Cluster Computing (2025)28:383
https://doi.org/10.1007/s10586-025-05274-4

BigOPERA: An OPportunistic and Elastic Resource Allocation for big data frameworks

Pablo V. Caderno[1] · Feras Awaysheh[3] · José C. Cabaleiro[1,2] · Tomás F. Pena[1,2]

Received: 11 December 2024 / Revised: 12 March 2025 / Accepted: 25 March 2025

Abstract
Efficient asset management is essential for optimizing the performance and scalability of modern Big Data (BD) frameworks. However, traditional resource allocation methods often suffer from static partitioning, inefficient resource utilization, and high operational costs, limiting their ability to adapt to fluctuating workloads dynamically. This paper introduces BigOPERA, an opportunistic and elastic resource allocation framework designed to enhance BD processing environments by integrating dedicated and non-dedicated computing assets. Leveraging containerization and a two-tiered scheduling mechanism, BigOPERA dynamically manages available resources to improve workload execution efficiency. Experimental results demonstrate that BigOPERA achieves up to 35% performance improvement over native Apache Spark configurations, significantly enhancing computational throughput while optimizing resource consumption. Our findings highlight the potential of BigOPERA in scalable, cost-effective, and sustainable BD processing.

Keywords BD · Apache Spark · Opportunistic scheduling · Dynamic resource provisioning · Resource allocation · Elastic computing · Green computing

1 Introduction

Big Data (BD) analytics, characterized by its volume, velocity, variety, and veracity, has propelled data analytics to the forefront of modern business strategies and scientific research [1]. It employs advanced techniques, algorithms, and technologies to extract meaningful insights from massive datasets [2]. This multidisciplinary field integrates data science, statistics, and computer science principles to enhance decision-making, optimize processes, and drive innovation across domains such as finance, healthcare, marketing, and scientific research [3, 4]. Consequently, BD plays a pivotal role in both industry and academia [5].

At its core, BD analytics relies on cluster computing to enable large-scale data processing in local and cloud environments [6]. However, traditional cluster computing solutions often statically partition resources into application-specific clusters, such as High-Performance Computing (HPC) and High-Throughput Computing (HTC). While this approach provides predictable performance for specific workloads, it also leads to infrastructure silos and inefficient resource utilization, preventing real-time reallocation based on demand fluctuations. As a result, resources remain idle during off-peak periods, while peak workloads experience bottlenecks, increasing operational costs and degrading system performance.

Moreover, resource fragmentation within enterprises further exacerbates inefficiencies. Many organizations

✉ Feras Awaysheh
feras.awaysheh@umu.se

Pablo V. Caderno
pablo.vazquez.caderno@rai.usc.es

José C. Cabaleiro
jc.cabaleiro@usc.es

Tomás F. Pena
tf.pena@usc.es

[1] Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS) & Departamento de Electrónica e Computación, Universidade de Santiago de Compostela, Santiago de Compostela, Spain

[2] Departamento de Electrónica e Computación, Universidade de Santiago de Compostela, Santiago de Compostela 15782, Spain

[3] Department of Computing Science, ADSLab, Umeå University, Umeå, Sweden

Published online: 16 June 2025

maintain multiple isolated clusters for different workloads, preventing resource sharing even when some clusters are underutilized while others are overloaded. This rigid structure increases infrastructure costs and reduces system flexibility. Addressing these challenges requires dynamic and adaptive resource management strategies to optimize utilization and enhance the efficiency of BD analytics environments.

The rise of containerization technologies, such as Docker and Kubernetes, has introduced new possibilities for resource management. However, these technologies primarily focus on container orchestration and do not provide a comprehensive solution for integrating dedicated and non-dedicated resources in a unified manner. Although Kubernetes excels at managing containerized applications in cloud environments, it does not address the challenge of dynamically scaling resources based on real-time demand or integrating opportunistic resources into the resource pool. Addressing these challenges requires a more dynamic and efficient approach to resource management, leveraging dedicated and opportunistic computing resources while minimizing environmental impact [7].

In this paper, we introduce BigOPERA: An OPportunistic and Elastic Resource Allocation for BD, a novel framework that enhances Apache Spark by integrating opportunistic computing. Traditional static resource allocation models often lead to inefficiencies, whereas BigOPERA leverages idle or underutilized resources to improve scalability, cost efficiency, and performance. Our framework dynamically integrates opportunistic resources with Spark, ensuring seamless operation with minimal modifications. Through comprehensive performance evaluation, we demonstrate its ability to handle fluctuating workloads, optimize resource utilization, and reduce operational costs. Additionally, BigOPERA contributes to sustainable computing by minimizing energy consumption and infrastructure costs.

The key contributions of this paper include:

- **A two-tiered, fine-grained scheduling mechanism.** BigOPERA introduces a dynamic resource allocation strategy that optimally manages dedicated and opportunistic computing resources in real-time.
- **Scalable and elastic containerization deployment architecture.** Leveraging Docker and Kubernetes containers, BigOPERA seamlessly integrates Apache Spark and HTCondor, reducing resource fragmentation and improving workload management.
- **Performance improvements and cost efficiency.** BigOPERA achieves up to 35% performance gains over native Spark by efficiently utilizing idle resources and ensuring optimal computational power allocation.
- **Fault tolerance and sustainable deployment environment.** The framework enhances Spark's resilience in dynamic environments, handling intermittent resource availability while promoting green computing practices through reduced energy consumption.

The paper is structured as follows: in Sect. 2, we discuss the current state of resource management in BD environments, emphasizing significant advancements and existing limitations. Section 3 describes the motivation behind developing BigOPERA, identifying key issues being addressed. Section 4 describes the design of the BigOPERA framework. Section 5 presents the experimental setup used to evaluate the framework performance, and Sect. 6 presents and discusses the results. Section 7 concludes the study and introduces future research directions to enhance and expand BigOPERA's capabilities.

2 Background and related work

Resource management in BD environments has become a complex and critical area of research, and researchers are striving to balance the demands of computational workloads with available infrastructure. Situated within the realm of scalable computing and data-intensive applications, this field has evolved through various approaches to improve resource allocation strategies and optimize computational throughput. This section concisely reviews crucial foundational and recent contributions that inform the development of our proposed BigOPERA framework.

2.1 BD challenges

BD has been a game-changer across many fields, letting us analyze massive, complex datasets that were once too much to handle. However, this wealth of information comes with its own set of problems that organizations and researchers need to tackle. These challenges include, among others, data privacy and security concerns [8], data quality and integrity, scalability and infrastructure costs, technical complexity, and the need for specialized expertise to manage and analyze large datasets effectively [9]. Implementing such systems often involves complex configurations and fine-tuning to ensure they operate efficiently and meet large-scale data operations' demands.

As the volume of data grows exponentially, organizations face significant complications with scaling their infrastructure to manage and process these vast datasets. Ensuring that the infrastructure can handle increasing amounts of data involves enhancing hardware capabilities, such as storage and processing power, and utilizing distributed computing frameworks like Apache Hadoop [10]

and Apache Spark [11]. However, this scalability comes with substantial infrastructure costs. These costs range from the initial capital investments in hardware to ongoing expenses for system maintenance, energy consumption, and cloud services. Balancing the need for scalability with budget constraints is a critical consideration for organizations leveraging BD.

BigOPERA explicitly aims to alleviate the last two key challenges by providing a seamless and straightforward solution that integrates effortlessly with existing Spark clusters. It does not require Spark architecture, applications, or cluster configuration modifications. Unlike other solutions that demand additional libraries, dependencies, or infrastructure changes [12, 13], BigOPERA functions as an add-on that works with the existing Spark ecosystem. It enables enterprises to scale their extensive data operations efficiently without costly infrastructure upgrades.

2.2 Resource management

Cluster resource management encompasses two fundamental functions: resource allocation and task scheduling. Resource allocation refers to the dynamic distribution of computational resources among users or applications, adhering to a predefined global policy that ensures equitable access. This policy typically considers fairness constraints or priority-based allocations to optimize resource utilization. Conversely, task scheduling involves mapping a scheduling unit such as a task, process, container, or virtual machine, to an appropriate node within the cluster, ensuring efficient execution with the allocated resources.

Traditional approaches in data centers often partition computational resources into static, application-specific clusters. Zaharia et al. [14] highlight that while this method may provide predictability, it leads to inefficient resource utilization due to its rigid, preallocated structure. The inherently dynamic and fluctuating nature of BD workloads often conflicts with such static partitioning, resulting in computational inefficiencies and resource underutilization.

The emergence of the MapReduce programming model [15] and its open-source implementation, Hadoop, has spurred significant research interest [16]. Among various research topics, task scheduling has gained substantial attention, leading to the development of numerous scheduling algorithms. These algorithms are generally categorized into two broad groups [17]:

- *Static (offline) scheduling* this approach assigns jobs to processors before execution begins, leveraging prior knowledge of task execution times and available computing resources. The allocation decisions are made during compilation, enabling optimized but inflexible scheduling strategies.
- *Dynamic (online) scheduling*: unlike static scheduling, dynamic scheduling assigns jobs to processors during execution, requiring minimal prior knowledge of job resource requirements. This approach accommodates uncertainty in the execution environment, enabling greater adaptability to workload fluctuations.

Dynamic scheduling plays a crucial role in optimizing performance and efficiency in BD processing environments [18]. Unlike static scheduling, which relies on predetermined resource assignments, dynamic scheduling adjusts resource allocation in real time to meet fluctuating workload demands. Advanced algorithms, including ML-based techniques, have been employed to predict workload patterns and adapt resource allocations accordingly [19, 20]. By dynamically balancing loads across nodes and mitigating bottlenecks, this approach ensures that high-priority tasks receive the necessary resources promptly.

Additionally, dynamic scheduling enhances resource utilization by reallocating idle or underutilized resources, thereby maximizing throughput and minimizing latency. Due to its inherent flexibility and responsiveness, dynamic scheduling is particularly well-suited for managing the complexity and unpredictability of BD applications, leading to more efficient and resilient data processing systems. A comparative analysis of various BD resource management approaches, including the methodologies, performance metrics, and findings of relevant studies, is provided in Table 1.

The elasticity of resource allocation has also been extensively explored in the context of cloud and distributed computing environments. Verma and Kaushal [21] emphasize the advantages of elastic resource allocation, highlighting its ability to improve computational throughput by dynamically adjusting resource availability in response to workload fluctuations. However, a persistent challenge in elastic resource allocation lies in ensuring that resources are optimally distributed in alignment with workload demands. Addressing these critical challenges, This paper introduces a two-tiered, fine-grained scheduling mechanism (*BigOPERA*) designed to enhance resource efficiency and adaptability in BD environments.

2.3 Related work

The paradigm of dynamically creating disposable clusters, as espoused by BigOPERA, finds echoes in the work of Hindman et al. [22], wherein Mesos is introduced as a platform for sharing commodity clusters between multiple diverse cluster computing frameworks, ensuring optimal

Table 1 Comparative analysis of different works in BD Resource Management

Paper	Scalability	Elasticity	Resource Allocation	Performance	Cost-Effectiveness	Usability	Interoperability	Adaptability
Mesos [22]	5	3	5	4	4	3	5	5
K8s [23]	3	5	5	5	4	4	5	5
Moon [12]	2	3	3	3	5	3	3	3
Sparrow [24]	4	2	4	4	4	3	3	3
BigOPERA	4	4	5	5	5	4	4	5

Table presents a comparative analysis of various BD Resource Management, with scores ranging from 1 to 5. Each score reflects how effectively the respective paper addresses key issues such as Scalability, Elasticity, Resource Allocation Efficiency, Performance, Cost-Effectiveness, Usability, Interoperability, and Adaptability. The values were calculated based on the degree to which each work provides solutions or strategies for managing these critical aspects within BD environments

resource utilization by sharing resources in a fine-grained manner among different applications. Similarly, MOON [12] introduces a novel approach to leveraging opportunistic computing environments for MapReduce workloads. This approach extends the concept of dynamic cluster creation by harnessing transient and opportunistic resources and optimizing resource utilization for distributed data processing tasks. The work described in [12] highlights the importance of adapting to changing resource availability and aligns with the broader trends in dynamic cluster management for efficient, cost-effective, and scalable distributed computing.

In the context of mobile edge computing (MEC), Chen et al. [25] propose a Mobility-aware Service Migration (MSM) scheme that leverages data-driven approaches to optimize service migration in dynamic environments. MSM focuses on reducing service delays by mining user mobility patterns and employing deep reinforcement learning (DRL) to make optimal migration decisions. This approach is particularly relevant to BigOPERA's goal of dynamic resource allocation, as both frameworks aim to optimize resource utilization in environments with fluctuating workloads and resource availability. While MSM targets service migration in MEC scenarios, BigOPERA extends this concept to big data processing by integrating opportunistic resources with Apache Spark, demonstrating the broader applicability of dynamic resource management strategies across different computing paradigms.

The increasing adoption of Apache Spark has brought resource utilization to the forefront of research. Both industry and academia have made significant efforts to enhance Spark's performance and efficiency, such as the TR-Spark project [26]. Other studies, particularly those focusing on cloud environments, have worked on improving Spark's resilience by optimizing the use of *spot* instances [27] and focusing on cost savings through better resource utilization [28].

OpERA [13] represents a significant effort to enhance resource utilization within BD frameworks by modifying the Fair Scheduler of Hadoop YARN [29]. The study proposes a strategy to increase resource utilization and reduce job execution times by reallocating available resources to pending tasks based on runtime utilization knowledge. The central concept is to improve resource sharing; the scheduler can then reallocate idle or underutilized resources from one task to another waiting task. This creates a more granular scheduling approach aimed at optimizing resource use within the dedicated resource pool. In contrast, BigOPERA extends this concept by incorporating non-dedicated workers, thus expanding resource utilization beyond the confines of the cluster's dedicated resources. Hence, many paradigms, such as data stream processing [30], can utilize it.

Our work distinguishes itself by proposing a hybrid model approach for Apache Spark, which, to our knowledge, remains unexplored in current literature. Moreover, our solution can be universally applied across cloud and local cluster systems.

3 Problem formulation and motivation

This section defines the problem, outlines the motivation behind our approach, presents our key contributions, and formalizes the mathematical framework guiding our proposed solution.

3.1 Problem statement

With the advent of the BD era, practitioners have started building larger clusters using data lake/data hub architectures to support data-intensive applications. These clusters aim to accommodate a variety of dissimilar workloads for the daily data analytics and operations of the organization. Meanwhile, HTCondor is a leading and primarily used at

High-Throughput Computing (HTC) facilities, such as particle accelerator research centres [31], with efficient resource-sharing across distributed nodes for the broader HTC applications.

BigOPERA aims to enable the coexistence of Apache Spark and HTCondor within the same physical confines of an enterprise or data centre. This initiative strives to reduce the fragmentation of data centre resources and eliminate computational silos. Additionally, BigOPERA manages the entire Spark lifecycle by leveraging containerization technology, specifically Docker containers.

Containerization technologies have revolutionized application deployment and management, addressing longstanding issues of resource inefficiency. With tools like Docker, which offer lightweight and isolated environments, applications have become significantly more scalable and more accessible to manage [23]. These advancements were pivotal in developing container orchestration systems such as Borg and Kubernetes [32].

Containers have changed how we build and deploy software. Bundling applications, configurations, libraries and dependencies in a consistent format allows software developers to create a new level of abstraction. These new containerized applications shifted the focus in data centres from hardware to the applications themselves, underscoring its far-reaching impact.

By using containers, BigOPERA allows Spark to run as an HTCondor framework. So, Spark applications (i.e., BD-like) can run side-by-side with HTCondor (i.e., HTC-like) and dynamically share cluster resources. BigOPERA abstracts the cluster resources to achieve efficient task scheduling and provides standard data-driven services in a sandbox (using container technology) environment. Moreover, the project aims to answer urgent questions concerning large-scale distributed clusters: utilization, flexibility, scalability, and performance. Table 2 illustrates the challenges and issues we are addressing with BigOPERA.

3.2 Motivation and objectives

Although *opportunistic computing* has been explored in various distributed systems, no existing work has combined Apache Spark and HTCondor on a unified resource pool with a common scheduling mechanism. Traditional BD frameworks primarily focus on static cluster management (e.g., Spark on YARN) or opportunistic job execution (e.g., HTCondor), but not both in a fully integrated system. Our work bridges this gap by harmoniously integrating Spark and HTCondor within the same computing infrastructure, allowing dynamic workload scheduling, better resource sharing, and improved execution efficiency.

This novel integration enhances the full utilization of containerized computing environments, allowing Spark workloads to leverage dedicated and non-dedicated resources effectively. By eliminating resource silos, the proposed framework unifies Spark's in-memory processing with HTCondor's batch job scheduling, enabling elastic, multi-framework computing where both interactive and batch-processing workloads coexist on the same infrastructure. This hybrid approach maximizes workload efficiency, minimizes idle resources, and offers new possibilities for real-time federated learning, multi-cluster resource sharing, and energy-efficient computing [33–35].

The primary objective of this study is to develop and evaluate a hybrid resource management framework that integrates opportunistic computing with Apache Spark through HTCondor. The proposed framework introduces an adaptive scheduling strategy that dynamically balances dedicated and opportunistic resources to optimize execution efficiency. By leveraging container technology (Docker/Kubernetes), the system ensures flexible, scalable,

Table 2 Defining BigOPERA's knowledge domain

Numbers	Problem description	Type
1	Low utilization of most modern commodity computers and clusters	Utilization
2	Resources static partitioning within a physical confine of an enterprise	Flexibility
3	Statically sizing the cluster on peak utilization and installing new servers to enhance the throughput when demand	Flexibility
4	Increasing gap between computation and I/O capacity on high-end workstations for scientific experiments	Performance
5	Dependency on multi-tenant (e.g., cloud computing) raises many issues, such as security, data movement, performance degeneration, etc.	Utilization
6	Wasting many computing cycles for testing middleware and applications that are not in production yet	Utilization
7	The need to enhance the performance with minimal cost of deployment	Performance
8	Running a new type of experiment on the same resources with minimal configuration and keeping it extensible to any prior design	Flexibility
9	Reducing carbon footprint	Optimization

and fault-tolerant resource utilization, making it suitable for large-scale, distributed data analytics.

A critical aspect of this work is to validate the performance improvements of this integration through comprehensive benchmarking, measuring workload execution time, resource efficiency, and scalability under various workload conditions. Furthermore, this research explores how the unified framework can contribute to more efficient modern computing workflows, cross-cluster resource allocation, and reduced environmental impact through optimized energy consumption. By addressing these challenges, this study lays the foundation for fully utilized, cost-effective, and scalable computing environments, ultimately improving the efficiency of modern BD processing frameworks.

3.3 Mathematical model for resource management

In BD processing systems, resource management can be formulated as an optimization problem to minimize costs or maximize performance, subject to various constraints.

3.3.1 Objective function

The objective function is defined as follows:

$$\min_{x} \sum_{i=1}^{N} C_i(x_i), \tag{1}$$

where:

- x_i: resources allocated to task i,
- $C_i(x_i)$: cost associated with allocating x_i resources to task i,
- N: total number of tasks.

3.3.2 Constraints

(1) *Resource Capacity Constraint* this constraint ensures that the total allocated resources do not exceed the available capacity R_{total}:

$$\sum_{i=1}^{N} x_i \leq R_{\text{total}}. \tag{2}$$

(2) *Task Requirement Constraint* each task i must receive at least its minimum required resources $R_i^{\min}$:

$$x_i \geq R_i^{\min}, \quad \forall i \in \{1, 2, \ldots, N\}. \tag{3}$$

(3) *Quality of Service (QoS) Constraint* the performance level of each task i, denoted as $P_i(x_i)$, must meet or exceed the minimum acceptable performance level $Q_i^{\min}$:

$$P_i(x_i) \geq Q_i^{\min}, \quad \forall i \in \{1, 2, \ldots, N\}. \tag{4}$$

The objective function minimizes the total cost of resource allocation across all tasks, while the constraints ensure:

- Resource usage remains within the system's capacity.
- The minimum resource requirements of each task are satisfied.
- The quality of service levels is maintained for all tasks.

4 BigOPERA framework: architecture and design

The architecture of BigOPERA integrates a hybrid computing environment combining HTCondor for opportunistic computing with Apache Spark for distributed data processing and Kubernetes for managing dedicated resources. This dual-faceted approach allows for dynamic workload execution by seamlessly integrating both dedicated and non-dedicated computing resources, optimizing both cost and performance.

The architecture visually depicted in the figure (see Fig. 1) is divided into two main sections: the Opportunistic Pool and the Dedicated Pool, highlighted by green and purple dashed lines.

The Opportunistic Pool, managed by HTCondor, consists of non-dedicated nodes that execute Spark tasks opportunistically whenever spare computational resources become available. These nodes must support Docker containers and maintain connectivity to the dedicated network directly or via VPN. As nodes become available, they automatically register with the HTCondor Manager, signalling their readiness to run Spark jobs. This dynamic registration enables the architecture to scale flexibly and efficiently based on resource availability.

The HTCondor Manager, including components like the Negotiator, Collector and Scheduler (Sched), oversees job scheduling and resource allocation. Worker nodes in this pool are equipped with Started and Starter daemons (shown in green color), and they run Spark Worker Nodes capable of executing tasks dynamically. These Spark workers (red color) utilize Docker and control groups (cgroups) to run multiple tasks concurrently, efficiently utilizing available resources.

HTCondor daemons periodically report node status and resources to the central manager, providing essential data such as CPU, memory, disk usage, and network availability. This information is crucial for making informed scheduling decisions and ensuring efficient task execution.

Conversely, the Dedicated Pool operates under Kubernetes, ensuring stable computational capacity for Spark

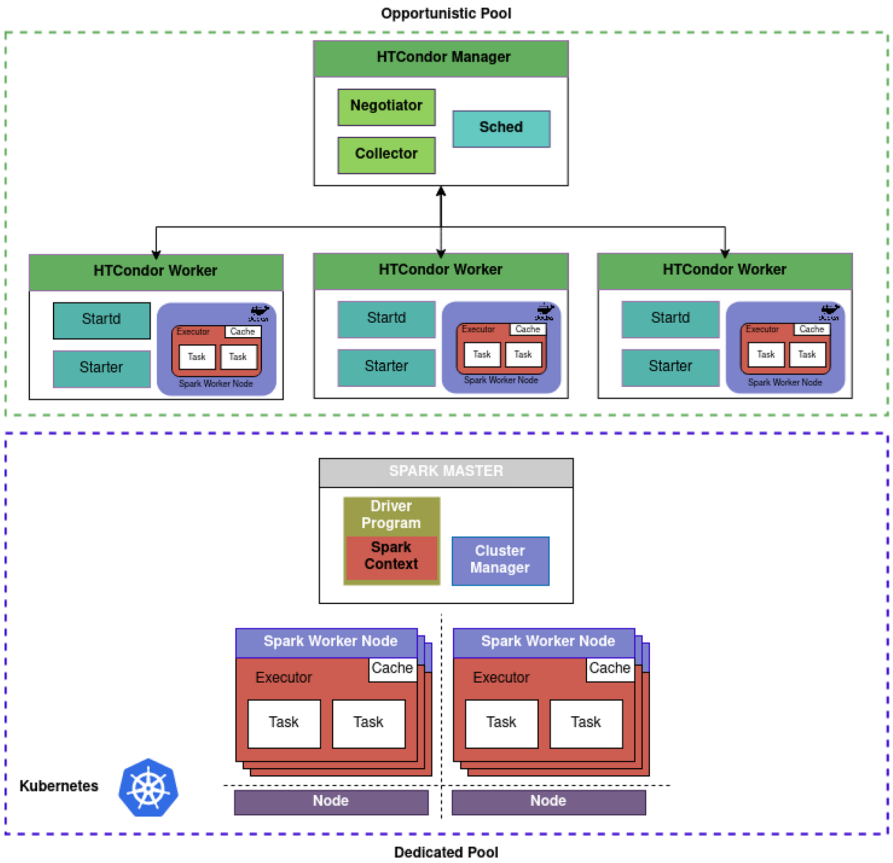


Fig. 1 BigOPERA architecture (Color figure online)

workloads. Here, the Spark Master orchestrates job execution through its Driver Program, Spark Context (red color), and Cluster Manager, distributing tasks across multiple Spark Worker Nodes (also shown in red). Each worker node hosts Executors to run Spark tasks and Caches to store frequently accessed data, thereby enhancing processing efficiency.

YARN NodeManagers within this architecture report resource availability to the YARN ResourceManager (not depicted for simplification), which then allocates containers and schedules tasks accordingly. This interaction optimizes resource usage and task execution across the cluster.

The combination of these two pools allows the architecture to balance cost and performance effectively. HTCondor, with its capability to manage opportunistic resources and Docker containerization, complements YARN's robust resource scheduling and container management. Together, they create a resilient, scalable, and efficient environment for running large-scale data analytics workflows with Apache Spark, dynamically leveraging both dedicated and available non-dedicated resources.

This comprehensive architecture, by integrating both opportunistic and dedicated computing resources, provides a flexible and scalable solution for handling extensive data processing tasks efficiently.

4.1 BigOPERA machine status

BigOPERA assigns machine statuses based on the availability of resources and their readiness to execute tasks. The system operates through five primary states, each representing a specific phase in a node's lifecycle within the BigOPERA framework. These states are designed to ensure efficient resource utilization and task scheduling, even in dynamic and heterogeneous environments. The machine states, and their transitions are as follows:

- OWNER indicates that the owner is using the node (e.g., running local tasks or processes); hence, it is not available to run a non-dedicated job.
- UNCLAIMED indicates that the node is idle and can run a non-dedicated job, yet it currently does not run any.
- MATCHED indicates that the node can run a non-dedicated job, and BigOPERA allocated it in the resource pool and matched it with a Driver Pod. That Driver Pod has not yet claimed this node by launching an Executor Pod. In this state, the machine cannot launch a job in the Executor Pod.
- CLAIMED indicates that the node can run a non-dedicated job, and BigOPERA assigned it with an Executor Pod, i.e., active to utilize this node.

– PREEMPTING: the node is in the process of being reclaimed by its owner, and the current task is being preempted.
– ERROR: the node has encountered an unrecoverable issue (e.g., hardware failure, network disconnection, or software crash) and cannot participate in task execution until the issue is resolved.

Another state not mentioned in Fig. 2 is DRAINED, in which the node does not respond to the owner or BigOPERA. The ERROR state is also not depicted in the figure for simplicity, but it plays a critical role in handling failures and ensuring system robustness.

State transitions: **A** The node switches from OWNER to UNCLAIMED whenever the START expression no longer locally evaluates to FALSE. This indicates that the node can run a BigOPERA job.

B The transition from UNCLAIMED to MATCHED happens whenever the DriverPod matches this resource with a BigOPERA job.

C The transition from UNCLAIMED directly to CLAIMED also happens if the DriverPod matches this resource with a BigOPERA job. In this case, the ExecutorPod receives the match and initiates the claiming protocol with the node before the DriverPod receives the match notification from BigOPERA DriverPod.

D The node moves from MATCHED to OWNER if the START expression locally evaluates to FALSE or the MATCH_TIMEOUT timer expires. This timeout ensures that if a node is matched with a given ExecutorPod but that ExecutorPod does not contact the DriverPod to claim it, the node will give up on the match and become available to be matched again. In this case, since the START expression does not locally evaluate to FALSE, the node will immediately enter the Unclaimed state again (via transition A). The node might also go from MATCHED to OWNER if the ExecutorPod attempts to perform the claiming protocol but encounters an error.

E The transition from MATCHED to CLAIMED occurs when the ExecutorPod completes the claiming protocol with the DriverPod.

F If the resource were matched to a job with a higher priority, it would move from PREEMPTING back to CLAIMED.

G The resource would move from PREEMPTING to OWNER if the PREEMPT expression had evaluated to TRUE or if the START expression locally evaluated to FALSE when the DriverPod had finished evicting whatever job it was running when it entered the PREEMPTING state.

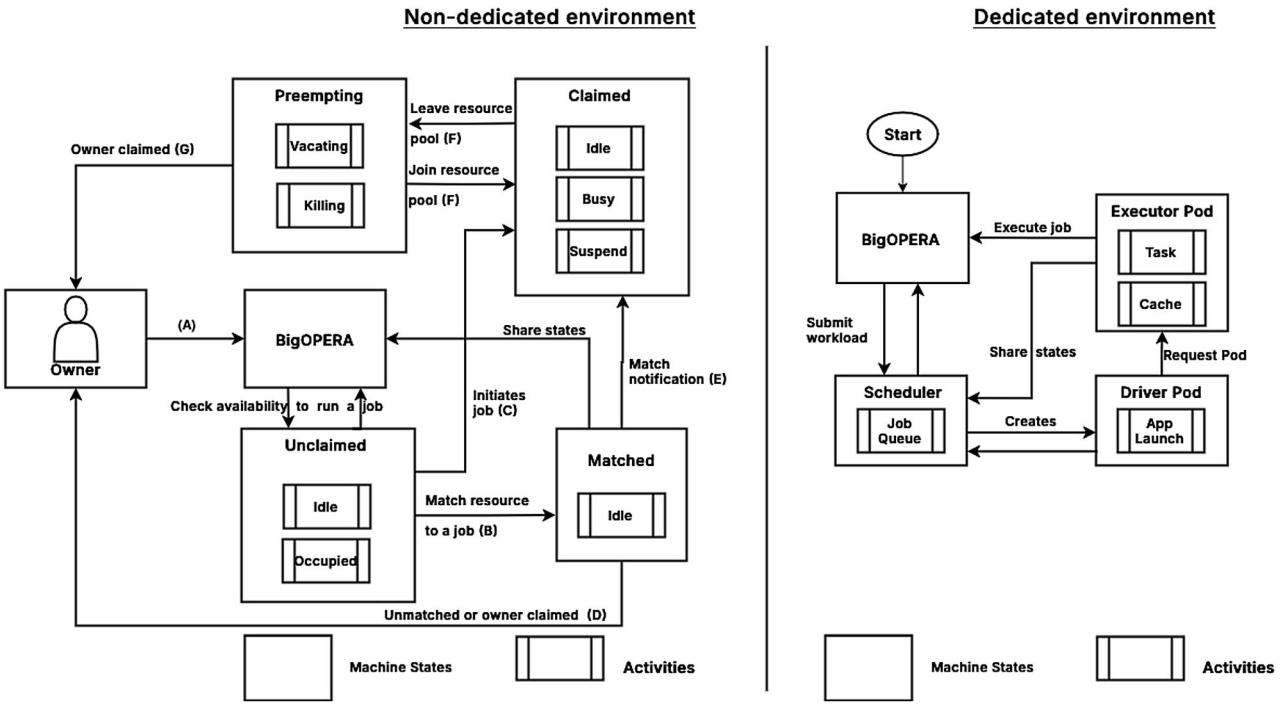


Fig. 2 BigOPERA environments machine states and activities

Summary:

- Transitions are event-driven: each state transition is triggered by specific events, such as task completion, owner reclaim, or task assignment.
- Timeout Mechanism: if a task does not start within a specified time after being matched, the node returns to the UNCLAIMED state to avoid resource wastage.
- Preemption Handling: the PREEMPTING state ensures that tasks can be gracefully interrupted if the owner reclaims the node, allowing for efficient resource sharing without disrupting ongoing operations.

4.2 BigOPERA algorithms

The BigOPERA framework relies on three core algorithms to manage resource allocation, task execution, and fault tolerance. These algorithms work together to efficiently utilise both dedicated and non-dedicated (opportunistic) resources while maintaining system reliability and performance. This section discusses BigOPERA algorithms by separating distinct processes: managing initialization and periodic monitoring of the state of the node, including transitions from OWNER to UNCLAIMED when idle and to ERROR upon fault detection; handling task allocation and execution, with state transitions between UNCLAIMED, MATCHED, and CLAIMED based on resource availability and task readiness; and focusing on reclaim management, preemption, and error recovery to ensure resources are appropriately freed and recovery attempts are made when transitioning from ERROR to UNCLAIMED, making the system easier to follow and maintain. Below, we break down each algorithm in a clear and structured way.

Algorithm 1 is responsible for initialization and state monitoring. In this initial part of the BigOPERA algorithm, the system begins by setting the state of the node to OWNER. The algorithm then periodically checks if the node becomes idle. At this point, it transitions to the UNCLAIMED state, making resources available for task allocation. It also monitors for any errors in the local function. If an error is detected, the node transitions to the ERROR state, where recovery can later be attempted. This section lays the groundwork for efficient resource management by continuously assessing the availability and integrity of the node.

Algorithm 2 is responsible for task allocation and execution. The algorithm focuses on matching resources to tasks and managing their execution. When the node is UNCLAIMED, the scheduler checks if resources meet any pending task requirements. If resources are sufficient, it assigns a task, reserves resources, and transitions to the MATCHED state. Once a task is ready to begin, the algorithm transitions to the CLAIMED state, where it executes the task. It also includes a timeout mechanism, releasing resources and returning to UNCLAIMED if the task does not start on time. This feature ensures that tasks are allocated and managed efficiently, maximizing resource utilization.

Algorithm 1 BigOPERA: Initialization and State Monitoring

```
1: States: OWNER, UNCLAIMED, MATCHED, CLAIMED, PREEMPTING, ERROR
2: Initialize resource state as OWNER
3: while True do {Periodically check when the node becomes idle}
4:    if START local function evaluates to false then
5:       Transition to UNCLAIMED
6:    end if
7:    if error detected in local function then
8:       Transition to ERROR
9:    end if
10: end while
```

Algorithm 2 BigOPERA: Task Allocation and Execution

```
while node is active do
    if state is UNCLAIMED then
        Scheduler Role: Check if resources meet the requirements of any pending tasks
        if sufficient resources are available for a task then
            Scheduler assigns task to resources
            Transition to MATCHED
            Reserve resources for the assigned task
        end if
    end if
    if state is MATCHED and task is ready to start then
        if owner requests reclaim of node then
            Transition to OWNER
        end if
        if timeout occurs before the task starts then
            Transition to UNCLAIMED
            Release reserved resources
        else if task is ready to start then
            Transition to CLAIMED
            Start executing the task using reserved resources
        end if
    end if
end while
```

Algorithm 3 BigOPERA: Reclaim Management and Error Handling

```
if state is CLAIMED then
    Check if the owner requests the node back:
    if owner requests reclaim of resources during task execution then
        Transition to PREEMPTING
        Notify Spark master of preemption
        Stop accepting new tasks
        Interrupt current task (if needed) or allow the task to finish
        Save task progress if applicable
        Free up resources
        Transition to UNCLAIMED
        Notify the master of resource availability
    end if
    if task completes (without preemption) then
        Free up resources
        Transition to UNCLAIMED
    end if
end if
if state is PREEMPTING then
    Wait for the current task to finish (if any)
    Free up resources
    Transition to UNCLAIMED
    Notify Spark master of resource availability
end if
if state is ERROR then
    Attempt to recover from error
    Transition to UNCLAIMED if recovery successful
end if
```

Algorithm 3 reclaims management and handles error. In particular, it handles owner reclaim requests, task preemption, and error recovery. If the node is in the Claimed state and the owner requests the node back, the algorithm transitions to Preempting, stops accepting new tasks, and interrupts or allows the current task to finish. It notifies the Spark master of resource availability and saves task progress if necessary. In the Error state, the algorithm attempts to recover and transition back to Unclaimed if successful. This part ensures that resources are managed responsibly in response to reclaim requests and errors, enhancing system reliability.

Following is a simplified flowchart of the state transitions in BigOPERA:

- OWNER → UNCLAIMED: node becomes idle and available for BigOPERA tasks.
- UNCLAIMED → MATCHED: resources are matched to a task.
- MATCHED → CLAIMED: task starts execution.
- CLAIMED → PREEMPTING: owner reclaims the node, and the task is preempted.
- PREEMPTING → UNCLAIMED: resources are freed after preemption.
- ERROR → UNCLAIMED: node recovers from an error and becomes available again.

5 Experiment setting

This section presents the performance evaluation of the BigOPERA system executing Spark jobs (BigOPERA Spark). First, we detail the cluster configurations used on Google Cloud Platform (GCP). Next, we describe the benchmarks selected for this evaluation, providing a comprehensive list of the applications tested and the data sizes used.

5.1 Environment

The evaluation of BigOPERA Spark was conducted by comparing the performance of a dedicated cluster with a hybrid cluster composed of dedicated and opportunistic resources, both running on Google's GCP. The dedicated cluster consisted of four worker nodes type *n1-standard-2*, with two vCPUs and 7.5 GB of memory plus an additional node for the control plane running on Kubernetes. In contrast, the hybrid setup leveraged a combination of that dedicated cluster alongside four additional *e2-medium* with 2vCPU, 4 GB of memory nodes, and a *e2-medium* HTCondor manager node (not running in Kubernetes), a 1:1 opportunistic to dedicated node ratio was created. We set a 2 GB limit per executor for the memory difference between instances. The experimental settings were primarily based on the default configurations of Intel HiBench. This ensures that the results are comparable to other studies using the same benchmark. Additionally, specific parameters were adjusted for our particular environment, as recommended in the official documentation [36]. The number of YARN executors was set to four/eight, and two cores, ensuring that each worker fully utilized its two cores for efficient parallel execution. Memory allocation was set to 2 GB, limiting executor memory to accommodate differences in available memory across instance types. Parallelism settings were configured to 8/16 for *map* and *shuffle*, as clusters require sufficiently high parallelism levels to maximize utilization. According to Spark documentation, setting these values to twice the number of cores is recommended [37].

These configurations align with Apache guidelines, strengthening the validity of the experimental setup.

5.2 Benchmark

Our workloads focused on several representative MapReduce applications: Wordcount, Sort, Terasort and SparkPi. To automate the configuration, execution and results gathering, we used Intel's HiBench Suite.[1]

HiBench [38] is an extensive benchmark suite for BigData platforms such as Hadoop, Spark, Storm, etc. It includes a total of 29 workloads divided into six categories: Micro, machine learning, SQL, graph, web search, and streaming. The applications used for this evaluation were:

- (1) Wordcount: this application counts how often each word appears within the input data generated using RandomTextWriter.
- (2) Sort: this application sorts its text input data, generated using RandomTextWriter.
- (3) Terasort: this is a standard benchmark created by Jim Gray. Hadoop TeraGen generates the input data.
- (4) SparkPi: computes an approximation to π by throwing darts at a circle. It is not included on HiBench but is available as an example on Spark.

Table 3 shows the input data sizes used for the HiBench benchmarks. In the case of SparkPi, we began with 200,000 partitions and incrementally increased the number by 100,000, reaching up to 800,000 partitions.

We simulated ideal conditions for these tests, where no opportunistic node was reclaimed during task execution. In the following section, we deeply analyze the performance impact of node failures.

6 Results and discussion

This section presents a comprehensive analysis of the performance of the system, followed by an in-depth examination of its fault tolerance capabilities. The performance analysis focuses on execution times using the benchmarks described in the previous section, providing insights into the system's efficiency and scalability. Following this, we investigate the fault tolerance of the

[1] All the code necessary for running these benchmarks has been ported to Python 3, and it is available at: https://github.com/kadern0/HiBench.

Table 3 Input datasizes

	Tiny	Large	Huge	Gigantic	Bigdata	2-Bigdata
Wordcount	*NU*	388.4 MB	3.79 GB	37.9 GB	89.6 GB	379.3 GB
Sort	*NU*	38.8 MB	388.4 MB	3.79 GB	35.5 GB	71.1 GB
Terasort	3.0 MB	305.18 MB	2.98 GB	29.8 GB	298.02 GB	*NU*

NU = Not Used for the corresponding benchmark

framework, evaluating its ability to maintain functionality and recover from node losses.

6.1 Performance

In Fig. 3a, the difference in execution times for the Wordcount benchmark is illustrated, with BigOPERA Spark showing a significant improvement over Apache Spark. On average, BigOPERA was approximately 35% faster, consistently outperforming Apache Spark alone.

In the Sort benchmark (Fig. 3b), we observed varying degrees of improvement with BigOPERA over Apache Spark. For datasets smaller than 400 MB, BigOPERA was slightly slower. However, for larger datasets, BigOPERA exhibited execution times between 10 and 30% faster than Apache Spark, with the performance advantage increasing with dataset size.

Similarly, in the Terasort benchmark (Fig. 3c), BigOPERA Spark demonstrated notable performance gains over Apache Spark. For datasets smaller than 300 MB, the execution times were comparable. For larger datasets, BigOPERA showed an improvement of 10% to 25% compared to Apache Spark across multiple trials. However, the degree of improvement diminished with increasing dataset size.

In the SparkPi benchmark (Fig. 3d), BigOPERA showcased substantial improvement in execution time over Apache Spark. On average, BigOPERA completed the computation approximately 30% faster. Similar to the Terasort benchmark, the performance gain decreased with larger workloads.

The observed decrease in performance is attributable to the usage of e2-medium instances, which are optimized for cost and employ dynamic resource management for efficient resource allocation. Unlike dedicated vCPUs, the vCPUs in e2 instances are shared among multiple tenants, leading to potential performance degradation under higher utilization. This dynamic allocation allows Google Cloud to maximize resource usage and cost efficiency, but it can result in inconsistent performance, particularly when the demand exceeds 25% CPU utilization. Therefore, while e2 instances are highly cost-effective and well-suited for applications with lower and predictable CPU usage, such as development and testing environments, they are less ideal for production workloads that require consistent and robust performance [39, 40].

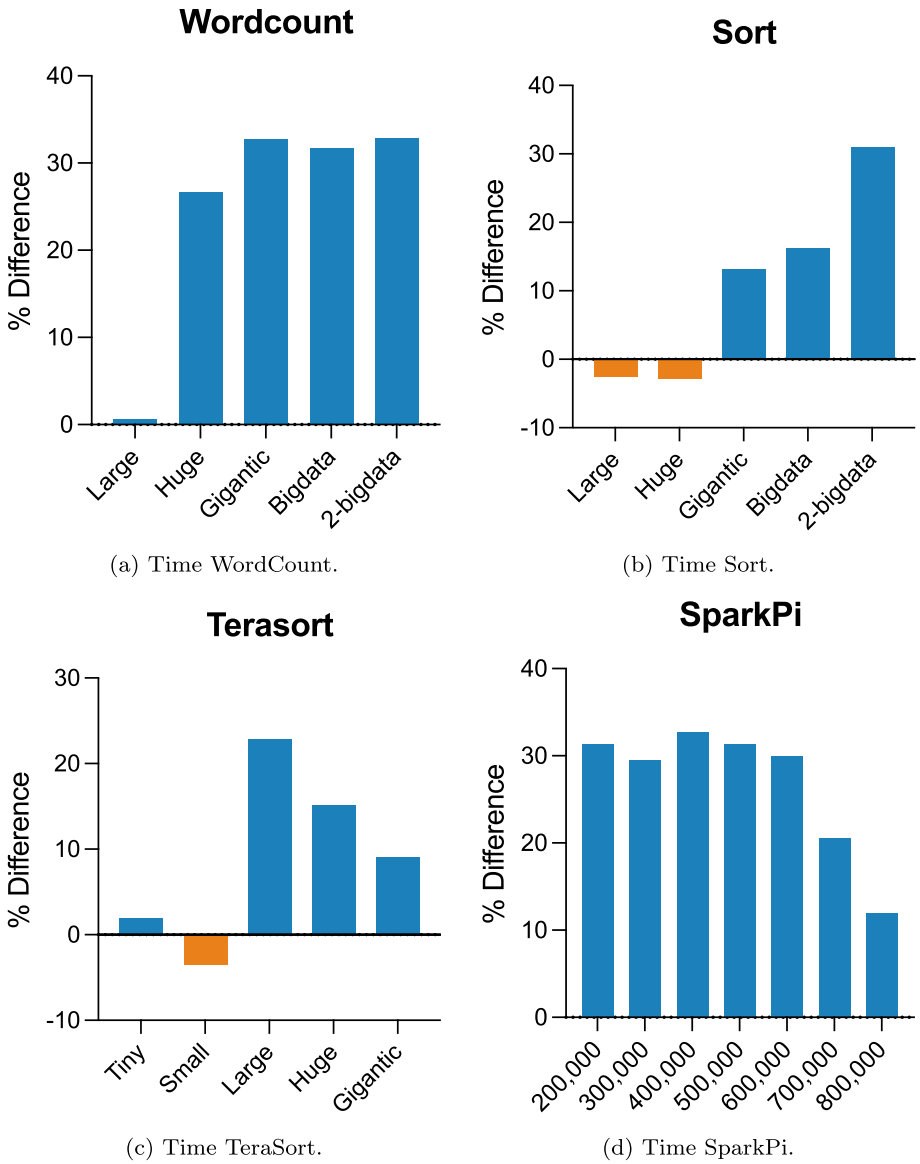
The effectiveness of this integrated approach is demonstrated in the various benchmarks completed during this research (see Sect. 5.2), showcasing the ability of the system to handle diverse and intensive data processing tasks efficiently. The combination of dedicated and non-dedicated nodes managed through sophisticated resource allocation and scheduling mechanisms ensures that BigOPERA can meet the demands of large-scale data analytics with high performance and reliability.

6.2 Fault tolerance

Given the transient nature of opportunistic resources, where nodes can be reclaimed, shut down, or disconnected from the network, it is crucial to assess the capacity of the system to cope with node losses. Initial Apache Spark revisions encountered significant challenges in dealing with tasks requiring moderately transient resources. This issue was particularly evident in scenarios where Spark struggled to complete tasks, leading to application failures that were difficult to diagnose and resolve.

The study by Bowen Yu et al. noted that the early versions of Spark did not handle tasks efficiently when resources were short-lived, highlighting a critical limitation in its original design [41]. The works on [26, 27, 42] improved Spark resiliency to node loss, focused mainly on improving reliability and performance when using spot instances in cloud environments. Nevertheless, these improvements also made possible opportunistic computing to run Spark jobs.

To evaluate the tolerance for BigOPERA node disruption, we measured the execution time of the SparkPi application with 800,000 partitions using the same setup as in Fig. 3d. In this case, we introduced node interruptions to assess their impact on overall duration. We used the execution time of the benchmark on the dedicated cluster (Fig. 3d) as the baseline measurement and subsequently simulated node failures by randomly terminating the opportunistic nodes, thus mimicking real-world conditions. Assuming mutual independence of node outages, we executed a script that randomly stopped and restarted the HTCondor tasks.



(a) Time WordCount.

(b) Time Sort.

(c) Time TeraSort.

(d) Time SparkPi.

Fig. 3 Difference in execution times between Apache Spark and BigOPERA Spark (the higher, the better) for different benchmarks

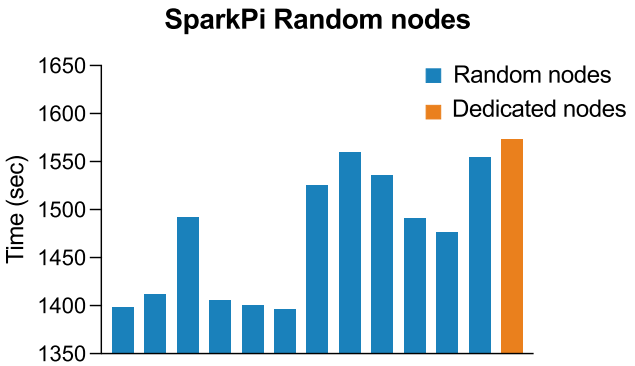


Fig. 4 SparkPi on random nodes

Figure 4 shows that the hybrid cluster was faster than the dedicated cluster. Despite the expected variability in execution times, the hybrid cluster was consistently quicker, demonstrating that the opportunistic nodes, despite being randomly stalled, could provide a performance advantage. These findings suggest that the ability of the opportunistic cluster to utilize idle resources effectively outweighed the potential disruptions caused by the random node loss. Overall, the experimental results indicate that BigOPERA Spark offers significant performance enhancements in various benchmark tasks compared to Apache Spark. These findings highlight the potential of BigOPERA Spark for improving the efficiency and scalability of large-scale data analytics workflows.

7 Conclusions and future work

The BigOPERA framework represents a significant advancement in resource management for BD processing by integrating dedicated and non-dedicated computing resources. By seamlessly combining Apache Spark and HTCondor within a unified architecture, BigOPERA enhances flexibility, scalability, and resource utilization, enabling more efficient task scheduling across heterogeneous environments. This integration eliminates resource silos, optimizes computational efficiency, and supports dynamic workload execution with minimal infrastructure modifications.

Experimental evaluations, including Wordcount, Sort, Terasort, and SparkPi, demonstrate substantial performance improvements of up to 35% over traditional Apache Spark setups. These enhancements are particularly evident in large-scale data processing tasks, where BigOPERA consistently outperforms conventional configurations, effectively managing complex and high-volume datasets. Additionally, rigorous fault-tolerance testing confirms BigOPERA's resilience in handling intermittent resource availability, ensuring stable and reliable execution even in dynamic and unpredictable environments. Furthermore, by leveraging otherwise idle resources, BigOPERA contributes to sustainable computing practices, minimizing energy consumption and reducing the carbon footprint of data centres.

Future work will optimize the integration and scheduling mechanisms to enhance the framework's efficiency across more diverse computing environments. One promising direction is incorporating advanced machine learning techniques for predictive resource management. Integrating Actor–Critic Deep Reinforcement Learning (A3C) [43] could enable BigOPERA to adjust resource allocations based on anticipated workload variations rather than reactively responding to demand fluctuations. Such an approach would enhance adaptive and energy-efficient resource provisioning, improving overall system performance and cost-effectiveness. A hybrid approach combining BigOPERA's opportunistic resource management with reinforcement learning-driven predictive optimization could further refine task scheduling strategies.

Building on this, future research will also explore the integration of advanced techniques like traffic-aware hierarchical offloading, as demonstrated in THOAS (Chen et al. 2024), and personalized federated learning, as proposed in PFR-OA (Chen et al. [44, 45]). These methods have the potential to significantly enhance BigOPERA's adaptability and efficiency in heterogeneous environments, particularly in scenarios with fluctuating workloads and diverse user demands. By incorporating these innovations, BigOPERA could achieve even greater performance and scalability, solidifying its position as a robust framework for dynamic resource management.

Author contributions Pablo V. Caderno: Designed the two-tiered scheduling mechanism, and led the experimental design and data analysis. Contributed significantly to the manuscript's writing and final review. Feras Awaysheh: Conceptualized the BigOPERA framework, and played a critical role in implementing the resource allocation algorithms and containerization setup. Assisted in analyzing experimental results and contributed to the manuscript preparation and revisions. José C. Cabaleiro: Provided expertise in Big Data frameworks and cluster computing. Assisted in refining the theoretical framework and contributed to the interpretation of results and manuscript writing. Tomás F. Pena: Supervised the research project, provided guidance on experimental methodologies, and critically reviewed the manuscript for intellectual content and clarity. All authors reviewed and approved the final version of the manuscript.

Funding Open access funding provided by Umea University. This work has received financial support from the Agencia Estatal de Investigación (Spain) (PID2022-141623NB-I00), the Xunta de Galicia - Conselleria de Educación, Ciencia, Universidades e Formación Profesional (Centro de investigación de Galicia accreditation 2024-2027 ED431G-2023/04) and Reference Competitive Group accreditation ED431C-2022/016) and the European Union (European Regional Development Fund - ERDF).

Data availability No datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors declare no competing interests.

References

1. Sun, Z.: Data, analytics, and intelligence. J. Comput. Sci. Res. **5**, 43–57 (2023). https://doi.org/10.30564/jcsr.v5i4.6072
2. Awaysheh, F.M., Alazab, M., Gupta, M., Pena, T.F., Cabaleiro, J.C.: Next-generation big data federation access control: a reference model. Future Gener. Comput. Syst. **108**, 726–741 (2020)
3. Fakhouri, H.N., Alawadi, S., Awaysheh, F.M., Alkhabbas, F., Zraqou, J.: A cognitive deep learning approach for medical image processing. Sci. Rep. **14**(1), 4539 (2024)

4. Fakhouri, H.N., Awaysheh, F.M., Alawadi, S., Alkhalaileh, M., Hamad, F.: Four vector intelligent metaheuristic for data optimization. Computing **106**(7), 1–39 (2024)
5. Nda, R., Tasmin, R.: Big data management in education sector: an overview. Path Sci. **5**(6), 5009–5014 (2019). https://doi.org/10.22178/pos.47-6
6. Awaysheh, F.M., Alazab, M., Garg, S., Niyato, D., Verikoukis, C.: Big data resource management and networks: taxonomy, survey, and future directions. IEEE Commun. Surv. Tutor. **23**(4), 2098–2130 (2021)
7. Xu, M., Buyya, R.: Managing renewable energy and carbon footprint in multi-cloud computing environments. J. Parallel Distrib. Comput. **135**, 191–202 (2020). https://doi.org/10.1016/j.jpdc.2019.09.015
8. Awaysheh, F.M., Aladwan, M.N., Alazab, M., Alawadi, S., Cabaleiro, J.C., Pena, T.F.: Security by design for big data frameworks over cloud computing. IEEE Trans. Eng. Manag. **69**(6), 3676–3693 (2021)
9. Kumar, N., Hema, K., Sai, S., Hordiichuk, V., Menon, R., Catherene, D., Aarthy, J., Gonesh, C., Saha, G., Balaji, K.: Harnessing the power of big data: challenges and opportunities in analytics. Tuijin Jishu/J. Propuls. Technol. **44**, 1001–4055 (2023)
10. Apache Software Foundation: Apache Hadoop. Apache Software Foundation. https://hadoop.apache.org. Accessed 22 Oct 2024
11. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: cluster computing with working sets. In: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'10, 2010, p. 10. USENIX Association (2010)
12. Lin, H., Ma, X., Archuleta, J., Feng, W.C., Gardner, M., Zhang, Z.: MOON: MapReduce on opportunistic environments. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC '10, 2010, pp. 95–106. Association for Computing Machinery, New York (2010). https://doi.org/10.1145/1851476.1851489
13. Yao, Y., Gao, H., Wang, J., Mi, N., Sheng, B.: OpERA: opportunistic and efficient resource allocation in Hadoop YARN by harnessing idle resources. In: 2016 25th International Conference on Computer Communication and Networks (ICCCN), 2016, pp. 1–9 (2016). https://doi.org/10.1109/ICCCN.2016.7568553
14. Zaharia, M., Konwinski, A., Joseph, A.D., Katz, R., Stoica, I.: Improving MapReduce performance in heterogeneous environments. In: Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, OSDI'08, 2008, pp. 29–42. USENIX Association (2008)
15. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
16. Polato, I., Ré, R., Goldman, A., Kon, F.: A comprehensive view of Hadoop research—a systematic literature review. J. Netw. Comput. Appl. **46**, 1–25 (2014). https://doi.org/10.1016/j.jnca.2014.07.022
17. Bawankule, K., Kumar Singh, A., Kumar Dewaang, R.: Analysis of task scheduling in Hadoop MapReduce framework. Aust. J. Wirel. Technol. Mobil. Secur. **1** (2022). https://ausjournal.com/index.php/j/article/view/19
18. Ragab, M., Awaysheh, F.M., Tommasini, R.: Bench-Ranking: a first step towards prescriptive performance analyses for big data frameworks. In: 2021 IEEE International Conference on Big Data (Big Data), 2021, pp. 241–251. IEEE (2021)
19. Khan, T., Tian, W., Zhou, G., Ilager, S., Gong, M., Buyya, R.: Machine learning (ML)-centric resource management in cloud computing: a review and future directions. J. Netw. Comput. Appl. **204**, 103405 (2022)
20. Khan, T., Tian, W., Ilager, S., Buyya, R.: Workload forecasting and energy state estimation in cloud data centres: ML-centric approach. Future Gener. Comput. Syst. **128**, 320–332 (2022)
21. Verma, A., Kaushal, S.: Cost-time efficient scheduling plan for executing workflows in the cloud. J. Grid Comput. **13**, 495–506 (2015). https://doi.org/10.1007/s10723-015-9344-9
22. Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A.D., Katz, R., Shenker, S., Stoica, I.: Mesos: a platform for fine-grained resource sharing in the data center. In: Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, NSDI'11, 2011, pp. 295–308. USENIX Association (2011)
23. Bernstein, D.: Containers and cloud: from LXC to Docker to Kubernetes. IEEE Cloud Comput. **1**(3), 81–84 (2014). https://doi.org/10.1109/MCC.2014.51
24. Ousterhout, K., Wendell, P., Zaharia, M., Stoica, I.: Sparrow: distributed, low latency scheduling. In: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, SOSP'13, 2013, pp. 69–84. Association for Computing Machinery, New York (2013). https://doi.org/10.1145/2517349.2522716
25. Chen, Z., Huang, S., Min, G., Ning, Z., Li, J., Zhang, Y.: Mobility-aware seamless service migration and resource allocation in multi-edge IoV systems. IEEE Trans. Mob. Comput. **1**, 1–17 (2025). https://doi.org/10.1109/TMC.2025.3540407
26. Yan, Y., Gao, Y., Chen, Y., Guo, Z., Chen, B., Moscibroda, T.: TR-Spark: transient computing for big data analytics. In: Proceedings of the Seventh ACM Symposium on Cloud Computing, SoCC '16, 2016, pp. 484–496. Association for Computing Machinery, New York (2016). https://doi.org/10.1145/2987550.2987576
27. Mehrotra, U.: Spark enhancements for elasticity and resiliency on Amazon EMR (2019). https://aws.amazon.com/blogs/big-data/spark-enhancements-for-elasticity-and-resiliency-on-amazon-emr/. Accessed 22 Oct 2024
28. Chockalingam, P., Liang, E., Das, T., Stephan, J.Y.: Introducing Databricks Optimized Autoscaling on Apache Spark™ (2018). https://www.databricks.com/blog/2018/05/02/introducing-databricks-optimized-auto-scaling.html. Accessed 22 Oct 2024
29. Apache Software Foundation: Hadoop: Fair Scheduler. Apache Software Foundation (nd). https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/FairScheduler.html. Accessed 22 Oct 2024
30. Awad, A., Awaysheh, F., López, H.A.: BEST: a unified business process enactment via streams and tables for service computing. arXiv preprint (2025). arXiv:2501.14848
31. CERN IT Batch Service Documentation: CERN Batch Service User Guide. https://batchdocs.web.cern.ch/. Accessed 22 Oct 2024
32. Burns, B., Grant, B., Oppenheimer, D., Brewer, E., Wilkes, J.: Borg, Omega, and Kubernetes. Commun. ACM **59**(5), 50–57 (2016). https://doi.org/10.1145/2890784
33. Kaminaga, H., Awaysheh, F.M., Alawadi, S., Kamm, L.: MPCFL: towards multi-party computation for secure federated learning aggregation. In: Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing, 2023, pp. 1–10 (2023)
34. Nascimento, L., Awaysheh, F.M., Alawadi, S.: Data skew in federated learning: an experimental evaluation on aggregation algorithms. In: 2024 2nd International Conference on Federated Learning Technologies and Applications (FLTA), 2024, pp. 162–170. IEEE (2024)
35. Awaysheh, F.M.: From the cloud to the edge towards a distributed and light weight secure big data pipelines for IoT applications. In: Trust, Security and Privacy for Big Data, pp. 50–68. CRC Press, Boca Raton (2022)
36. Overview—Spark 3.5.4 Documentation—spark.apache.org. https://spark.apache.org/docs/3.5.4/index.html

37. Tuning—Spark 3.2.1 Documentation—archive.apache.org. https://archive.apache.org/dist/spark/docs/3.2.1/tuning.html#level-of-parallelism
38. Huang, S., Huang, J., Dai, J., Xie, T., Huang, B.: The HiBench benchmark suite: characterization of the MapReduce-based data analysis. In: 2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010), 2010, pp. 41–51 (2010). https://doi.org/10.1109/ICDEW.2010.5452747
39. Google Cloud: General-purpose machine family for Compute Engine. Compute Engine Documentation. https://cloud.google.com/compute/docs/general-purpose-machines#sharedcore. Accessed 22 Oct 2024
40. Agarwal, S.: Google Cloud's E2 VMs compared to the N2 VMs (2021). https://www.bigbitbus.com/2021/06/10/Google-Cloud-E2-N2-VMs/. Accessed 22 Oct 2024
41. Yu, B., Cao, H., Shan, T., Wang, H., Tang, X., Chen, W.: Sparker: efficient reduction for more scalable machine learning with Spark. In: Proceedings of the 50th International Conference on Parallel Processing, ICPP '21, 2021. Association for Computing Machinery, New York (2021). https://doi.org/10.1145/3472456.3472499
42. Awaysheh, F.M., Pena, T.F., Cabaleiro, J.C.: EME: an automated, elastic and efficient prototype for provisioning Hadoop clusters on-demand. In: CLOSER, 2017, pp. 709–714 (2017)
43. Chen, Z., Hu, J., Min, G., Luo, C., El-Ghazawi, T.: Adaptive and efficient resource allocation in cloud datacenters using actor–critic deep reinforcement learning. IEEE Trans. Parallel Distrib. Syst. **33**(8), 1911–1923 (2022). https://doi.org/10.1109/TPDS.2021.3132422
44. Chen, Z., Xiong, B., Chen, X., Min, G., Li, J.: Joint computation offloading and resource allocation in multi-edge smart communities with personalized federated deep reinforcement learning. IEEE Trans. Mob. Comput. (2024). https://doi.org/10.1109/TMC.2024.3396511
45. Chen, Z., Zhang, J., Min, G., Ning, Z., Li, J.: Traffic-aware lightweight hierarchical offloading toward adaptive slicing-enabled SAGIN. IEEE J. Sel. Areas Commun. **42**(12), 3536–3550 (2024). https://doi.org/10.1109/JSAC.2024.3459020

Pablo V. Caderno is a PhD candidate at the CiTIUS Research Center, University of Santiago de Compostela, Spain. His research focuses on Big Data systems and opportunistic computing, with an emphasis on optimizing resource efficiency in distributed environments. He holds a Master's degree in High Performance Computing from the University of Santiago de Compostela, where he investigated running climate models in cloud environments.

Feras M. Awaysheh is an Associate Professor and senior researcher in Federated Learning, Edge Intelligence, and data privacy. He earned his Ph.D. in Big Data and Cloud Computing from the University of Santiago de Compostela, Spain, and holds an Honors MSc. in Information, Computer, and Network Security from NYIT, and a BSc. in Software Engineering from Al Balqa' Applied University. He has held academic positions in Estonia, Australia, and the UK. Dr. Awaysheh is an Associate Editor for Springer's Cluster Computing and the IEEE Transactions on Services Computing, with publications in IEEE Communications Surveys, TII, FGCS, and more. His research focuses on scalable privacy-preserving data analytics, Federated Learning, secure edge AI, and edge-to-cloud continuum.

José Carlos got a BS in Physics at the University of Santiago de Compostela (Spain) in 1989 and obtained a Ph.D. in Physics at the same university in 1994. From 1990 until 1994 he was an associate professor in the Faculty of Computing of the University of A Coruña (Spain). From 1994 he was an associate professor in the area of Computer Architecture in the Department of Electronics and Computing at the University of Santiago de Compostela, until 2022, when he became a full professor. Since 2010 he has been a senior researcher of CiTIUS. His research interests include the development of parallel applications like 3D point cloud processing, prediction and improvement of the performance of parallel systems, development of applications and middleware for cloud, and Big Data technologies.

Tomás F. Pena is a Full Professor at the University of Santiago de Compostela (Spain) and a senior researcher at the Research Center in Intelligent Technologies (CiTIUS) of the same university. His main research interests include high-performance computing, parallel system architectures, and the development of parallel algorithms for clusters and supercomputers. He also works on performance prediction and optimization of parallel applications, middleware and application development for cloud environments, the application of Big Data technologies to scientific computing, and the design of algorithms and libraries for quantum computing.