DECICE

DEVICE-EDGE-CLOUD INTELLIGENT COLLABORATION FRAMEWORK

Grant Agreement: 101092582

D2.3 AI Scheduler Prototypes for Storage and Compute

This project has received funding from the European Union's Horizon Europe Research and Innovation Programme under Grant Agreement No 101092582.

Document Information

Deliverable number:	D2.3
Deliverable title:	AI Scheduler Prototypes for Storage and Compute
Deliverable version:	0.1.0231111
Work Package number:	WP3
Work Package title:	Open Framework and Virtual Training Environment
Responsible partner	UniGö (UGOE) and (GWDG)
Due Date of delivery:	2023-11-30
Actual date of delivery:	2023-11-25
Dissemination level:	PU
Type:	R
Editor(s):	Michael Bidollahkhani (UGOE) Aasish Kumar Sharma (GWDG)
Contributor(s):	
Reviewer(s):	Jonathan Decker (UGOE) Mirac Aydin (GWDG)
Project name:	Device-Edge-Cloud Intelligent Collaboration framEwork
Project Acronym:	DECICE
Project starting date:	2022-12-01
Project duration:	36 months
Rights:	DECICE Consortium

Document History

Version	Date	Partner	Description
0.1	2023-07-29	NAG	Add-ons based approach literature review
0.2	2023-11-24	UGOE/GWDG	Internal review
1.0	2023-11-25	UGOE/GWDG	Finalizing the contents, redefining the method and descriptions

Acknowledgement: This project has received funding from the European Union's Horizon Europe Research and Innovation Programme under Grant Agreement No 10192582.

Executive Summary

In our pursuit of optimizing scheduling processes, we employ a Virtual Training Environment (VTE) to train, improve and test AI models for scheduling. This VTE simulates a Digital Twin (DT) based on predefined scenarios. Our training methodology involves running AI models against these scenarios, dynamically updating them, and assessing AI scheduler performance using specific metrics. This unique approach allows us to achieve faster-than-real-time training and enabling parallel training of multiple AI scheduler instances.

The proposed AI scheduler has the ability to view the entire DT, including the job queue and the state of running jobs. It optimizes resource allocation while respecting user specifications and can dynamically reschedule jobs for better placement. Unlike traditional heuristics-based schedulers, our approach leverages machine learning to gain a deeper understanding of system dynamics and optimizing resource allocation without increasing complexity unnecessarily.

Furthermore, we are actively exploring deep learning strategies to further enhance scheduling performance. The primary purpose of our AI scheduler is to optimize workflow and resource utilization, both with and without historical data. It assists the VTE controller in making informed decisions to handle incoming workloads effectively, contributing to enhanced resource pool management and workflow optimization. Our commitment to advancing scheduling capabilities within this framework underscores our dedication to improving resource utilization and workflow efficiency across various industries and applications.

Contents

1 Purpose and Scope of the Deliverable

The purpose of this deliverable is to outline the AI Scheduler, which is set to contribute to the optimization of workflows and resource utilization within a compute cluster based on available historical data, if any. The main purpose of the AI scheduler is to provide advanced scheduling capabilities to the DECICE framework. For the purpose of prototyping, any AI schedulers developed so far are integrated into the VTE (Virtual Training Environment), where they can be trained, tested and evaluated in terms of making optimal scheduling decisions for incoming workloads in the context of compute and storage.

2 Abstract / publishable summary

In this deliverable, we introduce innovative AI Scheduler prototypes for efficient storage and compute management in diverse computational environments. Our work focuses on developing AI-driven algorithms capable of optimizing task scheduling in compute environments consisting of cloud, high performance computing (HPC) and edge, which include diverse storage systems. We leverage deep learning models to predict the best allocation strategies, balance workloads and improve overall system performance. These AI Scheduler prototypes are designed to dynamically adapt to varying workloads, ensuring efficient resource utilization and reducing operational costs.

3 Project objectives

This deliverable contributes directly and indirectly to the achievement of all the macro-objectives and specific goals indicated in section 1.1.1 of the project plan:

Macro-objectives	Contribution of this deliverable
(O1) Develop a solution that allows to leverage a compute continuum ranging from cloud and HPC to edge and IoT.	The AI Scheduler prototypes aim to enhance the compute continuum capabilities by optimizing task allocation across diverse computational environments, including cloud, HPC, edge and IoT.
(O2) Develop a scheduler supporting dynamic load balancing for energy-efficient compute orchestration, improved use of green energy, and automated deployment.	The proposed prototypes contribute to dynamic load balancing, ensure energy-efficient orchestration and support the use of green energy in compute systems.
(O3) Design and implement an API that increases control over network, computing and data resources.	The deliverable contributes to the development of an API that enhances the control and efficiency of resource management, thereby improving the overall system performance.
(O4) Design and implement a Dynamic Digital Twin of the system with AI-based prediction capabilities as integral part of the solution.	This deliverable aids in implementing a Dynamic Digital Twin, equipped with AI-based predictive models, enhancing real-time decision-making and resource allocation.
(O5) Demonstrate the usability and benefits of the DECICE solution for real-life use cases.	The AI Scheduler Prototypes are tailored to demonstrate practical benefits in real-world scenarios, demonstrating their adaptability and efficiency in various use cases.
(O6) Design a solution that enables service deployment with a high level of trustworthiness and compliance with relevant security frameworks.	The proposed deliverable focuses on creating trustworthy and secure scheduling solutions that comply with relevant security frameworks to ensure the integrity and safety of the deployed services.

4 Changes made and/or difficulties encountered, if any

During the development of the AI Scheduler prototypes, we encountered several challenges. Initially, the complexity of integrating AI models into existing compute and storage systems posed significant technical difficulties. The target platform for our prototypes is Kubernetes, which offers dedicated options for integrating a compute scheduler, however, Kubernetes does not provide dedicated options for integrating a cluster-wide storage scheduler. In Kubernetes the provisioning of persistent volumes is up to storage drivers, which can apply their own rules and policies. Developing a fully agnostic storage scheduler is beyond the scope of the DECICE project so we had to compromise on integration with a few selected storage drivers. Furthermore, additional effort was required to identify and understand data sets, which could be used for training our prototypes.

5 Sustainability

The proposed AI Scheduler Prototypes contribute to the sustainability of computational resources in multiple ways. First, they promote efficient resource utilization, reducing energy consumption and operational costs in compute environments. Secondly, the prototypes are designed with scalability in mind, ensuring their long-term applicability in evolving computational landscapes. We have established links with other deliverables in the project, particularly those focusing on system architecture and cloud computing, to create synergies and integrate our solutions effectively. Key lessons learned include the importance of flexibility in AI model design to adapt to different system architectures and the need for continuous performance evaluation to ensure long-term viability.

6 Dissemination, Engagement and Uptake of Results

6.1 Target audience

As indicated in the Description of the project, the audience for this deliverable is:

✓	The general public (PU)
	The project partners, including the Commission services (PP)
	A group specified by the consortium, including the Commission services (RE)
	This report is confidential, only for members of the consortium, including the Commission services (CO)

6.2 Record of dissemination/engagement activities linked to this deliverable

See Table 1.

Type of dissemination and communication activities	Details	Date and location of the event	Type of audience activities	Zenodo Link	Estimated number of persons reached
None	N/A	N/A	N/A	N/A	0

Table 1: Record of dissemination / engagement activities linked to this deliverable

6.3 Publications in preparation OR submitted

See Table 2.

In preparation or submitted?	Title	All authors	Title of the periodical or the series	Is/Will open access be provided to this publication?
In preparation	HOSHMAND Algorithm: Next-Gen HPC Compute Scheduler	Michael Bidollahkhani, Aasish Kumar Sharma, Julian Kunkel	Pre-print	-

Table 2: Publications related to this deliverable

6.4 Intellectual property rights resulting from this deliverable

None.

6.5 GitHub

The source codes and the initial implementation of the AI schedulers can be found on the DECICE GitHub project page [DEC].

7 Detailed report on the deliverable

7.1 Background

In scenarios involving a compute continuum, which encompasses a range of devices from single board edge devices to servers with high-end hardware, identifying an optimal workflow is a complex task [Jan+23]. A compute continuum refers to a system that integrates cloud servers with HPC and edge devices such that managing workload distribution across such a diverse system requires a sophisticated control mechanism [DMO21]. The challenge lies in the inherent difficulty for human operators to effectively organize and administer these systems [Fan21]. This has led to the exploration of intelligent machine assistance, where machines are programmed to learn and apply their knowledge in action. In the field of Artificial Intelligence (AI), various algorithms, methods, and mathematical models are developed to simulate real system prototypes, aiming to enable machines to autonomously learn and respond to complex scenarios.

Specifically, for DECICE use cases, there is a need for an efficient resource orchestrator or scheduler capable of automating and optimizing workflows based on incoming user and system workloads. The goal is to create workflows that are highly efficient in terms of time and resource utilization, thereby reducing energy consumption and associated costs. The question then arises: What kind of scheduler can effectively meet these requirements? Our preliminary research, drawing upon recent AI advancements, suggests the potential of a scheduler such as *KAIROS* [Li+23], which manages workloads based on requested runtime and throughput. Building upon this, we propose a refined version named *Lil KAIROS*. This model is tailored to consider the available resources on each node during resource allocation, thereby enhancing the efficiency of runtime requests.

7.1.1 Illustrative Example

To elucidate the application of our AI-driven scheduling system, consider the following scenario: A job request necessitates a node equipped with 4 CPUs and 10 GB of memory for a specific computational task. Similarly, another job requests for 2 CPUs and 8 GB of memory. In our heterogeneous system environment, we have two nodes with distinct configurations. The first node features 2 CPUs and 4 GB of memory, while the second node comprises 1 CPU, 1 GPU, and 8 GB of memory. The challenge lies in determining the most efficient execution strategy for this job.

Key considerations include identifying the most suitable node(s) for job allocation and strategizing to minimize the total machine time, which encompasses both run time and waiting time. The ultimate objective is to optimize resource utilization within the constraints of our heterogeneous system. This scenario exemplifies the complex decision-making processes facilitated by our AI scheduling system, showcasing its capability to navigate and efficiently allocate resources in diverse computational landscapes.

7.2 System Architecture and Design

7.2.1 General Overview of Common Scheduling Systems Architecture

In addressing the challenges highlighted in our compute continuum scenario, particularly as depicted in the illustrative example in Section 7.1.1, it is imperative to have a thorough understanding of the system's architecture and resources. Traditionally, the Von-Neumann architecture, which primarily consists of CPU, Memory, Input and Output units, serves as the foundational model for most computing systems [AOE07].

Focusing on the CPU and Memory components, while temporarily setting aside the input and output aspects, we can explore various system designs. According to Flynn's taxonomy, systems can be classified into four distinct types:

1. SISD (Single Instruction Single Data), exemplified by Intel x86 and 64-bit architectures.
2. SIMD (Single Instruction Multi Data), such as NVIDIA AVX100, where CPUs and GPUs are combined.
3. MISD (Multi Instruction Single Data), a less common type not immediately relevant to our context.
4. MIMD (Multi Instruction Multi Data), for instance, Intel Xeon Phi, influenced by the Larrabee microarchitecture.

These types can further be categorized into systems with shared or distributed memory architectures.

In contemporary compute continuum environments as shown in Figure 1, hybrid or heterogeneous systems are prevalent. These systems feature a combination of SISD, SIMD, MISD and MIMD nodes, along with both shared and distributed memory models. To effectively manage such diverse configurations, our focus shifts to the development of specialized AI Schedulers:

1. **Compute AI Scheduler:** Tailored for managing computational tasks across varied processing units.

2. **Storage AI Scheduler:** Designed to handle data storage and retrieval, taking into account the unique requirements of different storage media.

The necessity for distinct AI Schedulers arises from the fact that compute and storage tasks have divergent characteristics and optimization parameters, making a one-size-fits-all approach impractical. This bifurcation allows for more targeted and efficient resource allocation and job scheduling in a complex, heterogeneous compute environment.

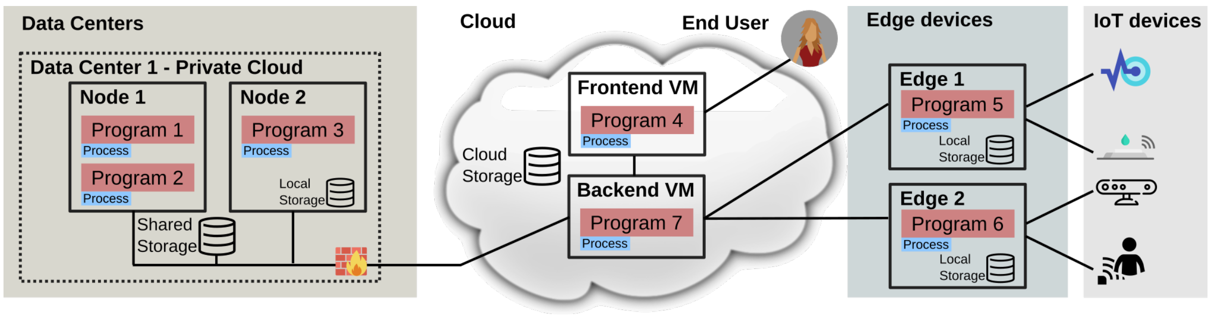

Figure 1: System Architecture Illustration

7.2.2 Architectural Overview of DECICE Project's AI Scheduler

In the context of the DECICE project, the architecture for the AI Scheduler is intricately designed to manage and optimize workloads within a heterogeneous computing environment. The prototype of the DECICE AI scheduler operates within a Synthetic Test Environment (STE), which functions as both a training and testing ground, simulating a production environment. This virtual training environment (VTE) is crucial for testing and training AI schedulers under controlled and repeatable conditions, especially when real-world training is impractical or too costly.

The architecture of the DECICE AI scheduler is comprehensive, encompassing various parts that interact with each other. The process initiates with the VTE controller, which triggers a metrics refresh in the Digital Twin (DT). The DT, a virtual representation of the physical compute continuum, accumulates node and job metrics and holds this data in memory for rapid processing. This data, stored as scenarios in the ScenarioDB, is processed and pushed to the DT. Once the metrics refresh is completed, the VTE controller is informed and can trigger the next operation via the scheduler controller.

The scheduler controller plays a pivotal role in prioritizing and filtering jobs and nodes to prepare them for scheduling. The DECICE framework incorporates multiple schedulers, including AI schedulers, integer linear programming (ILP) schedulers, and heuristic schedulers. Each scheduler assesses the suitability of nodes for job allocation based on a variety of factors, including resource availability and job requirements.

The scores generated by these schedulers are evaluated and aggregated, ensuring that the final scheduling decisions are well-balanced and optimized. This complex orchestration ensures efficient resource allocation, job scheduling, and workload management across the compute continuum, effectively addressing the challenges posed by the heterogeneous nature of modern computational environments.

7.3 System Modeling

The main elements in the HPC compute and storage system are shown in Figure 2, along with their respective interconnects. In the case of the compute continuum (Figure 1), there might be different

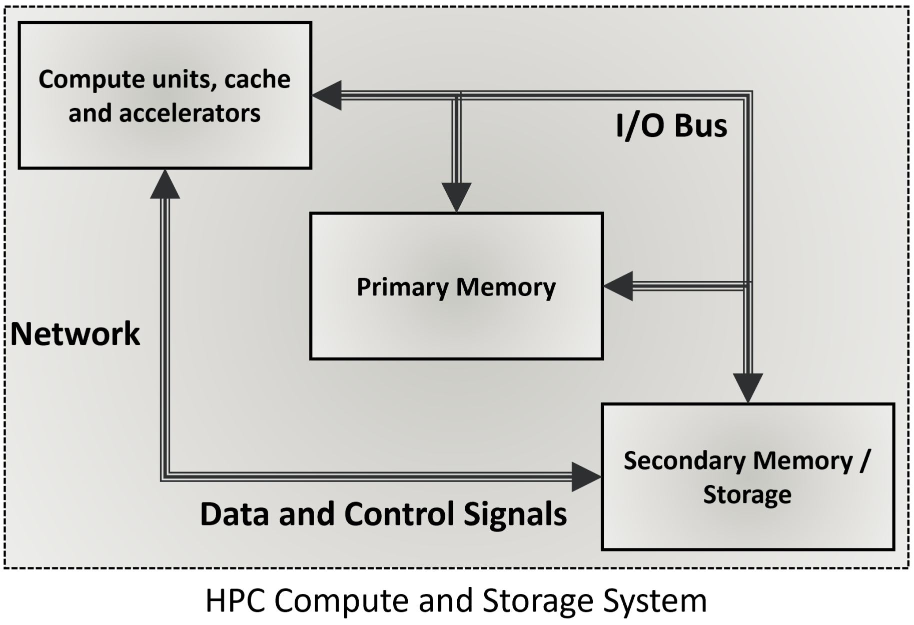
Compute units, cache and accelerators
I/O Bus
Primary Memory
Network
Secondary Memory / Storage
Data and Control Signals
HPC Compute and Storage System

Figure 2: A simple compute and storage nodes of an HPC system, along with their interconnects.

scenarios where the nodes have different compute capacities as well as storage systems (compute with cache, primary and secondary memory/storage) along with different connecting networks.

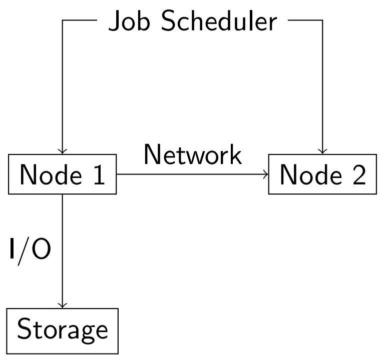

Figure 3: Simple scheduling instance with two compute nodes and shared storage.

7.4 Workflow Model

Figure 3 can be a good demonstration for a simplified workflow model of a single cycle scheduling, which is renovated for the complex systems with the design and implementation of the DECICE showcase workflow model (shown in Figure 4). Let us consider each cycle as a pool of Jobs J and there could be j number of jobs, i.e., $J = \{J_1, J_2, J_3, ..., J_j\}$. Each job has at least one Task T, where there might be n number of Tasks i.e., $T = \{T_1, T_2, T_3, ..., T_n\}$. Each task requests or

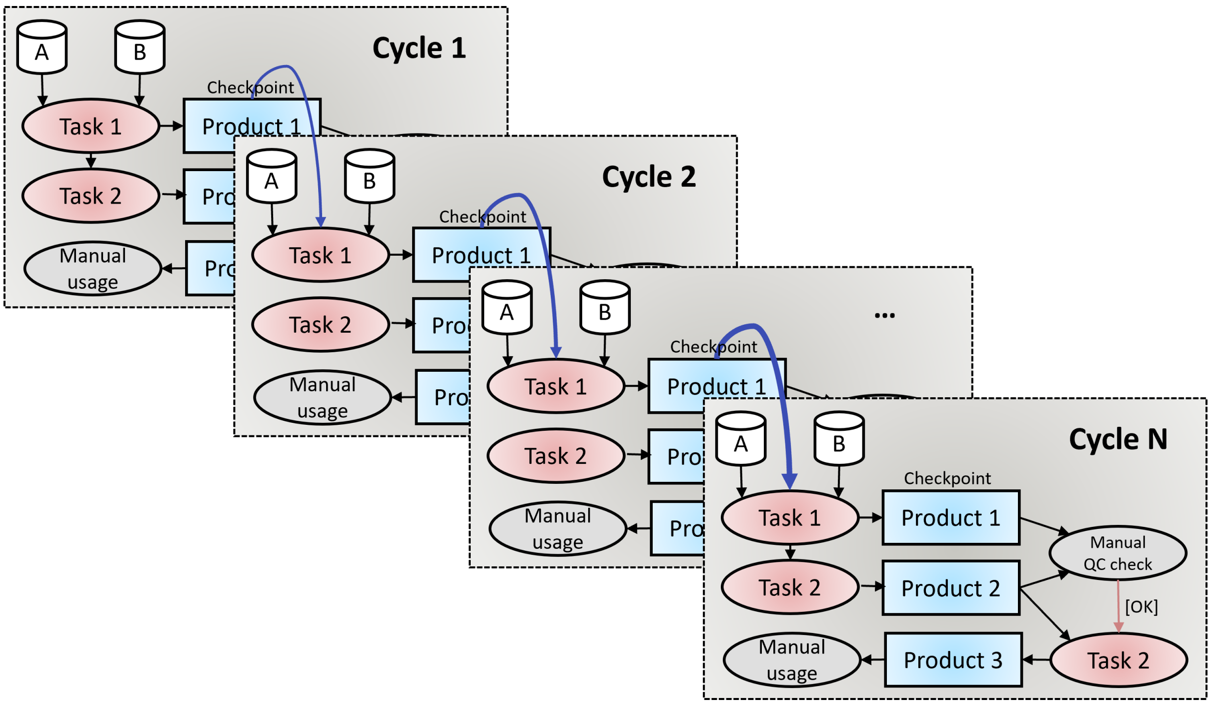

Figure 4: A cyclic workflow of an instance forecast (DECICE proposal).

demands some resources, suppose these resources are termed as machines M and there could be m number of machines with the same or different capacities, i.e., $M = \{M_1, M_2, M_3, ..., M_m\}$, finally, t_e be the amount of estimated time demanded/requested for a single machine. Now, the total requested time (t_e) for a single job with a single task executed in a single machine is given by

$$t_e(J_{j=1}) = t_e(T_{n=1}, M_{m=1}) \tag{1}$$

where $t_e(T_n, M_m)$ is the time taken for the task. This time could be different for different tasks executed in different machines that have different capacities. Therefore, the total time taken for a job with more than one task executed in different machines (same or different capacity) is given as

$$t_e(J_{j=1}) = \Sigma^n_{T_{n=1}} \Sigma^m_{M_{m=1}} t_e(T_n, M_m) \tag{2}$$

Likewise, the total work done by a machine (M_1) is given by

$$\eta = \frac{\Sigma^j_{J_{j=1}} \Sigma^n_{T_{n=1}} t_e(T_n, M_1, J_j)}{\Sigma^j_{J_{j=1}} \Sigma^n_{T_{n=1}} t_r(T_n, M_1, J_j)} \times 100\% \tag{3}$$

where $t_e(T_n, M_1, J_j)$ and $t_r(T_n, M_1, J_j)$ is the estimated-time and the real-time taken by the machine(M_1) to complete the job.

Now, the problem for a scheduler is how to map these jobs. Which machine to allocate for execution for a particular job? Finally, from where and how to handle the respective I/O (Input/Output device or storage) to optimally utilize the system.

Considering the I/O part of the evolving architecture of upcoming exascale High-Performance Computing (HPC) systems, traditionally, HPC storage systems were simple, with a parallel back-end file system and tape-based archives. However, the trend is shifting towards a more complex multi-tier storage hierarchy. This new architecture includes features such as node-local non-volatile main memory (NVMM) with performance comparable to DRAM, NVMe-based SSDs within compute nodes boasting high bandwidth, SSDs on I/O nodes, parallel file systems, campaign storage and archival storage [Car+20].

This highlights a challenge in the current architecture of supercomputers, where the I/O stack is not utilized optimally due to a lack of information about the state of HPC resources and I/O accessed by multiple applications. This sub-optimal usage results in missed opportunities for global I/O optimization, leading to issues such as redundant data movement, contention for large I/O accesses, and delayed overall performance. To address these issues and optimize I/O performance for multiple applications sharing resources, two main problems need attention: allocating applications to proper compute resources and managing I/O data through scheduling to prevent conflicts.

Suppose that, there are I/O or storage systems having a number of I/O nodes at different locations, i.e., $(IO_1, IO_2, ..., IO_n)$ that are available to perform I/O operations from the compute nodes to the respective storage location, then for each job $\frac{t_d(l,l')}{d_{BW}}$ there will be data which is transferred from location l to location l' using bandwidth d_{BW}. Likewise, there could be n_i successive blocking and non-blocking operations for compute nodes or machines adding up waiting time cost $t_w(l, l')$ for the job (in an aggregation of all the tasks of a job). So I/O overhead time costs t_{IO}, i.e., data transfer cost and waiting time cost for these are given by

$$t_{IO}(J_j) = \Sigma_1^j t_{w_j(l,l')} + \frac{t_{d_j(l,l')}}{d_{BW}} \tag{4}$$

With that, the total time cost or the overall time cost for the execution of j jobs is given by

$$t_o(J_j) = \Sigma_1^j t_e(J_j) + t_{IO}(J_j) \tag{5}$$

7.4.1 Mapping Use Cases

The possible mapping use cases are:

1. J job requesting for M machines (resources) for T amount of time
2. A job executing a single task in a single machine with no task splitting (homogeneous case)
3. A job executing many tasks across different machines (heterogeneous case)

If we evaluated the system I/O performance by executing the use cases under two different configurations like (i) single job and (ii) multiple jobs then:

1. In a single job case, one task could execute exclusively, having full access to the complete I/O bandwidth provided by the node.
2. In contrast, with multiple jobs, several task instances could execute simultaneously (accessing two different storage (files)) and competing for the same I/O resources.

In this case, the total bandwidth is degraded by d which is given by

$$d = 1 - \frac{\Sigma d_{BW_{(MultipleJobs)}}}{d_{BW_{(SingleJob)}}} \tag{6}$$

where $d_{BW_{(MultipleJobs)}}$ is the aggregated data bandwidth of multiple jobs configurations and $d_{BW_{(SingleJob)}}$ is the bandwidth of single job configuration.

7.4.2 Scheduling Use Cases

Besides, all the above cases, in the most trivial case, a scheduler should follow these policies:

- Case I:
  - When a job has at least a single task and the demand for resources is out of the capacity (available resource) then the job should be rejected and should be halted from the job queue (job cancel),
  - Otherwise, the task should be allocated/mapped respective resources as per the demand.
- Case II:
  - When a job has more than one task and the demand for resources for any single task is out of capacity then only that particular task should be rejected not the complete job, unless and until it has some dependency on the following other tasks.
  - In the case of task dependency all the dependent tasks should be rejected and finally the job should not be mapped with any resources. The case of task dependency is not included in this prototype. So, all the tasks listed in the queue, demanding resources out of capacity should be rejected.
  - Otherwise, the job should be allocated the respective resources.

As explained in the example Section 7.1.1 above there are two nodes, one node with 2 CPUs and 4 GB of memory, and another node with 1 CPU and 1 GPU with 8 GB of memory, and already clear that we have heterogeneous nodes. Moreover, as per the job request, the application demands 4 CPUs and 10 GB of memory whereas another job is requesting 2 CPUs and 8 GB of memory. So, in general view both the jobs have a single task. Where the tasks of the first and the second jobs are requesting the resources which seems out of capacity.

How should an AI scheduler handle this?

The proposed AI scheduler handles these requests and rejects the first job request whereas accepts and maps the respective resource for the second job (1 CPU and 1 GPU with 8 GB of memory). For this purpose, there is a score (an index) calculated based on available resources during prediction. So, to map the resources this index from the demand side is mapped with the predicted index and when this is under suitable conditions (demand is less than or equal to the available resources) then the respective demand is mapped with the respective resource.

We have already mathematically modeled these things in Section 7.4 with the scheduling use cases and further explained the AI-scheduler Section 7.5 and its technique in Algorithm 1 try to find the estimations so it would be helpful for the verification.

7.5 AI Scheduler: Compute Resource Mapper

The compute component of the AI Scheduler utilizes a sophisticated RNN architecture to manage and allocate computational tasks efficiently within HPC clusters. This AI-driven approach considers several factors, such as the specific characteristics of the jobs, the capabilities and current load of each node, and the overall system status. It intelligently schedules jobs using the method shown in Figure 5 and Algorithm 1 by predicting the optimal node assignments, balancing workload distribution, and minimizing job waiting times.

Algorithm 1 AI Compute Scheduler

```
 1: Input: List of Jobs (J), List of Nodes (N)
 2: Output: Job-Node Assignments
 3: for each job in J do
 4:     Determine job requirements (CPU, GPU, Memory)
 5:     for each node in N do
 6:         Evaluate node based on resource availability, workload, and performance
 7:     end for
 8:     Predict job runtime and system efficiency using RNN model
 9:     Assign job to node optimizing resource utilization and workload balance
10: end for
11: Monitor system performance and update RNN model
```

The scheduler adapts in real-time to the dynamic nature of the HPC environment, enhancing resource utilization and overall system performance.

In the provided scenario example in Section 7.1.1, the AI scheduler will first assess the resource requirements of the job, including CPU and memory needs. It then evaluates the available nodes in the HPC cluster. Using its RNN model, the scheduler predicts the runtime and efficiency of assigning the job to each node. Considering the limited resources of the first node and the specialized resources (GPU) of the second node, the scheduler may decide to split the job's tasks if possible, or it might queue the job until sufficient resources are available. The decision is based on optimizing the overall system efficiency and ensuring the job's requirements are adequately met. This example illustrates the AI scheduler's ability to make complex decisions in a dynamic and heterogeneous computing environment.

7.5.1 Data Handling and Pre-procession

Data handling and pre-processing is necessary because the historical abstract data includes information that might not be relevant and should be ETLed (Extract, Transformed, and Loaded) or cleaned into a meaningful format suitable for the scheduler. For the part of compute scheduler it requires only job characteristics, which includes job Id "Job_Id", requested CPU "CPU_req", requested memory "memory_req", amount of reservation time "time_eligible", and actual job runtime, which is calculated taking the difference of job execution start time and job execution end time. Likewise, for the system resource configuration part, it requires node characteristics that include, total CPU "CPU",

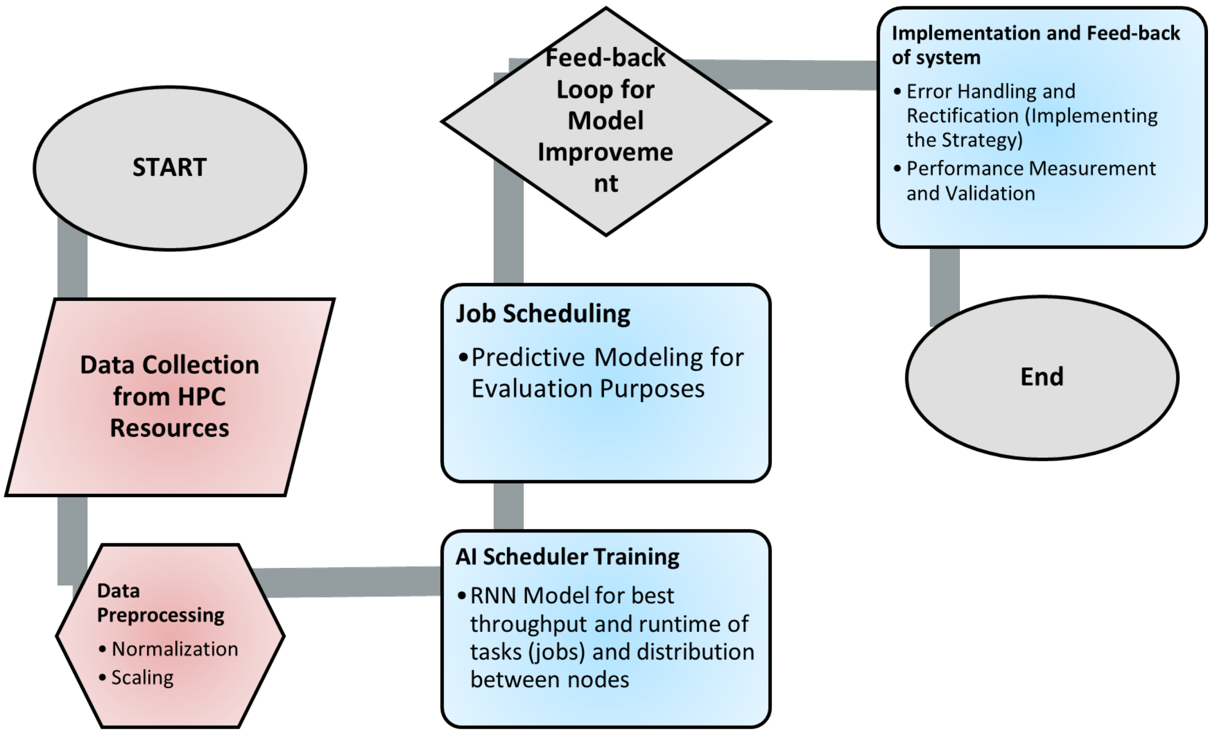
START
Data Collection from HPC Resources
Data Preprocessing
• Normalization
• Scaling
AI Scheduler Training
• RNN Model for best throughput and runtime of tasks (jobs) and distribution between nodes
Job Scheduling
• Predictive Modeling for Evaluation Purposes
Feed-back Loop for Model Improveme nt
Implementation and Feed-back of system
• Error Handling and Rectification (Implementing the Strategy)
• Performance Measurement and Validation
End

Figure 5: Flowchart for the AI scheduler algorithm using Recurrent Neural Network (RNN)

total memory "memory", occupied CPU "CPU_loaded", occupied memory "memory_loaded", and finally, a resource id column (for future development).

7.5.2 Modeling the Compute Resources

The AI Scheduler for compute resources is designed to optimize the allocation and management of computational tasks within a heterogeneous HPC system as shown in Figure 6. This system encompasses a variety of computing devices, each with distinct capabilities and resource availabilities. The scheduler's role is to efficiently map these diverse compute resources to the appropriate tasks. By leveraging AI techniques, the scheduler can predict and adapt to varying workloads and resource demands, ensuring that computational tasks are executed in the most efficient manner possible.

In addition to handling standard CPU-based tasks, the AI Scheduler is adept at managing jobs requiring specialized hardware, such as GPUs, which are commonly found in heterogeneous compute environments. This capability is particularly valuable for tasks that are compute-intensive or have specific hardware requirements. The scheduler dynamically allocates these specialized resources based on the needs of each job, thereby maximizing the utilization and throughput of the entire system. The AI scheduler's data handling framework is instrumental in achieving this level of efficiency. By pre-processing and transforming raw data into actionable insights, the scheduler can make informed decisions about resource allocation. This involves analyzing job characteristics, such as required computing power, memory needs, and expected runtime, as well as assessing the current state of system resources. The scheduler's ability to process and interpret this data is key to its success in managing a heterogeneous HPC environment.

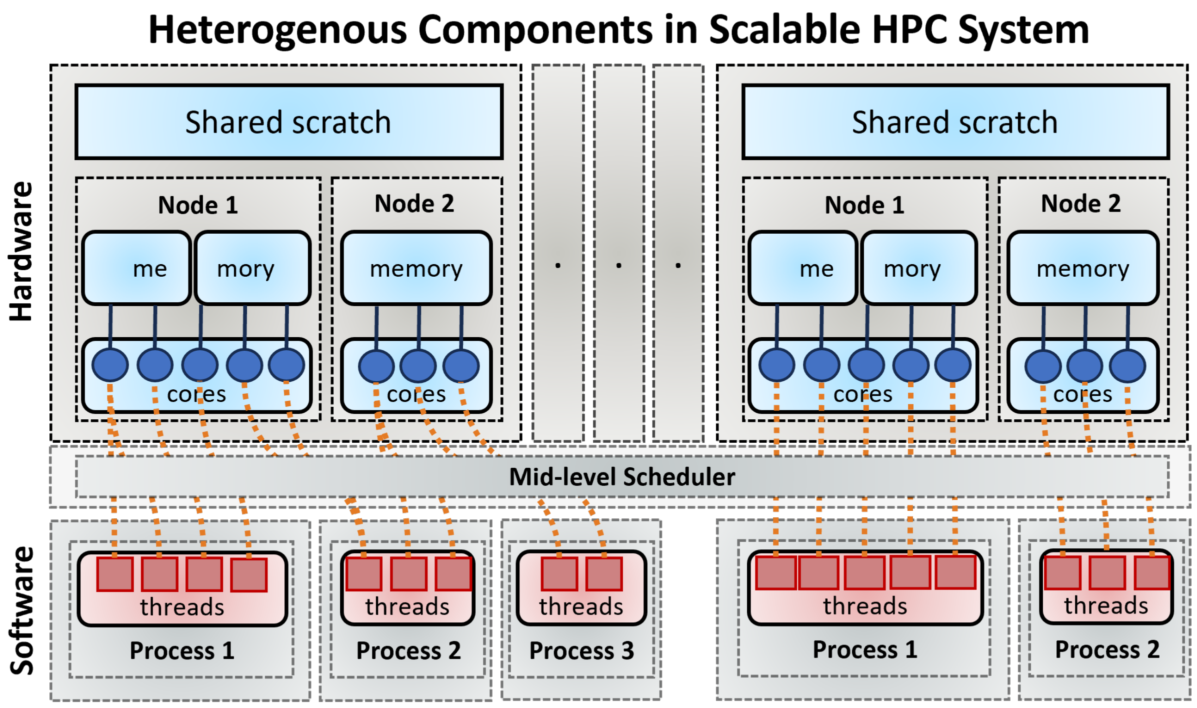

Figure 6: Heterogenous Components in Scalable HPC System

7.6 AI Scheduler: Storage Resource (I/O) Mapper

The AI scheduler for storage, implemented using fuzzy logic, focuses on optimizing data placement, migration, and replication across multiple storage solutions in the compute continuum. The scheduler evaluates the characteristics and performance of different storage systems, including their capacities, redundancy levels, and access speeds.

7.6.1 AI-Fuzzy Storage Scheduler Implementation

The AI Storage Scheduler (Algorithm 2) is implemented to dynamically adapt to changes in workload and resource availability. It processes storage data to assess available and used storage percentages and applies fuzzy logic for efficient decision-making.

Processing Storage Data The scheduler processes storage data from JSON, calculating the available and usage percentages of storage for each node. This data is critical for making informed decisions about storage management.

Fuzzy Logic Decision-Making

- The scheduler applies fuzzy logic based on the processed data to determine actions such as expanding, maintaining, or backing up storage.
- Fuzzy rules are defined based on the available and usage percentages of storage.
- Decisions are made for each node, considering the need for expansion, backup, or maintenance.

Algorithm 2 AI Storage Scheduler (Fuzzy Logic)

```
1:  Input: Data Requests (D), Storage Options (S)
2:  Output: Data Management Strategy
3:  for each data request in D do
4:      Analyze data attributes (size, access frequency, etc.)
5:      for each storage option in S do
6:          Apply fuzzy logic to determine storage suitability
7:          Define fuzzy sets and membership functions
8:          Apply fuzzy rules for decision making
9:          Example Rule: IF size is large AND frequency is high THEN prefer fast-access storage
10:     end for
11:     Use inference engine for rule evaluation
12:     Defuzzify output to concrete storage decisions
13: end for
14: Adapt to changing patterns and update rules based on performance
```

This implementation ensures that storage resources are optimally utilized, balancing performance needs against storage costs and ensuring high availability and fault tolerance.

7.6.2 Details on Fuzzy Logic Implementation

The AI Storage Scheduler's fuzzy logic decision-making process is detailed, with specific mention of the rules and membership functions used. The scheduler categorizes parameters like "data size" and "access frequency" into fuzzy sets and applies rules to decide on actions like expansion, backup, or maintenance based on the fuzzy logic output.

Fuzzy Rules and Actions

- Fuzzy rules are formulated based on the storage available and usage percentages.
- Actions such as "expand", "maintain", and "backup" are determined by these rules.
- The process includes defuzzification to translate fuzzy results into concrete actions for each storage node.

This approach allows the AI Scheduler to effectively manage diverse and uncertain data and system attributes, offering a sophisticated and adaptable resource management solution.

Applying Fuzzy Rules

- The scheduler utilizes fuzzy rules of the form: IF [condition] THEN [action]. An example rule could be, "IF data size is large AND access frequency is high THEN prioritize local storage."
- These rules are derived from expert knowledge or data analysis, tailored to reflect optimal data management decisions.

Inference Engine

- This component processes the fuzzy rules based on input data, evaluating the degree of satisfaction for each rule to inform the scheduling decision.

Defuzzification

- The process of defuzzification translates fuzzy results into concrete actions or values, such as determining a priority level for data placement.

Decision-Making Example

- Consider data with the following attributes: size = 500 GB, access frequency = 'frequent', and network bandwidth = 'moderate'. The scheduler evaluates applicable rules such as, "IF data size is large AND access frequency is frequent THEN prioritize high-speed storage." The inference engine processes these rules and, through defuzzification, concludes on a specific storage choice or priority level, optimally aligning with the given data attributes and system conditions.

The fuzzy logic-based AI scheduler effectively manages diverse and uncertain data and system attributes, offering a flexible and sophisticated approach to resource management. The subsequent sections will further elaborate on the mathematical modeling and implementation specifics of this system.

7.7 AI Scheduler: Challenges

The development and deployment of AI-driven scheduling systems, particularly in the context of distributed systems, present unique and significant challenges. These challenges necessitate the creation of a specialized project to address them effectively. Key challenges include:

7.7.1 Handling System Faults and Anomalies

- **System Disappearances**: In a distributed environment, system nodes may become intermittently unavailable or fail entirely. The AI Scheduler must be capable of detecting these disappearances and swiftly reallocating tasks to ensure continued system performance.
- **Garbage Data Management**: Erroneous or corrupt data, often referred to as 'garbage data', can significantly impede the efficiency of AI models. Developing robust methods for identifying and handling such data is crucial for maintaining the accuracy and reliability of the predictive maintenance system.

7.7.2 Dealing with Transient Network Losses

- **Resilience to Network Fluctuations**: Network instability or transient losses can disrupt the communication between distributed nodes, posing a significant challenge for real-time data processing and decision-making.
- **Adaptive Data Transfer Mechanisms**: Implementing strategies that adaptively manage data transfer and processing during periods of network instability is essential for maintaining system integrity and performance.

7.7.3 Scalability and Adaptability

- **Scalable AI Solutions**: As the system grows, the AI Scheduler must scale accordingly, both in terms of computational power and storage. Ensuring scalability while maintaining efficiency is a key challenge.
- **Adaptable Algorithms**: The algorithms employed must be flexible enough to adapt to the evolving nature of the distributed system, including changes in workload, node availability, and system topology.

7.7.4 Predictive Maintenance in Distributed Systems

- **Proactive Fault Detection**: Implementing AI-driven predictive maintenance requires the ability to not just react to system faults, but also to anticipate them based on patterns and anomalies detected in the data.
- **Customized Maintenance Strategies**: The system should be able to devise and execute tailored maintenance strategies for different parts of the distributed system, considering their unique operational characteristics and requirements.

These challenges underscore the need for an advanced AI Scheduler project tailored to distributed systems. Such a project should focus on developing innovative solutions that enhance predictive maintenance capabilities, ensuring the resilience, reliability, and efficiency of the overall system. The resulting system should be capable of intelligently managing resources, predicting and mitigating faults, and adapting to the dynamic nature of distributed computing environments.

8 References

[AOE07] II Arikpo, FU Ogban, and IE Eteng. "Von neumann architecture and modern computers". In: *Global Journal of Mathematical Sciences* 6.2 (2007), pp. 97–103.

[Car+20] Jesus Carretero et al. "Mapping and Scheduling HPC Applications for Optimizing I/O". In: *ICS2020 - 34th ACM International Conference on Supercomputing*. Barcelona, Spain, June 2020. (Visited on 12/28/2022).

[DEC] DECICE. *Device Edge Cloud Intelligent Collaboration framEwork — github.com/DECICE-project*. `https://github.com/DECICE-project`. [Accessed 17-11-2023].

[DMO21] Marc Duranton, Michael Malms, and Marcin Ostas. *HiPEAC's Vision for a New Cyber Era, a 'Continuum of Computing'*. `https://www.hpcwire.com/2021/01/21/hipeac-2021-vision-embraces-continuum-of-computing/`. 2021.

[Fan21] Yuping Fan. *Job Scheduling in High Performance Computing*. 2021. arXiv: `2109.09269 [cs.DC]`.

[Jan+23] M. Jansen et al. "The SPEC-RG reference architecture for the compute continuum". In: *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE. May 2023, pp. 469–484.

[Li+23] Baolin Li et al. "Kairos: Building cost-efficient machine learning inference systems with heterogeneous cloud resources". In: *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing*. 2023, pp. 3–16.