Report for COMP4651 Course Project

Google Cluster Traces Analysis

https://github.com/xinyi329/GoogleClusterTracesAnalysis
LIU Xinyi 20445141 xinyi329
LU Xinsen 20445165 xinsenlu

I. INTRODUCTION

The last few years have witnessed an increasing popularity of cloud computing. However, it's hard to design and maintain a cost-effective cloud computing system, as Barroso and Hölzle state that the key challenges include "rapidly changing workloads", "building responsive large scale systems" and "energy proportionality of Non-CPU components" [1]. In face of these challenges, engineers and scientists must learn from the performance of a cloud computing system under real workloads. This could be important for the design of next-generation schedulers. Google has released a cluster-usage dataset over 29 days in 2011 [2]. We have studied the machine- and job-level patterns and behavior with Spark, to learn how the cluster resources are utilized.

II. MACHINE-LEVEL PATTERNS

i) Machine Population

Machines involved in this cluster traces dataset are characterized by different capacity groups. Each variable has been normalized to the range from 0 to 1. Please refer to Table 1 for details. It is shown that 93% of the machines traced have a normalized CPU capacity of 0.5. While 1% of the machines have a relatively low capacity (normalized CPU = 0.25, normalized Memory ≈ 0.25), a small number of machines with high capacities might have replaced some of the former ones. The machines in this cluster are relatively homogeneous, which would be easy to maintain. Meanwhile, their capacities are somehow diversified, which could satisfy different requirements of users.

Table 1: Number of Machines with Different Capacities

CPU\Mem	**≤ 0.12**	**≈ 0.25**	**≈ 0.5**	**≈ 0.75**	**≈ 1**	**Total**
= 0.25	0	126 (1%)	0	0	0	126 (1%)
= 0.5	60	3866 (31%)	6732 (54%)	1003 (8%)	5	11666 (93%)
= 1	0	0	3	0	795 (6%)	798 (6%)
Total	60	3992 (32%)	6735 (54%)	1003 (8%)	800 (6%)	**12590 (100%)**

ii) Daily Machine Events

Figure 1 shows the daily machine events, where 0 denotes "ADD", 1 for "REMOVE", and 2 for "UPDATE". About 2% of machines are removed on average each day. This may due to system upgrades, network down, etc. [2] There exist local minima every seven days for all event types, from which we can infer that machine events decrease during weekends. Plus, many machines are removed and then added back soon, but some exceptions do occur. From Figure 2, we observe that quite a few machines experienced two or more times of addition after removal. Some machines were moved for ten or more times. A small number of

machines even experienced updates for more than thirty times.

Among 25303 machine events recorded during the trace period, we verify 8860 instances of adding a machine back after removal. Figure 3 is the distribution of machine downtime, which is the time between a machine removal and its re-addition. While more than 60% of the downtimes are less than 25 minutes, the longest recorded is about 17406 minutes. Therefore, we suspect that most machine down events may be caused by system upgrades [2]. Other events that take a longer time may involve human maintenances. The results show that some temporal and/or spatial localities do exist [3].

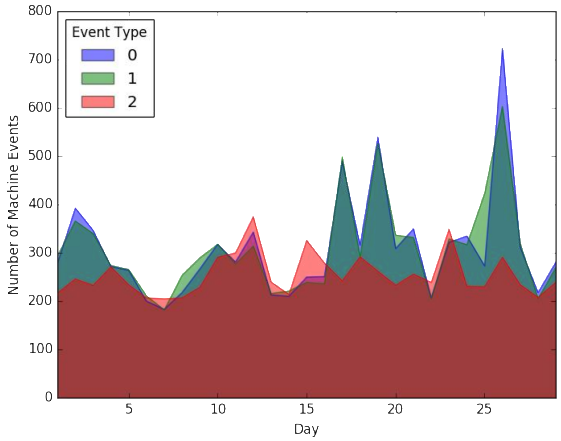

Figure 1: Number of Different Machine Events on Each Day of the 29-day Tracing Period

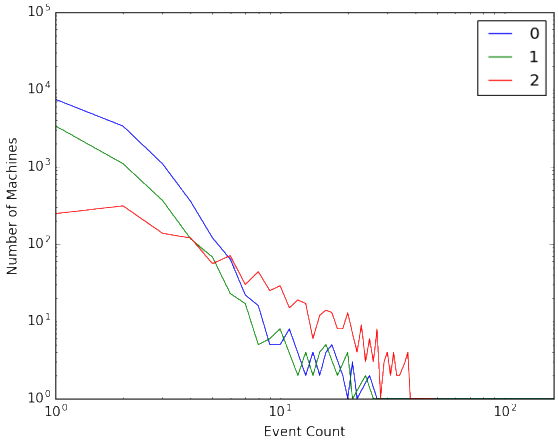

Figure 2: Machines Receiving the Same Event Count

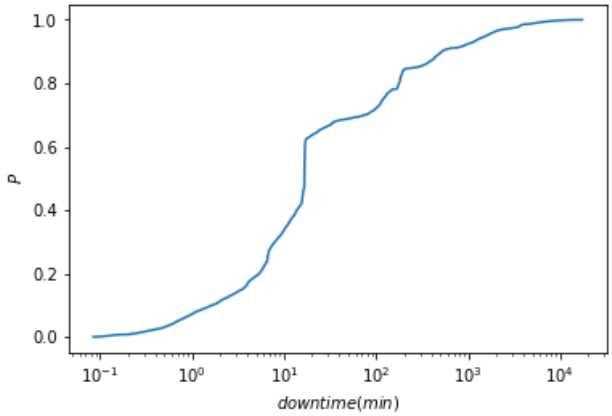

Figure 3: Machines' Downtime in CDF

III. JOB-LEVEL PATTERNS

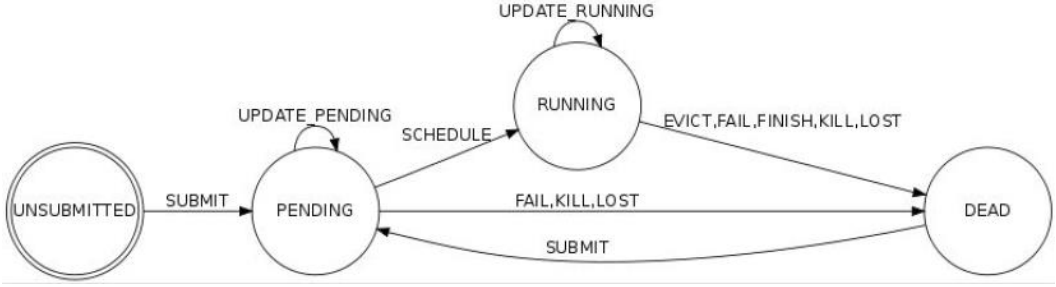

Figure 4: State Transitions for Jobs and Tasks [2]

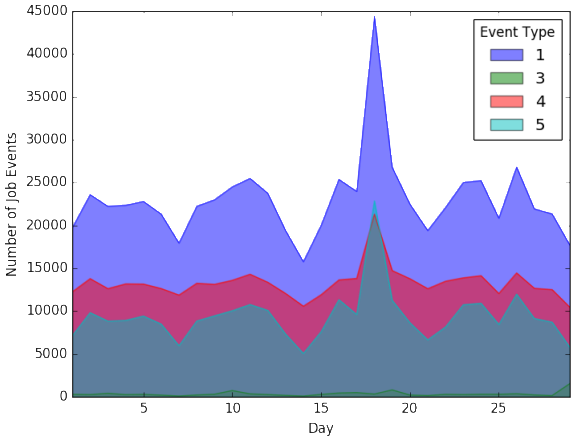

Figure 5: Number of Different Job Events on Each Day of the 29-day Tracing Period

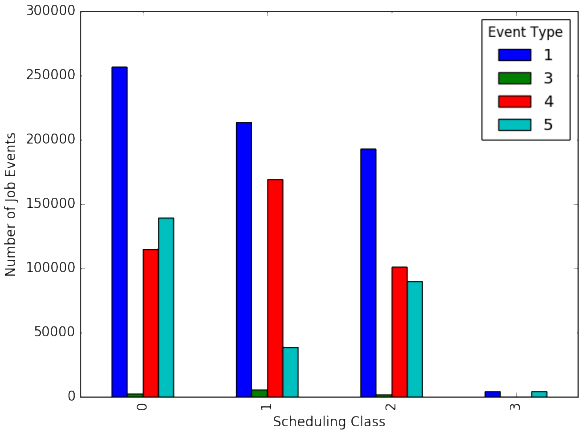

Figure 6: Job Events Counts per Scheduling Class

i) Job Events

Figure 4 shows that each job has four states and different state transition events. Among all these events, SCHEDULE (1) (following SUBMIT), FAIL (3), FINISH (4), and KILL (5) happen most frequently. Therefore, we will focus on these four types of job events. Figure 5 shows the occurrence of these job events mentioned above on each day. Again, we find that the trend of scheduled, failed and killed job number shows a strong periodicity, and the period is a week. We guess that fewer jobs are submitted on weekends. The probability that one job will be finished or killed is relatively stable. Jobs seldom fail, but are killed very frequently. However, this may also vary among different scheduling class. We find that about 50% of killed jobs are from class 0. We infer that part of those may be some test jobs submitted by Google engineers.

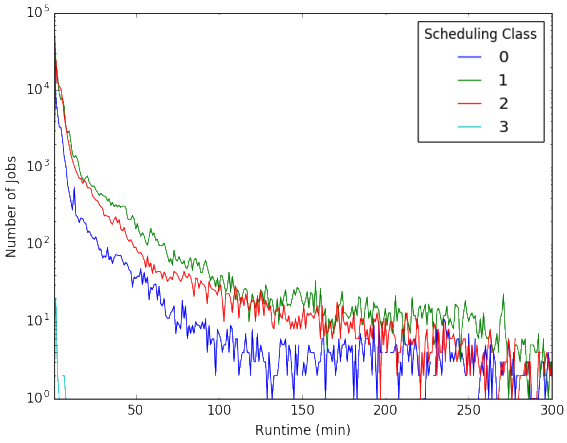

Figure 7: Job Size Distribution Based on Runtime

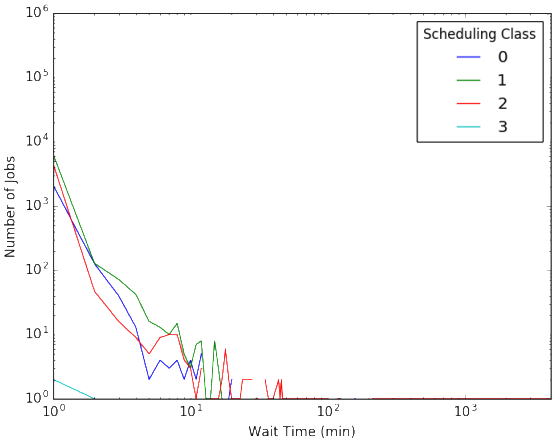

Figure 8: Job Size Distribution Based on Wait Time

ii) Scheduling Class and Execution Time

Figure 7 shows the job size distribution based on runtime (time from last SCHEDULE to last FINISH). Jobs belong to class 0, 1, and 2 show a similar behavior that most jobs are able to finish running in less than 20 minutes, while there are still some jobs executing for more

than 300 minutes. There are a small number of jobs in class 3 (latency-sensitive jobs) take less than 20 minutes to run.

Figure 8 represents the job size distribution based on their total waiting time (time from first SUBMIT to last FINISH). Similarly, most jobs belong to class 0, 1, and 2 wait for less than 10 minutes, but the tail is quite long. In comparison, the longest time that the jobs belong to class 3 is about 57 minutes.

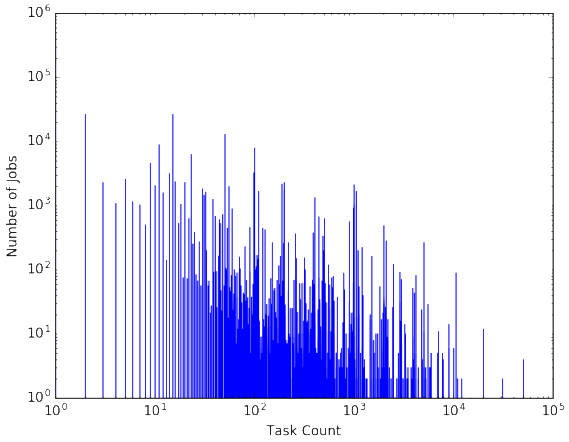

Figure 9: Job Counts by Task Count

iii) Jobs and Tasks

Figure 9 shows the size of jobs by the number of tasks. We can conclude that most jobs have less than 100 tasks, including more than 100000 jobs with only one task. However, the tail is still long. Some of the jobs have over 10000 tasks. Thus, we can derive that the majority jobs in the system are jobs with a few tasks rather than jobs with many tasks.

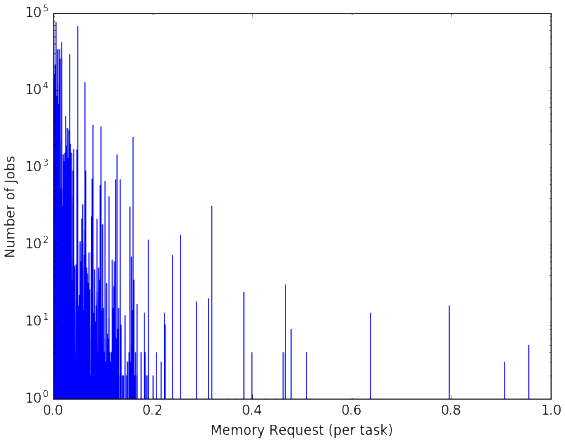

Figure 10: Job Counts by Memory Request Size

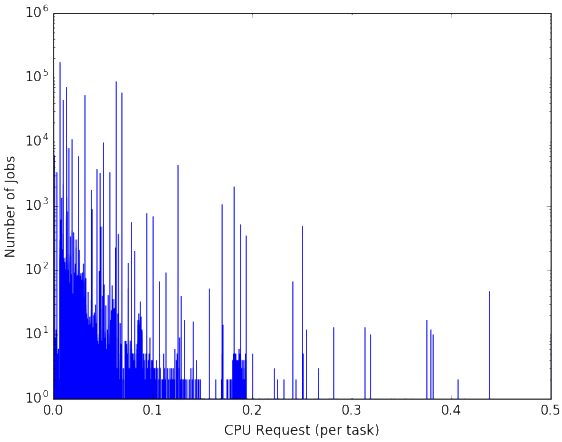

Figure 11: Job Counts by CPU Request Size

iv) Recourse Request and Usage

Figure 10 presents job counts by memory request, and Figure 11 by CPU. Here, memory and CPU requests are normalized to 1, corresponding to the machine attributions. We calculate the request for each job by averaging request values of every task, and round them to three digits. Most jobs require memory with a size less than 0.2, and CPU less than 0.15.

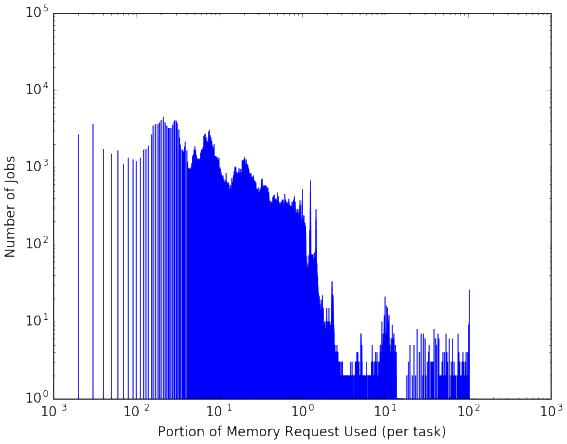

Figure 12: Job Counts by Portion of Memory Request Used

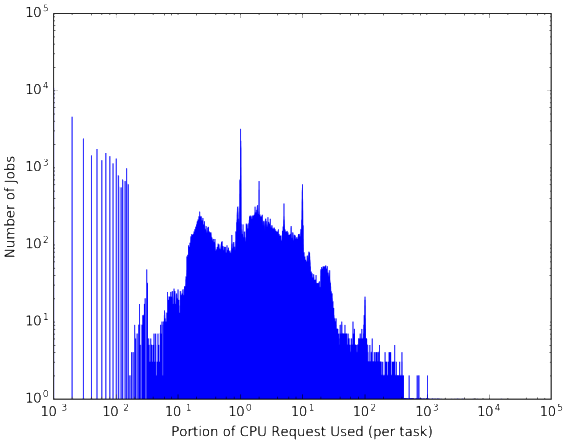

Figure 13: Job Counts by Portion of CPU Request Used

The difference between resources allocation and recourses usage do exist. It may partly because the requests for memory and CPU allocation input by users are not accurate enough. However, we are not able to rerun the program and see what the correct values are. We simply find the max usage of each task for each job, calculate the portion of memory/CPU request usage, and round the values to 3 digits. The results are presented in Figure 12 and Figure 13. It seems that most users may overestimate the usage of memory, while more may underestimate the usage of CPU. An algorithm that learns from past resources allocation and usage data and current input to predict a job's possible usage may be applied, to make the scheduling system more efficient.

IV. CONCLUSION

In this project, we study Google cluster traces data with Spark. We are able to get an insight at how Google clusters work. First, the machines in the clusters are continuously being updated. Upgrades also take place. However, the machines are fairly homogenous but provide different choices for different purpose of usage. Most job submitted are not latency sensitive, and some of the jobs probably are submitted for testing purpose, because more jobs are killed than complete fluently. Most jobs can be done in a short time, but the tail is quite long. The majority of jobs are jobs with a few tasks rather than jobs with many tasks. Most jobs require small sized of memory and CPU, but the gap between resources allocation and usage does exist. This project only discover a small part of the information that the dataset contains. When developing next-generation cluster management system, these features can be taken into consideration to improve the utility.

Reference

[1] L. A. Barroso and U. Hölzle, *The Data Center as a Computer*. Morgan & Claypool, 2009.
[2] Cluster workload traces, https://github.com/google/cluster-data.
[3] Liu Z and Cho S. "Characterizing machines and workloads on a Google cluster" in *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*, IEEE, 2012, pp. 397-403.