Future Generation Computer Systems 166 (2025) 107622

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

SAS: Speculative Locality Aware Scheduling for I/O intensive scientific analysis in clouds

Ali Zahir [a,*], Ashiq Anjum [a], Satish Narayana Srirama [b], Rajkumar Buyya [c]

[a] School of Computing and Mathematical Sciences, University of Leicester, UK
[b] School of Computer and Information Sciences, University of Hyderabad, India
[c] Cloud Computing and Distributed Systems (CLOUDS) Lab, School of Computing and Information Systems, The University of Melbourne, Australia

ARTICLE INFO

Keywords:
Big data analysis
Speculative scheduling
Locality Aware Scheduling
Edge enhanced clouds

ABSTRACT

The execution of data intensive analysis workflows in a multi-cloud environment, such as the World Large hadron collider Computing Grid (WLCG) at CERN, requires a large amount of input data, which is stored in multiple storage elements. The turnaround time taken by an individual analysis workflow running on an edge machine is mostly affected by the data reading time. Minimizing the data reading time can improve the overall efficiency of the data analysis process. To overcome this problem, we have used Speculative Scheduling to optimize the multi-cloud analysis workflows by intelligently streaming data before a task arrives for execution at the edge machine. We propose an *Event System (ES)* which is an in-memory Serverless process responsible for proactively providing input data to the workflow processes. It prefetches the data from the storage elements to the memory of the edge machine, which executes the workflow. Using locality aware scheduling and prefetching algorithms, it performs Speculative Scheduling on the basis of the evaluation of historic execution logs using the Bayesian Inference model. The Serverless *ES* learns about the incoming jobs ahead of time and makes use of intelligent data streaming to supply data to these jobs, thus reducing the overall scheduling and data access latencies and leading to significant improvements in the overall turnaround time. We have evaluated the proposed system using a large analysis workflow from High Energy Physics (HEP) by emulating the WLCG infrastructure in a controlled environment. The results have shown that by using speculative and locality aware scheduling techniques, significant improvements (i.e. over 30%) can be achieved in the execution of data intensive workflows in the cloud environment.

1. Introduction

Today's world is becoming more instrumented and digitally enhanced, leading to the production of vast amount of data. This data needs to be rapidly processed to produce key insights and forecasts in a timely manner. The scale of the data is so big that no single data centre or computational infrastructure is adequate enough to cope with the storage and data analytics challenges. Massive computational infrastructures are spreading all around the world to store, stream, and process this data in the shortest possible time. These decentralized infrastructures cooperate and coordinate to offer ubiquitous computational facilities; however, those need to be optimized, intelligently scheduled, and proactively exploited to produce the lowest possible turnaround times.

For example, the ALICE experiment at CERN records around 50 PB of data every year in its multi-cloud storage environment [1]. The total storage requirement only for proton–proton (pp) and lead-lead (pb-pb) collisions has become 15 PB [2]. This data is used by the High Eenergy Physics (HEP) users to perform analysis tasks such as particle identification, which requires data events as input data [3]. The input data contains multiple attributes of the particles, such as velocity, energy loss, and total energy. The measurement of the particle velocity is further based on the time of flight, Cherenkov angle, and transition radiation. A job contains multiple pattern matching tasks, for example, matching the time of flight for a specific particle and the cherenkov angle of a required particle. A single task performs pattern matching with reference to the preexisting values of a particular particle. The analysis tasks follow a workflow that enables a user to perform classification, regression, and function fitting to generate multi-dimensional histograms for producing novel physics insights [4]. The HEP data analysis is the least efficient of all the workloads in ALICE because it is massively I/O intensive [2,5].

Thousands of HEP users are analysing the experimental data simultaneously by performing the analysis tasks. These analysis tasks

* Corresponding author.
E-mail addresses: sazb3@leicester.ac.uk (A. Zahir), a.anjum@leicester.ac.uk (A. Anjum), satish.srirama@uohyd.ac.in (S.N. Srirama), rbuyya@unimelb.edu.au (R. Buyya).

https://doi.org/10.1016/j.future.2024.107622
Received 9 September 2023; Received in revised form 23 October 2024; Accepted 22 November 2024
Available online 29 November 2024

Table 1
Example of a Lego workflow [6].

Wagon	State	Mem total	Mem /evt	Time ms/evt	Total time ms	I/O wait ms
Baseline	OK	250 MB	0.182 KB/evt	34.10	–	–
Wagon 1	OK	591 MB	0.158 MB/evt	1.02	3815	3051
Wagon 2	OK	892 MB	0.113 MB/evt	1.04	8209	6567
Wagon 3	Fail	–	–	–	–	–
Full train	Fail	–	–	–	–	–

require a large number of events as input data. This data, generated from the Large Hadron Collider (LHC) experiments, is stored in a multi-cloud environment. Analysis tasks can be performed individually or collectively by using an analysis workflow system [6]. such as LEGO (Lightweight Environment for Grid Operators).

The LEGO workflow system was created to improve the efficiency of the analysis process in the ALICE experiment [6]. The users can submit their analysis tasks either individually or by using the LEGO workflow system. The LEGO workflow consists of different wagons containing various analysis jobs. It collects multiple analysis tasks that require the same dataset and runs them as one analysis job. During this process, the data is read once and then it is used for multiple analysis tasks in the Lego workflow. The workflow is managed by the workflow operators, who define the dataset and configure the jobs within the workflow. Table 1 shows an example of the Lego workflow run, containing measurements such as status of the test, memory consumption in total and per event, and the time used by each event.

The execution of a workflow, on an edge machine that runs from start to end is known as turnaround time and this is the most crucial parameter in terms of efficiency. The turnaround time is a combination of various time parameters, such as queue time, reading time and execution time. 70%–80% of this time consists of the I/O time based on waiting and data reading time. Table 1 shows that wagon 1 of the analysis workflow consumes 591 MB of total memory, has an event size of 0.158 MB/evt, consisting of 3740 events in total for the analysis job. The time taken by each event to execute is 1.02 ms and the total turnaround time for the job is 3815 ms. The waiting time and the reading time for this particular job are 3051 ms and 2136 ms respectively. Reading time comprises more than 70% of the total turnaround time. Reading a large number of events and executing thousands of jobs ultimately leads to a massive increase in turnaround time.

Data reading is the most time-consuming process in the whole scheduling process in a multi-cloud environment because of the number of events that are present in different storage places [7]. The analysis process, either performed by an individual user or by using LEGO workflow for an organized analysis, needs an efficient and intelligent data reading Serverless system to reduce data access times. The aim of this research is to design an efficient and optimized Serverless data reading system which reduces I/O activity and increases scheduling efficiency. We propose a speculative *Serverless Event System (ES)* which improves the efficiency of the data reading process by increasing the overall throughput of HEP data analysis process. *ES* is a novel model which has an efficient data scheduling and prefetching algorithm based on the locality of input data. It also performs task speculation and prefetches the input data for these tasks. A Serverless monitoring system on the edge machines records tasks execution logs based on their execution history. These logs are intelligently evaluated to speculate the upcoming tasks, on the basis of their arrival patterns, and the insights are used to stream data in advance, leading to reduced data access time for the upcoming tasks.

To optimize the overall efficiency of the scheduling process in Serverless edge computing environment, the system needs to know the capacity as well as the capability of the edge machine that executes the tasks. The system also needs to know how many jobs may be coming and what their requirements are. Then matchmaking is performed to allocate the best resources that can efficiently execute these jobs. Execution of these jobs becomes complex and time consuming due to the large input data sets. Multiple users generate multiple analysis jobs, which require multiple input events, resulting in increased data access latencies and scheduling delays. Studies [8,9] show that a single analysis job has to wait for 5 min on average before it starts execution.

This paper addresses the problem of a large turnaround time in the data analysis process using a Serverless Event System. The data analysis process requires a large amount of input data, which is stored in the storage elements. Reading the large amount of input data requires more time, which increases the total turnaround time of the analysis process.

The main contributions of this paper can be summarized as follows:

- **Serverless Event System (ES):** We introduce a novel ES that leverages serverless computing to dynamically manage resources and optimize the execution of data-intensive workflows.
- **Locality Aware Scheduling (LAS):** We develop a locality aware scheduling algorithm that prioritizes tasks with similar data requirements, reducing data load and unload cycles and improving memory usage efficiency.
- **Speculative Aware Scheduling (SAS):** Using historical execution logs and Bayesian inference, our ES predicts the arrival and type of future tasks, prefetching necessary data in advance to minimize data access latencies.
- **Performance Improvement:** Our proposed system demonstrates significant improvements in reducing the turnaround time for data-intensive workflows, with an observed reduction of up to 37%.
- **General Applicability:** While evaluated on ALICE workloads, our framework is adaptable to various data-intensive and computationally demanding applications, such as machine learning, big data analytics, and real-time data processing.

The rest of the paper is structured as follows: Section 2 surveys state-of-the-art scheduling and prefetching algorithms relevant to this work and identifies performance gaps in existing research. Section 3 defines the proposed architecture along with the design of Locality Aware Scheduling (LAS) and Prefetching. It also explains the concept of Speculation Aware Scheduling (SAS) on the basis of the arrival pattern of the tasks.

Section 4 presents the performance evaluation and the details about the experimental system. Finally, Section 5 concludes the paper along with the future directions.

2. Related work

An Analysis process, either performed by an individual user or by using a LEGO workflow in any multi-cloud environment [28], has to access a large amount of data. During task execution, a significant amount of time is taken by the data reading process due to the limited size of the host memory. Due to limited memory, a task has to wait for input data during execution. This waiting time can be reduced by using an efficient and intelligent data reading process. The data reading process consists of data scheduling and prefetching algorithms.

Anjum et al. [12,29] describe a Data Intensive and Network Aware Scheduling Algorithm (DIANA) and federated systems, which specifically addresses the problem of bulk jobs in an analysis environment. It mainly caters the network performance and the computation capability in the global job scheduling process. DIANA achieves efficient performance by considering the network latencies. ES on the other hand performs the scheduling on the basis of the locality of the input data which is required by the upcoming task in a queue. Ibrahim

Table 2
Logical division of scheduling and prefetching algorithms.

Scheduling technique	Strengths	Limitations	Category	Remarks
CTC [10]	Proposes an innovative scheduling algorithm considering both execution time and cost.	Complexity may pose implementation challenges.	Cost optimization	Limited scalability; complexity may hinder adaptability.
CMWSL [10]	Novel approach leveraging clustering for task scheduling.	Effectiveness may vary depending on workloads.	Load balancing	Adaptability to varying workloads suggests scalability.
CE-FT [11]	Fault-tolerant scheduling strategy for real-time tasks.	Balancing cost-effectiveness with fault tolerance can be challenging.	Cost optimization	Scalability, but may incur higher costs.
DIANA [12]	Optimizing scheduling for data-intensive and network-aware applications.	Requires significant network insights.	Network and data intensive	Limited scalability due to network requirements.
MAS [13]	Emphasizes memory considerations in scheduling.	Focus on memory may overlook comprehensive resource management.	Memory optimization	Scalability with adaptable memory management techniques.
RAS [14]	Significant improvement in execution throughput and resource utilization.	Does not address compatibility issues.	Attribute-based scheduling	Attribute selection suggests scalability potential.
Priority in MS [15]	Optimizing memory usage in scheduling.	Specific focus on memory.	Memory optimization	Memory optimization critical for scalability in HPC environments.
FCFS, SJF [16,17]	Better performance in small clusters, higher throughput.	No pre-emption, Longer processes have to wait more.	Basic scheduling	Higher throughput for simple scheduling scenarios.
DS [18,19]	High for small clusters, based on data locality.	Higher probability of job starvation.	Data locality	Improved data access performance.
CTC [20]	Instance-intensive cost-constrained.	Limited to cloud computing platform.	Cost optimization	Effective for cloud resource management.
RAFLOW [21,22]	Chain Reaction across multiple layers, short-long sequential reads.	High Latency because of retransmissions, Less I/O acceleration.	Prefetching	Efficient for sequential read operations.
AMP, SARC [23,24]	High throughput for multiple data streams, dynamic cache management.	Limited to sequential workload, naive, limited to sequential prefetching.	Cache management	Improved throughput for multiple data streams.
STEP [25]	Avoid disc thrashing.	Limited scope for input stream.	Prefetching	Reduces disc thrashing for sequential workloads.
FAST [22]	Overlap CPU time with I/O.	Limited to SSDs.	Prefetching	Efficient start-up for applications.
LYNX [26]	Prediction of future workload.	Limited size of meta data. Limited to SSDs.	Predictive scheduling	Efficient for SSD-based storage systems.
VICTREAM [27]	Extended locality aware scheduling.	Limited to GPU, naive.	Locality-aware scheduling	Effective for GPU workloads.
SAS (This paper)	Efficient data reading process, Speculation by using locality aware scheduling and prefetching.	Lacks machine learning techniques for increased efficiency.	Speculative scheduling	Improved data reading through speculation and prefetching.

et al. [30] discusses various task scheduling algorithms in cloud computing environment such as ERAS [31] which ensures reliability and performance during resource allocation. Zhao et al. [14] proposed a Resource Attribute Selection (RAS) scheduling algorithm by keeping in mind the attributes of the available resources. It schedules a task on a node on the basis of the compatibility and the fitness functions such as computing capability, storage space and network bandwidth. In comparison to this, ES does caters the compatibility of the tasks and the available resources but on the basis of its data locality. Kanemitsu et al. [10] proposed Clustering for Minimizing the Worst Schedule Length (CMWSL) concept of task clustering, in which every task in a task cluster is assigned to the same processor for execution to minimize the cost of scheduling. ES combines the tasks on the basis of their data requirements so that all of them can execute on the same resource holding the data.

In prefetching, the key problems are “what to prefetch” and “when to prefetch”. There are various prefetching algorithms available, for example Zhang et al. [22] discuss various data prefetching algorithms such as RA ReadAhead, Linux prefetching, SARC [24] and AMP [23]. ReadAhead also known as sequential prefetching is a technique to bridge a gap between the data present in storage and the application which requires it for processing. SARC, Sequential prefetching in Adaptive Replacement Cache, is based on two algorithms, prefetching and cache management. These prefetching algorithms are based on sequential data access and are not intelligent in terms of data prefetching. Our proposed algorithm performs task speculation and performs the prefethcing on the basis of execution logs.

Laga et al. proposed a prefetching mechanism called Lynx [26]. Lynx can pre-fetch data in both a sequential and random manner. It uses a mechanism to learn I/O patterns from applications and stores these patterns by using a machine learning technique based on Markov model. Lynx improved the Linux ReadAhead efficiency by 50% on execution time. Wu et al. [] introduced a hybrid scheduling algorithm that combines data locality and task prioritization, leveraging cloud elasticity to dynamically allocate resources and improve overall workflow efficiency

Jun et al. proposed Victream, a new framework for memory intensive processing on multiple GPUs [27]. It uses a novel scheduler which minimizes the data swapping between the GPU memory and a host memory. The idea is to extend the Locality Aware Scheduler, which uses the greedy approach to minimize the movement of input and output data. When the scheduler schedules a task, it optimizes this process by prioritizing tasks from the queue, which reuse the data already residing within the GPU and thus minimizes the amount of data swapping required. As an extension to the Locality Aware Scheduler, Victream performs prefetching of input data. This is performed asynchronously to task execution, so that prefetching can be done for upcoming tasks. The Victream is a naive algorithm as it only performs locality aware scheduling limited to GPUs. The proposed algorithm in the paper contains speculation based scheduling and prefetching

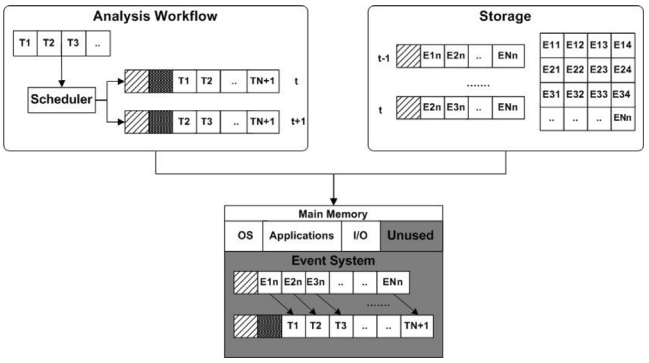

Fig. 1. Event system.

algorithm which speculates the tasks on the basis of their execution frequency and is more efficient than the existing techniques.

Table 2 explains the logical division of various existing data scheduling and prefetching algorithms. This comparative analysis has been done on the basis of their strengths and weaknesses. Few of the algorithms, such as RA, RAFLOW [21] and LYNX were implemented and embedded into the Linux kernel, while others like AMP, SARC, STEP [25], FAST [22], FLASHY [32], FLAP [33] and VICTREAM were carried out at user level applications. We have simulated the proposed algorithms and achieved the significant improvement in terms of latency in the data analysis process. The proposed system is a user level application but can be further embedded into the Linux kernel.

The prefetching algorithm in the Event system has the same objectives as RAFLOW, VICTREAM or any other prefetching algorithm; i.e. to efficiently read the data ahead of time. Its combination with locality aware scheduling to perform task speculation makes it effective and more efficient. It minimizes the amount of data swapping between the storage and memory of the node which makes the overall analysis process more efficient.

3. Proposed approach: An serverless event system

We propose a Serverless *Event System (ES)* for carrying out the workflow analysis. The ES is implemented in a serverless computing environment as an in-memory process on a host machine. The management and operations of the ES are handled by one or more CPUs of the host machine, while the ES focuses on rapidly providing input data in the form of events to tasks. The Serverless Event System includes an efficient scheduling algorithm that takes into account the locality of input data, as well as a prefetching algorithm responsible for delivering input data to analysis tasks ahead of their scheduled arrival. Moreover, the ES uses task execution logs to perform task speculation, thereby optimizing task execution based on historical data. This implementation utilizing serverless computing techniques offers advantages such as automatic resource management, scalability, and reduced server-side management, which contribute to the analysis workflow's efficiency and performance.

The Event System (ES) employs a speculative execution mechanism that uses historical execution logs and Bayesian inference to predict the arrival patterns and requirements of future tasks. This predictive approach allows the ES to prefetch data before the tasks are formally scheduled, optimizing data access times. Additionally, the scheduler utilizes this prefetched data to efficiently manage the sequence of tasks that are currently in the queue, ensuring seamless and rapid execution.

Fig. 1 shows the general working of the *ES* and its interaction with the analysis workflow, storage and main memory of the host machine. The main memory accommodates multiple processes simultaneously, such as operating system and its applications. The *ES* interacts with the main memory and reserves the unused memory slot for its operations.

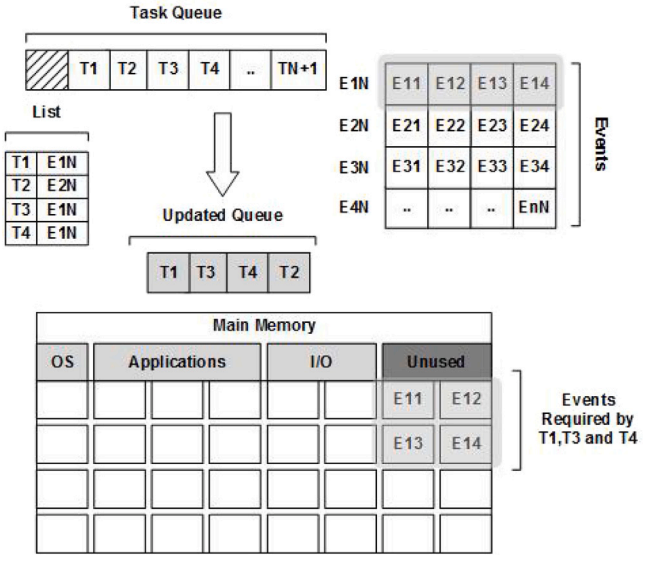

Fig. 2. Locality Aware Scheduling.

The size of the reserved memory slot is assigned dynamically and is dependent on the available memory and the requirements of the analysis task.

The analysis workflow contains multiple tasks in a task queue which are defined as T_1, T_2, $\ldots T_{N+1}$ as shown in the workflow in Fig. 1. Each task requires several events which are defined as E_{11}, $\ldots . E_{Nn}$ as an input data. For example, T_1 needs E_{11}, E_{12}, E_{13} and E_{14}. The *ES* schedules the tasks, prefetches the respective data events from the storage and subsequently makes them available for the upcoming tasks. The *ES* already has the information about the upcoming tasks, so the prefetching algorithm prefetches the required events for subsequent tasks. When the previous task is in the execution phase, the ES starts prefetching the events for upcoming tasks.

The data prefetching mechanism in the Event System (ES) is designed to operate seamlessly without requiring any modifications to the existing applications. The ES leverages historical execution logs and a Bayesian inference model to predict and prefetch the necessary input data from storage. This data is proactively loaded into the memory of edge machines before the tasks are scheduled, ensuring that data is readily available when needed.

The ES contains a task scheduler which is based on the locality of the input data, as described in Section 3.1. Section 3.2 describes the event prefetching algorithm for tasks at the local queue level. To increase the efficiency of the analysis process, Section 3.3 describes an intelligent speculative prefetching mechanism that is adopted by classifying the historical execution logs from the monitoring machine in the computing environment.

3.1. Locality Aware Scheduling (LAS)

An analysis workflow interacts with the ES to read data, and then the ES sends the query to storage to obtain the required events. The ES contains a novel scheduler which performs locality aware scheduling, based on both the task and the locality of input data. Tasks require input data from storage. The ES downloads the required data and stores it in memory until all the tasks with the same input data requirement successfully finish executing. It selects the tasks from the scheduling queue that reuse the data residing in memory, as much as possible. This technique reduces the number of data loading and unloading iterations, which ultimately reduces the input data reading time.

Fig. 2 shows the working model of the scheduling based on the locality of the task and the required input data. The task queue contains

multiple tasks, along with a list that contains the input requirements for all the tasks. When T_1 arrives for execution, ES downloads its respective input events E_{1N} and executes the task. After the completion of T_1, the scheduler checks the input requirements of the next task in the queue. If the input data requirement for T_2 differs from the previous task T_1, the scheduler checks the next task T_3 for its requirements. Having similar input requirements as T_1, T_3 gains priority over T_2 in the updated queue as shown in Fig. 3. The proposed scheduler schedules the tasks in such a manner that all the tasks in the task queue, which require similar input data events for execution, are prioritized. Algorithm 1 explains the scheduling process on the basis of the locality of the task. The ES checks the input requirements of the upcoming tasks $T_N + i$ and executes them if their input data events are present in the memory. As the task has to wait for the resources before the start of its execution, the Locality Aware Scheduling process reduces the I/O activity between the memory and the storage so that the task gets its required data in an efficient manner. The updated task queue reduces the I/O activity, which ultimately decreases the total turnaround time of the analysis task.

3.2. Locality Aware Scheduling and Prefetching (LASP)

To improve I/O performance, the ES uses a forward read or also known as prefetching mechanism. The key problems in prefetching are what to prefetch and when to prefetch. This mechanism fetches the data from storage and moves it into the memory, which is available for the task, ahead of time. That is, when the task requires the data, the data is already present in the memory. The ES contains a prefetching algorithm, as explained in Algorithm 1, which reduces the I/O time and thus improves the efficiency of the analysis process.

The algorithm begins by initializing the Particle Identification (PID) task, reading the sample data (S_V and S_A) that will be used for comparison with upcoming tasks. It then iterates over each task in the task queue to determine if the required input data (G_V and G_A) matches the given parameters. This helps in identifying tasks that can be prefetched and scheduled together. Next, the algorithm retrieves the available memory information and reserves the required memory slots for prefetching data, ensuring that sufficient memory is allocated for the upcoming tasks. For each task, the algorithm checks if the necessary data is already available in memory. If the data is present, the task is updated with the prefetching information, and PID is set for the task. The algorithm continues prefetching data for subsequent tasks if their input requirements match the available data in memory. The task queue is then updated to prioritize tasks with similar data requirements, reducing the number of data load and unload cycles. This step optimizes the scheduling process by ensuring tasks with the same input data are executed consecutively. Finally, the algorithm ensures efficient memory utilization by prefetching only the necessary data for the upcoming tasks, maintaining an optimal balance between memory usage and data access times. The overall time complexity of Algorithm 1 is O(n), where n is the number of tasks in the task queue. This complexity arises from the steps involving iterations over the task queue, including input requirements evaluation, data prefetching, task queue update, and memory optimization. Each of these steps involves a linear pass over the tasks, making the algorithm efficient for handling large task queues.

Fig. 3 shows the generic workflow of the ES execution. An analysis workflow containing multiple tasks interacts with the ES and requests for certain data events. The task queue contains multiple tasks (T_1, T_2, etc.) and their input data requirements. The first task starts execution at time t as shown in Fig. 3. It requires input data from the storage element in the form of events. The ES knows the schedule of upcoming tasks and their requirements from the task queue. From here, the ES interacts with the storage and starts prefetching data. The efficient prefetching algorithm brings the events, required by the respective task,

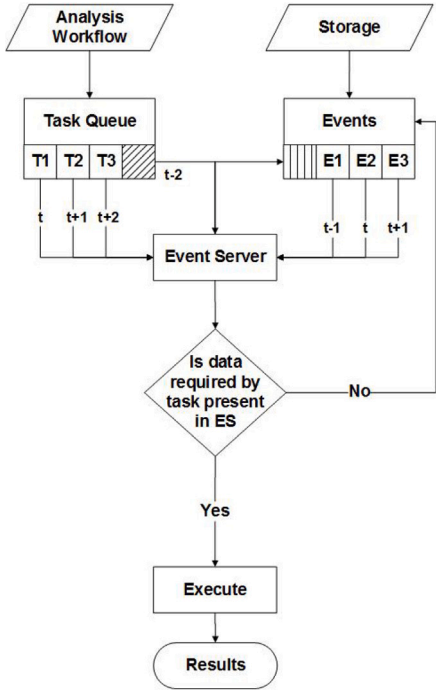

Fig. 3. Workflow execution flowchart.

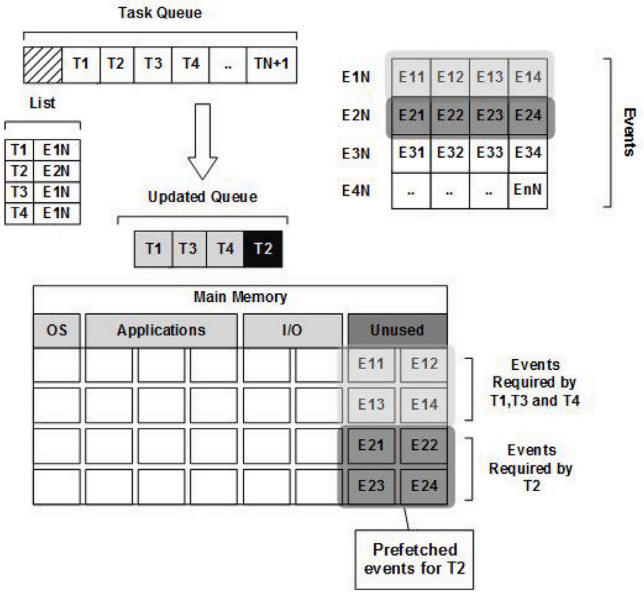

Fig. 4. Locality Aware Scheduling and Prefetching (LASP).

into memory ahead of its execution time. For example the first task executes at time t, it requires its respective input events at t−1.

As discussed previously, the locality aware scheduler schedules the tasks as per their input data requirements. All tasks with the same requirements get precedence over others. When a new task in the queue (with different input requirements) is about to start execution, the proposed prefetching algorithm starts downloading its required events in memory, parallel to the execution of previous tasks.

By implementing LASP within a serverless environment, the system can efficiently prefetch and manage data in memory, reducing the I/O wait time. The serverless model supports rapid provisioning of memory

Algorithm 1 Locality Aware Scheduling and Prefetching (LAS/LASP)

Input: Task Queue T_Q, Input Data Events E_{nN} (sample_velocity S_V, sample_angle S_A), Tasks T_N, given_angle G_A, given_velocity G_V
Output: Updated Task Queue, PID TRUE/FALSE

function ParticleIdentificationTaskinT_Q(PID)
PID = AssignRandomID
$READ(S_V, S_A)$
$E_{nN} = S_V + S_A$
for ($each\ task\ T_N\ in\ T_Q$) **do**
  if ($S_V == G_V$ && $S_A == G_A$) **then**
    PID TRUE
    else
    PID FALSE
  end if
end for
end function
READ T_Q
$mem\_free \leftarrow mem\_info$
ES reserves the memory
for ($each\ task\ T_N\ in\ T_Q$) **do**
  $mem\_free \leftarrow E_{nN}$
  if ($data\ required\ by\ T_N == evts\ in\ mem\_free$) **then**
    E_{nN} = mem_free
    PID for T_N
    i=1
    if ($data\ required\ by\ T_N + i == evts\ in\ mem\_free$) **then**
      PID for $T_N + i$
      i=i+1
      else
      discard E_{nN}
    end if
    $mem\_free \leftarrow E_{n+iN+i}$
    Check for next task: $T_N = T_N + i$
    Discard E_{n+iN+i}
  end if
  if ($mem\_free > 0$) **then**
    if ($size(E_{n+iN+i}) == mem\_free$) **then**
      $mem\_free \leftarrow E_{n+iN+i}$
      PID for $T_N + i$
    end if
  end if
end for

resources needed for prefetching data, which is vital for handling the large datasets involved in ALICE analysis. Fig. 4 shows the extension of the locality aware scheduler so that it can execute data prefetching for upcoming Particle identification (PID) tasks. To perform data prefetching as described in Algorithm 1, the schedule of the upcoming tasks in the Task Queue T_Q must be determined. Task T_2 has a different input data requirements as compared with T_1, T_3 and T_4. During the execution of T_4, the algorithm prefetches the data for T_2 in the available free memory (mem_free). This means that when $T_N + i$ arrives, its required events are already present in the memory. This technique significantly cuts down the data access time for the upcoming tasks with different input requirements and ultimately improves the efficiency of the task execution process.

3.3. Speculation Aware Scheduling (SAS)

As discussed in Section 3.2, a prefetching algorithm is based on temporal locality of the tasks. To further enhance the efficiency and effectiveness of the prefetching mechanism, we propose a distributed Speculation-Aware Scheduler (SAS) capable of optimizing the data reading process in a multi-cloud environment. In this approach, the Event System speculates tasks that are due for local execution by analysing historic execution logs. It proactively loads their respective input data into memory before they are scheduled for execution. Speculation-Aware Scheduling (SAS) involves predicting the routine and type of upcoming tasks based on their probability of occurrence by using Bayesian inferencing. The Event System anticipates tasks due for local execution by analysing historical execution logs, preemptively loading their corresponding input data into memory prior to their scheduled execution. This concept, known as Speculation-Aware Scheduling (SAS), involves predicting the course of upcoming tasks, basing this anticipation on their execution frequency and input data requirements, using Bayesian inference.

Bayesian inference enables an understanding of the probability of future events based on evidence provided by the past. Its advantage lies in its ability to continuously update the prediction model as more data becomes available, making it an ideal choice for the dynamic nature of task prediction in a computing environment.

A monitoring node in the computing environment maintains historical logs which includes system logs, event logs, and execution logs. The execution logs carry crucial parameters such as date, time, site information, execution frequency of tasks, and input data requirements.

The Bayesian inference model operates by updating our understanding of the task arrival rate based on new evidence (task arrivals) as they are observed. This model allows for more adaptability as it can accommodate shifts in task arrival patterns over time. Using historical logs as evidence, SAS predicts task arrival patterns, preemptively fetching data for upcoming tasks according to these predictions. Consequently, data analysis is expedited, and overall system performance enhanced. The Bayesian inference model deployed by SAS augments task scheduling, boosts data analysis speed, and enhances precision.

By analysing past execution logs and using Bayesian inference to interpret the evidence, SAS can discern patterns and trends in task arrivals. This enables SAS to accurately predict future task arrivals. Armed with this predictive ability, SAS can prefetch the necessary input data into memory before scheduling the tasks for execution. This proactive approach minimizes data reading time, optimizes resource utilization, and significantly improves the efficiency of the data analysis process.

We have mainly focused on six different types of HEP data analysis tasks in this study, which were previously discussed in the earlier sections. These tasks are identified and denoted as follows: Pattern Identification (PI) is represented as ‘a’, Pattern Comparison (PC) as ‘b’, String Clustering (SC) as ‘c’, String Merging (SM) as ‘d’, String Identification (SI) as ‘e’, and Pattern Matching (PM) as ‘f’. To organize these tasks, we have divided them into multiple finite sets, with each set containing all six different types of tasks.

We modelled the problem domain as a first-order Markov process. This assumption implied that the probability of an event at a given point in time “k” was solely influenced by the preceding event at time “k-1”. To denote this, we employed the notation $P(a_k)$, implying the probability of event “a” at instant “k”. In a similar manner, we represented $P(b_k), P(c_k)...P(f_k)$, where each stood for the probability of respective events occurring at instant “k”.

The main assumption of the first-order Markov was reflected in the conditional probabilities of these events. For instance, $P(a_k|a_{k-1})$ expressed the probability of event “a” happening at time “k”, conditioned on the fact that the same event “a” occurred at time “k-1”. Similarly, $P(a_k|b_{k-1})$ denoted the likelihood of event “a” at instant “k”, given that a different event “b” occurred at the preceding time step “k-1” and so on.

let $P(a_k|a_{k-1}) = [P(a_{k-1})]^{\alpha} = P(a)^{\alpha}$

The probability of “a” in “k” is based upon the probability “a” in “k-1”.

$$P(a_k|b_{k-1}) = [P(a_k|c_{k-1})].... = [P(a_k|f_{k-1})] = q_{\alpha}$$

$$P_a = P(a_k) = P(a_{k-1}) = P(a_{k+1}) \text{ for all k}$$

similarly, $P_b = P(b_k) = P(b_{k-1}) = P(b_{k+1})$ for all k

$$P(b_k|b_{k-1}) = [P(b_{k-1})]^{\alpha} = P(b)^{\alpha}$$

$$P(b_k|a_{k-1}) = [P(b_k|c_{k-1})]....[P(b_k|f_{k-1})] = q_{\beta}$$

and so on for c, d, e and f.

Now the total probability theorem is:

$$P(a_k) = P(a_k|a_{k-1})P(a_{k-1}) + P(a_k|b_{k-1})P(b_{k-1})....$$
$$...P(a_k|f_{k-1})P(f_{k-1}) = q_{\alpha}$$

So in general,

$$P_a = (P_a)^{\alpha+1} + q_{\alpha}[1 - P_a]$$

therefore,

$$q_{\alpha} = \frac{P_a(1 - (P_a)^{\alpha})}{1 - P_a}$$

Similarly, we can find the expressions for all the types b, c, d, e, and f. All the probabilities have been parameterized using a variable α which was extracted from the historical logs. If the value of α is greater than 1, then the probability of P_a is less and if the α is less than 1, the probability of P_a is higher. If the probability of the P_a is equal to 1 then all the tasks have similar probability of occurrence and there is no preference given to any tasks. The system executes the workflow by using traditional first come first serve algorithm. In case of $\alpha > 1$ for any given task, the Event System will prefetch the input requirements

The Event System processes monitoring logs and compiles the daily task queue based on the probability of occurrence of the tasks and their input data requirements. It constructs a Speculated Queue S_Q by examining the α values and probabilities. The Event System compares the S_Q which has the speculated tasks, with the local task queue. If S_Q aligns with task queue, the Event System executes Algorithm 1, engaging locality-aware scheduling and prefetching. Alternatively, if S_Q does not align with T_Q, the Event System applies the traditional First Come First Serve scheduling algorithm.

SAS in the ES involves predicting the arrival and type of future tasks using historical execution logs and Bayesian inference. The S_Q anticipates upcoming tasks based on their execution frequency, type, and input data requirements. The local task queue T_Q contains tasks that have been submitted for execution. The comparison between S_Q and T_Q is performed by matching significant task characteristics, including task type and input data requirements. When a match is found, the ES formulates a prefetch plan, identifying the data elements to be loaded into memory. The ES then proactively prefetches the required data, ensuring it is available in memory just before the tasks are scheduled for execution. This prefetch plan is dynamically adjusted in real-time based on the actual arrival of tasks, ensuring efficient data management and reduced wait times. By integrating speculative scheduling with proactive data prefetching, the ES optimizes the overall efficiency and performance of the data-intensive workflows.

Fig. 5 shows the workflow of the multi-cloud aware Event System operations. The monitoring logs are used to speculate the upcoming tasks for the Event System. The speculation is based on the arrival pattern i.e. execution frequency, probability of occurrence and the input requirements of the tasks. This technique eventually targets the data reading time which ultimately reduces the total turnaround time of the whole analysis process by prefetching data for upcoming tasks.

4. Performance evaluation

4.1. Experimental environment

The experiments were conducted in a cloud data centre located at the university, and the specifications of the data centre are outlined

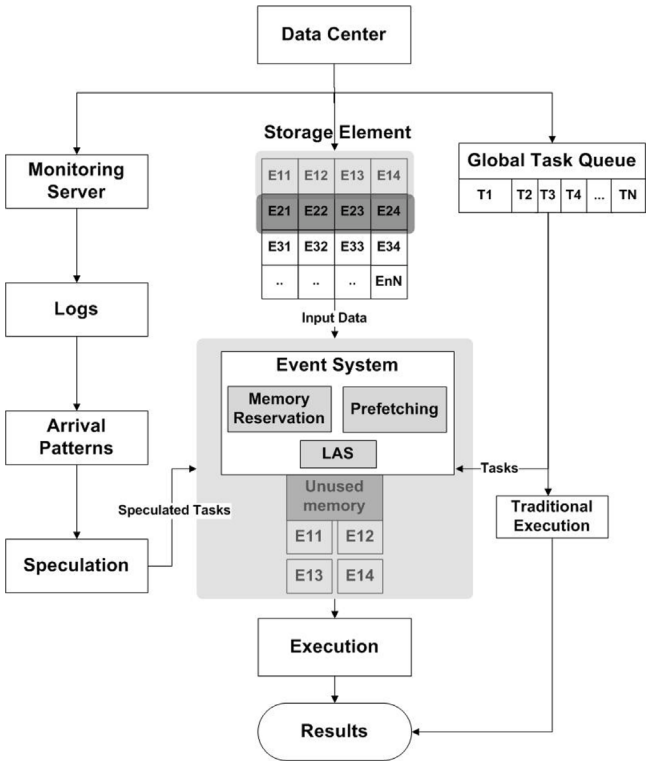

Fig. 5. Speculation Aware Scheduling (SAS).

below. Experiments were conducted in two distinct environments,as shown in Fig. 6, wherein the memory size of the edge nodes was systematically manipulated. The Serverless Event Systems is deployed on a computing infrastructure comprising two nodes, with each node being equipped with 32 cores. Both nodes are equipped with dual 8-core Intel Processors operating at a frequency of 2.2 GHz. They possess a storage capacity of 500 GB, with one node having 128 GB of RAM and the other node having 64 GB of RAM. The arrangement of the cloud resources is as follows: The cloud instance is equipped with Scientific Linux 6 and is currently operating on the OpenStack platform [34]. The OpenStack platform offers a graphical user interface known as a dashboard, which facilitates the management and administration of various resources, including storage, network, and computing resources. The cloud instance is provisioned with a total of 192 GB of random access memory (RAM), a storage capacity of 2 terabytes (TB), and a processing power of 64 cores. The proposed system was evaluated using the Virtual cloud *VC-1*, which comprises of a single compute node, one storage element, and four edge nodes. The compute node is equipped with a configuration consisting of 16 virtual central processing units (vCPUs) operating at a frequency of 2.2 GHz. It is accompanied by a random access memory (RAM) capacity of 32 GB (GB) and a storage capacity of 500 GB (GB). The storage element is configured with an 8×2.2 GHz VCPU, 64 GB of RAM, and 500 GB of storage capacity, which houses the input data set. The four edge machines are equipped with two virtual central processing units (vCPUs) running at a clock speed of 2.2 GHz. Each machine is allocated 4 GB of random access memory (RAM) and has a storage capacity of 50 GB. The primary distinction between the virtual cloud VC-2 and VC-1 lies solely in the variation of the random-access memory (RAM) capacity of the edge machines. Both the compute nodes and storage element possess identical specifications, while each of the four edge machines is equipped with 8 GB of RAM. Every node is equipped with JAVA version 8 and Eclipse Platform 4.6 Neon, which are utilized for executing the pattern matching tasks and implementing the proposed algorithms. Matlab was employed for the analysis and visualization of the obtained results (see Fig. 6).

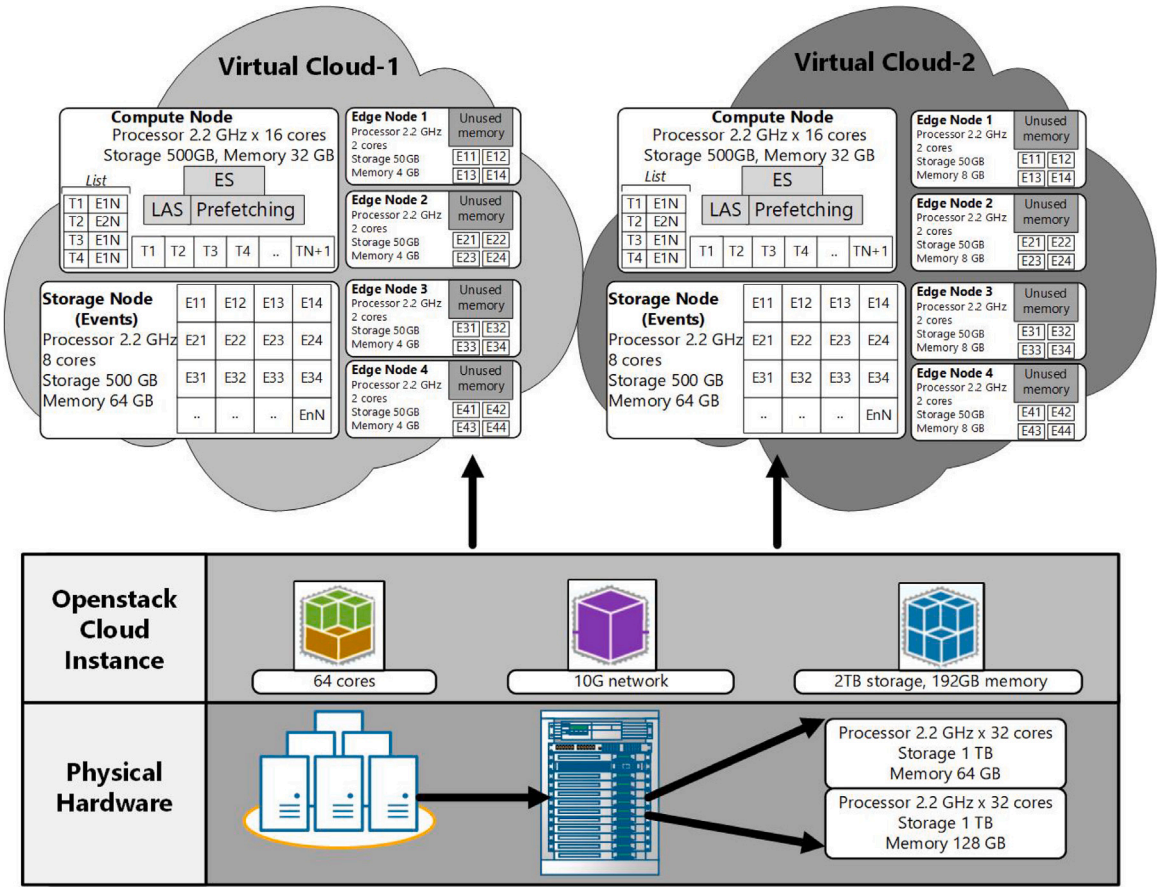

Fig. 6. Experimental environment of serverless for edge cloud configuration.

Our experiments were conducted using High Energy Physics (HEP) workflows, which are highly data-intensive and representative of large-scale scientific applications. The HEP workflows involve complex data analysis tasks such as particle identification, event reconstruction, and pattern recognition. The workflows used in our experiments are characterized as follows:

- Repetitive and Predictable Data Access Patterns: These workflows involve tasks such as particle identification and event reconstruction. These tasks frequently access the same or similar datasets for analysis, which makes data access patterns predictable.
- I/O-Intensive Operations: Each task typically involves large data volumes, where the input datasets are several gigabytes or even terabytes in size. These data-intensive operations benefit from effective prefetching mechanisms and speculative execution.
- Dependence on Distributed Data Storage: The experiments rely on distributed storage systems, where datasets are spread across multiple locations. This makes locality-aware scheduling critical for minimizing data transfer times and optimizing workflow performance.

We have designed the particle identification (PID) tasks which identify the specific patterns in the input data. There are 1000 finite sets of tasks, each containing 100 tasks of six different types in them. We have performed experiments with different sizes of input data sets i.e. 1 GB, 10 GB, and 100 GB.

A single task performs the pattern matching with reference to the preexisting values of a particular particle. The input data contains multiple attributes of the particles such as velocity of the particles, energy loss and the total energy. The measurement of the particle velocity is further based on the time of flight, cherenkov angle and transition radiation. There are multiple tasks such as matching the time of flight for a specific particle and the cherenkov angle of a required particle. The given angle or the required angle G_A in a task is compared with the data events in the sample angle S_A. If there is a match for the first letter of G_A, it checks if all the letters of G_A matches the S_A. If the given angle matches with the sample angle, the result will be TRUE and vice versa.

Multiple PID tasks arrive in the task queue T_Q at the compute node and require the respective execution resources i.e. S_A data from the storage element. The task queue maintains the input requirements of all the tasks. The Event System schedules the PID tasks to the edge machines as per their input requirements in such a manner that the tasks with the similar requirements get the execution priority. The memory holds the reusable data events S_A until all the tasks with similar requirements get executed. The mem_info process gives the information about the status of the available memory i.e. mem_free. If mem_free is greater than zero, the Event System prefetches the sample angle data for the upcoming tasks into the memory before their arrival.

The monitoring logs contains the information of the previously executed tasks. The Event System formulates the daily task queue from the monitoring logs as shown in Table 3, containing information based on the execution frequency, probability of occurrence and the input data requirements of the tasks. The daily task queue is divided into two speculated queues, S_{Q1} and S_{Q2}. S_{Q1} and S_{Q2} are based on the execution frequency and the input data requirements of the tasks respectively. The Event System knows the execution frequency of the tasks so it brings the data events in to the mem_free, before its arrival in the task queue. When the task arrives, it will not wait for the input data and can start the execution process immediately. The execution of a task leads to two outcomes: success or failure. In either of the outcomes, the execution of a task requires computing resources.

To compare the performance of the Event System, we compared the performance of our proposed LASP and SAS algorithms against two

Table 3
Experimental Results for FCFS (10 GB input data set).

No.	task_id	submit_time	start_time	end_time	exe_time	tot_time	wait_time	read_time
1	5439574	1 235 272 074	1 235 288 925	1 235 289 888	1963	17 814	15 851	9510.6
2	5439579	1 235 272 074	1 235 289 239	1 235 290 212	1973	18 138	16 165	10 022.3
3	5439580	1 235 272 074	1 235 289 888	1 235 290 851	1963	18 777	16 814	10 592.82
4	5439582	1 235 272 074	1 235 290 212	1 235 291 161	1949	19 087	17 138	11 653.84
5	5439594	1 235 272 575	1 235 290 851	1 235 291 752	1901	19 177	17 276	10 883.88
6	5439597	1 235 272 575	1 235 291 161	1 235 292 065	1904	19 490	17 586	10 727.46
7	5439574	1 235 272 074	1 235 291 752	1 235 292 719	1967	20 645	18 678	12 327.48
8	5439579	1 235 272 575	1 235 292 065	1 235 292 946	1881	20 371	18 490	12 758.1

recognized scheduling algorithms: LYNX and FCFS. The selection of these algorithms for comparison was based on their relevance to our key research aspects. LYNX, known for its effectiveness in reducing I/O wait times through data prefetching, serves as a suitable benchmark to evaluate our prefetching mechanism, demonstrating the additional benefits of integrating locality awareness and speculative scheduling. FCFS, a fundamental scheduling algorithm that processes tasks in the order they arrive, serves as a baseline to highlight the performance improvements of the proposed algorithms.

s of their arrival order i.e. whichever comes first gets the priority. The LYNX algorithm performs the read-ahead mechanism and reads the data ahead of time. We have implemented the proposed algorithms i.e. Locality Aware Scheduling, Locality Aware Scheduling and Prefetching and Speculation Aware Scheduling for evaluation and validation.

4.2. Generalization to other workflows

To demonstrate the generalizability of our approach, in addition to the ALICE analysis workflow, we applied it to a cosmological data analysis workflow. In Cosmological Analysis [35], a workflow involves predicting cosmic structures like galaxies and dark matter distributions. The process begins with a large dataset of cosmological simulations (e.g., N-body simulations), representing the evolution of the universe. The training data consists of snapshots of the universe at different time steps, including parameters like density fields, velocities, and positions of particles. The model is trained using mini-batch gradient descent, where mini-batches of data are iteratively fed into the network. For each mini batch, the model performs a forward pass to predict cosmic structures, followed by a backward pass to adjust weights based on the error between predictions and true outcomes. This iterative process continues until the model can accurately simulate cosmological phenomena. The proposed approach optimizes this workflow by prefetching the next mini batch of simulation data while the current batch is being processed, and speculative execution ensures efficient use of computational resources, avoiding idle time between data loads.

Our approach prefetches data for subsequent tasks and uses speculative execution to seamlessly process large volumes of repetitive data with minimal delays. These optimizations demonstrate the system's ability to generalize and enhance performance across diverse data-intensive workflows.

4.3. Results and analysis

Each task T_N in a Task Queue T_Q has been assigned a random ID known as a *task_id*. The following results are performed on Virtual Cloud *VC-1*, which include parameters such as the submission time, starting time, completion time, turnaround time, waiting time, execution time and the data reading time. The turnaround time is the total time taken by the analysis tasks T_N which includes the execution, waiting and the data reading time. The reading time is the time taken by input events E_{nN}, which are present in a storage, to move to the free memory i.e. mem_free (see Table 3).

Fig. 7 shows the performance evaluation of the first come first serve scheduling algorithm in terms of its execution, waiting and total time.

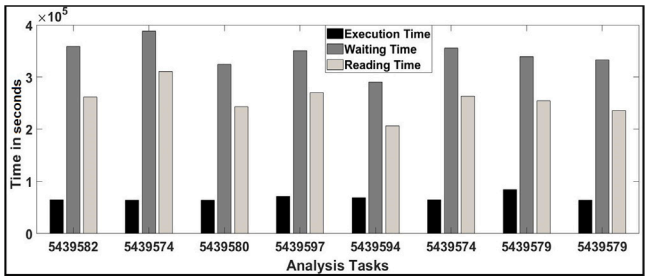

Fig. 7. First come first serve scheduler.

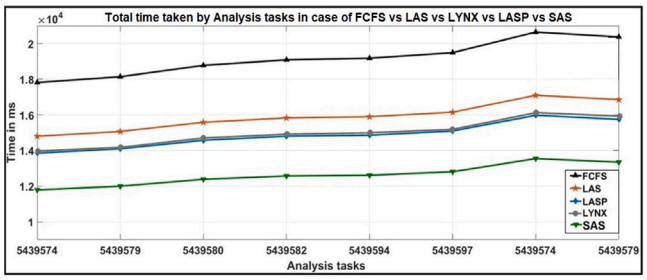

Fig. 8. Performance evaluation: Total time taken by analysis tasks in case of FCFS vs. LAS vs. LASP vs. LYNX vs. SAS.

The actual execution time of the task is far less than the total time because it has to wait for the input data and reading the data takes almost 60%–70% of the time. Table 4 shows the evaluation results and contains the performance detail of the tasks by using FCFS in the case of 10 GB input data set. The reading time for the task 5439574 is almost 55% of the total turnaround time. The proposed algorithms LAS, LASP and SAS specifically target the data reading time and reduced it up to 35% as shown in the following results.

We have compared FCFS and LYNX with the proposed algorithms and the results show significant improvement in the overall turnaround time. The proposed Locality Aware Schedular schedules the tasks in such a manner that the tasks with similar input events requirement, get the higher priority. In this scenario, memory does not load/unload the same data again and again, which reduces the data reading time. We have also implemented the scheduling along with prefetching. The system prefetches the required input events for the upcoming tasks ahead of time, in the unused part of memory. When the next task arrives for execution, its required events are already present in the memory. In Speculative Scheduling, the system predicts the upcoming tasks on the basis of their arrival patterns by using execution logs.

Fig. 8 shows the comparison of all the proposed algorithms, such as Locality Aware Scheduling, LAS along with prefetching and Speculative Scheduling with the FCFS and Lynx. As can be seen in Fig. 8, these algorithms significantly reduce the turnaround time of the analysis tasks in the workflow. In comparison to the FCFS, LAS reduced the turnaround time up to more than 17%. After performing the scheduling of the tasks on the basis of the locality of their input data, ES performs

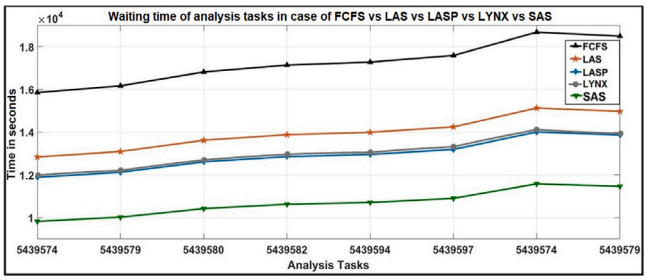

Fig. 9. Performance evaluation: Waiting time of analysis tasks in case of FCFS vs. LAS vs. LASP vs. LYNX vs. SAS.

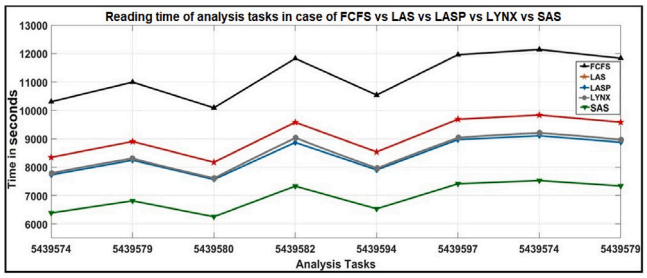

Fig. 10. Performance evaluation: Reading time of analysis tasks in case of FCFS vs. LAS vs. LASP vs. LYNX vs. SAS.

prefetching for the upcoming tasks. It forward reads the data for the upcoming tasks and the results show that this technique reduces the turnaround time up to 25% in comparison with traditional FCFS and is around 10% more effective than simple Locality Aware Scheduling. The Lynx algorithm also performs read ahead and locality based scheduling so its results are almost same as the LASP.

The Event System speculates the upcoming tasks on the basis of their historic execution logs. The speculation is based on the arrival pattern, i.e. execution frequency of the previously executed tasks and the input data requirements. The monitoring node records tasks execution logs based on their execution history. The ES makes itself aware about the upcoming tasks with the help of task execution logs and performs Locality Aware Scheduling and Prefetching. Table 5 shows a sample of 5 days monitoring log for a specific task 5439574. According to this table, ES realizes that task 5439574 follows a specific pattern, i.e. it arrives every day in the global queue for execution, at 0600 h and requires almost similar size of the input data. The Event System performs Locality Aware Scheduling and Prefetching, everyday before the arrival of 5439574. Similarly, other tasks shown in Table 4, follow a specific pattern, and the Event System performs Locality Aware Scheduling and Prefetching on the basis of their requirements.

The speculation based scheduling, significantly improves the efficiency of the analysis process and the results show that it can reduce the turnaround time up to 37%. Fig. 8 gives the overall idea about the performance of the proposed algorithms in terms of the total turnaround time.

Fig. 9 shows the comparison of the proposed algorithms in terms of the task waiting time. The waiting time is the time for which the task T_N has to wait in the task queue T_Q to get suitable resources. In the case of traditional scheduling algorithm i.e. FCFS, the task 5349574 waits for 15 851 s to get the required input data. The Locality Aware Scheduler schedules 5345974 with respect to its input data locality, which reduces its waiting time to 12 839.31 s i.e. almost 3000 s less than the FCFS. Furthermore, ES prefetches the input requirements of 5345974 before its arrival and this mechanism further reduces the waiting time to 11 888.25 s i.e. 1000 s less then the LAS. As discussed earlier, the monitoring logs contain the information of tasks such as their arrival pattern, input requirements etc. The speculative ES performs scheduling on the basis of these logs and further reduces the waiting time for 5345974 to 9827.62 which is almost 6000 s less than as compared to the FCFS and 2060 s in case of Lynx.

Fig. 10 depicts the comparison of the proposed scheduling and prefetching approaches along with the Speculation Aware Scheduling in terms of data reading time. As described before, more than 60% of the turnaround time is the waiting time which mostly consists of I/O i.e. data reading time. The proposed algorithm specifically reduces the data reading time by using the efficient data scheduling and prefetching mechanisms which ultimately improves the overall performance of the data analysis process.

Fig. 11 displays the comparative performance analysis of all the proposed algorithms in terms of total turnaround time. The stacked

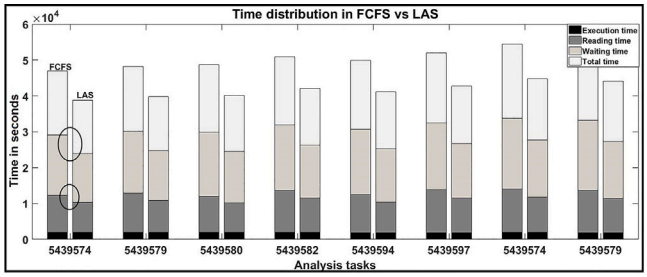

Fig. 11. Total turnaround time distribution in case of FCFS vs. LAS.

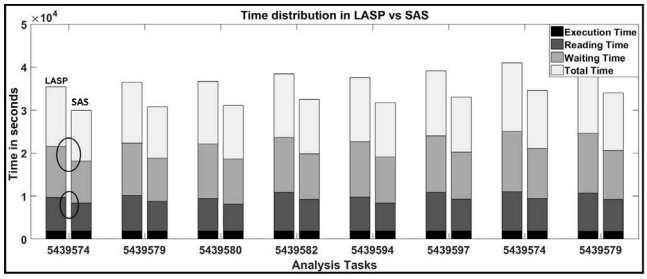

Fig. 12. Total turnaround time distribution for LASP vs. SAS.

bar graphs show the segregation of the time taken by each task when a particular algorithm is running. For example, *task 5439574* in the figure contains two stacked bar charts, FCFS and LAS respectively. The combination of execution time (in black), reading time (dark grey) and waiting time (in light grey) is the total turnaround time taken by each task in an analysis workflow. The proposed scheduling technique i.e. LAS, reduces the reading time in the bar chart of *task 5439574* which eventually reduces the total turnaround time.

Similarly, in Fig. 12, the stacked bar chart shows that the reading time is further reduced by LASP. The Prefeching algorithm, in the Event System, works together with the locality aware scheduler and reads the input data events for the task 5439574 before its arrival time. It will keep those events into the unused part of the memory. When the task 543974 arrives, it does not have to wait for its respective input events and straight away starts the execution. This reduces the waiting time, specifically the data reading time up to 1300 s in comparison with the LAS. The speculation based scheduling (SAS) is considered to be the most efficient among all the proposed algorithms as it further reduces the data reading time and improves the performance of the analysis process.

To compare the performance of all the proposed algorithms, we have designed the percentage improvement graph, shown in Fig. 13. The LAS algorithm is 19% better than the FCFS because of its efficient scheduling mechanism based on the locality of data. The combination of LAS and prefetching further improves the turnaround time of analysis process by 24% which is almost similar to Lynx. As discussed earlier,

Table 4
A sample of daily monitoring logs for a specific task 5439574.

Date	task_id	submit_time	start_time	end_time	site	TTL	f_q	IR	run_length	Status
20190221	5439574	1 233 723 420	1 233 740 004	1 233 739 041	0	24	0602	788	00:02:95	Success
20190222	5439574	1 233 722 967	1 233 739 551	1 233 738 588	0	24	0601	873	00:03:02	Success
20190223	5439574	1 233 723 872	1 233 740 456	1 233 739 493	0	24	0557	895	00:03:11	Success
20190224	5439574	1 233 722 999	1 233 739 583	1 233 738 620	0	24	0601	923	00:03:18	Success
20190225	5439574	1 233 723 476	1 233 740 060	1 233 739 097	0	24	0558	947	00:03:24	Success

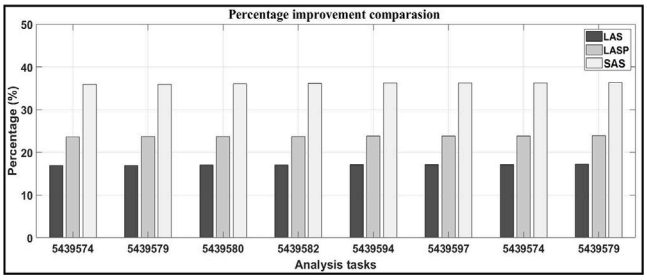

Fig. 13. Percentage improvement comparison among LAS, LASP and SAS.

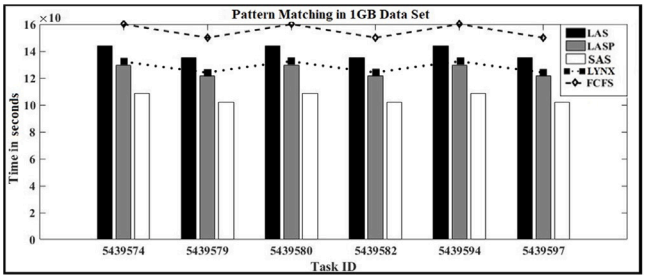

Fig. 14. Time distribution in case of 1 GB input data set.

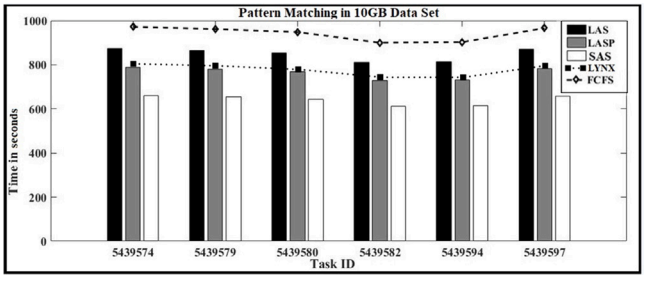

Fig. 15. Time distribution in case of 10 GB input data set.

SAS is declared as the most efficient among all the other proposed algorithms as it predicts the tasks as per their execution pattern and prefetches the required input data before the arrival of the task. By using the Speculation Aware Scheduling, we have achieved a significant performance improvement for the data analysis process by reducing the turnaround time up to 37%.

As discussed earlier, we have performed experiments by considering different sizes of input data sets, i.e. 1 GB, 10 GB and 100 GB. We have submitted the same job, containing eight tasks, with different input data requirements in terms of size i.e. 1 GB. Fig. 14 shows that when the input data size is 1 GB, the task 5439574 takes around 161.241 s with the existing scheduling technique. The proposed Locality Aware Scheduling improves its efficiency and reduces the turnaround time to around 144.734 s. By introducing the prefetching mechanism, it reduces to little less than 129.522 s and the Speculation Aware Scheduling further brings it down to around 105.401 s.

Fig. 15 shows that when the input data size becomes 10 GB, the same task 5439574 takes around 968.241 s with the existing scheduling technique. The Locality Aware Scheduling reduces the turnaround time to around 875.984 s and the prefetching mechanism brings it down to less than 734.003 s. The Speculation Aware Scheduling is the most efficient algorithm in this case as well, which further reduces the turnaround time to around 653.012 s.

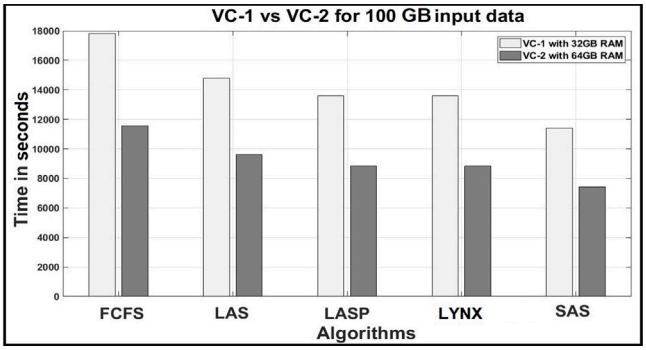

Fig. 16. Impact of memory in case of 100 GB input data.

Table 5
Turnaround time comparison of the proposed algorithms on two different experimental environments.

Mem MB	Data MB	FCFS ms	LAS ms	LASP ms	LYNX ms	SAS ms
VC-1	1	161.24	144.73	129.52	131	105.40
4 × 4	10	986.24	875.98	734	734.87	653.01
	100	17 814	14 802	13 601	13 606	11 410
VC-2	1	104.80	94.07	84.18	85.18	68.51
8 × 4	10	641.05	569.38	477.10	478.10	424.45
	100	11 579.1	9621.50	8840.81	8841.76	7416.90

We have simulated the analysis process on two different computing environments, *VC-1* and *VC-2*, which differ from each other on the basis of memory size of the edge machines. The four edge machines in *VC-1* and *VC-2* have 4 GB and 8 GB of RAM, respectively. Table 5 shows the turnaround time of the task *543974* running in two different experimental environments, *VC-1* and *VC-2*. It also shows the effect of turnaround time with different size of input data, i.e., 1 GB, 10 GB and 100 GB. We have implemented all the proposed algorithms, and the two current algorithms, i.e. FCFS and LYNX, in the *VC-2* environment and achieve more than 30% efficiency as compare to *VC-1*. Fig. 16 shows the turnaround time comparison between the two experimental environments, *VC-1* and *VC-2* in case of 100 GB of input data size. There is a significant decrease in turnaround time when the similar workload has been simulated on the edge machines with increased memory.

Fig. 17 compares the total execution time for a cosmological workflow under two scheduling algorithms: FCFS and SAS. The proposed SAS was chosen for comparison as it has demonstrated better performance in terms of execution efficiency compared to other scheduling algorithms. The bars represent the time breakdown across three components: execution, reading, and wait time. We have compared the workflow by varying its input data size to 32, 64 and 128 MBs. The results show that SAS significantly reduces reading time by 24% compared to FCFS, leading to an overall improvement in total time, especially as the batch size increases. This highlights the efficiency

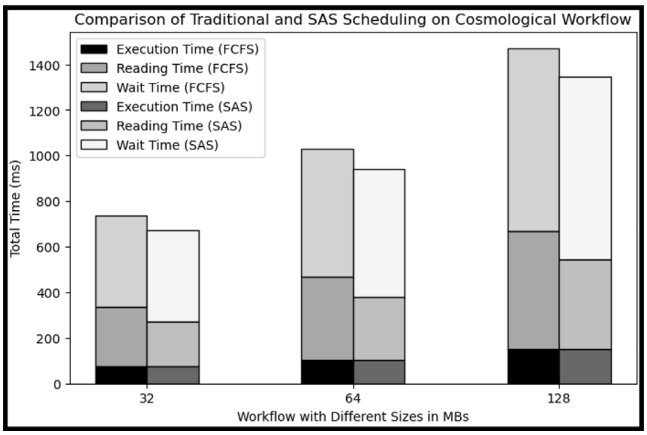

Fig. 17. Turnaround time comparison of cosmological workflow with speculative aware scheduling algorithm.

of the proposed framework in minimizing I/O delays and optimizing performance for large-scale workflows.

The above results show that the ES based solution for data reading process is more efficient than the traditional techniques. The speculative data prefetching, on the basis of task arrival pattern i.e. SAS, significantly reduces the data reading time and proved to be the most optimized algorithm for data analysis. It reduces the data reading latencies i.e. turnaround time, from storage to memory, and improves the overall efficiency of the analysis process.

5. Conclusions and future work

In this paper, we proposed a Serverless Event System based solution for optimizing the multi-cloud analysis workflows using Speculative Scheduling on the edge machines. Our experiments utilized High Energy Physics (HEP) workflows, which are highly data-intensive and representative of scientific applications requiring significant computational resources. The predictable access patterns of HEP workflows allowed us to effectively demonstrate the benefits of our proposed techniques. By improving the performance of HEP workflows, our methods have the potential to enhance data processing efficiency in various scientific domains. The Serverless Event System implements data prefetching and locality aware scheduling algorithms. It performs task arrival speculation and intelligently schedules workflows, on the basis of historic logs for improving the data reading time of analysis workflows in edge computing enviorment. This approach exploits data prefetching and locality aware scheduling techniques to reduce the data reading latencies, which have a major impact on the turnaround time of an analysis workflow running on the edge machines. To improve the accuracy of the speculation process, the proposed Serverless Event System predicts the probability of the upcoming tasks by using Bayesian Inference model.

This information is then used to improve scheduling and execution delays for the incoming jobs by reading data ahead of time and allocating the best resources to execute the analysis workflows. The results have shown that by using locality aware scheduling and prefetching techniques, significant improvements (i.e. over 25%) can be achieved, in terms of reducing the turnaround time. Furthermore, the speculative scheduling based on historic logs and probability of occurrence using Bayesian inference further reduced the overall turnaround time up to 37% by executing an HEP analysis workflows in a serverless edge computing environment. The experimental results demonstrate that our proposed framework is not limited to ALICE analysis workflows and can be generalized to other data-intensive workflows, such as cosmological data analysis process. By applying our approach, we observed significant improvements in turnaround time and resource utilization. These results confirm that the prefetching and speculative execution mechanisms are effective in optimizing workflows with repetitive data access patterns and high I/O demands, making the system highly adaptable to various domains beyond high-energy physics.

Currently, we have evaluated the proposed algorithms by performing speculation using the arrival patterns of tasks. In future, we intend to perform intelligent task speculation by using machine learning algorithms which will intelligently predict the upcoming types of tasks by using reinforcement learning algorithm and start streaming data ahead of time so that all data and execution resources are available before the workflow is scheduled for execution. We will also implement a decentralized Serverless communication mechanism [36] within the scheduling engine where a pilot job will trigger the data reading process in the Serverless Event System and will proactively and intelligently schedule the jobs when the Event System has the data to run these jobs.

CRediT authorship contribution statement

Ali Zahir: led the development of the Serverless Event System (ES), including LAS and SAS algorithms, and contributed extensively to research, implementation, data analysis, and manuscript preparation, . **Ashiq Anjum:** guided the ES design, SAS algorithm development, and manuscript refinement.. **Satish Narayana Srirama:** offered expertise in serverless and cloud-edge computing and guided the research and paper refinement.. **Rajkumar Buyya:** supervised the research, shaping objectives and methodologies for cloud efficiency..

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] Z. Zhang, A. Kulkarni, X. Ma, Y. Zhou, Memory resource allocation for file system prefetching: from a supply chain management perspective, in: Proceedings of the 4th ACM European Conference on Computer Systems, ACM, 2009, pp. 75–88.
[2] P. Buncic, M. Krzewicki, P. Vande Vyvre, Technical Design Report for the Upgrade of the Online-Offline Computing System, Tech. Rep., 2015.
[3] A. Di Mauro, Particle identification methods other than RICH, Nucl. Instrum. Methods Phys. Res. A (2019).
[4] P.C. Bhat, Advanced analysis methods in particle physics, Ann. Rev. Nucl. Part. Sci. 61 (FERMILAB-PUB-10-054-E) (2010).
[5] F. Van Lingen, M. Thomas, T. Azim, I. Chitnis, A. Anjum, D. Bourilkov, M. Kulkarni, C. Steenberg, R. Cavanaugh, J. Bunn, et al., Grid enabled analysis: architecture, prototype and status, 2005.
[6] M. Zimmermann, A. collaboration, et al., The ALICE analysis train system, in: Journal of Physics: Conference Series, vol. 608, IOP Publishing, 2015, 012019, 1.
[7] S. Sotiriadis, N. Bessis, A. Anjum, R. Buyya, An inter-cloud meta-scheduling (icms) simulation framework: Architecture and evaluation, IEEE Trans. Serv. Comput. 11 (1) (2015) 5–19.
[8] A. Ali, A. Anjum, J. Bunn, R. Cavanaugh, F. van Lingen, R. McClatchey, H. Newman, C. Steenberg, M. Thomas, I. Willers, et al., Job monitoring in an interactive grid analysis environment, in: In Computing in High Energy Physics. Interlaken, Switzerland, 2004.
[9] D. Lingrand, T. Glatard, J. Montagnat, Modeling the latency on production grids with respect to the execution context, Parallel Comput. 35 (10–11) (2009) 493–511.
[10] H. Kanemitsu, M. Hanada, H. Nakazato, Clustering-based task scheduling in a large number of heterogeneous processors, IEEE Trans. Parallel Distrib. Syst. 27 (11) (2016) 3144–3157.
[11] P. Guo, Z. Xue, Cost-effective fault-tolerant scheduling algorithm for real-time tasks in cloud systems, in: The Proceedings of 2017 IEEE 17th International Conference on Communication Technology, ICCT, IEEE, 2017, pp. 1942–1946.

[12] A. Anjum, R. McClatchey, A. Ali, I. Willers, Bulk scheduling with the DIANA scheduler, IEEE Trans. Nucl. Sci. 53 (6) (2006) 3818–3829.
[13] I. Dooley, C. Mei, J. Lifflander, L.V. Kale, A study of memory-aware scheduling in message driven parallel programs, in: 2010 International Conference on High Performance Computing, IEEE, 2010, pp. 1–10.
[14] Y. Zhao, L. Chen, Y. Li, W. Tian, Efficient task scheduling for many task computing with resource attribute selection, China Commun. 11 (12) (2014) 125–140.
[15] S.P. Muralidhara, L. Subramanian, O. Mutlu, M. Kandemir, T. Moscibroda, Reducing memory interference in multicore systems via application-aware memory channel partitioning, in: Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture, 2011, pp. 374–385.
[16] J.S. Manjaly, E.A. Chinnu, Relative study on task schedulers in hadoop mapreduce, J. Adv. Res. Comput. Sci. Softw. Eng. 3 (5) (2013).
[17] H.M. Patel, A comparative analysis of MapReduce scheduling algorithms for Hadoop, Int. J. Innov. Emerg. Res. Eng. 2 (2015).
[18] M. Zaharia, D. Borthakur, J.S. Sarma, K. Elmeleegy, S. Shenker, I. Stoica, Job Scheduling for Multi-User Mapreduce Clusters, Tech. Rep., Technical Report UCB/EECS-2009-55, EECS Department, University of California, Berkeley, 2009.
[19] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, I. Stoica, Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling, in: Proceedings of the 5th European Conference on Computer Systems, ACM, 2010, pp. 265–278.
[20] K. Liu, H. Jin, J. Chen, X. Liu, D. Yuan, Y. Yang, A compromised-time-cost scheduling algorithm in swindew-c for instance-intensive cost-constrained workflows on a cloud computing platform, Int. J. High Perform. Comput. Appl. 24 (4) (2010) 445–456.
[21] K. Matsuzawa, T. Nakamura, K. Sonoda, File readahead method with the use of access pattern information attached to metadata, 2006, US Patent 20080040538A1.
[22] Y. Joo, J. Ryu, S. Park, K.G. Shin, FAST: Quick application launch on solid-state drives, in: FAST, 2011, pp. 259–272.
[23] B.S. Gill, L.A.D. Bathen, AMP: Adaptive multi-stream prefetching in a shared cache, in: FAST, vol. 7, 2007, pp. 185–198, 5.
[24] B.S. Gill, D.S. Modha, SARC: Sequential prefetching in adaptive replacement cache, in: USENIX Annual Technical Conference, General Track, 2005, pp. 293–308.
[25] S. Liang, S. Jiang, X. Zhang, STEP: Sequentiality and thrashing detection based prefetching to improve performance of networked storage servers, in: 27th International Conference on Distributed Computing Systems, 2007, ICDCS'07, IEEE, 2007, 64–64.
[26] A. Laga, J. Boukhobza, M. Koskas, F. Singhoff, Lynx: a learning linux prefetching mechanism for SSD performance model, in: Non-Volatile Memory Systems and Applications Symposium (NVMSA), 2016 5th, IEEE, 2016, pp. 1–6.
[27] J. Suzuki, Y. Hayashi, M. Kan, S. Miyakawa, T. Takenaka, T. Araki, M. Kitsuregawa, Victream: Computing framework for out-of-core processing on multiple GPUs, in: Proceedings of the Fourth IEEE/ACM International Conference on Big Data Computing, Applications and Technologies, ACM, 2017, pp. 179–188.
[28] F. van Lingen, C. Steenberg, M. Thomas, A. Anjum, T. Azim, F. Khan, H. Newman, A. Ali, J. Bunn, I. Legrand, The clarens web service framework for distributed scientific analysis in grid projects, in: Parallel Processing, 2005. ICPP 2005 Workshops. International Conference Workshops on, IEEE, 2005, pp. 45–52.
[29] S.L. Kiani, A. Anjum, M. Knappmeyer, N. Bessis, N. Antonopoulos, Federated broker system for pervasive context provisioning, J. Syst. Softw. 86 (4) (2013) 1107–1123.
[30] I.M. Ibrahim, et al., Task scheduling algorithms in cloud computing: A review, Turk. J. Comput. Math. Educ. (TURCOMAT) 12 (4) (2021) 1041–1053.
[31] V.A. Lepakshi, C. Prashanth, Efficient resource allocation with score for reliable task scheduling in cloud computing systems, in: 2020 2nd International Conference on Innovative Mechanisms for Industry Applications, ICIMIA, IEEE, 2020, pp. 6–12.
[32] A.J. Uppal, R.C. Chiang, H.H. Huang, Flashy prefetching for high-performance flash drives, in: IEEE 28th Symposium on Mass Storage Systems and Technologies, MSST, IEEE, 2012, pp. 1–12.
[33] Y. Liu, W. Wei, FLAP: flash-aware prefetching for improving SSD-based disk cache, J. Netw. 9 (10) (2014).
[34] OpenStack, Build the future of open infrastructure, 2020, http://openstack.org. (Accessed 30 May 2019).
[35] L. Lucie-Smith, H.V. Peiris, A. Pontzen, M. Lochner, Machine learning cosmological structure formation, Mon. Not. R. Astron. Soc. 479 (3) (2018) 3405–3414.
[36] K. Hasham, A.D. Peris, A. Anjum, D. Evans, S. Gowdy, J.M. Hernandez, E. Huedo, D. Hufnagel, F. van Lingen, R. McClatchey, et al., CMS workflow execution using intelligent job scheduling and data access strategies, IEEE Trans. Nucl. Sci. 58 (3) (2011) 1221–1232.

Ali Zahir is a researcher at the University of Leicester in the United Kingdom, specialized in the field of distributed computing. His research is centred on enhancing data processing workflows in cloud and edge computing environments. Dr. Zahir's primary objective is to optimize the handling of data-intensive scientific analyses, with a particular focus on High-Energy Physics (HEP). He has made notable contributions to the development of advanced data retrieval algorithms and systems. Notably, he has been collaborating with CERN on the ALICE project since 2009. Beyond academia, Dr. Zahir actively engages in the realms of data science and high-performance computing. His involvement in various research and development initiatives underscores his commitment to bridging the gap between theoretical advancements and practical implementations within the field of distributed computing.

Ashiq Anjum is a renowned Professor of Distributed Systems at the University of Leicester in the UK. He is also the director of enterprise and impact for the School of Computing and Mathematical Sciences at Leicester. He is also the AI lead for the £60 million METEOR project at the Space Park Leicester. He used to work at the University of Derby, UK, as Professor of Distributed Systems and Director of the Data Science Research Centre. His research includes data-intensive distributed systems, distributed machine learning, self-learning digital twins, and high-performance analytics for streaming data. Professor Anjum has gotten research grants from well-known groups and worked on projects in many different fields, from medical research to telecommunications. In partnership with industry giants like Rolls Royce and BT, he was the first to create self-learning digital twins for cyberphysical systems. He has also worked with Space Park Leicester to look into how digital twins could be used in environmental applications. Professor Anjum works on high-performance analytics systems with healthcare providers and drug companies. In the field of distributed data analytics, he has worked with CERN Geneva for more than 15 years and closely with leaders in cloud computing, telecommunications, space, and other fields. Professor Anjum's research includes distributed systems, machine learning, artificial intelligence, and trustworthy distributed systems, with a focus on blockchains. He has a long list of publications and an H-index of 39.

Satish Narayana Srirama works as an Associate Professor in the School of Computer and Information Sciences at the University of Hyderabad. He was a Research Professor at Tartu University's Institute of Computer Science until June 2020. He was also a Visiting Professor and honorary head of the Mobile & Cloud Lab. In 2008, RWTH Aachen gave him a Ph.D. in computer science. His research interests include cloud computing, mobile web services, mobile cloud, the Internet of Things, fog computing, moving scientific and business applications to the cloud, and large-scale cloud data analytics. He is a Senior Member of the IEEE and is the Associate Editor of IEEE Transactions in Cloud Computing and a member of the programme committees for several international conferences and workshops. His journal, Wiley Software: Practice and Experience, has been around for 53 years. Srirama wrote or co-wrote 170 papers for international conferences and journals.

Rajkumar Buyya is a Fellow of IEEE, Professor of Computer Science and Software Engineering and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft, a spin-off company of the University, commercializing its innovations in Cloud Computing. He has authored over 525 publications and seven text books including "Mastering Cloud Computing" published by McGraw Hill, China Machine Press, and Morgan Kaufmann for Indian, Chinese and international markets respectively. He is one of the highly cited authors in computer science and software engineering worldwide (h-index=109, g-index=230, 58,300+ citations). Microsoft Academic Search Index ranked Dr.

Buyya as #1 author in the world (2005–2016) for both field rating and citations evaluations in the area of Distributed and Parallel Computing. "A Scientometric Analysis of Cloud Computing Literature" by German scientists ranked Dr. Buyya as the World's Top-Cited (#1) Author and the World's Most Productive (#1) Author in Cloud Computing. Recently, Dr. Buyya is recognized as "2016 Web of Science Highly Cited Researcher" by Thomson Reuters. Software technologies for Grid and Cloud computing developed under Dr.Buyya's leadership have gained rapid acceptance and are in use at several academic institutions and commercial enterprises in 40 countries around the world. Manjrasoft's Aneka Cloud technology developed under his leadership has received "2010 Frost & Sullivan New Product Innovation Award". Recently, Dr. Buyya received "Mahatma Gandhi Award" along with Gold Medal for his outstanding and extraordinary achievements in Information Technology field and services rendered to promote greater friendship and India-International cooperation. He served as the founding Editor-in-Chief of the IEEE Transactions on Cloud Computing. He is currently serving as Co-Editor-in-Chief of Journal of Software: Practice and Experience, which was established over 45 years ago. For further information on Dr. Buyya, please visit his cyberhome: www.buyya.com.