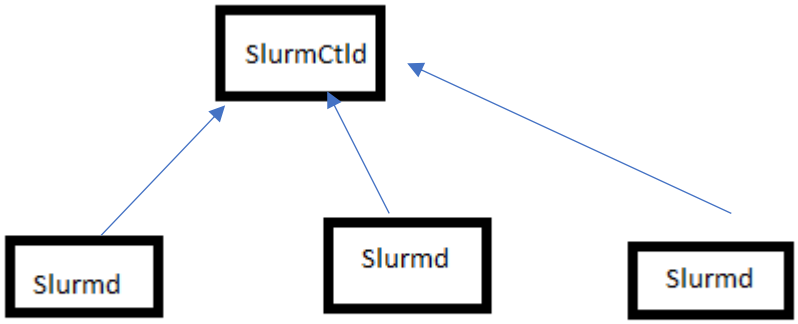
Slurm Job Failure Prediction

Slurm is a workload manager for Linux. Its used across the globe for HPC workloads that are known for their resource intensive nature. It has been used in several supercomputers.

A workload submitted to Slurm is called a Job. It can be something as trivial as printing hello world to as complex as hurricane velocity prediction.

Essentially, we can say that Slurm is a kind of system admin who upon receiving request for compute resources from a user (CPU, memory, GPU), allocates resources based on their availability and the priorities of the workloads.

Slurm has a lot of things going in the background for allocating resources to workloads. Its architecture is also a bit complex but can be summarized through the following picture.



Basically SlurmCtld (Slurm Controller Daemon) is the master while slurmds (slurm daemon) are the workers. All of these usually reside on different nodes. It looks quite similar to Kubernetes.

What if we could tell if a Slurm Job fails or not even before or atleast when the job just begins? This is what the project is all about. It was initiated by someone in my company.

Following that person's footsteps, I considered an open source dataset provided by NREL for training a model and using it for inference. In fact, this project has a messy story behind it.

In production, there wont be an NREL dataset. There will be the customer's data. NREL has collected 300,000 jobs data in three months; another one (1.2M) took nearly 2 years to generate. So the very first thing is that customers will have to wait for a pretty long time for seeing the project in action.

Second thing is that if we analyze the NREL dataset, it shows that the users who submit the jobs aren't slurm professionals; they just submit their jobs to Slurm and wait for the results. What this means is that they don't specify options. As a result, if available, Slurm just allocates all the resources that can be provided, not just the requested. Now, if we are to

say, using such predictable and “non interesting” features, that a job fails, how would that even work?(by features I mean the columns of the dataset).

Even NREL, in its Runtime prediction, has made use of XALT data which unfortunately it hasn’t published perhaps due to confidentiality. There is absolutely no dataset in the internet that can be used to train a general purpose model that can say if a job fails or not based on the characteristics of the application. So one has to turn inwards for procuring the dataset.

Ideally, if you ask me if this job will fail or not before the program can run or not, I will be able to answer to very good extent if I know what are the resources that my script would be needing. I also need to check if there is a chance of missing dependencies, code that isn’t compatible with the system, missing permissions, too much of I/O etc.

So the dataset must capture all these things. We should have access to all possible information; slurm logs, submit line and if possible, the script/the application itself. But we may not have information regarding the complete code. For that we might atleast consider the dependencies or the information of the libraries. Even the language that the script is using might be essential; python consumes more resources compared to C.

This was only part of the story. We need to have a mechanism to collect this data. I don’t know about the application related data, but for Slurm atleast, I was leaning towards an architecture based on Slurm Job Submit Plugins, PrologSlurmctld so that we can take an action as quickly as possible. However, as it turns out, a simple periodic polling mechanism that collects data is much better compared to the complex configuration. In fact, we might incur greater number of calls to SlurmCtld which may degrade the performance of the Slurm cluster. The polling mechanism is exactly what our company uses.

Overall, my initial focus would be first understand what each column in the dataset means. Then it was to understand what metrics were being collected and whether they would be available at the beginning of a job. Then slowly I understood the fundamentals of Slurm. I then pondered if the existing metric collection architecture of our company could be improved.

The following are now the points that I have learnt:

1. No changes to the existing metrics collection architecture is required.
2. During inference, JFP will collect metrics by itself. All it needs is an intimation about a new job. The new job, in order to be considered by JFP, has to be in one of the two states: Pending or Running. This intimation can be done by a Kafka topic, that’s currently there in our metric collection system.
3. I need a Slurm cluster. I have to find suitable scripts and run them in order to generate data. This will take a long time.
4. In Point 2, I not only meant Slurm metrics but also the application metrics.
5. All the training data has to be stored in a NoSQL database.
6. For Slurm metrics, scontrol would be much more than sufficient.