Constantinos Skevofilax
https://github.com/tapis-project/smart-scheduling
constantinos.skevofilax@gmail.com
512-662-6067

TAPIS Smart-Scheduling Project

Abstract

This document will cover the basis of the TAPIS Smart-Scheduling Project, including the basic concept, efforts taken, thinking processes, analyses, and notes for future project work.

Understanding the Problem

Currently at the Texas Advanced Computing Center (TACC) the queue system for the various High-Performance Computer's (HPC) operates on a first-come-first-serve basis, where researchers code or "jobs" go into a HPC queue to have their job run. However as time goes on, the queue begins to fill up which leads to jobs waiting for times multiple factors beyond what they requested to use on the HPC system. To avoid this the smart-scheduling project was put in place, which would look to dynamically reallocate jobs that are experiencing long queue times to a virtual machine or VM to run in the effort to save time. As per how the one would decided if the job would be reallocated, this comes from the hypothetical basis that follows: "Little jobs wait longer times in the queue when they are behind jobs that request higher amounts of resources or big jobs". This hypothesis was built on the general feel and understanding the SLURM system used by TACC's HPC systems and how different types of jobs tend to execute in the HPC queue. If we can identify when these "little jobs" were in a large queue we could therefore request for said job to be reallocated to a VM, which would reduce the overall job time-to-solution time. Based off of this hypothesis, the smart-scheduling project developed scripts to conduct the analysis of the TACC HPC system's historical job data to determine if our hypothesis was true and if so create a method to reallocate a job based on criteria we would establish.

Understanding the TACC SLURM System

For someone to submit a job to run on a TACC HPC system, a user would need to input the following resource requests, which are: *max_minutes*, *reqcpus*, and *nnodes*. *Max_minutes* is the maximum amount of time a user can request to have their job run on the HPC for. For instance, on the Stampede 2 normal queue, the maximum amount of time a user can request is 2880 or 48 hours. *Req_cpus* is the requested amount of cpus a user a can request their job to utilize. It may be something to look into to see how to optimize resource usage. *Nnodes* is the requested amount of nodes a user a can request their job to utilize. It is important to note that some jobs don't end up using all the resources they request, including *max_minutes*, *reqcpus*, and *nnodes*.

As users can request different amounts of HPC resources for their jobs, it is important to label different sizes of jobs based on their resource requests. For example, and this is what will be used to explain different concepts in this document, a "little job" would be classified as so if a user requested their job to have a small amount of max_minutes, req_cpus, and nnodes. Small refers to a max_minutes value that would result in a low amount of time waiting for your job to finish running, such as requesting <= 4 hours in the HPC system. The 4 is an arbitrary number and can be adjusted. In addition, said job requested a relatively small amount of computing power which is reflected in a low req_cpus and nnodes values such as 10 and 1. These values are still arbitrary and can be adjusted by the research team. On the flip side, a "big job" would be classified as so if a user requested a high amount of max_minutes, req_cpus, and nnodes. High refers to a max_minute value that would result in very long amounts of time for your job to complete running, such as requesting >= 1500 minutes in the HPC system. The 1500 is an arbitrary number and can be adjusted. In addition, said job is requesting a relatively large amount of computing power which is reflected in high req_cpus and nnodes values. You would expect for the "little jobs" to run and complete sooner than the "big jobs" because of their low amount of resource requests. As the TACC SLURM system waits for resources for to be free up on the HPC system to match the amount of resources requested on the next-up job in the queue, you can see how if a "big job" was in the queue ahead of several "little jobs" that the little jobs would have to wait for longer times than their job would have run for because of the first-come-first serve basis. However, this understanding would need to be proven mathematically and as such efforts were put in place to do so.

Standardization of TACC HPC Historical Job Data

At the end of each day, each HPC system creates a .txt file which houses all the information for all the jobs that were in the queue that ran, failed, and are still running as well the amount of resources they requested. It also includes the jobid, job name, start, end, and submit times for each job which we used to calculate job run time, queue time, and overall time to solution. Each .txt file is named for the current date. As all the data points were printed to the .txt file, it needed to spliced and standardized into a database. As such, I created the HPCDataLoad.py script which is on the tapis-project/smart-scheduling Github repository link posted above. I created a MySQL Database which stores the historical job data in a subdatabase for each of TACC's HPC systems. The MySQL Database can be accessed by using the MySQL Workbench and using the log in information found in the LoginInfo.txt file that is in this folder. The script status is considered complete, however future updates could be used to handle timing issues with daylight savings and leap years, as the script time functions are based on the Central Time zone and need to handle leap year and daylight saving conditions.

Analysis of the Historical Job Data

Taking this historical job data, Richard Cardone create a new table in MySQL called "stampede2_jobq", which is a table that holds all the successful historical job data for Stampede 2's normal queue, which successful implying the completed jobs that ran on Stampede 2's normal queue. He created this table to provided a better analysis of the historical job data because after loading the data into the database, I attempted to analyze the data holistically by running SELECT statements for different max_minutes, req_cpus, and nnodes values to see if there was a tendency in the data that matched my hypothesis. However, because there are over 4 million records in the Stampede 2 normal queue alone, the issue that arose was that the data set is inconsistent in its standard deviations. Running base level statistical analyses on the dataset by SELECTing all jobs for some WHERE clause would return extremely high standard deviations for the statistical analyses we calculated, such as job queue_time, because there is too much variance in this holistic approach to the dataset. For example, you would expect that running the following SELECT statement:

```
SELECT COUNT(max_minutes) AS TotalJobs,
                    AVG(start-submit) AS queue_time,
                stddev_samp(start-submit) AS STDFORQueue_time
                    FROM HPC_Job_Database.stampede2
            WHERE max_minutes BETWEEN 0 AND 120 AND reqcpus BETWEEN 1 AND 12;
```

to return a standard deviation relatively close to the mean however it produces the following:

TotalJobs	queue_time	STDFORQueue_time
1749424	105760.9020	22836044.762287553

This is because there is not a single trend when it comes to jobs queue times and their requested resources because of how the HPC queue is allocated for a given day. If you were to look at each of the jobs in this SELECT statement, you would see a wide variance in the queue times jobs experienced. Some jobs waited for times ranging from 0 seconds all the way to 9392997679 minutes, combined with over a million records belonging to this WHERE clause with varying queue_times, you could understand how coming form this SELECT statement approach would return unclear results that relate to the original hypothesis. Because this was the case and we ran varying manual trials in MySQL for different WHERE clauses that included varying max_minutes, reqcpus, nnodes, start-submit times values for different logical cases for the data set, we created the Stampede2_jobsq table to better analyze the data.

The Stampede2_jobsq table is made up of the following columns: jobid, submit, start, max_minutes, queue_minutes, backlog_minutes, backlog_num_jobs. Queue_minutes is the calculated queue_time jobs experienced before starting, which is taken by subtracting a job's start time by their submit time. Backlog_minutes is sum of all the max_minutes requested for a queue. For example, if there were 6 jobs in queue with all requested max_minutes values being 60, the backlog_minutes would be 360. For the backlog_num_jobs is the number of jobs in backlog, so going back to the original example it would be 6. Applying these calculations to a large dataset that is the Stampede 2 normal queue dataset, we could use the new columns to look at the context of each queue. Because the broad approach of searching for low queue_time standard deviations was unsuccessful, being able to use the job classifications we discussed earlier with the job's max_minutes value and looking at the backlog, we could better evaluate if our hypothesis was correct. Based on the unsuccessful first approach, we adjusted the hypothesis would still apply but you would need to look into the context of the queue, for instance the number of backlog jobs and backlog minutes as well as the requested resources. This is because there is no

consent trend for a broad understanding of the dataset, you could better evaluate when a job would need to be reallocated by looking at the queue in real time. With this new table we could look at examples of queues and their trends and find trends that would call for reallocation.

Discussion of Trends and Our Method for Finding Them

To do this, I created NewAnalysis.py, which is a script that takes the stamped2_jobq table's values and calculates the means and standard deviations for the following SELECT statement:

```
current_where_clause = " WHERE max_minutes BETWEEN " + str(i) + " AND " + str(i + max_min_step) + \
                       " AND backlog_minutes BETWEEN " + str(j) + " AND " + str(j + backlog_min_step) + \
                       " AND backlog_num_jobs BETWEEN " + str(k) + " AND " + str(k + backlog_jobs_step) + \
                       " AND queue_minutes BETWEEN " + str(m) + " AND " + str(m + queue_min_step) + ";"
# SQL SELECT statement that combined with current_where_clause returns the means and the sample standard
deviations for the max_minutes, queue_minutes, backlog_minutes, and backlog_num_jobs cols
query = "SELECT COUNT(max_minutes) AS TotalJobs," \
        "AVG(max_minutes) AS AverageMaxMinutesRequestedPerJob," \
        "AVG(queue_minutes) AS AverageQueueMinutesEachJobExperienced," \
        "AVG(backlog_minutes) AS AverageBacklogMinutes," \
        "AVG(backlog_num_jobs) AS AverageNumberOfBacklogJobs," \
        "stddev_samp(max_minutes) AS STDFORAverageMaxMinutesRequestedPerJob," \
        "stddev_samp(queue_minutes) AS STDFORAverageQueueMinutesEachJobExperienced," \
        "stddev_samp(backlog_minutes) AS STDFORAverageBacklogMinutes," \
        "stddev_samp(backlog_num_jobs) AS STDFORAverageNumberOfBacklogJobs" \
        " FROM HPC_Job_Database.stampede2_jobq" + current_where_clause
```

Here, the values between the BETWEEN statements are iterable values for a step size that the user can select. The user can also modify the starting and ending values for each of the WHERE clauses. For choosing step sizes, we chose that the max_minutes should iterate by 60 minutes, IE per hour up till the max amount of time a user can request on the HPC, which would be 2880. As per the max values that were listed in the latest push to the repository, to save time iterating through all the values, as some of the max values of the stampede2_jobq were too high. For instance, the actual maximum number fir max_minutes is 10080, which is beyond max 2880 a user can request. However, there are 101 jobs that have a requested max_minutes time > 2880. To iterate over 8000 values at a step size of 60 would be an overall waste of time to review 101 jobs while there are over 4 million in the dataset.

In this directory there are 4 .txt files and 4 .log files. Each of the 4 trials names represent different things. Stampede2_Normal_queue_Bin_Sweep_backlog_jobs_120.txt represents the sweep where the max_minutes step size was 120 and backlog_num_jobs was the main emphasis on analyzing different "bins" or different scenarios for different conditions a job may experience in the backlog. Backlog_num_jobs step size was 10 jobs. It is important to note that you notice that the starting values are 1. This is because we don't analyze jobs that have no queue time and we don't care about the jobs that waited no time in queue because we can never improve from that.

Stampede2_Normal_queue_Bin_Sweep_backlog_min_120.txt is the sane as the above but instead the backlog_minutes is iterated by a step size of 500. I will briefly discuss the findings in each of the these two files. There are two more trial data points to be discussed but I will discuss them later.

For Stampede2_Normal_queue_Bin_Sweep_backlog_min_120.txt here were the following results:

1) There were 270 "hits" or positive queries that had queue minute standard deviations that were close to each other

Notes on the data:
2) There were 16,189 total jobs that belong to these queries

3) The max_minutes requested for these hits was between 2761 - 2881, with a mean of 2858.6

4) The backlog_minutes for these hits was between 311501 - 671501

5) The mean queue_minutes experienced per each job for these hits was 700.661 per query with a standard deviation of 86.298 for all queries in this subset

6) The average std_for_average_queue_minutes was 20.804 with a standard deviation of 5.7317 per query

7) Per query there was a mean 59.977 jobs affected with a standard deviation of 4.9247

To make this more understandable, there were 270 positive queries that were found. How we determined what a positive query was that we created std_percentage which is found below:

```
std_percentage = abs(std_for_average_queue_minutes / average_queue_minutes) * 100
```

To see if the queue times were close to each other for some WHERE clause, we took the standard deviation for a sample and divided it over the mean queue minutes. If the the percentage calculated was less than a certain percentage that we determined, which in the trials were 50%, we would consider it a good query to analyze. This is because the mean queue_minutes was accurate to a point that reflected that if we were to put more jobs in this WHERE clause case we would be able to accurately predict a jobs queue time within some degree of freedom, being the standard deviation for the queue time, which would be within 50% of the queue time they would actually experience.

Taking this in our case, analyzing the Stampede2_Normal_queue_Bin_Sweep_backlog_min_120.txt results, we can see that there were 16,189 jobs that fit our model. It reflects that for jobs requesting max_minutes between 2761-2880 and the backlog_minutes was between roughly 310,000-671,000 that their queue minutes would most likely be on average 700.6 minutes with a standard deviation of 20.804. If we were able to detect these conditions when a new job was submitted we could predict its queue time. This is a general understanding from a skyscraper view of the dataset however to truly get a feel for the data you need to read through the text file and understand the statistics in contrast to the context of the queue system. It is important to note that for this iterative trial it only produced results towards the maximum of the max_minutes value, IE. 2880, so there was not a real “responds” to the step sizes we used in this trial to cover the entirety of the dataset and the various conditions that occur in queue.

In contrast the Stampede2_Normal_queue_Bin_Sweep_backlog_jobs_120.txt produced more results. There were 1716 successful queries and 365923 jobs affected. There was a greater variety in positive queries over the entirety of the dataset, with low std_percentages for all the queries, ranging around 19%. Not much in depth analysis went into this .txt file but there is a high probability in this file the key to understanding what kind of jobs wait in queue the longest and what could be reallocated. Summaries of the dataset can be found at the bottom of each .txt file. The first 2 trial.logs belong to their respective .txt files we discussed here, IE trial1.log belongs to the first .txt file discussed and trial2.log belongs to the .txt file above. As per the other two, they belong to Stampede2_Normal_Queue_Bin_Sweep_all_60.txt and Stampede2_Normal_Queue_Bin_Sweep_all_120.txt respectively. These trials take the long because their step size covers the entirety of the dataset however at the time of writing this they did not complete, they exited early by my choice, they will need to be rerun.

Final Notes and Possible Starting Points

In reviewing the project and where it could go, a good starting point would be to analyze the Stampede2_Normal_queue_Bin_Sweep_backlog_jobs_120.txt file as it would be the best shot to analyze the dataset and prove the hypothesis true. Further analysis could be applied in the cases as in the last two files but they would need to run and their iteration time was calculated to be 66 days so adjustments could be needed to better optimize the code. Please thoroughly understand the problem statement, the code written in the repository, the database, and my notes on what we are looking for statistically to decided if something can be predicted, IE. if queue time can be predicted. From the first recommended file it appears that this is true.