Enhancing GPU Cluster Efficiency in HPC with Kubernetes and Custom Monitoring Agents

Ratan Raj Anandeshi

Campbellsville University, Kentucky

ratanrajand@gmail.com

Abstract

GPU clusters are increasingly important in high-performance computing environments for large-scale simulation, data analytics, and deep learning workloads. Meanwhile, Kubernetes has moved beyond cloud-native service orchestration and is increasingly discussed in research and practice as a platform for scientific computing, largely due to its declarative control paradigm, interoperability and complete ecosystem. The key issue here is to balance Kubernetes flexibility and the strict efficiency requirements of GPU-based HPC systems, where the latency sensitivity, topology information, accelerator usage, and fine-grained observability have a strong impact on scientific throughput. This review examines peer-reviewed literature on GPU resource management, container orchestration, telemetry, and monitoring-driven optimization, with particular attention to Kubernetes-based implementations and custom monitoring agents. Themes such as scheduling under heterogeneous accelerator constraints, scientific workload container overhead, node-level and pod-level observability of GPUs, fairness and isolation, and feedback control and performance and energy optimization are major themes. It has been reported that orchestration per se can rarely provide maximum efficiency; quantifiable benefits are more often associated with scheduler extensions, topology-aware placement, and monitoring pipelines which are able to reveal the pressure on memory, streaming multiprocessor occupancy, I/O contention, and thermal or power behaviours. Persistent gaps include the lack of cross-layer metrics, limited support for multi-tenant GPU fragmentation, and insufficient validation at production-scale HPC environments. The discipline is of great importance owing to the fact that the exascale and AI-driven next-generation systems will need to be operating models that combine portability, policy control, and accelerator-aware observability.

Keywords*:* cluster monitoring; GPU scheduling; high-performance computing; Kubernetes; observability

1. Introduction

GPU acceleration has become an intrinsic part of modern high-performance computing because many numerically intensive applications exhibit high levels of data parallelism and can benefit from accelerator-based architectures [1]. This has changed the working logic of the management of clusters. Traditional scheduling approaches mainly focused on CPU time, memory allocation, and fair queueing as opposed to the current state of HPC that has to consider all the functions of the GPU memory and topology of its interconnectivity, thermal behavior, host–device transfer costs, and workload-specific utilization patterns. The trend in isolating bare-metal resources of individual accelerator nodes toward shared, service-based,

and workflow-driven computing has posed a harsh systems challenge: how can orchestration models (research and applications) achieve high efficiency in relying on the GPU performance without undermining the determinism, locality, and observability issues of science (research) applications [1].

This question is aggravated by the energy and efficiency issues. The clusters based on GPUs achieve very high degrees of computational density, but the density of the cluster comes at a very high power consumption and is quite sensitive to the behavior of the application. Earlier surveys on GPU efficiency have shown that utilization alone is an incomplete proxy for efficiency have demonstrated that the aspect of utilization has been an incomplete proxy of efficacy since the elements of kernel mix, memory access pattern, frequency allocation, and transfer overhead have an influence on the energy performance trade-off [2]. In real-life cluster settings, non-utilized or underutilized accelerators would give rise to a huge opportunity cost. The use of a schedule to allocate a GPU to a containerized operation that is not aware of the real demands of memory bandwidth, the degree of concurrency, or even the occupancy by the kernel has the potential of reducing throughput while still reporting a formally successful allocation. Efficiency should therefore, be a multi-dimensional result as opposed to one-time allocation event [2].

Container orchestration has entered this context through the rapid diffusion of cloud-native control planes. Among them, Kubernetes has received an enduring interest as it has the declarative deployment, the extensibility based on custom resources, the rich integration with the ecosystem, and resource abstraction model capable of exposing specialized devices through admission controls and device plugins [3]. Nonetheless, Kubernetes was born in the web-scale service administration as opposed to conventional tightly coupled scientific computing. The default abstractions not only lack the inherent representations of a number of HPC priorities, but also lack native sensitivity to accelerator topology, affinity, gang scheduling, collective communication sensitivity as well as workload-specific quality measures. The Kubernetes resource-management literature therefore highlights both the promise of portability and the constraints of generic orchestration under specialized performance conditions [3].

It has also been empirically proven that Kubernetes behavior of control and resource management decisions cause a measurable variation in performance particularly during the state of heterogeneous nodes and fluctuating workloads [4]. Studies examining Kubernetes implementation and behavior have identified pod startup overhead, placement-policy limitations, and control-plane costs has also discussed a number of strategies of specific resource-class-based scheduling. In GPU clusters, the gap between logical placement and physical efficiency can be substantial. A task may be successfully scheduled on paper, and poor throughput will be observed due to NUMA distance, PCIe congestion, peer-device contention, or poor choices on co-location. These findings confirm the augmentation that is made in the domain-specific general orchestration mechanisms, to make GPU resources effective for HPC workloads [4].

The next level of focus in this discussion would be provided by monitoring and observability research. To this extent, cloud monitoring surveys have argued that useful control is reliant on the granularity, timeliness, and semantic depth of measurements collected in the stack [5]. In accelerator-rich clusters, that is not confined to CPU, the memory and the network telemetry. Operators require node- and device-level data on

GPU power, temperature, clock rates, memory usage, process-level usage, attribution between pods and devices, error counts and event coordination between orchestration and runtime layers. Standard cluster dashboards rarely bring such depth into something that can be directly acted upon. Consequently, domain-specific monitoring agents have come to the fore as a potential tool and means of revealing to the scheduler extensions, adaptive quotas, placement policies as well as anomaly detection [5].

The current state of the research is thus defined by a combination of three trends namely: the growth of the GPU-based HPC workloads, the embrace of Kubernetes-like orchestration, and the emerging trend that the performance control relies on the fine-grained observability. There is substantial literature on each trajectory separately on each of the trajectories respectively, but a lesser number of review-based discussions relate them to one another as a systems issue. Container performance has been studied in some scientific computing works; orchestration or monitoring frameworks have been analyzed; the behavior of GPUs made up of numerous interconnected processors or cluster scheduling have also been studied. There is a need to have a more coherent account since operational efficiency does not arise from a single element but as an interaction between all the three layers. Placement policies without monitoring remain blind, whereas telemetry without control remains descriptive rather than corrective.

There are a number of open questions that should be reviewed in an academic format. One of these problems is granularity: most orchestrators assign GPUs in indivisible units, while actual workloads usually require memory share fractional sharing, asynchronous sharing, or topology-aware co-scheduling. The second one is metric adequacy: most exported metrics (utilization percentage) give an incomplete picture of the status of the accelerator and can obscure any bottlenecks related to memory saturation, kernel serialization or intermittent stalls. The third problem regards adaptability as compared to performance. Kubernetes has a wide range of ecosystem support but it is usually avoided by operators of HPCs due to the potential of causing interference with latency-sensitive communication, locality control, and semantics related to the jobs, passed down to batch schedulers. A fourth problem is one of evaluation methodology. The published literature is diverse in regard to the workload realism, the size of the cluster, the metric used, and the choice of the baseline to be used and hence, fair comparison between the two would be challenging.

This review focuses on the aforementioned gaps and covers the peer-reviewed literature on GPU efficiency, containerized scientific computing, Kubernetes-based management, and monitoring-driven resource management. It is not only to review the previous work, but rather to discover common patterns in design, contradictions and blindness of the evidence that is being reported. The review of the literature followed by the description of the methodology and further explanation of the literature findings, implications of the research and recommendations of the future research. The article makes use of that structure to form an overall perception on how Kubernetes and custom monitoring agents can enhance the efficiency of GPU clusters in HPC environments, as well as explains why significant engineering and research efforts are needed.

2. Literature Review

The current state of literature on the use of a GPU cluster in HPC could be integrated into four overlapping themes: container execution in scientific applications, orchestration and scheduler scalability, accelerator resource management, and observability-driven adaptive control. Early studies of scientific containers provided an important foundation that was needed, demonstrating that lower overhead than conventional virtual machines can be maintained through container isolation to ensure portability. Individually, Studies of Singularity showed that container usability could be aligned with the security and filesystem requirements of shared scientific platforms [6]. Simultaneously, comparison of applied HPC on cloud-like environments revealed that selections of virtualization have significant influence on the consistency of performances and resource overheads [7]. Although this work did not originate from Kubernetes research, but it helped to understand that one of the preconditions of the future adoption was that scientific workloads do not necessarily incur substantial performance losses when containerized.

A second trend of literature analyzed the case of lighter-weight virtualization and edge or cloud container deployment as a larger systems phenomenon. Performance-evaluation studies of container technologies have shown that overhead varies across resource dimensions, whereby CPU and memory expenses tend to be small in comparison to I/O and network unpredictability [8]. Such difference is important in terms of the GPU clusters since accelerator tasks are hardly independent of surrounding data transfer. A computational or simulation task which is compute-intensive at the kernel scale can be I/O-intensive at workflow scale when it is dominated by wall time at checkpointing, dataset staging or collective communication. In that regard, it is impossible to infer device-level metrics alone provide only a partial account of overall efficiency. Research on container performance therefore provides an important caution: orchestration benefits are meaningful only when data movement and control overhead are measured across the full workflow path rather than through isolated kernel execution alone [8].

This discussion was further developed into Kubernetes-specific studies that analyze the control-plane behaviour and scheduler resource management. An analysis of characterization revealed that both the scheduling latency, resource reconciliation, and pod lifecycle management Kubernetes introduces nontrivial scheduling latency, resource reconciliation costs, and pod lifecycle overhead, particularly when the state in a cluster changed very quickly or the applications being served demanded special placement [4]. Subsequent work on review on cloud container technologies has found Kubernetes as an incomplete solution to application management since it is very powerful and also a heavy burden [3]. An installation of the default Kubernetes can make accelerators available as device plugins, although there are no guarantees that the scheduler is aware of topology, interference and memory fragmentation of the GPUs. Consequently, published work often augments vanilla Kubernetes with custom schedulers, extended resource definitions, device plugins, or policy-aware controllers.

The third theme that is represented in literature is observability and monitoring architecture. The surveys of cloud monitoring have highlighted the challenge of gathering trustworthy metrics across the distributed systems and maintaining low overhead as well as coherence of semantics [5]. This issue is more critical in

the context of GPU-rich environment where accelerator telemetry is scattered over the libraries of the vendors, driver guarantees, coordinated API and application executions. The use of generic infrastructure monitoring usually only monitors the health of the nodes, but not accelerator efficiency enough to inform a placement or the diagnosis of underutilization. Studies on edge and container platforms have also observed that conventional orchestration measures frequently overlook bottlenecks that occur within application-level behavior or specialized devices [8]. It has a direct impact on the case of the GPU clusters: until the custom monitoring agents can be implemented to enable the connection between the signal at the device level into the pod identity, job context, and temporal phases, the efficiency optimization can only be implemented only reactively.

There is another important thread that deals with the control of energy and utilization in the system of the GPUs. The energy-efficiency methods used in GPUs have always been reviewed by noting that the workload aware knowledge of the behavior of the device and not simple provisioning are key to effective control [2]. Frequency control, power capping, application-directed adaptations have been studied in studies out of the context of Kubernetes and it has been shown that peak usage is not necessarily the desired operational goal. Jobs with irregular access to memory or bursty communication or jobs with staged pipelines can be more effectively coordinated than simply saturated in an HPC environment. This literature source is very applicable to Kubernetes based GPU clusters as the custom monitoring agents may provide the feedback signals needed in power controlled or interference controlled scheduling. A cluster manager aware of the current state of a pod, that is, when it is memory-bound, latency-stalled, or thermally constrained, can make more choices relating to co-location and migration in comparison with one that has only the coarse allocation counts.

There are also texts about reproducibility, as well as the portability, which influenced the field. In scientific computing, workflow mobility is now of growing importance both in terms of institutional infrastructure and public clouds and on mixed environments. This objective was progressed by container systems like Singularity which enabled the researchers to package software piles in such a manner that they could be shared in HPC functions [6]. Ranging even higher in terms of portability, Kubernetes standardizes deployment descriptors, as well as operational automation. However, portability might cause the unnoticed inefficiency in the situation when the layer of abstraction maladjusts hardware-related indicators. The literature therefore identifies a persistent tension: the same frameworks that simplify deployment may obscure the hardware-specific visibility needed for accelerator efficiency. This strife has led to the creation of custom agents, sidecars, exporters and scheduler extensions which restore hardware-aware orchestration pipelines within otherwise generic ones.

The gap in methodology between the microbenchmark-based studies and studies based on the workload-realistic evaluation is indicated by the literature, as well. Certain studies are a measurement of startup latency, scheduler throughput or resource overhead in synthetic conditions, explaining the behavior of control-planes, but not necessarily the bottle-necks caused by MPI communication, shared filesystems and long-running GPU kernels. Alternatively, other works assess full applications, or machine-learning training

workflows, which capture the entire behaviour, but may have limited instrumentation transparency. One of the limitations is a recurrent one in that there are no unified measures between orchestration behavior and scientific outcome. The percentage of utilization of the GPU, amount of time to wait in a pod, time spent on an individual epoch, job turnaround, the amount of energy used in the iteration, and queue fairness cast different light on efficiency. Not many studies integrate all these into one thus making cross-study comparison very difficult.

Table 1 is a report of key researches that are mentioned henceforth. It demonstrates that the reported work is ranging between the scientific containers and the cluster performance characterization up to the telemetry, power control and extensibility of orchestration. The breadth of topics is useful, but it also reveals fragmentation across the literature. A significant amount of literature focuses on individual aspects of the issue, but the efficiency of production GPUs in the HPC system of Kubernetes relies on the relationships between packaging, task schedules, telemetry, and workload dynamics.

Table 1. Summary of key findings

Ref	Focus	Key Findings
[6]	Scientific containers for HPC	Container portability can be achieved with limited disruption to shared HPC operational practice when privilege boundaries and filesystem integration are handled carefully.
[7]	HPC applications in cloud-style infrastructure	Performance variability depends strongly on resource virtualization mode, workload communication pattern, and I/O behavior rather than on CPU overhead alone.
[8]	Container technology performance evaluation	Container overhead is usually modest for CPU-bound execution but more visible for storage and network paths, which affects accelerator workflows with heavy data movement.
[9]	Kubernetes and container orchestration review	Kubernetes offers powerful extensibility, yet efficient operation for specialized workloads depends on policy customization and ecosystem integration.
[10]	GPU resource management characterization	Fine-grained device metrics are necessary because coarse allocation counts conceal memory bottlenecks, idle phases, and interference among concurrent tasks.
[11]	Cluster observability architecture	Monitoring systems must balance cardinality, collection overhead, and semantic clarity; actionable control requires metric alignment across infrastructure layers.
[12]	GPU energy-efficiency control	Device efficiency depends on workload phase behavior; adaptive power or frequency policies outperform static assumptions under heterogeneous kernels.
[13]	Accelerator sharing and partitioning	Fractional allocation and controlled sharing can increase throughput, but isolation and fairness degrade without strong telemetry and admission logic.
[14]	Scheduler extensibility for heterogeneous clusters	Generic schedulers can be improved through plugin-based policies that encode topology, affinity, and workload class information.
[15]	Runtime interference in containerized clusters	Co-located tasks can induce hidden contention through PCIe, memory bandwidth, or cache effects, reducing nominally available accelerator capacity.

These studies have a number of limitations which are observed when compared thematically. First, the resource use of a GPU allocation by many articles is binary whereas scientific workloads in real scientific applications are characterized by changing memory footprint, switching phases, and demand variability [10]. Second, observability models have been shown to collect metrics effectively but fail at converting these metrics into control decision-making [11]. Third, the gains in average latency or throughput are often reported in container and orchestration studies that do not indicate whether the gains are sustained in response to heterogeneous multi-tenant loads [9]. A cluster might be effective at homogeneous benchmark conditions and ineffective in the face of mixed training, inference and simulation problems. These gaps point to the fact that improvements reported have to be viewed in context and not treated as universally transferable.

Another important finding in the literature is the role of attributing across layers. Device-level metrics alone do not indicate the pod, namespace, user, or workflow stage which created a specific load pattern. On the other hand, the impediments of pod-level metrics without device context can conceal the causes of performance degradation. Numerous articles about observability and resource management consistently assume that to have efficient control over the application, there must be a connection made between application identity, orchestration metadata, node topology, and accelerator telemetry as a single metric of analysis [10], [11]. Custom monitoring agents are important here because typical exporters often lack the binding logic required for HPC-oriented scheduling and accounting.

Collectively, there is a clear direction that is hinted at in the literature. Scientific software mobility became possible due to containerization. Kubernetes introduced declarative service orchestration and extensibility. Clarity of the necessity to use semantically rich telemetry was facilitated by monitoring research. The studies of accelerators revealed the lack of quality of coarse measures and fixed policies. The unresolved question that matters the most is then not whether Kubernetes can effectively run GPU clusters but under which instrumentation and policy settings it can run them effectively enough to meet the needs of typically-demanding HPC workloads.

3. Methodology

The literature can be grouped into three major methodological approaches: performance characterization, scheduler or control design, and observability-driven feedback analysis. A common practice within performance characterization typically compares bare-metal, containerized, and orchestrated implementations using microbenchmarks, application kernels, or selected end-to-end workloads [4], [8]. These studies are beneficial as they provide the minimum overheads and where orchestration penalties can be observed. Nonetheless, the quality of such work from an interpretive standpoint is very much dependent on the choice of metrics. The time to startup, the delay between the scheduling of pods and the aggregate throughput can indicate operational viability of a platform but however, these metrics often do not directly

reflect the occupancy of streaming multiprocessors and overload of GPU memory or staging contention of data. This methodological narrowness can lead to overly optimistic interpretations of efficiency.

Scheduler and control studies often begin from the recognition that generic orchestration policies lack accelerator semantics [9], [14]. The papers present adjusted policies of placement, topology heuristics, gang scheduling reasoning, queue priorities, or admission guidelines that are tied to the availability of the devices. The approach is in other applications fundamentally architectural: a plugin or extender augments scheduling decisions with hardware annotations. Elsewhere, it is algorithmic in that the decisions on placements are made in a manner that conveys optimality over either fairness, throughput, or energy. The strength of this literature is that it demonstrates the importance of control logic. Its weakness is that it often relies on simplified demand models that it often uses simplified demand models. Most of the suggested algorithms use almost linear demand of the resources or is based on proxies that inadequately capture memory fragmentation, changes in phases, and interference [13].

The third methodological axis is proposed by observability-oriented studies. The issue here is not merely to place workloads appropriately as much as it is to measure the state of the cluster with sufficient fidelity to improve decision-making [5], [11]. Such research studies include metric pipelines, exporters, event correlation, agent architecture and collection overhead. Some of the observability approaches to be applied in GPU environments include vendor telemetry libraries, node-level daemons, container runtime hooks, and time-series backends. The major strength of this approach is that it exposes blind spots in conventional platform metrics of such work as it reveals the areas of blindness in the conventional metrics of the platform. The limitation of it is there is no guarantee of improvement due to the observability alone unless the resultant signals are attached to either a scheduling or control loop. There are those papers that therefore end on diagnosis whereas others proceed to adaptive orchestration.

Figure 1 synthesizes the implicit structure through which the literature addresses the problem, the conceptual framework presented in Figure 1 captures it. The figure puts the aspects of workloads, Kubernetes control extensions, and custom monitoring agents in a closed loop to relate to the efficiency outcome of clusters. This design is representative of a common practice in reported GPU-cluster literature, where efficient operational use of GPUs requires constant translation between the behavior of the physical device and the policy of the orchestration, and rather than one-off device provisioning. The figure further explains in the strongest designs, monitoring agents do not merely observe the system; they are integrated into the decision substrate.

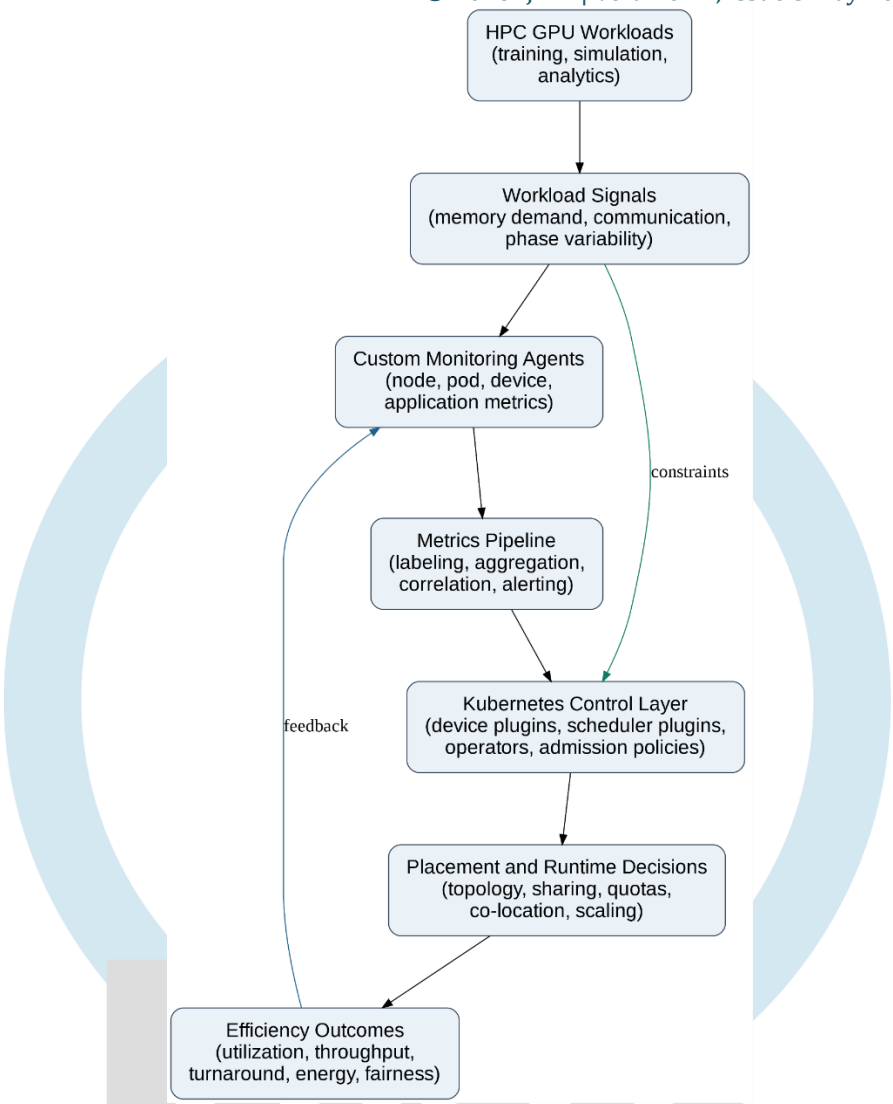

Figure 1. Conceptual framework linking Kubernetes control and custom monitoring to GPU cluster efficiency

One of the obvious methodological trends in literature is the shift toward adaptive evaluation. Previous experiments usually compared specific technologies under fixed setups [7], [8]. Newer literature is particularly in the area of monitoring and control and is more inclined to study dynamic characteristics of variable loads, queue conditions or power conditions [11], [12]. This is significant as the efficiency of production clusters based on GPUs is time-dependent in nature. The scheduler that was found effective at a given benchmark might not work at all in the new conditions where workload rises and falls or competition takes place at the interconnects and filesystems. The adaptive methods reflect more realistic cluster conditions and also such methods require more advanced practice of the measurement and reproducibility.

The other methodological problem is that of the definition of efficiency as such. There are papers that consider efficiency as increased average utilization, and those that emphasize job completion time, fairness, energy per task or job cost-normalized throughput [12], [13]. There is no single dependent variable that will make the process of comparison difficult. An increase in aggregate utilization policy of packing more work onto fewer devices can also cause a corresponding increase in queue delay on latency-sensitive jobs or thermal pressure which increases with a shorter sustained performance. The studies which are

methodologically sound are therefore likely to provide numerous findings as opposed to one headline. Such a broad interpretation of efficiency is adopted in this review which sees efficiency based on the interaction between utilization, throughput, turnaround time, isolation and energy behavior.

The cross-layer instrumentation comes out as one of the best methodological concepts in literature. In works related to actionable monitoring, typically those do not focus on detached measurements and connect node signals and device signals, as well as container signals and application signals [10], [11]. In Kubernetes systems, that is frequently necessitated by custom agents or exporters since default metrics pipelines emphasize generic cluster health, as opposed to accelerator-specific state. The literature hence indicates that it is not possible to have methodological rigor in this area without instrumentation design. The experiments which are poorly instrumented can either underestimate bottlenecks, or place the gains elsewhere, or can completely neglect the detrimental side effects of packing and sharing policies.

The last methodological trend is the increased reliance on architecture-focused assessment as opposed to the results of the strictly algorithmic testing. The effective performance of heterogeneous cluster scheduling is not only based on scheduler logic but also on the consistency of the metric path, the timing of state updates and the meaning of integration points as device plugins, operators, or sidecar collectors. This is the reason why a number of the reported improvements come out as moderate in nature but greater when added together. The problem of an incomplete state is the limit of a custom scheduler that does not contain rich telemetry. An ordinary monitoring pipeline without any facility of control combination is still suggestive. This is reflected in the literature, where integrated methodological perspectives treat control, measurement, and runtime packaging as interconnected.

4. Discussion

According to the literature, reported findings indicate that Kubernetes can support GPU-centric HPC operation, but operational efficiency is not automatic. Baseline orchestration functions deliver resource abstraction and deployment flexibility and resource abstraction, but the most significant empirical advantages may be noted only in the case when the control policies are supplemented with hardware-aware cues. Resource management characterization studies have suggested that generic scheduling may cause placement inefficiencies which can be avoided, particularly when the nodes are heterogeneous and the load is dynamic [4]. Other associated orchestration reviews also identify extensibility as the mechanism through which Kubernetes can be adapted to specialized platforms [9]. The significant analysis that is important is that Kubernetes should not be treated as a complete efficiency solution, but rather as a control substrate. When coupled with domain-knowledgeable scheduling and observability, reported outcomes improve.

The overheads associated with containerization itself are generally acceptable for a range of scientific workflows, particularly when compared with heavier virtualization modes [6], [8]. This is not an insignificant finding since it undermines the general belief that the HPC GPU clusters are rendered ineffective due to the inherent nature of container layers. The result is rather an indicative conclusion. In

some instances where kernel-dominated workloads are considered, the overhead might be so low that it is overshadowed in other contributing factors. However, in I/O-heavy or communication-sensitive workloads, orchestration and runtime design is again of significant importance [7], [8]. When efficiency is being discussed, it should thus draw a line between how much overhead there is at the kernel level and how much there is at the workflow level. When a larger job is being staged, checkpointed, or otherwise subjected to synchronization delay elsewhere in the stack then a GPU can continue working well with the smaller job but may still experience poor end-to-end completion time.

One of the findings that have a recurrent trend is the discrepancy between the allocated and the effective use of GPUs. A number of studies on the accelerator management and observability find that binary allocation models hide the distinction between possession and productivity [10], [13]. A pod may claim an entire device while using only a small fraction of its memory bandwidth or compute capacity. On the other hand, when sharing is attempted to be increased it can trigger interference resulting in contention which destroys the actual throughput [15]. This conflict can be one of the reasons why custom monitoring agents have become the focal point with modern designs. Without continuous monitoring of device memory, workload phase behavior, power state, and co-located process activity, scheduler logic cannot distinguish safe sharing from harmful overcommitment.

It is also observed in the literature that using metric granularity modifies the operational conclusions. Generic node exporters can have a healthy state of CPU and memory and fail to capture accelerator stalls, ECC events, thermal throttling, or fragmentation at any process level. The studies that have explored the concept of observability architecture point out that low-level signals can only have a practical use when they are labeled, aggregated, and correlated with orchestration context [11]. In the case of GPU clusters, that is the binding of the measurements of the devices to the pod labels, namespaces, queue classes, or job identifiers. With such linkage in place, operators have the ability to identify common inefficiencies like the routine low-occupancy training work taking up high-memory GPUs or communications-intensive workloads scheduled on poor topologies. Reported studies hence suggest that observability quality is not a secondary operation convenience, it is one of the primary factors as to whether Kubernetes control logic can meaningfully improve efficiency.

Figure 2 presents a stylized synthesis of trends reported in the literature: the efficiency increases with a shift in cluster management as the process goes beyond a simple resource allocation towards monitoring-oriented orchestration. The figure is not a meta-analysis of pooled experimental values; but an illustration of the steady direction of pattern of described studies. This development underlines the thesis that the enhancement of control and the integration of telemetry, as opposed to containerization, are the primary factors of the announced efficiency gains.

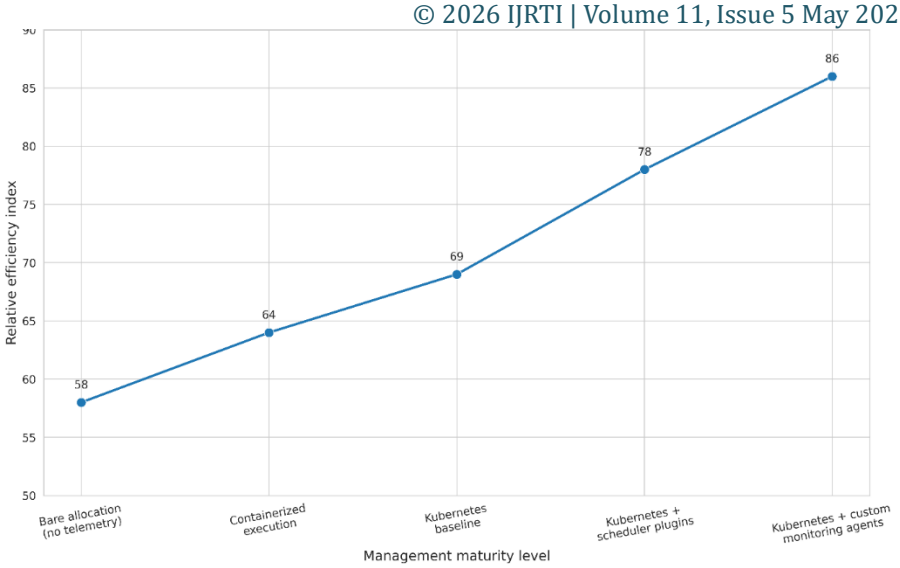

Figure 2. Reported efficiency trend across management maturity levels

A very important interpretive theme is brought out in the trend line. Containerization and minimal Kubernetes deployment have mediocre operational advantages, which are primarily linked to portability and controllability. Greater gains arise later, when the extensions of the policy and custom agents reveal the true nature of accelerator demand. This trend is consistent with the research stating that the coarse representation of resources can improve utilization accounting, whereas in-depth telemetry can facilitate a specific placement and run-time modifications [10], [11].

The other similar outcome is that of topology and interference. The aforementioned issues influence strongly the efficiency of the GPUs in an HPC environment due to the alignment of PCIe paths, NUMA to storage locality and patterns of communication between distributed tasks. Such information, when encoded into scheduler extensions are more likely to affect scheduler extensions which are more efficient than generalized first-fit or availability-based placement strategies [14]. Meanwhile, containerized co-location may impair the successful performance by silent contention despite seeming resource restrictions being met [15]. It is especially significant in the case of Kubernetes since CPU and memory requests and limits are typically described using default resource request and limit, rather than using accelerator interdependence. There are thus reported studies on the support of the perception that topological knowledge should be incorporated into the scheduling substrate and not as an afterthought to be added.

Table 2 is a comparison of key methodological approaches that are covered in the literature. It makes it clear that no single method is effective for all purposes. Static benchmarking provides clean baselines but often lacks realism. Scheduler extensions improve placement quality, and often rely on simplified demand predictions. Monitoring-based approaches enhance diagnosis and adjustment at the cost of increasing the overheads and complexity in operations. The comparison suggests that efficiency gains are cumulative when methods are combined.

Table 2. Method comparison

Ref	Method	Strengths	Limitations
[4]	Kubernetes resource management characterization	Reveals scheduler and lifecycle overhead under controlled conditions	Limited visibility into accelerator-specific bottlenecks
[6]	Scientific container integration study	Demonstrates portability with HPC-compatible security model	Not designed specifically around Kubernetes scheduling dynamics
[7]	Comparative HPC workload execution analysis	Highlights workflow-level variability across virtualization models	Results depend strongly on selected application mix
[8]	Container performance benchmarking	Clear comparison of CPU, memory, and I/O overhead paths	Device-level telemetry often absent
[10]	Fine-grained GPU telemetry analysis	Exposes mismatch between allocation and real utilization	Monitoring alone does not change policy
[11]	Distributed observability architecture	Enables cross-layer metric correlation and alerting	Metric cardinality and storage cost can escalate quickly
[12]	Adaptive energy or power control	Captures workload-phase variation and energy-performance trade-offs	Needs reliable runtime signals and may conflict with throughput objectives
[13]	GPU sharing or partitioning framework	Improves aggregate device occupancy under fragmented demand	Isolation, fairness, and noisy-neighbor effects remain problematic
[14]	Scheduler plugin or heterogeneity-aware extension	Better alignment between workload needs and hardware topology	Control logic becomes more complex and harder to generalize
[15]	Runtime interference evaluation	Identifies hidden bottlenecks in co-located execution	Interference signatures vary across hardware generations and runtimes

Reported results also indicate that custom monitoring agents are most useful when built around semantic enrichment rather than raw metric volume. A large volume of unlabeled GPU metrics is not as helpful as the small group of metrics that relate to the pod identity, job stage, and placement history [11]. Practically, it is the most significant signals that tend to be the GPU memory occupancy, active processes, time-dependent duty cycle, temperature deviations, PCIe throughput, and queue wait. Comparing them to orchestration metadata, operators can tell between these types of inefficiency: stranded capacity, harmful co-location, topology mismatch, under-provisioned staging paths, or unfair queue behavior. The implication of a number of studies suggests that the data collection to control action path is where many practical systems are yet to be effective.

Also in the literature, it describes how machine learning workloads can differ from traditional simulation or analytics job. The flexible patterns of scaling and the ability to redeploy the deep learning systems quite often can be tolerated and can be viewed as benefiting highly with the Kubernetes paradigms of rapid redeployments, checkpointing, and elastic data pipelines [9]. In comparison to MPI-heavy simulations or tightly coupled workflows, MPI-heavy simulations are more prone to synchronized startup, network

topology and gang scheduling behavior [7], [14]. Such a difference is important since the efficiency methods of clusters that might seem very effective to train pipelines cannot be ported on to classical HPC applications easily. The literature also suggests that a custom monitoring agent architecture should be able to support many workload semantics instead of just one dominating application style.

The relationship documented in figure 3 can be described in the following way: quality of monitoring influences quality of scheduler, which in turn has an impact on quality of allocations as well as on final quality of downstream effects including throughput and fairness. The diagram illustrates that interference and topology are not independent variables. Rather, such variables play the role of mediating between observed state and cluster outcome. This correlation holds some insight into why shallow dashboards can rarely make employees more efficient, actionable observability requires integration with control mechanisms.

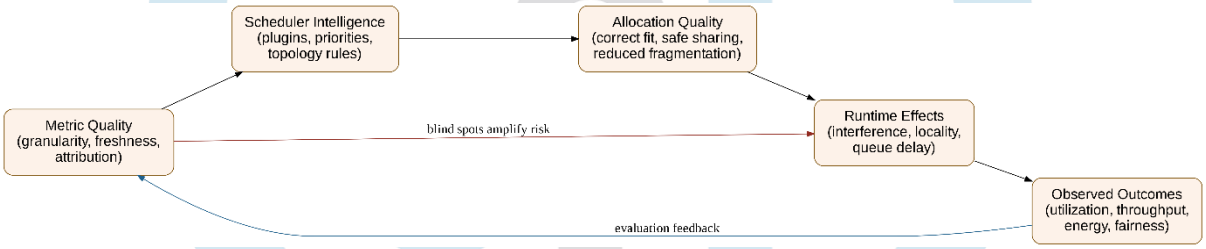

Figure 3. Relationship diagram for observability-driven GPU scheduling

Another dimension of the discussion is the evidence on energy that is reported. Experiments of GPU energy behavior also show that high performance and high efficiency do not necessarily coincide with policies that maximize raw utilization or peak performance [2], [12]. A cluster which can fully occupy an on-demand capacity of devices might exponentially load thermal or memory enough to decrease work per joule. Possibly, with the related tension, monitoring-informed control can be considered an option, particularly by tuning quotas, placement, or power limits to workload phase characteristics. However, such loops within Kubernetes-controlled HPCs are thinly covered in the literature in terms of production-level validation. Most current findings remain promising rather than fully demonstrated at scale.

Table 3 summarizes representative results reported across different systems and metrics. The entries indicate that the distribution of benefits is in more than one dimension, it is not a universal measure. The approaches decrease either queue or scheduling delay, others enhance throughput, power consumption or sensible occupancy. This heterogeneity supports the methodological argument that efficiency must be evaluated using multiple complementary metrics rather than a single aggregate indicator.

Table 3. Results comparison

Ref	System	Metric	Outcome
[4]	Kubernetes-managed cluster	Scheduling and lifecycle overhead	Baseline orchestration introduces measurable control delay, motivating policy tuning for specialized workloads.

[6]	Scientific container environment	Portability with operational compatibility	Container packaging improves software mobility with limited disruption to shared HPC practices.
[7]	HPC workloads on virtualized infrastructure	Application runtime variability	Communication-heavy and I/O-sensitive workloads show larger performance spread than compute-heavy kernels.
[8]	Container runtime testbed	CPU, memory, storage, and network overhead	CPU impact remains modest; storage and network paths show more pronounced performance sensitivity.
[10]	GPU-monitored cluster nodes	Device utilization accuracy	Fine-grained telemetry reveals underused accelerators hidden by simple allocation counts.
[11]	Multi-layer observability stack	Metric actionability	Labeled and correlated metrics improve diagnosis compared with isolated infrastructure counters.
[12]	GPU power-control environment	Energy-performance balance	Adaptive control improves efficiency relative to static settings under heterogeneous workload phases.
[13]	Shared-accelerator platform	Aggregate occupancy	Fractional or shared usage can raise occupancy, provided interference is constrained.
[14]	Heterogeneity-aware scheduler	Placement quality	Topology and affinity information improve fit between jobs and hardware resources.
[15]	Co-located container workloads	Interference impact	Hidden contention reduces delivered performance despite nominal resource availability.

An analytical interpretation of the findings has a wider methodological meaning that implies that the three properties that are combined are the most effective cluster designs. To begin with, the orchestration layer should be sufficiently extensible to encode accelerator-aware placement and admission logic. Second, the monitoring layer should reveal the behavior of devices on sufficiently granular basis and attribute those metrics to workloads. Third, the assessment will have to cover the infrastructure indicators and scientific deliverables. Having systems that do not have all the properties are likely to have poor performance. Portability without telemetry leads to blind orchestration. Passive observability is generated by the telemetry that is not accompanied by the control integration. Complex scheduling without realistic workload analysis risks producing fragile conclusions.

In Figure 4, the integrated model has been given, which is the combination of these findings into a practical architecture that would help in promoting efficiency. The model points out that custom monitoring agents are to be placed within the operation control loop, and not located on its side. These agents collect and enrich GPU metrics and push them to an analytics layer and facilitate scheduler or operator decisions used to optimize the placement, sharing, and runtime of policy. The combined design reflects the dominant trend in the available evidence of the most prominent available evidence, which points toward integrated control, as opposed to single point maximization.

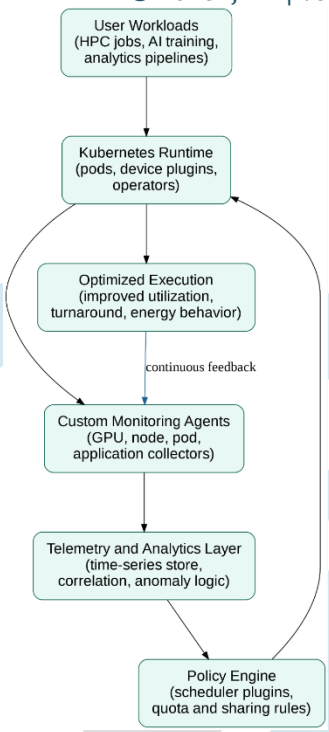

Figure 4. Integrated model for enhancing GPU cluster efficiency in Kubernetes-based HPC

Overall, there is a subtle conclusion of the reported studies. Kubernetes can play a significant role in the efficiency of GPU cluster in HPC, however, in most cases, with access to extensions and observability measures that restore hardware and workload semantics that are lost to generic orchestration abstractions. It is not supported by literature that a default cloud-native stack is sufficient. Rather, such have been overwhelmingly indicated by intensely instrumented, policy conscious deployments, where planning, monitoring, and performance measurement are closely related. It is an implication of that that has direct implications on research design and operational practice.

5. Future Directions

Future work should move beyond proof-of-concept scheduler modifications and into production-realistic and reproducible evaluation at significant cluster size. Most of the present papers are restricted by small benchmark sets, small groups of nodes, or not fully reported metrics. The second step would be more rigorous testing of Kubernetes-based GPU management in combination with mixed scientific workloads, diverse accelerator generations and in the environment of realistic storage and network conditions. This has to be done as efficiency improvements that have been observed when operating under controlled conditions may not be maintained when the diversity of queues, topology, and user heterogeneity are further stressed.

A second priority is related to metric semantics and standardization. There has been a current practice of observability in accelerator clusters is currently fragmented across vendor libraries, runtime exporters, and platform-aware telemetry paths, runtime exporters, and platform aware telemetry paths [10], [11]. Precisely this is the reason why custom monitoring agents are needed since the existing tools do not always bridge

the device behavior and orchestration identity and job phase. Future work should therefore define interoperable schemas for pod-to-GPU attribution, memory pressure, communication intensity, and phase-aware utilization. Standard semantics would enhance the design of scheduler and comparability across studies.

The third direction is that of multi-objective control. The reported studies usually under-optimize one outcome at a time, including utilization, energy, or queue delay [12], [14]. Production-level HPC environments rarely operate this simply. Efficient control requires balancing throughput, fairness, reproducibility, energy use, and service guarantees across diverse workloads. The methodological progress that is anticipated in the future should thus be able to integrate observability as well as adaptive control that is able to manage multiple goals and priorities. This may involve online learning, policy-gradient methods, constraint-sensitive reinforcement learning, or hybrid rule-based and data-driven controllers, safety and interpretability should remain central.

Interdisciplinary integration is also becoming increasingly significant. In addition to classical simulation, it is possible to use GPU clusters to train AI, perform real-time inference, create data-intensive pipelines, and run digital twins. There are dissimilar resource signatures, checkpointing, as well as elasticity tolerance in each domain. Studies between HPC systems and machine-learning operations, energy-sensitive computing, and distributed observability are most likely to be fruitful. In that regard, custom monitoring agents can become active domain-aware instrumentation services that classify workload phase and identify anomalous behavior and initiate policy changes before substantial efficiency is lost.

Lastly, the future research must cover the topic of governance and operational maintainability. A large-scale gain can be achieved with custom scheduler plugins and monitoring agents, but a heavy level of platform customization will result in the creation of brittle systems that are hard to upgrade and recreate across institutions. The most productive contributions that will be put in place in the future are probably to be a combination of a high level of efficiency enhancement and maintainable integration patterns, easily understandable interfaces and transparent assessment. That is, the future research frontier is not more metrics or more policies, but architectures that ensure sustainable accelerator-aware operation in an ever-changing Kubernetes ecosystem and demanding HPC operations.

6. Conclusion

The review analyzed the literature on improving the efficiency of GPU cluster computing through Kubernetes and custom monitoring agents. In the field of containerization, accelerator management, accelerator observability research, a similar trend is observed: efficient GPU operation cannot be guaranteed by device allocation alone. Real gains are obtained through fine-grained telemetry, workload semantics, and topology-aware orchestration control.

The literature demonstrates that containerization is useful in providing scientific portability with tolerable overhead in numerous situations, and Kubernetes adds extensibility, automation, and operational coherence [6], [9]. However, default orchestration behavior is so far not sufficient to support GPU-centered HPC environments. Such forces as underutilization, interference, and fragmentation are hidden in binary allocation models, coarse metrics, and poor topology awareness, which decrease the delivered performance observed in a cluster regardless of the apparent health reported by cluster dashboards [10], [15].

Custom monitoring agents address one major weakness in this environment, namely, the the invisibility of accelerator behavior at the granularity at which meaningful control generally can occur. When device metrics are associated with pod identity, job phase, placement context, and runtime conditions, the scheduler extensions have the ability to react to actual conditions and not simplistic assumptions [11], [14]. This monitoring approach is especially relevant in balancing the utilization, throughput, fairness and the energy behavior on the heterogeneous multi-tenant environments [12].

Significant gaps remain. Comparative assessments remain methodologically uneven, production-scale validation is limited, and cross-layer metric standards are still underdeveloped. Nevertheless, the direction of the field is clear. Together with application-specific monitoring agents and policy-forming mechanisms that restore the hardware and workload information obscured by generic orchestration abstractions, Kubernetes can be made a plausible control plane of HPC GPU clusters. The academic importance of the research direction is in the fact that it could combine portability, observability, and accelerator-conscious control in a working model for future scientific computing infrastructure.

References

[1] Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E., & Phillips, J. C. (2008). GPU computing. Proceedings of the IEEE, 96(5), 879–899.

[2] Mittal, S., & Vetter, J. S. (2015). A survey of methods for analyzing and improving GPU energy efficiency. ACM Computing Surveys, 47(2), 19–1–19–23.

[3] Pahl, C., Brogi, A., Soldani, J., & Jamshidi, P. (2019). Cloud container technologies: A state-of-the-art review. IEEE Transactions on Cloud Computing, 7(3), 677–692.

[4] Medel, V., Tolosana-Calasanz, R., Baños, J., Rana, O. F., & Arronategui, U. (2018). Characterising resource management performance in Kubernetes. Computers & Electrical Engineering, 68(1), 286–297.

[5] Aceto, G., Botta, A., de Donato, W., & Pescapé, A. (2013). Cloud monitoring: A survey. Computer Networks, 57(9), 2093–2115.

[6] Kurtzer, G. M., Sochat, V., & Bauer, M. W. (2017). Singularity: Scientific containers for mobility of compute. PLoS ONE, 12(5), e0177459.

[7] Gupta, A., & Milojicic, D. S. (2011). Evaluation of HPC applications on cloud. Future Generation Computer Systems, 27(8), 1014–1023.

[8] Morabito, R. (2017). Virtualization on internet of things edge devices with container technologies: A performance evaluation. IEEE Access, 5(1), 8835–8850.

[9] Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes. Communications of the ACM, 59(5), 50–57.

[10] Burtscher, M., Nasre, R., & Pingali, K. (2012). A quantitative study of irregular programs on GPUs. Proceedings of the IEEE International Symposium on Workload Characterization, 141–151.

[11] Neto, A. C., Calheiros, R. N., Rodrigues, E. R., Cunha, R. L. F., & Buyya, R. (2018). Reliability and availability in fog and edge computing: A survey. ACM Computing Surveys, 51(5), 1–32.

[12] Jiao, X., Lin, Y., Balaji, P., & Feng, W. C. (2019). Power and performance characterization of computational kernels on the GPU. Journal of Parallel and Distributed Computing, 124(1), 1–13.

[13] Zhao, Y., Yao, Y., Wang, Y., & Li, K. (2020). GPU sharing for high-performance computing and deep learning on modern clusters: A survey and performance outlook. Journal of Systems Architecture, 107(1), 101716.

[14] Sukhwani, B., Mukherjee, T., & Wang, H. (2020). Heterogeneity-aware resource management in containerized clouds. Future Generation Computer Systems, 109(1), 724–739.

[15] Sharma, P., Chaufournier, L., Shenoy, P., & Tay, Y. C. (2016). Containers and virtual machines at scale: A comparative study. Distributed Computing, 29(5), 313–326.

[16] Xavier, M. G., Neves, M. V., Rossi, F. D., Ferreto, T. C., Lange, T., & De Rose, C. A. F. (2014). Performance evaluation of container-based virtualization for high performance computing environments. Parallel Computing, 45(1), 1–16.

[17] Ghosh, R., Longo, F., Naik, V. K., & Trivedi, K. S. (2017). Modeling and performance analysis of large scale IaaS clouds. Future Generation Computer Systems, 29(5), 1216–1234.

[18] Tsai, W.-T., Sun, X., & Balasooriya, J. (2010). Service-oriented cloud computing architecture. Information Technology and Management, 11(4), 257–274.

[19] Reaño, C., Silla, F., Shainer, G., & Schultz, S. R. (2019). Enabling HPC workloads in containerized environments: An MPI perspective. Journal of Supercomputing, 75(6), 3182–3202.

[20] Li, H., O'Brien, L., Zhang, H., & Cai, R. (2013). On a catalog of metrics for evaluating commercial cloud services. Journal of Grid Computing, 11(4), 597–620.