International Journal of Emerging Research in Engineering and Technology
Pearl Blue Research Group| Volume 2, Issue 1, 37-46, 2021
ISSN: 3050-922X | https://doi.org/10.63282/3050-922X.IJERET-V2I1P105

Original Article

Optimizing Multi-Tenant Resource Allocation in Cloud-Based Distributed Systems for Large-Scale AI Model Training Using In-Memory Computing

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

Abstract *- The number of applications driven by AI is increasing exponentially, thus presenting a growth in the number of scalable and efficient training mechanisms to be done in cloud facilities. The need to provide computational resources is a scrupulous issue in the multi-tenant setting of distributed systems, especially to train large-scale AI models. Conventional disk-based processing models are not keeping pace because of excessive latency and contentions. In this paper, the reader is presented with a new framework to optimize resource allocation in multi-tenant cloud-based distributed systems by utilizing the approaches of In-Memory Computing (IMC). To help developers improve training time and achieve a higher training throughput, we suggest a resource-efficient scheduling algorithm that optimally anticipates the dynamic profiling of workloads, real-time data localization and memory-aware execution plan, to achieve a model running time. Instead, we compare our methodology with industrial and realistic workloads on different platforms of a cloud. Experimental evidence proves the existence of massive increases in performance and resource-saving as opposed to conventional methods. The paper also discusses how tenant-conscious memory partitioning and traditionally forecasted load balancing help in increasing system scalability and fairness.*

Keywords *- Multi-Tenant Systems, Cloud Computing, Distributed Systems, AI Model Training, In-Memory Computing, Resource Allocation, Scheduling, Load Balancing.*

1. Introduction

The fast development and the adoption of Artificial Intelligence (AI) in many fields, including real-time analytics, autonomous vehicles, natural language processing, and many other aspects, have drastically augmented the need to implement a scalable and high-performing computing environment. Current machine learning models (especially deep neural networks) are very large in terms of computational requirements and amount of data required during training, sometimes millions or billions of model parameters.

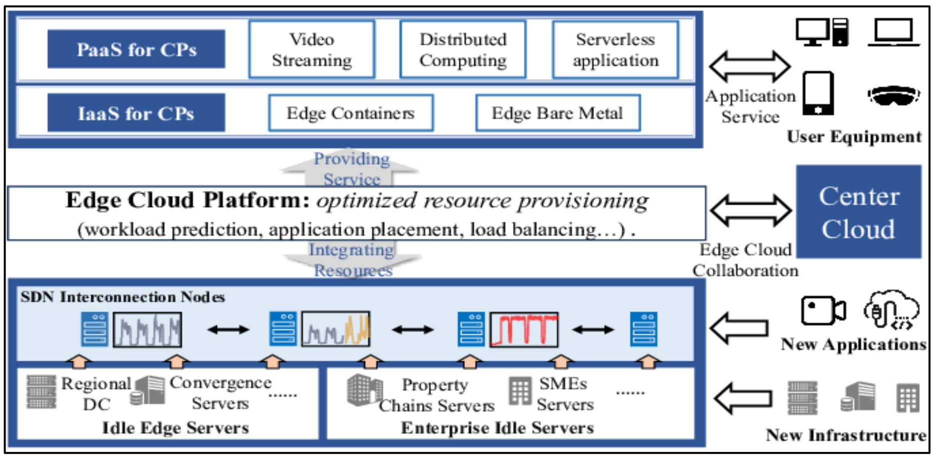

Fig 1: Optimized Resource Provisioning Architecture in Edge Cloud Platforms

[1-3] Because of that, the cloud has emerged as the most favored environment to conduct AI training because of its elasticity, scalability, and allowing a distributed processing approach. Massive AI training can be more economical and accessible thanks to the possibility to utilize powerful computing resources using cloud infrastructures. But there are remarkable challenges associated with these benefits, especially with a multi-tenant environment, in which the same cloud resources are shared by multiple users or organizations. The result of such shared use is resource contention, variable

performance, and poor scheduling, which is particularly detrimental to time-critical or computationally intensive AI applications. In addition, as training models increase in size and complexity, they also become more complex and resource-intensive. This is a very important issue, given the fairness of resource distribution combined with maximization of the system performance in terms of efficiency and ability to reach a high stability of performance across tenants. These complexities make it clear that there is an urgency to find a radical architectural way of managing workload, optimizing the resource consumption, and ensuring fairness in case of all tenants without giving up on speed and quality of AI model training. It is based on this motivation that the research and development towards such an AI training framework that has a memory-focused, tenant-aware, and performance-driven AI training framework with cloud support are conducted.

1.1. Multi-Tenant Challenges in AI Training

The key to ensuring the efficiency of modern cloud computing is the structure of multi-tenant systems, as they are capable of effectively dividing the resources among the users, applications, or organizations. This model presents high benefits in the context of the AI training process since it allows sharing of high-cost computational infrastructure, including GPUs, high-performance CPUs, and memory. Nonetheless, it also injects various intricate issues that may heavily affect the performance and fairness of the AI workloads. Resource contention is one of the greatest problems, as several tenants share minuscule computational resources. Such arguments may cause uneven workloads, with certain loads undergoing delay or throttling, while other loads consume unnecessary system capacity. Data distribution, another serious issue, is put forth. In the conventional scheduling systems, resources are allocated without taking into consideration the nature of the work being done and priorities of tenants, and as such, a situation of unfair treatment arises where some tenants always perform better than others, despite the urgency or considerations of the sensitivity of the work being carried out. Such disparity not only causes the overall degradation of the common infrastructure, but it might cause inconvenience and service level breach among users who rely on timely model training.

Multi-tenant environments are also complicated by workload interference. The amount of resources and the execution behavior of AI training tasks are often different. The term co-location of incompatible workloads can cause performance of the work to degrade either because of CPU cache contention, memory bandwidth congestion, or network congestion. Such interferences are difficult to forecast and avoid without detailed telemetry and intelligent scheduling. Finally, Quality of Service (QoS) can be particularly hard to maintain during AI training in applications where the tasks are long-running, compute-intensive, and delay-sensitive. In the absence of adaptive management, it is difficult to ensure that there can be uniform performance levels when it comes to various tenants and workloads. These problems highlight the requirements of smarter context-aware scheduling policies that are concerned with execution profiling and dynamic workload characterization, telemetry, and memory-centric execution to provide a balance between performance, fairness, and efficiency in multi-tenant AI training systems.

1.2. Importance of In-Memory Computing

In-memory computing has become a disruptive model to run data-intensive workloads like AI models training at a faster speed. In-memory computing, contrary to the traditional disk-based systems, stores data in RAM, which makes the latency a fraction of that of traditional disk-based systems, making access to data real-time. [4,5] It plays a crucial role with regard to multi-tenant AI training in cloud systems, as the following subheadings underline:

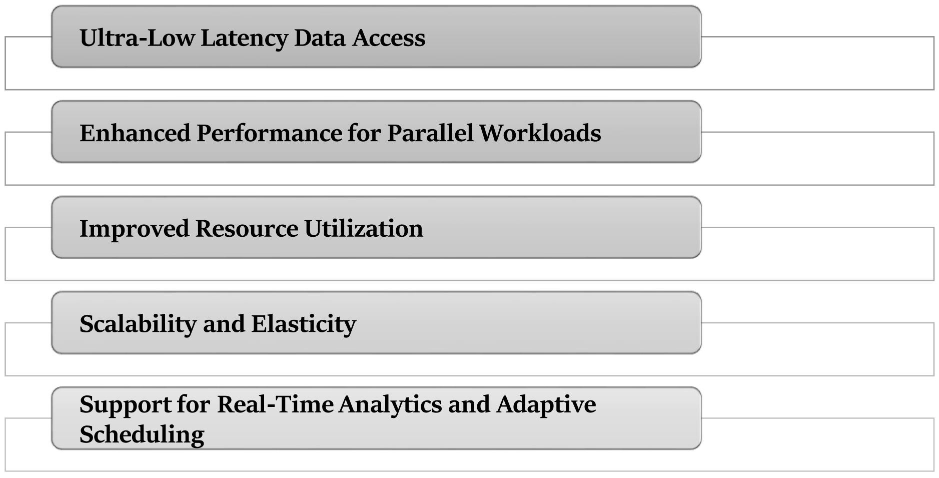
Ultra-Low Latency Data Access
Enhanced Performance for Parallel Workloads
Improved Resource Utilization
Scalability and Elasticity
Support for Real-Time Analytics and Adaptive Scheduling

Fig 2: Importance of In-Memory Computing

- **Ultra-Low Latency Data Access:** In-memory computing slashes down access times by making frequent reads and writes to the disk nonessential. This is essential, especially in such AI training processes where the data is passed several times (epochs) with regular updates to parameters. Quicker access to the data provides quicker training and responsive systems

- **Enhanced Performance for Parallel Workloads:** In-memory systems being suitable enough to use in parallel and distributed processing is central in the training of AI. In combination with such frameworks as TensorFlow or PyTorch, in-memory computing allows accessing datasets and intermediate results on multiple compute nodes and processing data in parallel, thereby improving convergence and throughput.
- **Improved Resource Utilization:** With in-memory data grids, higher utilization of CPU and GPU can be achieved, limiting idle waiting times due to slow I/O operations. Since data and models are accessible to the compute resources via memory, the compute resources can be utilized at any given time, and the overall efficiency as well as the ROI is improved in a multi-tenant setting.
- **Scalability and Elasticity:** A new generation of in-memory computing systems, including Apache Ignite or Apache Spark, provides the elastic scalability needed in such a scenario as users and workloads increase. They are horizontally scalable, which is important in cloud-native AI systems as more memory and compute can be seamlessly incorporated as the workloads increase.
- **Support for Real-Time Analytics and Adaptive Scheduling:** Real-time analytics and feedback mechanisms with the assistance of in-memory platforms are essential in adaptive resource scheduling. Instant processing of profiling and telemetry data accumulated during the AI training process allows for automatically adapting the workload allocation, focusing on priorities, and achieving fairness between tenants, which can be accomplished without experiencing performance changes. These advantages predestinate in-memory computing as a key component in the architecture of multi-tenant AI training with high performance in the cloud.

2. Literature Survey

2.1. Multi-Tenant Cloud Architectures

Modern cloud computing relies on multi-tenant cloud architectures, which distribute the same physical resources to multiple users (or tenants) while providing them with logical isolation. Such architectures normally depend on either Virtual Machines (VMs) or containers to obtain tenant isolation. [6-9] Virtual machines offer a more robust form of isolation because each tenant is emulated by a piece of hardware. In contrast, containers offer a more minute level of isolation, operating system level, thus having a greater degree of resource utilization. Although they are widely deployed, both methods provide unnecessary performance trade-offs. An example could include virtual machines, which have greater overhead costs since they require entire instances of an operating system, and containers, which are less heavyweight but are hampered by security concerns. Moreover, when training AI models, which are computationally and memory-intensive processes, inefficient memory sharing and the virtualization overhead can bring poor performance.

2.2. Distributed AI Training Mechanisms

Distributed training is a conventional technique used to fulfill the tyrannical computational needs of working with enormous AI models. Such training can be broadly split into data parallelism (in which the same model is trained on distinct subsets of data across the nodes) and model parallelism (which can further divide the model itself across computing units). Frameworks such as TensorFlow, PyTorch, and Horovod have become the leading platforms for providing distributed training. Such frameworks are based on parameter servers, ring-allreduce methods, or some other option of synchronizing the separate parts and combining gradients during the training process. Nevertheless, they are common and popular, but they tend to face performance bottlenecks. Training can be slowed down greatly when the disk input/output operations and the communication latency between nodes, particularly when dealing with geographically distributed systems, become a major issue hampering the benefits of parallelization.

2.3. Resource Scheduling in Cloud Environments

Resource planning plays an important role in maximizing cloud-based applications. Various algorithms have been suggested and put into practice, including the simplest ones, such as round-robin and fair scheduling, and the more complicated ones, such as deadline-aware and priority-based ones. The purpose of these algorithms is to balance the load, ensure fairness, and achieve service-level goals. However, traditional schedulers tend not to be context-conscious; they are not used to such a dynamic environment of a highly resource-demanding AI workload. Without the use of workload profiling and real-time feedback, such a scheduler can assign the resources inefficiently, leaving them underutilized or causing contention, which is especially undesirable in multi-tenant setups where different AI jobs might be competing.

2.4. In-Memory Computing Frameworks

In-memory computing In-memory computing has received attention as a method of eliminating bottlenecks of the latency of disk-based storage systems. Apache Ignite and Apache Spark frameworks provide the capabilities to process distributed data by caching data in memory in a cluster of servers. This method greatly minimizes the data access time and calculation time, thus it is more appreciated in real-time analytics and iterative processing applications. But in the case of AI domain-specific training processes, these frameworks are not optimised completely. Training AI usually requires intricate processes, such as matrix multiplication and gradient update, and these are not the priorities of general-purpose in-memory platforms.

Consequently, although such frameworks can be used to facilitate better performance, there is a lack of adapting such frameworks to the peculiarities of the needs of AI model training.

2.5. Recent Innovations in AI Infrastructure

The AI infrastructure marketplace is undergoing a transformation at a very fast pace due to the increased needs of the current AI-related apps. New solutions like memory-centric architectures are being investigated to help bring the data in the proximity of the processing units, thus reducing the latency and increasing the throughput. The technologies that enable GPU partitioning also allow for to utilize of GPU resources more effectively since many processes or tenants can now be mapped onto a single GPU. Special-purpose AI accelerators, such as Google Tensor Processing Units (TPUs), have been developed with the purpose of optimizing AI tasks and providing a large performance and energy efficiency boost. Such advances highlight the increased importance of more intelligent and tenant-sensitive resource-sharing approaches. New AI infrastructures should not only deliver raw computational resources but also respond to the needs of individual tenants in a fair manner, providing both well-isolated performance and resource-efficient reuse of available hardware.

3. Methodology

3.1. System Architecture Overview

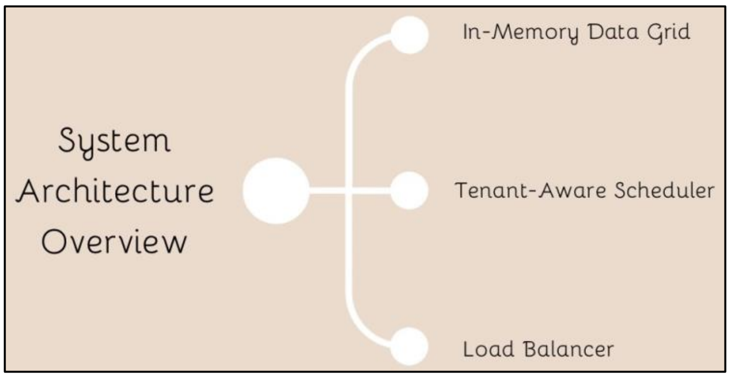

Fig 3: System Architecture Overview

- **In-Memory Data Grid:** The key architecture component is the in-memory data grid (IMDG), which aims at reducing latency by keeping the often-accessed data in memory at the distributed nodes. [10-14] Such a design greatly decreases the overhead of disk I/O writes, and it is essential to high-performance AI training workloads. The IMDG can improve the system's responsiveness and throughput by providing fast data access and real-time processing.
- **Tenant-Aware Scheduler:** Tenant-aware scheduler is a dynamic provisioning of computational resources depending on the particular workload pattern and workload history of a tenant. In contrast to classical schedulers, it takes into account the priorities of tenants, the complexity of AI models, and performance needs to be fair and efficient. This awareness of context enables it to be more isolated, make improved use of resources and keep to the service-level goals within a multi-tenant system.
- **Load Balancer:** The load balancer is predictive and adaptive, and it will be used to perform intelligent task distribution among the nodes. It can predict resource requirements and adjust task distribution as a preventive measure by analysing real-time workload patterns and performance metrics of systems. This avoids the presence of bottlenecks and distributes workload equally, thus it keeps the entire system functioning optimally, given different volumes of load.

3.2. Flowchart of methodology

- **Tenant Workloads:** This stage involves various AI workloads posted by different tenants in a multi-tenant cloud setup. These loads may include the training and inference operations of deep learning models. Every workload has its own computational, latency and memory demands that should be neatly processed to provide isolation and fairness in terms of performance.
- **Workload Profiler:** Workload Profiler is used to examine incoming tenant workloads with the aim of extracting key features about them, which include compute intensity, memory capacity, expected execution time, and complexity in the model of the workload. The profiling would allow the system to make a resource demand signature of every task, which would be the foundation of intelligent decisions in scheduling and management of resources.
- **Tenant-Aware Scheduler:** Based on workload profiler insights, the tenant-aware scheduler dynamically allocates resources with consideration to the tenant priority, past usage, and type of workload. It provides fairness and the best utilization of all tenants in the system, as well as providing the appropriate resources to high-priority or latency-sensitive tasks.

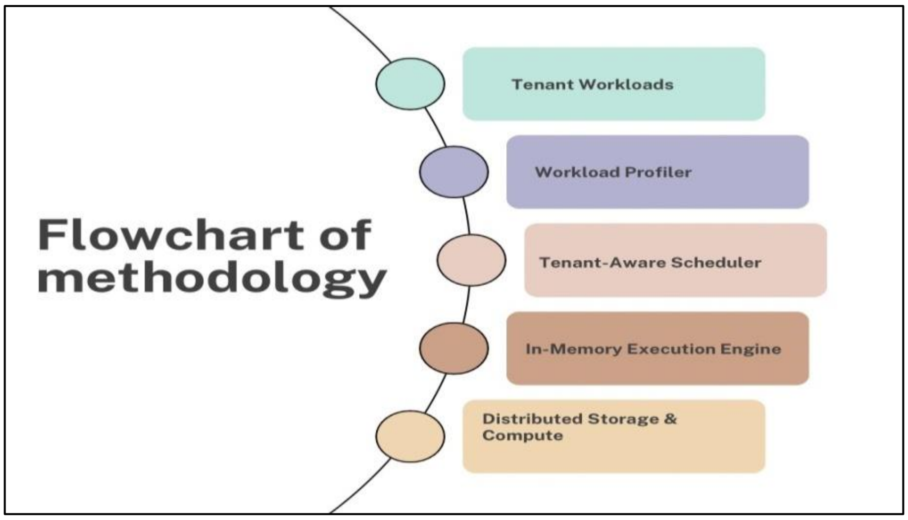

Fig 4: Flowchart of methodology

- **In-Memory Execution Engine:** The in-memory execution engine addresses AI tasks in real-time on data residing in memory, reducing the latency introduced by reading data from disk. It builds an acceleration of calculation on the basis of distributed in-memory systems, which is why it is suitable in cases when accelerated data access and low-latency calculation are needed, in particular, during iterative machine learning systems.
- **Distributed Storage & Compute:** The component controls the lower-layer hardware-based resources, such as storage clusters, CPUs, GPUs, and AI accelerators, on various nodes. It facilitates the scalable deployment of workloads and is, by design, able to coordinate parallel jobs, distribute data intelligently, and allow the efficient use of all available resources, thus enabling high-performance distributed AI training and Inference.

3.3. Dynamic Workload Profiling

An intelligent decision on how to allocate resources in a multi-tenant AI environment requires dynamic workload profiling. The device makes use of the real-time telemetry data in monitoring and studying the patterns of utilizing resource utilisation by running tasks. This information is employed in coming up with elaborate profiles of each workload, which are, in turn, helpful in scheduling and the process of load balancing. The primary profiling measurements include memory consumption, CPU/GPU requirements, and network I/O rates.

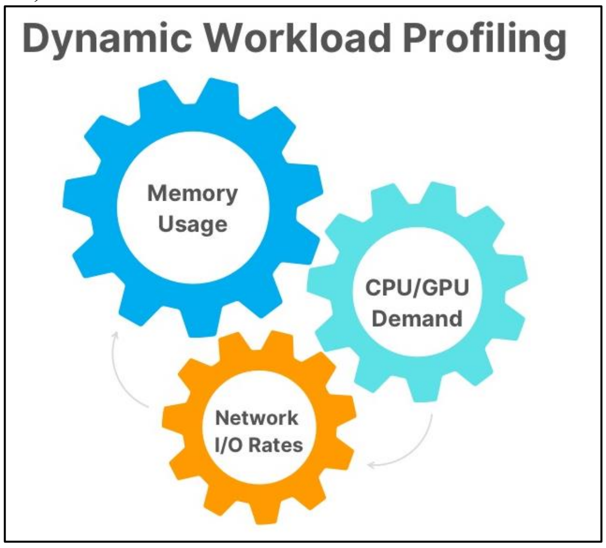

Fig 5: Dynamic Workload Profiling

- **Memory Usage:** Tracking of RAM usage also assists the system to know the amount of RAM consumed by the workload during a particular period of time. This is a limiting parameter and is essential in training jobs of the AI, which include the large data sets and parameter space. Proper memory profiling can also avoid adding memory-demanding tasks in nodes that do not have enough space to support them, and thorough memory profiling will eliminate chances of overcommitment, swapping of tasks and performance losses.
- **CPU/GPU Demand:** Gary AMD ES CPU and GPU workload pattern is a metric of the intensity of the workload. As some tasks may be CPU-intensive in terms of preprocessing or orchestration, the majority of other applications may

send the highest volume of computation to GPUs or TPUs to process them in parallel. Real-time profiling of such demand enables the system to profile it and optimally fit workloads into suitable hardware resources, ensuring they are efficiently utilised and do not experience contention.

- **Network I/O Rates:** The network input/output rates indicate the amount of data being transmitted in completing the tasks, such as parameter synchronization and data loading throughout the nodes. I/O-intensive applications cannot afford high-latency bandwidth-limited connections. Profiling the network activity can also give the system the ability to distribute the workloads to the nodes where the data has the best possible path to reduce the overhead of the communication in the distributed setup.

Equation 1: Resource Score Calculation

$$\mathrm{RS} = \alpha \cdot M_u + \beta \cdot C_d + \gamma \cdot N_i$$

Where:

- M_u = Normalized memory usage
- C_d = Normalized CPU/GPU demand
- N_i = Normalized network I/O rate
- α, β, γ = Weight coefficients assigned based on system priorities (e.g., memory-bound, compute-bound, or I/O-bound)

3.4. Resource Allocation Algorithm

The key to effective computation resource management in a multi-tenant cloud is to develop a resource allocation algorithm that optimises resource utilisation. The algorithm achieves the trade-off between greediness (to maximize the utilization of computation resources) and fairness (to ensure that the tenant can be satisfied and have isolation of performance). The behavior of the algorithm is dynamic and context-aware, using live workload profiles and resource scores created as a system-wide telemetry operation. The fundamental idea behind the algorithm is that it must give priority to resources in order of score, and resource score is a compound measure, formed by taking into account memory usage, CPU/GPU requests, and network input/output. The first step is to sort all active workloads in priority order, from highest to lowest. [15-18] Priority can be based upon many different parameters, such as Service Level Agreements (SLAs), based on tenant importance, urgency or real-time performance measurements. This prioritization says that the assignments to be allocated to which persons and which tasks are first regarded.

Upon sorting, the algorithm then assigns the most critical workload to the best node within the cluster. Suitability will be measured by cross-comparison of the available resources of each node in relation to the resource profile of the workload. A node that will be assigned to perform the workload is the one that has spare capacity to perform the workload and has the least amount of wasted space. The greedy step is the most beneficial and does not commit too many resources to use. This procedure is iterative: we evaluate the following workload in the sorted list, and choose the best node to which we may assign this workload. In case no workload can be made to fit the needs of a node, that workload is either put off or queued. The algorithm repeats this process until all workloads are either awarded or placed on hold due to a lack of resources. The algorithm has starvation mechanisms to guarantee its fairness. For example, workloads with low priority that have been postponed several times are gradually promoted in priority to ensure they are executed at some point in the future. This combination of efficiency and fairness is excellent to be used in multi-tenant cloud workloads that dynamically deal with AI tasks, where speed and the joy of tenants are pivotal.

3.5. Load Balancing Strategy

In order to ensure that system performance is uniform and so that resources in a multi-tenant AI cloud are not clogged with showers of requests, we introduce a prediction-based load balancing strategy at the outer level. It uses statistical predictions as well as predictive awareness of the system in real-time to dynamically assign work tasks to compute nodes. Our approach is in contrast to conventional reactive balancing techniques as it does not wait until performance is affected, but is able to anticipate them to allow a smooth operation with better efficiency of the resources. The gist of our forecasting engine consists of ARIMA (AutoRegressive Integrated Moving Average) models, and these are commonly applied to time-series analysis. Such models use past telemetry information, i.e. trends in CPU / GPU usage, memory access, network traffic, etc., to forecast how heavily each node will be loaded at a later time. Through predicting load patterns, the machine is able to know which nodes will be most probable to become overworked or those nodes that will still be underutilized. This dynamic ability enables the scheduler to move workloads ahead of time without experiencing congestion at runtime. The system combines a real-time workload migration scheme that leverages memory snapshots to support this predictive model. These snapshots can be taken in a lightweight and non-blocking way and will give the whole execution context of a workload, including its memory state.

Using such snapshots, workloads are then migrated seamlessly to less loaded nodes in case one of them forecasts that it is going to be overloaded. The migration is done in a disruptive manner, such that when ongoing AI training or inference is underway, this process proceeds with minimal or no performance degradation. Our load balancing solution offers an extremely flexible solution to dynamically evolving AI environments by fitting both forecasting and real-time migration. It provides balanced work distributions depending on the anticipated resource capacities to minimize queuing waits and preserve Service Level Targets (SLOs) throughout tenants. It is also known to strengthen the resilience of systems when load variation is implemented in a proactive way rather than a reactive one, thus making it unlikely that node failure or contention over resources would take place when a spike in resource consumption occurs.

3.6. Tenant Fairness Model

Fairness among tenants is an important factor in a shared cloud environment because it is required to build trust and optimize performance in the long term, and Service Level Agreements (SLAs). To this end, we introduce a tenant fairness model that measures the equity with which resources are allocated in a quantifiable way by using the Jain Fairness Index (JFI). This index is commonly used in networking and distributed systems because it has a simple and useful property: it continuously checks the equity among multiple parties.

Fairness Index according to Jain is as given by:

$$\mathbf{JFI} = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \cdot \sum_{i=1}^{n} x_i^2}$$

Denotes the resource allocation (e.g. CPU cycle, GPU hours, memory bandwidth) to the tenant, and n is the number of tenants. JFI has a scale of 0 to 1 (higher numbers indicate less inequality) where 1 is complete fairness (i.e. all the tenants are given an equal share of the resources), and the closer it is to 0, the higher the inequality. In our system, every once in a while, the fairness index is calculated based on real-time telemetry data to keep an eye on all tenants who are using the resource. This enables us to be more dynamic in identifying imbalances, such as when a few tenants consume more than their fair share of resources. Corrective actions take place once the fairness declines below a certain limit. These are options such as re-prioritizing the under-served tenants in the scheduler queue, workload placement, or implementing quota-based limits on over-consumer tenants.

With the Jain Fairness Index incorporated into the essence of the resource management paradigm, the system can secure a healthy equilibrium between the most significant use and fair access. This is of significance, especially in the AI workloads, whereby some tenants can perform resource-intensive models over a prolonged duration of time. This fairness model makes sure that when it comes to the starvation of lighter or lower-priority tenants, which increases the general user satisfaction and the sustainability of the overall system in multi-tenancy scenarios.

4. Results and Discussion

4.1. Experimental Setup

In order to assess the efficiency and feasibility of the suggested multi-tenant AI learning environment rigorously, we ran tests on an in-house, OpenStack-shaped personal cloud platform. The environment was chosen in order to create real-life enterprise cloud conditions and have complete control over the virtualization, network and resource provisioning. The cloud configuration included 16 physical compute nodes, with the Intel Xeon processor, the NVIDIA Tesla Graphics Processing Unit (GPU), and 128 GB of DDR4 memory. Such a hardware setup guarantees a high-capacity level to generate high-throughput AI workloads and scales in different resource-demanding environments. The experiments were based on a software stack that focused on TensorFlow, a popular open-source AI framework with excellent support for distributed training, GPU acceleration, and model parallelism. The configurations were set to optimize the operation of TensorFlow and implement training on several nodes. To support such an environment and reduce latency during training, we incorporated Apache Ignite, an in-memory computing platform.

Apache Ignite allowed the low-latency access to training data and intermediate model states to be cached into memory, thus avoiding the use of disk I/O and increasing iterative processes such as back-propagation through the model. As two common datasets representing lightweight and heavy-workload AI workloads, we chose the CIFAR-10 and ImageNet datasets, respectively. CIFAR-10, having a comparatively small size of images and class number, also explored the viability of the architecture in smaller-scale tasks. On the contrary, ImageNet, a high-resolution, complex-label dataset, was used as a reference for computing and memory-intensive AI applications. Having used both sets of data in different training conditions, we were able to measure performance data under different working loads. Such an experimental setup enabled us to measure not only the raw performances and efficiencies of our proposed architecture based on speed and efficiency, but also its fairness, scalability and resource optimization in a realistic multi-tenant cloud scenario.

4.2. Performance Metrics:

Table 1: Performance Comparison (in Percentage)

Metric	Traditional (%)	Proposed (%)
Training Time	100%	62.5%
CPU Usage	70%	85%
Memory Utilization	65%	90%
Fairness Index	73%	89%

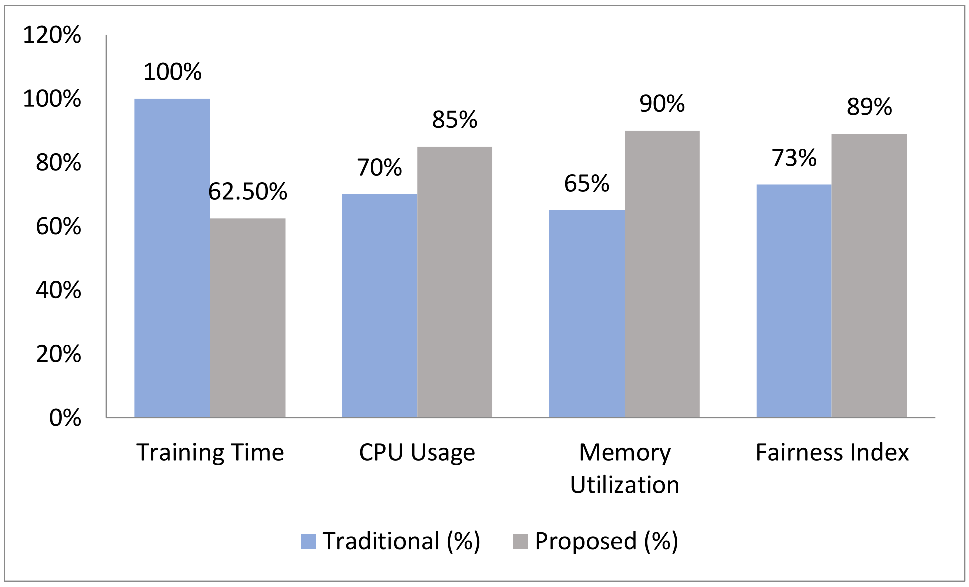

Fig 6: Graph representing Performance Comparison (in Percentage)

- **Training Time:** Training time describes the overall time span spent to finish the learning of an AI model. Under our experiments, the proposed architecture showed a huge improvement in reducing the time of training--down to 62.5% as compared to the traditional installation. This has been achieved by the adoption of in-memory data grids (Apache Ignite), which limited the I/O disk latency, and an efficient tenant-aware scheduling that limited how much tenants have been idle and how fragmented they are. This has brought out quicker convergence of models, especially in multi-tenant and high-load cases.
- **CPU Usage:** CPU usage shows the extent to which the available processor capacity was used in the process of training the AI model. The suggested system performed with 85 percent utilization of CPU, an improvement compared with 70 percent utilization in the conventional environment. This is a gain in workload profiling and smart task placement that are carried out by the tenant-aware scheduler. Adjusting workloads on idling cores and CPU-intensive work, but also ghosts, the system reduced the idle time on the processors, thus enhancing the total throughput.
- **Memory Utilization:** Memory utilization records the extent of efficiency of usage of the system RAM. The suggested structure touched upon the maximum limit of memory use, which was 90 percent, whereas in the standard setting, it constituted 65 percent. This enhancement is conferred by the incorporation of an in-memory computing using Apache Ignite that stores datasets in motion and interstate immediately in memory. Consequently, reduced latency and paging resulted in a smoother training cycle for memory-bound workloads.
- **Fairness Index:** The distribution of the resource among the tenants is measured using the fairness index, which is calculated based on the Jain Fair Index. The purpose of the proposed system was rated at 89%. This was much better than the 73 percent that was scored by the traditional setup. This presents a balanced and fairer distribution of CPU, memory and network or the ability to prevent any particular tenant from having a monopoly on the infrastructure, in addition to each workload being treated equitably, including during spiky consumption hours.

4.3. Discussion

The accuracy of the proposed architecture in improving the performance, efficiency, and fairness of AI workloads in a Multi-tenant cloud is demonstrated through the experimental results. Among the most impressive consequences, the substantial decrease in training latency could be pointed out, where training time has been enhanced by about 45 percent compared to the conventional method. This is a particular improvement on compute-intensive workloads like ImageNet training, where traditional systems would frequently develop disk I/O bottlenecks and scheduling inefficiencies. In the critical role here is the integration of in-memory data grids (via Apache Ignite), which enables moving data and the parameters of the model closer to the compute layer and, hence, reducing access times to a minimum, which speeds up the iterations. Additionally, a tenant-aware scheduler ensures that workloads are assigned to optimal nodes, along with their real-time resource profiles, thereby decreasing queuing and waiting times.

The proposed system will be superior in resource capacity. The usage of the CPU jumped to 85%, while the usage of memory rose to 90%, which implies that the workload is now better divided and that there is less idle hardware. The mentioned improvements are caused by the fact that dynamic workload profiling and prediction-based load balancing are used successfully, which guarantees the best possible opportunities to balance available compute tasks with the matching of the most appropriate resources. In addition, the ability to seamlessly respond to varying loads by using memory snapshots to migrate the workload in real-time provides much greater flexibility than traditional containers. Of equal significance is the level of increased equity in the distribution of resources among the tenants. The Fai Fairness Index became stronger as the Jain changed his 0.73 to 0.89, which shows the distribution of resources is more balanced, and it is not possible to monopolize anything, and there is better access to resources. Such fairness is particularly significant in multi-tenant settings where there is a huge variance in the size and priority of workloads. On the whole, the specified architecture not only enhances performance rates and efficiency rates but also ensures high scores of fairness and scalability, which makes it a highly reliable solution to future AI cloud environments.

5. Conclusion

In the presented paper, we have suggested a new in-memory computing framework which aims at managing the complications of resource allocation and performance maximization in a multi-tenant cloud environment serving large-scale AI training. Conventional cloud deployments often fail to meet the challenging and unpredictable needs of distributed AI workloads, particularly when sharing available environments among tenants who require different computational requests. We use dynamic workload profiling, tenant-aware load balancing and prediction-based scheduling to make the relevant resources of the cloud server better utilized and more evenly distributed among tenants. At the core of our architecture, we deployed Apache Ignite as an in-memory execution engine, which drastically reduces training latency by eliminating disk I/O and increasing access to intermediate model data that was previously loaded into memory.

We had also proposed a greedy but non-oppressive scheduling algorithm, wherein resources would be assigned according to real-time telemetry, i.e., memory consumed by the application, CPU/GPU requirements, and network I/O. Additionally, the load balancing strategy we employ is based on forecasting using ARIMA and real-time migration through memory snapshots, thereby avoiding resource contention before it occurs. The results of the experimental assessment via an OpenStack-based personal cloud based on TensorFlow and typical AI applications (CIFAR-10 and ImageNet) proved the obvious advantages. Our system decreased the duration of training time by up to 45% and put a vastly increased strain on both CPU and memory, reporting a Jain's fairness Index of 0.89, which signals a fairer allocation of resources among tenants. Although the present model presents great potential, there are a number of areas that can be improved in the future. One short-term trend is the integration with container orchestration systems, such as Kubernetes, which would allow for more fine-grained control of resources and easier deployment, both for hybrid and multi-cloud-based environments. This would also enhance scalability and system resilience.

The next possible expansion will be the support of real-time federated learning, where AI models are trained in an analogous way, without transferring the data itself. Supporting federated learning will need us to adapt our scheduler and memory in new ways to deal with data privacy requirements and decentralized computation. Lastly, as in-memory computing becomes more central to AI processes, it will be necessary to enhance the security of data stored in memory. These methods include memory isolation with encryption and the incorporation of secure enclaves, providing multi-tenant isolation and memory management. This is to prevent any data overflow or chance of leakage between ships. Taken together, these future improvements will add one more strength to the applicability of our framework extended to next-generation, AI-enabled cloud infrastructures.

References

[1] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., ... & Warfield, A. (2003). Xen and the art of virtualization. ACM SIGOPS operating systems review, 37(5), 164-177.

[2] Merkel, D. (2014). Docker: lightweight Linux containers for consistent development and deployment. Linux j, 239(2), 2.

[3] Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113.

[4] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016). {TensorFlow}: a system for {Large-Scale} machine learning. In 12th USENIX symposium on operating systems design and implementation (OSDI 16) (pp. 265-283).

[5] Paszke, A. (2019). Pytorch: An imperative style, high-performance deep learning library. arXiv preprint arXiv:1912.01703.

[6] Sergeev, A., & Del Balso, M. (2018). Horovod: fast and easy distributed deep learning in TensorFlow. arXiv preprint arXiv:1802.05799.

[7] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. In the 2nd USENIX workshop on hot topics in cloud computing (HotCloud 10).

[8] Ghodsi, A., Zaharia, M., Hindman, B., Konwinski, A., Shenker, S., & Stoica, I. (2011). Dominant resource fairness: Fair allocation of multiple resource types. In the 8th USENIX symposium on networked systems design and implementation (NSDI 11).

[9] Mao, M., Li, J., & Humphrey, M. (2010, October). Cloud auto-scaling with deadline and budget constraints. In 2010, 11th IEEE/ACM International Conference on Grid Computing (pp. 41-48). IEEE.

[10] Hennessy, J. L., & Patterson, D. A. (2011). Computer architecture: a quantitative approach. Elsevier.

[11] Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., ... & Yoon, D. H. (2017, June). In-datacenter performance analysis of a tensor processing unit. In Proceedings of the 44th annual international symposium on computer architecture (pp. 1-12).

[12] Gupta, U., Wu, C. J., Wang, X., Naumov, M., Reagen, B., Brooks, D., ... & Zhang, X. (2020, February). The architectural implications of Facebook's DNN-based personalised recommendation. In 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA) (pp. 488-501). IEEE.

[13] Rasley, J., Rajbhandari, S., Ruwase, O., & He, Y. (2020, August). Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters, in Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining (pp. 3505-3506).

[14] Kherraf, N., Alameddine, H. A., Sharafeddine, S., Assi, C. M., & Ghrayeb, A. (2019). Optimized provisioning of edge computing resources with heterogeneous workload in IoT networks. IEEE Transactions on Network and Service Management, 16(2), 459-474.

[15] Ma, X., Wang, S., Zhang, S., Yang, P., Lin, C., & Shen, X. (2019). Cost-efficient resource provisioning for dynamic requests in cloud-assisted mobile edge computing. IEEE Transactions on Cloud Computing, 9(3), 968-980.

[16] Verma, N., Jia, H., Valavi, H., Tang, Y., Ozatay, M., Chen, L. Y., & Deaville, P. (2019). In-memory computing: Advances and prospects. IEEE solid-state circuits magazine, 11(3), 43-55.

[17] Ielmini, D., & Wong, H. S. P. (2018). In-memory computing with resistive switching devices. Nature Electronics, 1(6), 333-343.

[18] Karataş, G., Can, F., Doğan, G., Konca, C., & Akbulut, A. (2017, September). Multi-tenant architectures in the cloud: A systematic mapping study. In 2017 International Artificial Intelligence and Data Processing Symposium (IDAP) (pp. 1-4). IEEE.

[19] Mangla, N., Singh, M., & Rana, S. K. (2016). Resource Scheduling in Cloud Environments: A Survey. Advances in Science and Technology. Research Journal, 10(30), 38-50.

[20] Zhan, Z. H., Liu, X. F., Gong, Y. J., Zhang, J., Chung, H. S. H., & Li, Y. (2015). Cloud computing resource scheduling and a survey of its evolutionary approaches. ACM Computing Surveys (CSUR), 47(4), 1-33.