Modeling, classifying and generating large-scale Google-like workload

Georges da Costa, Léo Grange, Inès de Courchelle

Modeling, classifying and generating large-scale Google-like workload

Georges Da Costa [a,*], Léo Grange [a], Inès de Courchelle [a,b]

[a] IRIT Laboratory, University of Toulouse, France
[b] CNRS, LAAS, Toulouse, France

A B S T R A C T

One of the key elements 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 studies are able to compare themselves but also capable of diversity as other wise it would risk to limit them 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. The methodology to produce the model can be used at different precision levels and can classify the input level without human supervision. 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.

Keywords:
Workload analysis
Workload generation
Classification
Large scale
Google dataset
Reproducible experiments

1. Introduction

With the increase of demand from the scientific community, HPC infrastructures are subject to ever increasing demands. New datacenters are built and already existing ones are subject to high demand from users. In order to improve the usage of these infrastructures, a large computer science community is researching innovative algorithms and middlewares [1]. The three main tools actually used are mathematical models, simulations and experimentations 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 [2,3].

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 elements needed to reliably evaluate these contributions is a precise, realistic and adapted workload model. Most studies replay workload extracted from particular infrastructures [4]. For the purpose of evaluation of scheduling, the most used way to take advantage of these data is to directly replay the traces, thus reducing the test possibilities to a specific case with particularities. Few studies [5] provide more detailed information on the global characteristics of usable traces but are not sufficient to generate similar ones for replaying purpose. Most workload studies consider an intrinsic number of different behaviors while in reality there are as many behaviors as there are applications. The aimed number of behaviors is linked to the precision needed. The more classes of behavior, the more precise but also the more complex the models will be. To improve the precision of the resulting studies, the creation of models should be possible at different precision levels while limiting a priori bias.

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 a Google cluster [6] used for 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 a large scale HPC workload. Such workload is necessary to evaluate the schedulers and middlewares capabilities from several points of view, performance, energy and resilience. Indeed, the scale of the proposed models is necessary for these types of evaluation in order

* Corresponding author.
E-mail addresses: Georges.Da-Costa@irit.fr (G. Da Costa), Leo.Grange@irit.fr (L. Grange), Ines.De-Courchelle@irit.fr (I. de Courchelle).

Table 1
Distribution of number of events per task.

Number of events	1	2	3	4	5
Number of tasks	248	1,093,992	24,258,907	19,378	17,215

Number of events	6	7	8	9+	Total
Number of tasks	8510	5093	3028	18,360	25,424,731

Table 2
Distribution of finishing event for all tasks with three events.

Events type	Finish	Kill	Fail	Other
Number of tasks	17,775,284	6,381,906	86,348	15,369
Percentage	73.31%	26.31%	0.35%	0.03%

to exhibit particular behaviors such as failing servers or thermal effects with impact on servers energy consumption.

This article is structured as follows: 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. In the next section, we will focus on using automatic clustering to identify the possible classes of tasks in those traces. Before the conclusions and perspectives, a last section will propose a reusable workload generator.

2. Workload characteristics

The data used in this study and analyzed below come from one of the datacenters of Google. Google provided a dataset [6] of some of its servers usage in 2011. The monitored datacenter is composed of 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 of one or many tasks. The present article focuses on the *task_events* table. This record contains 500 CSV files, with information on 25 million tasks. This record encompasses 100 million 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 transition 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 by the management middleware or when a linked task died; Finally *Lost* is an event to characterize a task for which the ending reason is missing. Actually, most tasks finish with either *Finish* or *Fail* due to the limits of the monitoring infrastructure.

Table 3
Number of tasks and frequency for each scheduling class.

Scheduling class	0	1	2	3
Tasks number	20,317,398	3,327,283	533,330	49,614
Percentage	83.86%	13.73%	2.20%	0.21%

Considering the 500 CSV files which describe task events, an analysis shows that tasks are linked with several events. Table 1 describes 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 *Submit* one, a *Schedule* one and an ending event. Most other tasks (4.5%) are the ones which are submitted, scheduled but do not finish during the time frame of the acquired data. The remaining (<0.5%) are tasks that have intermediate events of reconfiguration. In the following of this article, tasks taken into account are the ones composed of three events.

The way the tasks are ending is also important. Table 2 describes the distribution of finishing states for all tasks with three events. 73% completed normally, 26% are killed (in the *Kill* state), and less than 1% finish in another state. As stated in the description of the workload by Google [6], nearly all tasks that do not finish normally are in the *Kill* state. No additional information allows to know the actual reason that caused the tasks to finish abnormally. Almost all the tasks (99.6%) are finishing in either *Finish* or *Kill* states, therefore we will focus only on them in the rest of the article.

The behavior of these two types of tasks are quite different. As an example, the total time of execution of the 73% of tasks with the *Finish* state are accumulating 13×10^9 s (with an average makespan of 1694 s) whereas the 26% of tasks with the *Kill* finish state are accumulating 20×10^9 s (with an average of 7994 s).

All tasks are not equals. Some tasks are production one and cannot be stopped or postponed, some can be postponed but not stopped such as accounting, and some statistical verification can be stopped but there is no more interest to restart them latter if they were postponed.

Schedulers and middlewares need information on the requirements of tasks in order to evaluate the quality of their decision. One important requirement is the impact of delay of 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 labeled *priority* and *scheduling class*. According to the Google's documentation [6], *priority* is used by the cluster's scheduler, whereas *scheduling class* is used locally by a machine to manage resource usage.

Table 3 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 time-critical tasks.

Priority level is a number between 0 and 11, with 0 as the lower priority. Table 4 provides the count of tasks per priority value.

Fig. 1 shows the average makespan for all the tasks with three events, grouped by priority level. The tasks with the highest priority level (9 and 10) have a longest average makespan. However, the tasks with a priority 3 have a longer average makespan than the others (except 9 and 10) but their presence is low in the dataset (1027 tasks with a priority 3).

Table 4
Number of tasks for each priority level.

Priority	0	1	2	3	4	5
Tasks	5,701,546	2,357,274	1,078,476	1027	13,975,078	104

Priority	6	7	8	9	10	11
Tasks	633,445	0	249,932	230,164	579	0

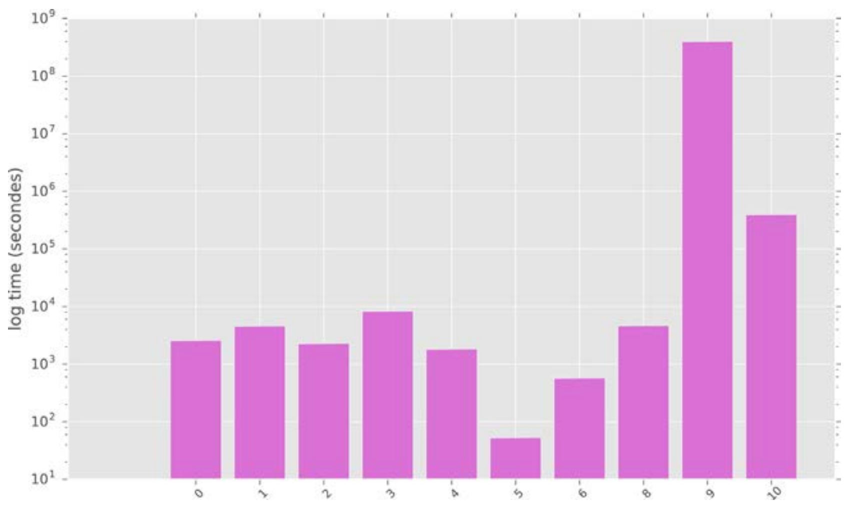

Fig. 1. Average task makespan for each priority.

Table 5
Number of tasks and frequencies for the four priority groups.

Priority group	Free	Normal	Production	Monitoring
Tasks number	8,058,820	15,938,062	230,164	579
Percentage	33.26%	65.78%	0.95%	0.002%

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 5 is obtained by grouping the data from Table 4 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.

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, Fig. 2 shows the statistical relationship between these. For each scheduling class, it shows 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 into a metric showing the quality of the allocation.

3. 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. This literature allows to know:

- the heterogeneity of the physical resource infrastructure of data centers,
- the resources requested by the tasks,
- the classification of the tasks that make up this workload.

In [7], 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 analysis, they have shown that 2% of tasks represent 80% of CPU, Memory and that 92% are long tasks with a *Free* priority. In [8], 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 analysis of the traces allows us to understand the Google dataset, the importance of an heterogeneous datacenter to adapt the resources to the demand. However, they did not provide information related on how to use this workload in another context.

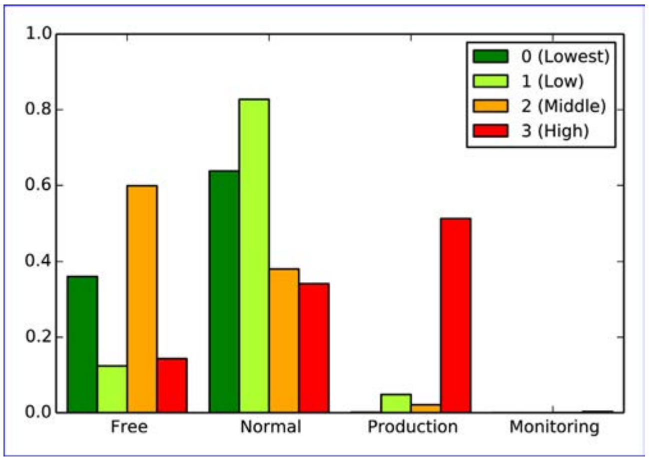

Fig. 2. Distribution of priority groups, for each scheduling class.

The paper [9] 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 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. [10] 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*

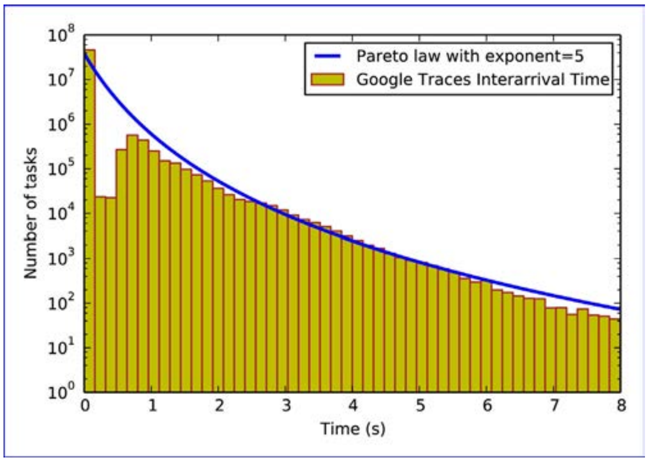

Fig. 3. Cumulative distribution of inter-arrival time (s) between tasks (log-scale).

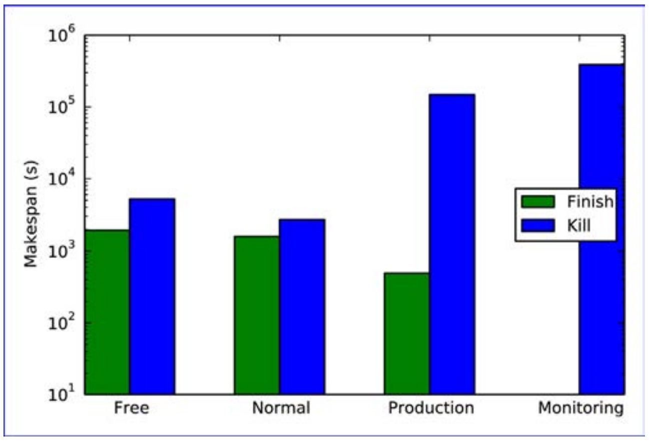

Fig. 4. Average makespan for each priority group, depending if the task finished by a *Finish* or a *Kill* event (log scale).

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 [11]. These approaches using k-means lead to difficulties to generalize the workload for producing new instance of similar workload.

In [12], 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 machine-related events on average each days. In this study, authors have clustered the machine population per same CPU and memory (15 groups). Contrary to [8], Liu and Cho [12] shown that the machines are almost homogeneous, 93% machines have the same CPU capacity and 86% of the machines have the same memory size. 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.

In [13], authors analyzed and elaborated the characteristics of the Google traces. They exposed the duration of execution, waiting time of jobs. Based on this analysis they present an Ensemble scheme Workload Prediction method. To explore the Google dataset, authors leveraged an Apache Hive infrastructure. They demonstrate that 80% of the jobs are shorter than 1000 s and that the average waiting time is less than 800 s except for the last day where it is less than 500 s. The prediction model is based on the correlation between the job event and the number of actives machines. Our work provide the implementation of a workload generator without being limited by the number of resources.

Di et al. [14] 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 min for the Google dataset. Other 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 [15], 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 application (joint ratio almost 10%). Di et al. used a k-means clustering algorithm based on task events and resource utilization to classify applications. The number of applications in the k-means clustering sets follows 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 publications show that, with the exception of the length of jobs, the behavior of tasks in the Google workload is similar to the ones seen in Grid/HPC systems.

These studies show that workload data is characterized (priority, latency sensitivity). The goals of these studies were mostly to understand and characterize the workload rather than to propose a complete set of models and their implementation for providing the ability to generate new similar workload. The next part presents the generic model for HPC workload along-with models for each of its characteristics.

4. 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.

4.1. Submission time

As described in the state of the art [7], tasks are submitted continuously on the monitored platform which is highly charged. Very few inter-arrival times are over 10 s, and the average is 0.052 s. The inter-arrival law is well modeled by a Pareto distribution, with a λ parameter set to 5 in the case of the Google Dataset as shown in Fig. 3.

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.

4.2. Type

As described in Section 2, most of the tasks end with one of the *Finish* or *Kill* event. In the first case, the task just finishes its work and ends normally. In the second case, the documentation is

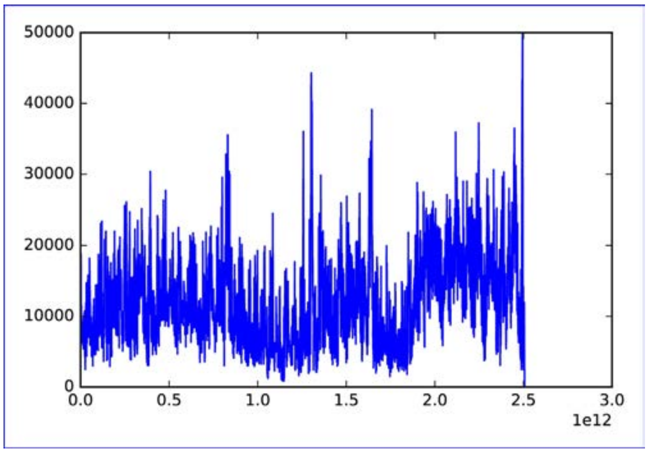

Fig. 5. Number of allocated tasks over the month for tasks ending with a *Finish* event.

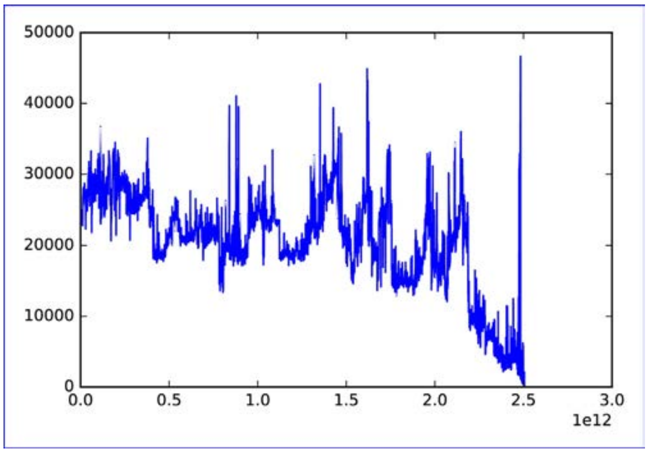

Fig. 6. Number of allocated tasks over the month for tasks ending with a *Killed* event.

unclear about which are the possible causes of a *Kill* event and the following behavior concerning the possible re-submission of the related tasks.

By analyzing these two sets, it appears they do not follow the same statistical properties. Fig. 4 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 increases, 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 s for finished group compared to 148,424 s for killed tasks.

Fig. 5 shows the effective mass of jobs ending with a *Finish* event. It shows the simultaneous running tasks over time that finish well. The mass follows a fractal structure coherent with the inter-arrival and makespan laws. It also demonstrate daily and weekly patterns. For long-term characterization of traces, such long-term patterns should be taken into account and modeled. Near the end of the monitoring there is a peak, with a high number of jobs. They are executed before the end of the experiment window and due to lack of information it is unknown if it comes from an actual peak of submitted jobs (similar to the ones in the middle of the time window) or if it is an artifact due to the end of the monitoring of the cluster.

Fig. 6 shows the effective mass of jobs ending by a *Kill* event. Contrary to the previous mass, the effective mass of executing jobs with a non-finish event follows a less regular curve. The average makespan of these tasks is higher than that of the *Finish* category and thus the mass toward the end of the monitoring window

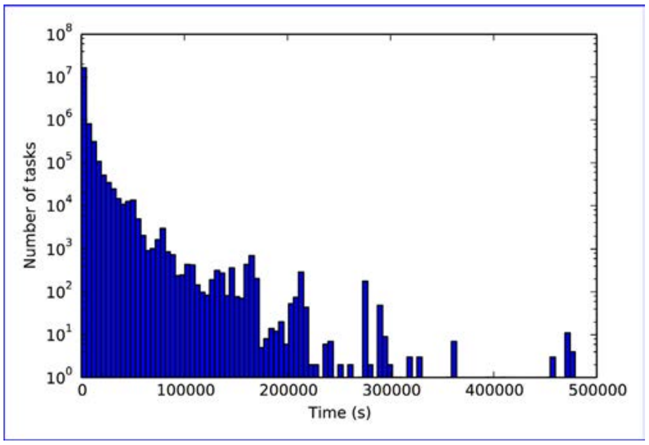

Fig. 7. Distribution of makespan for tasks finished normally (log scale).

decreases as these long-lasting jobs start finishing out of the monitoring window. Even with a lower number of tasks, this figure shows that the overall mass is higher due to these longer tasks. The high variability is less linked to daily and weekly timescale, as these jobs are more related to infrastructure services whereas the one ending with a *Finish* event are more often jobs submitted by actual users of the platform.

Overall, the average number of running tasks at a particular time is largely superior to the number of servers. It shows that most tasks use less than 100% of a server resources. Actually, other files in the Google Dataset contains information about processor and memory consumption over time for each job and show that these are often not using whole servers.

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 usage. This kind of service has 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 cannot 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 such a 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, has no precise amount of work to do, and therefore must run until stopped.

4.3. *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. [14]. As most other characteristics are similar, a method to have more HPC-like traces would be to increase the average makespan.

4.3.1. *Tasks*

In Fig. 7, 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 6 gives more precise statistical information. The shape of the distribution is confirmed by the gap between the median value

Table 6
Statistics for execution time of task finished by a Finish event.

Priority	Count	Average	Std. dev.	Median
Free	5,081,775	1937.9	4518.7	345.7
Normal	12,677,000	1600.3	5231.4	492.6
Production	49,190	493.9	1955.1	39.1
Monitoring	0	NA	NA	NA
All	17,807,965	1693.6	5034.5	448.9

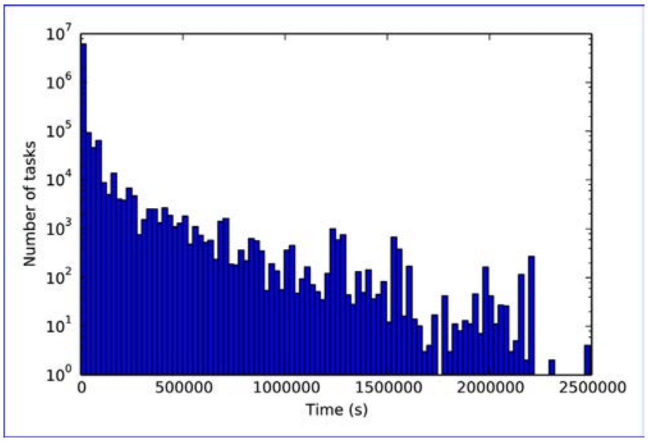

Fig. 8. Distribution of execution time before the *Kill* event for killed tasks.

Table 7
Statistics for execution time of tasks finished by a Kill event.

Priority	Count	Average	Std. dev.	Median
Free	2,977,045	5195.6	24,336.4	595.4
Normal	3,261,062	2688.5	19,468.9	213.9
Production	180,974	148,424.4	294,182.3	31,429.4
Monitoring	579	388,232.5	579,101.9	71,831.6
All	6,419,660	7994.3	59,363.8	333.8

and the average. Standard deviation is also important compared to average, another property of long-tailed distributions.

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. The parameters of the fitted log-normal distribution are $\sigma = 1.42$ and $\mu = 6.25$, with a KS statistic value of 0.057. By looking at the KS statistic, the better model to describe this characteristic is the log-normal law.

4.3.2. Services

Fig. 8, similarly, shows the same kind of distribution, with even longer makespans. By comparing Table 7 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.

4.4. 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.

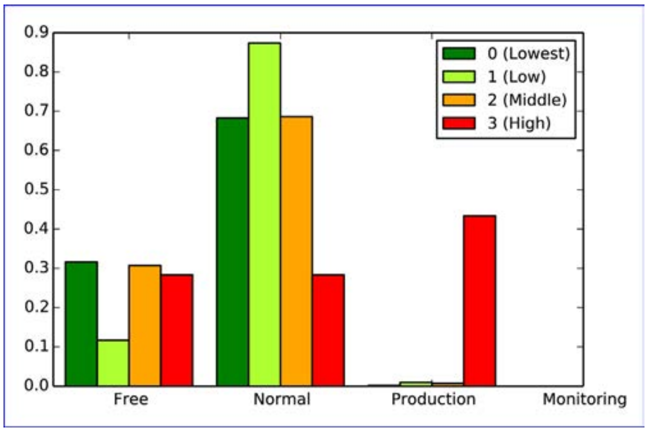

Fig. 9. Distribution of priority groups, for each scheduling class, among the task finished by a *Finish* event.

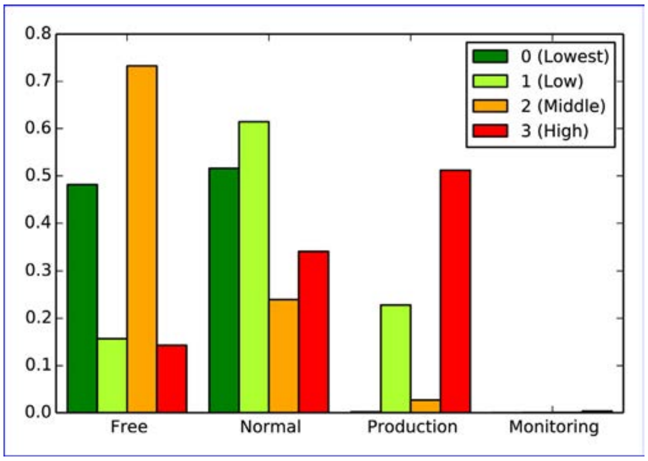

Fig. 10. Distribution of priority groups, for each scheduling class, among the task finished by a *Kill* event.

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.

4.4.1. Tasks

For tasks, as shown in Fig. 9, there is a strong relationship 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 λ parameter.

4.4.2. Services

For services, as shown in Fig. 10, the relationship between priority and scheduling class is less visible but present. A simple model would be also a truncated exponential law, with a lowest λ value compared to the tasks priority model.

4.5. Comparison with other workloads

Such an analysis of this single dataset can only allow us to model the workload for this specific Google's cluster. However, several reasons give us confidence that our model can be used to describe workloads run by other cloud and grid clusters.

Those traces are known to contain multiple and heterogeneous kinds of applications. This is also the case of datacenters used by

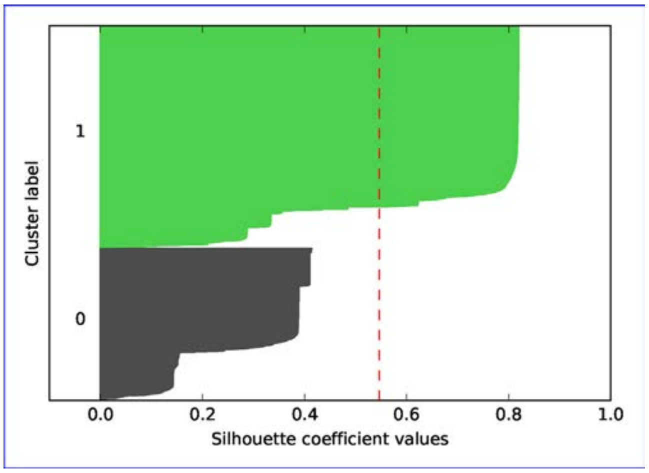

Fig. 11. Silhouette analysis of a 2-clustering of tasks using k-means.

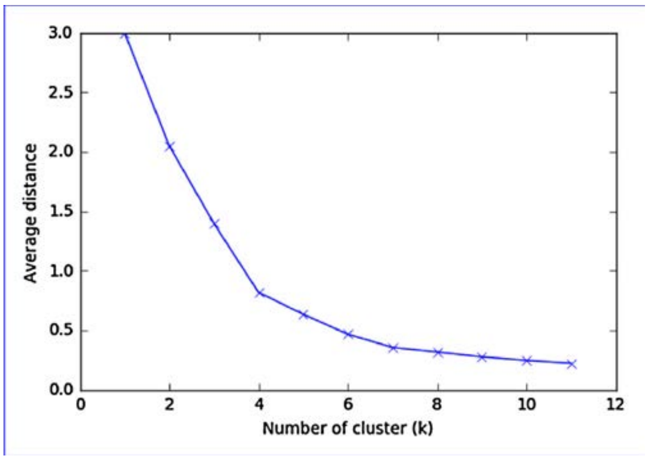

Fig. 12. Variation of the average distance to the closest centroid after a k-mean, for different values of *k*.

Table 8
Centroids and analysis of 2-clustering.

Cluster	Makespan	Priority	Sched. class	Frequency	Disparity
0	3508	0.51	0.72	40.5%	17.2
1	2456	4.1	0.11	59.5%	6.2

Table 9
Centroids and analysis of 4-clustering.

Cluster	Makespan	Priority	Sched. class	Frequency	Disparity
0	1911	0.9	1.64	17.9%	19.8
1	2482	0.53	0	26.2%	6.2
2	1662	4.2	0.1	55.2%	4.4
3	137,630	3.39	0.96	0.7%	1.4

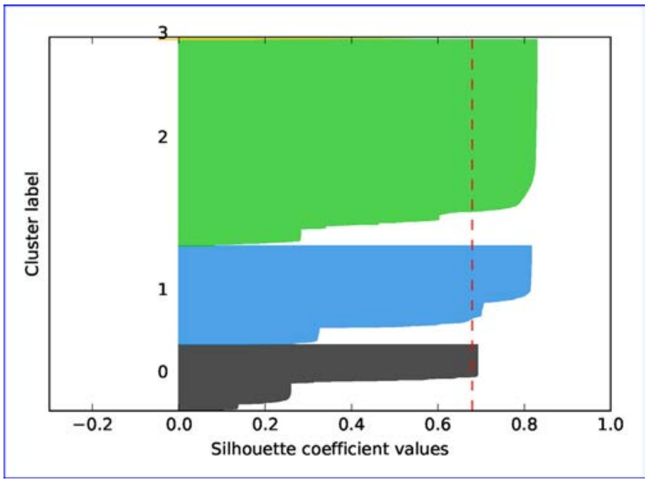

Fig. 13. Silhouette analysis of a 4-clustering of tasks using k-means. The cluster labeled 3 contains less than 1% of the tasks and is therefore difficult to see here.

most IaaS and PaaS cloud providers, and we think this diversity gives good insights on their typical workloads.

For datacenters running more homogeneous applications, such as HPC jobs, Di et al. have done detailed comparisons between the traces from Google and those provided by several grid infrastructures [14]. They found some differences, such as the lack of periodic patterns in tasks submission over time for the traces we studied here, compared to the grid ones. Particularly, diurnal submission pattern is a common characteristic of datacenter workloads, and may be added to our model to fit better to a typical grid workload.

Di et al. also discovered several similarities which make our model relevant for grid computing. Notably, distribution of makespan have a long-tailed shape in every workload they studied, which is a key characteristic of Pareto and log-normal laws. They observed differences, such as a smaller average makespan, a higher maximum and more short tasks in the Google traces compared to the grid ones. Those characteristics can be obtained, using our model, by adjusting the parameters of the probability distribution functions.

In [16], Kavulya et al. analyzed the traces of a production Hadoop cluster owned by Yahoo!. They found the same kind of distribution in the Hadoop jobs completion time, and 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.

5. Automatic classification

By assuming two main categories, *tasks* and *services*, we worked on a priori classification. It allowed us to establish a preliminary analysis on the dataset and to understand some of its characteristics. However, this classification may seem artificial, and it may be interesting to compare it to the result of an automatic classification.

Some previous works already studied those traces with clustering methods [17,10], but were focused on different properties than the ones we covered in this paper. We used a k-means algorithm to classify all the tasks (ending either by a *finish* or *kill* event) with respect to three properties: the makespan, the priority and the scheduling class.

5.1. 2-Clustering

In the case of our initial assumption, we had only 2 classes. Therefore, it is interesting to study the results of an automatic 2-clustering on the same dataset.

The results of the 2-clustering are summarized in Table 8. In this table, the frequency is the part of the tasks belonging to a cluster, and the *disparity* is the ratio between the average makespan and the median one for a given cluster. The first category seems to be relatively long tasks, with very low priority. The second one is centered on medium execution time, but higher priority. Moreover, this second class represents around 60% of the total amount of tasks, confirming the high quantity of small tasks in these traces.

With $k = 2$, the clustering is far to be perfect. Fig. 11 shows the silhouette plot using such clustering. With an average silhouette width of 0.55, we can deduce that those two categories are relevant, without being very accurate. The cluster 0, in particular, gives relatively bad results, probably because it is too wide.

5.2. *General k-clustering*

In order to know which value of k is the most relevant for this dataset, different values were tested. Fig. 12 gives the average distance, for each of the normalized data, to the closest centroid for those different k. The elbow method [18] is quite difficult to apply here due to the lack of a clear angle. However, we chose $k = 4$ as a reasonable value: it is still possible to decrease the average distance by using a higher value, but it is a good compromise between number of classes and accuracy.

The results of clustering using 4-means are given in Table 9. Among the 4 different classes of tasks, the clusters 0 and 1 seems associated to medium tasks with low priority. The first one contains shortest tasks with relatively high scheduling class value, whereas the second one contains longer tasks with very low scheduling class. The cluster labeled 2 contains shorter tasks in average, but with high priority, which still represent more than the half of the total amount of tasks. Finally, the cluster number 3 is quite interesting, despite the very low proportion of tasks (0.7%) it represents. It consists in the longest tasks (an average of 38 h), which also have a high priority.

According to the silhouette plot, given in Fig. 13, this clustering seems quite accurate, having an average silhouette width of 0.68. The different classes seem to fit decently the characteristics of the tasks that belong to them, at the exception of the cluster 3, corresponding to the very long tasks.

Our initial, a priori, assumption of the existence of two distinct population of tasks and services in those traces may be questioned. However, some of the results obtained by automatic clustering tends to confirm this hypothesis, like the small group of very long tasks in 4-clustering. Moreover, the results confirm that using only 2 categories leads to a decent clustering.

6. Generating workload

The law described in Section 4 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 is one main characteristics needed to define a particular workload: *Dynamism*

- **Dynamism**: Represents the dynamism of task submissions, as the average duration between two consecutive tasks.

  Three other characteristics are to be specified for each category (tasks and services) : *Frequency*, *Mass* and *Disparity*.
- **Frequency**: Is a value between 0 and 1 representing the percentage of tasks of this category in the total workload. The sum of the frequencies for the different categories must be equals to 1.
- **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 their *frequency*, and 0.3 for the services.

The *mass* and *disparity* para meters are calculated using the average and median values from Tables 6 and Table 7. 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 14 in the Python3 language. In addition to the workloads laws from Section 4, it is also possible to use this algorithm with parameters to generate a workload similar to the k-clustering analysis from Section 5. The values of the parameters can be directly retrieved from Tables 8 and 9.

Algorithm. Reference implementation of tasks generator implemented in 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, ratio, names,
                  masses, disparities, priorities):
  """
  returns the next event (timestamp, name of the event, priority
                          of the event, makespan of the event)
  input:
    timestampLastEvent: reference time of the last event
    dynamism: density of simultaneous tasks
    ratio: list of weights for each type of task
    names: list of names for each type of task
    masses: list of masses for each type of task
    disparities: list of mass variations for each type of task
    priorities: list of priorities for each type
  """
  arrival = scipy.stats.pareto.rvs(4, loc=-1) * 3.0 * dynamism
  newTimestamp = timestampLastEvent + arrival
  typeIndex = random.choices(list(range(len(ratio))), weights=ratio)[0]
  name = names[typeIndex]
  priority = truncated_expon(priorities[typeIndex])
  makespan = get_makespan(masses[typeIndex], disparities[typeIndex])

  return (newTimestamp, name, priority, makespan)
```

The parameters we are using are important characteristics of a workload with intuitive enough meaning (average makespan and their disparity, average inter-arrival time and frequency of each task group). Therefore, it is also easy to use the proposed workload generator to create a workload from scratch with specific characteristics, for instance for test purposes.

7. 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) realistic data from a Google internal datacenter. The second contribution is an actual implementation of the workload generator with possibilities to adapt to several use cases. The proposed categories (services and tasks) are backed with elements from the evaluation of different possible classifications using k-means methods. The model and generator are adapted to a generic number of categories which can be updated considering the aimed precision of the Workload generator. 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. This generator can help us to improve the scalability of large datacenter to provide an adaptable workload for a green datacenter. This will help us to understand the energy flows and adapt the workload to improve efficiency energy. In the future, we plan to augment the generator with more diverse possibilities such as diagram of tasks, constrained resources or MPI tasks. The next step will be to use it in large scales simulators in order to evaluate scheduling policies. In order to increase precision for long-term analysis, daily-, weekly- and seasonal-patterns will be also necessary.

Finally, another important step will be to generalize the proposed laws using several other datasets such as the one of the Grid Workload Archive (http://gwa.ewi.tudelft.nl/) and to provide relevant parameters to take advantage of these laws.

Acknowledgments

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.

References

[1] K. Krauter, R. Buyya, M. Maheswaran, A taxonomy and survey of grid resource management systems for distributed computing, Softw. Pract. Exp. 32 (2) (2002) 135–164.
[2] H. Wang, J.X. Zhang, B. Yang, F. Li, On time-sensitive revenue management in green data centers, Sustain. Comput. Inform. Syst. 14 (2017) 1–12.
[3] T. Le, D. Wright, Scheduling workloads in a network of datacentres to reduce electricity cost and carbon footprint, Sustain. Comput. Inform. Syst. 5 (2015) 31–40.
[4] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, D.H. Epema, The grid workloads archive, Future Gen. Comput. Syst. 24 (7) (2008) 672–686.
[5] G.D. Costa, M.D. Dikaiakos, S. Orlando, Nine months in the life of EGEE: a look from the south, 2007 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (2007) 281–287, http://dx.doi.org/10.1109/MASCOTS.2007.43.

[6] C. Reiss, J. Wilkes, J.L. Hellerstein, Google Cluster-Usage Traces: Format + Schema, Technical Report, Google Inc., Mountain View, CA, USA, 2011, Revised 2012. Posted at http://code.google.com/p/googleclusterdata/wiki/TraceVersion2.

[7] C. Reiss, A. Tumanov, G.R. Ganger, R.H. Katz, M.A. Kozuch, Towards Understanding Heterogeneous Clouds at Scale: Google Trace Analysis, Technical Report, Intel Science and Technology Center for Cloud Computing, 2012, pp. 84.

[8] C. Reiss, A. Tumanov, G.R. Ganger, R.H. Katz, M.A. Kozuch, Heterogeneity and dynamicity of clouds at scale: Google trace analysis, in: Proceedings of the Third ACM Symposium on Cloud Computing, ACM, 2012, pp. 7.

[9] F. Gbaguidi, S. Boumerdassi, É. Renault, E. Ezin, Characterizing servers workload in cloud datacenters, in: 2015 3rd International Conference on Future Internet of Things and Cloud (FiCloud), IEEE, 2015, pp. 657–661.

[10] M. Alam, K.A. Shakil, S. Sethi, Analysis and Clustering of Workload in Google Cluster Trace Based on Resource Usage, 2015 arXiv:1501.01426.

[11] Y. Chen, A.S. Ganapathi, R. Griffith, R.H. Katz, Analysis and Lessons from a Publicly Available Google Cluster Trace, Technical Report, UCB/EECS-2010-95 94, EECS Department, University of California, Berkeley, 2010.

[12] Z. Liu, S. Cho, Characterizing machines and workloads on a Google cluster, in: 2012 41st International Conference on Parallel Processing Workshops, IEEE, 2012, pp. 397–403.

[13] B. Liu, Y. Lin, Y. Chen, Quantitative workload analysis and prediction using Google cluster traces, in: Computer Communications Workshops (INFOCOM WKSHPS), IEEE, 2016, pp. 935–940.

[14] S. Di, D. Kondo, W. Cirne, Characterization and comparison of cloud versus grid workloads, in: 2012 IEEE International Conference on Cluster Computing, IEEE, 2012, pp. 230–238.

[15] S. Di, D. Kondo, F. Cappello, Characterizing cloud applications on a Google data center, in: 2013 42nd International Conference on Parallel Processing, IEEE, 2013, pp. 468–473.

[16] S. Kavulya, J. Tan, R. Gandhi, P. Narasimhan, An Analysis of Traces from a Production MapReduce Cluster, 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid) (2010) 94–103, http://dx.doi.org/10.1109/CCGRID.2010.112.

[17] I.S. Moreno, P. Garraghan, P. Townend, J. Xu, Analysis, modeling and simulation of workload patterns in a large-scale utility cloud, IEEE Trans. Cloud Comput. 2 (2) (2014) 208–221, http://dx.doi.org/10.1109/TCC.2014.2314661.

[18] R. Tibshirani, G. Walther, T. Hastie, Estimating the number of clusters in a data set via the gap statistic, J. R. Stat. Soc. Ser. B Stat. Methodol. 63 (2) (2001) 411–423.