Provision and terminate compute instances
- Configure networking and access controls
- Bootstrap the HPC software stack
- Register nodes with the SLURM scheduler

Once workloads complete, idle cloud nodes are automatically decommissioned, ensuring cost-efficient resource utilization.

1. Control and Data Flow Integration

The architecture explicitly separates:

- Control signals (job dispatch, scaling triggers, provisioning commands)
- Telemetry data (logs, metrics, scheduler state)

This separation ensures:

- Predictable scheduling behavior
- Safe automation boundaries
- High observability and debuggability

The system thus operates as a closed-loop intelligent HPC environment that continuously senses, reasons, and acts.

2. Architectural Significance

The proposed architecture advances traditional HPC cluster management by introducing:

- AI-driven operational intelligence
- Autonomous fault diagnosis
- Elastic hybrid scaling
- Minimal user-side complexity

By preserving SLURM as the authoritative scheduler while augmenting it with AI-based decision support, the architecture achieves practical deployability, scalability, and robustness, making it well-suited for modern research and enterprise HPC environments.

4. Cluster Setup and Configuration

4.1 Base Layer: DNS, NTP, and SSH Passwordless Communication

In an AI-augmented HPC cluster, the Base Layer is the "nervous system." Without these three elements, the Cloud Bursting logic would not be able to safely reach the AWS nodes, Slurm operations would fail due to clock desynchronization, and our agents would not be able to communicate.

1. DNS (Domain Name System)
Role in the Project:
By using human-readable names (such as master or compute1) rather than static IP addresses, DNS makes sure that your agents and Slurm services can locate one another.

IP Address	Machine Name
192.168.86.15	login
192.168.86.32	master
192.168.86.17	compute1
192.168.86.34	compute2
192.168.86.40	compute3

2. NTP (Network Time Protocol)
Role in the Project:
Munge, Slurm's authentication solution, is based on precise timing. Munge will fail and your jobs will remain in a "Pending" status indefinitely if the clocks on your local 5-PC cluster and your AWS cloud nodes diverge by more than a few seconds.
Agent Logging: For the dashboard or terminal to show the logs in the proper chronological order, Agent-1's and Agent-3's timestamps must coincide when they identify a failure and evaluate it.

- **Configuration**

  On Ubuntu 24.04, chrony is the preferred service.

- **File Name:** /etc/chrony/chrony.conf
- **Configuration (on Login Node):**

```
# Allow local nodes to sync from this master
  allow 192.168.86.0/24
  local stratum 8
```

- **Configuration (on Compute Nodes):**

  # Point to the Login Node as the source
  server 192.168.86.15 iburst

3. Passwordless SSH Communication

Role in the Project:

This is our Ansible automation's "Hands-free" method.

Autonomous Scaling: Ansible is activated when the Python agent notices a queue overflow. Ansible installs Slurm/LDAP automatically and logs onto the new AWS instance using SSH keys.

Slurm Job Launch: Slurm securely starts processes on the compute nodes using SSH-based protocols (via Munge).

Configuration Details

This involves generating a cryptographic key pair on the Login Node and distributing the "Public" half.

- **Key Files (Login Node):**

  * ~/.ssh/id_rsa (Private Key)

  ~/.ssh/id_rsa.pub (Public Key)

- **Key Files (Compute Nodes):**

  ~/.ssh/authorized_keys (Contains the string from the Login Node's .pub file)

4.2 Identity Management: LDAP Centralized Authentication

The "source of truth" for user identity in your MAS-HPC project is the implementation of LDAP (Lightweight Directory Access Protocol).You centralize the credentials on an LDAP server rather than handling local users on each individual node (Login, Master, and Computes).
User profiles and permissions are kept in a central database called LDAP. Any node that a user tries to log into "asks" the LDAP server whether the credentials are legitimate.

- **Roles of Users:-**

A. Cluster-admin (Internal Cluster User):
Access Level: Complete administrative access to the Master (192.168.86.32) and Login (192.168.86.15) nodes.
Role: In charge of launching the 4-agent system, maintaining luster_state.json, and supervising AWS Cloud Bursting.
UID/GID: Typically, all physical and cloud nodes are given a low-range UID (such as 1001).

B. External Cluster User (ext-researcher)
Access Level: Limited to job submission on the Login node (192.168.86.15).
Role: Uses Slurm (sbatch) to submit workloads, but is unable to access AWS scaling keys or change the MAS-HPC agent code.
UID/GID: Given a typical user UID, such as 2001. This UID must be recognized by the AWS cloud nodes so that Slurm can track file ownership.

- **The Authentication Flow in the Cluster**

Centralized Storage: we install an LDAP provider on the Login node. **Client Configuration:** All nodes (compute1, compute2, compute3, and the AWS node) are configured as LDAP Clients using sssd (System Security Services Daemon).

The Login Process: ext-researcher logs into the Login node via SSH. The Login node checks its local /etc/passwd. Finding nothing, it queries the LDAP server at 192.168.86.15.

Slurm Integration: Slurm uses these same UIDs to ensure that a job submitted by ext-researcher on the Login node has the correct permissions when it executes on compute1 or aws-node1.

1.1 Storage: LVM Elastic Management and NFS Shared Filesystems

Local storage flexibility and centralized data sharing are powerfully combined when LVM (Logical Volume Manager) and NFS (Network File System) are integrated into a cluster. This configuration guarantees that all nodes in a cluster environment have access to the same datasets and that your storage can expand as needed.

1. Elastic Storage Management, or LVM

Between the filesystem and physical disks, LVM serves as an abstraction layer. It is utilized on the Storage Node in a cluster.

- **Important Use Cases for Our Cluster:**

Dynamic Resizing: You can add a new physical disk to the Volume Group and increase the Logical Volume (LV) without unmounting the drive or halting services if your project data expands unexpectedly.

Striping for Performance: To improve I/O performance for demanding cluster applications, you can distribute data across several physical drives (LVM Striping).

2. Shared Filesystems (NFS)

The "glue" that enables several nodes—such as your login node and worker nodes to access the same files at once is NFS.

- **Server Configuration File:** /etc/exports

```
# /etc/exports
/mnt/shared_data 192.168.86.0/24 (rw,sync,no_root_squash,no_subtree_check)
```

- **Client Configuration File:** /etc/fstab

```
# /etc/fstab
192.168.86.34:/storage/input /storage/input nfs defaults 0 0
192.168.86.34:/storage/output /storage/output nfs defaults 0 0
192.168.86.15:/home /home nfs defaults 0 0
```

.

Key Use Cases in our Cluster:

Centralized Home Directories: Users can view the identical home folder (/home) by logging into the login node or any worker node.

Shared Software Stack: You install Python, compilers, and libraries once in an NFS-mounted directory (such as /opt/software), and all nodes run them from there rather than installing them on each individual node.

Data Consistency: Node B of the cluster can read an output file that is written by a job on Node A to an NFS-mounted /data directory immediately

1.2 Security: Munge Authentication for Slurm Nodes

MUNGE (MUNGE Uid Native Group Enforcement) is the "handshake" protocol in your cluster. Slurm need a method to verify that a user submitting a job from the login node is who they claim to be before the compute node runs the code because we have a distributed configuration with a login node (192.168.86.15) and compute nodes like Compute2 (192.168.86.34).

- **Important Use Cases for Our Cluster:**

In the absence of MUNGE, your cluster would be susceptible to "Identity Spoofing."

Trustless Security: It enables Slurm to safely send user credentials over the network.

Non-Root Execution: This prevents unwanted access to other users' data by ensuring that when you submit a task as cluster_user, the compute node executes it as cluster_user with the appropriate permissions.

Feel of Single Sign-on: Slurm transports tasks between nodes without requiring you to re-authenticate or enter passwords once MUNGE is operational.

5..Working and Component Analysis

5.1 Slurm Workload Manager: Local Queue Management

Slurm's architecture is designed to provide fault tolerance and scalability while offering flexibility in job scheduling and resource management. Below are the main components involved in Slurm's working. The overall workflow of job scheduling and execution in Slurm follows a sequence of steps:

5.1.1 Job Submission

- **User Submits a Job:** The user submits a job using the sbatch command. The job submission includes resource requirements (e.g., number of CPUs, GPUs, memory), job runtime, and the partition to use.
- **Job Queueing:** Once submitted, the job is placed in the queue. The job will stay in the Pending (PD) state until the required resources are available for scheduling.

5.1.2 Job Scheduling

- **Slurmctld Scheduler:** The slurmctld daemon checks the available resources and attempts to find a suitable place for the job. If there is a matching node in the specified partition that satisfies the job's resource requirements, the job is scheduled to run.
- **Scheduling Policies:** Slurm uses various scheduling policies to determine the order of job execution. These include:
    - Priority-based Scheduling: Jobs are scheduled based on priority, which can be influenced by job type, user fairshare, and other factors.
    - Fairshare: Jobs are assigned priorities based on historical resource usage to ensure equitable access to resources.
    - Backfilling: Smaller jobs that do not require the full resource allocation may "backfill" available gaps in the schedule, helping to reduce idle time and improve resource utilization.
    - Job Preemption: In some cases, lower-priority jobs may be preempted to make room for higher-priority jobs.

5.1.3 Resource Allocation

- **Node Assignment:** Once the job is scheduled, slurmctld selects one or more available compute nodes that match the job's resource requirements (e.g., number of cores, memory, GPUs). These nodes are allocated for the job.
- **Job Execution on Nodes:** The slurmd daemons on the allocated nodes are responsible for executing the job. They set up the job environment, manage job execution, and monitor job progress.

5.1.1 Job Execution

- **Job Starts:** Once resources are allocated, slurmd starts the job by executing the user's program/script. During execution, slurmd continuously monitors job status and resource
- **Resource Usage Monitoring:** Slurmd keeps track of resource consumption (e.g., CPU time, memory usage, and optionally, GPU usage).

5.1.1 Job Completion

- **Job State Transition:** When the job finishes executing, slurmd marks the job as COMPLETED and sends the exit status back to slurmctld. The job is removed from the queue.
- **SlurmDBD Accounting:** If configured, SlurmDBD records the job's resource usage, duration, and other job-related data in its database for historical tracking and reporting.

5.1.2 Job Monitoring and Feedback

- **Job Status Queries:** Users can monitor the status of their jobs during execution using commands like squeue, scontrol, and sacct. These commands provide real-time information on job state, resource usage, and progress.
- **Job Feedback:** If the job encounters issues or crashes, slurmd reports the failure, and users can query job logs and diagnostic information to troubleshoot the problem.

5.2 AI-Agent: LLM-Powered Root-Cause Analysis

- The AI agent leverages LLMs to perform sophisticated analysis of system behavior, error patterns, and performance metrics. Built on a multi-agent framework, the system employs specialized agents for different diagnostic domains: job scheduling analysis, resource contention detection, network performance evaluation, and storage system diagnostics. LLM-based agentic systems are increasingly being explored for autonomous health monitoring and fault diagnosis in complex industrial. These systems can diagnose faults directly from data and provide explainable outputs through natural language.

5.2.1 Root-Cause Analysis Process

The diagnostic workflow follows a systematic approach:

- Data Collection: Aggregates logs, metrics, and system states from multiple sources.
- Pattern Recognition: Identifies anomalies and correlates events across components.
- Hypothesis Generation: Formulates potential root causes based on observed patterns.
- Validation: Tests hypotheses against historical data and known issue signatures.
- Recommendation: Provides actionable remediation steps with confidence scores.

LLMs are particularly effective in this process due to their ability to understand complex text, extract semantic meaning from code and logs, and perform automated root cause analysis. They can also classify failure types and provide contextual information for troubleshooting. An LLM-agent can gather relevant log information and performance metric.

5.2.2 Learning and Adaptation

The system continuously improves through feedback loops. When administrators confirm or correct diagnoses, this information updates the agent's knowledge base. Over time, the system develops cluster-specific expertise, recognizing unique failure modes and optimization opportunities particular to the deployment environment. Continuous learning and adaptation are crucial for LLM-based diagnostic systems, especially in dynamic environments where new failure modes can emerge, although current LLM systems

5.2.3 Integration with Decision-Making

Beyond diagnostics, the AI agent participates in proactive decision-making. By analyzing queue patterns and resource utilization trends, it generates scaling recommendations, suggests job scheduling optimizations, and identifies opportunities for workload consolidation or redistribution. This proactive approach can significantly reduce resource waste and improve scheduling performance in HPC environments.

5.3 Ansible Playbook: Automated Cloud Node Configuration

5.3.1 Automation Framework

Ansible playbooks orchestrate the entire lifecycle of cloud-burst nodes, from initial provisioning through configuration, integration with the Slurm cluster, monitoring, and eventual decommissioning. Ansible is an open-source IT configuration management, deployment, and orchestration tool that allows for reproducible environments to be created and configured from simple textual descriptions called "Playbooks". These playbooks are designed for idempotency, meaning they can be executed multiple times without altering the outcome beyond the initial application, and can safely handle partial failures and retries.

5.3.2 Node Provisioning Workflow

The provisioning workflow executes the following stages:

- Infrastructure Preparation: Creates security groups, networking configurations, and IAM roles.
- Instance Launch: Provisions EC2 instances with optimized configurations for HPC workloads.
- Base System Configuration: Installs required packages, configures kernel parameters, and sets up monitoring.
- Slurm Integration: Installs Slurm client components and registers nodes with the cluster. This may involve pushing a list of cluster hosts to each member upon setup.
- Application Deployment: Configures shared filesystems, software stacks, and job environments.
- Validation Testing: Runs diagnostic tests to verify node functionality before accepting jobs.
- Ansible is particularly useful for automating the installation and configuration .

5.3.3 Configuration Management

Configuration templates support multiple instance types and cloud providers. Variables control instance specifications, network settings, storage configurations, and software versions. The system maintains configuration consistency across on-premises and cloud nodes while accommodating cloud-specific optimizations. Ansible's YAML-formatted playbooks allow for clear orchestration of complex multi-tier workflows and can be organized into roles for better project management.

5.3.4 Decommissioning and Cleanup

When cloud resources are no longer needed, automated decommissioning ensures clean shutdown. The process drains jobs from nodes, captures final logs and metrics, unregisters nodes from Slurm, terminates instances, and cleans up associated cloud resources. This

5.3.5 Decommissioning and Cleanup

When cloud resources are no longer needed, automated decommissioning ensures clean shutdown. The process drains jobs from nodes, captures final logs and metrics, unregisters nodes from Slurm, terminates instances, and cleans up associated cloud resources. This prevents orphaned resources and unnecessary costs.

5.1 AWS Integration: Dynamic Cloud Bursting Logic

5.1.1 Cloud Bursting Architecture

The cloud bursting system implements intelligent scaling logic that determines when to burst to AWS, what instance types to use, and how many resources to provision. The decision engine considers multiple factors including queue depth, job characteristics, cost constraints, and performance requirements. Cloud bursting extends on-premises HPC infrastructure by dynamically adding cloud resources as needed, providing a seamless experience for users.

5.1.2 Scaling Decision Logic

The scaling algorithm evaluates burst triggers based on configurable thresholds. Primary triggers include sustained queue wait times exceeding acceptable limits, resource utilization above capacity thresholds, and predicted demand based on historical patterns. The system employs predictive scaling during known high-demand periods to ensure resources are available before queues build up. Intelligent prediction and dynamic scheduling frameworks can combine machine learning-based workload forecasting with adaptive resource allocation algorithms to anticipate resource demands and optimize scheduling for burst scenarios.

5.1.3 Instance Selection and Optimization

Instance type selection considers job requirements, cost efficiency, and availability. The system maps Slurm job specifications to optimal AWS instance types, considering CPU, memory, network performance, and specialized hardware like GPUs. Spot instance integration provides significant cost savings for fault-tolerant workloads, with automatic fallback to on-demand instances when spot availability is limited. Spot instances, while significantly

5.1.4 Cost Management

Cost optimization strategies include automatic termination of idle instances, preferential use of spot instances, consolidation of workloads to minimize instance count, and right-sizing based on actual utilization. The system maintains detailed cost tracking and provides reports comparing on-premises vs. cloud costs for different workload types. Combining conventional EC2 instances with cheaper spot and burstable instances can be a cost-effective approach, with mechanisms to handle potential performance degradation from spot instance revocations.

5.2 Log Analysis: Converting Cryptic Errors to Plain English

5.2.1 Log Processing Pipeline

The log analysis system processes messages from multiple sources including Slurm logs, system logs, application logs, and cloud provider logs. A unified ingestion pipeline normalizes log formats, extracts structured data, and correlates events across different components and time periods. Modern HPC systems generate vast amounts of log data, making automated analysis essential. Processing pipelines often involve parsing unstructured log text into a suitable input for data mining and anomaly detection.

Error Interpretation and Translation

Natural language processing transforms technical error messages into human-readable explanations. The system maintains a knowledge base of common errors, their causes, and solutions. When encountering new error patterns, the AI agent generates explanations based on code analysis, documentation, and similar historical issues. NLP techniques, including sentiment analysis, can be leveraged to automatically mine log messages for error detection and identify erroneous behaviors. The goal is to move from manual operation to automated error handling. This approach can significantly improve the debugging experience by analyzing diverse data sources, including log files, source code, and documentation.

5.2.2 Context Enhancement

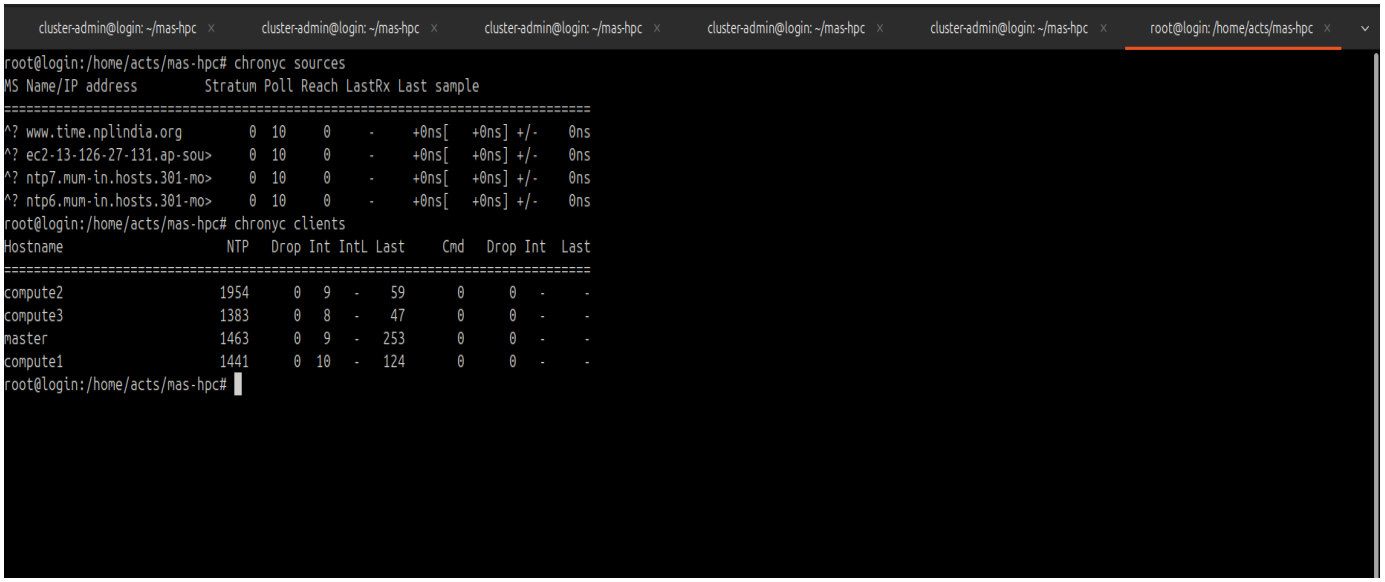
Raw error messages are enriched with contextual information. The system adds details about affected jobs, users, nodes, recent system changes, and related errors. This context helps administrators quickly understand the scope and impact of issues without extensive investigation. Correlating events across different components and time periods is crucial for comprehensive context enhancement.

5.2.3 Actionable Insights

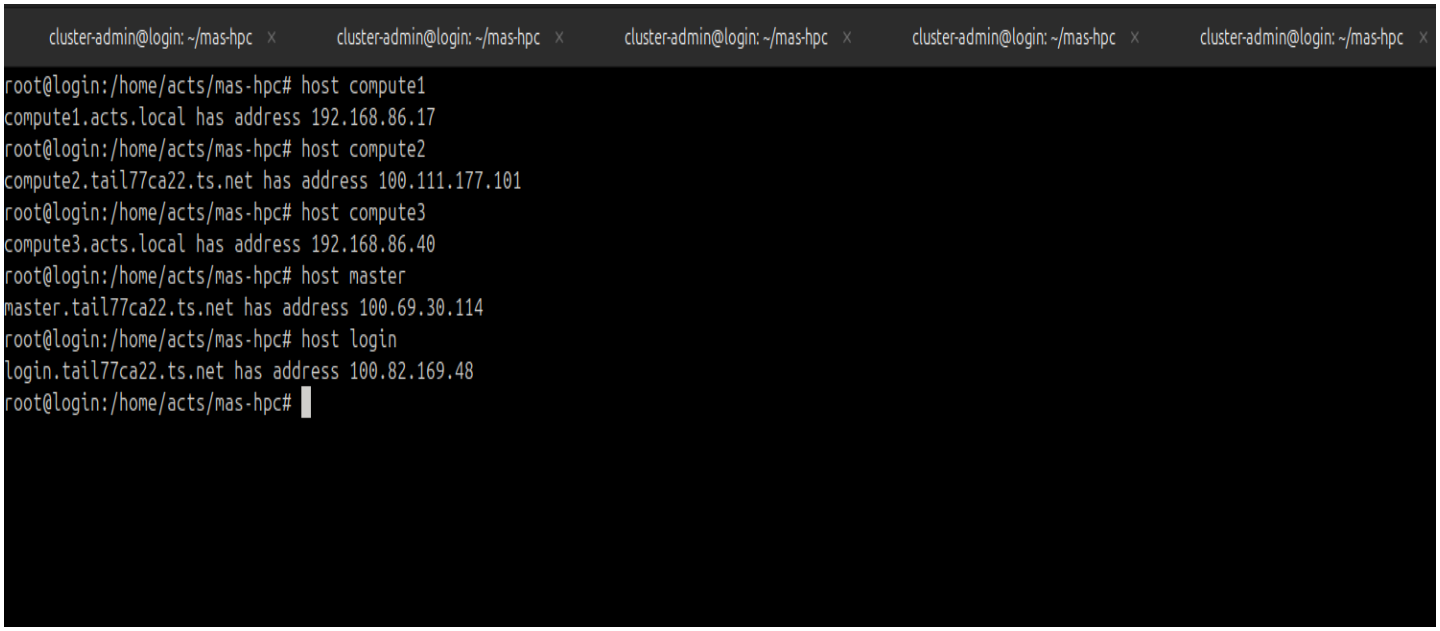
Beyond translation, the system provides specific remediation steps. For each identified error, it suggests potential solutions ranked by likelihood of success, provides relevant documentation links, and can automatically execute safe remediation actions when configured to do so. The system tracks which solutions prove effective, continuously improving its recommendations. This moves beyond simple error detection to providing actionable intelligence for system administrator

6.Implementation and Output

.NTP



DNS



SSH

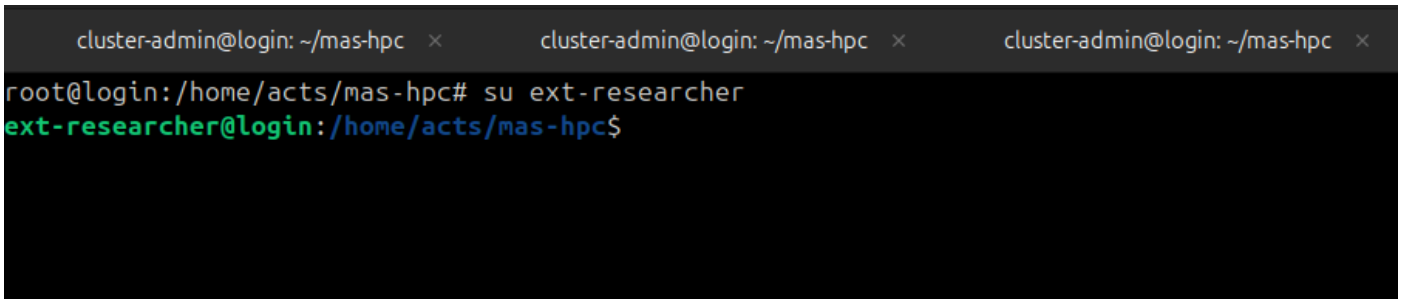
```
cluster-admin@login: ~/mas-hpc   cluster-admin@login: ~/mas-hpc   cluster-admin@login: ~/mas-hpc   cluster-admin@login: ~/mas-hpc
root@login:/home/acts/mas-hpc# ssh acts@compute1
Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 6.14.0-37-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

1 device has a firmware upgrade available.
Run `fwupdmgr get-upgrades` for more information.


Expanded Security Maintenance for Applications is not enabled.

271 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


1 device has a firmware upgrade available.
Run `fwupdmgr get-upgrades` for more information.

Last login: Fri Jan 23 13:29:44 2026 from 192.168.86.40
acts@compute1:~$
```

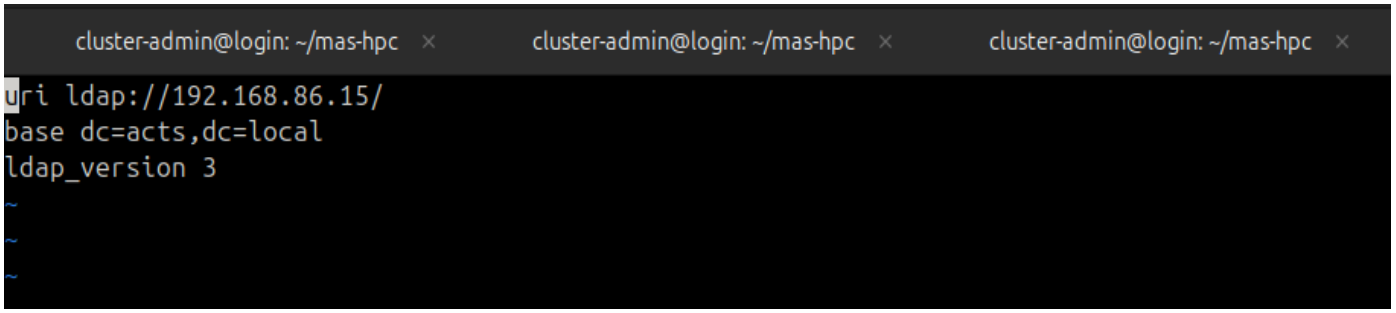
LVM & NFS

```
cluster-admin@login: ~/mas-hpc   cluster-admin@login: ~/mas-hpc   cluster-admin@login: ~/mas-hpc   cluster-admin@login: ~/mas-hpc
root@login:/home/acts/mas-hpc# df -h
Filesystem                   Size  Used Avail Use% Mounted on
tmpfs                        6.3G  2.7M  6.3G   1% /run
/dev/nvme0n1p5               330G   88G  225G  29% /
tmpfs                         32G  126M   32G   1% /dev/shm
tmpfs                        5.0M   16K  5.0M   1% /run/lock
efivarfs                     512K   50K  458K  10% /sys/firmware/efi/efivars
/dev/nvme0n1p1                96M   55M   42M  58% /boot/efi
192.168.86.34:/storage/input  157G     0  149G   0% /storage/input
192.168.86.34:/storage/output 157G     0  149G   0% /storage/output
tmpfs                        6.3G  196K  6.3G   1% /run/user/1000
/dev/sda2                    932G  585G  348G  63% /media/acts/New Volume
root@login:/home/acts/mas-hpc#
```

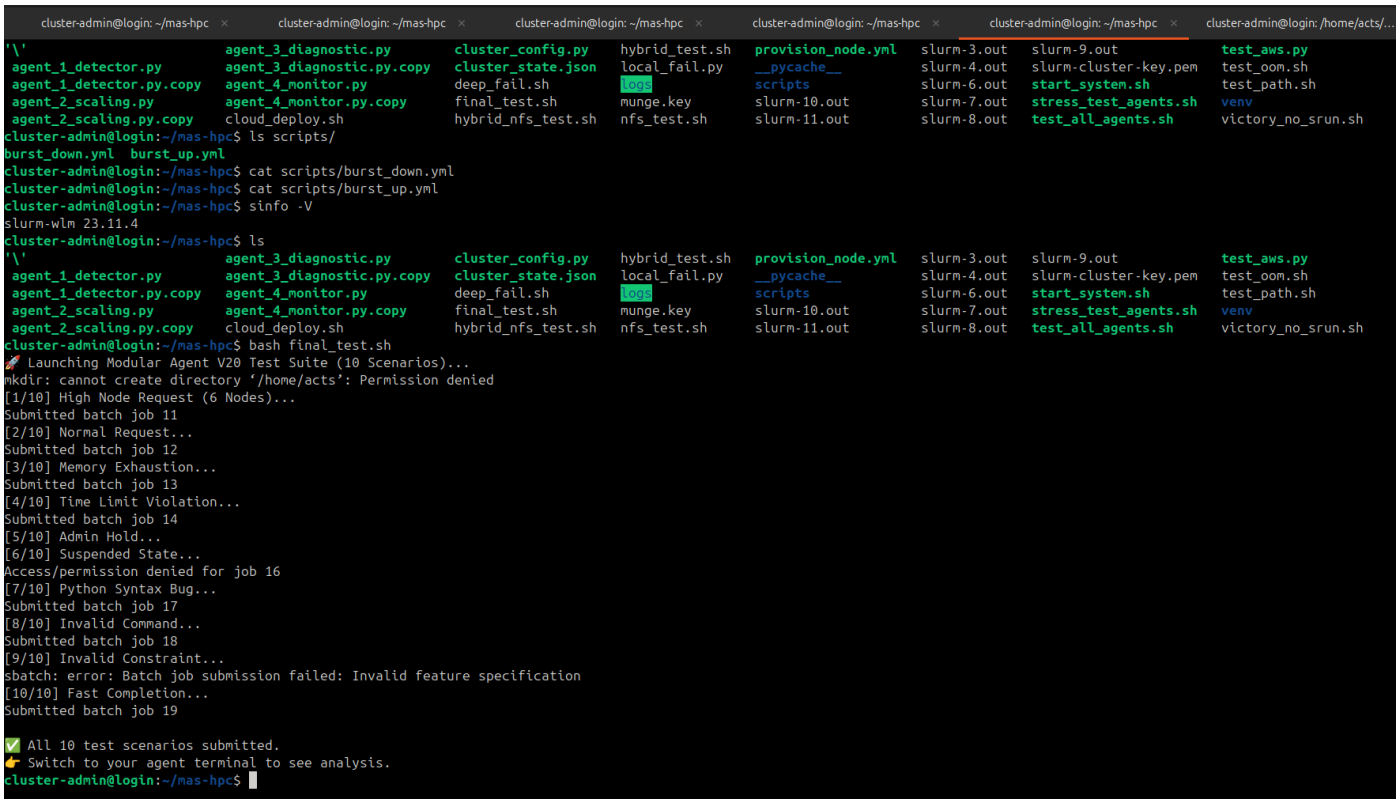
```
cluster-admin@login: ~/mas-hpc   cluster-admin@login: ~/mas-hpc   cluster-admin@login: ~/mas-hpc   cluster-admin@login: ~/mas-hpc
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
# / was on /dev/nvme0n1p5 during curtin installation
/dev/disk/by-uuid/00d97c25-193a-4e9e-af38-d621ad91b106 / ext4 defaults 0 1
# /boot/efi was on /dev/nvme0n1p1 during curtin installation
/dev/disk/by-uuid/0290-148C /boot/efi vfat defaults 0 1
/swap.img       none    swap    sw      0       0
192.168.86.34:/storage/input   /storage/input   nfs  defaults  0  0
192.168.86.34:/storage/output  /storage/output  nfs  defaults  0  0
~
~
~
```

LDAP:

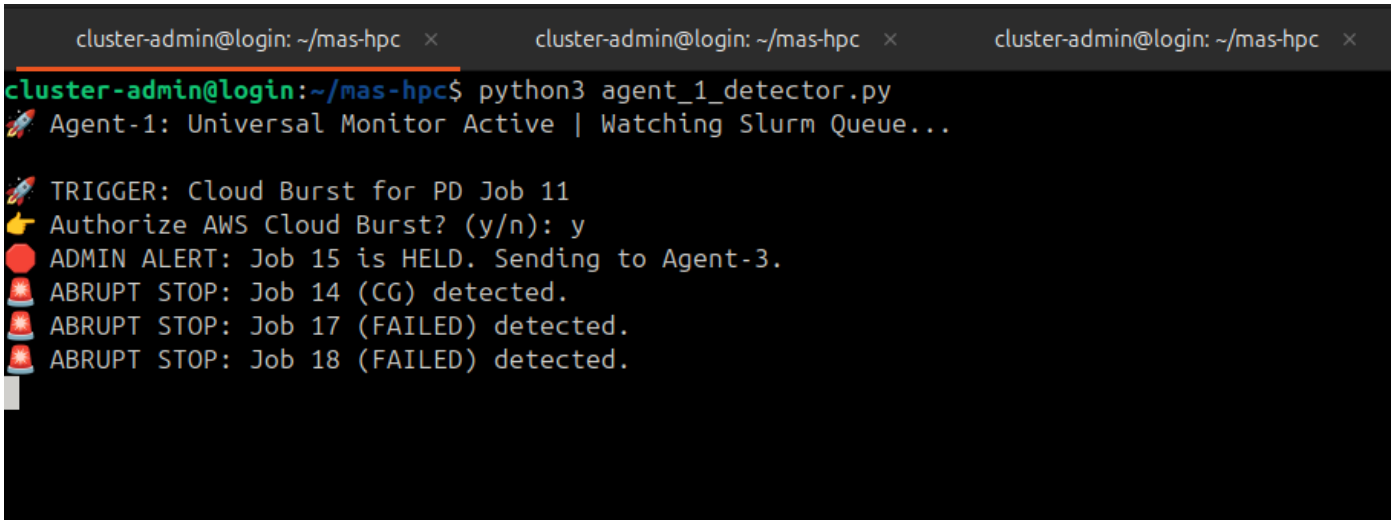

cluster-admin@login: ~/mas-hpc
cluster-admin@login: ~/mas-hpc
cluster-admin@login: ~/mas-hpc
root@login:/home/acts/mas-hpc# su ext-researcher
ext-researcher@login:/home/acts/mas-hpc$


cluster-admin@login: ~/mas-hpc
cluster-admin@login: ~/mas-hpc
cluster-admin@login: ~/mas-hpc
uri ldap://192.168.86.15/
base dc=acts,dc=local
ldap_version 3
~
~
~

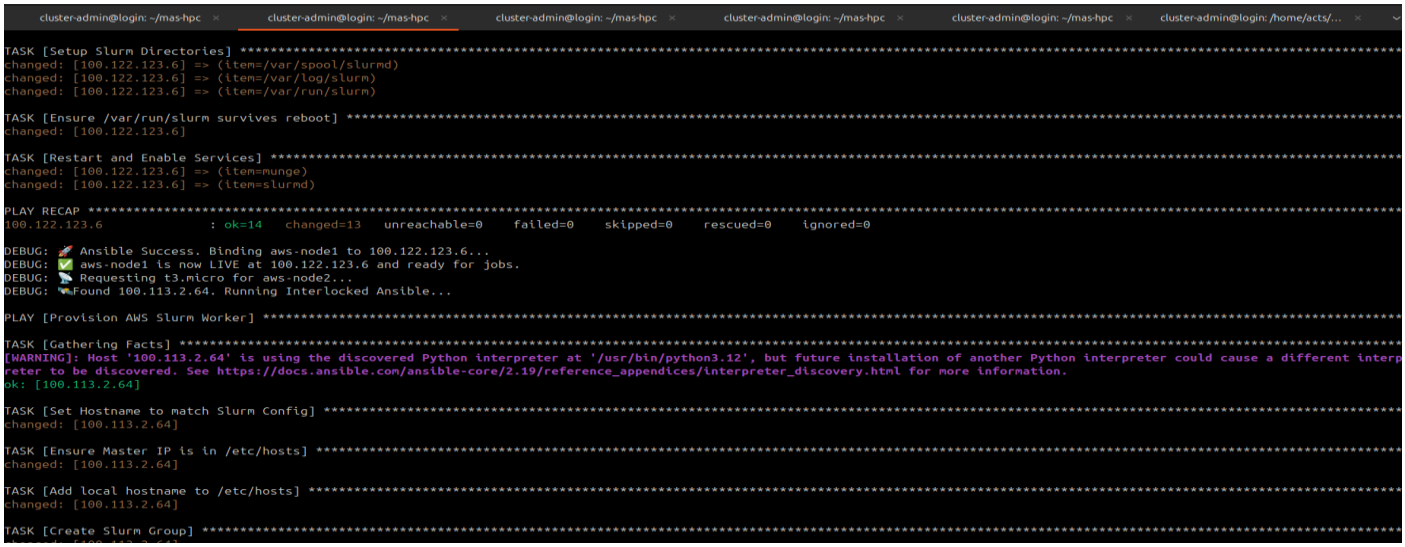
JOBS:


cluster-admin@login: ~/mas-hpc
cluster-admin@login: ~/mas-hpc
cluster-admin@login: ~/mas-hpc
cluster-admin@login: ~/mas-hpc
cluster-admin@login: ~/mas-hpc
cluster-admin@login: /home/acts/...
'\'
agent_1_detector.py
agent_1_detector.py.copy
agent_2_scaling.py
agent_2_scaling.py.copy
agent_3_diagnostic.py
agent_3_diagnostic.py.copy
agent_4_monitor.py
agent_4_monitor.py.copy
cloud_deploy.sh
cluster_config.py
cluster_state.json
deep_fail.sh
final_test.sh
hybrid_nfs_test.sh
hybrid_test.sh
local_fail.py
logs
munge.key
nfs_test.sh
provision_node.yml
__pycache__
scripts
slurm-10.out
slurm-11.out
slurm-3.out
slurm-4.out
slurm-6.out
slurm-7.out
slurm-8.out
slurm-9.out
slurm-cluster-key.pem
start_system.sh
stress_test_agents.sh
test_all_agents.sh
test_aws.py
test_oom.sh
test_path.sh
venv
victory_no_srun.sh
cluster-admin@login:~/mas-hpc$ ls scripts/
burst_down.yml burst_up.yml
cluster-admin@login:~/mas-hpc$ cat scripts/burst_down.yml
cluster-admin@login:~/mas-hpc$ cat scripts/burst_up.yml
cluster-admin@login:~/mas-hpc$ sinfo -V
slurm-wlm 23.11.4
cluster-admin@login:~/mas-hpc$ ls
cluster-admin@login:~/mas-hpc$ bash final_test.sh
Launching Modular Agent V20 Test Suite (10 Scenarios)...
mkdir: cannot create directory '/home/acts': Permission denied
[1/10] High Node Request (6 Nodes)...
Submitted batch job 11
[2/10] Normal Request...
Submitted batch job 12
[3/10] Memory Exhaustion...
Submitted batch job 13
[4/10] Time Limit Violation...
Submitted batch job 14
[5/10] Admin Hold...
[6/10] Suspended State...
Access/permission denied for job 16
[7/10] Python Syntax Bug...
Submitted batch job 17
[8/10] Invalid Command...
Submitted batch job 18
[9/10] Invalid Constraint...
sbatch: error: Batch job submission failed: Invalid feature specification
[10/10] Fast Completion...
Submitted batch job 19
All 10 test scenarios submitted.
Switch to your agent terminal to see analysis.
cluster-admin@login:~/mas-hpc$

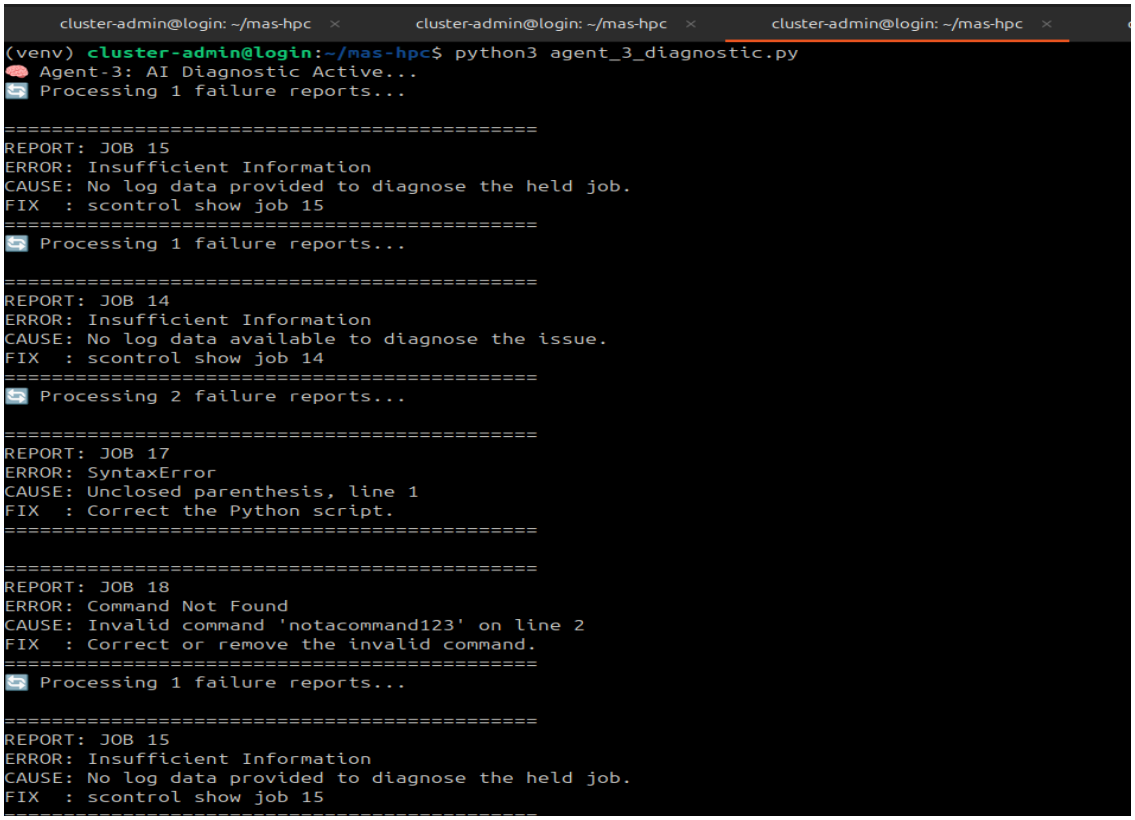
AI-Agent 1 Detector:



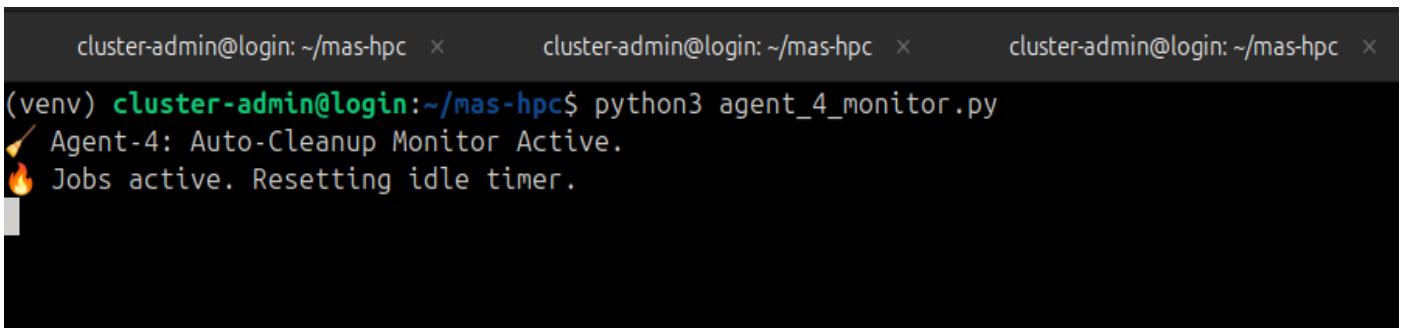
AI-Agent 2 Scaling:



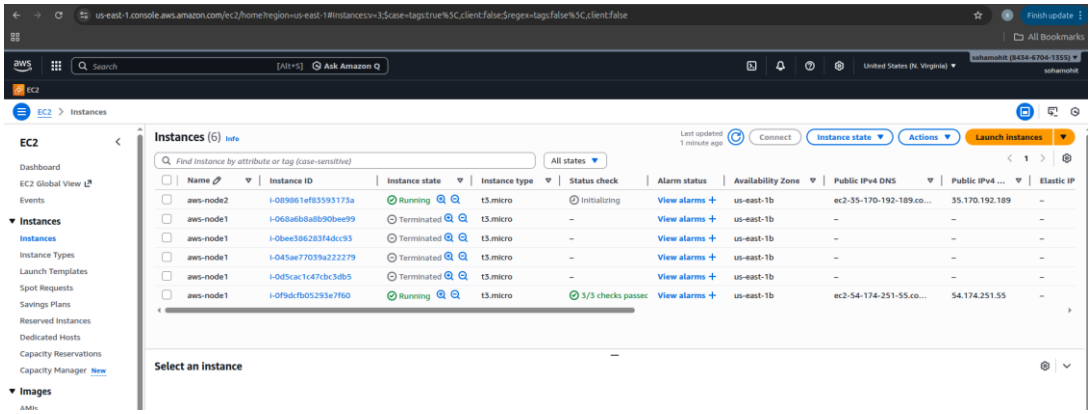
AI-Agent 3 Diagnostics



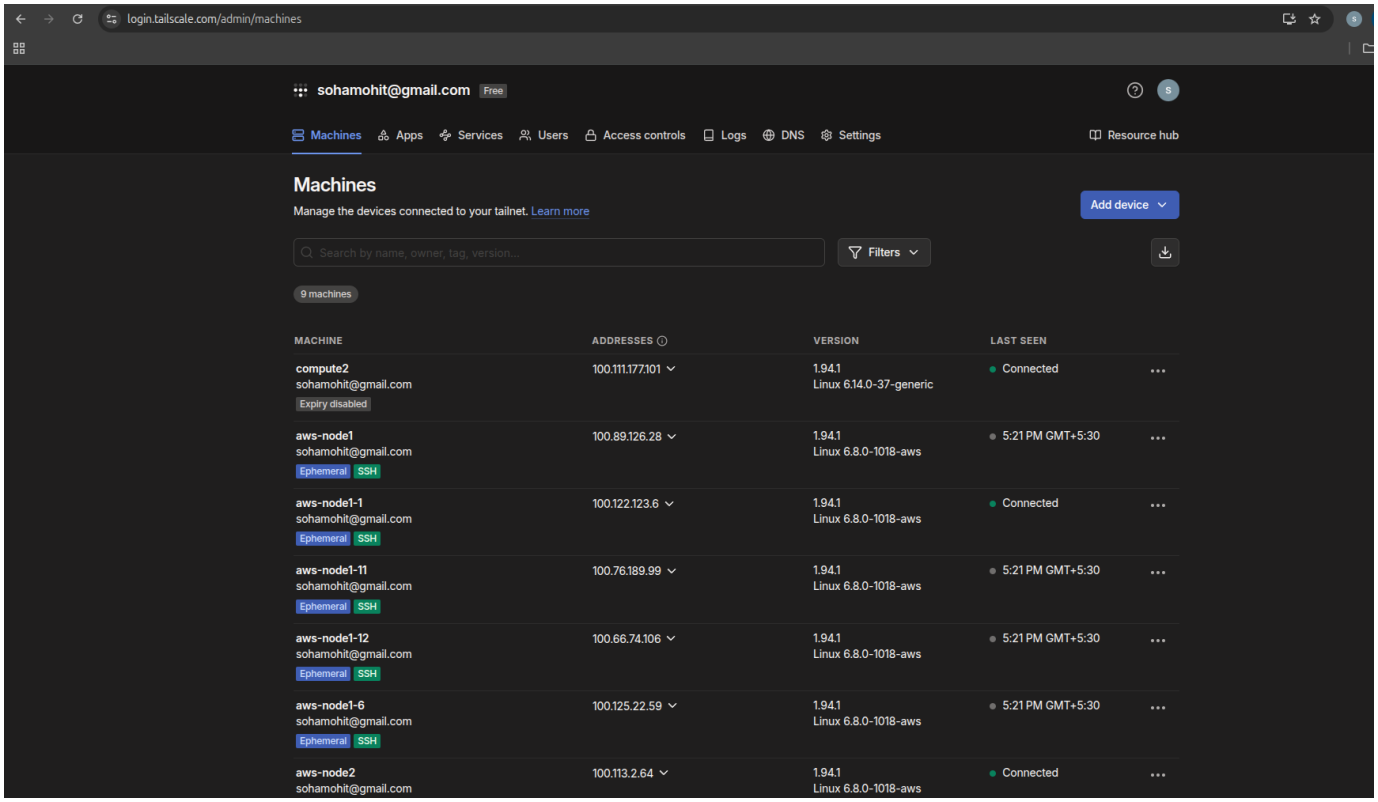
AI Agent 4 Monitoring



AWS –Instance



Tailscale:



7. Conclusion & Future Scope

This project presents a self-optimizing hybrid HPC environment that integrates a local Slurm cluster with an AI-driven orchestration layer to overcome the limitations of traditional static HPC infrastructures. Built on a reliable foundation of DNS, NTP, SSH, LDAP, NFS, and LVM, the system ensures secure authentication, consistent shared storage, and stable cluster operations across both on-premise and cloud resources.

A key contribution of the project is the Intelligent Diagnostics framework, where a Python-based AI agent leverages Large Language Models (LLMs) to perform real-time analysis of system and job logs. By translating complex and cryptic error messages into clear root-cause explanations and actionable fixes, the system significantly reduces troubleshooting time and enhances operational efficiency for HPC administrators.

In addition, the project implements an Autonomous Cloud Bursting mechanism that dynamically provisions AWS compute nodes using Ansible when local resources are insufficient. These cloud nodes are automatically synchronized with the existing cluster environment, enabling seamless, zero-touch scaling during peak workloads. Together, these innovtions demonstrate how AI-driven diagnostics and hybrid cloud automation can create a scalable, intelligent, and future-ready HPC management framework.

Future Scope of the Project

- Integration of **advanced AI/ML models** for predictive failure detection and proactive maintenance
- Extension to **multi-cloud environments** (Azure, GCP) for improved resilience and vendor flexibility
- Implementation of **fully self-healing automation**, where detected issues are automatically fixed without human intervention
- Support for **containerized workloads** using Singularity, Docker, or Kubernetes within the Slurm ecosystem
- Introduction of **cost-aware and energy-efficient scheduling** to optimize cloud usage and power consumption
- Enhancement of **security automation**, including AI-driven anomaly detection and compliance monitoring
- Scaling the architecture for **large enterprise, research, and national-level HPC deployments**

8.REFERENCES

https://www.isc.org/bind/
https://chrony-project.org/documentation.html
https://tailscale.com/kb/
https://slurm.schedmd.com/documentation.html
https://www.openldap.org/doc/admin26/
https://dun.github.io/munge/
https://docs.ansible.com/
https://ai.google.dev/gemini-api/docs/
https://boto3.amazonaws.com/v1/documentation/api/latest/index.html
https://osc.github.io/ood-documentation/

•