Modeling and Generating large-scale Google-like Workload
Georges Da Costa, Léo Grange, Inès De Courchelle

To cite this version:
Georges Da Costa, Léo Grange, Inès De Courchelle. Modeling and Generating large-scale Google-like Workload. International Workshop on Resilience and/or Energy-aware techniques for High-Performance Computing, Nov 2016, Hangzhou, China. 2016. <hal-01472021>
Modeling and Generating large-scale Google-like Workload

Georges Da Costa*, Léo Grange*, Inès De Courchelle**†
* IRIT Laboratory, University of Toulouse, France, {Georges.Da-Costa,Leo.Gran,Ines.De-Courchelle}@irit.fr
† CNRS, LAAS, Toulouse, France

Abstract—One of the key element needed to test most large-scale scheduling algorithms is a testing infrastructure. Large scale is of upmost importance as failures and complex behaviors are common occurrences only at such scale. In order to test the reaction of a system to failures or extreme behaviors, it is necessary to be able to create large scale environments. Such an infrastructure must be reproducible so that several work are able to compare themselves but also capable of diversity as otherwise it would risk to lead to particular subcases. In this article, we propose a generic adaptable and reusable model of large scale workload. The original schema comes from the Google Cluster Workload Traces which is a perfect representative of large scale production workload. Contrary to most model analysis of such traces, we propose along with our model a reference implementation in order for other studies using our results to produce comparable experiments.

I. INTRODUCTION

With the increase of demand from the scientific community, HPC infrastructure are subject to more constraints. New datacenters are built and already existing one are subject to high demand from users. In order to improve the usage of these infrastructure, a large computer science community research innovative algorithms and middlewares [9]. The three main tools actually used are mathematical models, simulation and experimentation on real hardware. Contributions are new scheduling and methods of resource management in order to improve several metrics such as energy, performance, resilience, throughput or dynamism.

In most works, these contributions are evaluated with perfect hardware. At large scale, several hardware elements will fail even during a small time period. Thus, in order to evaluate these contributions using simulations a complete environment is needed, from the input workload to hardware failure models.

One of the key element needed to reliably evaluate these contributions are precise, realistic and adapted models of workload. Most study use workload extracted from particular infrastructures [7]. For the purpose of evaluation of scheduling, the mostly used way to take advantages of these data is to directly replay the traces, reducing the possibilities of test to a specific case with particularities. Few studies [3] provide more detailed information on the global characteristics of usable traces but are not sufficient to generate similar ones for replaying purpose.

This article will model and propose an implementation of a generic trace generator based on realistic traces from a production environment with a large number of different usages. The traces used as reference are from the Google Cluster [13] because they were used for internal heavy computing and data treatments internally on more than 10,000 servers for over a month. In this matter, the presented workload is similar to the one running on an HPC infrastructure shared by a large number of users.

By using the configuration possibilities of the proposed model and implementation it becomes possible to simulate large scale HPC workload. Such workload is necessary to evaluate the schedulers and middlewares capabilities from several point of view, performance, energy and resilience. Indeed, the scale of the proposed models is necessary for these types of evaluation in order to exhibit particular behaviors such as failing servers or thermal effects with impact on servers energy.

This article is structured as follow: The next section will describe the inner characteristics of the Google Cluster Workload. Then, the next section will describe the state of the art of the workload models extracted from these traces. The following section will propose a generic model for HPC workload along-with models for each of its characteristics. Before the conclusions and perspectives, a last section will propose a reusable workload generator.

II. WORKLOAD CHARACTERISTICS

The data used in this study and analyzed below come from one of the datacenter of Google. Google provided a dataset [13] of some of its servers usage in 2011. The monitored datacenter is composed by more than 12,000 servers with heterogeneous characteristics. The different statistics were collected for a period of 29 days. Data have been anonymized to protect the servers configuration (CPU, Memory, Disk) by being normalized between 0 and 1. The data consists in a zipped collection of files taking 43 GB compressed. It includes six main data tables: job_events, machine_attributes, machine_events, task_constraints, task_events and task_usage. A job is composed by one or many tasks. The present article focuses on the task_events table. This record contains 500 CSV files, 25 millions tasks. This record encompass 100 millions events which describe each task and their life cycles. During its lifetime, a task has a state: Unsubmitted, Pending, Running or Dead. An event indicates a transitions between states (9 events):

The initial task event is Submit which allows the task to be scheduled. The Schedule event means that a task has been...
scheduled on a particular server. There are ending events such as *Evict* which means that a task has been unscheduled because of a higher priority task; as *Fail* is when a task was unscheduled because of a task failure; *Finish* means that a task finishes normally; *Kill* is when a task is canceled by a user or a driver program or when a linked task died; Finally *Lost* is an event to characterize missing info. Actually, most tasks finish with either *Finish* or *Fail* due to limits of the monitoring infrastructure.

Considering the 500 CSV files which describe task events, the results show that task pass throws several events. Table I describe the distribution of number of events for each tasks. 95% (24,258,907 out of 25,424,731) of tasks have only the three classical events: An initial one *Submit*, a *Schedule* one and an ending event. Most other tasks (4.5%) are the one which are submitted, scheduled but do not finish during the time frame of the acquired data. The remaining (less than 0.5%) are tasks that have intermediate events of reconfiguration. In the following of this article, tasks taken into account are the one composed of three events.

The way the tasks are ending is also important. Table II, describes the distribution of finishing state for all tasks with three events.

### Table I

<table>
<thead>
<tr>
<th>Number of events</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of tasks</td>
<td>248</td>
<td>1,093,992</td>
<td>24,258,907</td>
<td>19,378</td>
<td>17,215</td>
</tr>
<tr>
<td>Number of events</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9+</td>
<td>total</td>
</tr>
<tr>
<td>Number of tasks</td>
<td>8,510</td>
<td>5,093</td>
<td>3,028</td>
<td>18,360</td>
<td>25,424,731</td>
</tr>
</tbody>
</table>

### Table II

<table>
<thead>
<tr>
<th>Events type</th>
<th>Finish</th>
<th>Kill</th>
<th>Fail</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of tasks</td>
<td>17,775,284</td>
<td>6,381,906</td>
<td>86,348</td>
<td>15,369</td>
</tr>
<tr>
<td>Percentage</td>
<td>73.31 %</td>
<td>26.31 %</td>
<td>0.35 %</td>
<td>0.03 %</td>
</tr>
</tbody>
</table>

Schedulers or middleware need information on the requirements of tasks in order to evaluate the quality of their decision. One important such requirement is the impact of time on their allocation. In most systems, tasks are labeled with their priority or deadline. These tasks properties affect the policy of the scheduler.

The Google Workload Traces collection contains, for each task, two properties for their *priority* and *scheduling class*. According to the Google’s documentation [13], the priority is used by the cluster’s scheduler, whereas scheduling class is used locally by a machine to manage resource usage. The following studies are from a sample of 11 millions tasks from the tasks files.

The table III gives the number of tasks for each scheduling class. This class value is between 0, for the least latency-sensitive tasks, to 3 for the most critical tasks.

### Table III

<table>
<thead>
<tr>
<th>Scheduling class</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks number</td>
<td>20,317,398</td>
<td>3,327,283</td>
<td>533,330</td>
<td>49,614</td>
<td></td>
</tr>
<tr>
<td>Percentage</td>
<td>83.86 %</td>
<td>13.73 %</td>
<td>2.20 %</td>
<td>0.21 %</td>
<td></td>
</tr>
</tbody>
</table>

Priority level is a number between 0 and 11, with 0 as the lower priority. Table IV provides the count of tasks per priority value.

### Table IV

<table>
<thead>
<tr>
<th>Priority</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks</td>
<td>5,701,546</td>
<td>2,357,274</td>
<td>1,078,476</td>
<td>1,027</td>
<td>13,975,078</td>
<td>104</td>
</tr>
<tr>
<td>Percentage</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>Tasks</td>
<td>633,445</td>
<td>0</td>
<td>249,932</td>
<td>230,164</td>
<td>579</td>
<td>0</td>
</tr>
</tbody>
</table>

For this specific dataset, the priorities are logically grouped in 4 categories. Those groups are, sorted by increasing priorities, *free*, *normal*, *production* and *monitoring*. The monitoring group is, however, also included in the *production* group, according to the dataset documentation. We decided to consider the priority groups exclusive, as it seems to be a mistake in the documentation. Table V is obtained by grouping the data from table IV using those priority groups. It appears that, in terms of task count, the *monitoring* group is almost insignificant, with about 0.002 % of the total.

### Table V

<table>
<thead>
<tr>
<th>Priority group</th>
<th>Free</th>
<th>Normal</th>
<th>Production</th>
<th>Monitoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks number</td>
<td>8,058,820</td>
<td>15,938,062</td>
<td>230,164</td>
<td>579</td>
</tr>
<tr>
<td>Percentage</td>
<td>33.26 %</td>
<td>65.78 %</td>
<td>0.95 %</td>
<td>0.002 %</td>
</tr>
</tbody>
</table>

There is no direct and absolute relationship between scheduling class and priority. A task with a high priority may also have a low scheduling class and vice versa. However, the figure 1 shows the statistical relationship between those.
For each scheduling class, it gives the distribution of corresponding tasks between the priority groups. Except from monitoring class, there is a small correlation between priority and scheduling class. The higher the scheduling class, the higher the priority. Concerning the global distribution, most tasks are in the normal scheduling class for each priority.

Depending on the targeted systems, these data can be directly transposed in a metric showing the quality of the allocation.

III. STATE OF THE ART

Several studies have previously focused on a better understanding of the Google traces dataset and of the characteristics of the described tasks.

In [12], authors described the heterogeneity of the hardware resources in the Google traces. They classified priorities in five categories Infrastructure(11), Monitoring(10), Normal production (9), other (2-8) and Gratis(0-1). They identify the boulders and sand of this workload: They have classified tasks in a majority of small tasks (sand) and an important number of large tasks (boulders). In this analyze, they shown that 2% of tasks represent 80% of CPU, Memory and that 92% are long tasks with a Free priority. In [11], Reiss et al. analyzed the Google cluster performance. According to this article, many long jobs have stable resource utilization, which helps the adaptive resource schedulers. They concluded that machine configuration and workload composition are heterogeneous. This analyzed trace allows us to understand the Google dataset, the importance of an heterogeneous datacenter to adapt the resources to the demand. However, they did not provided information related on how to use this workload in another context.

The paper [6] evaluates the gap between the requested resources and the one consumed by tasks within the datacenter. The requested average load for a processor is 10% on the Google datacenter. This Google trace analysis shows that processors are overall under utilized which leads to an increase of the energy consumed (90% of available processor computing power is not used). Moreover, most of the workload has a low priority and is not sensitive to the latency. It shows that there is only a very few number of sensitive tasks (scheduling class: 2 or 3). In this paper, authors focus on the lack of energy consideration in the Google trace.

Alam et al. [1] provided a statistical analysis of the traces to make some reference job profiles emerge. They based this approach on resource usage, clustering of workload patterns and classification of jobs with k-means clustering. They have demonstrated that jobs are trimodal in nature. They can be Long jobs, Short jobs and Medium jobs. Each job type can also be sub-categorized as Less resource usage, Mid resource usage and Resource hungry. They clustered jobs with a k-mean with k equals to 5 (number of classes). Jobs can also be classified into short, approaching mid, mid, receding long and long. This clustering aims at use the workload in a simulator. Clustering of tasks profiles is also the main result of [2]. These approaches using k-means lead to difficulties to generalize the workload for producing new instance of similar workload.

In [10], authors analyzed how the servers are managed in the cluster and how the workload behaves during the period of monitoring. According to them, there is over 870 machines events on average each days. In this study, they have clustered the machine population per same CPU and memory (15 groups). Contrary to [11], [10] shown that the machines are almost homogeneous, 93% machines have the same CPU capacities and 86 % of the machines have the same memories capacities. Also they have explored the workload behavior. A lot of jobs are frequently killed, and during the job lifetime, 40.52% which are scheduled are killed at least once.

Di et al. [5] evaluated the Google trace compared to other Grid/HPC systems. They found that the Google dataset have a finer resource allocation with respect to CPU and memory than Grid/HPC systems. They compared the CDF (cumulative distribution function) of the job length, 55% of tasks finish within 10 minutes for the Google dataset. Others studied traces have shorter task, they explain this difference because of the users and applications which can include commercial applications such as web services. In addition, Google jobs are submitted with much higher and more stable frequency than that of Grid jobs. In [4], Di et al. have observed that, for the Google workload, the resource utilization per application (logic job name) follows a typical Pareto principle (joint ratio < 2%) as for the jobs/events per applications (joint ratio almost 10%). Di et al. used a K-means clustering algorithm based on task events and resource utilization to classify application. The number of applications in the K-means clustering sets follow a Pareto-similar distribution. They shown that all applications can be split into 4 types (single-task application, sequential-task application, batch-task application and mix-mode application). They exposed a correlation between task events and the four application types. For example, 81.3% of failed task events belong to batch-task applications. These two publication shows that except for the length of jobs, the behavior of tasks in the Google workload is similar to the one seen in Grid/HPC systems.
These studies goals mostly were to understand and characterize the workload rather than to propose complete models and implementation for providing the ability to generate new similar workload.

IV. WORKLOAD LAWS

In a general way for a large scale datacenter, a workload is a list of tasks defined by their:

- Starting time: Usually defined as the inter-arrival time between tasks
- Type: Service or task
- Priority: Used to evaluate the relative importance of tasks
- Makespan: Length of the tasks in seconds

Based on the provided data, we can extract the required laws.

A. Submission Time

As described in the state of the art [12], tasks are submitted continuously on the monitored platform which is highly charged. Very few inter-arrival times are over 10s, and the average is 0.052s. The inter-arrival law is well modeled by a Pareto distribution, with a $\lambda$ parameter set to 5 in the case of the Google Dataset as shown on figure 2.

A gap for very small intervals can be explained as a large number of tasks can be simultaneously submitted, breaking the assumption of independent tasks. But even with this remark, a Pareto distribution simulates well the inter-arrival time.

B. Type

As described in section II, most of the tasks end with one of the Finish or Kill events. In the first case, the task just finishes its work and ends normally. In the second case, the documentation is unclear about which are the possible causes of a Kill event.

By analyzing those two sets, it appears they don’t follow the same statistical properties. Figure 3 shows the average execution time, depending on both priority and finished or killed groups. Whereas the makespan for the finished group tends to decrease when the priority increase, the opposite is observed for killed group. In addition, the killed group is always longer in average. The difference is the most significant for production level, with about 493 seconds for finished group compared to 148424 seconds for killed tasks.

Those different characteristics, especially the important execution time and the number of high-priority tasks in the killed group, suggest two different populations. Our hypothesis is that most of the killed tasks are, in fact, long-running services. It may be some monitoring services, web servers, or any other task implementing a service for other internal or external tasks. This kind of service have no pre-determined duration, and may be started and killed to scale with the demand or with the cluster usage. The killed set should also contain some tasks killed manually for other reasons, but it can’t explain the amount of production tasks stopped this way.

In the following sections, we will use task to refer to a finite task, which have a given amount of work to do before to finish normally. As an example, typical kinds of tasks in a such cluster are MapReduce jobs, each containing a set of map and reduce tasks. The word service will be used to refer to a task which, at the opposite, have no precise amount of work to do, and therefore just run until stopped.

C. Makespan

The execution time of tasks is one of the main characteristics of a workload, as it differs a lot depending on the kind of application it contains. Particularly, the task makespan is clearly different between traditional grid workloads and this cloud trace, as pointed by Di et al. [5]. As most other characteristics are similar, a method to have more HPC-like traces would be to increase the average makespan.

1) Tasks: In figure 4, the distribution of finished tasks makespan shows a huge number of small tasks, and few very long ones. The shape is typical of long-tailed distributions, like Pareto or log-normal. Table VI gives more precise statistical information. The shape of the distribution is confirmed by
Fig. 4. Distribution of makespan for tasks finished normally

Fig. 5. Distribution of execution time before the Kill event for killed tasks

TABLE VI

<table>
<thead>
<tr>
<th>Priority</th>
<th>Count</th>
<th>Average</th>
<th>Std. dev.</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>Free</td>
<td>5,081,775</td>
<td>1,937.9</td>
<td>4,518.7</td>
<td>345.7</td>
</tr>
<tr>
<td>Normal</td>
<td>12,677,000</td>
<td>1,600.3</td>
<td>5,231.4</td>
<td>492.6</td>
</tr>
<tr>
<td>Production</td>
<td>49,190</td>
<td>493.9</td>
<td>1,955.1</td>
<td>39.1</td>
</tr>
<tr>
<td>Monitoring</td>
<td>0</td>
<td>N.A.</td>
<td>N.A.</td>
<td>N.A.</td>
</tr>
<tr>
<td>All</td>
<td>17,807,965</td>
<td>1,693.6</td>
<td>5,034.5</td>
<td>448.9</td>
</tr>
</tbody>
</table>

The gap between the median value and the average. Standard deviation is also important compared to average, another property of long-tailed distribution.

To use the more appropriate distribution for modeling the makespan, we compared the Kolmogorov–Smirnov (KS) statistic between several fitted distribution and the original data. The more the KS statistic is close to 0, the more the chosen distribution is similar to the real data. A fit with a Pareto distribution gives a Pareto coefficient of 0.113 and a scale factor of 0.073 for these tasks. Its calculated KS statistic is 0.5. By looking at the KS statistic, the better model to describe this characteristic is the log-normal law.

2) Services: The figure 5, similarly, shows the same kind of distribution, with even longer makespans. By comparing table VII with the statistics of finished tasks, it appears that the average makespan is about 5 times higher, with a lower median. A fit with a Pareto distribution gives a Pareto coefficient of 0.0916 and a scale factor of 0.0077 for these tasks. Using a log-normal law, fitting it gives \( \sigma = 2.06 \) and \( \mu = 6.14 \). The calculated values of KS statistic are 0.49 for the Pareto distribution, and 0.064 for the log-normal one. For the services too, using a log-normal distribution to model their makespan is accurate.

D. Priority

The exact semantic of priority and scheduling classes is not specified in the original document. From the combination of priority and scheduling class, importance of tasks and services can be deduced. To represent this relative importance in our model, we use a value in the continuous interval \([0, 1]\). The lower this value is for a task, the lower its priority is.

1) Tasks: For tasks, as shown on figure 6, there is a strong relation between priority and scheduling class. To model this behavior, a simple categorization is sufficient. From the statistic of occurrences, 84% are of the lowest priority, 15% of the next one, and the remaining 1% of the highest ones. It can be simulated with an exponential law, truncated between 0 and 1, with a high \( \lambda \) parameter.

2) Services: The figure 5, similarly, shows the same kind of distribution, with even longer makespans. By comparing table VII with the statistics of finished tasks, it appears that the average makespan is about 5 times higher, with a lower median. A fit with a Pareto distribution gives a Pareto coefficient of 0.0916 and a scale factor of 0.0077 for these tasks. Using a log-normal law, fitting it gives \( \sigma = 2.06 \) and \( \mu = 6.14 \). The calculated values of KS statistic are 0.49 for the Pareto distribution, and 0.064 for the log-normal one. For the services too, using a log-normal distribution to model their makespan is accurate.
used a log-normal law to model it. Such similarities between several kinds of workloads show that our model, based on a specific dataset, keeps some important characteristics shared by workloads in other datacenter use cases.

V. GENERATING WORKLOAD

The law described in the previous section are simulating exactly the same behavior as the workload traces from the Google Datacenter. To create more generic workloads it is needed to be able to provide configuration possibilities needed to adapt to different types of experiments and context.

There are two main characteristics needed to define a particular workload: Dynamism and Type

- **Dynamism:** Represents the dynamism of tasks, the number of arriving tasks
- **Type:** Is a value between 0 and 1 representing the percentage of tasks in the total workload.

Two other characteristics are to be specified for both tasks and services: Mass and Disparity

- **Mass:** Represents the average makespan
- **Disparity:** The ratio between the average and the median makespan, must be greater than 1

For these Google traces, the dynamism, which is the average inter-arrival time, has a value of 0.05 (in seconds) for Google Traces. As the percentage of tasks in this workload is 70%, we use a value of 0.7 for type.

The mass and disparity parameters are calculated using the average and median values from the tables VI and VII. For tasks, we have a mass of 1700 and a disparity of 3.8. The values found for services are respectively 8000 and 24.

The implementation of the generator is shown in algorithm 8 in the Python3 language.

VI. CONCLUSION

In this article we proposed the models needed to create a Workload generator aimed at large scale homogeneous clusters. The traces used as basis of this work are large scale (from time and space point of view) realist data from a Google internal datacenter. The second contribution is an actual implementation of the workload generator with possibilities to adapt to several usecases. This generator will open new possibilities for testing large scale datacenter and will help providing insight for several scheduling methods based on different metrics such as performance, energy, reliability and dynamism.

The next step will be to augment the generator with more diverse possibilities such as diagram of tasks, constrained resources or MPI tasks. Another important step will be to use it in large scales simulators in order to evaluate scheduling policies.

VII. ACKNOWLEDGMENT

This work was supported by the ANR DATAZERO project, grant ANR-15-CE25-0012 of the French Agence Nationale de la Recherche and by the neOCampus operation funded by University Paul Sabatier, Toulouse, France.
```python
import scipy.stats
import random
import numpy

def truncated_expon(lamda):
    while True:
        val = scipy.stats.expon.rvs(scale=1.0/lamda)
        if val <= 1.0:
            return val

def get_makespan(mass, disparity):
    mu = numpy.log(mass / disparity)
    sigma = numpy.sqrt(2*(numpy.log(mass) - mu))
    return scipy.stats.lognorm.rvs(sigma, scale=mass / disparity)

def get_next_task(timestampLastEvent, dynamism, ratioTask, tasksMass, tasksDisp, servMass, servDisp):
    arrival = scipy.stats.pareto.rvs(4, loc=-1) * 3.0 * dynamism
    newTimestamp = timestampLastEvent + arrival
    if random.random() < ratioTask:
        ttype = 'TASK'
        priority = truncated_expon(6)
        makespan = get_makespan(tasksMass, tasksDisp)
    else:
        ttype = 'SERVICE'
        priority = truncated_expon(3)
        makespan = get_makespan(servMass, servDisp)
    return (newTimestamp, ttype, priority, makespan)
```

Fig. 8. Reference implementation of tasks generator implemented in python

REFERENCES