Survey on Google traces analysis concerning task per job and task per machine

Francesca Cecilia Schiavi

24/02/2016

1 Introduction

1.1 Goal

The goal of this document is to describe the analysis performed on Google traces, in particular the evaluation of tasks per jobs and task per machines, and the discussion of the results obtained from this analysis.

The description will be mainly textual with integration of graphical representations and portions of code.

1.2 Aims of the project

This document has been edited for all the people who will be involved in the process of analysis and integration of the traces.
Data obtained by our analysis, focused on the evaluation of the tasks per job and task for the machine (marked respectively tasks / job and tasks / machine), will be integrated with data from other analysis of the traces of the same time period and with traces that are not included in the temporal subset under evaluation.

1.3 Analysis conditions

In order to be able to integrate different data, conventions of analysis have been established:

- Temporal:
  the trace under eveluation has a duration of one day, 2nd May 2011. Regard the day a subset of traces has been identified with time-stamp
    start time = 18600000000 (May 2nd, 2011 - 00:00 EDT)
    stop time = 105000000000 (May 2nd, 2011 - 23:59 EDT)
- Input:
  <trace>, <condition>
  Condition, indicates the analysis' constraints
- Output:
  number of tasks per job and number of tasks per machine. The results are reported in a text file.
- Analysis:
  statistics on the number of tasks that can run concurrently in the same job and in the same machine.
  Various windows within the temporal subset have been used to identify particular trends.

1.4 Time interval considerations

The trace analyzed is one-day long, 2nd May 2011. It can be identified using timestamps

start time = 18600000000 (May 2nd, 2011 – 00:00 EDT)
stop time = 105000000000 (May 2nd, 2011 – 23:59 EDT)

1.5 Google traces description

The tracks are considerable datasets based on Google's cluster , a Google cluster is a set of machines connected by a high-bandwidth cluster network. A cell is a set of machines, that share a common cluster-management management system that allocates work to machines. Work arrives at a cell in the form of jobs. A job is comprised of one or more tasks, each of which is accompanied by a set of resource requirements used for scheduling the tasks onto machines. Each task represents a Linux program, possibly consisting of multiple processes, to be run on a single machine.

To each task, job and machine is assigned a unique ID:

- The taskID is generated using in JobID joined to an internal index. The ID can remain the same when the task is stopped and reallocated.
- The JobID generally does not remain the same when the job is stopped and reallocated.
- The machineID can remain the same when the machine is removed from the cluster and then reinserted.

2 Analysis

As specified in the preceding paragraph, the aim of our analysis is to analyze recurring statistics within tasks per job and tasks per machine.

To do this, we analyzed the *task event table*.

2.1 Task event table

The *task event table* contains the following fields:

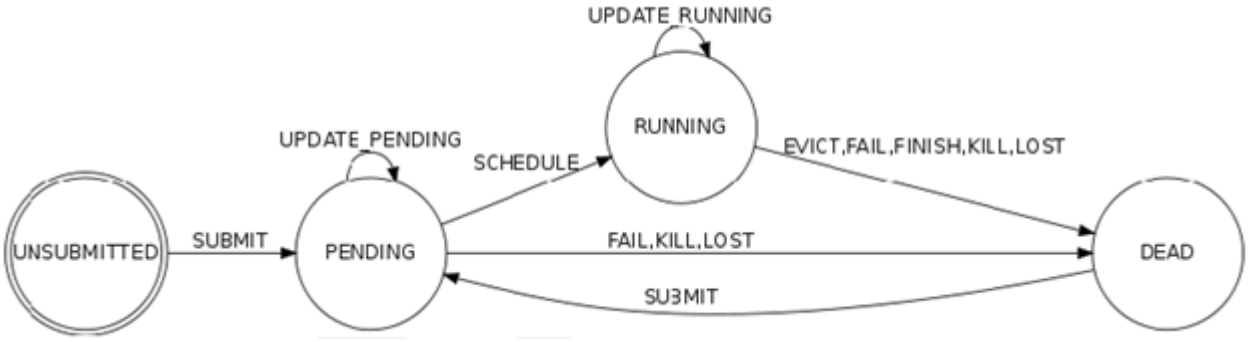
1. timestamp
2. missing info
3. *job ID*
4. *task index* - within the job
5. machine ID
6. event type
7. user name
8. scheduling class
9. priority
10. resource request for CPU cores
11. resource request for RAM
12. resource request for local disk space
13. different-machine constraint

The fields that we take into consideration for the analysis are:

- Timestamp, necessary to identify the daily subset and to identify the analysis window. [Microseconds]

- JobID, need to identify the tasks within the job
- Event type, numeric values that define the state of a task (or a job):
  - SUBMIT (0): A task or job became eligible for scheduling.
  - SCHEDULE (1): A task (or a job) was allocated on a machine. The fact that a task (or a job) has been allocated does not mean that it has already started running. For an event job it is defined schedule when his task is allocated on a machine.
  - EVICT (2): A task or job was descheduled because of a higher priority task or job, because the scheduler overcommitted and the actual demand exceeded the machine capacity, because the machine on which it was running became unusable, or because a disk holding the task's data was lost.
  - FAIL (3): A task (or a job) is deallocated because of a failure of the task.
  - FINISH (4): A task (or a job) has completed its cycle normally.
  - KILL (5): A task (or a job) is canceled because a task (or a job) from which was dependent is killed.

The table below shows the state diagram (Fig.1)



[Fig.1 - State diagram]

2.2 Task per job

2.2.1 Output file structure

startTime[micros],endTime[micros],mean,variance

- File " ".txt
- Internal structure:
  - Start time - the beginning of the evaluation period
  - End time - the end of the evaluation period
  - Mean - the average number of tasks per job in the time interval [end time - start time]
  - Variance - variance of tasks per job in the time interval [end time - start time]

Example

startTime[micros]	*endTime[micros]*	*mean*	*Variance*
18600000000	*22200000000*	*21.060357*	*15426.772763*

2.2.2 Analysis

First we analyzed the files.csv, storing them in an RDD according to the structure

[timestamp [micros], jobID, event_type]

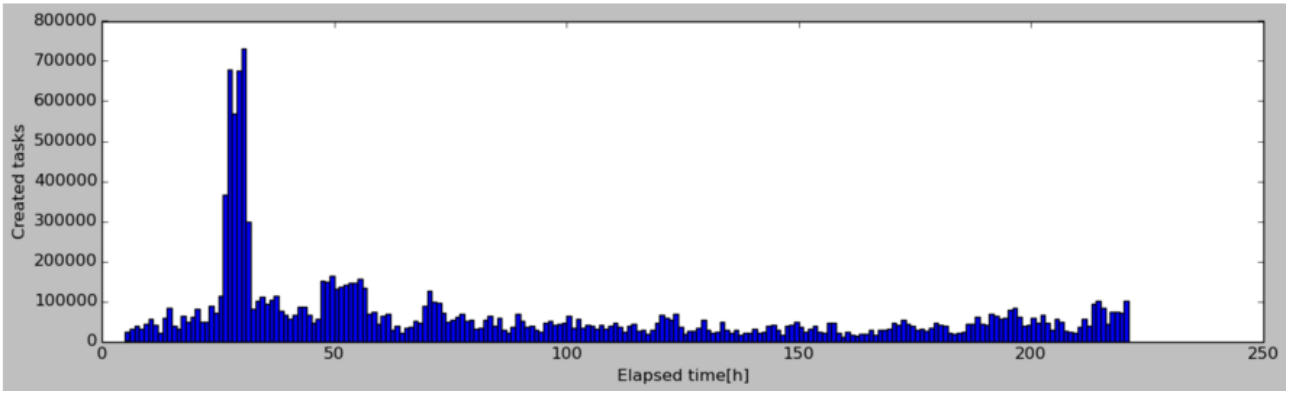
selecting from the *task event table* only the fields of our interest, filtering the traces according to the time interval indicated in the analysis conditions.

```
googleRDD=(sc.textFile(path)
        .map(lambda x:x.split(','))
        .map(lambda x:(int(x[0]),int(x[2]),int(x[5])))
        .filter(lambda x:x[0]>=startTimeMs and x[0]<=endTimeMs))
```

On this set of data we then carried out the evaluations.

2.2.2.1 Number of created tasks – Time interval of 10 days (2ND May – 10 May)

Although the target was the temporal analysis of 24 hours, 2nd May, we decided to show a histogram (Fig. 2.1) that would cover a 220-hour interval (10 days) to highlight an important aspect of the analysis, arosed during the generation phase of the chart of 24 hours.



***[Fig. 2.1* - Histogram. Created tasks - 10 days]**

As you can easily be seen from the graph, the slots:

- [25 – 32] h
- [48 – 52] h

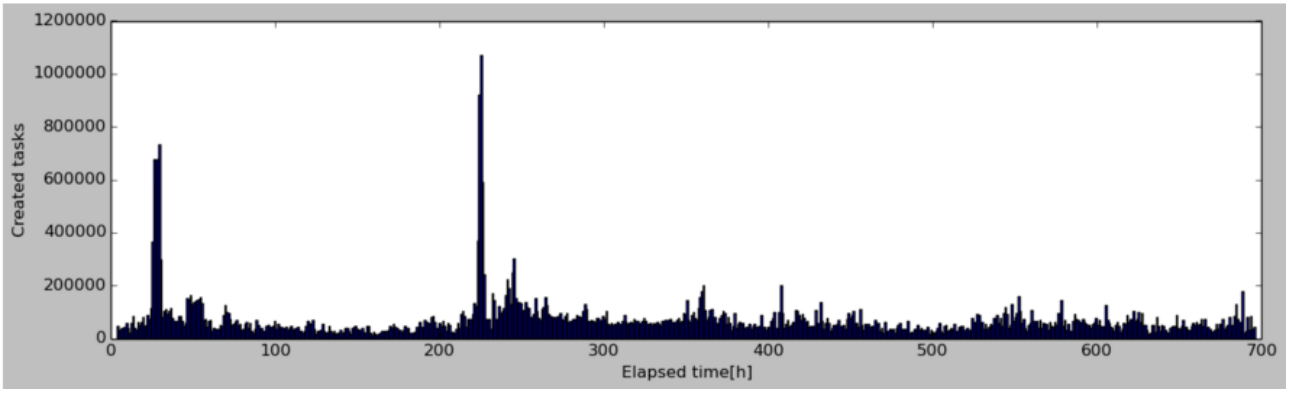
have an average number of tasks created in a hour which differs significantly from the average result per hour in the other time slots.

This anomalous behavior of the task allows us to identify clusters, clusters are groups of homogeneous elements with great statistical variation, which we will analyze separately from each other to ensure that the peaks do not influence other values.

The growth of generated task can be attributable to several reasons, we hypothesized some of these: raise of the number of tasks due to the initial setting of the working cluster, or increase the number of tasks at the beginning of each month. But not having a feedback on what type of task have been submitted and the lack of data on the previous or next month, we have no possibility to confirm our hypotheses.

In order to complete the analysis of the created task it is reported the range of 30 days.

The time interval not analyzed in the previous histogram (Fig.2.1), from the day May 11, identifies the presence of a single significant peak in the time interval [220-230] h, and minor peaks in the intervals [245-250] h and [360-362] h.



[Fig. 2.1.1 - Histogram. Created tasks – 30 days]

For all reported histograms we preferred to partition the time in hours, to get a clear but at the same time significant view of the phenomenon.

2.2.2.2 Number of created task for each job

A fundamental aspect in the analysis is to determine the number of tasks per job, giving the user the possibility to choose what type of event types want to analyze.

```
googleRDD=(sc.textFile(path)
        .map(lambda x:x.split(','))
        .map(lambda x:(int(x[0]),int(x[2]),int(x[5])))
        .filter(lambda x:x[0]>=startTimeMs and x[0]<=endTimeMs)
        .filter(lambda x:x[2]==event_type))
```

2.2.2.3 Event type probability

In order to analyze the probability on the types of events it was necessary to filter the RDD described above according to the *event_type* field.

Identifying the probability as $\frac{number\ of\ tasks\ with\ event_{type}=k}{number\ of\ total\ jobs}$.

- Probability that a task is allocated ($event_{type}$ = 3)
- The probability that a task should be evicted in the state ($event_{type}$ = 2)

- The probability that a task is killed ($event_{type}$ = 5)
- The probability that a task can finish the cycle ($event_{type}$ = 4)

2.2.2.4 Mean and variance

Define a new RDD that represents a subset of google RDD defined previously, identifying only submitted tasks

```
dataRDD=googleRDD.filter(lambda x:x[2]==0)
```

because the goal is to find what is the mean and variance of tasks that are actually ready to be allocated.

In the following graphs are presented through a broken line the mean and variance of the number of generated tasks in the time interval, we decided to split this time interval in hours to avoid losing their significance and make the graph readable.

2.2.2.4.1 Mean

To find the average of created task per job we applied the following formula

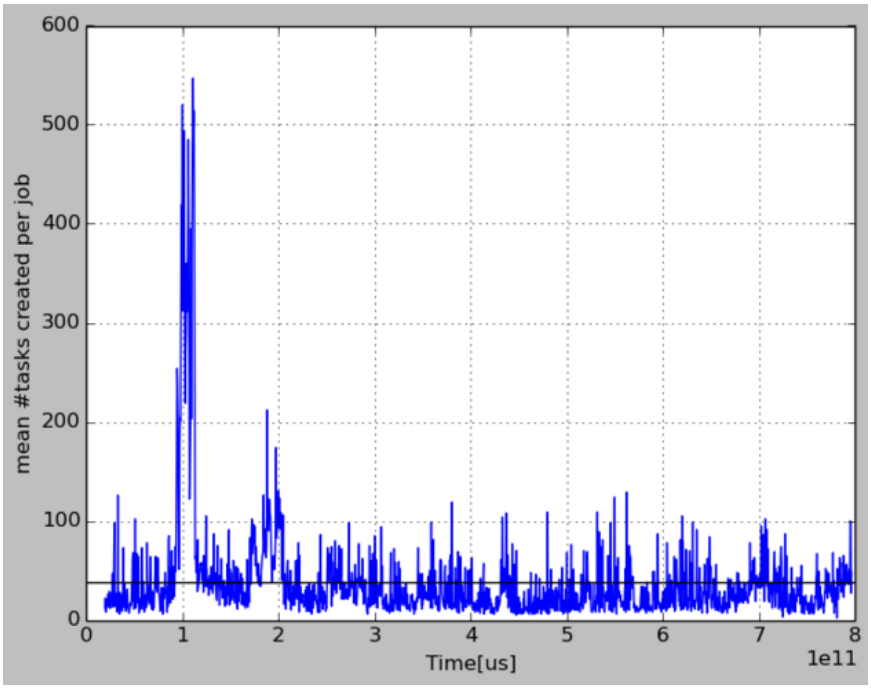
$$\frac{\sum_{i=0}^{i=num\ job} number\ of\ generated\ tasks_i}{number\ of\ created\ jobs}$$

```
def meanTaskPerJobDict(jobDict):
    if(len(jobDict)==0):
    return 0
    values=jobDict.values()
    s=sum(values)
    return float(s/len(jobDict))
```

Where jobDict is a python dictionary as: {job ID 1: tasks generated for job1, jobID2: tasks generated for job2, ..., JobIDn: tasks generated for jobn}.

The graphs reported below show the relationship between the number of tasks submitted to the job in the time interval (blue graph) and the total average (black line). The number of submitted tasks per job is graphically represented with a broken line, obtained by averaging the task / job granularity indicated in the range of time in analysis.

We report (Fig. 2.2) the average of the number of tasks generated per job in a week to check, even in the case of the average, the presence of clusters (see histogram Fig. 2.1).



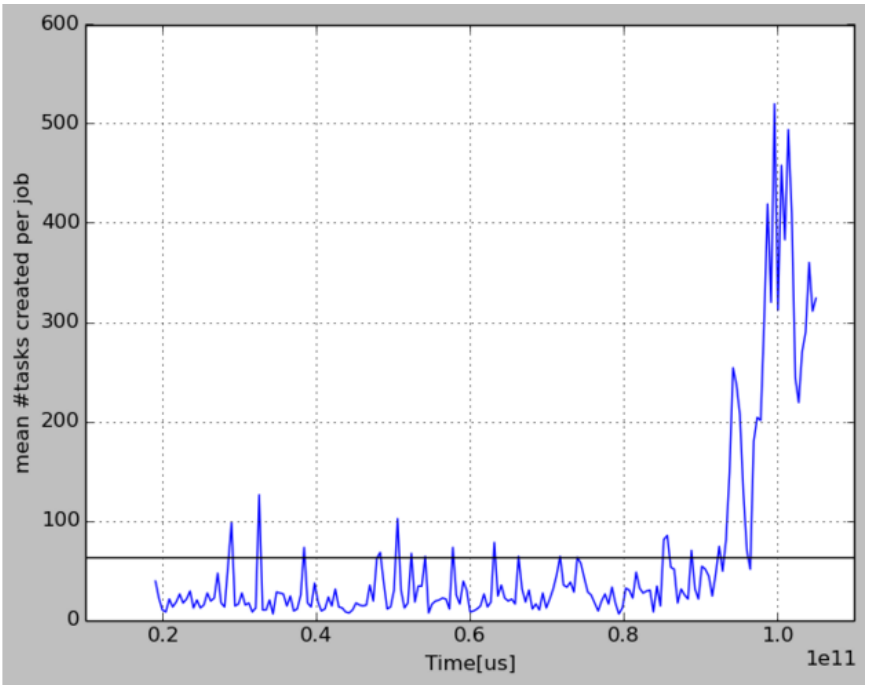
[Fig. 2.2 - Mean number of tasks created in a week]

In this case the average (black line) is consistent with the average value of the broken line (blue line) because the presence of the peak is damped by the almost constant values of the other time intervals.

Consistently with the conditions of analysis, have been created detailed graphs of the mean of tasks per job.

2.2.2.4.1.1 *Time interval of 24 hours –granularity is 8 points per hour*

As well as for the histogram (Fig. 2.1), the following graph makes clear the presence of a peak in the time interval [h 25-32].

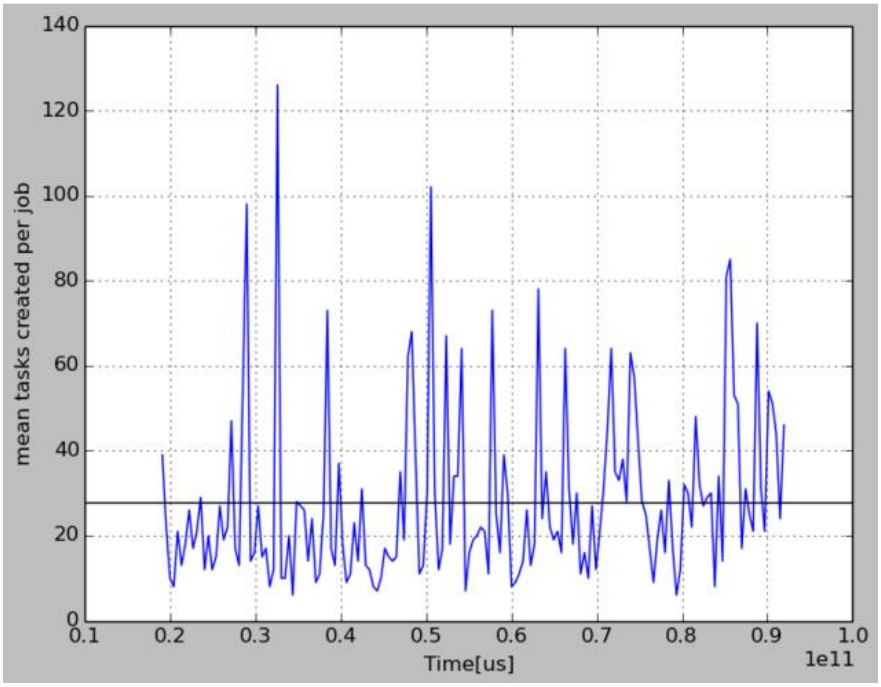


[Fig. 2.3 - Mean number of tasks created in 24 hours (granularity 8 pt)]

As can be seen from the graph, the presence of peaks, in the evolution of the average in the number of task / job, increases the total average value (black line). For most of the time in fact, the trend of the average (blue line) is below the total average value (less than 100 tasks).

The 24-hour chart (Fig. 2.3) is then divided into the graphs of the mean for each cluster (Fig. 2.3.1 - 2.3.2). The separation into subgroups was necessary because of misbehaving at the peak, which increases the overall average value.

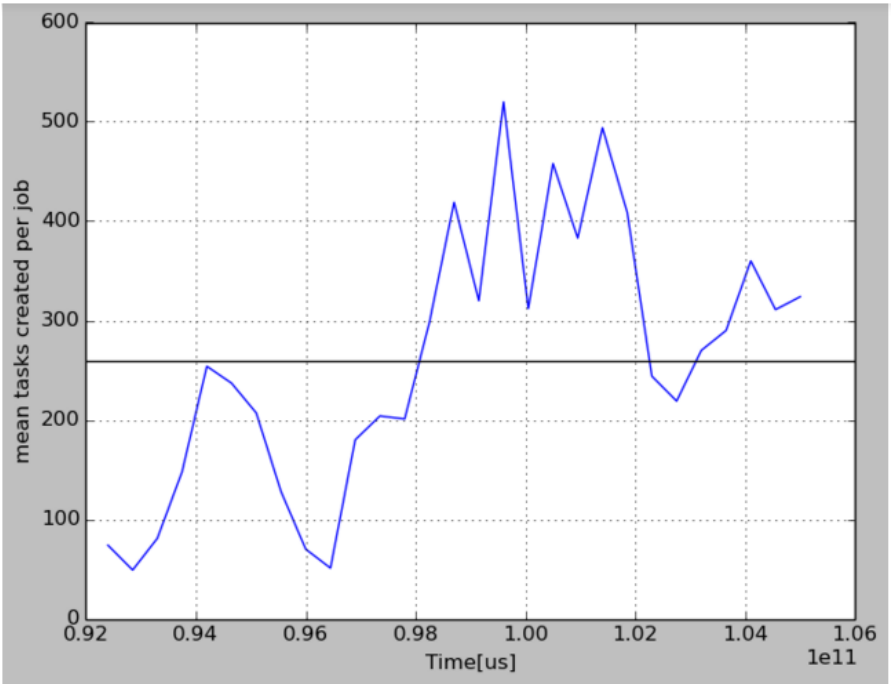
2.2.2.4.1.1.1 Cluster 1: mean number of task created per job (8 points per hour)



[Fig. 2.3.1 - Mean number of tasks created in 24 hours (granularity 8 pt)]

The first cluster identified (Cluster 1) shows the trend during the interval [18.600 s – 92.625 s]. As can be easily seen from the graph the total average value is now consistent with the trend of the blue broken line; which highlighted that the presence of the peak increased significantly the total average (about 80 average tasks - about 30 average tasks).

2.2.2.4.1.1.2 Cluster 2: mean number of task created per job (8 points per hour)



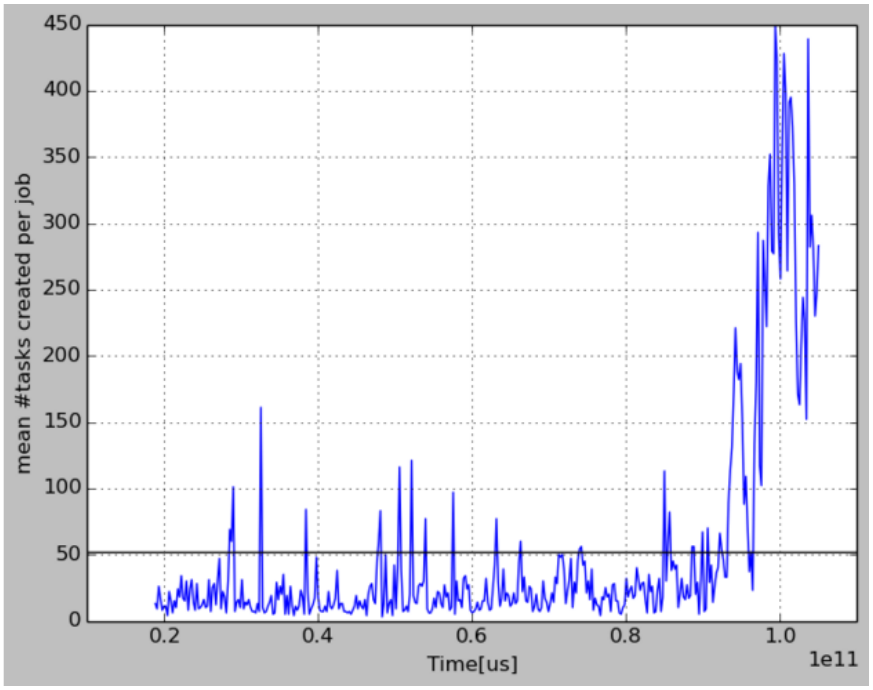
[Fig. 2.3.2 - Mean number of tasks created in 24 hours (granularity 8 pt)]

Analyzing the performance of the second cluster (Cluster 2) for the time interval [92.625 s – 105.000 s], is obvious that the value of the total average (about 250 tasks) justifies the increase in the average in the graph (Fig . 2.3) without subdivision into clusters.

In the following charts the granularity has been decreased for the interpolation of midpoints. The decrease of the granularity of the sub intervals makes more evident the trend of the mean of tasks / job.

The remarks made in the paragraph on the variability of the total number of tasks created also apply to the following graphs.

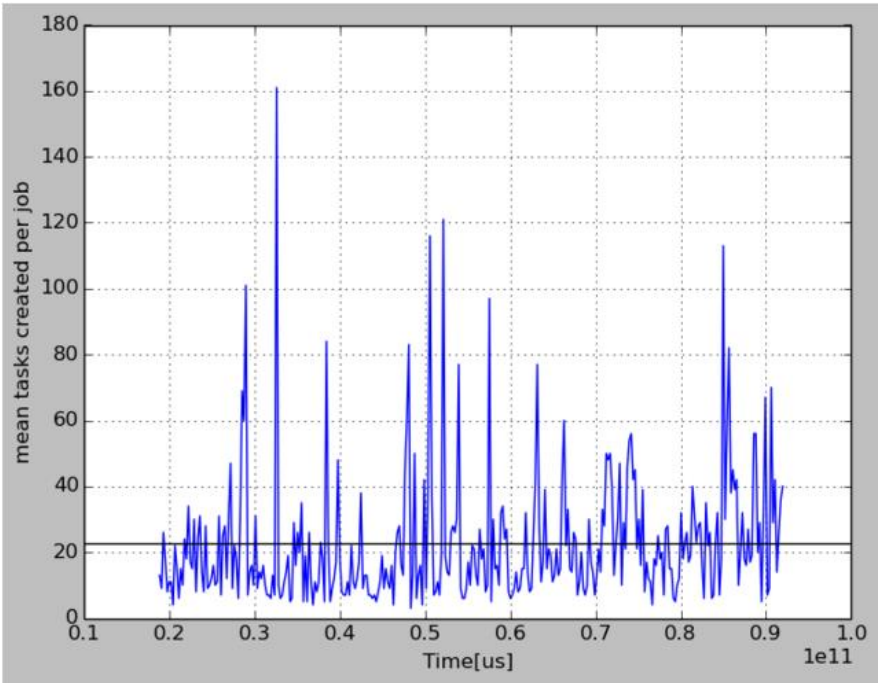
2.2.2.4.1.2 *Time interval of 24 hours –granularity is 16 points per hour*



[Fig. 2.4 - Average number of tasks created in 24 hours (granularity 16 pt)]

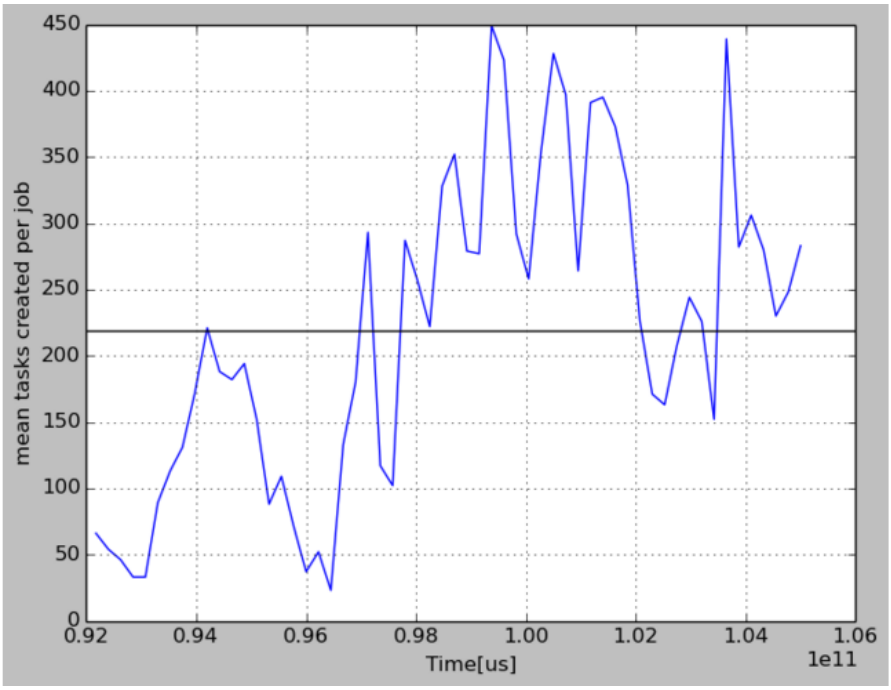
Coherently with the assumptions made in the previous paragraph we highlight (Fig. 2.4) as the peak between the second and the third day significantly increases the total average value (black line). Even in this case the study is divided in Cluster 1 and Cluster 2 (Fig. 2.4.1 - 2.4.2) in order to show better the relationship between the total average value and mean value with granularity.

2.2.2.4.1.2.1 Cluster 1: mean number of task created per job (16 points per hour)



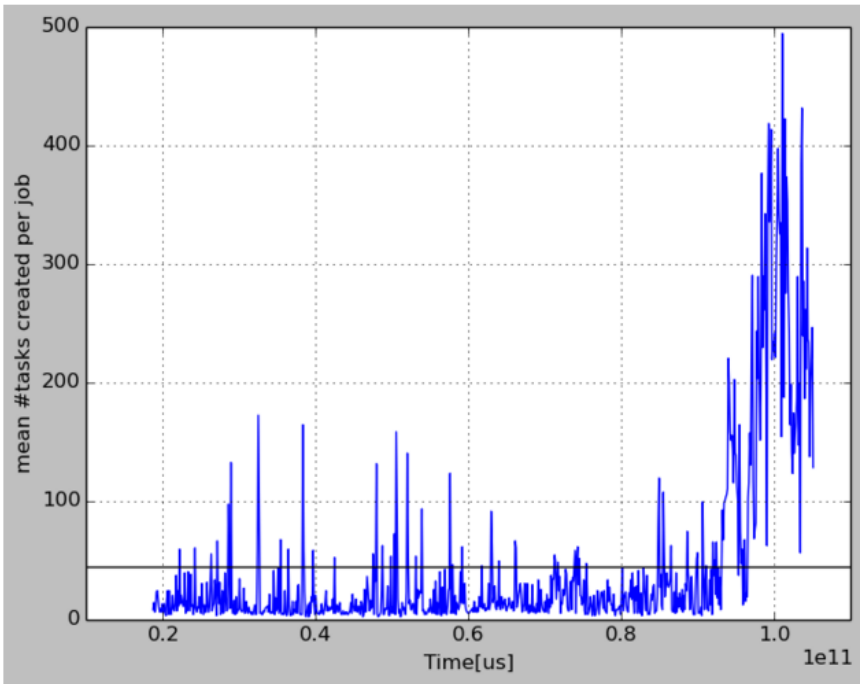
[Fig. 2.4.1 - Average number of tasks created in 24 hours (granularity 16 pt)]

2.2.2.4.1.2.2 Cluster 2: mean number of task created per job (8 points per hour)



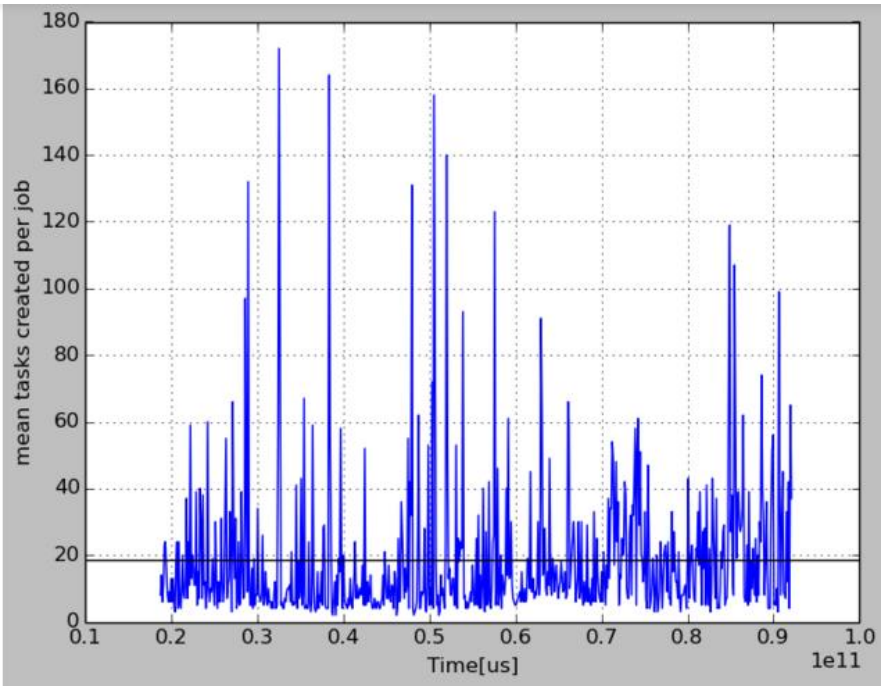
[Fig. 2.4.2 - Average number of tasks created in 24 hours (granularity 16 pt)]

2.2.2.4.1.3 *Time interval of 24 hours –granularity is 32 points per hour*



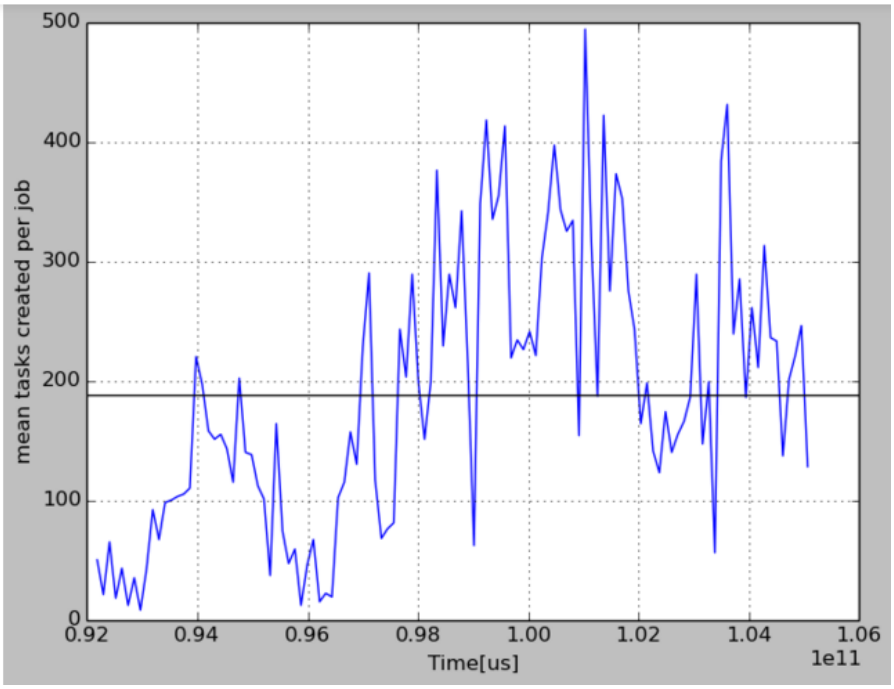
[Fig. 2.5 - Average number of tasks created in 24 hours (granularity 32 pt)]

2.2.2.4.1.3.1 Cluster 1: mean number of task created per job (32 points per hour)



[Fig. 2.5.1 - Average number of tasks created in 24 hours (granularity 32 pt)]

2.2.2.4.1.3.2 Cluster 2: mean number of task created per job (32 points per hour)



[Fig. 2.5.2 - Average number of tasks created in 24 hours (granularity 32 pt)]

2.2.2.4.2 Standard deviation

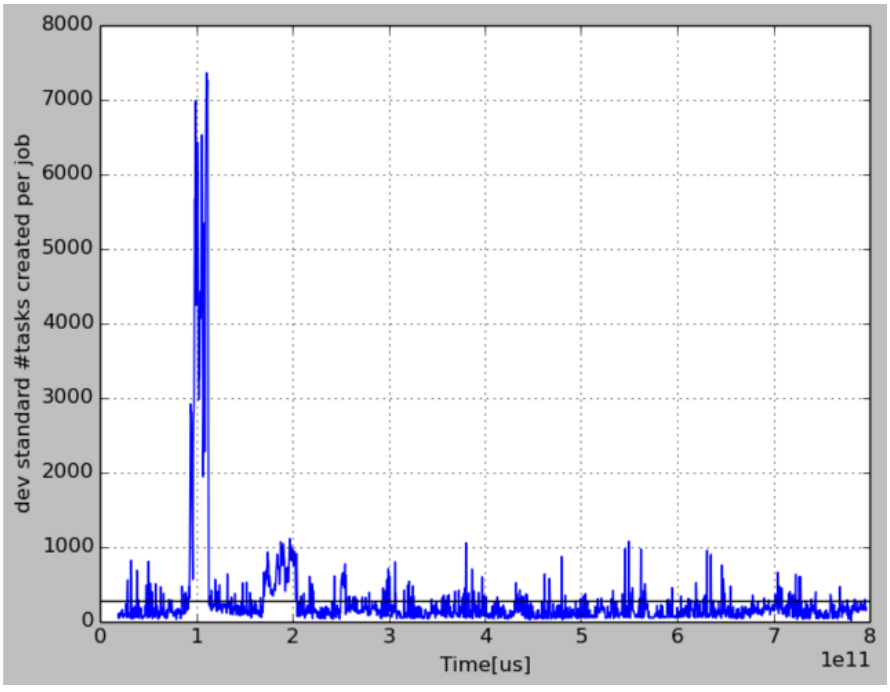
The standard deviation is defined as $\sqrt[2]{\frac{\sum(value - mean)^2}{tot\ values}}$

```
def varianceTaskPerJobDict(jobDict,mean):
    if(len(jobDict)==0):
        return 0
    squareSum=0
    for v in jobDict.values():
        squareSum+=pow(v-mean,2)
    return float(squareSum/len(jobDict))
```

Where jobDict is a python dictionary as: {job ID 1: task generated for job1, jobID2: task generated for job2, ..., JobIDn: task generated for jobn}.

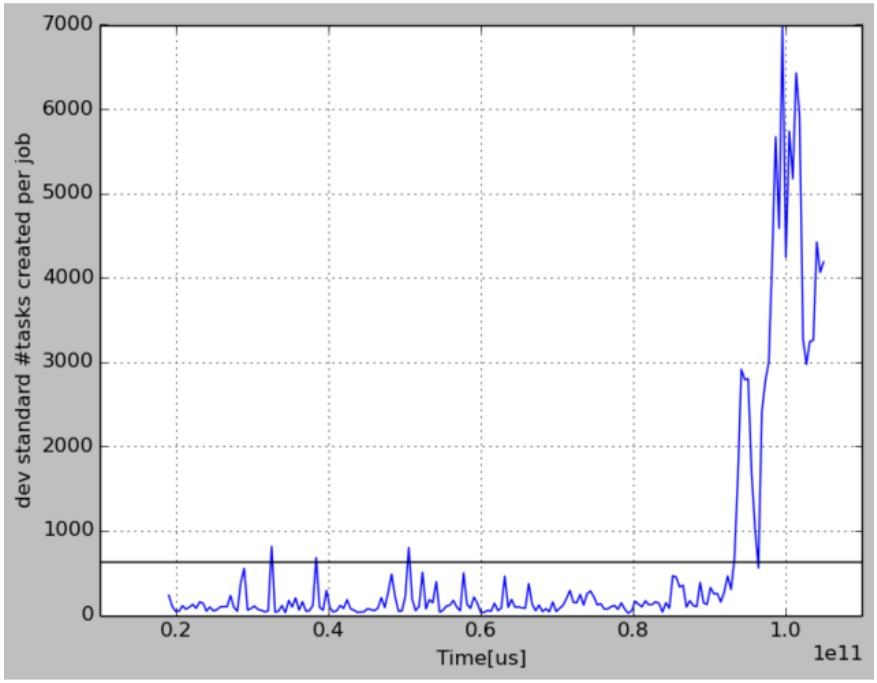
As in the case of the mean, it was considered appropriate to analyze the standard deviation initially in a time interval of one week (Fig. 2.6) and then for 24 hours (Fig. 2.7 and following). In the latter case, the graphics has been divided according to clusters.

Even for these analysis, the average standard deviation on the complete period is increased significantly by the value of the standard deviation of the Cluster 2. While in the time interval of the whole week the increase is less significant.



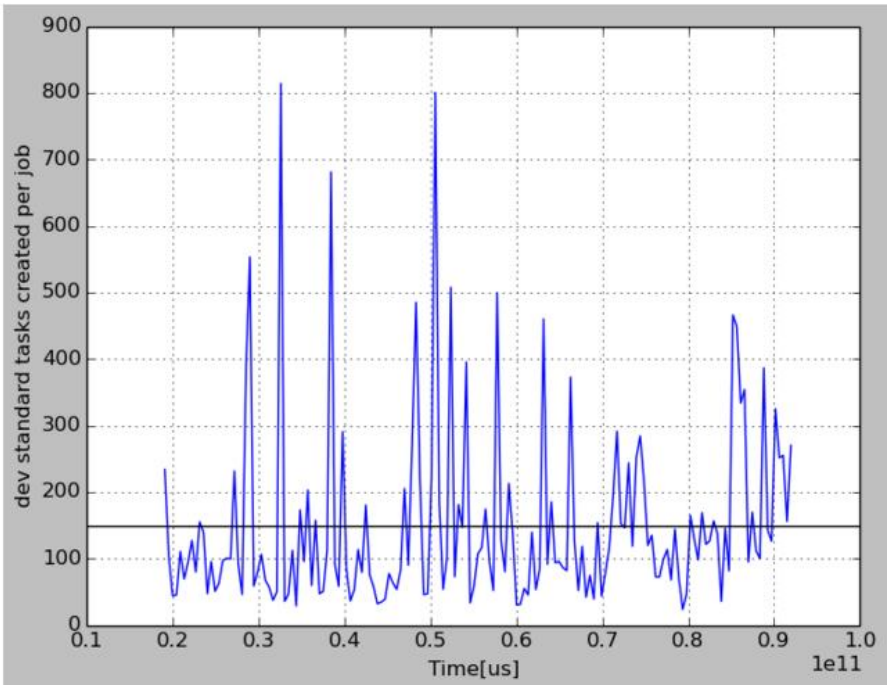
[Fig. 2.6 – Standard deviation of the number of tasks created in ten days]

2.2.2.4.2.1 *Time interval of 24 hours –granularity is 8 points per hour*



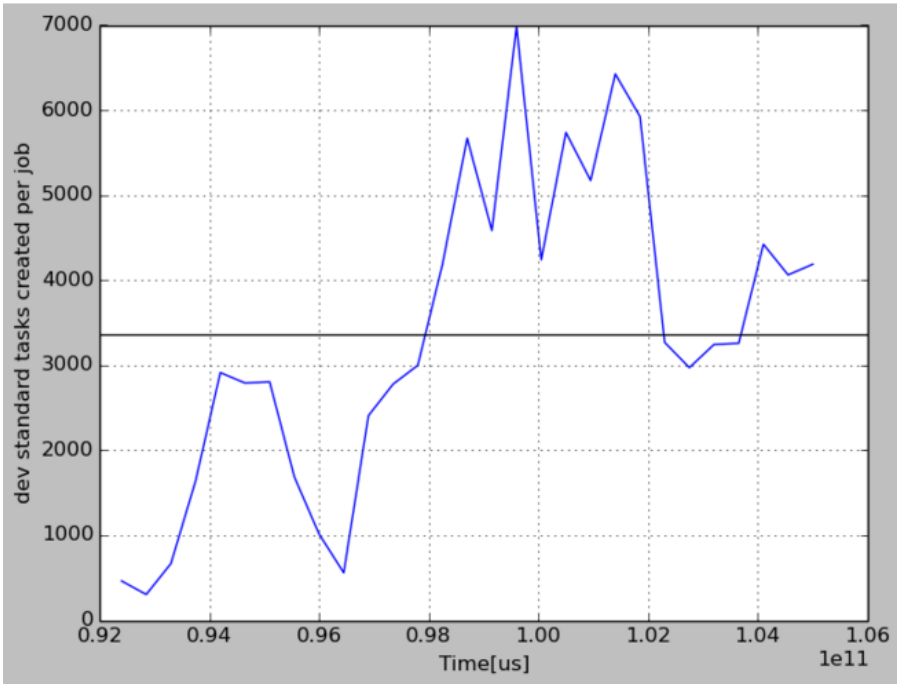
[Fig. 2.7 – Standard deviation of the number of tasks created in 24 hours (granularity 8 pt)]

2.2.2.4.2.1.1 Cluster 1: standard deviation of task created per job (8 points per hour)



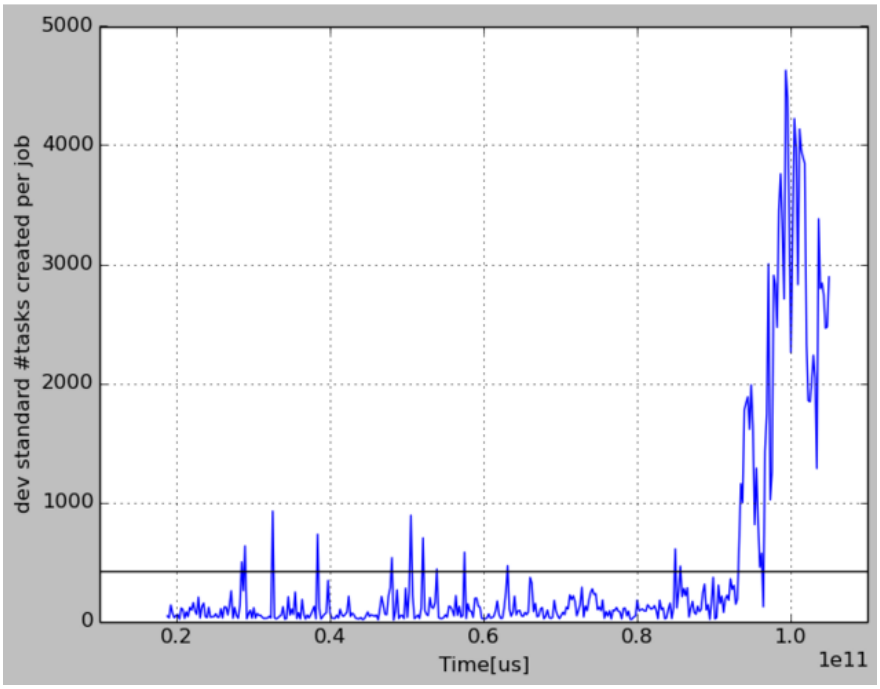
[Fig. 2.7.1 – Standard deviation of the number of tasks created in 24 hours (granularity 8 pt)]

2.2.2.4.2.1.2 Cluster 2: standard deviation of task created per job (8 points per hour)



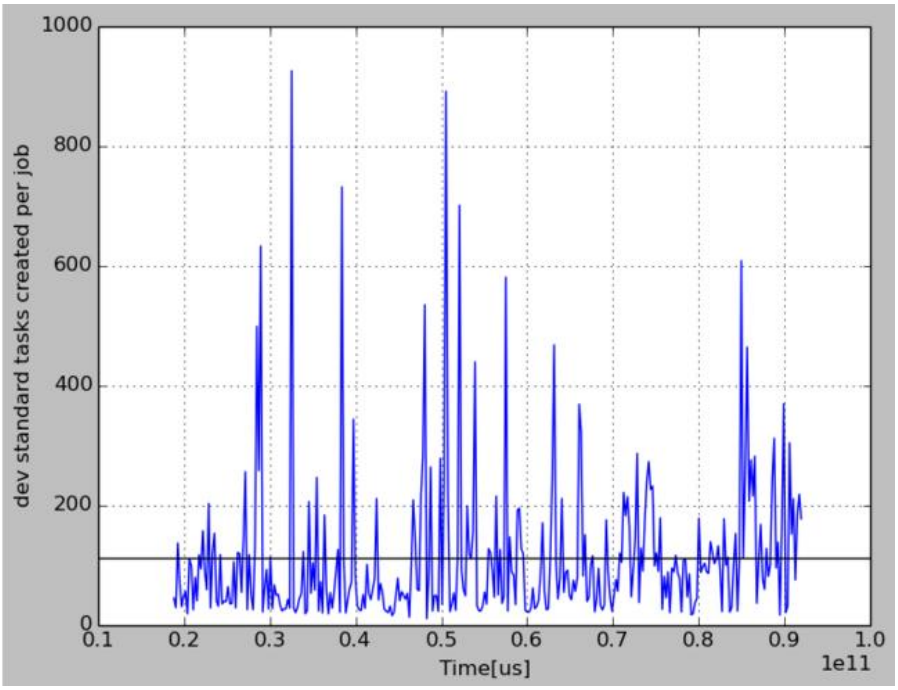
[Fig. 2.7.2 – Standard deviation of the number of tasks created in 24 hours (granularity 8 pt)]

2.2.2.4.2.2 *Time interval of 24 hours –granularity is 16 points per hour*



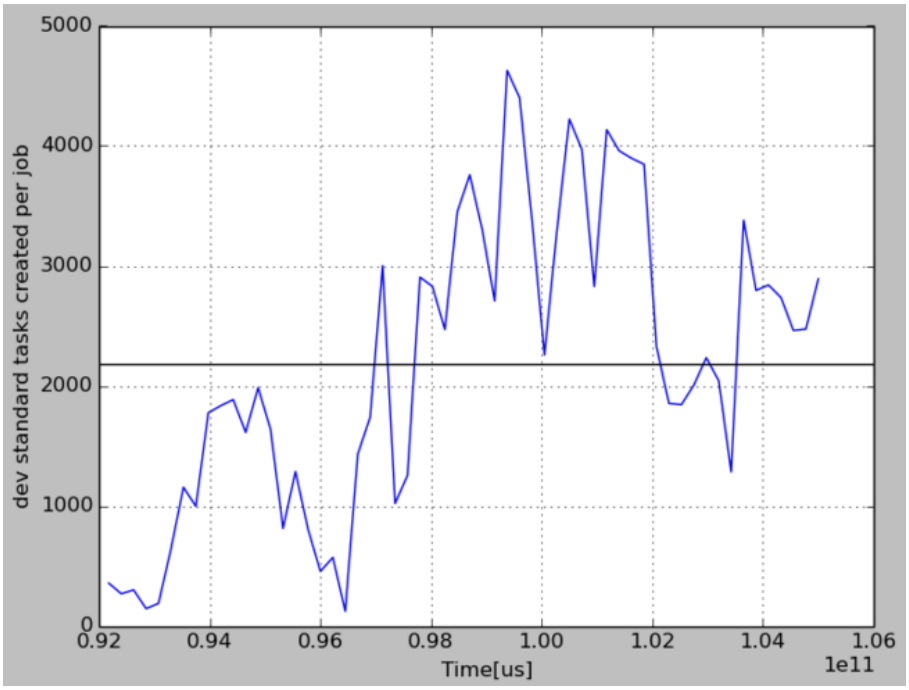
[Fig. 2.8 – Standard deviation of the number of tasks created in 24 hours (granularity 16 pt)]

2.2.2.4.2.2.1 Cluster 1: standard deviation of task created per job (16 points per hour)



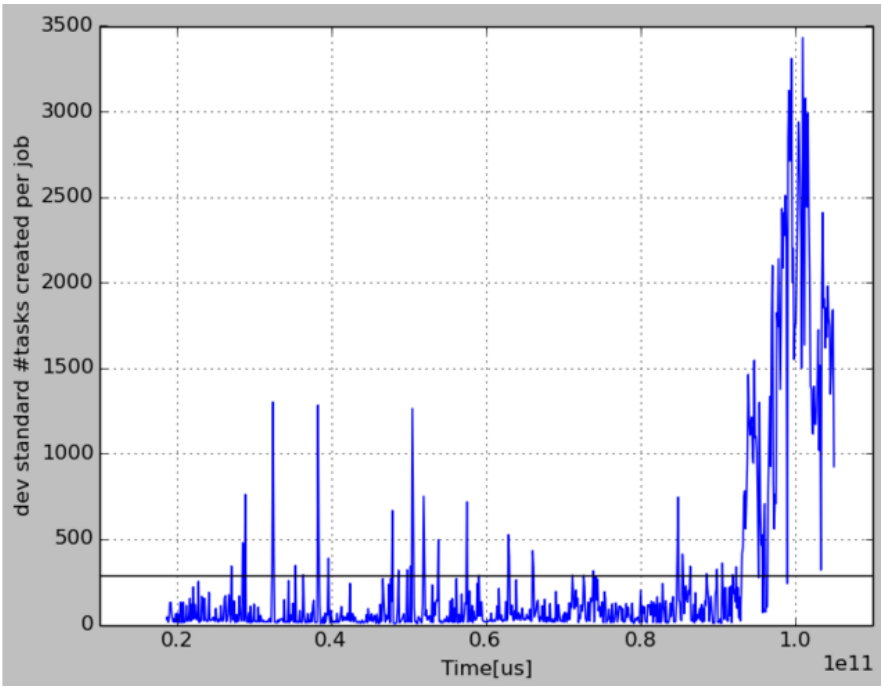
[Fig. 2.8.1 – Standard deviation of the number of tasks created in 24 hours (granularity 16 pt)]

2.2.2.4.2.2.2 Cluster 2: standard deviation of task created per job (16 points per hour)



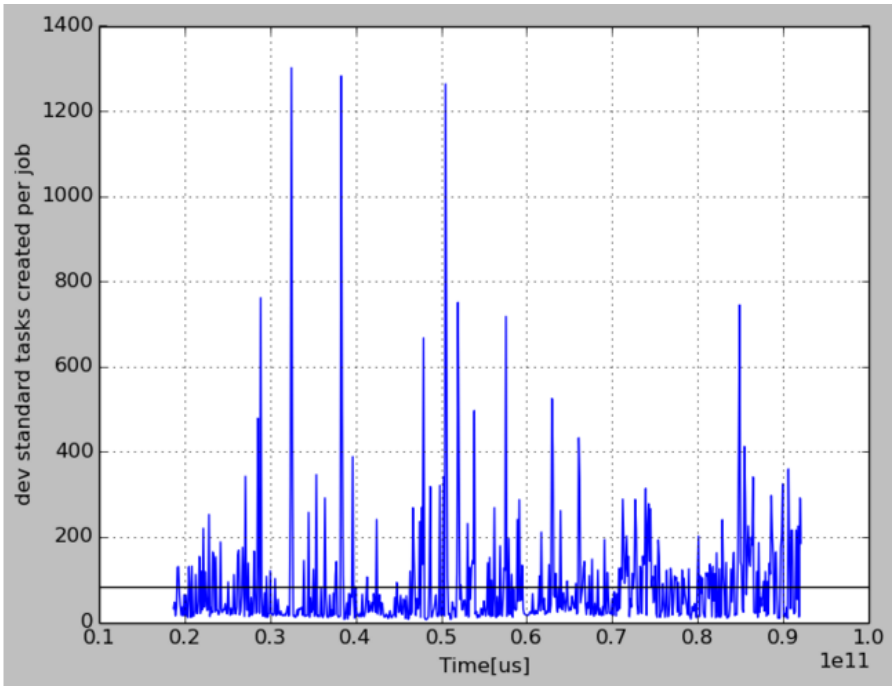
[Fig. 2.8.2 – Standard deviation of the number of tasks created in 24 hours (granularity 16 pt)]

2.2.2.4.2.3 *Time interval of 24 hours –granularity is 32 points per hour*



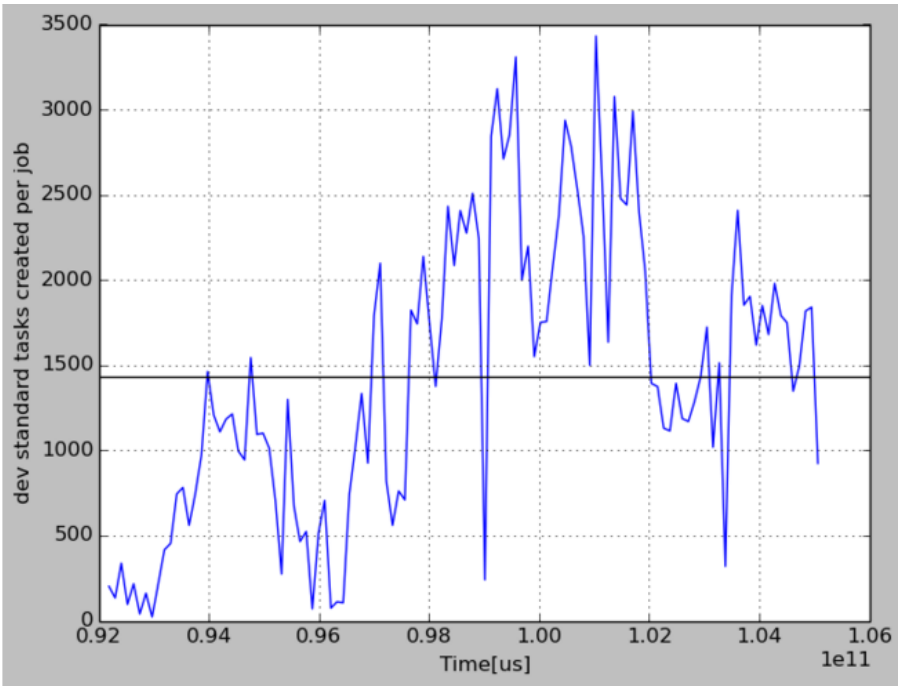
[Fig. 2.9 – Standard deviation of the number of tasks created in 24 hours (granularity 32 pt)]

2.2.2.4.2.3.1 Cluster 1: standard deviation of task created per job (32 points per hour)



[Fig. 2.9.1 – Standard deviation of the number of tasks created in 24 hours (granularity 32 pt)]

2.2.2.4.2.3.2 Cluster 2: standard deviation of task created per job (32 points per hour))



[Fig. 2.9.2 – Standard deviation of the number of tasks created in 24 hours (granularity 32 pt)]

2.3 Task per machine

2.3.1 Output file structure

startTime[micros],endTime[micros],mean

- File "start time[h] – machineId – event: event_type – granularity" .txt - .csv
- Internal structure:
  - Start time - beginning of the evaluation period
  - End time - end of the evaluation period
  - Mean - number of tasks (scheduled - completed) per machine in the time interval [end time - start time]
  - MachineID - identifier of the machine analyzed

Example - 5L-906113event-4-3600L

Machine Id: 906113

Event type: 4 (finish)

Granularity: 3600

startTime[micros]	*endTime[micros]*	*mean*
29400000000	33000000000	2.000000

2.3.2 Analysis

First we analyzed the traces of our interest, achieving the machineID and producing the tuple (timestamp [Micros], machineID, event_type).

Then calculated the number of machines, more than 12,000.

```
googleRDD=(sc.textFile(path)
    .map(lambda x:x.split(','))
    .filter(lambda x:x[4]!='')
    .map(lambda x:(int(x[0]),int(x[2]),int(x[4]),int(x[5])))
    .filter(lambda x:x[1]==machine_id)
    .filter(lambda x:x[0]>=startTimeMs and x[0]<=endTimeMs))

dataRDD=googleRDD.filter(lambda x:x[2]==event_type)
```

Initially the analysis was designed to analyze for each machine the number of tasks created.
Since we should have to make calculations on huge amount of tasks for all of the machines, which would be extremely large in terms of time cost, we decided to focus only on the most significant machines. Significant machine means the first 10 cars that schedule more tasks (event_type = 1).

```
def findMostUsedMachines(path):
    machineRDD=(sc.textFile(path)
                .map(lambda x:x.split(','))
                .filter(lambda x:x[4]!='')
                .map(lambda x:(int(x[4]),int(x[5])))
                .filter(lambda x:x[1]==1)
                .map(lambda x:(x[0],1))
                .reduceByKey(lambda x,y:x+y))
    print "First 10 most used machines"
    print machineRDD.takeOrdered(10,lambda k:-k[1])
```

At this point, we calculated the number of tasks completed for the selected machines (event_type = 4) in the time interval, value that will allow to find the throughput.

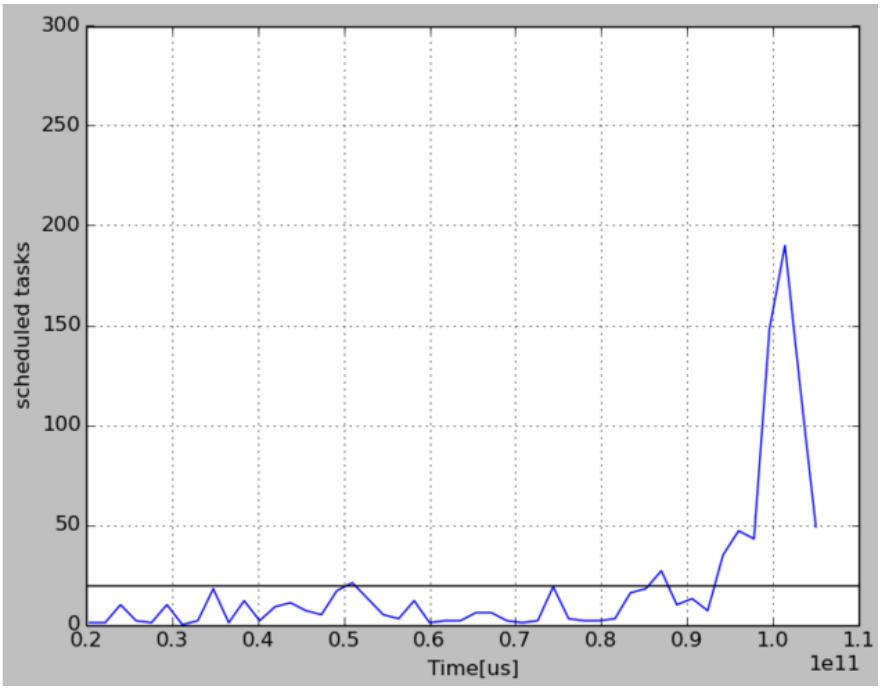
To obtain comparison data, we calculated the mean of the number of tasks generated per machine in the time interval. The average value for machine is negligible compared to the total number of tasks that are generated, supposing that the reasons is because we are in the presence of a large number of machines and then to each machine will be submitted on average a low number of tasks.

Afterwards there are graphs of some machines, in order, graph of scheduled task and of completed tasks per machine. Is important to underline that the graphs of tasks / machines does not show clusters for completed task as it was for the trend of the histogram (Fig. 1), for this reason it was not necessary to make a separate analysis for each cluster. While the presence of the clusters remains in the case of the scheduled tasks for the machine.

The following graphs show the relationship between the number of tasks created per machine in the time interval (blue broken line) and the total mean of tasks created by that machine (black line). The number of tasks created for machine is graphically represented with a broken line, obtained by calculating the mean of the task / machine with a granularity of one hour in the time interval under consideration.

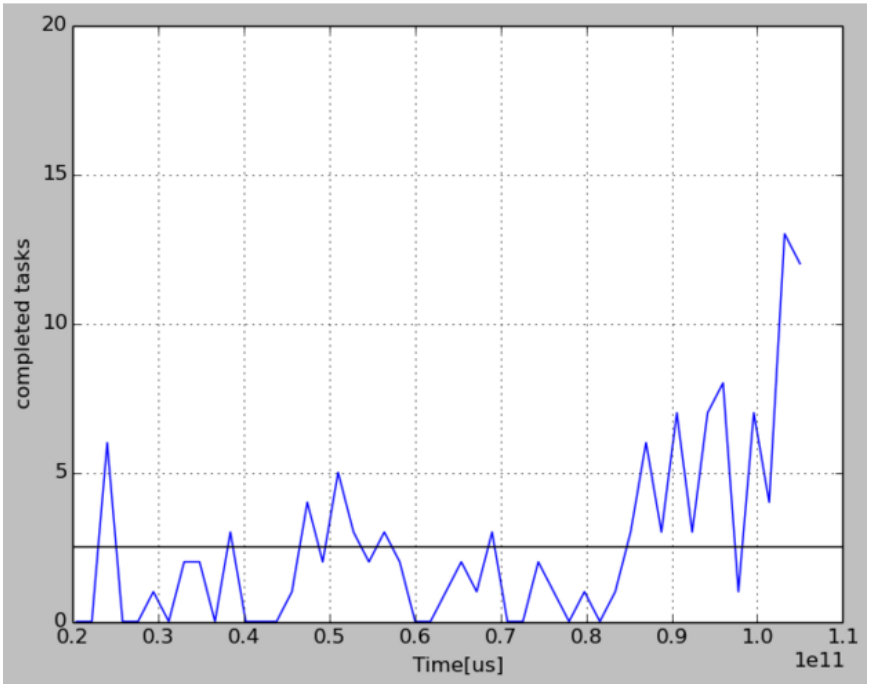
2.3.2.1.1 Machine ID 257407216

2.3.2.1.1.1 *Number of tasks scheduled in machine 257407216*



[Fig. 2.10.1 Number of scheduled tasks by the machine 257407216]

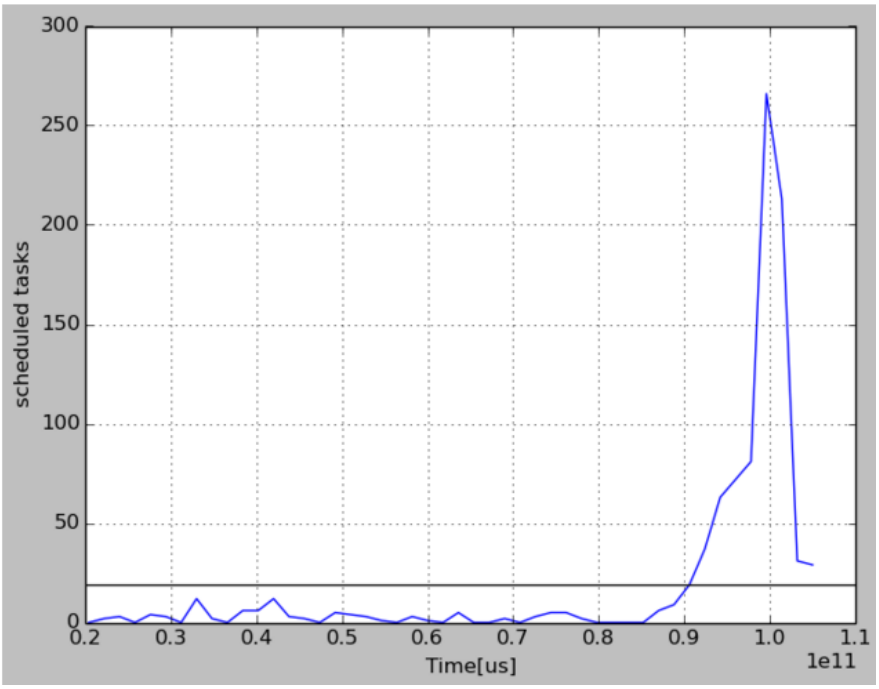
2.3.2.1.1.2 *Number of tasks completed in machine 257407216*



[Fig. 2.10.2 Number of completed tasks by the machine 257407216]

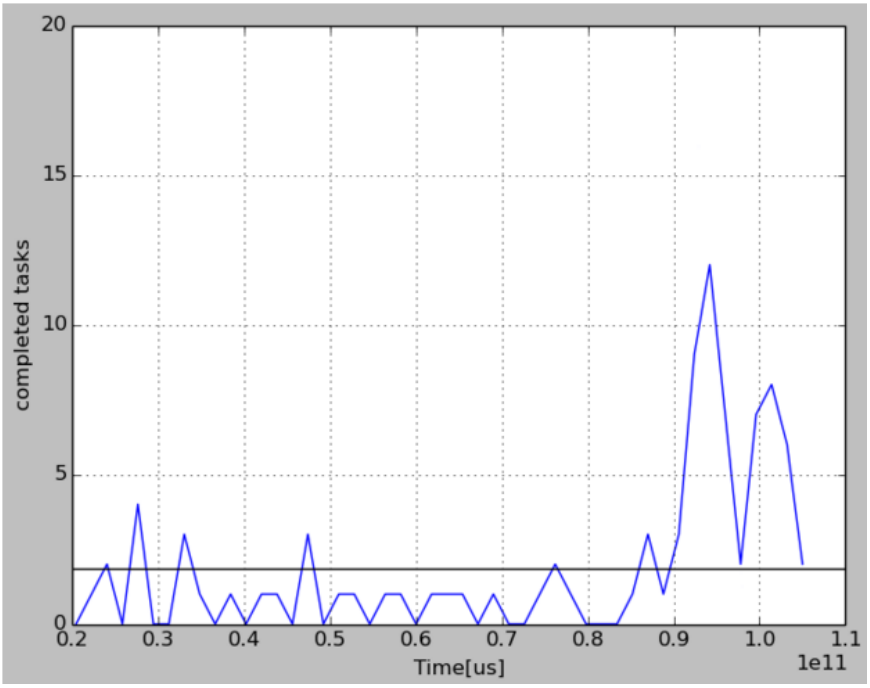
2.3.2.1.2 Machine ID 4820095525

2.3.2.1.2.1 *Number of tasks scheduled in machine 4820095525*



[Fig. 2.11.1 Number of scheduled tasks by the machine 4820095525]

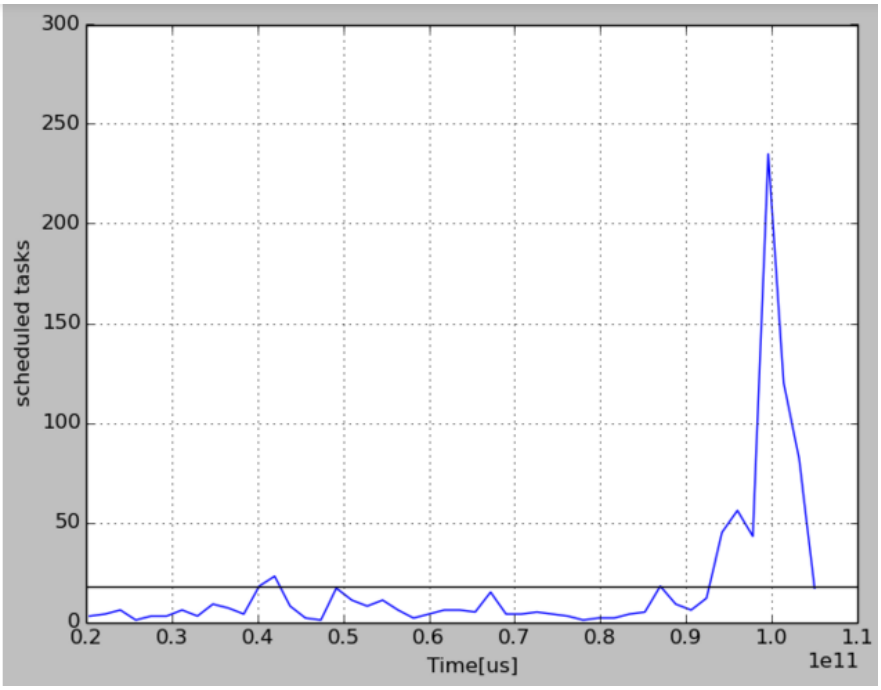
2.3.2.1.2.2 *Number of tasks completed in machine 4820095525*



[Fig. 2.11.2 Number of completed tasks by the machine 4820095525]

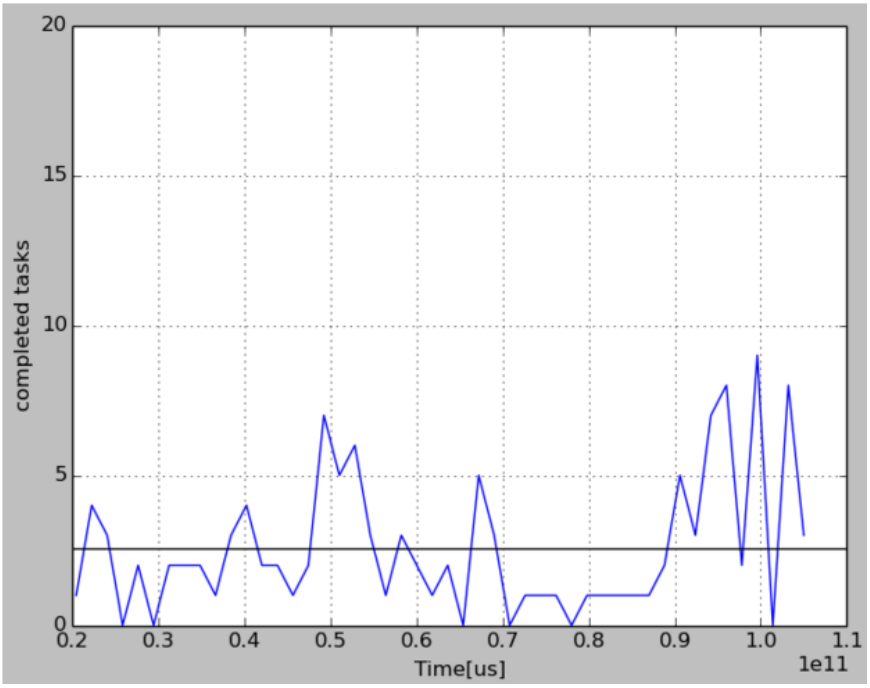
2.3.2.1.3 Machine ID 4820310861

2.3.2.1.3.1 *Number of tasks scheduled in machine 4820310861*



[Fig. 2.12.2 Number of scheduled tasks by the machine 4820310861]

2.3.2.1.3.2 *Number of tasks completed in machine 4820310861*



[Fig. 2.12.2 Number of completed tasks by the machine 4820310861]

As a result the table (Table 1) summarizes schematically and justifies the results obtained from the graphs. The table is obtained by calculating the mean in one hour of tasks for each machine for each .csv file, each of which is identified by MachineID –eventType notation.

ID macchina	Scheduled tasks (1)	Evicted tasks (2)	Failed tasks (3)	Completed tasks (4)	Tasks in execution
257407216	38,958	9,191	21,625	5,080	3,062
4820095525	38,333	7,000	25,833	3,750	1,75
4820310861	36,208	8,208	20,625	5,208	2,167

[Table 1]

The values obtained in the table of the number of scheduled and completed tasks for each machine is consistent with the average value obtained graphically (black line on the graph).

The hourly mean of tasks /machine is calculated, for each type of event (where the event type is identified in the name of the file MachineID –eventType), such as:

$$\frac{\sum_{i=1}^{i=24} number\ of\ task_{eventType}}{total\ number\ of\ tasks}$$

The last column (task still in execution) identifies the number of remaining running task in the machine t the end of the temporal interval in analysis.

3 Conclusions

3.1 Tasks per job

During the analysis of tasks per job, we noticed a constant trend in the average number of tasks generated for each job, apart from some irregular points (clusters). We have presumed different hypotheses, one of the most reliable deals with large operations that are carried out at the beginning of the month, for example, backup operations.

But having no information on the type of task , it is not possible to confirm these conjectures.

3.2 Tasks per machine

Following the results obtained in Table 1, it is evident that most of the tasks are submitted to a machine will end with a completion (of type 4) or will be killed (event type 2, 3). The hypothesis that has been made to justify the large number of failed tasks, is that during the execution of a task, can arrive a more important task that causes the stop or killing of the others; otherwise that is launched parallel research on multiple machines and submitted tasks are killed when one of them finds the search result.

But, as well as for the tasks per job, having no knowledge on what type of tasks are performed by machines, we cannot corroborate the hypothesis.

4 Technologies

- Spark
- Phyton

5 Documentation

- ***"Google cluster-usage traces: format + schema".*** *Charles Reiss, John Wilkes, Joseph Hellerstein. Version of 2013-05-06, for trace version 2. Revised 2014-11-17 for trace version 2.2.*
- ***"Heterogeneity and dynamicity of clouds at scale: Google trace analysis".*** *Proceedings of the Third ACM Symposium on Cloud Computing. ACM, 2012.*
- *https://commondatastorage.googleapis.com/clusterdata-2011-2/SHA256SUM*