A Characterisation Of The Workload On An Engineering Design Grid

Andrew Burkimsher, Iain Bate, Leandro Soares Indrusiak
Department of Computer Science, Deramore Lane, University of York, Heslington, York, YO10 5GH, UK
{amb502, iain.bate, leandro.indrusiak}@york.ac.uk

Keywords: Workload Characterisation, Workload Generation, Grid Computing, Engineering Design, Dependencies

Abstract

Computer-aided engineering design uses simulations to explore a design space and identify promising regions. The hierarchical structure of the engineering design process suggests distinctive workload patterns that we believe are common in industry, yet have been little captured by previous characterisations. Selecting a scheduling policy that is ill-suited to the workload it serves may lead to poor performance. This paper characterises the workload run on the private grid of a large aircraft manufacturer over a period of 30 months. Cycles of load at daily and weekly scales are observed, and there are extended periods where the grid operates at saturation and work must queue. A method of creating workloads containing such cycles at a given total load percentage by adjusting inter-arrival times is given. Task execution times are demonstrated to be distributed in a log-uniform way, and an algorithm is given to generate execution times following this distribution. Graphs of dependencies between tasks are shown to have variation in node degree greater than that of random graphs, and a process of constructing dependency graphs following the observed distribution is described.

1. Introduction

High-Performance Computing systems (*HPCs*) made up of a large number of parallel processors have become ever more popular in recent years, due to their ability to provide significant computing capacity at a relatively low cost. Where a single HPC cluster cannot satisfy an organisation's desire for computing power, geographically-distributed networks of such clusters have been created, and these are known as *grids*.

The activities undertaken on grids support industrial operation, and the grids' performance is key to enabling the productivity of their users. Grid scheduling to meet this performance demands is an area of much research. The selection of the most appropriate scheduling policy, however, can only be made in the context of the workload run and the requirements of users. An inappropriate or poorly tuned scheduling policy can cause performance to suffer. Proper evaluation of scheduling policies therefore requires appropriate workload characterisations [1].

Computer-aided engineering design is a growing field, as more computational capacity becomes available and simulation tools become more refined. The engineering design process follows the hierarchical decomposition model - the classic divide-and-conquer pattern. The composition of work leads to simulation being necessary at all levels of detail - from small simulations for small problems, to large-scale simulations for validation of the final compositions. We believe that workloads similar to the one studied are necessary across all kinds of engineering problems.

Workloads with a wide variation in execution times can pose particular problems for classes of scheduler that do not consider execution times [2]. This is because it is not usually desirable for a task that takes a few minutes of execution time to ever queue behind one that takes weeks or even months.

This paper will characterise the workload run on the private grid of an engineering corporation whose primary business is that of the design and manufacture of aircraft. In order to meet the need for quicker turnaround times than are available from wind tunnels, a large amount of early-stage design now takes place with CFD simulations. This corporation engaged the authors to study the workload placed on their grid, with the ultimate aim of improving the performance of the workflows that execute on the grid. Logs were provided for a period of 30 months up until the end of August 2012.

To generate realistic workloads, several aspects are required. This paper considers the inter-arrival times and size of tasks along with the internal dependency structure of jobs (collections of tasks). The size of tasks is shown to follow a log-uniform pattern. The rate of task arrival is shown to follow daily, weekly and seasonal patterns. The grid spends significant periods of time in a saturated state. The internal dependency graphs have a high variation in node degrees.

The paper will summarise related work. A detailed characterisation of the workload is undertaken, based on log files obtained from the partner, including submission patterns (Section 2.) and execution volumes (Section 3.). The structure of dependencies within the workload is presented (Section 4.). In some figures, the scales on the axes have been obscured to protect the interests of the industrial partner, although this does not influence the trends and distributions observed. Algorithms to generate synthetic workloads matching the observed distributions and structures are presented in each section.

1.1. Related Work

Although workload characterisations of web services and research-oriented grids are well-represented in the literature, there is little in the way of characterisations for engineering design workloads. Some recent examples of web service characterisations include [3, 4]. However, web services tend to run well below their maximum possible utilisation, so as to have the capacity to scale up with peaks in traffic. An overall utilisation of just 6% is noted by [3], 5-10% by [5], while average of 50%, rising to 70% at peak was observed by [4]. Utilisations this low pose little challenge to a scheduler, as tasks can run immediately. Workload characterisations also exist for research-oriented grids, where [6] and [7] observed utilisation between 90 and 100%, where the implication is that tasks usually queue for some time (or *pend*) before execution. Where tasks queue and the grid is saturated, the scheduling policy to manage the queue becomes important.

The management of the queue is especially important in where there is a wide variation in runtimes. [6] and [1] observed that many workloads have a large number of small tasks that contribute only a small fraction of the load. Conversely, only the small proportion of large tasks contribute the bulk of the load. Effective prioritisation by the scheduler is required to keep the system responsive for the smallest tasks but also to avoid starvation for the largest ones.

The patterns in arrival times over working days and weeks were noted by [6], [1] and [7]. It comes as no surprise that the peak of task submissions appear during normal working hours. This feature of academic grids is in contrast to the web workloads, where peaks appear before and after usual working hours [4]. [6] had a roughly even distribution of tasks between the working days, whereas [7] had a high peak on Mondays, which decreased during the week.

Much work has been performed on how to schedule in the presence of dependencies between tasks on a grid [8] and how they can be modelled using Directed Acyclic Graphs [9]. However, the difficulty of scheduling DAGs on a grid depends significantly on the structure of dependencies within such graphs and the opportunity or otherwise of extracting parallelism. Little work seems to be have been performed on characterising the dependency structures within grid workloads. Examples of structured graph topologies were given by [10], but these looked at the internal structure of algorithms, rather than that of workflows. In Section 4., we present dependency graphs from the industrial workload and characterise them using graph-theoretic metrics.

2. Working pattern of designers

The users who place the vast majority of the load on the grid are the designers. These people work in a way which could be considered typical for an engineering group. The staff in the design team follow many natural rhythms in their work. This section will describe the rhythms found in the submission and execution of work on the partner's industrial grid. The figures throughout this paper were created using the data from the 30 months of log files.

2.1. Submission Cycles

The most basic cycle is that related to the working day. Figure 1 displays the number of tasks submitted per 15-minute block throughout the day, averaged over the whole workload sample. The highest rate of submission is during working hours (08:00-17:00). There is a distinct drop in the middle of the day when workers break for lunch.

The pattern observed fits with similar observations of such daily rhythms in [11, 12, 3]. It is natural that the lowest level of work submission is overnight, when most workers are sleeping. There is a baseline level of work submissions even when no-one is at work, as the result of automated scripts. As the users only work Monday-Friday, submission peaks only appear on those days, in contrast to the observations of [3].

The rate of task arrival per hour can be normalised to a probability mass function (*pmf*) by dividing the counts by the total number of tasks submitted. Table 1b gives the pmf values for arrivals on each day of the week. There were too many samples made on the time of day to give the pmf value for each time, so instead, the coefficients of a polynomial fitted using the least-squares method are given in Table 1a. Counts were grouped into 15-minute bins over 24 hours, giving 96 samples for the mass function.

The results returned by the jobs with the smallest execution times (minutes up to a few hours) can usually be analysed by the users within the same day, if they arrive in time. However, many results take longer to calculate, so designers follow a daily design cycle instead. They tend to expect results to be ready when they arrive at work. They then analyse results and work on new designs during the day before submitting more simulations at the end of the working day.

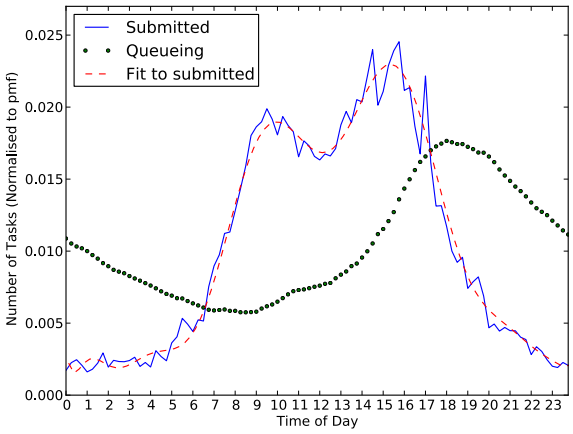


Figure 1: Daily Submissions and Queueing

for $0 \leq y < 24$, $y =$	
$3.41 \cdot 10^{-3}$	$+1.88x^{8} \cdot 10^{-5}$
$-1.29x \cdot 10^{-2}$	$-1.34x^{9} \cdot 10^{-6}$
$+3.06x^{2} \cdot 10^{-2}$	$+6.62x^{10} \cdot 10^{-8}$
$-3.13x^{3} \cdot 10^{-2}$	$-2.24x^{11} \cdot 10^{-9}$
$+1.73x^{4} \cdot 10^{-2}$	$+4.93x^{12} \cdot 10^{-11}$
$-5.82x^{5} \cdot 10^{-3}$	$-6.38x^{13} \cdot 10^{-13}$
$+1.26x^{6} \cdot 10^{-3}$	$+3.67x^{14} \cdot 10^{-15}$

(as shown in Figure 1)

(a) Daily

Mon	0.167
Tue	0.191
Wed	0.192
Thu	0.198
Fri	0.187
Sat	0.042
Sun	0.012

(b) Weekly

Table 1: Probability Mass Functions for Submission Rates

Achieving responsiveness to serve these cycle times is paramount, because jobs that are 'late' affect the productivity of designers. The stages of aircraft design usually have fixed time budgets that the designers have to work to. The quality of a design is usually determined by the number of iterations the designers can perform within the given time frame.

Most of the time, more work is submitted during working hours than can be processed immediately. Instead, work queues up during the day and this queue is drawn down overnight (Figure 1). This build up of work also happens over the scale of a week (Figure 2, averaged over log duration), where the queue length increases during the week, and is drawn down again at the weekend. From this, it can be seen that the grid spends a significant proportion of its time in a saturated utilisation state.

Submission Cycle Generation

These distinct patterns of variation in submissions pose significant challenges to schedulers, especially when the load is high enough to lead to periods of platform saturation. In order to properly evaluate scheduling policies, it is necessary to generate workloads that follow these patterns.

Previous work by [13] gave a method of adjusting the target load on a cluster by varying the inter-arrival times of jobs. This approach is advantageous, because it allows schedulers to be evaluated with the same workload at different loading levels. However, the

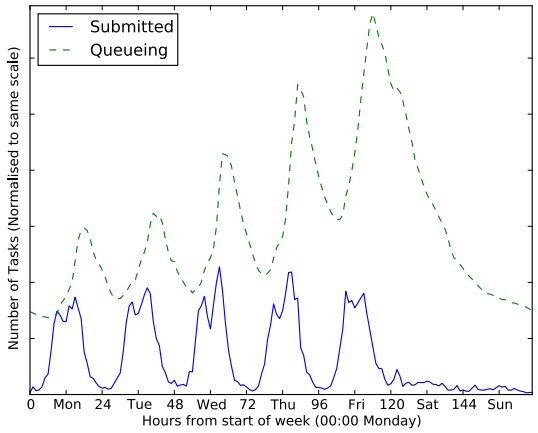


Figure 2: Weekly Submissions and Queuing

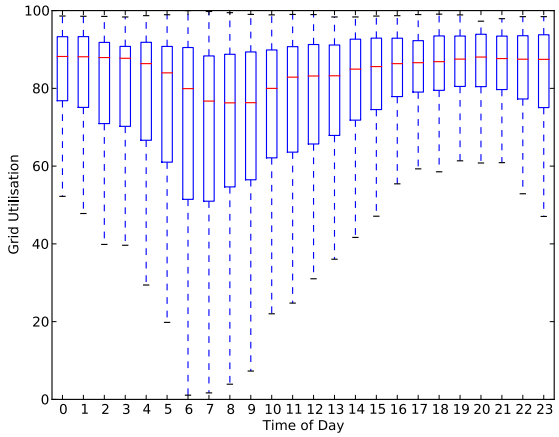


Figure 3: Daily Utilisation

Algorithm 1 To generate submission patterns by changing inter-arrival times

Symbol	Parameter
c	Number of processing cores in the system
l	Desired loading factor (full load = 1)
j	Array of all jobs in workload
j^i_{exec}	Total load (core-seconds) of Job j^i
j^i_{sub}	Submit time of Job j^i
$\text{pmf}_{\text{day}}(h)$	Probability mass function of arrivals over a day (by hour), such as in Table 1a
$\text{pmf}_{\text{week}}(d)$	Probability mass function of arrivals over a week (by day), such as in Table 1b

$\texttt{set\_arrival\_times}(\text{c, l, j, num\_samples}):$
$\texttt{id}, \texttt{binid}, \texttt{last\_fill}, \texttt{last\_sub}, \texttt{time\_increase} = 0$
$\textbf{for}\ \text{day}\ \textbf{in}\ 0..6:$
  $\textbf{for}\ \texttt{smp}\ \textbf{in}\ 0..\texttt{num\_samples}:$
    $\texttt{min\_bin}[\texttt{id}] = 60 \cdot 24 \cdot 7 \cdot \text{pmf}_{\text{day}}(\text{day}) \cdot \text{pmf}_{\texttt{week}}(\texttt{smp})$
    $\texttt{id} = \texttt{id} + 1$
$\textbf{for}\ j^i\ \textbf{in}\ j:$
  $\texttt{newmins} = \frac{j^i_{\text{exec}}}{c \cdot l}$
  $\texttt{binfill} = \texttt{last\_fill} + \texttt{newmins}$
  $\textbf{if}\ \texttt{binfill} < \texttt{min\_bin}[\texttt{binid}]:$
    $\texttt{last\_fill} = \texttt{binfill}$
    $\texttt{time\_increase} = \frac{60}{\texttt{num\_samples}} \cdot \frac{\texttt{newmins}}{\texttt{min\_bin}[\texttt{binid}]}$
  $\textbf{else}:$
    $\texttt{acc} = 0$
    $\textbf{while}\ \texttt{binfill} \geq \texttt{min\_bin}[\texttt{binid}]:$
      $\texttt{acc} = \texttt{acc} + \frac{60}{\texttt{num\_samples}} \cdot \frac{\texttt{min\_bin}[\texttt{binid}] - \texttt{last\_fill}}{\texttt{min\_bin}[\texttt{binid}]}$
      $\texttt{last\_fill} = 0$
      $\texttt{binfill} = \texttt{binfill} - \texttt{min\_bin}[\texttt{binid}]$
      $\texttt{binid} = (\texttt{binid} + 1) \bmod \texttt{id}$
    $\texttt{last\_fill} = \texttt{binfill}$
    $\texttt{time\_increase} = \texttt{acc} + \frac{60}{\texttt{num\_samples}} \cdot \frac{\texttt{binfill}}{\texttt{min\_bin}[\texttt{binid}]}$
  $\texttt{last\_sub} = \texttt{last\_sub} + \texttt{time\_increase}$
  $j^i_{\text{submit}} = \texttt{last\_sub}$

algorithm they suggested is limited to giving a constant-load arrival rate. The patterns we have demonstrated here would be poorly modelled by such an approach.

However, their approach can be extended in order to model the arrival patterns over a working day and a working week. Their approach for load levels involves calculating the arrival time of the next job in a generated workload, and increasing or decreasing this time dependent on the desired level of load. Algorithm 1 presents a method that sets arrival times for every job in a workload by using probability mass functions (pmf) for the time of day and day of week. It works by calculating what the next arrival time would be if the current job could be perfectly parallelised across the whole grid. This new time point is then scaled on the load level desired and the pmf of the daily and weekly load distributions.

2.2. Grid Utilisation Cycles

In this section, we investigate the utilisation of the grid over the course of a day and week. This is done using distributions of utilisation for each hour or day encountered in the logs. In calculating the utilisation, only the fraction of time used by any task running within that hour or day was counted.

Figure 3 shows the distribution of utilisation of the cluster cores by time of day. This chart shows the utilisation of all cores in the grid, including those of specialised architectures. Above about 80%, some work will be almost certainly be queueing, because it is limited as to which cluster or architecture it can run on. When a multicore task heads the queue, the current scheduling policy waits until sufficient cores are free before starting the task. This means that full utilisation of the grid is almost unachievable and that tasks will be queuing well below 100% utilisation on some clusters.

There is significant variation because of the large number of days that were sampled. However, the variance decreases at the end of the day, showing how cluster utilisation rises to saturation at the end of every working day. The work submitted each day is only caught up on overnight, reinforcing the impression from Figure 1. The lowest point of utilisation tends to be around the time people arrive at work in the morning, when work has been caught up on overnight. This is in distinct contrast to [12], who observed

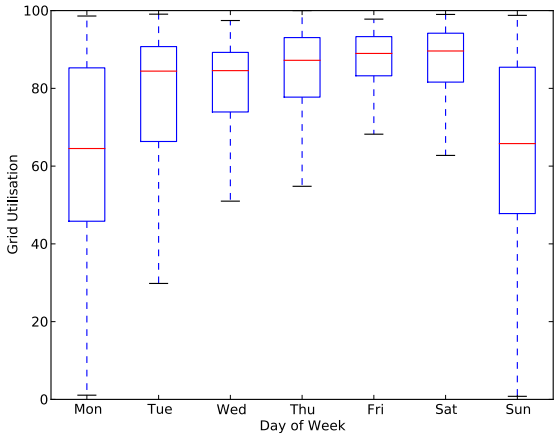


Figure 4: Weekly Utilisation

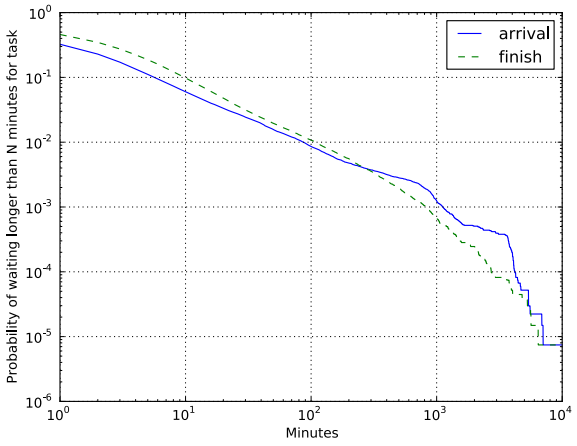


Figure 5: Inter-arrival & inter-finish time probabilities

peak utilisations of around 30%, or [5] with a utilisation of 10%.

This cycle of queue buildup during working ours is manifest on a weekly basis as well (Figure 4). Only Sunday and Monday have median utilisations much below saturation point. During the week, the average utilisation increases as more work is submitted during each working day than can be processed by the next day (corroborated by the average queue length in Figure 2). Monday has somewhat lower average utilisation because the most likely times for the grid to have any idle time is before the staff arrive on Monday morning.

In such a sizeable grid, tasks will be arriving and finishing at a fairly high rate. Figure 5 shows the probability of having to wait longer than a certain number of minutes for a task to arrive or finish. Because of the high variability in arrival rates, sometimes the arrival rates are very high. This is why the the probability of having to wait a long time for the next arrival of a task is low. Above about 120 minutes, the daily, weekly and seasonal cycles mean that there is more variability in the arrival rate, giving a higher probability of waiting longer for the next task to arrive than finish.

The finish rate of tasks is more constant, which is why the probability of a task finishing in a given time is lower under about 120 minutes. However, the probability of finish is still remarkably high, with a 10% chance of waiting longer than 10 minutes for the next task to finish, to 0.1% for 1000 minutes. Beyond 1000 minutes, the lines become aliased because of the very few occurrences of there ever being wait periods this long.

In summary, the findings of this section are that work is submitted in cycles following the rhythm of working hours. During working hours, work is submitted faster than it can be processed, and queue length increases accordingly, only being drawn down outside of working hours. The grid therefore spends a great deal of its time in a saturated state. Due to the large volumes of work passing through, the inter-arrival and inter-finish times of work are low.

3. Workload Composition

Engineering designs are made by hierarchically decomposing the problem into small parts, and then composing the completed designs until a final, complete design is reached. Early stage designs require low-fidelity and so need only a small amount of computation time for each CFD simulation. However, these are iterated over quickly (up to several iterations per day) and so there are many small tasks. As designs progress, the models considered get more complicated and require greater fidelity. This naturally requires more compute time for simulations. The largest jobs used for certification of an entire aircraft in full fidelity are very compute-intensive, and may need to execute over many months.

The characterisation in this section demonstrates that the hierarchical composition of the design process gives a workload that follows exponentially-distributed patterns. Notably, this is in contrast to previous research that has suggested alternative distributions of work found in large-scale grid systems [5], who observed a log-normal distribution.

3.1. Volume

Figure 6 shows the execution times of all the tasks in the 2.5-year workload, sorted by execution time. For multicore tasks, the execution time is given multiplied by the number of cores used. The striking feature of the graph is the straightness of the line, when the task size is plotted on a logarithmic scale. This can be compared to the linear fit obtained using least-squares. This suggests that the distribution of execution times follows a log-uniform distribution, at least between 10^1 and 10^5 core-minutes. This suggests that there are a roughly similar number of large and small tasks, with the median task execution time being approximately 1000 core-minutes.

There are few jobs that take less than 10 core-minutes of computing time, likely because jobs this small can be run on a local PC. The flattening of the slope in the middle of the curve indicates a particular peak of tasks around 10^3 core-minutes. This is likely to show the peak of jobs submitted where the results are needed within the same day for fast iteration.

An alternative view of this data is through a logarithmic histogram of the tasks' execution times, shown in Figure 7. Here, the uniform nature of the distribution is still apparent, at least between 10^1 and 10^5 core-minutes. In this view of the data, three distinct peaks of work are apparent. The first peak, centered on 10^1 core-minutes is likely to correspond to small tasks used for system

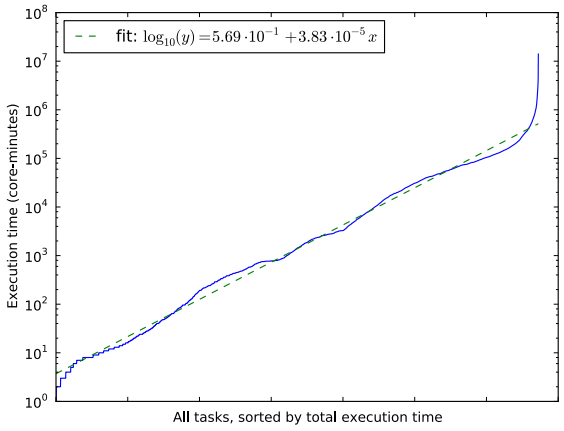


Figure 6: Task Volume Distribution

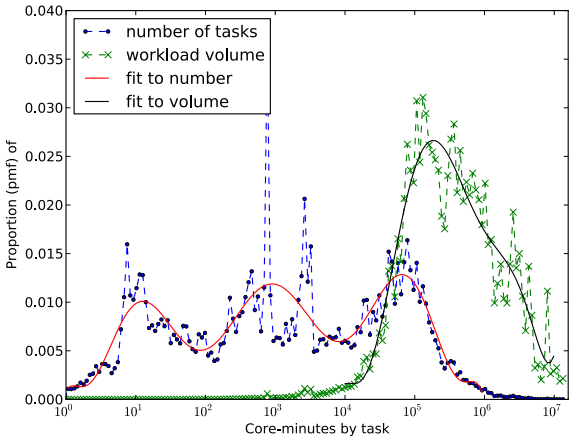


Figure 7: Task Volume

Algorithm 2 Task Execution Time Generation

$$\begin{aligned} \text{base\_samples}[1..n] &= \text{uniform}\left(0, 1.34 \cdot 10^4, n\right) \\ j^i_{\text{exec}} &= 10^{\left(3.83 \cdot 10^{-5} \cdot \text{base\_samples}[i] + 56.9\right)} \end{aligned}$$

maintenance or data transfer. The second peak, at around 10^3 core minutes, or 16 core-hours, corresponds to the tasks submitted where results are required during the same working day. If 64 cores were allocated to a task of this size, the computation time would be 15 minutes. The final peak, at around 10^5 core-minutes or 70 core-days, corresponds to the tasks that need to be returned overnight. If 128 cores were dedicated to this job, about 13 hours would be required.

An important feature of the distribution, however, is the small number of tasks that are very large. These are the tasks that are run in order to put a high-quality airframe model through a rigorous testing regime, which go towards the certification of an aircraft. Within the logs that were analysed, there were 28 tasks that took over 10 core-years of CPU time (10^7 core-minutes) to complete. Even with 128 cores allocated to them, these tasks would take over 2 months to complete execution. These are not tasks that have overrun in error, because their sheer size means that they would have been closely monitored by system administrators, and have required specific approval to run.

The fact that the workload has a similar number of small and large tasks could distract from the fact that the larger tasks represent a much larger fraction of the load placed on the cluster. Figure 7 also shows the proportion of the workload volume placed on the cluster by task size. While the majority of tasks in terms of numbers execute in less than 10^4 core-minutes, this figure shows that their contribution to the load is small. The bulk of the load comes from tasks between $10^{4.5}$ and $10^{6.5}$ core-minutes. This poses further challenges to schedulers, because of the risk of the shorter tasks, which require higher responsiveness, having to queue behind the large tasks.

number (log pmf)	volume (log pmf)
$0 \le \log_{10}(y) \le 6$	$4 \le \log_{10}(y) \le 7$
$8.04 \cdot 10^{-4}$	$-5.81 \cdot 10^{2}$
$+1.41x \cdot 10^{-2}$	$+7.82x \cdot 10^{2}$
$-1.14x^2 \cdot 10^{-1}$	$-4.47x^2 \cdot 10^{2}$
$+3.71x^3 \cdot 10^{-1}$	$+1.41x^3 \cdot 10^{2}$
$-5.06x^4 \cdot 10^{-1}$	$-2.64x^4 \cdot 10^{1}$
$+3.60x^5 \cdot 10^{-1}$	$+2.94x^5 \cdot 10^{0}$
$-1.49x^6 \cdot 10^{-1}$	$-1.81x^6 \cdot 10^{-1}$
$+3.69x^7 \cdot 10^{-2}$	$+4.72x^7 \cdot 10^{-3}$
$-5.42x^8 \cdot 10^{-3}$	
$+4.35x^9 \cdot 10^{-4}$	
$-1.47x^{10} \cdot 10^{-5}$	from 200 samples

Table 2: Task Volume Distribution Polynomial

Volume Distribution Generation

Generating workloads with task execution times that conform to a realistic distribution is crucial when evaluating the effectiveness of scheduling policies to apply to a grid for a given organisation. This is especially the case where the workload has such a wide variation of execution times as the one observed here.

In Algorithm 2, we give a method of creating workloads sampled from the log-uniform distribution observed. The expression $\text{uniform}(a,b,k)$ represents a function returning k random samples from the uniform distribution $[a,b)$.

3.2. Multi-Core Tasks

A feature of intensive simulation workloads are tasks that must execute over a number of cores. Particularly in the case of our workload, multi-core tasks must execute on the number of cores specified simultaneously and in lockstep. This is due to the structure of the particular CFD flow solvers used. The volume of space to be simulated is broken up into segments using a mesh. Each point in the mesh has a calculation performed for each time step, and then the results of that point are cascaded to all its neighbouring points. A large

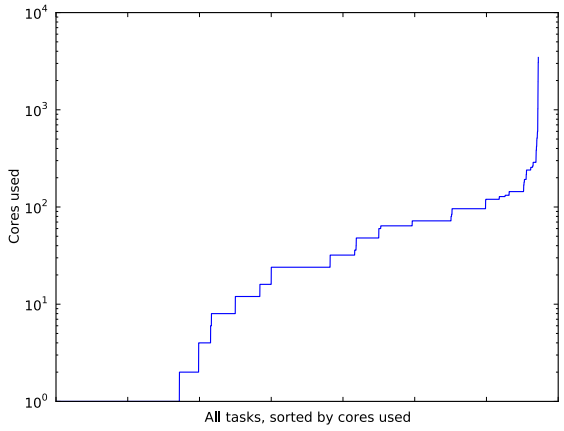


Figure 8: Workload by cores used, number of tasks

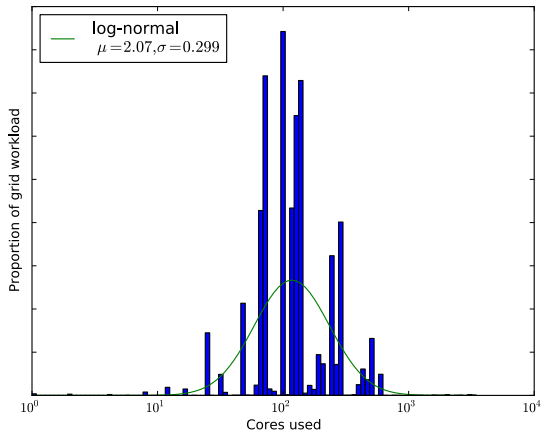


Figure 9: Workload by cores used, volume

number of time steps are usually needed to achieve convergence.

Each multi-core task is considered by the grid system to be single tasks, as one piece of software executes, just over multiple cores. This is in contrast to the dependencies between potentially different pieces of software, described in Section 4., which join tasks together to form jobs.

Users specify the number of cores required before the task has started, and this decision is informed by several constraints. RAM capacity constraints and a reduction in turnaround time give a lower limit on the number of cores used. On the other hand, there is an upper limit on the appropriate number of cores because of the network bandwidth requirements and the longer pend time for more parallel tasks.

The distribution of cores per task is shown in Figure 8. This shows that most tasks use less than 100 cores. Around a quarter of tasks run on only a single core. The step-function nature of the distribution shows the effect of users being instructed to use multiples of the number of cores in the servers available. This enables tasks to use complete multi-core servers to work on, with the aim of reducing memory and network conflicts between tasks.

Although the single-core tasks are a quarter of the number of tasks submitted, these tasks place very little load on the cluster. Unsurprisingly, the tasks that place more load on the cluster are those that are assigned more cores to execute with. The load placed on the cluster by tasks with a given number of cores is shown in figure 9. This roughly approximates a log-normal distribution with a mean of 100 cores. As before, the bunching of results shows where core counts have been rounded to an appropriate multiple of the number of cores per server.

The most highly-parallel tasks here do not actually contribute most of the load to the cluster, even though figure 7 shows that the largest tasks contribute most of the load. This means that at least some of the largest tasks are not run on the largest number of cores available. This is likely due to several factors. Firstly, the largest tasks are also some of the least urgent, and so users do not mind waiting a long time. As previously mentioned, the inefficiencies inherent in scaling to larger levels of parallelism may also mean that some of these large tasks do not actually benefit all that much from further parallelism. In fact, they may take up more of the grid's resources at one time (disadvantaging other users), without much of a net gain for the user who submitted the job. Furthermore, in order to achieve good packing of tasks to clusters, tasks of the same size are preferred.

This section has shown that task volumes fairly closely follow a log-uniform distribution in their execution times, which can be described by a log-linear trend through a sorted list of their execution times. As has been found by previous studies [6, 1], there are a smaller number of large tasks, but they contribute a very large share of the workload. This is to be expected from such a log-uniform distribution of task execution times. The volume of work by cores required is found to follow a log-normal distribution with a mean of 100 cores. This reinforces that the 25% of tasks that are single-core contribute little to the workload volume.

4. Dependencies

The execution of each task on the grid does not take place in isolation. Instead, each task usually forms part of a user's workflow. This paper uses the terminology of workflows to describe sets of tasks that are performed with a person involved. Groupings of tasks that can execute without human involvement are termed jobs. Tasks are the indivisible (yet possibly multi-core) units of work that execute on a single cluster in one go.

The tasks that compose a job have significant dependencies between them. Many different pieces of software are composed inside jobs, in order to set up, generate, post process and visualise the CFD simulations appropriately. Each task takes some input data, executes for a period of time, and produces output data. Some of this output data is valuable to the end users, and some is relevant to subsequent tasks. Executing the tasks in the right order is imperative, because later tasks in the dependency chain require data from the earlier tasks to run.

The structure of dependencies is a key aspect of characterising this sort of engineering workload, where each task run on the grid is part of a job and a higher-level workflow. A challenge for us in analysing dependency patterns was that the grid manager did

not include dependencies in its log files. However, the submission software employed by the users does store the structure of the workflows. Although was not possible therefore to make statistical generalisations of the frequency of dependency patterns within the workflows, common structures can be described. Figure 10 shows three workflow structures obtained from the workflow submission tool. The three examples represent the least complex, the average and the most complex workflow examples we found in our analysis.

Dependencies have been widely modelled in previous work by assuming that they are Directed Acyclic Graphs (DAGs) [9, 8]. However, simply stating that the dependencies are DAGs gives no further information about the internal structure of these graphs. A selection of structured dependency graph patterns used for evaluations were given by [10]. These patterns followed common algorithmic structures, such as fork-join, diamond (called mean value analysis by [10]) and in and out-trees. Methods of generating some of these patterns were given by [13], who also suggested linear chains of tasks or collections of tasks known as blocks.

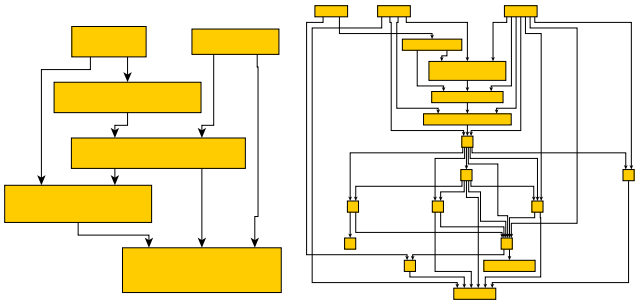
The strictly regular structures of [10], however, were not observed in these industrial workflows. The fork-join model of computation happens inside each multi-core task, rather than at the job level. Some chaining is present, but it is not perfectly linear. Tasks with large computation times tend to take a proportionately large number of inputs and have their output consumed by a proportionately large number of successors. Furthermore, the industrial job dependency graphs have somewhat less structure than these fully-structured graphs.

A common alternative way of generating DAGs with random structure is to use the Erdős–Rényi [14] model to create random graphs. A method of doing so was outlined by [13]. However, this method of generating graphs tends to produce only a narrow spread of in-degree and out-degree over the nodes in the graph. The distribution for the complex industrial pattern is noticeably more dispersed than that generated by the Erdős–Rényi model, under the same conditions. The Erdős–Rényi model specifically has a very low likelihood of nodes with a large in- or out-degree. The mean and standard deviations, as well as the number of source and sink nodes are also different to that observed (See table in Figure 10)

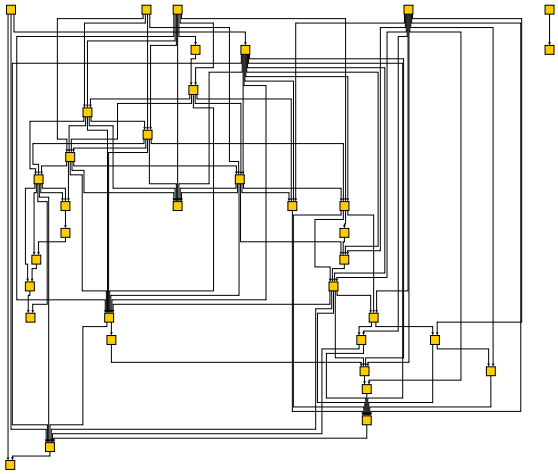
Because the Erdős–Rényi (E-R) model has these shortcomings, we developed Algorithm 4 to generate random graphs that better approximate the degree distribution of nodes in the observed industrial graph. This method uses an integer version (Algorithm 3) of the UUnifast algorithm of [15] to generate a logarithmic distribution on the in- and out-degree for the nodes, and then creates random dependency connections that satisfy these distributions. Our method gives greater importance to some nodes, representing a higher level of structure than a purely random graph, and this more closely parallels the structures observed in industry.

5. Conclusion

The aim of this paper was to characterise the grid workload of an engineering design department of a large aircraft manufacturer by analysing log files that spanned a period of 30 months. It was found that there are distinctive daily and weekly patterns, with work being submitted faster than it can be processed during working hours - a distinctive feature of this characterisation. A means of generating a workload with peaks corresponding to those observed is presented. The task execution times are shown to follow a



(a) Simple (b) Average



(c) Complex

Figure 10: Dependency Patterns

	Nodes	Edges	Edge Density	Sources	Sinks	In-degree		Out-degree	
Simple	6	8	26.6%	2	1	μ	σ	μ	σ
Average	18	39	12.7%	3	3				
Complex	36	98	7.78%	5	5	2.69	2.15	2.64	2.61
E-R	36	98	7.78%	15	9	1.33	1.31	1.33	1.2
New	36	98	7.78%	6	6	2.72	2.37	2.72	2.61

Table 3: Dependency Graph Metrics

Algorithm 3 Pseudocode for UUnifast Integer

```
UUnifInt(samples, sum_of_samples):
vectU = []
sumU = sum_of_samples
for i in 0..(samples − 1):
  nextSumU = round(sumU · (random())^(1/(samples−i)))
  vectU.append(sumU) − nextSumU
  sumU = nextSumU
vectU.append(SumU)
return vectU
```

Algorithm 4 Python code for random graph generation with high spread of in- and out-degrees

```
def GraphGen(n, e):
found_outer = False
while not found_outer:
 ins_by_node = sorted(UUnifInt(n-1, e)+[0])
 outs_by_node = sorted(UUnifInt(n-1, e)+[0])[::-1]
 found_inner = False
 itercount = 0
 while (not found_inner) and (itercount <= 40):
  itercount += 1
  found_inner = True
  o = {x: outs_by_node[x] for x in range(n)}
  i = {x: ins_by_node[x] for x in range(n)}
  edges = []
  while len(edges) < e:
   K = max([x[0] for x in i.items() if x[1] > 0])
   P = [x[0] for x in o.items() if x[1] > 0 and x[0] < K]
   D = [(u,K) for u in P if (u,K) not in edges]
   if len(D) == 0:
    found_inner = False
    break
   else:
    new_edge = random.choice(D)
    o[new_edge[0]] -= 1
    i[new_edge[1]] -= 1
    edges.append(new_edge)
 found_outer = not len(edges) < e:
return edges
```

log-uniform distribution, and a algorithm to generate a sample of such a distribution is given. Dependency patterns within workflows were characterised in a way few authors have done before, and an algorithm given to create jobs with such patterns.

The parameters of this characterised workload are likely to be common to engineering design workloads, where the appetite for computational capacity is large and a hierarchical decomposition of work is followed. Synthetic workloads based on the parameters shown can be generated using the methods presented and be used in the evaluation of scheduling policies within industrially-relevant contexts.

Acknowledgements

The authors wish to thank the EPSRC (grant number EP/F501374/1) for funding this research through the UK's Large-Scale Complex IT Systems (LSCITS) programme.

References

[1] D. G. Feitelson and B. Nitzberg, "Job characteristics of a production parallel scientific workload on the nasa ames ipsc/860," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, D. Feitelson and L. Rudolph, Eds. Springer Berlin Heidelberg, 1995, vol. 949, pp. 337–360. [Online]. Available: http://dx.doi.org/10.1007/3-540-60153-8_38

[2] A. Burkimsher, I. Bate, and L. S. Indrusiak, "A survey of scheduling metrics and an improved ordering policy for list schedulers operating on workloads with dependencies and a wide variation in execution times," *Future Generation Computer Systems. In press, Corrected Proof. Available Online 27 December 2012*. [Online]. Available: http://dx.doi.org/10.1016/j.future.2012.12.005

[3] N. Poggi, D. Carrera, R. Gavalda, J. Torres, and E. Ayguade, "Characterization of workload and resource consumption for an online travel and booking site," in *Proceedings of the IEEE International Symposium on Workload Characterization*, ser. IISWC '10. Washington, DC, USA: IEEE Comput. Soc., 2010, pp. 1–10.

[4] Z. Ren, J. Wan, W. Shi, X. Xu, and M. Zhou, "Workload analysis, implications and optimization on a production hadoop cluster: A case study on taobao," *IEEE Transactions on Services Computing*, vol. 99, p. 1, 2013.

[5] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan, "An analysis of traces from a production mapreduce cluster," in *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, 2010, pp. 94–103.

[6] S.-H. Chiang and M. Vernon, "Characteristics of a large shared memory production workload," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, D. Feitelson and L. Rudolph, Eds. Springer Berlin Heidelberg, 2001, vol. 2221, pp. 159–187. [Online]. Available: http://dx.doi.org/10.1007/3-540-45540-X_10

[7] H. You and H. Zhang, "Comprehensive workload analysis and modeling of a petascale supercomputer," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, W. Cirne, N. Desai, E. Frachtenberg, and U. Schwiegelshohn, Eds. Springer Berlin Heidelberg, 2013, vol. 7698, pp. 253–271. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35867-8_14

[8] H. Topcuouglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, March 2002.

[9] M. Maheswaran, T. D. Braun, and H. J. Siegel, "Heterogeneous distributed computing," in *In Encyclopedia of Electrical and Electronics Engineering*. John Wiley, 1999, pp. 679–690.

[10] Y. Kwok and I. Ahmad, "Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 5, pp. 506–521, May 1996.

[11] D. Feitelson and E. Shmueli, "A case for conservative workload modeling: Parallel job scheduling with daily cycles of activity," in *Proceedings of the IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems, 2009.*, ser. MASCOTS '09, 2009, pp. 1–8.

[12] H. Li, D. Groep, and L. Wolters, "Workload characteristics of a multi-cluster supercomputer," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Springer Berlin Heidelberg, 2005, vol. 3277, pp. 176–193. [Online]. Available: http://dx.doi.org/10.1007/11407522_10

[13] A. Burkimsher, "Dependency patterns and timing for grid workloads," in *Proceedings of the 4th York Doctoral Symposium on Computer Science*, October 2011, pp. 25–33. [Online]. Available: http://www.cs.york.ac.uk/ftpdir/reports/2011/YCS/468/YCS-2011-468.pdf

[14] P. Erdős and A. Rényi, "On the evolution of random graphs," in *Publication Of The Mathematical Institute Of The Hungarian Academy Of Sciences*, 1960, pp. 17–61.

[15] E. Bini and G. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.