Cumulus Cloud - Project Description

Problem Statement:
GPU markets are inefficient. Across the world, vast amounts of compute power sit idle while GPU access remains expensive and fragmented. The same virtualization and scheduling advancements that made CPU-based products like AWS Lambda and Kubernetes successful have not been mirrored for GPUs.

Our Solution:
We make GPUs as liquid as possible. By separating GPUs from virtual machines and virtualizing them at the software level, we enable a new layer of flexibility that captures underutilized compute across the globe. You can train or infer from any laptop or VM using our **s**hared pool of virtualized GPUs, keeping costs low and efficiency high.

Our product aims to be the most flexible market. By defining price or performance metrics on a deployment, users can customize how much they want to pay vs the priority of their jobs.

Ex: an Nvidia A100 might rent for $1/hr/GPU at market price. If your budget is $0.80/hr/GPU and you're willing to tolerate a few extra milliseconds of latency as your job contends with higher bidders, you can rent at that price. The result is a dynamic, demand-driven market that makes cutting-edge GPUs accessible at flexible rates.

Our product is also cloud agnostic. This means our pool of GPUs can be as cheap as possible and connect compute across different platforms (AWS, Lambda Labs, etc.) to assemble the cheapest and most performant cluster possible. Host GPUs can come from anywhere across the globe, allowing us to connect to smaller datacenters and rigs that want to list on our marketplace. This allows for even cheaper prices that we can pass on to customers.

Current Progress:
Over the past few weeks, significant progress has been made validating the feasibility of GPU virtualization. This demo video successfully transmits and executes CUDA calls for matrix multiplication, memory allocation, and deallocation across networked machines, proving that GPU workloads can be shared and routed remotely. I also implemented PyTorch call interception, allowing existing deep learning workloads to run on our backend with no code changes. Our core abstraction layer is already coming together.

Initial Audience:

1. Smaller Scale ML Devs and Researchers
    a. These users fine-tune open-source models (Llama, Stable Diffusion, etc.) and often can't justify full-time GPU rentals.
    b. They need fractional, on-demand access to pay only for the compute they actually use.
    c. We allow them to rent high-end GPUs (A100s, H100s) at flexible rates, without committing to a full instance or worrying about interruptions.
2. Startups and Small ML Labs
    a. Early-stage companies running training/inference workloads but lacking dedicated infrastructure.
    b. They want predictable access to GPUs with the flexibility to scale up or down quickly.
    c. Our pricing and performance-based job scheduling lets them choose between speed or savings, similar to how AWS offers spot vs. on-demand but with finer granularity and cross-cloud liquidity.
3. Academic and Research Institutions
    a. Universities and research labs that share limited GPUs.
    b. Our virtualization layer lets them maximize utilization by pooling resources and efficiently running multiple users concurrently.
    c. This reduces idle time, queues, and admin overhead.

Costs of This Solution:

This technology comes with its own drawbacks that we need to address as we build

1. Network Latency: We are challenging the notion that a CPU and GPU need fast physically connected communication. This belief is outdated given the power of current GPUs and how much computation is done purely on the card itself. Still, network latency is a big factor. We plan to use the fastest networking protocols (QUIC, etc.) and utilize batching of jobs to speed up networking and amortize its cost on larger scales. In the future, we can think about owning our own infra so networking speed between CPU and GPU is lowered with physical connection.
2. Context Switching: The most costly operation in our setup is a context switch (i.e. Job A gets evicted from GPU and all memory is cleared so we can serve Job B). Every context switch involves destroying and recreating CUDA contexts, reloading model weights, and reallocating GPU memory buffers. We want to find a balanced approach so we can keep

queue times low and perform fewer context switches. We plan to use host based caching so data is preserved after eviction.

   a. Our approach aims to decouple eviction from total teardown. When a job is paused or evicted, instead of immediately destroying all GPU state, we offload its data to host memory or SSD cache on the same machine. This allows quick resumption of jobs with minimal re-upload latency. For example, large model weights can remain in pinned host memory after eviction, while only transient tensors are freed. When the job is rescheduled onto the same or nearby GPU, it can recover its context in milliseconds rather than seconds.
   b. To safely end a job, we plan to use a reference-counted allocation tracker across all device mallocs. Each time the client allocates GPU memory via our virtual layer, the host increments a counter. When the client explicitly frees it or disconnects, the counter decrements. Once all allocations for a job reach zero, we can safely destroy the job's context and release cached host memory asynchronously.
   c. In the future, we can also implement ML based eviction policies that prioritize which cached jobs to retain based on recent access patterns and probabilities.

3. Job Matching: Job matching poses a potential bottleneck as well. We plan to get around this by having multiple load balancers that scale with the customers. Again, with intelligent batching, we can further amortize the cost of job matching to a GPU.

Why This Hasn't Been Done:

Large cloud providers are not incentivized to make their products more efficient. People are left to either rent from them for market price or spin up their own infrastructure which is much more unfeasible. The larger providers also benefit from inefficient usage since they get paid by time rented as opposed to time used.

Smaller cloud providers don't want to waste capital or time on building out this technical solution when they can rely on smaller enterprise contracts and individual customers in certain niches to stay immediately profitable. This fails to look at the bigger picture- that true efficiency and scalability in GPU compute come from liquidity, not ownership. By focusing only on short-term margins, these providers miss the opportunity to create a global, shared GPU economy where utilization is maximized and costs are driven down naturally.

Progress in Industry:

Individual Host Marketplaces (Vast.ai, TensorDock):

These marketplaces aim to connect individual hosts around the world and allow unused rigs to be listed for rent. This solves the problem of connecting individual hosts so underused compute power can be rented. However, there are still drawbacks:

1. Interruptible defaults: Because many hosts are individual, the risk of the host going offline is large. For many jobs you cannot assume uninterrupted execution.
2. Multi-GPU orchestration is weak: While Vast allows searching for multiple GPUs, connecting multiple of them to form a cluster is difficult and user-managed. Running large jobs that need multiple GPUs to be managed and kept up is difficult.
3. Lack of persistent context / caching: In the marketplace model, when you stop using a host or it gets reclaimed, you typically lose everything (data in memory, caches, etc.). This means you need jobs to be fault-tolerant or restartable rather than seamlessly resuming with preserved state.

We introduce host-based caching and lightweight GPU virtualization so our platform can tolerate a baseline level of “chaos” (transient disconnects, evictions, and context switches) without forcing jobs to restart from scratch. Rather than treating these interruptions as edge cases, we treat them as normal operating conditions and design around them. Thus, we are built to handle these faults rather than treating them as edge cases. Additionally, we allow for even better prices and efficiency/utilization with the virtualization of our GPUs which **has not been done with these products**.