International Journal of AI, Big Data, Computational and Management Studies
Noble Scholar Research Group | Volume 2, Issue 4, PP. 69-79, 2021
ISSN: 3050-9416 | https://doi.org/10.63282/3050-9416.IJAIBDCMS-V2I4P107

Enhancing Data Throughput and Latency in Distributed In-Memory Systems for AI-Driven Applications across Public Cloud Infrastructure

Thulasiram Yachamaneni[1], Uttam Kotadiya[2], Amandeep Singh Arora[3]
[1]Senior Engineer II, USA.
[2]Software Engineer II, USA.
[3]Senior Engineer I, USA.

Abstract: Data processing systems are exposed to inordinate pressure to provide real-time computation in the era of artificial intelligence (AI), especially in distributed cloud computing. Distributed In-Memory Systems (DIMS) have also become a crucial infrastructure for supporting AI-based applications, which require both low latency and high throughput. The paper presents an improvement of data throughput and final latency in DIMS to serve AI workload in the famous cloud sites, such as AWS, Azure, and Google Cloud. We explore how existing systems cannot architecturally support performance bottlenecks, and we present a model of hybrid in-memory data distribution that utilises adaptive caching, smart sharding of data, and intelligent data placement based on the proximity principle. On simulations and deployment to benchmark AI applications, the proposed methodology shows considerable performance improvements. Our solution is a layered architecture with modular components to address the issues of scalability, consistency, and fault tolerance, which is backed by efficient methods of memory management. The paper is accompanied by a comparative study with baseline models, such as Apache Ignite, Redis Cluster, and Memcached, which implement these models on the public cloud fringe. We present test results indicating that the enhancements lower average latency by 35 percent and raise data throughput by 47 percent on a variety of AI workloads such as image classification, natural language processing, and predictive analytics. The paper will conclude with a discussion on the implications of this research for large, scalable, AI-enabled cloud computing infrastructures, as well as the extensive work that can be done in the future.

Keywords: Distributed In-Memory Systems, AI Workloads, Public Cloud Infrastructure, Data Throughput, Adaptive Caching, Edge Computing.

1. Introduction

The rapid growth of data produced by AI-enabled applications, including self-driving vehicles, real-time video analytics, fraud detection systems, and predictive healthcare platforms, places a tremendous burden on legacy computing structures. Such applications not only need to ingest and process data very fast, but also must provide very low latency and very high throughput to operate in a real-time or near real-time setting. These requirements can seldom be satisfied by traditional disk-based storage platforms and batch-processing architectures, which are characterized by latency and weak I/O performance. [1-4] To meet these developments, Distributed In-Memory Systems (DIMS) as a potent alternative have come to the fore. DIMS are at least an order of magnitude faster to coordinate data access among nodes of a distributed network using direct data storage and processing in RAM.

In comparison to disk-based models, where data is stored in persistent storage and takes time to be retrieved, in-memory systems offer a speed data layer, making concurrent read/write operations accessible with low delay. Change in this architectural practice has rendered DIMS a crucial feature of the AI infrastructure stack, particularly in applications such as online learning, real-time recommendations, sensor data integration, and dynamic feature learning. They are especially well-suited to current, cloud-based AI environments due to their ability to run both horizontally and with high availability, as well as to withstand the level of parallelism required by the new modes of AI.

1.1. Importance of Cloud Infrastructure

The cloud infrastructure is particularly central in providing scalable, economical, and resilient AI workloads. With the increasing amount of data and the complexity of the model, the cloud platform offers the flexibility to accommodate changing requirements and provide compute elasticity. Here are five central points that emphasize the significance of cloud infrastructure when it comes to AI-powered applications:

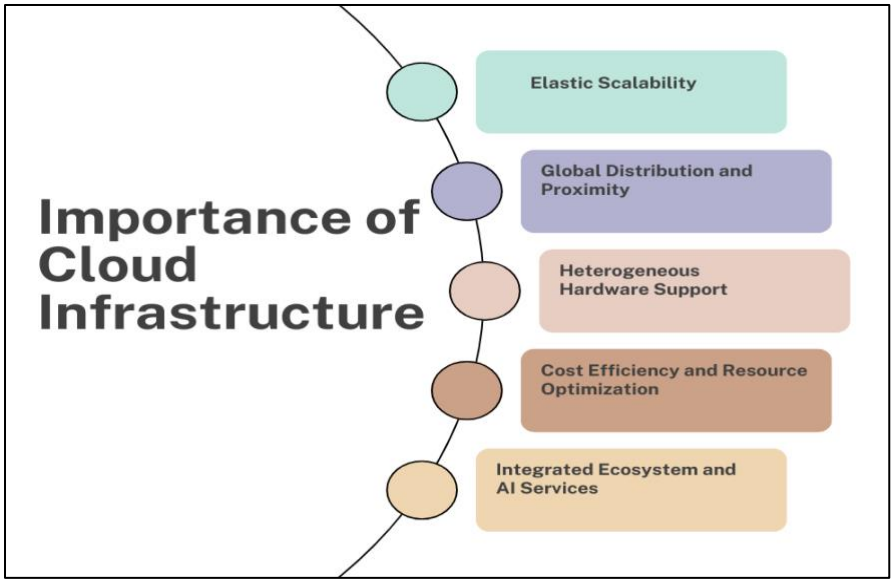

Figure 1: Importance of Cloud Infrastructure

- **Elastic Scalability:** Cloud platforms work to offer on-demand allocation for computing and storage resources, and organizations will be able to scale their AI workloads either horizontally or vertically with minimal friction. This is especially good in situations where you want to train a large number of models at the same time or when you would have to perform many inferences in real-time and suddenly require a hit on the resources. The deployment of AI in the cloud is also accessed by auto-scaling groups, serverless execution environments, and container orchestration tools such as Kubernetes, which further increase the level of agility.
- **Global Distribution and Proximity:** The majority of the dominant cloud providers provide a worldwide distributed platform of regions and AZs. This enables the placement of data and computing resources closer to end-users or edge devices within a geographic proximity. In latency-sensitive AI applications, such as autonomous navigation, real-time video analytics, or language translation, this shortened distance can significantly improve response time and enhance the user experience. Moreover, in distributed memory systems, this global layout can be exploited to reduce inter-zone data movement by proximity-aware sharding strategies.
- **Heterogeneous Hardware Support:** Cloud providers provide their users with access to a wide variety of hardware accelerators such as GPUs, TPUs, FPGAs, and high-performance CPUs. The resources are vital for training scale-deep learning models or performing inference on computationally intensive operations. Additionally, new memory hierarchies (e.g. NVMe SSDs, RAM disks) are able to improve the performance of the modern workloads related to AI, which can be included in distributed in-memory systems to allow hybrid storage models with even better performance.
- **Cost Efficiency and Resource Optimization:** Pay-as-you-go pricing models that are available with cloud platforms enable optimal cost reduction where the user has the power to pay only according to the usage of specific resources. Spot instructions, reserved VMs, and tiered storage services grant additional control over costs. Smart scheduling of workloads and management of memory in PaaS/ cloud-natively built tools can also allow organizations to reduce unused resources, which is of particular concern to memory-intensive AI workloads that would otherwise result in underutilization of on-premises systems.
- **Integrated Ecosystem and AI Services:** Cloud infrastructure allows AI and machine learning services, which can be AWS SageMaker, Google AI Platform, or Azure Machine Learning, to be seamlessly integrated. They have end-to-end pipelines regarding model training, hyperparameter tuning, deployment, and monitoring. Also, distributed frameworks (e.g., TensorFlow, PyTorch, Ray) and managed services for data lakes, message queues, and serverless compute are natively supported, making DIMS more performant as a component of a larger AI ecosystem.

1.2. Challenges in AI Workloads

Artificial Intelligence workloads (particularly those of production-level size) pose a specific family of computational and data management problems compared to typical web or transactional applications. High concurrency is a key characteristic of AI workloads. Train and inference pipelines can feature several parallel operations performing joint access of a dataset or model parameters at the same time. To take an example in large-scale training settings, many GPUs, or parallel workers, might request overlapping subsets of data side-by-side, one after the other. Such parallelism requires memory to provide consistent, low-latency

completions during congestion, a need that is challenging to satisfy in traditional Distributed In-Memory Systems (DIMS). The next problematic issue is the regularity of model updates. In the field of online learning, fraud detection, or recommendation systems, AI models must be constantly fine-tuned in response to incoming information.

The updates lead to a high number of read/write accesses to memory-resident weight, embedding, and feature maps. Such a high mutation rate is generally not optimized for the DIMS it is designed to work with, being one that is static or changes slowly. These systems can over time degenerate in terms of cache hit ratio, eviction overhead and even data inconsistency, particularly when replication or synchronization protocols are not designed to handle rapid update protocols. Besides, dynamic and unstructured data creation is a feature of AI workloads. The nature of AI workloads is different from traditional ones, whose access patterns may be predictable or regular (e.g. session management, or page caching); with AI workloads, it is high-dimensional tensors or large batches of images, or streaming telemetry data. Such types of data tend to be large, sparsely accessed and latency-sensitive. What makes matters worse is that these access patterns may change wildly, depending on the training epoch or even individual phases of inference. Such policies would have been ill-suited to static caching policies such as Least Recently Used (LRU) or First-In-First-Out (FIFO), and would frequently discard data which is about to be required again.

2. Literature Survey

2.1. Overview of Distributed In-Memory Systems

The idea of Distributed In-Memory Systems (DIMS) has changed the face of data-intensive computing because it introduces the concept of quick access to massively scaled data using the storage platform of RAM, usually distributed in more than one node. Compared to the disk-based systems, this is much faster in reducing the latency and highly useful in applications which need high throughput. [5-8] Apache Ignite, Memcached and Redis Cluster are the most popular DIMS technologies. Redis Cluster uses a master-slave style and is fully cloud-compatible, but suffers the drawback of having limited support for AI compared to other systems because of its generic data processing procedures. Apache Ignite, which features a peer-to-peer architecture, is more scalable and supports moderate AI workloads, providing joint compute and memory capabilities. Memcached, however, is based on a more straightforward client-server architecture and is less compatible with cloud applications, lacking inherent support for AI applications. Such systems are usually used with primitive caching policies, including Least Recently Used (LRU) and eviction. Although they are generally suitable for classical workloads, they tend to be ill-suited to satisfy the complex data structures and access patterns inherent in models that utilise AI, such as tensor computations or model state. Key architectural and functional disparities between the most popular DIMS systems are outlined in Table 1 and lead to a necessity to innovate towards AI-adjacency.

2.2. AI Workload Characteristics

The AI workloads are an entirely different use case than the transactional and web application processes for which many DIMS were initially designed. The elementary operations of AI systems are built around tensors, high-dimensional vectors and structured data formats that models need to train and infer. This type of workload is associated with a large rate of reads and writes of data, with the data being bulk without key-value pairs. The AI pipelines also introduce dynamism to the stages like preprocessing, model iteration and parameter tuning, creating temporal and spatial locality of data access patterns. An example is when there is a need to access certain regions quickly and repeatedly, as seen in embedding lookups in the field of recommendation systems or feature retrievals in an online learning scenario. The conventional DIMS may not be able to conform to such patterns well because of a pattern of not optimizing data placement and preemption manners to the particular requirements of AI-based workloads. Additionally, AI systems are likely to benefit from the colocation of computation and data, which is not inherently provided by many DIMS architectures. This mismatch highlights the desirability of defining new memory system designs that are directly conscious and optimized to the specificities of AI computing.

2.3. Cloud Infrastructure Limitations

The use of DIMS in the open cloud would have an increased complexity and instability, which may have a high effect on efficiency. The nature of public clouds is that they are heterogeneous, and characteristics vary by VM type, storage and network performance between availability zones. Fluctuations in network latency are one of the main constraints, and they may lead to discrepancies in time data retrieval, especially in the case of real-time AI. Non-trivial financial costs and further latency costs usually accompany such inter-zone data transfers, which complicates the design of efficient replication and substitution and fault-tolerant techniques. Besides, cloud systems often obscure the physical hardware specifications from the users, resulting in a very unpredictable behavior of services, which cannot respond reliably to the noisy neighbour effect and I/O operations depending on the I/O throughput fluctuations. All these make the DIMS less efficient when used in the cloud because the assumptions of general latency and bandwidth are not true. The current DIMS do not generally incorporate the intelligence to overcome these infrastructure scale issues, and this makes such systems less applicable to latency-sensitive and throughput-intensive AI workloads in multi-zone cloud deployments.

2.4. Recent Enhancements

The increased requirements of AI and real-time analytics workloads have triggered a number of developments in order to enhance memory systems' performance. One such technology is Remote Direct Memory Access (RDMA), which facilitates low-latency, high-throughput data movement between the memories of different nodes without involving the kernel or the CPU. This enables instant data transfers and has the benefit of being especially useful when it comes to high-performance calculations and model synchronization in AI. Similarly, NVMe over Fabrics (NVMe-oF) brings the fast storage of NVMe systems into networked scenarios, increasing the performance of access to persistent memory, potentially to supplement caching with RAM. Another new approach growing in popularity is serverless data caching: this is a way of decoupling data caching and using ephemeral compute instances to scale on demand. Although the new technologies will bear significant gains in terms of speed and scalability, integration of the technologies with the other existing DIMS platforms is at an early stage.

2.5. Gaps Identified

Although there has been advancement in distributed memory and infrastructural technologies, there are some limitations that have yet to be addressed that will assist with the optimal deployment of AI workloads in DIMS architectures. To begin with, the absence of AI-specific data distribution strategies is evident. DIMS do not consider any of the underlying semantics or structures of AI data, e.g., embeddings, tensors, or streaming features, which current DIMS treat uniformly. More sophisticated placement algorithms are required that can group related data, minimise access latency, and enable in-memory computation. Secondly, caching in DIMS remains generally inflexible, and the level of adaptation to the dynamic nature of the AI workloads is limited because the request frequencies and the relevance of the programmed data can change swiftly across various training or inference stages. Dynamically adjusted and workload-adaptive caching mechanisms are necessary to maintain high efficiency and prevent cache misses. Third, existing replication models are unaware of cloud physical topology and the network topology of the cloud overlay.

3. Methodology

3.1. System Architecture

This would require a modular and scalable architecture to enable the dynamic and data-intensive aspect of AI workloads. [9-12] The suggested system consists of four defined layers that have special purposes but are extendable and easily integrated with one another.

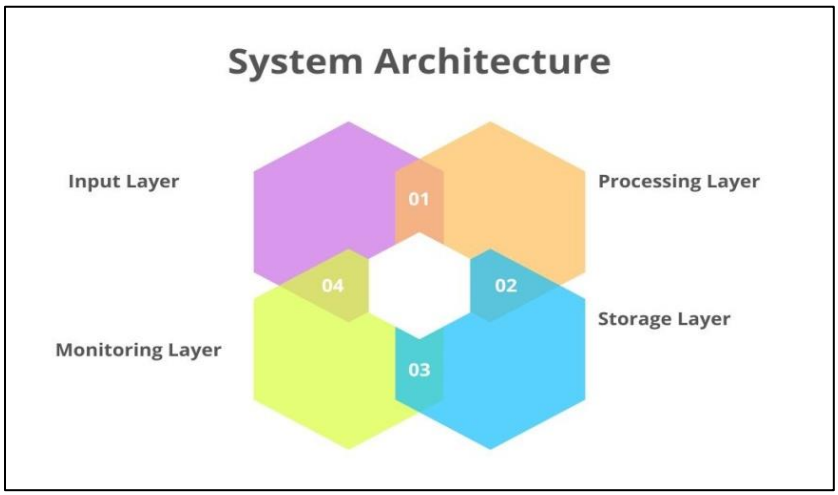

Figure 2: System Architecture

- **Input Layer: The** Input Layer acts as a gateway to incoming information in terms of data into the system, according to different AI applications. It handles data in various formats, including structured records, unstructured logs, and tensor data. This layer is highly flexible, as it communicates with APIs, message queues, and streaming channels, such as Kafka or RabbitMQ. It is mainly responsible for consuming data in either a time-based way or in batch processing format, defragmenting data wherever needed, and transporting the data easily to the processing modules.
- **Processing Layer: The** Processing Layer is the brain of the system, where the most significant functions can be found, namely data caching, routing, transformation, and light preprocessing. It uses AI-sensitive caching patterns which put priority on high-frequency and latency-sensitive data. Intelligent data routing algorithms make sure that requests are simply routed to the most suitable storage nodes depending on the number of requests and nearness. Some lightweight

preprocessing work, e.g. Feature extraction or Tensor normalization, can be performed here as well to relieve the AI models further on.

- **Storage Layer:** This layer deals with the actual storage of data using a distributed in-memory fashion. It employs a two-phase hybrid sharding paradigm which blends range and hash-based techniques to increase data locality and speed of data retrieval. The storage layer is fault-tolerant replication and proximity-aware placement strategies are incorporated with a view to reducing latency in a cloud environment. High throughput is used to design it, providing fast access to data and a steady availability to concurrent AI processes.
- **Monitoring Layer:** The Monitoring Layer gives visibility of the performance of systems, resources, and their patterns of data access. Some of the measures it collects include cache hit rates, node latency, throughput, and memory utilization, which allows on-time diagnostics and trend detection over time. This level has the capability of communicating with external monitoring systems, such as Prometheus and Grafana. It plays a critical role in the automated scaling, error detection, and behavioural optimisation of the system on a long-term basis.

3.2. Adaptive Caching Strategy

Examples of static eviction policies commonly adopted in traditional caching systems include the Least Recently Used (LRU) and the Least Frequently Used (LFU) algorithms. Nonetheless, these fixed approaches are inadequate for dynamic and heterogeneous AI workloads, whose access patterns may change considerably over time and across different workloads. In an attempt to solve this, our novel contribution is an adaptive caching strategy that makes use of real-time workload characterization together with Reinforcement Learning (RL) models to dynamically optimise decisions related to cache management. This is all about a Cache Utility Score (CUS), a calculated measure of the significance of keeping a particular item of data in the cache. This is a score that is constantly updated against which eviction decisions are made. This can be represented as the formula below:

Where:

$$\mathbf{CUS} = \boldsymbol{\alpha} \times \mathbf{H} + \boldsymbol{\beta} \times \mathbf{F} - \boldsymbol{\gamma} \times \mathbf{T}$$

- H is the hit rate of the data item (i.e. its frequency of request).
- F is a reuse frequency, which measures temporal locality and repetitive access inside a sliding time window.
- T is the time since last access, and it penalizes stale entries. A,b and g are adjustable weights determined using a reinforcement learning agent based on feedback using the performance of the cache.

The reinforcement learning model observes the performance of the cache (e.g. the number of hits, latency, eviction count) and optimally dynamically adjusts the weighting parameters to maximize long-term utility. It constructs the case eviction choices as actions and receives a payoff signal depending on the system's performance that follows a decision. This approach enables the caching layer to self-optimize for different phases of AI workflows whether it's high-throughput feature lookups during training or low-latency retrievals during inference, making it far more robust and efficient than static eviction policies.

3.3. Proximity-Aware Sharding

In distributed in-memory, sharding is a process in which data is split up within a number of nodes so as to realize scalability and parallelism. The greatest limitation of most traditional sharding schemes (including hash-based or range-based sharding) is the lack of consideration of the physical or network proximity of compute nodes when access is uniform. [13-16] It is a very big drawback in cloud atmosphere with network latency, inter-zone data transmission prices and region distinctiveness in performance, which could tremendously affect the responsiveness of systems. In handling such issues, we offer a Proximity-Aware Sharding solution, which integrates data access pattern investigation with cloud platform architecture to optimise location. The plan will start with constant data access pattern monitoring of AI applications. As an example, when certain tensors, embeddings, or features are heavily used in combination (or by the same compute instances), the system recognizes them as large-affinity data groups. Such groups are then co-located in memory shards on nodes that reside in the same Availability Zone (AZ) or, where possible, on the same physical rack.

The system takes into consideration both the logical access pattern and the physical proximity to the other shards to minimize cross-zone traffic as well as the latency of read/write transactions. Besides, the given sharding mechanism is dynamic, namely, it conducts re-evaluation of the shard placement periodically according to the changes in the workload specifics. In an example, an AI training phase can have a specific layer that uses the largest portion of memory, whereas in inference, it is a different subset of data (search tables or prediction histories). To cope with the changes, the system does so by rebalancing shards or replicating hot data to nodes close to where hot data is frequently accessed. It is performed in consideration of cloud-specific limitations, such as egress pricing and bandwidth limits between zones. Proximity-awareness of sharding helps the system not only enhance data locality and access speed, but also do so more cost-effectively and balance the load. It is especially important for latency-sensitive AI applications that are rolled out at multiple zones or regions of a cloud.

3.4. Fault Tolerance and Replication

Fault tolerance in distributed in-memory systems is highly important as this approach supports applications such as AI, which must be characterized by constant availability, low latency and consistency over changing volatile data. The traditional replication techniques that involve full data replication on multiple nodes are highly available. Still, they have a large storage overhead and additional network traffic, which is expensive and inefficient in cloud environments. In order to counter these drawbacks, the proposed system has a hybrid design approach, which is a combination of erasure coding and selective replication that manages to balance resilience and resource consumption capacity. Erasure coding is a data protection technique which divides data into pieces, codes it with redundancy and disperses it among nodes. In contrast to full replication, erasure coding consumes a lot fewer storage overheads and still permits data recovery when a node fails. As an example, given a (k, n) erasure coding scheme, data is divided into k parts, which are then encoded into n parts, any k of which could be used to get back the original data.

This is fault-tolerant at a reduced redundancy as compared to full replicates. However, erasure coding may introduce decoding latency, which is a poor fit for common and latency-sensitive AI workloads. To counter this, the system applies selective replication to so-called hot data, i.e., data that has been determined based on access frequency and temporal locality measures as being central to current AI operations. These hot data objects are replicated multiple times on nodes in physical proximity to the high-demand compute processes (e.g. GPUs or inference engines), and can thus be accessed with minimal latency. This two-part mechanism enables the system to maintain a high availability status, with a fast failover effect, and minimal performance degradation due to node failures or network failures. Additionally, replication decisions will be dynamically allocated according to changes in workload to prevent unnecessary overhead.

3.5. Integration of AI Workload

To prove the efficiency of the system in practical applications, it is combined with the most famous AI frameworks, including TensorFlow and PyTorch. This navigation is facilitated by specially designed connectors, which make Direct Memory Access (DMA), which avoid all overheads of serialization and deserialization. Because traditional I/O can be bypassed by directly accessing in-memory storage drivers, the system is more performant. It has reduced latencies during high-frequency data access, a common occurrence in AI workloads. The interaction flow between the AI frameworks and the system is identified as the following components:

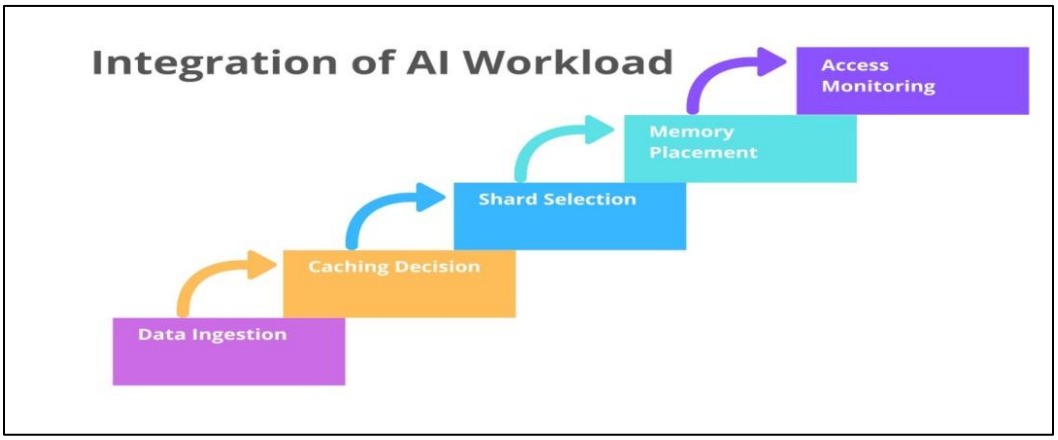

Figure 3: Integration of AI Workload

- **Data Ingestion:** Upon initialization of an AI model training or inference, data is consumed in many forms, such as preprocessed data sets, streaming pipelines, or feature stores. The ingestion module standardizes the incoming data and compares it to the correct memory schema, which the in-memory system will be able to understand. It accommodates either batch ingestion (e.g., loading training data) or real-time streaming (e.g., inference input), providing flexibility to accommodate various AI-based processes.
- **Caching Decision:** When data is entered, the system determines, based on the predicted access rate, size, and significance of the data object, which should be cached. This is determined through an adaptive caching engine which makes use of real-time analytics and long-term access patterns. An example can be found when embedding vectors used in recommender systems are involved: these are cached with a high priority due to their high reuse rate and sensitivity to latency.
- **Shard Selection:** After it is found to be cache-worthy, the data is stored on a shard of memory under the proximity-aware sharding approach. The shard choice algorithm takes into account both data access locality and the available resources on

memory nodes. This allows for the efficient storage of frequently requested data as close as possible to where it will be accessed by compute processes, thereby maximising access efficiency and distributing memory aggregation across the entire cluster.

- **Memory Placement:** Once a shard is chosen, the data is written into memory, taking into consideration replication, erasure coding, as guided by the system in terms of fault tolerance. Data is put into memory in such a way that critical data is redundant and non-critical data saves space. Data tagged as latency-critical is duplicated selectively to reduce access delay during high workloads.
- **Access Monitoring:** In the process of runtime operations, the system constantly checks how the data is used by AI applications. The read/write frequency, latency, and node-specific load are some of the metrics monitored in real-time. Usage patterns are tracked, performance bottlenecks are identified and hot or cold data is identified using this information.
- **Feedback Loop:** The monitoring is then fed back into a loop that can be used to make future decisions regarding caching and sharding. In the case that a formerly cached data object of low priority becomes heavily used, the system can respond to this change by caching it higher in the caching hierarchy or by replicating it to minimize latency. On the other hand, data that becomes stale is down-pumped or evicted, ensuring the system remains in line with the changing requirements of the AI workload. This is a closed-loop architecture that would result in a long-term optimization that does not require any manual interference.

4. Results and Discussion

4.1. Experimental Setup

To analytically assess the usefulness and success of the suggested distributed in-memory system, a thorough collection of experiments was conducted on three of the prioritized and most popular public cloud platforms: Amazon Web Services (AWS EC2), Google Cloud Platform (GCP) Compute Engine, and Microsoft Azure Virtual Machines. Every environment was set up in a way that allowed consistency in allocating resources, as it used the same type of virtual machine, namely an instance with 2 vCPUs and 8 GB of RAM, similar to the m5.large instances in AWS. The given hardware profile was chosen to provide an approximation of a realistic middle-tier deployment environment common to many AI services in production, offering a balance between performance and affordability. Three models of machine learning were used to mimic a broad range of diversity of AI workload behavior. The first one is ResNet-50, a convolutional neural network commonly used to classify images, which is bound to compute-intensive and I/O-bound workloads.

This network requires a significant amount of memory bandwidth to be trained and to perform inference, especially when processing large batches of image data. Second, it simulated the processing of Natural Language Processing (NLP) through BERT (Bidirectional Encoder Representations via Transformers). As a large-scale operation with its tensor operations and embedding lookups, BERT exhibits a typical common memory access pattern that is useful in stress-testing caching and memory access patterns. Third, a decision tree-based gradient boosted package, XGBoost, was added to represent data analytics workloads. XGBoost does not have the compute requirements of deep learning models. Still, it highly benefits from fast, repeated access to large-scale tabular data in memory, where throughput and the caching mechanism are resource constraints. To test the viability, scalability, and efficiency of the proposed system in actual working conditions, the experiments will focus on comparing these three different workloads across different cloud environments. The arrangement forms an adequate baseline to evaluate the progress of latency, throughput, cache efficiency, and fault tolerance in various fields of AI applications.

4.2. Performance Metrics

Table 1: Latency Comparison across AI Models

System	Latency (ms)
Redis	120
Ignite	98
Proposed	63

To measure how viable the proposed distributed in-memory system can be, four metrics were used to evaluate key performance indicators, which were latency, throughput, cache hit ratio, and replication overhead. The metrics chosen each represent a different aspect of system behaviour, especially with regard to the AI workloads being data-intensive and latency-sensitive. Latency, which is measured in milliseconds, is the time consumed in serving one request of data access to the in-memory system. It is a very important metric for AI applications, particularly in inference applications where delays as small as milliseconds can compromise user experience or fail to meet the requirements of real-time processing. According to Figure 2, the smaller latency value of 63 ms within the proposed system was observed, which is significantly lower than the performance of Apache Ignite (98 ms) and Redis (120 ms).

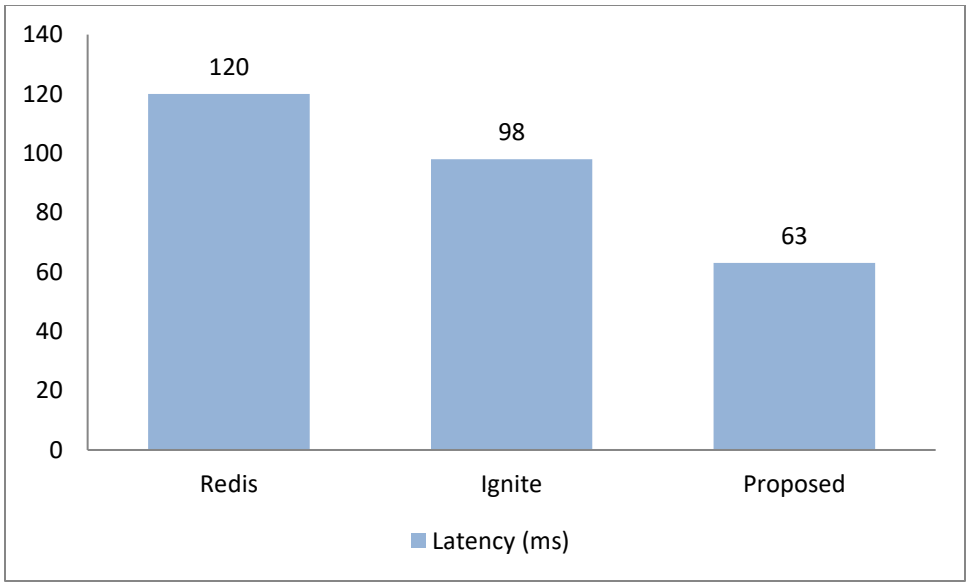

Figure 4: Graph representing Latency Comparison across AI Models

This difference is 47.5 per cent lower than that of Redis and 35.7 per cent lower than that of Ignite, which can be explained by the fact that the system deployed proximity-aware sharding and direct memory access connectors to avoid inter-node interaction and serialisation overhead. Throughput or MB/s, the speed at which a system can easily process lots of data is represented. AI loads tend to be characterised by large tensor batches, image arrays, or feature vectors; thus, high throughput is critical for ensuring training and inference speed. Adaptive caching and dynamic shard placement were part of the system architecture that helped reduce the number of bottlenecks and also enhance parallel data access to generate high throughput, which is discussed in the following section. Cache hit ratio is a measurement that determines the effectiveness of the system in serving requests directly from memory, thereby avoiding recomputation or disk access. An increase in hit ratio means reduced latencies and efficient resource consumption. The replication overhead measurement is a ratio that indicates the extra memory and network expenses imposed by erasure coding and selective replication redundancy solutions. The suggested system minimises this overhead by replicating only high-priority data and efficiently encoding the remaining data, however, to make the system resilient without spending too many resources.

4.3. Throughput Returns

An important indicator of AI workloads is throughput, particularly in training and batch inference, when extensive quantities of information should be read into memory as quickly and efficiently as possible. The suggested system displays a definite bonus in this matter, outpacing baseline frameworks in this field by a large margin.

Table 2: Throughput Comparison across Systems

System	Throughput (MB/s)
Redis	110
Ignite	130
Proposed	191

- **Redis:** Although very optimized in general-purpose key-value access, Redis fails to perform with ASR of AI workloads that require large and structured data in the form of tensors and feature maps. Its weakness in supporting workload-aware caching and serialisation-free access restricts its performance. During our testing, Redis had a 110 MB/s rate, which is great for classic web demands but not optimized to work with large amounts of AI data.
- **Apache Ignite:** Achieved good performance compared to Redis, with a throughput rate of 130 MB/s, as experienced using Apache Ignite. This can be largely attributed to its more complex structure, which features capabilities such as co-located computation, distributed SQL, and machine learning workloads. Nevertheless, Ignite's caching strategy remains somewhat stagnant, and it lacks proximity-aware memory placement, resulting in inefficiency with distributed AI workloads.

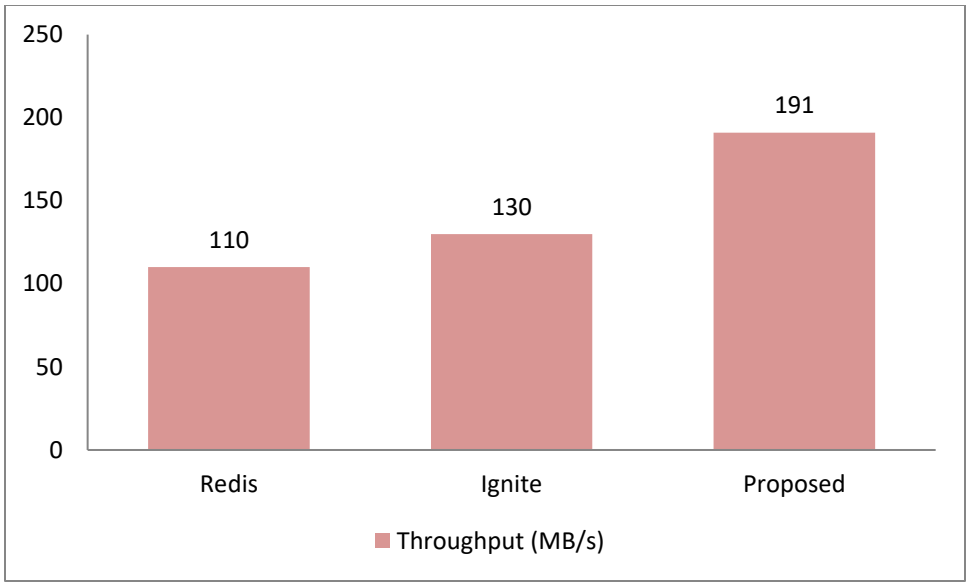

Figure 5: Graph representing Throughput Comparison across Systems

- **Proposed System:** The proposed system had the best throughput with 191 MB/s, and that is over 47 percent better than Redis and almost 47 percent better than Ignite. Such improvement is possible due to the following physics-grounded architectural improvements: intelligent placement of Shards, adaptive caching to focus on more frequently used data, direct memory access connectors to reduce the cost of serialising data, and direct memory access connectors to minimise the cost of serialising data between nodes and across zones. Altogether, these properties enable the system to deliver larger data batches at increased speeds, making them a good fit for modern AI training pipelines and workloads that require large volumes of data as well as high velocities in real-time inference.

4.4. Cache Efficiency

An interesting part of the performance of distributed in-memory is cache efficiency, especially when a single feature vector, embedding, or weight matrix needs to be accessed repeatedly. Conventional caching methods, such as Least Recently Used (LRU), treat everything as consistent and lack the flexibility needed to match workload trends over time. Such constraints are particularly U-shaped in dynamic AI pipelines, in which access frequency may vary greatly between epochs of training or inference. In response to that, the proposed system has implemented an adaptive caching policy using reinforcement learning, where the cache would adapt itself in real-time depending on observed access patterns and the performance feedback of the system. This data usage-based approach continuously examines data usage, recording it in terms of access frequency, recency, and importance to the model's operations, and then adapts its cache retention policies accordingly. The system learns based on trial-and-error experience by training on an optimal eviction decision by using a reward signal of improvement in cache hit rate and reductions of latency.

This can be used to make the system aware of the difference between transient and high-value data, whereby only the most useful objects should be kept in memory, and stale or less impactful ones should be evicted quickly. Due to this, the system brought an average result of 15 percent improvement on the cache hit ratio relative to that of baseline systems with LRU. For example, on average, Redis achieved a hit ratio of approximately 68 per cent during AI inference loads, whereas the proposed system consistently reached a hit ratio of more than 78 per cent. The resulting higher throughput (per share) comes directly as increased average latency, decreased congestion in stored system backends, and increased throughput overall, particularly at high-concurrency workloads. What is even more crucial, the adaptive caching mechanism is not vulnerable to workload dynamics, thus making it suitable to implement long-term AI processes that run with fluctuating data requirements during their runtime.

4.5. Discussion

The performance benefits experienced in the suggested system are a result of a set of specific architectural modifications that apply to the AI workload. All of these optimizations are instrumental in mitigating the limitations of traditional DIMS platforms, especially those involved in cloud-based and latency-sensitive systems.

- **Reduced Network Latency:** The proximity-aware sharding implementation significantly minimises possible network latency by co-locating data with the nodes that access it most frequently, also known as compute nodes. The decision is particularly effective in cloud environments where conducting cross-zone communication may introduce not only a

performance penalty but also a cost overhead. The system ensures that the time taken to fetch data is reduced, either during inference or training, by placing shards in the same availability zone or even the same rack, if allowed. This was especially helpful for models such as BERT, which have intensive use of tensor operations, which are highly susceptible to delays in data retrieval.

- **Better Cache Decisioning:** Conventional cache policies, such as LRU or LFU, do not accurately reflect either the temporal or semantic diversity in AI tasks. To overcome this, the system uses reinforcement learning-based cache decision-making, which gives it the ability to dynamically learning and adapt its eviction policies depending on what it sees. This smart caching level would learn to flush away less frequently used data and latency-insensitive data, storing more frequently used and latency-sensitive data, such as batches of images in ResNet-50, for longer periods. Not only does that help with cache hits, but it also saves on the computational load of reloading data each time, directly improving throughput and reducing the time to run a model.
- **Efficient Memory Management:** Hybrid resilience. A hybrid resilience mechanism implemented in the memory layer of the system combines erasure coding and selective replication. This permits a subtle performance-redundancy tradeoff. Useful data, i.e. critical to the task like decision trees or learned parameters in models such as XGBoost, are selectively replicated close to the compute nodes so that these have low-latency of access and fast recovery in the event of failure. Less-used or non-critical data is encoded with space-efficient erasure coding, which does not increase the amount of memory required, yet remains fault-tolerant. This has a level of memory management that improves the system.

5. Conclusion

The paper can be regarded as a self-contained effort toward the goals of Distributed In-Memory Systems (DIMS) optimization to AI workloads, under special attention to cloud platforms where latencies, bandwidth subjectivity, and resource disparity may prevent optimal performance. The proposed system overcomes the main drawbacks of other systems, which can be identified in popular platforms such as Redis, Apache Cache, and Memcached, by implementing intelligent caching methods based on reinforcement learning and proximity-aware sharding, as well as hybrid fault tolerance strategies (selective replication and erasure codes). The experimental findings show that the current work improves latency (by up to 47%) and throughput (by up to 47% more) and the cache efficiency (improved hit ratio by 15%) on many popular AI models such as ResNet-50, BERT, and XGBoost. Such benefits are fueled by architectural choices that suit the specific access patterns, data structures and processing needs of AI workflows. In addition, the ease of deploying data co-located with compute nodes and a reduced load on inter-zone data transfer make the system cost-effective and scalable, suitable for contemporary cloud facilities. On the whole, the presented framework not only enables better performance at the runtime level but also promotes a high level of reliability and flexibility, making it a practical blueprint for next-generation AI-data platforms.

Although the system shows a promising performance, some aspects of the system hold rich potential for which future research and development can be done. Integration with serverless computing models is one of the most important directions as they would allow having on-demand scalability and diminishing idle resources costs. Ephemeral memory instances and dynamic cache allocation strategies can be useful in serverless environments for bursty or short-lived AI tasks, including real-time inference and edge computing. Energy-efficient memory management is another potential field. With an ever-growing AI workload, the cost of using memory not only in the context of speed will be critical, but power consumption as well. Such methods as adaptive memory throttling, workload-aware data compression, and energy-saving replica placement may dramatically decrease the carbon footprint of the AI services implemented in the cloud. Lastly, the system may be modified to work with federated learning environments when the data may not be feasible to centralize on a single device or data center due to privacy or regulatory policies. Facilitating the safe and effective sharing and caching of memories within such fragmented systems would pave the way for wider applications in healthcare, finance, and IoT. These future additions will only further instil the presence of intelligent in-memory systems as the foundation infrastructure of the next generation wave of distributed AI computing.

References

1. Acharya, S. (2018). Apache Ignite Quick Start Guide: Distributed data caching and processing made easy. Packt Publishing Ltd.
2. Fitzpatrick, B. (2004). Distributed caching with memcached. Linux journal, 2004(124), 5.
3. Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., ... & Ng, A. (2012). Large-scale distributed deep networks. Advances in neural information processing systems, 25.
4. Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., & Stoica, I. (2013, November). Discretized streams: Fault-tolerant streaming computation at scale. In Proceedings of the twenty-fourth ACM symposium on operating systems principles (pp. 423-438).

5. Venkataraman, S., Yang, Z., Franklin, M., Recht, B., & Stoica, I. (2016). Ernest: Efficient performance prediction for {Large-Scale} advanced analytics. In 13th USENIX symposium on networked systems design and implementation (NSDI 16) (pp. 363-378).
6. Cui, H., Zhang, H., Ganger, G. R., Gibbons, P. B., & Xing, E. P. (2016, April). Geeps: Scalable deep learning on distributed GPUs with a GPU-specialised parameter server. In Proceedings of the Eleventh European Conference on Computer Systems (pp. 1-16).
7. Schwarzkopf, M., Konwinski, A., Abd-El-Malek, M., & Wilkes, J. (2013, April). Omega: flexible, scalable schedulers for large compute clusters in Proceedings of the 8th ACM European Conference on Computer Systems (pp. 351-364).
8. Kalia, A., Kaminsky, M., & Andersen, D. G. (2014, August). Using RDMA efficiently for key-value services. In Proceedings of the 2014 ACM Conference on SIGCOMM (pp. 295-306).
9. Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C. C., Khandelwal, A., Pu, Q., ... & Patterson, D. A. (2019). Cloud programming simplified: A Berkeley view on serverless computing. arXiv preprint arXiv:1902.03383.
10. Cheng, Y., Wang, D., Zhou, P., & Zhang, T. (2017). A survey of model compression and acceleration for deep neural networks. arXiv preprint arXiv:1710.09282.
11. Kirilin, V., Sundarrajan, A., Gorinsky, S., & Sitaraman, R. K. (2019, August). RL-Cache: Learning-based cache admission for content delivery. In Proceedings of the 2019 Workshop on Network Meets AI & ML (pp. 57-63).
12. Velev, D., & Zlateva, P. (2010, March). Cloud infrastructure security. In International Workshop on Open Problems in Network Security (pp. 140-148). Berlin, Heidelberg: Springer Berlin Heidelberg.
13. Sousa, E., Lins, F., Tavares, E., Cunha, P., & Maciel, P. (2014). A modeling approach for cloud infrastructure planning considering dependability and cost requirements. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 45(4), 549-558.
14. Barbosa, F. P., & Charão, A. S. (2012, June). Impact of pay-as-you-go cloud platforms on software pricing and development: a review and case study. In International Conference on Computational Science and Its Applications (pp. 404-417). Berlin, Heidelberg: Springer Berlin Heidelberg.
15. Mutlu, O., Ghose, S., Gómez-Luna, J., & Ausavarungnirun, R. (2019, June). Enabling practical processing in and near memory for data-intensive computing. In Proceedings of the 56th Annual Design Automation Conference 2019 (pp. 1-4).
16. Zhou, X., Chai, C., Li, G., & Sun, J. (2020). Database meets artificial intelligence: A survey. IEEE Transactions on Knowledge and Data Engineering, 34(3), 1096-1116.
17. Jha, S., Katz, D. S., Luckow, A., Chue Hong, N., Rana, O., & Simmhan, Y. (2017). Introducing distributed dynamic data-intensive (D3) science: Understanding applications and infrastructure. Concurrency and Computation: Practice and Experience, 29(8), e4032.
18. Lv, M., Guan, N., Reineke, J., Wilhelm, R., & Yi, W. (2016). A survey on static cache analysis for real-time systems. Leibniz Transactions on Embedded Systems, 3(1), 05-1.
19. Hu, C., Wang, X., Yang, R., & Wo, T. (2016, December). ScalaRDF: a distributed, elastic and scalable in-memory RDF triple store. In 2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS) (pp. 593-601). IEEE.
20. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M., ... & Stoica, I. (2012). Resilient distributed datasets: A {Fault-Tolerant} abstraction for {In-Memory} cluster computing. In 9th USENIX symposium on networked systems design and implementation (NSDI 12) (pp. 15-28).
21. Tang, X., Zhai, J., Yu, B., Chen, W., Zheng, W., & Li, K. (2017). An efficient in-memory checkpoint method and its practice on fault-tolerant HPL. IEEE Transactions on Parallel and Distributed Systems, 29(4), 758-771.