

Google Cluster Trace Analysis with Apache Spark

Kadir Korkmaz

19/01/2019

Contents

Abstract	2
Introduction	2
Trace Analysis of the Cluster	3
1. Machine Distribution	3
2. On average, how many tasks compose a jobs?	5
3. Job/Task that got killed or evicted	5
4. Eviction Probability of Tasks with Respect to Priority	6
5. The relation between the Priority of a Task and Machine Resources(CPU, Memory)	7
6. Tasks consume significantly less resource than what they requested	9
7. Machine availability	9
8. Machines which resources are largely underused	10
Performance Analyses of Apache Spark	11
1. Effect of Number of Worker Thread	11
2. Effect of JVM Memory Size	11
3. Effect of Lazy Evaluation	11
4. Effect of Cache (Persisting)	12
Conclusion	12

Abstract

In this work, I have analyzed the Google Cluster trace using Apache Spark. I have used the Data Frame API of Spark. I have conducted proposed analyses. Google Cluster trace contains data for 29 days. In the cluster, there are 12,583 machines. In the cluster, there is only 10 different kind of machine when we consider the CPU and memory. More than 90% of the machines have 0.5 CPU capacity. Only 6.32 percent of the machines have the highest CPU and memory capacity. Machines have three different kinds of hardware platforms. During the trace period, 672,004 Job is scheduled. Every job has one or more task. The total number of tasks is 25.4 million. The average number of task per job is 37.83. Because of eviction and killing scheduled task count 47.3 million. There is no linear relation between the priority of a task and probability of eviction but it is clear that some priority levels have a higher evicted and killed task percentage. There is no relation between the priority of a task and resources of the machines which task is scheduled. Tasks are almost uniformly distributed on machines. There are tasks which consume significantly fewer resources than what they requested. In the cluster, there is a maintenance or machine failure for every 279 seconds. There is a machine which never a task scheduled. There are 45 machines which have less than 10 tasks scheduled. There are 52 machines which have not finished any task.

The number of worker threads has a big effect on performance if there is enough input file to process for every worker thread. JVM memory size affecting performance in different ways. Lazy evaluation mechanism of Spark creates an opportunity to Spark optimize user requests to return the results in a shortest possible time. Caching has a big impact on performance. If it is possible to reuse an RDD, caching increasing performance linearly.

Introduction

In this work, I have analyzed Google cluster-usage traces using Apache Spark and I have created the document using R Studio.

You can access the subject of this work from this link. You can find the detailed documentation of traces from this link.

You can access the details of this work and materials(source code, intermediate results, CSV files) from this link.

Trace Analysis of the Cluster

1. Machine Distribution

In this section, I have shown the distribution of the machines according to CPU and memory capacity. Also, I have shown machine distribution according to platform type but this information is not significant because machine distribution according to platform type is same with according to CPU distribution.

Machine Distribution According to CPU and Memory Capacity

I have used machine events table to calculate machine distribution. In this table, we have CPU and memory information of the machines. I have ordered data in descending order according to time then I have grouped by according to machine id. I get the first CPU and memory information from every group. So this machine distribution shows the final state of the cluster. It is possible to get slightly different machine distribution according to order because the cluster is not something static. It is changing with time. Graph 1 shows the machine distribution according to CPU and memory. From this graph, it is easy to observe that more than 90% of the machines have 0.5 CPU and very few machines have 0.25 CPU.

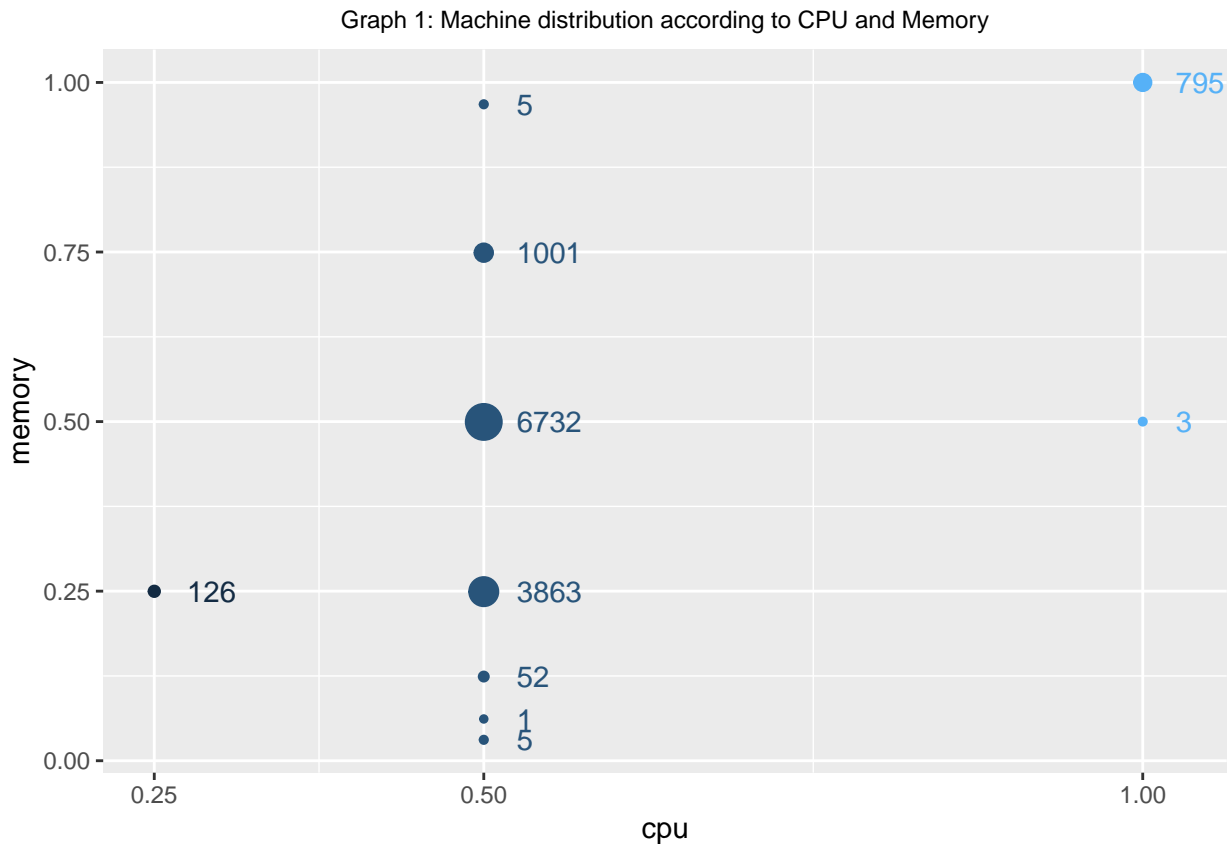


Table 1 show the same information with Graph 1 additionally it shows **capacity** metric and percentages of the machines in every capacity. I have created the capacity metric. Capacity is average value of CPU and Memory capacity($\text{Capacity} = (\text{CPU} + \text{Memory}) / 2$). This table shows the percentages of machines for every capacity. In the next sections, I will use this metric on plots.

Table 1: Machine distribution according to capacity

cpu	memory	capacity	number_of_machines	machine_percentage
1.00	1.00000	1.000000	795	6.32
1.00	0.50000	0.750000	3	0.02
0.50	0.96780	0.733900	5	0.04
0.50	0.74900	0.624500	1001	7.96
0.50	0.49950	0.499750	6732	53.50
0.50	0.24930	0.374650	3863	30.70
0.50	0.12410	0.312050	52	0.41
0.50	0.06158	0.280790	1	0.01
0.50	0.03085	0.265425	5	0.04
0.25	0.24980	0.249900	126	1.00

Machine Distribution According to Platform

Machine distribution according to the platform is not significant because it is the same with CPU distribution. On the documentation, it is stated that two machines with the same platform id can have very different CPU and memory resources. Table 2 shows the machine distribution according to the platform.

Table 2: Machine distribution according to platform id

platform_id	number_of_machines
HofLGzk1Or/8Ildj2+Lqv0UGGvY82Nloni8+J/Yy0RU=	11659
GtXakjpd0CD41brK7k/27s3Eby3RpJKy7taB9S8UQRA=	798
70ZOvysYGtB6j9MUHMPzA2Iy7GRzWeJTdX0YCLRKGvg=	126

2. On average, how many tasks compose a jobs?

I have used task events table to count the number of jobs and number of task per job. I have select distinct rows according to job_id and task_index then I grouped by according to job_id to count the number of task per job. Table 3 shows the total number of jobs, the total number of tasks and the average number of task per job.

Table 3: Job and task counts

number_of_jobs	number_of_tasks	avg_number_of_task_count
672,004	25,424,731	37.8342

Table 4 shows the statistics about the number of tasks per job. From these statistics we can conclude that a big part of the jobs contains only 1 task. An average number of task per job is **37.8342**. The job with the biggest number of the task has **90050** task.

Table 4: Summary statistics of the number of task per job

	statistics
Min.	1.0000
1st Qu.	1.0000
Median	1.0000
Mean	37.8342
3rd Qu.	1.0000
Max.	90050.0000

3. Job/Task that got killed or evicted

I have used job and task event tables to count the number of evicted and killed jobs and tasks. Table 5 shows the figures related to the job/task schedule, evicted and killed events.

Table 5: Job/task evicted killed count

type	scheduled_count	evicted_count	killed_count
Job	672,075	22	272,341
Task	47,351,173	5,864,353	103,496,805

4. Eviction Probability of Tasks with Respect to Priority

I have used task events table to calculate eviction percentages of tasks according to priority. I have counted the number of events according to priority and then I have calculated percentages of events on every priority. Table 6 shows the percentages of events according to priority. In this table, percentages are calculated according to task counts. *task_count* column shows the number of **unique** task on every priority level.

When we analyze the data on the table we can not see any linear relationship between the priority of task and eviction counts, however, some priority levels have higher eviction percentages. On the documentation it had been stated that there are some special priority ranges as “free”, “production” and “monitoring”.

Table 6: Task event percentages according to priority of tasks

priority	killed_percent	evicted_percent	failed_percent	lost_percent	task_count
11	25.6	0.0	0.0	0.0	7,538
10	668.8	35.9	1,526.9	0.1	1,403
9	79.6	5.9	457.6	0.0	286,269
8	54.2	1.2	6.7	0.0	254,680
7	199.0	0.0	0.0	0.0	400
6	21.2	0.1	8.4	0.0	639,784
5	0.0	0.0	0.0	0.0	104
4	23.7	0.3	4.5	0.0	14,197,733
3	6.0	14.9	0.1	0.0	1,027
2	14.8	4.3	2.4	0.0	1,111,810
1	37.6	18.8	29.4	0.0	2,453,482
0	83.2	81.7	170.5	0.1	6,472,128

5. The relation between the Priority of a Task and Machine Resources(CPU, Memory)

Is there any relation between the priority of tasks and machine resources which tasks are scheduled? I have used task events and machine events tables to answer this question. I have calculated distinct machine table from machine events and I have joined it with task events table than I have grouped by priority and *capacity* then I have count the number of tasks scheduled.

Graph 2 shows a heat map. In this heat map, on the y-axis, we have the priorities of the tasks and on the x-axis, we have the capacity of the machines which are tasked scheduled. On the cell's we have the percentages of tasks which are scheduled on that kind of machines. Table 7 shows the number of tasks according to the priority of a task. Also, Table 1 shows the percentages of the machines according to capacity. **When we consider all the information on the heat map and tables we can conclude that tasks are uniformly distributed on the machines according to priority. There is no relation between the priority of task and capacity of the machines.**

For example, In the cluster, 53.5% of the machines have the capacity of 0.49975(from Table 1). When we look at the values of the cell's on the heat map with capacity 0.49975 we can see that approximately 55% of the tasks on all categories are scheduled on this kind of machines. We can do the same observation for other capacities.

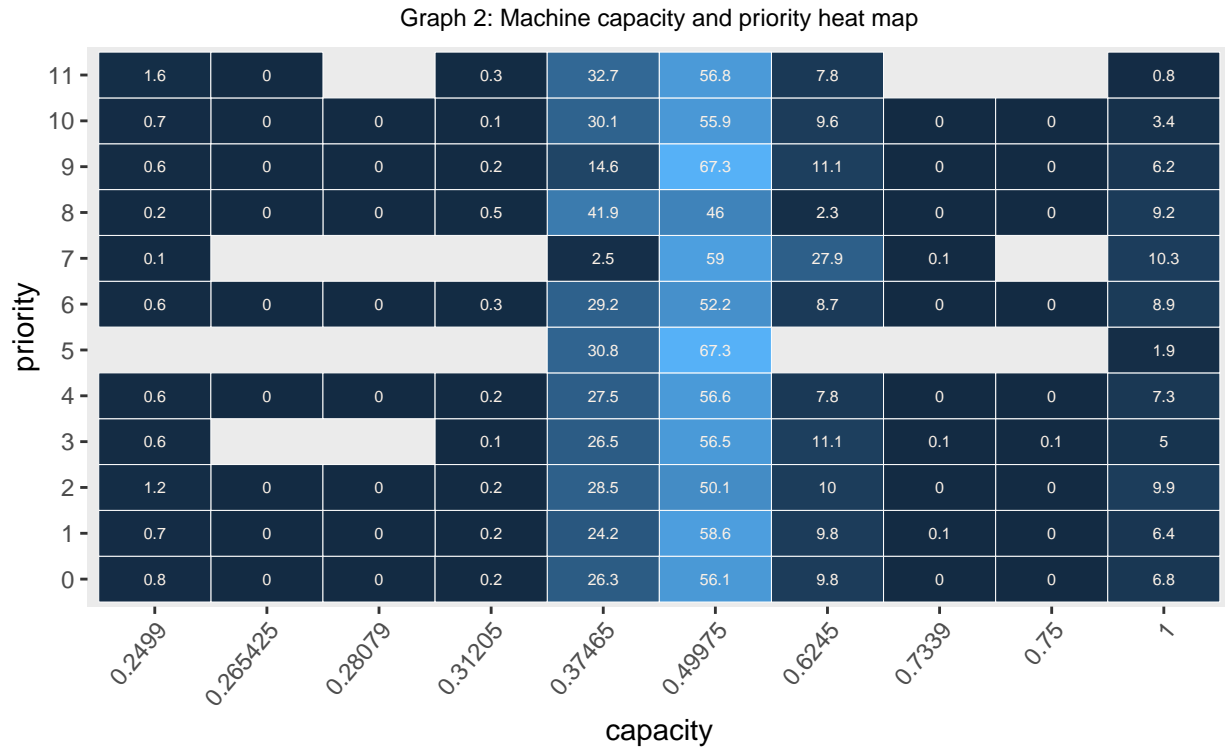


Table 7 shows the task distribution according to priority. In this table, task counts are not unique task counts because some tasks are rescheduled because of eviction or failure.

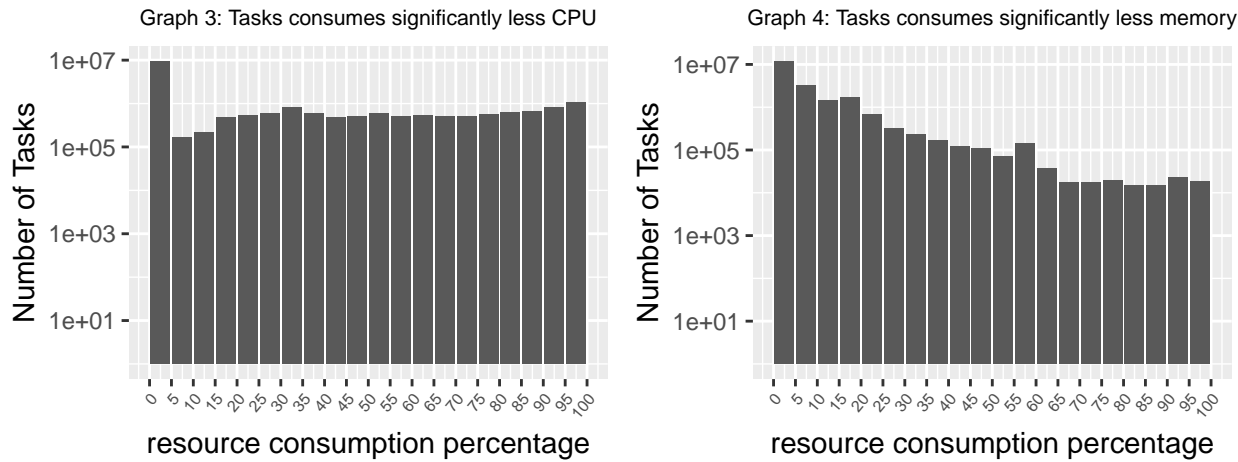
Table 7: Task counts according to priority of the task

priority	total_task_count
0	24,640,306
1	3,745,699
2	1,200,766
3	1,203
4	15,099,442
5	104
6	689,599
7	796
8	274,962
9	1,656,760
10	32,128
11	9,405

6. Tasks consume significantly less resource than what they requested

I have used a special method to decide whether or not a task is using significantly less resource from what it requested. To decide this, I have looked at the tasks' maximum CPU and memory resource consumption. If both of the maximum values never reaches what it requested. I assumed that this task is using significantly less resource from what it requested. I have used **task events** and **task usage** tables to find the specific tasks. I have joined this two table then I have grouped by according to job_id and task_index and I have looked at the max values of maximum CPU usage and maximum memory usage. Then I have decided whether or not a task using significantly less resource.

I have decided to create a histogram to have a better understanding of the resource consumption behavior of tasks which are using significantly less resources from what they requested. I have created two histograms. On the **X** axis I showed the resource consumption percentage according to what it requested on the **Y** I have shown the number of tasks. Graph 3 shows the CPU consumption behavior and Graph 4 shows the memory consumption behavior of the tasks which are using significantly fewer resources from what they requested.



From these two histograms, we can conclude that there is around 10 million scheduled task which is using less than 5 percent of CPU and memory from what they requested.

7. Machine availability

I have conducted some analysis to have an idea related to machine availability.

1- *What is the average time between two consecutive machines remove event.*

Average time between two consecutive remove event is 279.7 Seconds.

According to the documentation, the machine remove events can occur because of failures or maintenance so we can conclude that every 279.7 second there is a failure or maintenance in the cluster.

2- *Are there any machine which is removed from the cluster and did not add to the back.*

97 of the machines are removed from the cluster and they are not added back.

8. Machines which resources are largely underused

I have used task events table and task usage table to find machines which are resources largely underused.

When we check according to scheduled task counts: In the cluster, there is 1 machine which has not any task scheduled. In the cluster, there are 45 machines which have less than 10 tasks scheduled. There are 238 machines which have less than 100 tasks scheduled.

When we check according to finished task counts: In the cluster, there are 52 machines which have not any task finished. In the cluster, there are 98 machines which have less than 10 tasks finished. There are 336 machines which have less than 100 tasks finished.

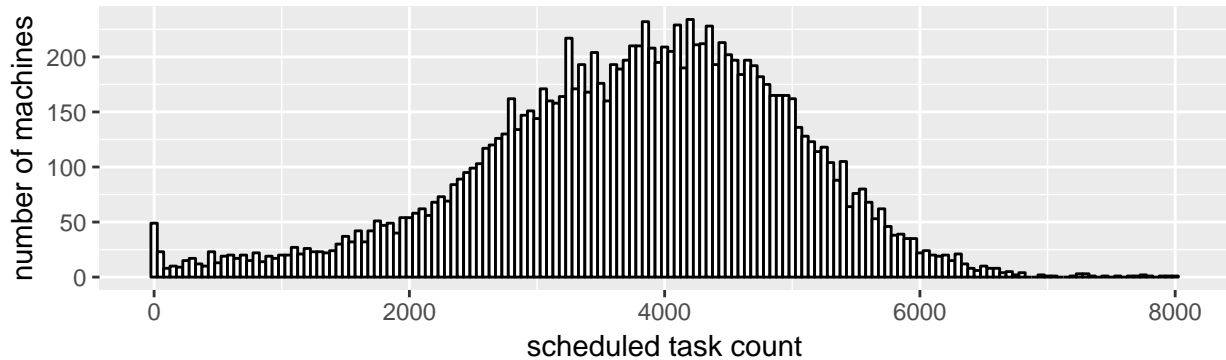
When we try to see the underused machines from task usage table we have a different story. When we grouped all the task usages according to the machine there are 36 machines which have not used. I have tried to measure a different metric which is machine utilization metric. For every task usage, I have calculated utilization as follows

$$\text{machine_utilization} = ((\text{task_end_time} - \text{task_start_time}) / 1000000) * \text{assigned_memory_usage} * \text{sampled_CPU_usage}$$

Then, I have grouped by task machine utilization parameters according to machine id and I have summed the machine utilization metric. According to this calculation, there are 36 machines which never utilized.

Following two histogram shows the task schedule counts per machine and task finished counts per machine. There is a machine which has not a task scheduled on and there are some machines which have not finished a task.

Graph 5: Scheduled task count per machine histogram



Graph 6: Finished task count per machine histogram

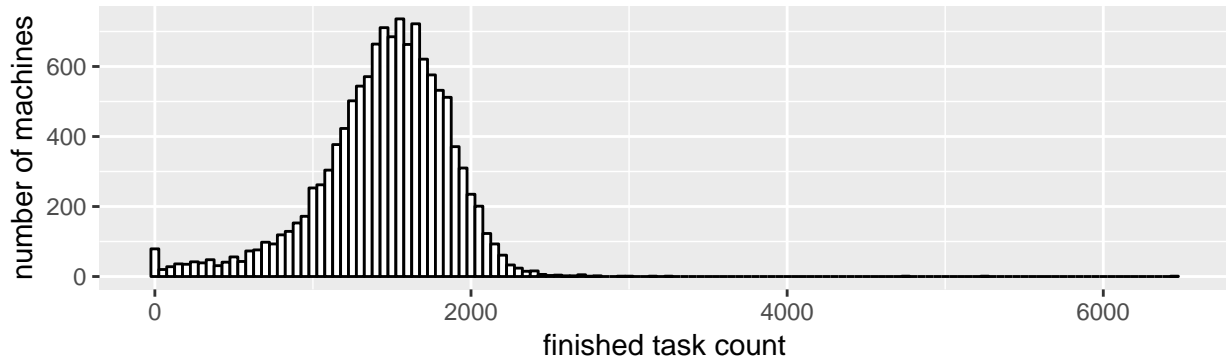


Table 8 shows the statistics related to the scheduled task per machine and finished task per machines.

Table 8: Machine scheduled and finished task statistics per machine

	scheduled_task_stats	finished_task_stats
Min.	0.000	0.000
1st Qu.	3023.000	1237.000
Median	3868.000	1496.000
Mean	3763.107	1447.824
3rd Qu.	4625.000	1725.000
Max.	8009.000	6456.000

Performance Analyses of Apache Spark

1. Effect of Number of Worker Thread

If there is enough input file to process than in this case the number of worker threads affecting performance. If there is only one input file to process than the number of worker threads has a small performance effect. This Effect is not linear. For example, to count all the distinct jobs in job tables with 8 worker thread takes 43 seconds. The same process takes 120 seconds with only one thread. For only one input file with a single worker thread the same job takes 15 seconds and with 8 worker thread, it takes 14 seconds.

2. Effect of JVM Memory Size

Normally in the spark cluster, there is two kind of process driver and executor. There is two different parameter to control the JVM memory size. These parameters are “spark.driver.memory” and “spark.executor.memory”. Normally this two-parameter have a different effect but in our case “spark.executor.memory” parameter is not useful because we are running Spark in the local mode which means that driver process spawning executor threads to do the processing so there is only one JVM which is driver JVM. Because of this reason, I can only assess the effect of “spark.driver.memory” parameter.

When we increase the memory size of the driver process, we can easily observe via using system tools(system monitor, top, htop) driver JVM is instantiated with specified memory and it is using the reserved memory. Also for caching, spark is using memory on default so if there is enough memory caching effect could be big in case of all the cached data small enough to fit in memory.

During my study, I have java.lang.OutOfMemoryError in case of collecting a huge amount of data at driver process. Arranging the JVM memory size could solve this problem.

3. Effect of Lazy Evaluation

Spark has two important notion: transformation and action. If there is no action than spark does not apply the transformations. This is called lazy evaluation. If there is action than spark applies transformations to RDD's and it produces results. This means that if you do not have any action in your spark code than spark will not do anything and it will return very fast. Lazy evaluation gives the opportunity to spark optimize user queries in terms of processed data. There is no way to force Spark to evaluate transformations eagerly.

4. Effect of Cache (Persisting)

Cache has a big impact on performance. In normal conditions spark is not caching RDD's. Assume that you are making a calculation and you are counting the number of elements in the RDD later you tried to save the RDD to a CSV file. Without caching Spark will calculate same RDD two times, once for counting once for writing to CSV file. With caching it will only calculate once and it will use the result of the same calculation for counting and writing to CSV file.

At the beginning of this study I was showing the result of the RDD's and later I was trying to do further processing without caching. In this case, it was taking two times more time to finish the job. Later, I realized the effect of caching and I tried to use caching when it is necessary.

Conclusion

In this worked, I have used Spark to analyze the trace of the Google cluster. I have used the Data Frame API. I have an opportunity to see the workload of a real cluster. I have evaluated the cluster load and machines of the cluster. I have evaluated the performance of the Spark with big data.