Open Journal of Robotics, Autonomous Decision-Making, and Human-Machine Interaction, pp. 1–11

Real-time Machine Learning Pipelines for Big Data in Cloud Environments: Implementing Streaming Algorithms on Apache Kafka

Ahsan Raza[1]

1. *University of Malakand, Department of Computer Science, Chakdara, Lower Dir, Khyber Pakhtunkhwa, Pakistan.*

Abstract

Real-time machine learning pipelines in large-scale cloud environments demand robust streaming capabilities that handle massive volumes of continuously generated data. Implementing such pipelines involves designing data ingestion mechanisms with low latency, ensuring fault tolerance across distributed nodes, and balancing computational overhead to maintain near-instantaneous processing. This work explores the architecture and implementation of real-time machine learning pipelines on platforms that utilize streaming frameworks for ingesting and routing data, with a particular focus on Apache Kafka as a core messaging backbone. The approach encompasses techniques for model updates, online training procedures, and high-throughput inference, where each component interacts seamlessly within a highly scalable infrastructure. The discussion addresses methods for ensuring consistent and accurate data flow, together with stream partitioning strategies that minimize load imbalance. The emphasis is on constructing efficient pipelines by deploying advanced methods for compressing model parameters, optimizing queue buffers, and orchestrating dynamic resource allocation. Mathematical modeling is presented to capture the stochastic behavior of data arrival processes and to formalize the performance metrics governing throughput, latency, and reliability. Implementation aspects reveal how fault tolerance is achieved through replication mechanisms and leader election, while the theoretical underpinnings highlight the advantages of incremental updates and approximate computations to reduce overhead. Ultimately, this research provides a cohesive foundation for real-time machine learning workflows on modern cloud systems.

Introduction

The growing reliance on continuous data generation from diverse sensors, applications, and networked devices has escalated the need for real-time analytics in distributed environments [1]. The merging of big data and the imperatives of immediate insights has led to the integration of streaming technologies with sophisticated machine learning models. The paradigm centers on ingesting vast data streams with minimal delay and adapting predictive models in a manner that aligns with fluctuations in the underlying data distribution. This shift to continuous processing necessitates robust frameworks that can address the challenges of load balancing, fault tolerance, performance bottlenecks, and the intricacies of implementing online learning techniques. [2]

The foundation of real-time machine learning pipelines in cloud environments emerges from the interplay of virtualized infrastructure, distributed storage, and advanced messaging systems. One of the critical challenges is the concurrency management across the different nodes responsible for data ingestion, preprocessing, feature extraction, model training, and inference. Data streaming systems must ensure data integrity, maintain sequential order where relevant, and handle a substantial rate of incoming records [3]. The task involves not merely capturing the data but also orchestrating its flow so that each transformation and learning task remains scalable.

A significant driver for adopting streaming solutions is the pervasive requirement for low latency in applications such as fraud detection, recommendation engines, user behavior analysis, and network anomaly detection. The continuous reevaluation of model parameters based on incoming data fosters agility, but it also raises questions about how to handle model staleness, concept drift, and distributed synchronization. The computational overhead grows as the pipeline expands, since each module may need to operate on partial subsets of data or maintain specialized memory structures for

efficient updates [4]. Ensuring these modules coordinate effectively in a dynamic environment calls for advanced cluster management strategies and messaging solutions that can buffer, queue, and route data in near real time.

Another pressing concern is how to properly incorporate fault tolerance and recovery mechanisms into the design. In a cloud environment with volatile resource availability and failures at both the network and node levels, the pipeline must remain resilient and automatically recover [5]. This situation is typically addressed through replication, snapshotting of system states, transactional guarantees, and leader election protocols in messaging layers. Such procedures must execute without introducing prohibitive overhead or latency spikes. Ensuring that the pipeline does not lose data segments or corrupt model parameters is essential for maintaining accuracy and continuity in predictive performance. [6]

Achieving high throughput depends on implementing strategies for parallelization and partitioning of the data. However, partitioning must be accomplished judiciously to avoid excessive skew in the distribution of topics or keys, which can result in some nodes becoming bottlenecks. This challenge requires robust hashing or partition assignment logic, along with dynamic load balancing mechanisms capable of shifting partitions among brokers to adapt to varying workload conditions. The same adaptive principles extend to autoscaling of compute instances, particularly for online learning components that demand specialized processing power at peak times. [7]

The interplay between model design and streaming infrastructure is a vital aspect of building these pipelines. Models that support incremental or mini-batch updates are favored in streaming scenarios because they seamlessly integrate with the flow of arriving data. Conversely, batch-oriented models require chunked reprocessing of data subsets, which may introduce lag and potentially undermine the overall real-time objective [8]. The design also considers approximate algorithms that provide rapid estimations and can handle the inherently noisy environment of streaming data. These algorithms must be carefully analyzed to guarantee acceptable error bounds while preserving speed.

The remainder of this work examines the architecture of a real-time machine learning pipeline in a cloud setting, with a particular focus on Apache Kafka for handling high-throughput data streams. The discussion introduces advanced mathematical concepts to capture the behavior of streaming algorithms, building on the fundamentals of large-scale distributed systems [9]. Attention is given to strategies for partition management, synchronization, incremental updates, and the role of probabilistic data structures. Implementation details reveal how to integrate streaming ingestion with cloud-native deployment practices, while advanced mechanisms for replication and state management ensure reliability. Collectively, these components form a cohesive solution that addresses the challenges inherent in streaming machine learning pipelines at scale. [10]

System Architecture and Dataflow

The creation of a comprehensive pipeline for real-time machine learning in the cloud begins with a well-structured system architecture that streamlines dataflow. The architecture often includes a data producer layer, a messaging and ingestion layer, and multiple downstream consumers that handle feature extraction, model training, and prediction. The message broker sits at the heart of this pipeline, distributing data among multiple consumers and ensuring consistent load distribution. [11], [12]

The initial stage involves producers, which might include IoT sensors, transactional systems, or monitoring tools that continuously generate data in structured or unstructured formats. These producers push messages containing raw data to topics in a high-performance messaging system. A critical design goal is to minimize the overhead for producers by leveraging efficient data serialization and ensuring that publish operations to the messaging system remain asynchronous, with minimal blocking. This approach allows producers to sustain high throughput while delivering messages to the broker. [13]

At the messaging and ingestion layer, Apache Kafka exemplifies a solution that persists messages in a fault-tolerant manner through replication across a cluster of brokers. The incoming streams are divided into partitions, each of which can be handled by a different broker. The selection of partition keys can influence the distribution of data and thereby impact load balancing [14], [15]. The data, once stored, can be pulled or streamed by consumers that execute different tasks in the pipeline. This separation of producer and consumer roles, combined with efficient disk-based persistence, yields a robust queueing mechanism that can buffer massive volumes of data with low overhead.

Downstream consumers include modules for feature extraction, transformation, and eventually model training and inference. Each consumer can be scaled independently according to its computational needs [16]. The asynchronous nature of consumption allows for variable processing times, ensuring that stragglers do not stall the entire pipeline. In scenarios where real-time feedback is crucial, a low-latency path can be established for priority data, or specific partitions can be reserved for expedited processing.

Dataflow within this pipeline hinges on distributing tasks logically among multiple computing nodes [17]. To handle high throughput, the system can be configured such that different partitions of the same Kafka topic feed separate consumer instances in parallel. This design requires careful synchronization mechanisms and offset management to ensure the stateful nature of the learning algorithms remains consistent. Failure recovery

must be transparent, and new consumer instances or brokers should be able to resume operations without losing intermediate states or duplicating processing on messages. [18]

The pipeline also requires a separate channel for sending updated model parameters back into the system. One approach involves producing updated model snapshots to a dedicated topic, which the inference nodes consume. Alternatively, a shared in-memory data store can be used for extremely low-latency deployments. In either case, the system must ensure coherence between the current data being processed and the model version [19]. Overlapping transitions between old and new model versions may be unavoidable, so the architecture should support a gradual switchover to prevent abrupt shifts in predictions.

In large deployments, distributed file systems and key-value stores can serve as persistent storage for model checkpoints, feature embeddings, or aggregated state. By persisting these artifacts, the system can maintain historical context, revert to stable checkpoints in case of anomalies, and facilitate offline batch processing if necessary [20]. However, since the emphasis is on real-time performance, the pipeline must strike a balance between storing sufficient historical data and minimizing the overhead of persistent I/O.

The interaction with external services, such as dashboards or alerting systems, can be managed through additional topics that relay processed data or predictions to front-end applications. This integration ensures that insights derived from continuous data analysis are propagated to the end-users or other systems in real time. Because the pipeline is designed for large-scale scenarios, attention must be paid to message formats and data schemas to ensure forward and backward compatibility, thus allowing components to evolve without incurring version mismatches. [21]

The dataflow described here encapsulates a broad spectrum of moving parts, each essential for a robust real-time machine learning solution. Each partition, consumer, producer, and storage system must be orchestrated to work together seamlessly under fluctuating workloads. The subsequent sections delve into the mathematical underpinnings of streaming algorithms, the detailed implementation steps for the Kafka-based ingestion pipeline, the strategies to mitigate latency and scaling challenges, and the optimization techniques that enable advanced functionality in production settings. [22]

Mathematical Foundations of Streaming Algorithms

A comprehensive understanding of streaming algorithms underpins the effective operation of real-time machine learning pipelines. The perpetual flow of data demands methods that can keep pace with high ingestion rates while updating model parameters incrementally [23]. The constraints of memory, processing time, and the potential for concept drift necessitate mathematical formulations that are both efficient and robust. [24]

The streaming problem can be conceptualized within a probabilistic framework where data arrives as a sequence of observations $x_1, x_2, \ldots, x_t$, potentially from an evolving distribution. A real-time learning system must approximate certain functions of this data, such as frequency moments, correlation structures, or classification boundaries, within stringent resource limits. One fundamental concept in this area is the notion of sketches, which are compact data structures that retain partial information about a data stream. A popular type of sketch is based on hashing, where each incoming data point is mapped to a hash bucket and used to update counters or other accumulators. [25]

To formalize this, consider a frequency estimation scenario where n distinct items appear in a stream of length L. We seek an approximation of the count c_i for each item i such that

$$\max_i |\hat{c}_i - c_i| \leq \varepsilon L$$

with high probability, where $\hat{c}_i$ is the estimate provided by the streaming algorithm. One approach leverages a count-min sketch, which uses d hash functions each mapping an item to one of w buckets, resulting in a table of size $d \times w$ [26], [27]. For an incoming item i, the algorithm updates one counter in each row based on the corresponding hash function. The estimate of the count is given by the minimum among the d counters for item i. Under appropriate parameterizations, this sketch achieves an error bound of εL with probability $1-\delta$, where δ depends on the choice of d and w.

Another class of algorithms addresses the requirement of dimensionality reduction for feature spaces [28]. Techniques such as random projection or matrix factorization can be adapted into a streaming setting. Suppose there is a large-scale data vector $x_t \in \mathbb{R}^d$ arriving at time t. A streaming approach may maintain a matrix M of size $k \times d$ (with $k \ll d$) that is updated incrementally to capture the principal components of the data distribution. One incremental procedure involves performing a rank-k approximation at each step: [29]

$$M_{t+1} = M_t + \alpha_t f(x_t, M_t),$$

where α_t is a learning rate, and $f(\cdot)$ is a function that computes the adjusted gradient or residual based on the new data vector. Over time, this matrix converges to a basis that preserves most of the variance of the high-dimensional data. The memory usage is contained to the size of M, and each update can be computed in $O(kd)$ time, which is often feasible for moderate k. [30]

For classification or regression in streaming contexts, one frequently encounters the online optimization paradigm. Suppose the model is represented by parameters θ. At time t, the algorithm observes (x_t, y_t), where y_t might be a label or a continuous target. The online

update follows a rule such as: [31]

$$\theta_{t+1} = \theta_t - \eta_t \nabla \ell(\theta_t; x_t, y_t),$$

where ℓ is a loss function and η_t is the step size. This stochastic gradient descent approach ensures that memory usage is proportional to the parameter space, rather than the entire dataset. Convergence analyses typically rely on martingale properties of the gradient updates, and they often come with bounds that relate the regret over the data stream to the model's capacity and the smoothness of ℓ. [32]

In many real-time pipelines, concept drift occurs when the underlying distribution shifts over time, leading to outdated model parameters if the updates are not sufficiently responsive. One mathematical strategy for handling drift is to introduce a forgetting factor $\lambda \in (0, 1)$ in the updates:

$$\theta_{t+1} = \lambda \theta_t - \eta_t \nabla \ell(\theta_t; x_t, y_t).$$

This factor gradually diminishes the impact of older data, allowing the model to focus on recent observations [33]. An alternative technique is to use sliding windows, where older samples are discarded after a fixed period, though this can introduce discontinuities.

Probabilistic data structures and approximation techniques often play a central role in these pipelines, helping manage large-scale computations with minimal overhead. For instance, a Bloom filter can be employed to detect the presence or absence of items in a stream with a known probability of false positives. This approach can be integrated into data cleansing pipelines, allowing real-time machine learning modules to skip repeated checks on whether incoming data has already been processed. [34]

The interplay of these mathematical tools forms the backbone of efficient streaming pipelines. Each technique must be carefully tuned to the volume and velocity of data, the desired accuracy, and the system's computational constraints. Subsequent sections will detail how these foundations integrate with the practicalities of implementing real-time data ingestion and model updating on messaging systems like Apache Kafka, as well as how latency and resource allocation challenges can be addressed through cloud-native architectures.

Implementation Aspects in Apache Kafka

The practical implementation of real-time machine learning pipelines often leverages Apache Kafka as a key messaging backbone. Its distributed, fault-tolerant architecture, combined with robust handling of high-throughput data streams, makes it suitable for large-scale solutions in the cloud. Configuring Kafka properly and integrating it with upstream producers and downstream consumers pose several unique considerations tied to performance, reliability, and ease of management.

Kafka clusters typically consist of multiple brokers, each responsible for storing and serving partitions of various topics [35]. Each topic partition is replicated across brokers to achieve fault tolerance. If the leader for a partition fails, one of the replicas is elected as the new leader with minimal downtime. This resilience is crucial for ensuring that real-time pipelines do not lose data during transient failures, although it introduces overhead related to replication and synchronization [36]. Tuning the replication factor, in-sync replica criteria, and leader election timeouts helps strike a balance between reliability and efficiency.

Producers connect to one or more brokers, dispatching data to specific topics and partitions based on configuration or hashing logic. In many implementations, the partition assignment function is deterministic to ensure that messages with the same key land in the same partition, facilitating stateful processing [37]. For machine learning tasks requiring partition-specific updates, this can be leveraged to maintain partial aggregates or parameters associated with particular keys. Producer optimizations include controlling the batch size, compression algorithm, and acknowledgement settings to minimize network overhead and ensure timely delivery. The latency-throughput trade-off is managed by adjusting how aggressively producers wait for broker confirmations before sending new messages.

Consumers, often running within streaming frameworks, subscribe to topics and pull messages in batches [38]. Each consumer group tracks offsets to mark how far it has progressed in reading from each partition. Failure of a consumer instance triggers a rebalancing event, during which remaining instances redistribute partition assignments. This process must occur seamlessly so that data processing resumes without duplication or missed records [39]. The overhead of rebalancing can become pronounced if the pipeline experiences frequent instance restarts. Proper group management and partition assignment strategies help mitigate disruptions. To maintain stateful machine learning logic, each consumer instance may keep local caches or partial model parameters that relate to the data in its assigned partitions.

Integration with a streaming processing layer adds further complexity [40]. This layer can be a standalone application or embedded in frameworks that automatically handle partition assignments, checkpointing, and state management. The stateful stream processing modules often rely on replicated stores to track intermediate aggregates, making it essential to ensure these stores scale in tandem with the overall data volume. Custom state stores can be designed to hold partial gradients, computed features, or model updates [41]. The frequency and manner of checkpointing this state can have a significant impact on performance and data correctness.

A high-performance design also involves optimizing cluster-wide settings such as the number of partitions per topic. While increasing the number of partitions can improve parallelism, it also heightens overhead during

rebalances and metadata propagation [42]. Balancing these factors requires empirical testing under anticipated workload conditions. The trade-off between consumer lag (how far behind the head of the stream each consumer is) and resource usage is often monitored closely. If consumers fall too far behind, the entire pipeline becomes less real time.

Securing data in transit and at rest is another important aspect [43]. Kafka supports encryption of data over the wire, requiring certificate management for brokers and clients. Authentication and authorization mechanisms determine which producers and consumers can write to or read from specific topics. In a multi-tenant environment, these controls ensure isolation of data and compliance with regulatory requirements [44]. Nonetheless, cryptographic operations introduce computational overhead, which must be factored into the design of a low-latency machine learning pipeline.

On the cloud infrastructure side, deploying Kafka on managed services or using container orchestration platforms requires an additional layer of configuration management. Automatic scaling policies, environment variable injection for partition counts, and dynamic configuration of brokers can simplify the process, but also introduce more moving parts that may affect stability if not carefully orchestrated. Network latency between brokers and consumers can fluctuate in certain cloud regions, impacting throughput [45]. Reducing cross-region traffic or placing critical services in proximity to the Kafka cluster is one way to mitigate this issue.

The final objective is to produce a reliable and flexible messaging layer that seamlessly integrates with both data producers and consumers engaged in complex machine learning tasks. Kafka's distributed design and partition-based architecture provide a robust substrate for real-time analytics, though it requires thorough understanding of configuration, tuning, and operational best practices [46]. The pipeline's responsiveness to dynamic loads relies on the synergy between these configurations and the mathematical principles discussed earlier. In the next section, the focus shifts to examining how to mitigate latency and maintain scalability when faced with unpredictable workloads and diverse data streams.

Handling Latency and Scalability Challenges

Real-time pipelines in cloud environments must address latency constraints while concurrently scaling to accommodate unpredictable surges in data volume [47], [48]. The intricacies of distributed systems exacerbate this task, since network partitions, hardware failures, and load imbalances can degrade performance. Analyzing and mitigating these effects involves leveraging queueing theory and control strategies to maintain stable system behavior under varying conditions.

A key performance metric is the end-to-end latency between data production and the time at which a machine learning prediction or decision is available for consumption. This latency is governed by several factors, including network delays, message serialization overhead, and the time required for data processing [49]. In mathematical terms, consider a pipeline as a sequence of stages $S_1, S_2, \ldots, S_m$. Each stage S_i can be modeled with a service rate μ_i and an arrival rate λ_i. Under conditions of steady-state flow, one seeks [50]

$$\lambda_i < \mu_i \quad \text{for all } i,$$

to ensure the queue for each stage remains stable. If λ_i approaches μ_i, backlogs form and latency grows exponentially. Thus, thorough capacity planning and architectural design are essential to guarantee each stage can handle the incoming rate.

Load balancing strategies often center on partitioning data such that each processing node receives approximately equal volumes [51]. This approach entails hashing on keys or round-robin partition assignments. Non-uniform key distributions can cause skew, overwhelming a subset of partitions. A solution is to apply dynamic rebalancing or leveraged partition expansion at runtime [52]. In such a scenario, new partitions can be introduced, and existing data is gradually reallocated. The overhead of rebalancing is not negligible, but if done predictively before high loads reach critical thresholds, it can mitigate latencies effectively.

When analyzing consumer-side processing, one can utilize concurrency [53]. If each consumer instance processes messages in parallel threads, it may reduce processing times per message, although concurrency introduces synchronization overhead. A theoretical assessment can be performed by modeling each consumer instance as having multiple servers, each with its own service rate. The relationship between concurrency level, system utilization, and latency can be examined by extending single-server queueing models to multi-server settings. Fine-tuning concurrency to ensure that the throughput matches or slightly exceeds the arrival rate yields stable performance without excessive overhead. [54]

In addition to balancing load across consumers, the pipeline must handle potential bottlenecks in data flow. For example, if a certain feature extraction module requires more computation than others, it can degrade overall throughput. Methods such as micro-batching, incremental checkpointing, and distributed caching of intermediate results can alleviate these issues [55], [56]. However, each technique trades off memory usage, processing latency, and fault tolerance guarantees.

Scalability in a cloud environment is frequently tied to autoscaling policies. The pipeline can be monitored by collecting metrics on resource usage, consumer lag, and system throughput [57]. Automated rules can trigger the provisioning or deprovisioning of instances to match current loads. This elasticity is one of the advantages

of operating in a virtualized cloud environment, yet it must be carefully tuned. Overly aggressive scaling can lead to oscillations, in which nodes are repeatedly added and removed. A stable policy will often incorporate buffer thresholds and smoothing mechanisms to avoid quick changes in capacity when short-term bursts occur. [58]

Cloud networking introduces further nuances. The latency between a Kafka broker and a consumer can vary if they are placed in different availability zones or regions. The pipeline must be architected to minimize cross-region traffic or to replicate data regionally if latency-sensitive tasks must remain close to producers [59]. Additional concerns include ephemeral IP addresses, container restarts, and the need to maintain service discovery entries. Balancing these factors is crucial for ensuring that partition leadership remains collocated with appropriate consumers where possible, thus reducing network hop time.

Another challenge arises from maintaining real-time model updates alongside streaming data ingestion. When new parameters for an online model are published, there is a synchronization event that updates the inference nodes [60]. If updates occur very frequently, one can observe short bursts of increased latency while model refreshes propagate. One technique to mitigate this issue is to apply asynchronous updates, whereby inference nodes fetch the newest parameters at a controlled interval. In a more advanced scenario, a multi-version concurrency approach is employed, where different consumers may temporarily use different model versions until all are in sync [61]. This approach ensures that inference does not stall for parameter updates.

Understanding latency and scalability from a theoretical perspective informs practical solutions. The interplay of queueing models, concurrency strategies, network topology, and dynamic resource management offers multiple avenues for performance optimization [62]. The next section discusses advanced optimization strategies, integrating approximate algorithms, optimized data structures, and additional orchestration approaches to further refine the pipeline's operation for demanding production environments.

Advanced Optimization Strategies

Achieving high performance at scale necessitates a range of optimization strategies that address both algorithmic and infrastructural concerns. These strategies often involve approximate computations, specialized data structures, and refined orchestration mechanisms. The methods introduced here deepen the discussion of how real-time pipelines can be tuned to reach operational excellence in large-scale cloud deployments. [63]

Approximate algorithms can dramatically reduce computation times and memory overhead. In high-frequency data streams, it may be acceptable to introduce a small, probabilistic error if it significantly reduces resource consumption. Sketch-based algorithms serve this purpose in estimating frequency moments, set intersections, or quantiles [64]. This computational model implies a trade-off between accuracy and computational resource usage. When deploying these in production, a sophisticated balancing technique ensures that the error bounds remain within acceptable limits for the downstream machine learning tasks. Calibration experiments can characterize the relationship between approximation level and downstream model performance.

Model compression is another optimization strategy that facilitates real-time inference, especially when dealing with deep learning or large ensembles [65]. Techniques such as low-rank factorization, weight pruning, or quantization convert a large, high-precision model into a compact version that is faster to load, evaluate, and update. Although these compressed models introduce some level of approximation, they often retain most predictive power if the compression is executed judiciously. In a streaming context, an incremental approach to model compression can be employed, where parts of the model are compressed over time without halting real-time processing. [66]

Memory optimization in streaming systems relies heavily on data structures tailored to handle large volumes. Caches for partial results, intermediate feature representations, or parameter segments can be stored in specialized memory regions using lock-free data structures. By eliminating locking, concurrency overhead is reduced [67]. However, one must remain vigilant about concurrency hazards such as lost updates. Additional concurrency control methods like versioning or advanced compare-and-swap operations might be integrated to ensure consistent updates with minimal blocking.

Resource allocation strategies become more nuanced in the context of advanced optimization. Instead of statically allocating the same resources for each partition or consumer, the pipeline can dynamically assign more powerful compute nodes to topics exhibiting higher load [68]. If a particular partition corresponds to a high-traffic region of the data, scheduling it on a more capable server ensures latency remains within acceptable bounds. Conversely, lower-load partitions might share a single machine. This dynamic provisioning is orchestrated through cluster managers that monitor metrics and automatically redeploy Kafka brokers, consumers, or associated services. [69]

Scheduling and resource arbitration extend beyond simple CPU or memory considerations. In some cases, hardware acceleration via GPUs or specialized inference accelerators can be introduced. The data pipeline might route certain partitions or specific model processing tasks to nodes equipped with accelerators, significantly reducing inference times. To enable effective scheduling, an analysis that accounts for the overhead of data transfer to and from these accelerators is necessary [70]. Balancing

the gains in computation speed with potential increases in I/O latency ensures the pipeline remains efficient.

Model updates can also be optimized by employing a distributed parameter server. In this paradigm, model parameters are sharded across multiple nodes [71]. Each consumer or trainer node fetches and updates its relevant partition of parameters. The incremental updates can be aggregated in an asynchronous manner, reducing the need for global synchronization. The average or a more sophisticated merge function across parameter shards leads to a globally updated model state [72]–[74]. The mathematics behind these merges may involve consensus algorithms that ensure consistency even when partial updates fail or arrive out of order.

Increasingly, reinforcement learning based approaches to scheduling and resource management are appearing in real-time pipelines [75]. A streaming system might treat each partition assignment or machine scheduling decision as an action and measure latency or throughput as a reward. Over time, the system adjusts its strategy to maximize overall performance [74]. Although training a reinforcement learning model in a live production environment introduces complexities, it may discover non-trivial scheduling policies that outperform hand-tuned heuristics.

The final aspect of advanced optimization involves continuous monitoring and adaptation. Observability in real-time pipelines relies on detailed instrumentation for each component [76]. Metrics on throughput, consumer lag, model accuracy, error rates, memory consumption, and network latency feed into a centralized analytics system. If anomalies are detected, corrective measures can be automatically triggered, such as reassigning partitions, spinning up additional processing nodes, or adjusting model parameters. Over time, historical performance data can guide strategic improvements in the pipeline architecture.

These advanced optimization strategies underscore the complexity of running real-time machine learning pipelines at scale [77]. The interplay of approximate algorithms, resource management, compressed models, and dynamic scheduling fosters a highly adaptive system. The next section concludes this examination by summarizing key insights and outlining directions for further exploration in the realm of real-time streaming analytics and machine learning in cloud environments.

Conclusion

Real-time machine learning pipelines for big data in cloud environments involve intricate interactions among data ingestion, online model updates, scalable messaging infrastructures, and advanced streaming algorithms [78]. Central to these pipelines is the capacity to handle vast volumes of continuously arriving data with minimal latency, ensuring that insights and predictions remain relevant in rapidly changing contexts. The reliance on frameworks like Apache Kafka, which provide durable and high-throughput message brokers, enables partition-based distribution, fault tolerance, and dynamic elasticity that align well with the demands of cloud-scale operations.

The mathematical foundations of streaming algorithms offer principled approaches to incremental learning, approximate data structures, and probabilistic methods [79]. These techniques collectively address challenges such as concept drift, limited memory, and the need to maintain real-time responsiveness. When integrated with Kafka-based ingestion pipelines, they allow for efficient updates of model parameters and controlled error bounds on estimates, fostering robust analytics in complex, distributed settings. Achieving low latency requires close attention to load balancing, concurrency, and partition management, often guided by queueing theory and advanced strategies for autoscaling. The resilience against hardware and network failures is preserved through replication, coordination protocols, and snapshot-based recovery, keeping the pipeline reliable under diverse operational conditions.

Optimization strategies go further by incorporating approximate computations, compressed models, specialized memory structures, and dynamic resource orchestration. By carefully calibrating these elements, a pipeline can accommodate steep surges in traffic without sacrificing accuracy or throughput. The synergy of real-time data streams, online optimization, and cloud-native deployments unlocks innovative opportunities for applications requiring immediate insight [80]. Modern reinforcement learning-based scheduling may further refine resource allocation, offering a glimpse of automated, adaptive pipelines that can respond intelligently to changing workloads and error distributions.

Real-time machine learning in large-scale cloud environments continues to evolve. Future directions may emphasize deeper integration with specialized hardware accelerators, more sophisticated multi-region replication for truly global streaming, and refined algorithms that unify batch and streaming paradigms in hybrid workflows. Nonetheless, the principles explored here—system architecture, robust mathematical foundations, optimized implementation details, and comprehensive optimization—provide a cohesive framework for designing, deploying, and operating real-time machine learning pipelines that can scale to meet emerging demands in data-driven applications. [81]

Conflict of interest

Authors state no conflict of interest.

References

[1] P. Avesani, B. McPherson, S. Hayashi, *et al.*, "The open diffusion data derivatives, brain data upcycling via integrated publishing of derivatives and reproducible open cloud services," *Scientific data*, vol. 6, no. 1, pp. 69–69, May 23, 2019. DOI: 10.1038/s41597-019-0073-y.

[2] L. Ding, "Multimodal transport information sharing platform with mixed time window constraints based on big data," *Journal of Cloud Computing*, vol. 9, no. 1, pp. 1–11, Feb. 7, 2020. DOI: 10.1186/s13677-020-0153-8.

[3] J. Liu and Y. Chen, "Segmented in-advance data analytics for fast scientific discovery," *IEEE Transactions on Cloud Computing*, vol. 8, no. 2, pp. 432–442, Apr. 1, 2020. DOI: 10.1109/tcc.2016.2541142.

[4] C. Hogendorn and B. M. Frischmann, "Infrastructure and general purpose technologies: A technology flow framework," *European Journal of Law and Economics*, vol. 50, no. 3, pp. 469–488, Feb. 18, 2020. DOI: 10.1007/s10657-020-09642-w.

[5] I. Fajjari, F. A. Tobagi, and Y. Takahashi, "Cloud edge computing in the iot," *Annals of Telecommunications*, vol. 73, no. 7, pp. 413–414, Aug. 3, 2018. DOI: 10.1007/s12243-018-0651-6.

[6] M. Kotliar, A. V. Kartashov, and A. Barski, "Cwl-airflow: A lightweight pipeline manager supporting common workflow language.," *GigaScience*, vol. 8, no. 7, Jul. 1, 2019. DOI: 10.1093/gigascience/giz084.

[7] H. Malik and E. M. Shakshuki, "Performance evaluation of counter selection techniques to detect discontinuity in large-scale-systems," *Journal of Ambient Intelligence and Humanized Computing*, vol. 9, no. 1, pp. 43–59, Jul. 4, 2017. DOI: 10.1007/s12652-017-0525-1.

[8] A. P. Carrieri, W. P. M. Rowe, M. Winn, and E. O. Pyzer-Knapp, "Aaai - a fast machine learning workflow for rapid phenotype prediction from whole shotgun metagenomes," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 9434–9439, Jul. 17, 2019. DOI: 10.1609/aaai.v33i01.33019434.

[9] M. M. Babu, M. Rahman, A. Alam, and B. L. Dey, "Exploring big data-driven innovation in the manufacturing sector: Evidence from uk firms.," *Annals of operations research*, vol. 333, no. 2-3, pp. 1–28, Apr. 21, 2021. DOI: 10.1007/s10479-021-04077-1.

[10] S. Joshi, S. Mittal, P. H. Holloway, P. R. Shukla, B. Ó. Gallachóir, and J. Glynn, "High resolution global spatiotemporal assessment of rooftop solar photovoltaics potential for renewable electricity generation.," *Nature communications*, vol. 12, no. 1, pp. 1–15, Oct. 5, 2021. DOI: 10.1038/s41467-021-25720-2.

[11] J. Woo and M. Mishra, "Predicting the ratings of amazon products using big data," *WIREs Data Mining and Knowledge Discovery*, vol. 11, no. 3, Dec. 12, 2020. DOI: 10.1002/widm.1400.

[12] R. Avula, "Overcoming data silos in healthcare with strategies for enhancing integration and interoperability to improve clinical and operational efficiency," *Journal of Advanced Analytics in Healthcare Management*, vol. 4, no. 10, pp. 26–44, 2020.

[13] M. Ryan, J. Antoniou, L. Brooks, T. Jiya, K. Macnish, and B. C. Stahl, "Research and practice of ai ethics: A case study approach juxtaposing academic discourse with organisational reality," *Science and engineering ethics*, vol. 27, no. 2, pp. 16–16, Mar. 8, 2021. DOI: 10.1007/s11948-021-00293-x.

[14] Z. Yang, W. Wang, Y. Huang, and X. Li, "A multi-grained log auditing scheme for cloud data confidentiality," *Mobile Networks and Applications*, vol. 26, no. 2, pp. 842–850, Aug. 14, 2019. DOI: 10.1007/s11036-019-01328-1.

[15] S. Shekhar, "An in-depth analysis of intelligent data migration strategies from oracle relational databases to hadoop ecosystems: Opportunities and challenges," *International Journal of Applied Machine Learning and Computational Intelligence*, vol. 10, no. 2, pp. 1–24, 2020.

[16] D. S. W. Ting, D. V. Gunasekeran, L. Wickham, and T. Y. Wong, "Next generation telemedicine platforms to screen and triage.," *The British journal of ophthalmology*, vol. 104, no. 3, pp. 299–300, Dec. 3, 2019. DOI: 10.1136/bjophthalmol-2019-315066.

[17] J. Grandinetti, "Welcome to a new generation of entertainment: Amazon web services and the normalization of big data analytics and rfid tracking," *Surveillance & Society*, vol. 17, no. 1/2, pp. 169–175, Mar. 31, 2019. DOI: 10.24908/ss.v17i1/2.12919.

[18] Z. Li, K. Lin, S. Cheng, L. Yu, and J. Qian, "Energy-efficient and load-aware vm placement in cloud data centers," *Journal of Grid Computing*, vol. 20, no. 4, Nov. 24, 2022. DOI: 10.1007/s10723-022-09631-0.

[19] Z. Cai, L. Deng, D. Li, X. Yao, and H. Wang, "Retraction note to: A fcm cluster: Cloud networking model for intelligent transportation in the city of macau," *Cluster Computing*, vol. 24, no. 1, pp. 587–587, Feb. 3, 2021. DOI: 10.1007/s10586-021-03250-2.

[20] C.-O. Truica, E. Apostol, J. Darmont, and I. Assent, "Textbends: A generic textual data benchmark for distributed systems," *Information Systems Frontiers*, vol. 23, no. 1, pp. 81–100, Mar. 6, 2020. DOI: 10.1007/s10796-020-09999-y.

[21] M. J. Heaton, A. Datta, A. O. Finley, *et al.*, "A case study competition among methods for analyzing large spatial data," *Journal of agricultural, biological, and environmental statistics*, vol. 24, no. 3, pp. 398–425, Dec. 14, 2018. DOI: 10.1007/s13253-018-00348-w.

[22] G. Zu, S. Wei, Y. Yao, H. Liu, H. Liang, and D. Ji, "Design of online monitoring system for distribution transformer based on cloud side end collaboration of internet of things," *International Journal of Wireless Information Networks*, vol. 28, no. 3, pp. 276–286, Jun. 29, 2021. DOI: 10.1007/s10776-021-00521-y.

[23] R. Avula, "Optimizing data quality in electronic medical records: Addressing fragmentation, inconsistencies, and data integrity issues in healthcare," *Journal of Big-Data Analytics and Cloud Computing*, vol. 4, no. 5, pp. 1–25, 2019.

[24] M. Obschonka and D. B. Audretsch, "Artificial intelligence and big data in entrepreneurship: A new era has begun," *Small Business Economics*, vol. 55, no. 3, pp. 529–539, Jun. 6, 2019. DOI: 10.1007/s11187-019-00202-4.

[25] P. G. Boyd, Y. J. Lee, and B. Smit, "Computational development of the nanoporous materials genome," *Nature Reviews Materials*, vol. 2, no. 8, pp. 17 037–, Jul. 4, 2017. DOI: 10.1038/natrevmats.2017.37.

[26] S. Shahzadi, M. Iqbal, T. Dagiuklas, and Z. U. Qayyum, "Multi-access edge computing: Open issues, challenges and future perspectives," *Journal of Cloud Computing*, vol. 6, no. 1, pp. 30–, Dec. 21, 2017. DOI: 10.1186/s13677-017-0097-9.

[27] M. Kansara, "Cloud migration strategies and challenges in highly regulated and data-intensive industries: A technical perspective," *International Journal of Applied Machine Learning and Computational Intelligence*, vol. 11, no. 12, pp. 78–121, 2021.

[28] J. Yan, D. Wu, C. Zhang, H. Wang, and R. Wang, "Socially aware d2d cooperative communications for enhancing internet of things application," *EURASIP Journal on Wireless Communications and Networking*, vol. 2018, no. 1, pp. 1–12, May 25, 2018. DOI: 10.1186/s13638-018-1127-0.

[29] J. M. Johnson and T. M. Khoshgoftaar, "Survey on deep learning with class imbalance," *Journal of Big Data*, vol. 6, no. 1, pp. 1–54, Mar. 19, 2019. DOI: 10.1186/s40537-019-0192-5.

[30] M. Demertzis, S. Merler, and G. B. Wolff, "Capital markets union and the fintech opportunity," *Journal of Financial Regulation*, vol. 4, no. 1, pp. 157–165, Jan. 19, 2018. DOI: 10.1093/jfr/fjx012.

[31] H. F. Atlam, R. J. Walters, and G. Wills, "Fog computing and the internet of things: A review," *Big Data and Cognitive Computing*, vol. 2, no. 2, pp. 10–, Apr. 8, 2018. DOI: 10.3390/bdcc2020010.

[32] R. Towe, G. Dean, L. Edwards, *et al.*, "Rethinking data-driven decision support in flood risk management for a big data age," *Journal of Flood Risk Management*, vol. 13, no. 4, Aug. 11, 2020. DOI: 10.1111/jfr3.12652.

[33] B. Qolomany, I. Mohammed, A. Al-Fuqaha, M. Guizani, and J. Qadir, "Trust-based cloud machine learning model selection for industrial iot and smart city services," *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2943–2958, Feb. 15, 2021. DOI: 10.1109/jiot.2020.3022323.

[34] Y. Zhang, X. Ma, S. Wan, H. Abbas, and M. Guizani, "Crossrec: Cross-domain recommendations based on social big data and cognitive computing," *Mobile Networks and Applications*, vol. 23, no. 6, pp. 1610–1623, Aug. 29, 2018. DOI: 10.1007/s11036-018-1112-1.

[35] B. Sendir, M. Govindaraju, R. Odaira, and P. Hofstee, "Low latency and high throughput write-ahead logging using capi-flash," *IEEE Transactions on Cloud Computing*, vol. 9, no. 3, pp. 1129–1142, Jul. 1, 2021. DOI: 10.1109/tcc.2019.2906613.

[36] T. Ahmed, S. Rahman, M. Tornatore, K. Kim, and B. Mukherjee, "A survey on high-precision time synchronization techniques for optical datacenter networks and a zero-overhead microsecond-accuracy solution," *Photonic Network Communications*, vol. 36, no. 1, pp. 56–67, May 24, 2018. DOI: 10.1007/s11107-018-0773-9.

[37] Y. Zhu, M. Interlandi, A. Roy, *et al.*, "Phoebe: A learning-based checkpoint optimizer.," *Proceedings of the VLDB Endowment*, vol. 14, no. 11, pp. 2505–2518, Oct. 27, 2021. DOI: 10.14778/3476249.3476298.

[38] B. Balducci and D. Marinova, "Unstructured data in marketing," *Journal of the Academy of Marketing Science*, vol. 46, no. 4, pp. 557–590, Jun. 12, 2018. DOI: 10.1007/s11747-018-0581-x.

[39] M. S. Carolan, "Acting like an algorithm: Digital farming platforms and the trajectories they (need not) lock-in," *Agriculture and Human Values*, vol. 37, no. 4, pp. 1041–1053, Apr. 13, 2020. DOI: 10.1007/s10460-020-10032-w.

[40] D. Ojika, A. Gordon-Ross, H. Lam, and B. Patel, "Faam: Fpga-as-a-microservice - a case study for data compression," *EPJ Web of Conferences*, vol. 214, pp. 07 029–, Sep. 17, 2019. DOI: 10.1051/epjconf/201921407029.

[41] P. Kathiravelu, A. Sharma, H. Galhardas, P. V. Roy, and L. Veiga, "On-demand big data integration: A hybrid etl approach for reproducible scientific research.," *Distributed and Parallel Databases*, vol. 37, no. 2, pp. 273–295, Sep. 1, 2018. DOI: 10.1007/s10619-018-7248-y.

[42] K. Neshatpour, M. Malik, A. Sasan, S. Rafatirad, and H. Homayoun, "Hardware accelerated mappers for hadoop mapreduce streaming," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4, no. 4, pp. 734–748, Oct. 1, 2018. DOI: 10.1109/tmscs.2018.2854787.

[43] N. Yamanaka, S. Okamoto, M. Hirono, *et al.*, "Application-triggered automatic distributed cloud/network resource coordination by optically networked inter/intra data center [invited]," *Journal of Optical Communications and Networking*, vol. 10, no. 7, pp. 15–24, May 17, 2018. DOI: 10.1364/jocn.10.000b15.

[44] A. Kocheturov, P. M. Pardalos, and A. Karakitsiou, "Massive datasets and machine learning for computational biomedicine: Trends and challenges," *Annals of Operations Research*, vol. 276, no. 1, pp. 5–34, May 15, 2018. DOI: 10.1007/s10479-018-2891-2.

[45] A. Rezgui, N. Davis, Z. Malik, B. Medjahed, and H. Soliman, "Cloudfinder: A system for processing big data workloads on volunteered federated clouds," *IEEE Transactions on Big Data*, vol. 6, no. 2, pp. 347–358, Jun. 1, 2020. DOI: 10.1109/tbdata.2017.2703830.

[46] U. Paščinski, J. Trnkoczy, V. Stankovski, M. Cigale, and S. Gec, "Qos-aware orchestration of network intensive software utilities within software defined data centres: An architecture and implementation of a global cluster manager," *Journal of Grid Computing*, vol. 16, no. 1, pp. 85–112, Nov. 27, 2017. DOI: 10.1007/s10723-017-9415-1.

[47] I. Tanaka, K. Rajan, and C. Wolverton, "Data-centric science for materials innovation," *MRS Bulletin*, vol. 43, no. 9, pp. 659–663, Sep. 10, 2018. DOI: 10.1557/mrs.2018.205.

[48] R. Avula, "Architectural frameworks for big data analytics in patient-centric healthcare systems: Opportunities, challenges, and limitations," *Emerging Trends in Machine Intelligence and Big Data*, vol. 10, no. 3, pp. 13–27, 2018.

[49] M. Li, N. Xiong, Y. Zhang, and Y. Hu, "Priority-mece: A mobile edge cloud ecosystem based on priority tasks offloading," *Mobile Networks and Applications*, vol. 27, no. 4, pp. 1768–1777, Feb. 26, 2022. DOI: 10.1007/s11036-022-01930-w.

[50] R. Agrawal and S. Prabakaran, "Big data in digital healthcare: Lessons learnt and recommendations for general practice," *Heredity*, vol. 124, no. 4, pp. 525–534, Mar. 5, 2020. DOI: 10.1038/s41437-020-0303-2.

[51] D. Reynolds, J. Ball, A. Bauer, R. P. Davey, S. Griffiths, and J. Zhou, "Cropsight: A scalable and open-source information management system for distributed plant phenotyping and iot-based crop management," *GigaScience*, vol. 8, no. 3, Jan. 31, 2019. DOI: 10.1093/gigascience/giz009.

[52] J. W. Woodworth and M. A. Salehi, "S3bd: Secure semantic search over encrypted big data in the cloud," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 11, Dec. 11, 2018. DOI: 10.1002/cpe.5050.

[53] W. Liu, P. Cui, J. K. Nurminen, and J. Wang, "Special issue on intelligent urban computing with big data," *Machine Vision and Applications*, vol. 28, no. 7, pp. 675–677, Sep. 12, 2017. DOI: 10.1007/s00138-017-0877-8.

[54] Y. Wang, J. Li, and H. H. Wang, "Cluster and cloud computing framework for scientific metrology in flow control," *Cluster Computing*, vol. 22, no. 1, pp. 1189–1198, Sep. 21, 2017. DOI: 10.1007/s10586-017-1199-3.

[55] V. de Paul Obade and C. Gaya, "Digital technology dilemma: On unlocking the soil quality index conundrum," *Bioresources and bioprocessing*, vol. 8, no. 1, pp. 6–6, Jan. 10, 2021. DOI: 10.1186/s40643-020-00359-x.

[56] 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.

[57] M. Miller, C. Zhu, and Y. Bromberg, "Clubber: Removing the bioinformatics bottleneck in big data analyses.," *Journal of integrative bioinformatics*, vol. 14, no. 2, pp. 20 170 020–, Jun. 13, 2017. DOI: 10.1515/jib-2017-0020.

[58] C. C. Wall, C. Anderson, A. Spring, and J. Gedamke, "Increasing access to big bioacoustic data through cloud-based systems," *The Journal of the Acoustical Society of America*, vol. 144, no. 3, pp. 1886–1886, Sep. 1, 2018. DOI: 10.1121/1.5068271.

[59] M.-T. Cynthia, P.-L. Ingrid, and Y.-M. Alicia, "Digitization trends in hospitality and tourism," *Smart Tourism*, vol. 2, no. 2, Nov. 1, 2021. DOI: 10.54517/st.v2i2.1709.

[60] S. Gao, S. Newsam, L. Zhao, *et al.*, "Geoai 2019 workshop report: The 3nd acm sigspatial international workshop on geoai: Ai for geographic knowledge discovery: Seattle, wa, usa - november 5, 2019," *SIGSPATIAL Special*, vol. 11, no. 3, pp. 23–24, Feb. 13, 2020. DOI: 10.1145/3383653.3383662.

[61] J. Woo, S.-J. Shin, W. Seo, and P. Meilanitasari, "Developing a big data analytics platform for manufacturing systems: Architecture, method, and implementation," *The International Journal of Advanced Manufacturing Technology*, vol. 99, no. 9, pp. 2193–2217, Jul. 20, 2018. DOI: 10.1007/s00170-018-2416-9.

[62] D. J. Varon, D. J. Jacob, M. Sulprizio, *et al.*, "Integrated methane inversion (imi 1.0): A user-friendly, cloud-based facility for inferring high-resolution methane emissions from tropomi satellite observations," *Geoscientific Model Development*, vol. 15, no. 14, pp. 5787–5805, Jul. 27, 2022. DOI: 10.5194/gmd-15-5787-2022.

[63] S. Sharma, J. Powers, and K. Chen, "Privategraph: Privacy-preserving spectral analysis of encrypted graphs in the cloud," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 5, pp. 981–995, May 1, 2019. DOI: 10.1109/tkde.2018.2847662.

[64] B. Ibrahim, D. P. McMahon, F. Hufsky, *et al.*, "A new era of virus bioinformatics," *Virus research*, vol. 251, pp. 86–90, May 8, 2018. DOI: 10.1016/j.virusres.2018.05.009.

[65] J. C. Jeong, I. Hands, J. M. Kolesar, *et al.*, "Local data commons: The sleeping beauty in the community of data commons.," *BMC bioinformatics*, vol. 23, no. Suppl 12, pp. 386–, Sep. 23, 2022. DOI: 10.1186/s12859-022-04922-5.

[66] Z. Zhang, K. Barbary, F. A. Nothaft, *et al.*, "Kira: Processing astronomy imagery using big data technology," *IEEE Transactions on Big Data*, vol. 6, no. 2, pp. 369–381, Jun. 1, 2020. DOI: 10.1109/tbdata.2016.2599926.

[67] B. B. Gupta, S. Yamaguchi, and D. P. Agrawal, "Advances in security and privacy of multimedia big data in mobile and cloud computing," *Multimedia Tools and Applications*, vol. 77, no. 7, pp. 9203–9208, Nov. 13, 2017. DOI: 10.1007/s11042-017-5301-x.

[68] J. H. Kim, "A review of cyber-physical system research relevant to the emerging it trends: Industry 4.0, iot, big data, and cloud computing," *Journal of Industrial Integration and Management*, vol. 02, no. 03, pp. 1 750 011–, Nov. 30, 2017. DOI: 10.1142/s2424862217500117.

[69] W. Liao, C. Luo, S. Salinas, and P. Li, "Efficient secure outsourcing of large-scale convex separable programming for big data," *IEEE Transactions on Big Data*, vol. 5, no. 3, pp. 368–378, Sep. 1, 2019. DOI: 10.1109/tbdata.2017.2787198.

[70] S. Paul, M. Riffat, A. Yasir, *et al.*, "Industry 4.0 applications for medical/healthcare services," *Journal of Sensor and Actuator Networks*, vol. 10, no. 3, pp. 43–, Jun. 30, 2021. DOI: 10.3390/jsan10030043.

[71] J. Vivian, A. A. Rao, F. A. Nothaft, *et al.*, "Toil enables reproducible, open source, big biomedical data analyses," *Nature biotechnology*, vol. 35, no. 4, pp. 314–316, Apr. 11, 2017. DOI: 10.1038/nbt.3772.

[72] A. Magdy, L. Abdelhafeez, Y. Kang, E. Ong, and M. F. Mokbel, "Microblogs data management: A survey," *The VLDB Journal*, vol. 29, no. 1, pp. 177–216, Sep. 18, 2019. DOI: 10.1007/s00778-019-00569-6.

[73] 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.

[74] W. Cai, J. Zhu, W. Bai, W. Lin, N. Zhou, and K. Li, "A cost saving and load balancing task scheduling model for computational biology in heterogeneous cloud datacenters," *The Journal of Supercomputing*, vol. 76, no. 8, pp. 6113–6139, May 26, 2020. DOI: 10.1007/s11227-020-03305-y.

[75] A. Sharma and K. D. Forbus, "Graph-based reasoning and reinforcement learning for improving q/a performance in large knowledge-based systems," in *2010 AAAI Fall Symposium Series*, 2010.

[76] Y.-Y. Teing, A. Dehghantanha, and K.-K. R. Choo, "Cloudme forensics: A case of big-data investigation.," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 5, Jul. 31, 2017. DOI: 10.1002/cpe.4277.

[77] S. Zaheer, A. W. Malik, A. Rahman, and S. A. Khan, "Locality-aware process placement for parallel and distributed simulation in cloud data centers," *The Journal of Supercomputing*, vol. 75, no. 11, pp. 7723–7745, Aug. 28, 2019. DOI: 10.1007/s11227-019-02973-9.

[78] J. Xu, Y. Shangshu, W. Lu, L. Xu, and D. Yang, "Incentivizing for truth discovery in edge-assisted large-scale mobile crowdsensing.," *Sensors (Basel, Switzerland)*, vol. 20, no. 3, pp. 805–, Feb. 2, 2020. DOI: 10.3390/s20030805.

[79] E. Casalicchio and S. Iannucci, "The state-of-the-art in container technologies: Application, orchestration and security," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 17, Jan. 19, 2020. DOI: 10.1002/cpe.5668.

[80] C.-T. Yang, S.-T. Chen, J.-C. Liu, Y.-W. Chan, C.-C. Chen, and V. K. Verma, "An energy-efficient cloud system with novel dynamic resource allocation methods," *The Journal of Supercomputing*, vol. 75, no. 8, pp. 4408–4429, Mar. 6, 2019. DOI: 10.1007/s11227-019-02794-w.

[81] B. Li, H. Ke, S. Zhou, J. Impagliazzo, and M. Zhang, "Turc - chinese perspectives on it education," *Proceedings of ACM Turing Celebration Conference - China*, pp. 39–46, May 18, 2018. DOI: 10.1145/3210713.3210726.