End-to-end data generation, training, and evaluation of time-series networks for electromagnetic wave propagation on a large dataset using Kubernetes

Andy G. Varner, agkgd4

***Abstract*—This project aims to optimize and expedite the training of time-series neural network models on a large dataset. We previously addressed the challenges of CPU-intensive generation of custom simulated datasets. Now we transition the training phase of the project to the Kubernetes cluster. This involves learning to attach PVCs to containers and launching Kubernetes jobs/pods with GPU resources. To assess Kubernetes' scaling capabilities, we repeat training on the small dataset on the cluster and compare the results to the initial 31-hour processing time. Subsequently, we scale up to the full dataset for further analysis. Results show that GPU acceleration on the Nautilus cluster significantly speeds up computation time, but data conditioning on a large dataset is costly with the current setup.**

I. Motivation and previous work

The purpose of this project is to develop a pipeline for data generation, training, and evaluation of a neural network for a large dataset using Kubernetes. The benefits of using the Nautilus cluster for this project are twofold: Not only does it speed up research due to the quality and quantity of available GPU and CPU resources, but it also provides access to storage otherwise unavailable. This research requires the creation of a custom dataset generated by high resolution physics simulations. The outputs of these simulations are extensive, requiring terabytes of storage not locally available.

Prior to the current project, a workflow [1] was established, as in Fig. 1, to accelerate the generation of a 10 TB dataset using Kubernetes resources, including CPU parallelization and the use of a persistent volume claim (PVC). However, the full dataset has yet to be utilized. The preliminary model was trained locally with a small subset of the data, which was transferred to local compute. The end-to-end system is illustrated in Fig. 1, demonstrating the various processes required to prepare the data for the model, conduct training, load in results, and perform evaluation.

II. Experiment / Methods

To establish the Kubernetes pipeline before scaling up the full dataset, we performed the following tasks:

- Created multiple persistent volume claims (PVCs) to replace local storage, and mounted multiple PVS to a single pod.
- Created Kubernetes containers / pods to replace local processes including a script to manage the parallelization of training 24 models.
- Allocation of GPU resources and mounting shared memory to a pod.

The resulting process is outlined in Fig. 2. For tasks involving data reduction and data preprocessing, the full, large dataset is used. To ensure correct scaling and to obtain a one-to-one timing comparison with the local process, training is conducted on the small dataset by launching parallel Kubernetes pods such that all 24 models train in parallel.

To interact with Kubernetes, we used the `gpn-mizzou-muem` namespace. For each task, a job was created to launch a single pod. Job files were generated using the Jinja2 templating library and, depending on the task, a file named `launch_{task}.py` handled the templating.

Algorithm 1 Launch `launch_{task}.py` Script

0: **procedure** LaunchTask($task$){Launches a task script with specified task - task might be reduce_data, preprocess, load_results or evaluate}
0: `// Read config file with resource limits, paths, etc.`
0: `// Get template from {task}_job.txt`
0: `// Render template`
0: `// Save job file`
0: `// Launch job using kubectl apply -f path_job`
0: **end procedure**=0

Each job launches a single pod, depending on the specified task. For data generation, we rely on the repository located at https://github.com/Kovaleski-Research-Lab/general_3x3/tree/andy_branch, and for the time series neural network, we rely on the repository located at https://github.com/Kovaleski-Research-Lab/meta_atom_rnn.git. The image required to run the code is pulled from docker hub at https://hub.docker.com/repository/docker/kovaleskilab/meep/general. For parallel jobs, we used Algorithm 2.

III. Discussion of Timing

In the current configuration, we have only parallelized the training portion, so this is the only aspect that sees a timing improvement. Comparing the timing of Fig. 1 with that of Fig. 2, the time taken per sample to reduce and preprocess data remains unchanged. However, training time improved from 26.3 hours to just 72 minutes. Loading results and evaluation, which are not parallelized, have comparable timing to the local configuration, but become bottlenecks when scaling up to a large dataset.

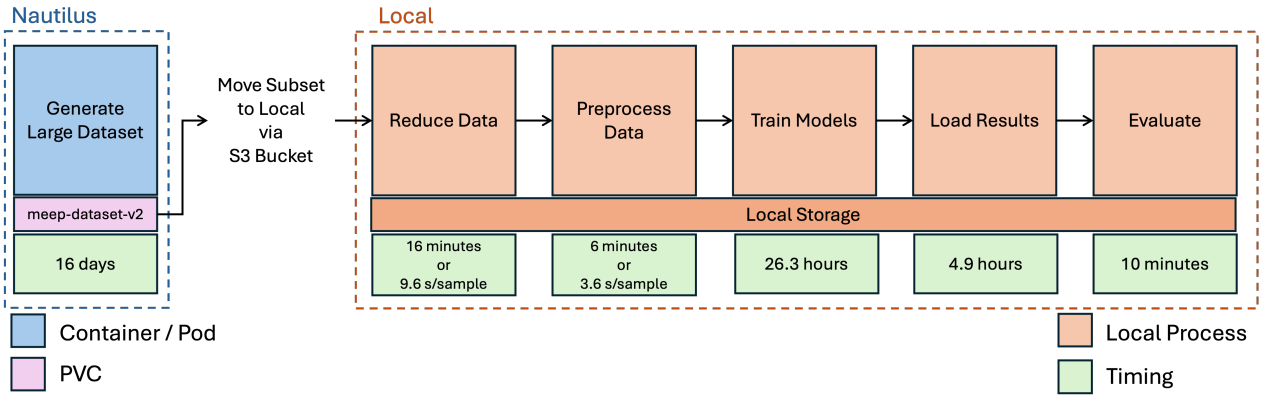

Fig. 1. Overview of Previous Experiment Pipeline: The full dataset was generated and stored using Nautilus resources. A subset of the data was moved to local compute for data conditioning, training (24 models trained sequentially), loading results, and evaluation, as well as their respective time requirements. Note: Data generation took 16 hours using Kubernetes. It is estimated that this compute would take 20 days or longer using local resources, and local storage for the dataset is unavailable.

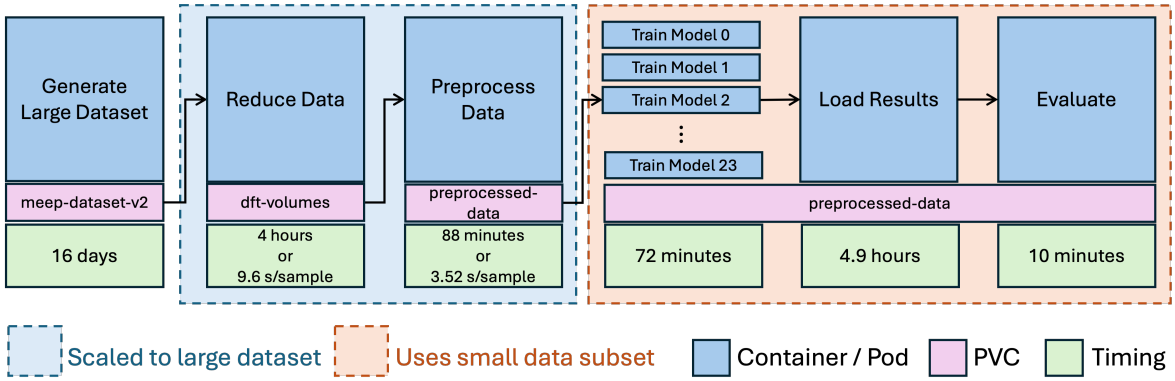

Fig. 2. End-to-end pipeline on Nautilus. Checked against Fig. 1, all local processes have been replaced by a Kubernetes pod. Local storage has been replaced by PVCs. Data reduction and preprocess steps use the full dataset, but training was conducted for the samll dataset to faciliate comparison between the local and scaled-up approaches.

Algorithm 2 Launch `launch_training.py` Script

procedure LAUNCHTASK($task$){Launches a task script with specified task - train_models}
  `// Read config file with resource limits, paths, etc.`
  `// Get template from {task}_job.txt`
  for sequence_length **in** sequence_list **do**
    `// Assign unique job name`
    `// Render template`
    `// Save job file`
    `// Launch job using kubectl apply -f path_job`
  end for
end procedure=0

It is worth noting that the training portion of the new pipeline does not use the full dataset. Instead, we repeat the use of the smaller dataset for two reasons: 1) to facilitate a one-to-one timing comparison with the local model, resulting in a notable improvement from about a day to just 72 minutes; and 2) to ensure the proper transition of the process to a Kubernetes environment, which was successfully achieved.

IV. Conclusions and Next Steps

The transition of the entire experiment pipeline to Kubernetes required the creation of additional PVCs, learning to run GPU jobs, including mounting shared memory, and figuring out how to mount multiple PVCs to a single pod. The work presented here demonstrates that parallelizing GPU jobs on Kubernetes offers a substantial speedup. Moreover, eventual training with such a large dataset would be impossible without the storage capabilities provided by the Nautilus cluster. However, conditioning the large dataset for training models emerges as a significant bottleneck, necessitating future work to parallelize this process.

Training on the small dataset allowed us to validate the scalability of the approach and observe a one-to-one timing comparison. Moving forward, the next step is to train on the full dataset to determine whether more data will lead to improved model performance.

References

[1] A. G. Varner, M. B. Lindsay, C. T. Veal, J. Shin, S. D. Kovaleski, D. T. Anderson, S. R. Price, and S. R. Price, "Time-series networks to predict electromagnetic wavefront propagation," in *Advanced Optics for Imaging Applications: UV through LWIR VIII*, vol. 12530. SPIE, Apr. 2024, pp. 110–120.