(distributed-serving)=

Distributed Inference and Serving

How to decide the distributed inference strategy?

Before going into the details of distributed inference and serving, let's first make it clear when to use distributed inference and what are the strategies available. The common practice is:

- **Single GPU (no distributed inference)**: If your model fits in a single GPU, you probably don't need to use distributed inference. Just use the single GPU to run the inference.
- **Single-Node Multi-GPU (tensor parallel inference)**: If your model is too large to fit in a single GPU, but it can fit in a single node with multiple GPUs, you can use tensor parallelism. The tensor parallel size is the number of GPUs you want to use. For example, if you have 4 GPUs in a single node, you can set the tensor parallel size to 4.
- **Multi-Node Multi-GPU (tensor parallel plus pipeline parallel inference)**: If your model is too large to fit in a single node, you can use tensor parallel together with pipeline parallelism. The tensor parallel size is the number of GPUs you want to use in each node, and the pipeline parallel size is the number of nodes you want to use. For example, if you have 16 GPUs in 2 nodes (8 GPUs per node), you can set the tensor parallel size to 8 and the pipeline parallel size to 2.

In short, you should increase the number of GPUs and the number of nodes until you have enough GPU memory to hold the model. The tensor parallel size should be the number of GPUs in each node, and the pipeline parallel size should be the number of nodes.

After adding enough GPUs and nodes to hold the model, you can run vLLM first, which will print some logs like `# GPU blocks: 790`. Multiply the number by `16` (the block size), and you can get

roughly the maximum number of tokens that can be served on the current configuration. If this number is not satisfying, e.g. you want higher throughput, you can further increase the number of GPUs or nodes, until the number of blocks is enough.

:::{note}
There is one edge case: if the model fits in a single node with multiple GPUs, but the number of GPUs cannot divide the model size evenly, you can use pipeline parallelism, which splits the model along layers and supports uneven splits. In this case, the tensor parallel size should be 1 and the pipeline parallel size should be the number of GPUs.
:::

Running vLLM on a single node

vLLM supports distributed tensor-parallel and pipeline-parallel inference and serving. Currently, we support [Megatron-LM's tensor parallel algorithm](https://arxiv.org/pdf/1909.08053.pdf). We manage the distributed runtime with either [Ray](https://github.com/ray-project/ray) or python native multiprocessing. Multiprocessing can be used when deploying on a single node, multi-node inferencing currently requires Ray.

Multiprocessing will be used by default when not running in a Ray placement group and if there are sufficient GPUs available on the same node for the configured `tensor_parallel_size`, otherwise Ray will be used. This default can be overridden via the `LLM` class `distributed_executor_backend` argument or `--distributed-executor-backend` API server argument. Set it to `mp` for multiprocessing or `ray` for Ray. It's not required for Ray to be installed for the multiprocessing case.

To run multi-GPU inference with the `LLM` class, set the `tensor_parallel_size` argument to the number of GPUs you want to use. For example, to run inference on 4 GPUs:

```python
from vllm import LLM
llm = LLM("facebook/opt-13b", tensor_parallel_size=4)
output = llm.generate("San Franciso is a")
```

To run multi-GPU serving, pass in the `--tensor-parallel-size` argument when starting the server. For example, to run API server on 4 GPUs:

```console
 vllm serve facebook/opt-13b \
     --tensor-parallel-size 4
```

You can also additionally specify `--pipeline-parallel-size` to enable pipeline parallelism. For example, to run API server on 8 GPUs with pipeline parallelism and tensor parallelism:

```console
 vllm serve gpt2 \
     --tensor-parallel-size 4 \
     --pipeline-parallel-size 2
```

Running vLLM on multiple nodes

If a single node does not have enough GPUs to hold the model, you can run the model using

multiple nodes. It is important to make sure the execution environment is the same on all nodes, including the model path, the Python environment. The recommended way is to use docker images to ensure the same environment, and hide the heterogeneity of the host machines via mapping them into the same docker configuration.

The first step, is to start containers and organize them into a cluster. We have provided the helper script <gh-file:examples/online_serving/run_cluster.sh> to start the cluster. Please note, this script launches docker without administrative privileges that would be required to access GPU performance counters when running profiling and tracing tools. For that purpose, the script can have `CAP_SYS_ADMIN` to the docker container by using the `--cap-add` option in the docker run command.

Pick a node as the head node, and run the following command:

```console
bash run_cluster.sh \
        vllm/vllm-openai \
        ip_of_head_node \
        --head \
        /path/to/the/huggingface/home/in/this/node \
        -e VLLM_HOST_IP=ip_of_this_node
```

On the rest of the worker nodes, run the following command:

```console
bash run_cluster.sh \
```

```
        vllm/vllm-openai \
        ip_of_head_node \
        --worker \
        /path/to/the/huggingface/home/in/this/node \
        -e VLLM_HOST_IP=ip_of_this_node
```

Then you get a ray cluster of containers. Note that you need to keep the shells running these commands alive to hold the cluster. Any shell disconnect will terminate the cluster. In addition, please note that the argument `ip_of_head_node` should be the IP address of the head node, which is accessible by all the worker nodes. The IP addresses of each worker node should be specified in the `VLLM_HOST_IP` environment variable, and should be different for each worker node. Please check the network configuration of your cluster to make sure the nodes can communicate with each other through the specified IP addresses.

Then, on any node, use `docker exec -it node /bin/bash` to enter the container, execute `ray status` to check the status of the Ray cluster. You should see the right number of nodes and GPUs.

After that, on any node, you can use vLLM as usual, just as you have all the GPUs on one node. The common practice is to set the tensor parallel size to the number of GPUs in each node, and the pipeline parallel size to the number of nodes. For example, if you have 16 GPUs in 2 nodes (8 GPUs per node), you can set the tensor parallel size to 8 and the pipeline parallel size to 2:

```console
 vllm serve /path/to/the/model/in/the/container \
     --tensor-parallel-size 8 \
     --pipeline-parallel-size 2
```

```

You can also use tensor parallel without pipeline parallel, just set the tensor parallel size to the number of GPUs in the cluster. For example, if you have 16 GPUs in 2 nodes (8 GPUs per node), you can set the tensor parallel size to 16:

```console
vllm serve /path/to/the/model/in/the/container \
    --tensor-parallel-size 16
```

To make tensor parallel performant, you should make sure the communication between nodes is efficient, e.g. using high-speed network cards like Infiniband. To correctly set up the cluster to use Infiniband, append additional arguments like `--privileged -e NCCL_IB_HCA=mlx5` to the `run_cluster.sh` script. Please contact your system administrator for more information on how to set up the flags. One way to confirm if the Infiniband is working is to run vLLM with `NCCL_DEBUG=TRACE` environment variable set, e.g. `NCCL_DEBUG=TRACE vllm serve ...` and check the logs for the NCCL version and the network used. If you find `[send] via NET/Socket` in the logs, it means NCCL uses raw TCP Socket, which is not efficient for cross-node tensor parallel. If you find `[send] via NET/IB/GDRDMA` in the logs, it means NCCL uses Infiniband with GPU-Direct RDMA, which is efficient.

:::{warning}
After you start the Ray cluster, you'd better also check the GPU-GPU communication between nodes. It can be non-trivial to set up. Please refer to the [sanity check script](#troubleshooting-incorrect-hardware-driver) for more information. If you need to set some environment variables for the communication configuration, you can append them to the
```

`run_cluster.sh` script, e.g. `-e NCCL_SOCKET_IFNAME=eth0`. Note that setting environment variables in the shell (e.g. `NCCL_SOCKET_IFNAME=eth0 vllm serve ...`) only works for the processes in the same node, not for the processes in the other nodes. Setting environment variables when you create the cluster is the recommended way. See <gh-issue:6803> for more information.

:::

:::{warning}

Please make sure you downloaded the model to all the nodes (with the same path), or the model is downloaded to some distributed file system that is accessible by all nodes.

When you use huggingface repo id to refer to the model, you should append your huggingface token to the `run_cluster.sh` script, e.g. `-e HF_TOKEN=`. The recommended way is to download the model first, and then use the path to refer to the model.

:::

:::{warning}

If you keep receiving the error message `Error: No available node types can fulfill resource request` but you have enough GPUs in the cluster, chances are your nodes have multiple IP addresses and vLLM cannot find the right one, especially when you are using multi-node inference. Please make sure vLLM and ray use the same IP address. You can set the `VLLM_HOST_IP` environment variable to the right IP address in the `run_cluster.sh` script (different for each node!), and check `ray status` to see the IP address used by Ray. See <gh-issue:7815> for more information.

:::