To virtualize or not to virtualize AI Infrastructure: A perspective

Seetharami Seelam, Apoorve Mohan, Ming-Hung Chen, I-Hsin Chung
IBM Research
Yorktown Heights, NY

ABSTRACT

Modern data-driven applications (such as AI training, Inference) are powered by Artificial Intelligence (AI) infrastructure. AI infrastructure is often available as bare-metal machines (BMs) in on-premise clusters but as virtual machines (VMs) in most public clouds. Why is this dichotomy of BMs on-prem and VMs in public clouds? What would it take to deploy VMs on AI Systems while delivering baremetal-equivalent performance? We will answer these questions based on experiences building and operationalizing a large-scale AI system called Vela in IBM Cloud. Vela is built on open-source Linux KVM and QEMU technologies where we are able to deliver near-baremetal (within 5% of BM) performance inside VMs. VM-based AI infrastructure not only affords BM performance but also provides cloud characteristics such as elasticity and flexibility in infrastructure management.

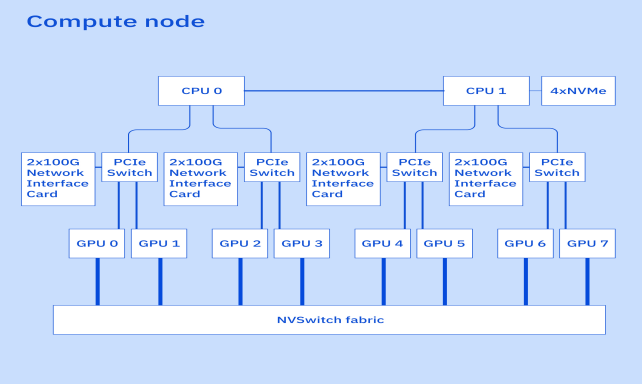

Figure 1: Compute node architecture of a typical AI System

1 WHY THE DICHOTOMY OF BMS ON-PREM AND VMS IN CLOUD?

Traditionally high-performance computing (HPC) systems [12, 13] are built with Baremetal Metal compute nodes, high performance networking and distributed file systems. HPC systems happen to perform very well for AI workloads, as a result baremetal systems are used on-prem for AI. In addition, on-prem hardware tends to be very static, typically booted once, made part of an HPC scheduler and end users simply submit jobs to systems that may run for days and months on end. The system and software stack is fixed by the system administrators, and it is only updated on scheduled time windows in the order of months to years. As a result, BM is perfectly sufficient for on-prem private clusters.

On the other hand, cloud systems [1–4] started with virtual machines for general purpose computing and started offering AI capabilities using the same VMs. In addition, public cloud AI systems are used in a very dynamic way. They are often provisioned and re-provisioned to different customers potentially multiple times a day. Customers require the ability to customize the systems software (including the operating system) and the user software (such as TensorFlow, Pytorch, CUDA) to meet the needs of their developers. For this reason, cloud providers use VM as a deployment unit and deploy fresh VMs across customer allocations. This provides cloud providers the flexibility to keep the bare-metal up like a private cluster but dynamically provision required software stacks to customers quickly. Customers can also scale-up and scale-down their AI clusters based on their workload requirements.

In summary, this dichotomy comes from the evolution of on-prem and Cloud systems and their usage patterns. The next key question is what would it take to expose AI capabilities using virtual machines?

2 HOW TO ENABLE AI INFRASTRUCTURE IN VIRTUAL MACHINES?

We set out to investigate how to enable AI infrastructure with Linux KVM. We hypothesize that configuring virtual machines with GPU and network-virtual-functions passthrough with KVM can significantly enhance the performance of AI training workloads. PCIe device passthrough enables virtual machines (VMs) to directly access the physical devices installed on the host system. As a result, applications running inside VMs can take advantage of the full power of devices like GPUs and enable performant execution of applications such as gaming, CAD, and machine learning.

However, as shown in Figure 1, modern AI compute nodes tend to have complex intra-node topology with multiple pci-e switches connecting GPUs, network cards, and CPUs and thus can be intricate to virtualize. Moreover, when virtualizing such a compute node, we noticed that the out-of-the-box system deployment results in poor AI workload performance and network performance (see Figure 2 VM Default columns). We provisioned and compared KVM-QEMU-based VMs for Transmission Control Protocol (TCP), RDMA over Converged Ethernet (RoCE), and GPU Direct RoCE communication models with default configuration and optimized configuration. The optimized configuration achieves 2-10x improvement in network performance over the default configuration (Figure 2).

Enabling a performant VM-based execution environment for AI workloads required configuration changes at different system layers [8]. Specific optimizations were made in

- the system BIOS (enable IOMMU, ACS, and SRIOV support),
- the network adapter (disable relaxed PCI ordering, increase maximum accumulative outstanding read bytes, and enable selective repeat, direct access to ATS from the NIC to GPU buffers using PCIe peer-to-peer transactions, and ATS),

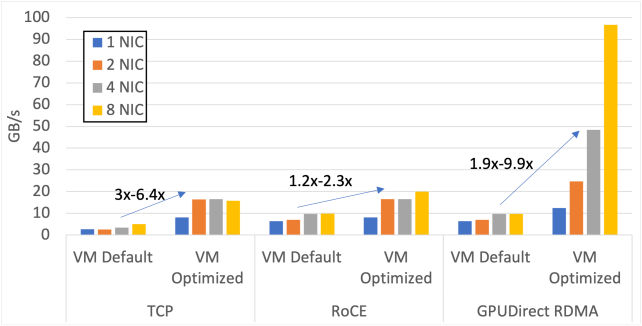

Figure 2: Performance tuning significantly improves all_reduce_perf throughput

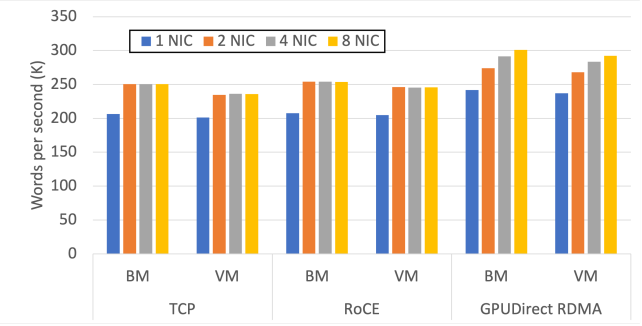

Figure 3: Acceptable virtualization overhead for NMT-12 two-node training job

- the hypervisor (enable NVFs, huge pages, ACS on the PCI controllers, and ATS on the NVFs, and increase maximum PCI read request size to 4KB),
- the guest XML (enable huge pages, NUMA domains, Device-NUMA mapping, host-physical-bit model for large memory VMs, and ATS on PCI controllers), and
- the guest operating system (increase maximum PCI read request size to 4KB) configurations.

For evaluating these optimizations, we use two compute nodes each with 8-NVIDIA A100-80GB GPUs and 4-Mellanox-CX-6Dx Dual Port cards (i.e., 800 Gbps aggregate b/w), with the hypervisor and guest running Ubuntu 20.04 (with Linux 5.4) operating system, and the latest software stack from NVIDIA and Mellanox.

We ran the NMT-12 [9] model training job on single-node in VM and BM with the same configuration to demonstrate the virtualization overhead on real-world applications. The result shows that we can achieve 147K words-per-second (WPS) on BM and 140K WPS inside the VM, i.e., a virtualization overhead of about 5%. We also evaluated BM and VM performance with Cuda Samples [5] (bandwidthtest, conjugrategradient, eigentxt, matrixmul, mergesort, p2pbandwidthlatency, simplemultigpu, simplep2p, traspose, unifiedmemoryperf), BERT-Large [6], MegaTron [11], and T5 11B [10], and the overhead of VM is less than 0% to a maximum of 5%. Less than 0% meaning that VM execution is faster and this is due to configurations such as large pages typically set for VMs as default but not in BM.

For distributed AI training workloads, the network virtualization must be performant. To benchmark the network performance, we executed `all_reduce_perf` from *nccl-tests* (a micro-benchmark suite provided by NVIDIA) to evaluate the network performance with TCP, RoCE, and NVIDIA GPUDirect RoCE protocols. Figure 2 shows that the optimizations/changes we applied at different system layers improved the network performance significantly. We also executed the distributed NMT-12 model training job to get an initial understanding. The result in Figure 3 demonstrates that we can achieve similar performance in VM and BM environments. We also tested a few other distributed AI training jobs, and the performance loss is less than 5% in general.

3 CONCLUSION

In this paper, we explained the reasons for BM on-prem and VMs in public cloud for AI infrastructure. For the first time, we explained the optimizations to enable AI infrastructure as VMs with open-source Linux KVM and achieve near-baremetal performance for micro-benchmarks and real applications. This techniques are part of Vela [7], our cloud native supercomputer inside IBM public cloud. We hope that practitioners will leverage these techniques and build AI systems with virtualization to bring the agility of cloud to on-prem AI infrastructure.

REFERENCES

[1] 2023. AWS Recommended GPU Instances. https://docs.aws.amazon.com/dlami/latest/devguide/gpu.html
[2] 2023. GCE GPU Platforms. https://cloud.google.com/compute/docs/gpus
[3] 2023. GPU optimized virtual machine sizes. https://learn.microsoft.com/en-us/azure/virtual-machines/sizes-gpu
[4] 2023. IBM Cloud GPU Servers. https://www.ibm.com/cloud/gpu
[5] 2023. NVIDIA CUDA Samples. https://github.com/NVIDIA/cuda-samples
[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:cs.CL/1810.04805
[7] Talia Gershon, Seetharami Seelam, Jay Jubran, Eran Gampel, and Drew Thorstensen. 2023. Why we built an AI supercomputer in the cloud. https://research.ibm.com/blog/AI-supercomputer-Vela-GPU-cluster
[8] Apoorve Mohan and Matthew Sheard. 2022. How to Deploy a High-performance Distributed AI Training Cluster with NVIDIA GPUs and KVM. https://www.nvidia.com/en-us/on-demand/session/gtcspring22-s42633/
[9] Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018. Scaling Neural Machine Translation. arXiv:cs.CL/1806.00187
[10] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. arXiv:cs.LG/1910.10683
[11] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. arXiv:cs.CL/1909.08053
[12] C. B. Stunkel, R. L. Graham, G. Shainer, M. Kagan, S. S. Sharkawi, B. Rosenburg, and G. A. Chochia. 2020. The high-speed networks of the Summit and Sierra supercomputers. *IBM Journal of Research and Development* 64, 3/4 (2020), 3:1–3:10. https://doi.org/10.1147/JRD.2020.2967330
[13] Junqi Yin, Shubhankar Gahlot, Nouamane Laanait, Ketan Maheshwari, Jack Morrison, Sajal Dash, and Mallikarjun Shankar. 2019. Strategies to Deploy and Scale Deep Learning on the Summit Supercomputer. In *2019 IEEE/ACM Third Workshop on Deep Learning on Supercomputers (DLS)*. 84–94. https://doi.org/10.1109/DLS49591.2019.00016