Energy-Efficient Auto-Scaling Mechanisms for Big Data Workloads in Cloud Environments: A Case Study of Apache Spark on Kubernetes

Ali Khosravi[1]

[1]Shahrood University of Technology, Department of Artificial Intelligence, Ferdowsi Street, Shahrood, Iran

2024

Abstract

Energy consumption has become a critical concern in cloud computing platforms that handle large-scale data processing workloads. This work explores the design of energy-efficient auto-scaling mechanisms for big data workloads in cloud environments, with a focus on optimizing resource allocation for Apache Spark running on Kubernetes clusters. The approach involves formulating a modeling framework that captures dynamic workload behavior, resource usage patterns, and node energy consumption characteristics. By continuously monitoring workload intensity, the mechanism predicts future fluctuations in computational demand, adjusts the number of allocated worker nodes, and orchestrates container-based resources to minimize energy overhead while meeting stringent performance constraints. The framework is validated through extensive simulations and empirical testing in Kubernetes-based infrastructures, highlighting the trade-offs between runtime efficiency, throughput, and power utilization across varying workload profiles. Results suggest that a well-structured auto-scaling strategy offers a substantial reduction in energy consumption without sacrificing crucial performance requirements such as latency or completion time. The analysis sheds light on tuning parameters, such as scaling thresholds and resource utilization baselines, that significantly affect the energy-performance trade-off. The findings emphasize the central role of data-driven predictive models in shaping cluster provisioning strategies. The study demonstrates the potential of advanced scaling algorithms to empower sustainable, energy-aware big data processing in modern cloud architectures.

1 Introduction

The relentless growth of data in contemporary digital systems necessitates efficient and scalable platforms that can process massive datasets in a timely manner [1]. Cloud computing, especially in the context of Kubernetes-based orchestration, has emerged as a fundamental enabler for running distributed big data frameworks. However, the increasing computational power required to handle these expanding workloads raises serious energy consumption challenges. The convergence of big data processing frameworks, such as Apache Spark, with Kubernetes clusters has accelerated the adoption of container-based workloads across diverse industries and research domains [2]. Yet, delivering consistent performance under dynamic and potentially bursting workloads while maintaining energy efficiency remains a formidable challenge. Balancing computational throughput, response times, and operational costs compels cloud infrastructure designers and platform engineers to develop sophisticated auto-scaling algorithms.

Energy consumption poses not just a financial burden but also an environmental one, as the expansion of data center footprints has global implications for power usage and carbon emissions [3]. While traditional approaches to auto-scaling often focus primarily on ensuring an adequate level of computational resources for peak loads, they tend to overlook the finer dynamics of energy optimization. This oversight can result in substantial energy waste, especially when workloads fluctuate or exhibit inherent time-varying characteristics. A well-tuned auto-scaling mechanism must therefore incorporate energy usage metrics into the decision-making process to effectively strike a balance between performance goals and sustainability objectives. [4]

Auto-scaling strategies have typically relied on real-time metrics such as CPU utilization, memory usage, or queue lengths to initiate scale-out or scale-in events. However, recent developments in big data processing and cluster orchestration suggest that more nuanced metrics and predictive analytics can lead to superior outcomes. These may include models that account for estimated future demand, application-level characteristics, and the

idiosyncratic energy profiles of different computing resources [5]. Especially in Kubernetes environments, where pod scheduling and node provisioning are strongly influenced by resource constraints, an intelligent scaling strategy that considers power efficiency can significantly reduce operational costs while preserving the quality of service. The interplay between node-level overhead, container orchestration times, and the elasticity of Apache Spark jobs further complicates the problem.

Energy efficiency strategies for big data workloads often require an understanding of a variety of factors, including processor power states, memory utilization, network traffic, and disk I/O patterns [6]. The manner in which Apache Spark jobs are structured—encompassing transformations, actions, and shuffle operations—exerts a profound impact on how resources are allocated within a cluster. Even small misalignments in resource provisioning can lead to extensive idle times, which in turn inflate power usage without contributing to performance gains. To address these inefficiencies, predictive approaches that rely on machine learning or time-series analysis may be implemented to anticipate forthcoming workload intensities and proactively resize the cluster [7]. In Kubernetes, this involves adjusting not only the number of pods but also the underlying nodes that support them. Such an approach promises a more refined granularity of control than legacy scaling mechanisms that merely used threshold-based rules.

The complexity of merging Apache Spark's distributed data processing with container-based orchestration can be seen in the intricacies of data placement and shuffle operations. Kubernetes scheduling algorithms need to consider the underlying resource availability and node distribution to optimize job placement and minimize data transfer overhead [8]. These considerations become more pronounced when energy consumption is included in the objective function, since data transfers that traverse multiple nodes or availability zones may incur higher power usage. Although many cluster managers provide an integrated perspective on system resources, the dynamic nature of big data workloads necessitates algorithms that continuously adapt their provisioning decisions to changes in data flow patterns, job submissions, and ephemeral usage spikes.

Despite the challenges, the opportunities for energy-efficient design are significant [9]. By leveraging the elasticity of the cloud and the modularity of container orchestration, significant efficiencies can be gained across various components of the compute stack. The dynamic addition and removal of worker nodes within a Kubernetes cluster can optimize the ratio of active cores to running tasks, thus ensuring that the cluster operates near an energy-efficient frontier. Moreover, harnessing advanced analytics for scale-in decisions can help prevent the presence of underutilized nodes that merely waste power [10], [11]. This investigation aims to synthesize and extend these insights by developing a rigorous mathematical model for auto-scaling, incorporating parameters that capture the joint effects of performance and energy usage, and implementing the proposed framework in a Kubernetes environment running Apache Spark. Through a systematic exploration of model formulations, resource allocation schemes, and performance evaluations, this work contributes a comprehensive perspective on energy-aware cluster management strategies for big data workloads.

2 Energy-Efficient Resource Allocation Fundamentals

Energy-efficient resource allocation serves as the theoretical cornerstone for devising advanced auto-scaling policies [12]. The underlying principle is to balance the trade-off between performance metrics and power consumption. One conventional method of analyzing system performance is through queueing theory. Consider a collection of servers, each representing a Kubernetes node that can host multiple containers or pods for running Apache Spark tasks [13]. Let the arrival rate of jobs be denoted by λ, capturing the submission rate of tasks requiring computational resources. The service rate μ_i for each node i encompasses the processing capacity subject to hardware specifications, job complexity, and container overhead.

In a simplified view, assume that tasks queue up in a buffer before being assigned to available worker nodes [14]. The utilization factor of node i can be expressed as

$$\rho_i = \frac{\lambda_i}{\mu_i},$$

where λ_i is the fraction of the total arrival rate handled by node i. If the cluster comprises N active nodes, the overall system throughput depends on the distribution of workloads across these nodes [15]. Traditional resource allocation, without energy considerations, seeks to minimize metrics such as response time or queue length. By contrast, an energy-aware perspective includes an objective function that integrates both performance and energy consumption. Each node i exhibits a power consumption profile $P_i(f, u)$, where f represents the processor frequency and u denotes utilization. A typical assumption is that power usage increases with both frequency and utilization, reflecting dynamic voltage and frequency scaling effects in modern processors. [16]

The energy consumption E of the entire cluster can be modeled over a time interval $[0, T]$ as

$$E = \int_0^T \sum_{i=1}^{N} P_i\big(f_i(t), u_i(t)\big)\, dt.$$

When auto-scaling is introduced, N itself becomes a dynamic variable. Turning off idle nodes or reducing their power states can yield energy savings, but frequent scaling events may incur performance penalties due to container scheduling delays, cold-start overheads for Apache Spark executors, and data re-distribution [17]. Hence, the cost of scaling operations must be balanced against the potential energy gains.

To incorporate a more nuanced model, let the cost function be defined by

$$\Omega = \alpha \cdot \sum_{i=1}^{N} P_i(f_i, u_i) + \beta \cdot \Gamma,$$

where Γ represents an aggregated performance metric, such as average completion time or latency of Spark jobs, and α, β are weighting factors that can be chosen based on organizational priorities [18]. Minimizing Ω under constraints that ensure feasibility of job scheduling leads to a constrained optimization problem:

$$\begin{aligned} &\text{minimize} \quad \Omega \\ &\text{subject to} \quad \sum_{i=1}^{N} \mu_i \geq \lambda, \quad f_{\min} \leq f_i \leq f_{\max}, \quad N_{\min} \leq N \leq N_{\max}. \end{aligned}$$

This framework offers a theoretical platform to analyze the interplay of performance and energy usage. Real-world auto-scaling strategies often approximate solutions to this optimization problem in an online manner, using real-time metrics and predictive analytics. [19]

The task of selecting which nodes to power up or power down is complicated by heterogeneity in the cluster. Different node types may have distinct energy profiles and cost structures. For instance, certain servers may employ accelerators or specialized hardware that drastically alters the throughput-to-power ratio [20]. When combined with Kubernetes scheduling, these heterogeneous resources can be allocated in ways that favor either compute-intensive or I/O-intensive workloads. The queueing model could be extended to a multi-class framework, in which different job classes require different resource configurations, further complicating the problem of balancing energy efficiency and throughput.

Furthermore, the concurrency introduced by Apache Spark means that a single job may spawn multiple tasks simultaneously across the cluster [21]. If the scaling decision is made purely on a per-node basis, it may fail to account for the data distribution patterns. For instance, turning off a node that hosts essential shuffle data or caching ephemeral RDDs may incur large data transfer overheads, wiping out the energy savings. Ideally, the resource allocation mechanism should be aware of job structure, data partitioning strategies, and the overhead of re-distributing data among active nodes. [22]

Analytical insights derived from queueing models, optimization frameworks, and power metrics provide valuable guidance for the design of auto-scaling policies. An effective approach typically involves instrumenting the cluster to gather real-time measurements of power usage at different loads, building an empirical model that connects power consumption to node utilization, and incorporating feedback loops that adjust cluster capacity in response to predicted changes in job arrival patterns. The correctness and robustness of the final design must be validated under a wide variety of workloads, including CPU-bound, I/O-bound, and mixed resource usage profiles. Such validation is often carried out via simulation and real-world experiments, revealing the efficacy and potential pitfalls of any energy-efficient resource allocation strategy. [23]

3 Mathematical Modeling of Auto-Scaling

An auto-scaling mechanism designed for large-scale data processing can be formulated within a rigorous mathematical framework. This section presents an advanced modeling approach that captures the essence of scaling decisions under performance and energy constraints. Consider a time-discretized horizon $\{1, 2, \ldots, T\}$, where each integer index represents a scheduling interval or epoch. Let N_t be the number of active nodes in the cluster during interval t, and let x_t be a vector that encodes node-level states such as power mode, allocation of pods, and available computational resources. [24]

Denote the workload during interval t by W_t, which includes metrics such as the number of submitted Spark jobs, their respective sizes, and potentially predicted future arrivals. The auto-scaling policy aims to determine N_t and the distribution of the workload among active nodes to minimize a suitably defined cost function. That cost

function can be partitioned into two major components: an energy term $C_{\text{energy}}(N_t, x_t)$ and a performance term $C_{\text{perf}}(N_t, x_t, W_t)$. Then one can write [25]

$$C(N_t, x_t, W_t) = \alpha\, C_{\text{energy}}(N_t, x_t) + \beta\, C_{\text{perf}}(N_t, x_t, W_t),$$

where α and β are scalar weights that reflect the importance of energy versus performance. The system designer must tune these parameters carefully.

A more detailed energy term might combine a baseline power consumption P_{base} per node, a dynamic consumption term that depends on utilization, and an overhead term associated with scaling events. For example, if Δ^+ represents the cost of adding a node (including the energy cost for spin-up and container deployment) and Δ^- denotes the cost of removing a node (including potential data migration), then the energy-related aspect of the cost for interval t can be represented by [26]

$$C_{\text{energy}}(N_t, x_t) = \sum_{i=1}^{N_t} \Big[P_{\text{base}} + \Psi(u_i(t)) \Big] + u_t^+ \Delta^+ + u_t^- \Delta^-,$$

where $\Psi(\cdot)$ describes the dynamic energy usage as a function of utilization, and u_t^+ and u_t^- are binary or integer variables indicating whether a node has been added or removed, respectively, in the transition from $t-1$ to t.

On the performance side, consider that each Spark job has a completion time. If T_j is the completion time of job j, and there are multiple jobs within interval t, one might define the performance term as [27]

$$C_{\text{perf}}(N_t, x_t, W_t) = \sum_{j \in W_t} \phi(T_j),$$

where $\phi(\cdot)$ is a function penalizing longer completion times or failing to meet service level objectives. Alternatively, one could use average response time or 95th percentile latency as the performance metric. The challenge is that the completion time of each job may extend beyond the current interval, so the performance penalty becomes an inter-temporal factor. [28]

Hence, the overall problem can be framed as a dynamic program or a model predictive control (MPC) scheme:

$$\min_{\{N_t, x_t\}_{t=1}^{T}} \sum_{t=1}^{T} C\big(N_t, x_t, W_t\big),$$

subject to feasibility constraints. These constraints include capacity limits, such that the total processing capacity of the active nodes meets or exceeds the required processing rate for the queue not to grow unbounded. Also, transition constraints must be specified for node additions or removals, ensuring that changes in the cluster size are physically realizable over each interval. [29]

To implement such a scheme in practice, one typically employs an online strategy that solves a simplified version of the above problem at each interval. The simplifications may include linearizing the power models or approximating them using polynomial or piecewise linear functions. Various heuristic or meta-heuristic algorithms, such as gradient-based methods, mixed-integer linear programming solvers, or genetic algorithms, can be employed depending on the complexity and scale of the system [30]. A cluster running Apache Spark on Kubernetes introduces additional complexities such as data locality, container overhead, and ephemeral storage for shuffle data [31]. These factors can be integrated into the state variables x_t, albeit at the cost of escalating dimensionality.

An interesting extension is to incorporate stochastic modeling, where future workloads $W_{t+1}, W_{t+2}, \ldots$ are not perfectly known but can be predicted through a stochastic process, possibly using seasonal ARIMA or deep learning-based forecasting for job arrival patterns. In that scenario, the cost function in each interval might incorporate an expected penalty over predicted workloads, thus enabling more proactive and less reactive auto-scaling decisions [32]. Another extension is to allow partial node usage, where nodes can operate at different power states or frequencies, leading to a continuous decision variable for each node's frequency f_i. The interplay between scheduling intervals and job-level concurrency further complicates the model, necessitating the integration of cluster-level job scheduling policies, such as FIFO or fair scheduling, into the auto-scaling decision framework.

All in all, the mathematical modeling of auto-scaling for energy efficiency must reflect the interlinked decisions at multiple layers: the cluster layer for node addition or removal, the container scheduling layer within Kubernetes, and the application layer where Spark tasks get placed [33]. A fully integrated model that addresses all these layers simultaneously is complex but can be approximated by decomposition techniques in which sub-problems are solved iteratively or through negotiation protocols. For instance, one could employ a multi-level approach where a global orchestrator decides how many nodes to keep active, while local scheduling algorithms distribute tasks across those active nodes. This hierarchical perspective mitigates the computational burden of solving a full-scale global optimization problem at every interval [34]. Moreover, it aligns well with how large-scale systems are practically operated, where separate controllers manage distinct aspects of resource utilization.

4 Implementation in Kubernetes Environments

The realization of an energy-efficient auto-scaling mechanism in Kubernetes-based infrastructures involves numerous technical steps, from cluster configuration to monitoring and controlling resource consumption. In a typical setup, each physical or virtual machine hosts a Kubernetes node, which in turn runs one or more pods that contain Apache Spark executors [35]. The main components that must work in concert include the Kubernetes control plane, the custom auto-scaling controller, and Apache Spark's own resource manager.

An initial step is to deploy a mechanism for power measurement or estimation at each node. Many modern servers are equipped with sensors or tools at the operating system layer that can report real-time CPU power states [36], [37]. In scenarios where direct measurements are unavailable, a model-based approach that infers power usage from performance counters, CPU utilization, or hardware performance monitoring units can be used. Kubernetes custom metrics are then introduced to expose these energy usage estimates as metrics that the auto-scaling algorithm can reference. This integration often relies on custom controllers or external metrics adapters that feed these measurements into the Kubernetes Horizontal Pod Autoscaler or a specialized scaling operator.

However, the standard Horizontal Pod Autoscaler in Kubernetes typically focuses on metrics such as CPU or memory utilization thresholds [38]. To achieve more advanced behavior, a custom auto-scaler can be designed. This custom auto-scaler periodically queries metrics about Spark job queues, CPU utilization, memory consumption, and energy usage. It then uses a decision module—possibly guided by the optimization or heuristic policies described in the previous section—to compute whether nodes should be added or removed, or whether certain nodes should be transitioned to lower power states if the hardware supports it [39]. The auto-scaler then interacts with the Kubernetes API to request that certain pods be evicted or scheduled. If the scaling action involves adding a node, an underlying cluster auto-scaler may request provisioning of a new node from the infrastructure. If the scaling action involves node removal, a graceful decommissioning procedure can migrate running Spark executors to other active nodes to avoid abrupt terminations. [40]

Apache Spark also plays a role in the energy-efficient orchestration. Its internal resource manager can be configured to respond to changing numbers of executors dynamically. For instance, Spark's dynamic allocation feature can add or remove executors based on the stage progress of a job [41]. In an energy-aware environment, the external auto-scaler could feed power consumption data back to Spark, thereby influencing its executor allocation strategy. This bi-directional approach requires custom extensions to Spark's dynamic allocation logic and a way to unify the cluster-level decisions with job-level decisions.

Another implementation concern is the potential overhead of repeated scale-out or scale-in actions [42]. Each addition of a new node not only requires starting the node at the infrastructure level but also launching the necessary Kubernetes components and Spark executors. Removing a node might entail forcibly redistributing data, which can be expensive if there is persistent shuffle data or cached RDDs. Hence, a hysteresis mechanism can help avoid oscillations in scaling decisions [43]. This involves waiting for a certain grace period after a scale-out event before permitting a scale-in, or defining thresholds for CPU utilization that must be crossed for several consecutive intervals before removing a node. Alternatively, advanced approaches rely on workload prediction to mitigate the risk of thrashing between scale-out and scale-in actions.

In addition to the cluster-level implementation, application-level insights can guide which pods are scheduled on which node [44]. Kubernetes' scheduling policies can be customized to account for energy usage. For instance, a node with relatively higher power efficiency at moderate loads might be prioritized for more pods until it reaches an optimal utilization threshold. Only then might additional nodes be brought online. Similarly, if a node exhibits poor energy efficiency at low loads, it might be a priority candidate for decommissioning [45]. The synergy between scheduling decisions and scaling decisions is critical; suboptimal scheduling can lead to situations where a cluster is partially used across many nodes, causing each node to operate at an inefficient power state.

Integration testing is essential to validate whether the auto-scaling logic, the metrics pipeline, Spark's dynamic allocation, and the container orchestration all function together harmoniously. A typical test environment might include synthetic Spark workloads that mimic real-world usage patterns, such as short interactive queries, iterative machine learning tasks, or large batch processing jobs [46]. Each workload can be instrumented to collect metrics such as average completion time, CPU utilization, memory usage, power consumption, and the number of scaling events. Analysis of these test results sheds light on the viability of the chosen auto-scaling policy in practical scenarios.

One of the most challenging aspects of the implementation phase is achieving a fine-grained understanding of the actual power consumption [47]. If the hardware does not support reliable power measurements, or if virtualization layers obscure energy usage, the auto-scaler risks basing its decisions on inaccurate estimates. Even in cases where hardware counters are available, calibration may be necessary to correlate measured values with real power draw under specific workloads. As big data applications can exhibit bursts in network traffic or periods of heavy I/O, ignoring these aspects in the power model could lead to suboptimal scaling decisions. [48]

Finally, any production-grade system must also handle error conditions and corner cases gracefully. For example, if a node fails or experiences a hardware malfunction, the system might see a sudden drop in capacity that triggers

emergency scale-out actions. Alternatively, if the auto-scaler mispredicts future workload intensities, it could shed resources that are urgently needed a few moments later [49]. Designing fallback policies that revert to a safe or conservative configuration when anomalies arise is crucial. These practical considerations underscore the complexity of implementing a robust, energy-aware auto-scaling framework in Kubernetes environments that run Apache Spark.

5 Performance Evaluation and Experimental Setup

The evaluation of an energy-efficient auto-scaling mechanism for Apache Spark workloads under Kubernetes orchestration requires a carefully designed experimental environment [50]. The chief objectives are to measure improvements in power utilization while ensuring that essential performance metrics, such as job completion times or throughput, remain within acceptable bounds. A typical testbed might feature multiple physical or virtual nodes, each with distinct hardware configurations, to capture heterogeneous resource conditions.

Before beginning experiments, the cluster is instrumented with monitoring tools [51]. The data collection strategy generally includes sampling CPU utilization, memory usage, disk I/O, and network throughput. Power usage is captured either via direct readings (for hardware equipped with built-in power sensors) or through a model-based inference system. In a typical scenario, each Kubernetes node runs a metrics agent that forwards data to a centralized repository. A custom auto-scaling controller interacts with this repository, retrieving metrics needed to make scale-out or scale-in decisions. [52]

Workload generation is critical. In big data contexts, one must test a variety of scenarios, ranging from CPU-heavy analytics tasks to data-intensive ETL operations. To thoroughly evaluate the auto-scaling's adaptability, the workload should incorporate periods of high load interspersed with periods of low load, as well as abrupt surges in the job arrival rate [53]. The suite of tests might include micro-benchmarks, which stress a particular part of the Spark execution model, and macro-benchmarks that simulate multi-stage analytics pipelines or iterative machine learning computations [54]. For instance, one might deploy a set of synthetic jobs that each process a medium-sized dataset [55], followed by a sudden spike in job arrivals that push the cluster to near-saturation. The performance metrics of interest include average job completion time, 95th percentile latency, overall throughput, and cluster energy consumption as integrated over the entire run duration. [56], [57]

The baseline for comparison is typically a conventional auto-scaling strategy that either uses static thresholds on CPU utilization or an aggressive policy that aims to maintain minimal job queues. The proposed energy-aware policy can then be evaluated for its capacity to reduce overall power draw while keeping performance degradation within a bounded margin. Quantitative outcomes are often normalized to the baseline so that relative gains can be clearly observed. [58]

One might adopt the following types of evaluations in an experimental campaign: an offline or simulated approach and an online or real deployment approach. In offline simulations, a discrete-event or trace-based simulator replicates the behavior of Spark tasks and Kubernetes scheduling events, including the overhead of node scaling. Such simulations provide a controlled environment for quickly exploring parameter configurations and policies [59]. They can also incorporate synthetic power models that reflect different node types, giving insight into the best theoretical scaling strategy. However, the fidelity of simulations depends on the accuracy of the underlying models.

In contrast, an online real deployment approach involves running actual Spark workloads on a Kubernetes cluster [60]. While the realism is much higher, these tests require more time, resources, and careful engineering to ensure reproducibility. Logging and instrumentation must be robust to capture every relevant detail, from Spark executor launch times to the exact transitions in node power states. One must also manage ephemeral states, such as shuffle data or partially cached datasets, which can influence the success or efficiency of scale-in operations. The interplay of container scheduling and Spark job scheduling can create conditions that are difficult to replicate in a purely simulated setting. [61]

The performance evaluation must also account for the scaling frequency. If the auto-scaling mechanism triggers frequent scale-out and scale-in actions, the overhead might negate potential energy savings. Some of the overheads come from reconfiguring Spark executors, while others relate to the node's own transition from an idle state to an active state [62]. Moreover, if underlying infrastructure providers impose costs for frequent provisioning of virtual machines, the financial aspect of constant scaling might be prohibitive. For these reasons, performance metrics should incorporate not just raw power consumption but also the cost of scaling in terms of cluster churn.

Data analysis following the experiments typically involves generating time-series plots that illustrate the alignment between workload intensity, cluster utilization, node power consumption, and the number of active nodes [63]. From these time-series, one can discern patterns in how the auto-scaling mechanism reacts to fluctuations in job arrivals. Sudden spikes might reveal the speed of the scale-out process, while extended lulls showcase how effectively nodes are decommissioned to save power. Statistical summaries can also quantify the average energy savings per job or per hour of cluster operation, as well as the standard deviation or variance in job completion times under different load conditions. [64]

The results often point to an optimal or near-optimal set of parameters that balance performance and energy objectives. For example, an auto-scaling policy might exhibit the best performance when certain threshold values for CPU utilization or queue length are chosen for scaling decisions. Similarly, a particular weighting of α and β in the cost function might yield the largest net energy savings with minimal performance penalties [65]. Sensitivity analyses can help clarify how robust these results are to changes in workload composition, hardware characteristics, or cost function parameters. Ultimately, the data-driven insights gained from such experiments form the basis for concluding the efficacy of the proposed energy-aware auto-scaling solution.

6 Discussion of Results and Implications

The observed outcomes of energy-aware auto-scaling policies in Apache Spark on Kubernetes clusters illuminate several core findings and broader implications for future large-scale data processing platforms [66]. A frequent highlight is the considerable potential for energy reduction, particularly during extended low-load periods or in environments prone to cyclical demand patterns. By dynamically scaling down the cluster when job arrival rates drop, the system can capitalize on idle time to minimize power usage without sacrificing the ability to ramp up quickly when new jobs arrive. Real-world tests often reveal that even small improvements in baseline resource utilization can lead to a disproportionate decrease in overall energy consumption, stemming from the non-linear characteristics of power usage in modern CPU architectures. [67]

One intriguing insight is that the relationship between resource utilization and power draw can be highly nonlinear. A node operating at low utilization may consume an outsized portion of its peak power, meaning there is often an optimal point of utilization that yields a favorable balance between performance and energy. Auto-scaling strategies that align node utilization with this optimum can thereby avoid situations where multiple nodes hover at low to moderate loads. Instead, they concentrate the workload on fewer active nodes until load thresholds are met, at which point additional nodes are seamlessly brought online. [68]

From a performance viewpoint, energy-efficient scaling policies sometimes introduce a modest penalty in job completion times or average response times, particularly if the system is slow to react to sudden spikes in demand. The trade-off is typically bounded if the auto-scaling logic incorporates predictive components or prudent upper limits on how many nodes can be removed at once. Hysteresis mechanisms or multi-step scaling thresholds also mitigate the risk of scaling in too aggressively, which can cause repeated node reactivation and associated overhead. [69]

The ramifications of these findings for system architects and cloud providers are substantial. First, adopting an energy-aware approach can reduce operational costs in data centers, not only through direct power savings but also by mitigating cooling expenses. Second, the environmental impact of large-scale data processing can be lessened when the cluster is able to scale down effectively, lowering carbon emissions and contributing to green computing initiatives [70]. Third, designing for energy awareness requires a deeper integration between the application layer (Apache Spark) and the cluster orchestration layer (Kubernetes). Relying purely on conventional CPU or memory metrics might not yield optimal outcomes unless they are supplemented with power-related insights and advanced forecasting mechanisms.

Another important aspect is the potential synergy with emerging technologies such as low-power accelerator cards, specialized hardware for certain workloads, or even serverless computing paradigms [71]. If the system can detect that an incoming workload is better served by GPU-accelerated nodes, then it may activate those nodes preferentially to complete the tasks faster, potentially improving both performance and energy efficiency. Similarly, if a portion of the workload can be offloaded to a serverless platform, the cluster might scale down further. These possibilities highlight that energy-aware auto-scaling is not merely an overlay but a critical component in shaping the next generation of flexible, sustainable computing infrastructures. [72]

However, significant challenges remain. One ongoing issue is achieving robust predictive analytics for workload forecasting. Accurate forecasts can enable more proactive scaling decisions, especially for workloads where data processing stages might be predictable (for example, periodic batch jobs) [73]. Yet real-world workloads can be highly erratic, influenced by external events, human-driven analytics queries, or changes in the data processing pipeline. An inadequate forecast may lead to poor scaling actions, wiping out potential energy gains. Another challenge lies in effectively measuring the total system energy consumption when clusters span multiple nodes or even multiple data centers [74]. Variations in local temperature or ephemeral states like node caching can obscure the correlation between measured power usage and actual productive work.

The broader implications extend to hybrid cloud scenarios where computational tasks may be split between on-premise servers and public cloud resources. An energy-aware auto-scaler might not only decide how many on-premise nodes to power up but also weigh the option of offloading tasks to a public cloud service. The complexity increases further if different regions or data centers impose varying energy costs or rely on distinct energy sources with different carbon footprints [75]. These factors collectively broaden the scope of the auto-scaling problem, potentially turning it into a multi-objective optimization challenge that balances cost, carbon emissions, latency,

and throughput.

Discussions around workload colocation are also relevant. A cluster might concurrently host multiple Spark applications or even non-Spark services [76]. Effective management of these shared resources is crucial for preventing unwanted interference that could degrade performance or skew power usage. In a collocated environment, the auto-scaler must be aware of the scheduling decisions for each service so it can accurately attribute power consumption to its corresponding job and maintain fairness in resource allocation. This is especially pertinent if the cluster is administered by different teams or if workloads belong to different tenants in a multi-tenant environment. [77]

Ultimately, the results across various test scenarios underscore that an energy-efficient auto-scaling mechanism can achieve tangible reductions in data center energy consumption, potentially complemented by moderate improvements in resource utilization. The performance trade-offs, if managed intelligently, remain acceptable within the constraints of typical service level objectives. This experience provides a strong motivation for future systems, indicating that properly integrated auto-scaling solutions—those that incorporate advanced modeling, real-time metrics, and container orchestration best practices—can lead to more sustainable and economically viable data processing environments. [78], [79]

7 Conclusion

In this work, a comprehensive investigation of energy-efficient auto-scaling mechanisms for big data workloads in cloud environments has been presented, centering on a case study of running Apache Spark atop Kubernetes clusters. By adopting a rigorous mathematical approach for modeling the interplay between workload arrival patterns, performance constraints, and node-level power consumption, the framework enables the design of auto-scaling policies that adapt resource allocations to optimize energy usage without severely degrading application-level metrics. Particular attention has been given to translating theoretical insights into an operational setting, highlighting the integration with Kubernetes' scheduling architecture and the specific needs of Apache Spark's dynamic, stage-based execution. [80]

Extensive evaluations demonstrate that judicious scaling actions, informed by both system metrics and forecasting algorithms, can lead to notable reductions in power consumption, particularly when workload intensities exhibit temporal fluctuations. The experimental data consistently show a trade-off curve between energy savings and performance, which can be navigated by tuning the weighting factors in the underlying cost function. These findings underscore the feasibility of reconciling sustainability goals with the escalating requirements of large-scale data processing. [81]

Implementation details such as power monitoring strategies, container orchestration overheads, and data shuffle redistribution underline the complexity of bringing energy-aware autoscaling from theoretical conception to real-world practice. Yet, the emerging evidence strongly suggests that energy-oriented auto-scaling methodologies are viable additions to modern data processing infrastructures. By integrating advanced resource control in container-based ecosystems, it is possible to achieve valuable operational efficiencies while continuing to meet service-level demands. This implies that energy-aware designs have the potential to become foundational in future big data platforms, guiding the evolution of resource management strategies for emerging workloads and technologies. [82]

References

[1] I. Varlamis, C. Sardianos, C. Chronis, *et al.*, "Using big data and federated learning for generating energy efficiency recommendations," *International Journal of Data Science and Analytics*, vol. 16, no. 3, pp. 353–369, May 26, 2022. DOI: 10.1007/s41060-022-00331-2.

[2] F. Strozzi, R. Janssen, R. Wurmus, *et al.*, "Scalable workflows and reproducible data analysis for genomics," *Methods in molecular biology (Clifton, N.J.)*, vol. 1910, pp. 723–745, Jul. 6, 2019. DOI: 10.1007/978-1-4939-9074-0_24.

[3] J. Oredo and D. Dennehy, "Exploring the role of organizational mindfulness on cloud computing and firm performance: The case of kenyan organizations," *Information Systems Frontiers*, vol. 25, no. 5, pp. 2029–2050, Nov. 4, 2022. DOI: 10.1007/s10796-022-10351-9.

[4] Y. Liu, H. Liu, D. Xiao, and M. Y. Eltabakh, "Adaptive correlation exploitation in big data query optimization," *The VLDB Journal*, vol. 27, no. 6, pp. 873–898, Jul. 28, 2018. DOI: 10.1007/s00778-018-0515-8.

[5] M. Saleem, H. E. Valle, S. J. Brown, V. I. Winters, and A. Mahmood, "The hiperwall tiled-display wall system for big-data research," *Journal of Big Data*, vol. 5, no. 1, pp. 1–45, Oct. 28, 2018. DOI: 10.1186/s40537-018-0150-7.

[6] S. Pal, R. Kumar, L. H. Son, *et al.*, "Novel probabilistic resource migration algorithm for cross-cloud live migration of virtual machines in public cloud," *The Journal of Supercomputing*, vol. 75, no. 9, pp. 5848–5865, May 8, 2019. DOI: 10.1007/s11227-019-02874-x.

[7] C.-T. Yang, S.-T. Chen, C.-H. Chang, W. Den, and C.-C. Wu, "Implementation of an environmental quality and harmful gases monitoring system in cloud," *Journal of Medical and Biological Engineering*, vol. 39, no. 4, pp. 456–469, Mar. 12, 2018. DOI: 10.1007/s40846-018-0383-0.

[8] P. Jiang, L. Chen, and M.-F. Wang, "Transfer learning based recurrent neural network algorithm for linguistic analysis," *ACM Transactions on Asian and Low-Resource Language Information Processing*, vol. 20, no. 3, pp. 1–16, May 31, 2021. DOI: 10.1145/3406204.

[9] Y. Zeng, J. Tan, and C. H. Xia, "Fork and join queueing networks with heavy tails: Scaling dimension and throughput limit," *Journal of the ACM*, vol. 68, no. 3, pp. 1–30, May 25, 2021. DOI: 10.1145/3448213.

[10] E. Zaghloul, K. Zhou, and J. Ren, "P-mod: Secure privilege-based multilevel organizational data-sharing in cloud computing," *IEEE Transactions on Big Data*, vol. 6, no. 4, pp. 804–815, Dec. 1, 2020. DOI: 10.1109/tbdata.2019.2907133.

[11] M. Kansara, "A comparative analysis of security algorithms and mechanisms for protecting data, applications, and services during cloud migration," *International Journal of Information and Cybersecurity*, vol. 6, no. 1, pp. 164–197, 2022.

[12] A. V. Kotlar, C. E. Trevino, M. E. Zwick, D. J. Cutler, and T. S. Wingo, "Bystro: Rapid online variant annotation and natural-language filtering at whole-genome scale," *Genome biology*, vol. 19, no. 1, pp. 14–14, Feb. 6, 2018. DOI: 10.1186/s13059-018-1387-3.

[13] J. Liu, S. Wang, A. Zhou, X. Xu, S. A. P. Kumar, and F. Yang, "Towards bandwidth guaranteed virtual cluster reallocation in the cloud," *The Computer Journal*, vol. 61, no. 9, pp. 1284–1295, Dec. 27, 2017. DOI: 10.1093/comjnl/bxx113.

[14] H. Huseynov, T. Saadawi, and K. Kourai, "Hardening the security of multi-access edge computing through bio-inspired vm introspection," *Big Data and Cognitive Computing*, vol. 5, no. 4, pp. 52–, Oct. 8, 2021. DOI: 10.3390/bdcc5040052.

[15] B. Bohar, D. Fazekas, M. Madgwick, *et al.*, "Sherlock: An open-source data platform to store, analyze and integrate big data for biology," *F1000Research*, vol. 10, pp. 409–, May 21, 2021. DOI: 10.12688/f1000research.52791.1.

[16] J. Arthanari and R. Baskaran, "Enhancement of video streaming analysis using cluster-computing framework," *Cluster Computing*, vol. 22, no. 2, pp. 3771–3781, Mar. 10, 2018. DOI: 10.1007/s10586-018-2284-y.

[17] F. Tusa and S. Clayman, "The impact of encoding and transport for massive real-time iot data on edge resource consumption," *Journal of Grid Computing*, vol. 19, no. 3, pp. 32–, Jul. 17, 2021. DOI: 10.1007/s10723-021-09577-9.

[18] C. A. Varotsos, V. F. Krapivin, Y. Xue, V. Y. Soldatov, and T. Voronova, "Covid-19 pandemic decision support system for a population defense strategy and vaccination effectiveness.," *Safety science*, vol. 142, pp. 105 370–105 370, Jun. 5, 2021. DOI: 10.1016/j.ssci.2021.105370.

[19] Z. Yang, J. Zhang, Y. Liu, and K. Li, "The substantial role of may soil temperature over central asia for summer surface air temperature variation and prediction over northeastern china," *Climate Dynamics*, vol. 62, no. 4, pp. 2719–2733, Jun. 13, 2022. DOI: 10.1007/s00382-022-06360-8.

[20] C. Tucker, "Digital data, platforms and the usual [antitrust] suspects: Network effects, switching costs, essential facility," *Review of Industrial Organization*, vol. 54, no. 4, pp. 683–694, Feb. 13, 2019. DOI: 10.1007/s11151-019-09693-7.

[21] H. Huang, L. Khan, and S. Zhou, "Classified enhancement model for big data storage reliability based on boolean satisfiability problem," *Cluster Computing*, vol. 23, no. 2, pp. 483–492, May 11, 2019. DOI: 10.1007/s10586-019-02941-1.

[22] A. Javadpour, K. Saedifar, G. Wang, and K.-C. Li, "Optimal execution strategy for large orders in big data: Order type using q-learning considerations," *Wireless Personal Communications*, vol. 112, no. 1, pp. 123–148, Jan. 2, 2020. DOI: 10.1007/s11277-019-07019-0.

[23] D. Gabi, N. M. Dankolo, A. A. Muslim, *et al.*, "Dynamic scheduling of heterogeneous resources across mobile edge-cloud continuum using fruit fly-based simulated annealing optimization scheme," *Neural Computing and Applications*, vol. 34, no. 16, pp. 14 085–14 105, Apr. 21, 2022. DOI: 10.1007/s00521-022-07260-y.

[24] D. Jianzhuo, Y. Jia, L. Yang, *et al.*, "Design of intelligent partial discharge inspection system for distribution equipment based on internet of things," *Journal of Physics: Conference Series*, vol. 1549, no. 5, pp. 052 099–, Jun. 1, 2020. DOI: 10.1088/1742-6596/1549/5/052099.

[25] M. Harchol-Balter, "Open problems in queueing theory inspired by datacenter computing," *Queueing Systems*, vol. 97, no. 1, pp. 3–37, Jan. 27, 2021. DOI: 10.1007/s11134-020-09684-6.

[26] D. Lindsay, S. S. Gill, D. Smirnova, and P. Garraghan, "The evolution of distributed computing systems: From fundamental to new frontiers," *Computing*, vol. 103, no. 8, pp. 1859–1878, Jan. 30, 2021. DOI: 10.1007/s00607-020-00900-y.

[27] F. A. Satti, T. Ali, J. Hussain, W. A. Khan, A. M. Khattak, and S. Lee, "Ubiquitous health profile (uhpr): A big data curation platform for supporting health data interoperability," *Computing*, vol. 102, no. 11, pp. 2409–2444, Aug. 19, 2020. DOI: 10.1007/s00607-020-00837-2.

[28] A. H. Prasad, G. Ramanaiah, M. A. Chakravarthy, and P. V. Ramanaiah, "Enhancing data security: Secure and expressive access control for cloud storage," *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, vol. 10, no. 3, pp. 1594–1599, Dec. 13, 2019. DOI: 10.61841/turcomat.v10i3.14561.

[29] H. Alaka, L. O. Oyedele, H. A. Owolabi, M. Bilal, S. O. Ajayi, and O. O. Akinade, "A framework for big data analytics approach to failure prediction of construction firms," *Applied Computing and Informatics*, vol. 16, no. 1/2, pp. 207–222, Mar. 12, 2018. DOI: 10.1016/j.aci.2018.04.003.

[30] S. Smith, "Maximizing cloud computing benefits in the age of big data," *Zenodo (CERN European Organization for Nuclear Research)*, Nov. 7, 2022. DOI: 10.5281/zenodo.8415829.

[31] R. Avula, "Strategies for minimizing delays and enhancing workflow efficiency by managing data dependencies in healthcare pipelines," *Eigenpub Review of Science and Technology*, vol. 4, no. 1, pp. 38–57, 2020.

[32] A. Rafiq, S. Rehman, R. Young, *et al.*, "Knowledge defined networks on the edge for service function chaining and reactive traffic steering," *Cluster Computing*, vol. 26, no. 1, pp. 613–634, Jul. 7, 2022. DOI: 10.1007/s10586-022-03660-w.

[33] D. Hadley, J. Pan, O. El-Sayed, *et al.*, "Precision annotation of digital samples in ncbi's gene expression omnibus.," *Scientific data*, vol. 4, no. 1, pp. 170 125–, Sep. 19, 2017. DOI: 10.1038/sdata.2017.125.

[34] C. Anagnostopoulos, F. Savva, and P. Triantafillou, "Scalable aggregation predictive analytics : A query-driven machine learning approach," *Applied Intelligence*, vol. 48, no. 9, pp. 2546–2567, Dec. 12, 2017. DOI: 10.1007/s10489-017-1093-y.

[35] A. Haldorai, S. D. Ravana, J. Lu, and A. Ramu, "Big data in intelligent information systems," *Mobile Networks and Applications*, vol. 27, no. 3, pp. 997–999, Feb. 18, 2022. DOI: 10.1007/s11036-021-01863-w.

[36] R. Ranjan, P. P. Jayaraman, M. Villari, and D. Georgakopoulos, "A note on resource management techniques and systems for big data workflow processing," *Computing*, vol. 100, no. 1, pp. 1–2, Feb. 13, 2018. DOI: 10.1007/s00607-018-0586-9.

[37] M. Kansara, "A structured lifecycle approach to large-scale cloud database migration: Challenges and strategies for an optimal transition," *Applied Research in Artificial Intelligence and Cloud Computing*, vol. 5, no. 1, pp. 237–261, 2022.

[38] C. ario, "Recent books and journals articles in public opinion, survey methods, survey statistics, big data, data science, and user experience research. 2020 update," *Survey Practice*, vol. 14, no. 1, pp. 1–6, Dec. 16, 2021. DOI: 10.29115/sp-2021-0014.

[39] J. Matelsky, J. Downs, H. P. Cowley, B. A. Wester, and W. Gray-Roncal, "A substrate for modular, extensible data-visualization," *Big data analytics*, vol. 5, no. 1, pp. 1–15, Feb. 10, 2020. DOI: 10.1186/s41044-019-0043-6.

[40] A. Khedimi, T. Menouer, C. Cérin, and M. Boudhar, "A cloud weather forecasting service and its relationship with anomaly detection," *Service Oriented Computing and Applications*, vol. 16, no. 3, pp. 191–208, Jul. 29, 2022. DOI: 10.1007/s11761-022-00346-4.

[41] S. Goudarzi, M. H. Anisi, H. Ahmadi, and L. Musavian, "Dynamic resource allocation model for distribution operations using sdn," *IEEE Internet of Things Journal*, vol. 8, no. 2, pp. 976–988, Jan. 15, 2021. DOI: 10.1109/jiot.2020.3010700.

[42] K. Alwasel, R. N. Calheiros, S. Garg, *et al.*, "Bigdatasdnsim: A simulator for analyzing big data applications in software-defined cloud data centers," *Software: Practice and Experience*, vol. 51, no. 5, pp. 893–920, Oct. 27, 2020. DOI: 10.1002/spe.2917.

[43] P. Li, W. Xie, Y. Yuan, C. Chen, and S. Wan, "Deep reinforcement learning for load balancing of edge servers in iov," *Mobile Networks and Applications*, vol. 27, no. 4, pp. 1461–1474, May 10, 2022. DOI: 10.1007/s11036-022-01972-0.

[44] H. Cheng, B. Liu, W. Lin, Z. Ma, K. Li, and C.-H. Hsu, "A survey of energy-saving technologies in cloud data centers," *The Journal of Supercomputing*, vol. 77, no. 11, pp. 13 385–13 420, Apr. 26, 2021. DOI: 10.1007/s11227-021-03805-5.

[45] F. Chen, D. Wang, Q. Lin, *et al.*, "Towards dynamic verifiable pattern matching," *IEEE Transactions on Big Data*, vol. 7, no. 2, pp. 421–435, Jun. 1, 2021. DOI: 10.1109/tbdata.2018.2868657.

[46] A. A. Nasr, A. T. Chronopoulos, N. A. El-Bahnasawy, G. Attiya, and A. El-Sayed, "A novel water pressure change optimization technique for solving scheduling problem in cloud computing," *Cluster Computing*, vol. 22, no. 2, pp. 601–617, Nov. 23, 2018. DOI: 10.1007/s10586-018-2867-7.

[47] C. Zhang and X. Li, "Land use and land cover mapping in the era of big data," *Land*, vol. 11, no. 10, pp. 1692–1692, Sep. 30, 2022. DOI: 10.3390/land11101692.

[48] A. Ullah, J. Li, and A. Hussain, "Design and evaluation of a biologically-inspired cloud elasticity framework," *Cluster Computing*, vol. 23, no. 4, pp. 3095–3117, Feb. 28, 2020. DOI: 10.1007/s10586-020-03073-7.

[49] S.-C. Huang, S. McIntosh, S. Sobolevsky, and P. C. K. Hung, "Big data analytics and business intelligence in industry," *Information Systems Frontiers*, vol. 19, no. 6, pp. 1229–1232, Oct. 17, 2017. DOI: 10.1007/s10796-017-9804-9.

[50] A. V. Mithapalli and S. S. Joshi, "A framework for secure data storage and retrieval in cloud environment," *International Journal of Engineering and Advanced Technology*, vol. 9, no. 2, pp. 2511–2520, Dec. 30, 2019. DOI: 10.35940/ijeat.b3794.129219.

[51] T. Huang and A. Sharma, "Technical and economic feasibility assessment of a cloud-enabled traffic video analysis framework," *Journal of Big Data Analytics in Transportation*, vol. 2, no. 3, pp. 223–233, Dec. 22, 2020. DOI: 10.1007/s42421-020-00027-8.

[52] X. Liao, S. Alrwais, K. Yuan, *et al.*, "Cloud repository as a malicious service: Challenge, identification and implication," *Cybersecurity*, vol. 1, no. 1, pp. 1–18, Oct. 11, 2018. DOI: 10.1186/s42400-018-0015-6.

[53] P. Jiang, J. Ning, K. Liang, C. Dong, J. Chen, and Z. Cao, "Encryption switching service: Securely switch your encrypted data to another format," *IEEE Transactions on Services Computing*, vol. 14, no. 5, pp. 1357–1369, Sep. 1, 2021. DOI: 10.1109/tsc.2018.2876849.

[54] A. Sharma and K. M. Goolsbey, "Simulation-based approach to efficient commonsense reasoning in very large knowledge bases," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 1360–1367.

[55] R. Avula, "Addressing barriers in data collection, transmission, and security to optimize data availability in healthcare systems for improved clinical decision-making and analytics," *Applied Research in Artificial Intelligence and Cloud Computing*, vol. 4, no. 1, pp. 78–93, 2021.

[56] E. Korot, Z. Guan, D. Ferraz, *et al.*, "Code-free deep learning for multi-modality medical image classification," *Nature Machine Intelligence*, vol. 3, no. 4, pp. 288–298, Mar. 1, 2021. DOI: 10.1038/s42256-021-00305-2.

[57] M. Kansara, "A framework for automation of cloud migrations for efficiency, scalability, and robust security across diverse infrastructures," *Quarterly Journal of Emerging Technologies and Innovations*, vol. 8, no. 2, pp. 173–189, 2023.

[58] M. P. Ueckermann, J. Bieszczad, D. Entekhabi, *et al.*, "Podpac: Open-source python software for enabling harmonized, plug-and-play processing of disparate earth observation data sets and seamless transition onto the serverless cloud by earth scientists," *Earth Science Informatics*, vol. 13, no. 4, pp. 1507–1521, Aug. 28, 2020. DOI: 10.1007/s12145-020-00506-0.

[59] H.-Y. Lin and S.-Y. Yang, "A smart cloud-based energy data mining agent using big data analysis technology," *Smart Science*, vol. 7, no. 3, pp. 175–183, Apr. 14, 2019. DOI: 10.1080/23080477.2019.1600112.

[60] X. Li, F. He, and W. Li, "A cloud-terminal-based cyber-physical system architecture for energy efficient machining process optimization," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 3, pp. 1049–1064, May 10, 2018. DOI: 10.1007/s12652-018-0832-1.

[61] M. E. S. Saeed, L. Qunying, G. Y. Tian, B. Gao, and F. Li, "Akaiots: Authenticated key agreement for internet of things," *Wireless Networks*, vol. 25, no. 6, pp. 3081–3101, Mar. 10, 2018. DOI: 10.1007/s11276-018-1704-5.

[62] A.-V. Vo and D. F. Laefer, "A big data approach for comprehensive urban shadow analysis from airborne laser scanning point clouds," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 4, pp. 131–137, Sep. 23, 2019. DOI: 10.5194/isprs-annals-iv-4-w8-131-2019.

[63] F. Barreiro, D. Benjamin, T. Childers, *et al.*, "The future of distributed computing systems in atlas: Boldly venturing beyond grids," *EPJ Web of Conferences*, vol. 214, pp. 03 047–, Sep. 17, 2019. DOI: 10.1051/epjconf/201921403047.

[64] P. Fahr, J. M. Buchanan, and S. Wordsworth, "A review of the challenges of using biomedical big data for economic evaluations of precision medicine," *Applied health economics and health policy*, vol. 17, no. 4, pp. 443–452, Apr. 3, 2019. DOI: 10.1007/s40258-019-00474-7.

[65] J. S. Yoo, D. Boulware, and D. Kimmey, "Parallel co-location mining with mapreduce and nosql systems," *Knowledge and Information Systems*, vol. 62, no. 4, pp. 1433–1463, Aug. 21, 2019. DOI: 10.1007/s10115-019-01381-y.

[66] S. Dash, S. K. Shakyawar, M. Sharma, and S. Kaushik, "Big data in healthcare: Management, analysis and future prospects," *Journal of Big Data*, vol. 6, no. 1, pp. 1–25, Jun. 19, 2019. DOI: 10.1186/s40537-019-0217-0.

[67] K. Yeung, "Algorithmic regulation: A critical interrogation," *Regulation & Governance*, vol. 12, no. 4, pp. 505–523, Jul. 31, 2017. DOI: 10.1111/rego.12158.

[68] Y. Cai, D. Li, and Y. Wang, "Medical big data intrusion detection system based on virtual data analysis from assurance perspective," *International Journal of System Assurance Engineering and Management*, vol. 12, no. 6, pp. 1106–1116, Aug. 25, 2021. DOI: 10.1007/s13198-021-01279-5.

[69] D. Zhang and M. R. Kabuka, "Distributed relationship mining over big scholar data," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 1, pp. 354–365, Jan. 1, 2021. DOI: 10.1109/tetc.2018.2829772.

[70] S. M. Lee, D. Lee, and Y. S. Kim, "The quality management ecosystem for predictive maintenance in the industry 4.0 era," *International Journal of Quality Innovation*, vol. 5, no. 1, pp. 1–11, Mar. 27, 2019. DOI: 10.1186/s40887-019-0029-5.

[71] R. Moreno-Vozmediano, R. S. Montero, E. Huedo, and I. M. Llorente, "Efficient resource provisioning for elastic cloud services based on machine learning techniques," *Journal of Cloud Computing*, vol. 8, no. 1, pp. 1–18, Apr. 16, 2019. DOI: 10.1186/s13677-019-0128-9.

[72] M. Zook, S. Barocas, danah boyd, *et al.*, "Ten simple rules for responsible big data research," *PLoS computational biology*, vol. 13, no. 3, pp. 1–10, Mar. 30, 2017. DOI: 10.1371/journal.pcbi.1005399.

[73] C. Anagnostopoulos and P. Triantafillou, "Query-driven learning for predictive analytics of data subspace cardinality," *ACM Transactions on Knowledge Discovery from Data*, vol. 11, no. 4, pp. 47–46, Jun. 29, 2017. DOI: 10.1145/3059177.

[74] D. Makatun, J. Lauret, and H. Rudová, "Planning of distributed data production for high energy and nuclear physics," *Cluster Computing*, vol. 21, no. 4, pp. 1949–1965, Aug. 25, 2018. DOI: 10.1007/s10586-018-2834-3.

[75] H. Labbaci, B. Medjahed, and Y. Aklouf, "A social network approach for recommending interoperable web services," *Distributed and Parallel Databases*, vol. 38, no. 4, pp. 927–961, Aug. 17, 2020. DOI: 10.1007/s10619-020-07308-9.

[76] J. Duarte, P. Harris, S. Hauck, *et al.*, "Fpga-accelerated machine learning inference as a service for particle physics computing," *Computing and Software for Big Science*, vol. 3, no. 1, pp. 1–15, Oct. 14, 2019. DOI: 10.1007/s41781-019-0027-2.

[77] A. Wang, O. E. Fakir, J. Liu, Q. Zhang, Y. Zheng, and L. Wang, "Multi-objective finite element simulations of a sheet metal-forming process via a cloud-based platform," *The International Journal of Advanced Manufacturing Technology*, vol. 100, no. 9, pp. 2753–2765, Oct. 20, 2018. DOI: 10.1007/s00170-018-2877-x.

[78] T. C. Lim, "Use of the mchargian lusa in agricultural research and decision-making in the age of non-stationarity and big earth observation data," *Socio-Ecological Practice Research*, vol. 1, no. 3, pp. 297–324, Aug. 20, 2019. DOI: 10.1007/s42532-019-00022-6.

[79] R. Avula, "Assessing the impact of data quality on predictive analytics in healthcare: Strategies, tools, and techniques for ensuring accuracy, completeness, and timeliness in electronic health records," *Sage Science Review of Applied Machine Learning*, vol. 4, no. 2, pp. 31–47, 2021.

[80] K. Mershad, Q. M. Malluhi, M. Ouzzani, M. Tang, M. Gribskov, and W. G. Aref, "Audit: Approving and tracking updates with dependencies in collaborative databases," *Distributed and Parallel Databases*, vol. 36, no. 1, pp. 81–119, Sep. 21, 2017. DOI: 10.1007/s10619-017-7208-y.

[81] W. S. Pittard, C. K. Villaveces, and S. Li, "A bioinformatics primer to data science, with examples for metabolomics," *Methods in molecular biology (Clifton, N.J.)*, vol. 2104, pp. 245–263, Jan. 18, 2020. DOI: 10.1007/978-1-0716-0239-3_14.

[82] S. Bella, G. Belalem, A. Belbachir, and H. Benfriha, "Hmdcs-uv: A concept study of hybrid monitoring, detection and cleaning system for unmanned vehicles.," *Journal of intelligent & robotic systems*, vol. 102, no. 2, pp. 44–44, May 25, 2021. DOI: 10.1007/s10846-021-01372-8.