Sit Back and Relax with Fault Awareness and Robust Instant Recovery for Large Scale AI Workloads

DaoCloud Fanshi Zhang, Kebe Liu

KubeCon CloudNativeCon THE LINUX FOUNDATION OPEN SOURCE SUMMIT AI_dev


DaoCloud
Kubernetes

Kebe Liu

Senior software engineer

kebe7jun

Fanshi Zhang

Senior software engineer

nekomeowww

Let’s jump right into the rabbit hole

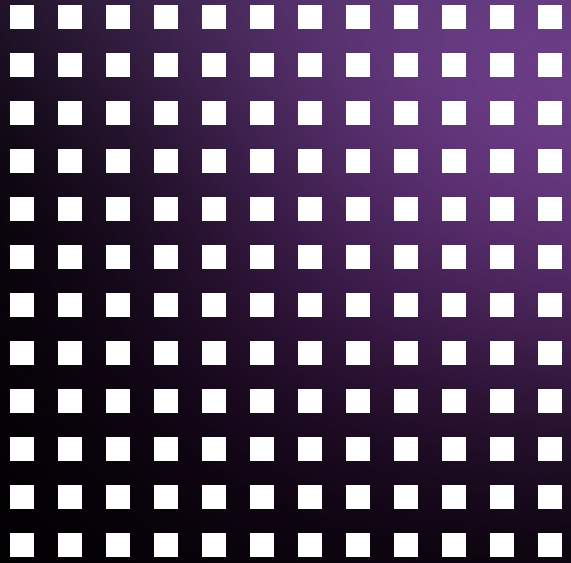
Machine Learning 101

Let's start with the basics

To	be	or	not	to	be	this	is	the	word
5.4	6.2	5.8	6.1	5.5	6.3	5.9	6.0		6.4
7.1	8.5	6.9	7.2	7.4	8.6	7.0	7.1		8.7
6.0	5.9	7.0	6.5	6.1	6.0	6.9	6.8		5.8
5.4	5.1	4.8	5.2	5.3	5.2	4.9	5.0		5.3
4.2	4.5	4.1	3.9	4.3	4.6	4.2	4.0	...	4.7
6.4	6.7	6.3	6.0	6.5	6.8	6.4	6.1		6.9
4.3	4.8	4.0	4.6	4.4	4.9	4.1	4.5		5.0
8.8	7.3	7.9	8.1	8.5	7.4	8.0	8.2		7.5

Machine Learning 101

Let's start with the basics



GPU

CPU

Distributed Training 101

Why even bother with distributed systems?

LLMs are large

Single GPU doesn't fit in

Distribute them to GPU clusters

Failures occur

Why failures occur?

What so criticals are those so irreversibles

Hardware issues

Network issues

Framework bugs

Irreversible hardwares - GPU & PCIe

GPU issues, PCIe issues, RDMA issues, so many of them

```
[ 4254.197816] NVRM: GPU at PCI:0000:5d:00: GPU-f1906b9b-557a-e961-045c-9fe4be3ce012
[ 4254.197854] NVRM: GPU Board Serial Number: 1653923026510
[ 4254.197860] NVRM: Xid (PCI:0000:5d:00): 79, pid='<unknown>', name=<unknown>, GPU has fallen off the bus.
[ 4254.197871] NVRM: GPU 0000:5d:00.0: GPU has fallen off the bus.
[ 4254.197878] NVRM: GPU 0000:5d:00.0: GPU serial number is 1653923026510.
[ 4254.197913] NVRM: A GPU crash dump has been created. If possible, please run
               NVRM: nvidia-bug-report.sh as root to collect this data before
               NVRM: the NVIDIA kernel module is unloaded.
```

```
[14387.209961] NVRM: The NVIDIA GPU 0000:5d:00.0
               NVRM: (PCI ID: 10de:2330) installed in this system has
               NVRM: fallen off the bus and is not responding to commands.
[14387.210134] nvidia: probe of 0000:5d:00.0 failed with error -1
[14387.274303] NVRM: The NVIDIA probe routine failed for 1 device(s).
[14387.274366] NVRM: loading NVIDIA UNIX x86_64 Kernel Module  525.125.06  Tue May 30 05:11:37 UTC 2023
[14387.511008] nvidia_uvm: module uses symbols from proprietary module nvidia, inheriting taint.
[14387.548839] nvidia-uvm: Loaded the UVM driver, major device number 502.
[14387.573380] nvidia-modeset: Loading NVIDIA Kernel Mode Setting Driver for UNIX platforms  525.125.06  Tue May 30 04:
```

Irreversible networks - NCCL

GPU issues, PCIe issues, RDMA issues, so many of them

```
node-1:185:1027 [7] NCCL INFO [Service thread] Connection closed by localRank 0
node-1:180:1028 [2] NCCL INFO [Service thread] Connection closed by localRank 0
node-1:184:1030 [6] NCCL INFO [Service thread] Connection closed by localRank 0
node-1:178:828 [0] NCCL INFO comm 0x55555da97ba0 rank 16 nranks 32 cudaDev 0 busId 1b000 - Abort COMPLETE
[E ProcessGroupNCCL.cpp:481] Some NCCL operations have failed or timed out. Due to the asynchronous nature of CUDA kerr
[E ProcessGroupNCCL.cpp:487] To avoid data inconsistency, we are taking the entire process down.
[E ProcessGroupNCCL.cpp:852] [Rank 16] NCCL watchdog thread terminated with exception: NCCL error: remote process exite
ncclRemoteError: A call failed possibly due to a network error or a remote process exiting prematurely.
Last error:
NET/IB : Got completion from peer 10.42.0.2<47534> with error 5, opcode 48, len 32764, vendor err 244
terminate called after throwing an instance of 'std::runtime_error'
  what():  [Rank 16] NCCL watchdog thread terminated with exception: NCCL error: remote process exited or there was a r
ncclRemoteError: A call failed possibly due to a network error or a remote process exiting prematurely.
Last error:
NET/IB : Got completion from peer 10.42.0.2<47534> with error 5, opcode 48, len 32764, vendor err 244
[2024-07-25 06:36:18,293] torch.distributed.elastic.multiprocessing.api: [WARNING] Sending process 179 closing signal S
[2024-07-25 06:36:18,294] torch.distributed.elastic.multiprocessing.api: [WARNING] Sending process 180 closing signal S
```

Irreversible softwares - PyTorch

GPU issues, PCIe issues, RDMA issues, so many of them

```
Traceback (most recent call last):
  File "/home/neko/Git/test/test1.py", line 13, in <module>
    out = model(torch.FloatTensor(src), src_key_padding_mask = mask, is_causal = True)
  File "/opt/miniforge/conda/lib/python3.10/site-packages/torch/nn/modules/module.py", line 1518, in _wrapped_call_impl
    return self._call_impl(*args, **kwargs)
  File "/opt/miniforge/conda/lib/python3.10/site-packages/torch/nn/modules/module.py", line 1527, in _call_impl
    return forward_call(*args, **kwargs)
  File "/opt/miniforge/conda/lib/python3.10/site-packages/torch/nn/modules/transformer.py", line 387, in forward
    output = mod(output, src_mask=mask, is_causal=is_causal, src_key_padding_mask=src_key_padding_mask_for_layers)
  File "/opt/miniforge/conda/lib/python3.10/site-packages/torch/nn/modules/module.py", line 1518, in _wrapped_call_impl
    return self._call_impl(*args, **kwargs)
  File "/opt/miniforge/conda/lib/python3.10/site-packages/torch/nn/modules/module.py", line 1527, in _call_impl
    return forward_call(*args, **kwargs)
  File "/opt/miniforge/conda/lib/python3.10/site-packages/torch/nn/modules/transformer.py", line 678, in forward
    return torch._transformer_encoder_layer_fwd(
RuntimeError: expected scalar type BFloat16 but found Float
```

Where is the so called "Irreversible" issues?

What is happening?

For a distributed PyTorch training job...

- Instead of `Deployment` like workloads, they are more like `StatefulSet`
- When bootstrapping, the main node (`rank 0`) will be the first to start
- Then negotiate with other nodes (`rank != 0`) to join the training through NCCL
  - Calculate topology
  - Calculate connectivity
  - Calculate bandwidth
- Once everyone is ready, minibatch will be calculated and sent to each node
- Every step, or epoch, a Ring AllReduce or AllReduce operation will be performed across the nodes

Simulate the failure

What is happening?

Let's say we have a distributed training job running on a GPU cluster...

Main node ⬡ Worker node

Why are they hard to deal with?

Causes, reason, and potential solutions

Blackbox with in another blackbox

Distribution algorithm is purely implemented by PyTorch, NCCL itself.

Hard to debug Hard to trace Hard to make it managed Hard to make it controlled

Unreachable

Unlike nowadays Kubernetes Operators, healing, orchestrating still hard to achieve.

Hard to auto-heal Hard to auto-recover Hard to auto-mitigate

Missing block of jigsaw

Detecting failures of drivers, hardwares, GPUs, or even network is still a challenge.

Hard to know root cause Hard to collect needed NPD events & logs Lack of observability

Beyond all of these issues

Checkpoints, weights are more even critical

Checkpoint

Storage

Checkpoints are large

LLAMA 2 has roughly 83GB of checkpoint files

Limited bandwidth of NFS, shared Volumes, RDMA

Saving 80G and above levels checkpoint files require high speed of IO to reduce the downtime

Mitigation requires transferring across nodes

If one of the GPU node went down, hundreds GB of files must be transferred to another node

Tune the factors

Checkpoints, weights are more even critical

$$
\begin{aligned}
T = & T_{\text{train}} + (T_{\text{save}} \times G) \\
& + (T_{\text{diagnostic}} \times N \times P) \\
& + (T_{\text{reconcile}} \times N \times P) \\
& + (T_{\text{transfer}} \times C) \\
& + (T_{\text{load}} \times G)
\end{aligned}
$$

T_{train} = Training Epoch Elapsed
T_{save} = Checkpoint Saving
$T_{\text{diagnostic}}$ = Diagnostic Time per Node/Pod
$T_{\text{reconcile}}$ = Reorchestrating Time per Node/Pod
T_{transfer} = Checkpoint Transferring Time per File
T_{load} = Checkpoint Loading Time per GPU

N = Number of Nodes
P = Number of Pods
C = Number of Checkpoint Files
G = Number of GPUs

Tune the factors

What are the things we can improve

Reduce diagnostic time

Reduce reconcile time

Speed up checkpoints

SOTA & ~~Reseaarches~~

Some of the state of the art blogs, tryouts, and researches

Hardware issues:

> The main type of issue encountered during training were hardware failures. As this was a new cluster with about 400 GPUs, on average we were getting 1-2 GPU failures a week. We were saving a checkpoint every 3h (100 iterations) so on average we would lose 1.5h of training on hardware crash.[1]

Stuck:

> Sometimes the training gets stuck despite a crashing process and it won't quit, which seems to be related to some torch.launch and possibly NCCL.[2]

They managed to automate most things...

> We automated most things, including recovery from hardware crashes, but sometimes a human intervention was needed as well.[1:1]

1. The Technology Behind BLOOM Training ↩ ↩
2. Training chronicles ↩

SOTA & Researches

Some of the state of the art blogs, tryouts, and researches

Meta

They got a massive 24,000 GPU Cluster...

> We performed training runs on two custom-built 24K GPU clusters. To maximize GPU uptime, we developed an advanced new training stack that automates error detection, handling, and maintenance.[1]

To solve the problems that we introduced...

> We also greatly improved our hardware reliability and detection mechanisms for silent data corruption, and we developed new scalable storage systems that reduce overheads of checkpointing and rollback.[1:1]

After improvements...

> Those improvements resulted in an overall effective training time of more than 95%. Combined, these improvements increased the efficiency of Llama 3 training by ~three times compared to Llama 2.[1:2]

1. Introducing Meta Llama 3: The most capable openly available LLM to date ↩ ↩ ↩

Who else have tried

Some of the state of the art blogs, tryouts, and researches

- FastPersist: Accelerating Model Checkpointing in Deep Learning
- DLRover-RM: Resource Optimization for Deep Recommendation Models Training in the Cloud

DLRover

DLRover
Deep Learning Rover

ANT GROUP

So what have we done?

So what have we done?

Kcover

So what have we done?

Kcover

One simple intall-to-go plugin solution combines both NPD (Node Problem Detector) and operator

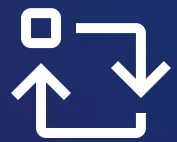
Features

We solved the problems, partially

Firewatch of workloads

Enhanced observability

Periodic inspection

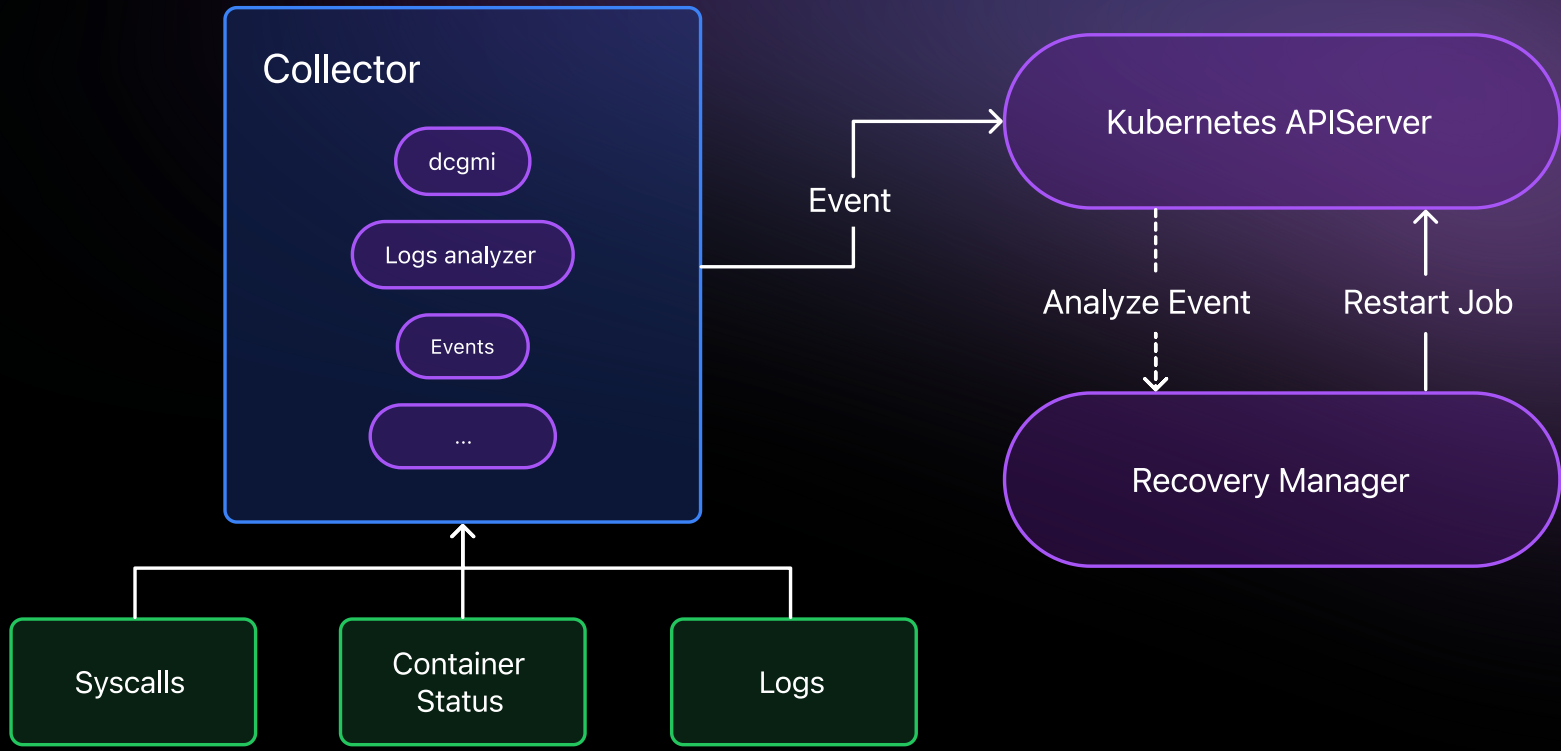


Cascading shutdown

Intelli-migration

Architecture

Overview of the internals



How to use it?

Easy. simple, two commands

Install

```
helm install kcover kcover/kcover --namespace kcover-system
```

Label the should watched resources - With `kubectl`

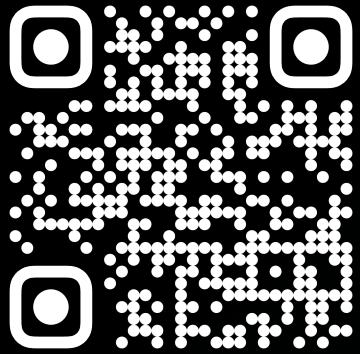
```
kubectl label pod pytorchjob-example-0 kcover.io/cascading-recovery=true
```

Label the should watched resources - With `yaml`

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pytorchjob-example-0
  labels:
    kcover.io/cascading-recovery: "true"
  # ...
```

Let’s build it together

Open sourced, already



BaizeAI/kcover

KubeCon CloudNativeCon THE LINUX FOUNDATION OPEN SOURCE SUMMIT AI_dev

Futures

Foresight from our perspective

More advanced event analysis

- `dcgmi` integrations
- `nvidia-smi` integrations
- `nccl-test` integrations
- cuDNN error analysis
- TensorFlow error analysis
- Jax error analysis

More types of analysis

- IO
- Node Problem Detector integrations
- NCCL integrations

More integrated solutions

- Operator
- Dedicated CRD for configuring informer
- Dedicated CRD for configuring mitigator
- Automatically watch for CRD with selectors
- Better cascading mitigation

To community

Let's improve it together

- Propose universal trainer health check implementation for PyTorch
- Together to build better analysis and root cause debugging on top of Kubernetes
  - Perhaps we can corporate with NVIDIA
- Try to expose more observability metrics for tracing, logging, and monitoring
  - Proper implementation of health check endpoint from distributed trainers for analysis
  - Prometheus & OpenTelemetry integrations
- How about implement a stateless negotiator layer on top of TensorFlow, PyTorch and Jax?

Thank you


KubeCon


CloudNativeCon


THE LINUX FOUNDATION
OPEN
SOURCE
SUMMIT


AI_dev