Workload Modeling for Performance Evaluation

Dror G. Feitelson

School of Computer Science and Engineering
The Hebrew University, 91904 Jerusalem, Israel
feit@cs.huji.ac.il
http://www.cs.huji.ac.il/~feit

Abstract. The performance of a computer system depends on the characteristics of the workload it must serve: for example, if work is evenly distributed performance will be better than if it comes in unpredictable bursts that lead to congestion. Thus performance evaluations require the use of representative workloads in order to produce dependable results. This can be achieved by collecting data about real workloads, and creating statistical models that capture their salient features. This survey covers methodologies for doing so. Emphasis is placed on problematic issues such as dealing with correlations between workload parameters and dealing with heavy-tailed distributions and rare events. These considerations lead to the notion of structural modeling, in which the general statistical model of the workload is replaced by a model of the process generating the workload.

1 Introduction

The goal of performance evaluation is often to compare different system designs or implementations. The evaluation is expected to bring out performance differences that will allow for an educated decision regarding what design to employ or what system to buy. Thus it is implicitly assumed that observed performance differences indeed reflect important differences between the systems being studied.

However, performance differences may also be an artifact of the evaluation methodology. The performance of a system is not only a function of the system design and implementation. It may also be affected by the workload to which the system is subjected. For example, communication networks have often been analyzed using Poisson-related models of traffic, which indicated that the variance in load should smooth out over time and when multiple data sources are combined. But in 1994 Leland and co-workers showed, based on extensive observations and measurements, that this does not happen in practice [52]. Instead, they proposed a self-similar traffic model that captures the burstiness of network traffic and leads to more realistic evaluations of required buffer space and other parameters [24].

Analyzing network traffic was easy, in a sense, because all packets are of equal size and the only characteristic that required measurement and modeling was the arrival process. But if we consider a complete computer system, the problem becomes more complex [13, 11]. For example, a computer program may require a certain amount of CPU time, memory, and I/O, and these resource requirements may be interleaved in various ways during its execution. In addition there are several levels at which we might model the system: we can study the functional units used by a stream of instructions, the subsystems used by a job during its execution, or the requirements of jobs submitted to the system over time. Each of these scales is relevant for the design and evaluation of different parts of the system: the CPU, the hardware configuration, or the operating system.

The main domain used as a source of examples in this survey is that of parallel job scheduling. Workloads in this field are interesting due to the combination of being relatively small and at the same time relatively complex. The size of typical workloads is tens of thousands of jobs, as opposed to millions of packets in communication workloads. These workloads are characterized by a large number of factors, including the job sizes, runtimes, runtime estimates, and arrival patterns. The complexity derives not only from the multiple factors themselves, but from various correlations between them. Research on these issues is facilitated by the availability of data and models in the Parallel Workloads Archive [60]. In addition, there are several documented cases of how workload parameters influence the outcomes of performance evaluation studies [53, 57, 25].

2 Data Sources

The suggestion that workload modeling should be based on measurements is not new [32, 4]. However, for a long time relatively few models based on actual measurements were published. As a result, many performance studies did not use experimental workload models at all (and don't to this day).

It is true that real-world data is not always available, or may be hard to obtain. But not using real data may lead to flawed evaluations [26]. This realization has led to a new wave of workload analyses in various fields of system design in recent years. Maybe the most prominent are the study of Internet traffic patterns [52, 62, 75] and world-wide web traffic patterns, with the intent of using the knowledge to evaluate server performance and caching schemes [5, 18, 6]. Other examples include studies of process arrivals and runtimes [12, 37], file systems [36], and video streams [48]. In the area of parallel systems, descriptive studies of workloads have only started to appear in recent years [29, 76, 58, 27, 14]. There are also some attempts at modeling [10, 28, 21, 41, 23, 54, 15] and on-line characterization [34].

But where does the data come from? There are two main options: use data that is available anyway, or collect data specifically for the workload model. The latter can be done in two ways: active or passive instrumentation. Importantly, collected data can and should be made publicly available for use by other researchers [60, 33].

2.1 Using Accounting and Activity Logs

The most readily available source of data is accounting or activity logs. Such logs are kept by the system for auditing, and record selected attributes of all activities. For example, many computer systems keep a log of all executed jobs. In large scale parallel systems, these logs can be quite detailed and are a rich source of information for workload studies [60]. Another example is web servers, that are often configured to log all requests.

A good example is provided by the analysis of three months of activity on the 128-node NASA Ames iPSC/860 hypercube supercomputer. This analysis provided the following data [29]:

- The distribution of job sizes (in number of nodes) for system jobs, and for user jobs classified according to when they ran: during the day, at night, or on the weekend.
- The distribution of total resource consumption (node seconds), for the same job classifications.
- The same two distributions, but classifying jobs according to their type: those that were submitted directly, batch jobs, and Unix utilities.
- The changes in system utilization throughout the day, for weekdays and weekends.
- The distribution of multiprogramming level seen during the day, at night, and on weekends. This also included the measured down time (a special case of 0 multiprogramming).
- The distribution of runtimes for system jobs, sequential jobs, and parallel jobs, and for jobs with different degrees of parallelism. This included a connection between common runtimes and the queue time limits of the batch scheduling system.
- The correlation between resource usage and job size, for jobs that ran during the day, at night, and over the weekend.
- The arrival pattern of jobs during the day, on weekdays and weekends, and the distribution of interarrival times.
- The correlation between the time of day a job is submitted and its resource consumption.
- The activity of different users, in terms of number of jobs submitted, and how many of them were different.
- Profiles of application usage, including repeated runs by the same user and by different users, on the same or on different numbers of nodes.
- The dispersion of runtimes when the same application is executed many times.

Note, however, that accounting logs do not always exist at the desired level of detail. For example, even if all communication on a web server is logged, this is at the request level, not at the packet level. To obtain packet-level data, specialized instrumentation is needed.

2.2 Passive and Active Instrumentation

If data is not readily available, it should be collected. This is done by instrumenting the system with special facilities that record its activity. A major problem with this is being unobtrusive, and not modifying the behavior of the system while we measure it.

Passive instrumentation refers to designs in which the system itself is not modified. The instrumentation is done by adding external components to the system, that monitor system activity but do not interfere with it. This approach is commonly used is studies of communication, where it is relatively easy to add a node to a system that only listens to the traffic on the communication network [52, 73, 35]. A more extreme example is a proposal to add a shadow parallel machine to a production parallel machine, with each shadow node monitoring the corresponding production node, and all of them cooperating to filter and summarize the data [66].

Active instrumentation refers to the modification of the system so that it will collect data about its activity. This can be integrated with the original system design, as was done for example in the RP3 [43]. However, it is more commonly done after the fact, when a need to collect data about a specific system arises. A good example is the Charisma project, which set out to characterize the I/O patterns on parallel machines [59]. This was done by instrumenting the I/O library and requesting users to re-link their applications; when running with the instrumented library, all I/O activity was recorded for subsequent analysis.

Obviously, instrumenting a system to collect data at runtime can affect the systems behavior and performance. This may not be very troublesome in the case of I/O activity, which suffers from high overhead anyway, but may be very problematic for the study of fine grain events related to communication, synchronization, and memory usage. One possible solution to this problem is to model the effect of the instrumentation, thereby enabling it to be factored out of the measurement results [55]. This leads to results that reflect real system behavior (that is, unaffected by the instrumentation), but leaves the problem of performance degradation while the measurements are being taken. An alternative is to selectively activate only those parts of the instrumentation that are needed at each instant, rather than collecting data about the whole system all the time. Remarkably, this can be done efficiently by modifying the system's object code as it runs [38].

2.3 Data Sanitation

Before data can be used to create a workload model, it has to be cleaned up. This has several aspects.

One important aspect is the handling of outliers. Workload logs sometimes include uncommon events that “don't make sense”. Examples include

– In the two-year log of jobs run on the LANL CM-5 parallel machine, there is a 10-day stretch in which a single user ran about 5000 instances of a job that executed in 1–2 seconds on 128 nodes.

- In the two-year log of jobs run on the SDSC Paragon parallel machine, there is a large concentration of short jobs that arrive at 3:30 AM on different days. This is probably due to periodic invocation of administrative scripts.
- In the two-year log of jobs run on the SDSC SP2 parallel machine, there is a single hour in which a single user submitted some 580 similar jobs.

Of course, the decision that something is "uncommon" is subjective. The purist approach would be to leave everything in, because in fact it did happen in a real system. But on the other hand, while strange things may happen, it is difficult to argue for a specific one; if we leave it in the workload that is used to analyze systems, we run the risk of promoting systems that specifically cater for a singular unusual condition that is unlikely to ever occur again.

A procedure that was advocated by Cirne and Berman is to use clustering as a means to distinguish between "normal" and "abnormal" data [15]. Specifically, they characterize days in a workload log by an n-valued vector, and cluster these vectors into two clusters in R^n. If the clustering procedure distinguishes a single day and puts it in a cluster by itself, this day is removed and the procedure is repeated with the data that is left. Note, however, that this has its risks: first, abnormal behavior may span more than a single day, as the above examples show; moreover, removing days may taint other data, e.g. when interarrival times are considered.

Another aspect of workload sanitation involves errors. Workload logs may contain data about activities that failed to complete successfully, e.g. jobs that were submitted and either failed or were killed by the user. Should these jobs be included or deleted from the data? On one hand, they represent work that the system had to handle, even if nothing came of it. On the other hand, they do not represent useful work, and may have been submitted again later. An interesting compromise is to keep such data, and explicitly include it in the workload model [15]. This will enable the study of how failed work affects system utilization and the performance of "good" work.

Finally, an important issue is determining the degree to which data is generally representative. One problem is that data may be affected by local procedures and constraints where it was collected. For example, data on programs run on a machine equipped with only 32MB memory will show that programs do not have larger resident sets, but this is probably an artifact of this limit, and not a real characteristic of general workloads. A more striking example is provided by the NASA iPSC log mentioned above. In this log a full 57% of the jobs are invocations of the Unix pwd command on various nodes, which was the technique used by system personnel to verify that the system was working [29]. Another problem is that workloads may evolve with time [39], especially on large and unique installations such as parallel supercomputers. It is therefore important to capture data from a mature system, and not a new (or old) one.

3 Workload Modeling

There are two common ways to use a measured workload to analyze or evaluate a system design [32]: (1) use the traced workload directly to drive a simulation, or (2) create a model from the trace and use the model for either analysis or simulation. For example, trace-driven simulations based on large address traces are often used to evaluate cache designs [45, 42]. But models of how applications traverse their address space have also been proposed, and provide interesting insights into program behavior [71, 72].

3.1 Why Model

The advantage of using a trace directly is that it is the most "real" test of the system; the workload reflects a real workload precisely, with all its complexities, even if they are not known to the person performing the analysis.

The drawback is that the trace reflects a specific workload, and there is always the question of whether the results generalize to other systems or load conditions. In particular, there are cases where the workload depends on the system configuration, and therefore a given workload is not necessarily representative of workloads on systems with other configurations. Obviously, this makes the comparison of different configurations problematic. In addition, traces are often misleading if we have incomplete information about the circumstances when they were collected. For example, workload traces often contain intervals when the machine was down or part of it was dedicated to a specific project, but this information may not be available.

Workload models have a number of advantages over traces [70].

- It is possible to change model parameters one at a time, in order to investigate the influence of each one, while keeping other parameters constant. This allows for direct measurement of system sensitivity to the different parameters. It is also possible to select model parameters that are expected to match the specific workload at a given site.
  In general it is not possible to manipulate traces in this way, and even when it is possible, it can be problematic. For example, it is common practice to increase the modeled load on a system by reducing the average interarrival time. But this practice has the undesirable consequence of shrinking the daily load cycle as well. With a workload model, we can control the load independent of the daily cycle.
- Using a model, it is possible to repeat experiments under statistically similar conditions that are nevertheless not identical. For example, a simulation can be run several times with different seeds for the random number generator. This is needed in order to compute confidence intervals.
- Logs may not represent the real workload due to various problems: a limit of 4 hours may force users to break long jobs into multiple short jobs, jobs killed by the system may be repeated, etc. If taken at face value this may be misleading, but the problem is that often we do not know about such problems.

Conversely, a modeler has full knowledge of model workload characteristics. For example, it is easy to know which workload parameters are correlated with each other because this information is part of the model.

– Finally, modeling increases our understanding, and can lead to new designs based on this understanding. For example, identifying the repetitive nature of job submittal can be used for learning about job requirements from history. One can design a resource management policy that is parameterized by a workload model, and use measured values for the local workload to tune the policy.

The main problem with models, as with traces, is that of representativeness. That is, to what degree does the model represent the workload that the system will encounter in practice? The answer depends in part on the degree of detail that is included. As noted above, each job is composed of procedures that are built of instructions, and these interact with the computer at different levels. One option is to model these levels explicitly, creating a hierarchy of interlocked models for the different levels [13, 10, 64]. This has the obvious advantage of conveying a full and detailed picture of the structure of the workload. In fact, it is possible to create a whole spectrum of models spanning the range from condensed rudimentary models to direct use of a detailed trace.

For example, the sizes of a sequence of jobs need not be modeled independently. Rather, they can be derived from a lower-level model of the jobs' structures [30]. Hence the combined model will be useful both for evaluating systems in which jobs are executed on predefined partitions, and for evaluating systems in which the partition size is defined at runtime to reflect the current load and the specific requirements of jobs.

The drawback of this approach is that as more detailed levels are added, the complexity of the model increases. This is detrimental for three reasons. First, more detailed traces are needed in order to create the lower levels of the model. Second, it is commonly the case that there is wider diversity at lower levels. For example, there may be many jobs that use 32 nodes, but at a finer detail, some of them are coded as data parallel with serial and parallel phases, whereas others are written with MPI in an SPMD style. Creating a representative model that captures this diversity is hard, and possibly arbitrary decisions regarding the relative weight of the various options have to be made. Third, it is harder to handle such complex models. While this consideration can be mitigated by automation [70, 44], it leaves the problem of having to check the importance and impact of very many different parameters.

3.2 How to Model

The most common approach used in workload modeling is to create a statistical summary of an observed workload. This is applied to all the workload attributes, e.g. computation, memory usage, I/O behavior, communication, etc. [46]. It is typically assumed that the longer the observation period, the better. Thus we can summarize a whole year's workload by analyzing a record of all the jobs that ran on a given system during this year. A synthetic workload can then be generated according to the model, by sampling from the distributions that constitute the model.

The question of what exactly to model, and at what degree of detail, is a hard one. On one hand, we want to fully characterize all important workload attributes. On the other hand a parsimonious model is more manageable, as there are less parameters whose values need to be assessed and whose influence needs to be studied. Also, there is a danger of over-fitting a particular workload at the expense of generality.

Fitting Distributions The goal of a model is to be able to create a synthetic workload that mimics the original (possibly with certain modifications, according to the effects we wish to study). The statistical summary is therefore a distribution, or collection of distributions for various workload attributes. By sampling from these distributions we then create the model workload [49].

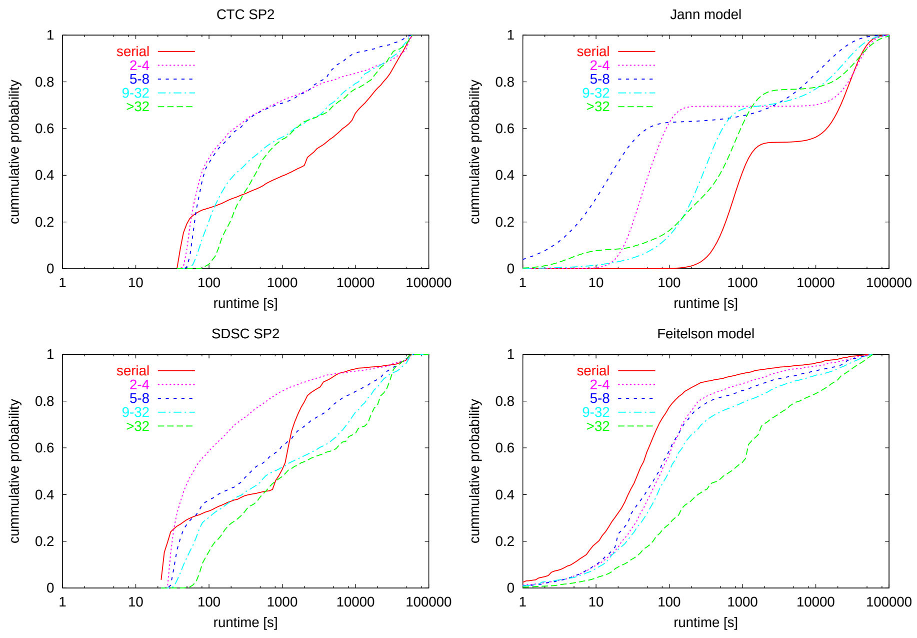


Fig. 1. *Distributions of runtimes for different ranges of job sizes, in two workload logs and two models of parallel jobs.*

One way to select suitable distributions is based on moments, and especially the mean and the variance of the sample data [23]. For example, these statistics indicate that the distribution of job runtimes has a wide dispersion, leading to a preference for a hyper-exponential model over an exponential one. Jann et al.

have used hyper-Erlang distributions to create models that match the first 3 moments of a distribution [41]. However, such summaries may be misleading, because they may not represent the shape of the distribution correctly. Specifically, in the Jann models, the distributions become distinctly bimodal, whereas the original data is much more continuous (Figure 1). The Feitelson model, which uses a three-stage hyper-exponential distribution, more closely resembles the original data in this respect.

The use of distributions with the right shape is not just an esthetic issue. Some 25 years ago Lazowska showed that using models based on a hyper-exponential distribution with matching moments to evaluate a simple queueing system leads to inaccurate results [50], and advocated the use of distributions with matching percentiles instead. He also noted that a hyper-exponential distribution has three parameters, whereas the mean and standard deviation of data only define two, so many different hyper-exponential distributions that match the first two moments are possible — and lead to different results.

Rec's omitted		statistic (% change)		
(% of total)		mean [sec]	CV	median [sec]
0	(0%)	9371	3.1	552
5	(0.01%)	9177 (-2.1%)	2.2 (-29%)	551 (-0.2%)
10	(0.02%)	9094 (-3.0%)	2.0 (-35%)	551 (-0.2%)
20	(0.04%)	9023 (-3.7%)	1.9 (-39%)	551 (-0.2%)
40	(0.08%)	8941 (-4.6%)	1.9 (-39%)	550 (-0.4%)
80	(0.16%)	8834 (-5.7%)	1.8 (-42%)	549 (-0.5%)
160	(0.31%)	8704 (-7.1%)	1.8 (-42%)	546 (-1.1%)

Table 1. *Sensitivity of statistics to the largest data points. Data regarding runtimes on the CTC SP2 machine from [23] courtesy of Allen Downey.*

Another problem with using statistics based on high moments of the data is that they are very sensitive to rare large samples [23]. Table 1 shows data based on the runtimes of 50866 parallel jobs from the CTC SP2 machine. Removing just the top 5 values causes the mean to drop by 2%, and the coefficient of variation (the standard deviation divided by the mean) to drop by 29%. The median, as a representative of order statistics, only changes by 0.2%. As the extreme values observed in a sample are not necessarily representative, this implies that the model may be largely governed by a small number of unrepresentative samples.

Finding a distribution that matches given moments is relatively easy, because it can be done based on inverting equations that relate a distribution's parameters to its moments. Finding a distribution that fits a given shape is typically harder [54]. One possibility is to use a maximum likelihood method, which finds the parameters that most likely gave rise to the observed data. Another option is to use an iterative method, in which the goodness of fit at each stage is quantified using the Chi-square test, the Kolmogorov-Smirnov test, or the

Anderson-Darling test (which is like the Kolmogorov-Smirnov test but places more emphasis on the tail of the distribution).

Correlations Modeling the distribution of each workload attribute in isolation is not enough. An important issue that has to be considered is possible correlations between different attributes.

Correlations are important because they can have a dramatic impact on system behavior. Consider the scheduling of parallel jobs on a massively parallel machine as an example. Such scheduling is akin to 2D bin packing: each job is represented by a rectangle in processors$\times$time space, and these rectangles have to be packed as tightly as possible. Assuming that when each job is submitted we know how many processors it needs, but do not know for how long it will run, it is natural to do the packing according to size. Specifically, packing the bigger jobs first may be expected to lead to better performance [16]. But what if there is a correlation between size and running time? If this is an inverse correlation, we find a win-win situation: the larger jobs are also shorter, so packing them first is statistically similar to using SJF (shortest job first) [47]. But if size and runtime are correlated, and large jobs run longer, scheduling them first may cause significant delays for subsequent smaller jobs, leading to dismal average performance [53].

System	Correlation
CTC SP2	−0.029
KTH SP2	0.011
SDSC SP2	0.145
LANL CM-5	0.211
SDSC Paragon	0.305

Table 2. *Correlation coefficient of runtime and size for different parallel supercomputer workloads.*

Establishing whether or not a correlation exists is not always easy. The commonly used correlation coefficient only yields high values if a strong linear relationship exists between the variables. In the example of the size and runtime of parallel jobs, the correlation coefficient is typically rather small (Table 2), and a scatter plot shows no significant correlation either (Figure 2). However, these two attributes are actually correlated with each other, as seen from the distributions for the CTC and SDSC logs in Figure 1. In both of these, the distribution of runtimes for ranges of larger job-sizes distinctly favors longer runtimes, whereas smaller jobs sizes favor short runtimes[1].

A coarse way to model correlation, which avoids this problem altogether, is to represent the workload as a set of points in a multidimensional space, and apply

[1] The only exception is the serial jobs on the CTC machine, which have very long runtimes; but this anomaly is unique to the CTC workload.

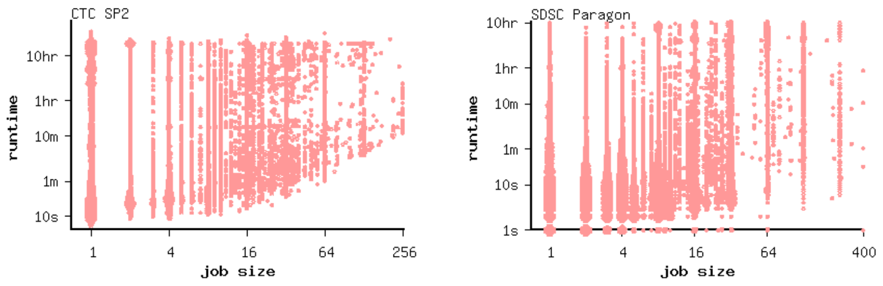


Fig. 2. *The correlation between job sizes and runtimes on parallel supercomputers. The scatter-plot data is from the SDSC Paragon parallel machine.*

clustering [13]. For example, each job can be represented by a tuple including its runtime, its size, its memory usage, and so on. By clustering we can then select a small number of representative jobs, as use them as the basis of our workload model; each such job comes with a certain (representative) combination of values for the different attributes. However, many workloads do not cluster nicely — rather, attribute values come from continuous distributions, and many different combinations are all possible.

The direct way to model a correlation between two attributes is to use the joint distribution of the two attributes. This suffers from two problems. One is that it may be expected to be hard to find an analytical distribution function that matches the data. The other is that for a large part of the range, the data may be very sparse. For example, most parallel jobs are small and run for a short time, so we have a lot of data about small short jobs. But we may not have enough data about large long jobs to say anything meaningful about the distribution — we just have a small set of unrelated samples.

The typical solution is therefore to divide the range of one attribute into sub-ranges, and model the distribution of the other attribute for each such sub-range. For example, the Jann model of supercomputer workloads divides the job size scale according to powers of two, and creates an independent model of the runtimes for each range of sizes [41]. As can be seen in Figure 1, these models are completely different from each other. An alternative is to use the same model for all subranges, and define a functional dependency of the model parameters on the subrange. For example, the Feitelson model first selects the size of each job according to the distribution of job sizes, and then selects a runtime from a distribution of runtimes that is conditioned on the selected size [28]. Specifically, the runtime is selected from a two-stage hyperexponential distribution, and the probability for using the exponential with the higher mean is linearly dependent on the size:

$$p(n) = 0.95 - 0.2(n/N)$$

Thus, for small jobs (the job size n is small relative to the machine size N) the probability of using the exponential with the smaller mean is 0.95, and for large jobs this drops to 0.75.

Stationarity A special type of correlation is correlation with time. This means that the workload changes with time: it is not stationary.

On short time scales, the most commonly encountered non-stationary phenomenon is the daily work cycle. In many systems, the workload at night is quite different from the workload during the day. Many workload models ignore this and focus on the daytime workload, assuming that it is stationary. However, when the workload includes items whose duration is on the scale of hours (such as parallel jobs), the daily cycle cannot be ignored. There are two typical ways for dealing with it. One is to divide the day into a number of ranges, and model each one separately assuming that it is stationary [14]. The other is to use parameterized distributions, and model the daily cycle by showing how the parameters change with time of day [54].

Over long ranges, a non-stationary workload can be the result of changing usage patterns as users get to know the system better. It can also result from changing missions, e.g. when one project ends and another takes its place. Such effects are typically not included in workload models, but they could affect the data on which models are based. We return to this issue in Section 5.

Assumptions An important point that is often overlooked in workload modeling is that everything has to be modeled. It is not good to model one attribute with great precision, but use unbased assumptions for the others.

The problem is that assumptions can be very tempting and reasonable, but still be totally untrue. For example, it is reasonable to assume that parallel jobs are used for speedup, that is, to complete the computation faster. After all, this is the basis for Amdahl's Law. But other possibilities also exist — for example, parallelism can be used to solve the same problem with greater precision rather than faster. The problem is that assuming speedup is the goal leads to a model in which parallelism is inversely correlated with runtime, and this has an effect on scheduling [53, 26]. Observations of real workloads indicate that this is not the case, as shown above.

Another reasonable assumption is that users will provide the system with accurate estimates of job runtimes when asked to. At least on large scale parallel systems, users indeed spend significant effort tuning their applications, and may be expected to have this information. Also, backfilling schedulers reward low estimates but penalize underestimates, leading to a convergence towards accurate estimates. Nevertheless, studies of user estimates reveal that they are often highly inaccurate, and often represent an overestimate by a full order of magnitude [57]. Surprisingly, this can sway results comparing schedulers that use the estimates to decide whether to backfill jobs (that is, to use them to fill holes in an existing schedule) [25].

4 Heavy Tails, Self Similarity, and Burstiness

A major problem with applying the techniques described in the previous section occurs when the data is "bad" [3]. This is best explained by an example. If the data fits, say, an exponential distribution, then a running average of growing numbers of data samples quickly converges to the mean of the distribution. But bad data is ill-behaved: it does not converge when averaged, but rather continues to grow and fluctuate. Such effects have received considerable attention lately, as many different data sets were found to display them. For more technical detail on this topic, see [62, 61].

4.1 Distributions with Heavy Tails

A very common situation is that distributions have many small elements, and few large elements. For example, there are many small files and few large files; many short processes and few long processes. The question is how dominant are the large elements relative to the small ones. In heavy-tailed distributions, the rare large elements (from the tail of the distribution) dominate.

In general, the relative importance of the tail can be classified into one of three cases [62]. Consider trying to estimate the length of a process, given that we know that it has already run for a certain time, and that the mean of the distribution of process lengths is m.

– If the distribution of process lengths has a *short* tail, than the more we have waited already, the less additional time we expect to wait. The mean of the tail is smaller than m. For example, this would be the case if the distribution was uniform over a certain range.
– If the distribution is *memoryless*, the expected additional time we need to wait for the process to terminate is independent of how long we have waited already. The mean length of the tail is always the same as the mean length of the whole distribution. This is the case for the exponential distribution.
– But if the distribution is *heavy* tailed, the additional time we may expect to wait till the process terminates grows with the time we have already waited. The mean of the tail is larger than m, the mean of the whole distribution. An example of this type is the Pareto distribution.

An important consequence of heavy tailed distributions is the mass disparity phenomenon: a small number of samples account for the majority of mass, whereas all small samples account for negligible mass [17]. Conversely, a typical sample is small, but a typical unit of mass comes from a large sample. Using concrete examples from computers, a typical process is short, but a typical second of CPU activity is part of a long process; a typical file is small, but a typical byte of storage belongs to a large file (Figure 3). This disparity is sometimes referred to as the "mice and elephants" phenomenon. But this metaphor may conjure the image of a bimodal distribution[2], which could be misleading: in most cases, the distribution is continuous.

[2] A typical mouse weighs about 28 grams, whereas an elephant weighs 3 to 6 tons, depending on whether it is Indian or African. Cats, dogs, and zebras, which fall in

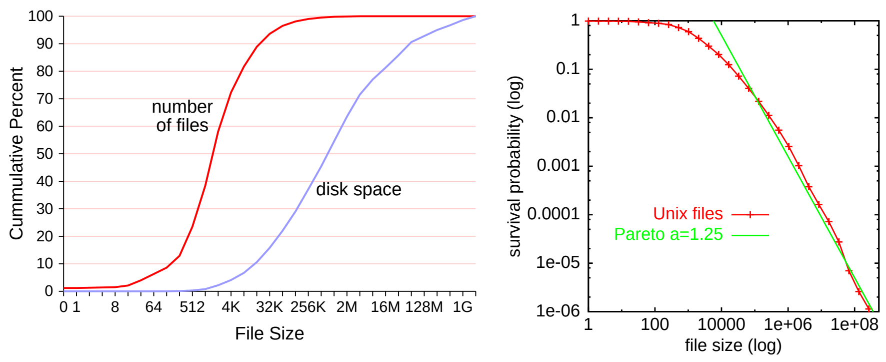


Fig. 3. *The distribution of file sizes, from a 1993 survey of 12 million Unix files [40]. Left: 90% of the files are less than 16KB long, and use only some 10% of the total disk space. Half the disk space is occupied by a very small fraction of large files. Right: log-log complementary distribution plot, with possible Pareto model of the tail; see Equation (2).*

Formally, it is common to define heavy tailed distributions to be those whose tails decay like a power law — the probability of sampling a value larger than x is proportional to one over x raised to some power [62]:

$$\bar{F}(x) = \Pr[X > x] \sim x^{-a} \qquad\qquad 0 < a < 2 \tag{1}$$

where $\bar{F}(x)$ is the survival function (that is, $\bar{F}(x) = 1 - F(x)$), and $\sim$ means "has the same distribution". This is a very strong statement. Consider an exponential distribution. The probability of sampling a value larger than say 100 times the mean is e^{-100}, which is totally negligible for all intents and purposes. But for a Pareto distribution with $a = 2$, this probability is $1/40000$: one in every 40000 samples will be bigger than 100 times the mean. While rare, such events can certainly happen. When the shape parameter is $a = 1.1$, and the tail is heavier, this probability increases to one in 2216 samples.

An important characteristic of heavy tailed distributions is that some of their moments may be undefined. Specifically, using the above definition, if $a \leq 1$ the mean will be undefined, and if $a \leq 2$ the variance will be undefined. But what does this mean? Consider a Pareto distribution with $a = 1$, whose probability density is proportional to x^{-2}. Trying to evaluate its mean leads to

$$E[x] = \int cx \frac{1}{x^2} \mathrm{d}x = c \ln x$$

so the mean is infinite. But for any finite number of samples, the mean obviously exists. The answer is that the mean grows logarithmically with the number of observations. However, this statement is misleading, as the running mean does

between, are missing from this picture.

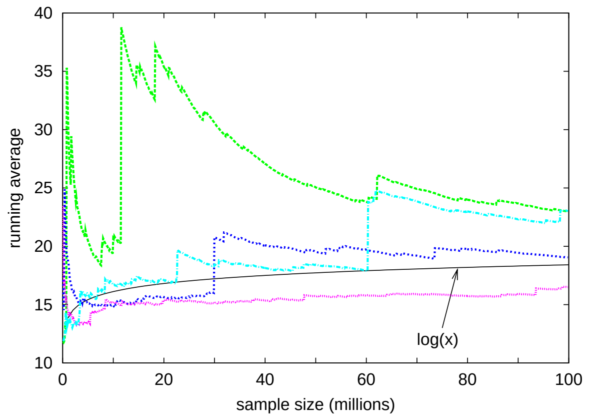


Fig. 4. *Examples of the running mean of samples from a Pareto distribution. Four plots using different random number generator seeds are shown.*

not actually resemble the log function. In fact, it grows in big jumps every time a large observation from the tail of the distribution is sampled, and then it slowly decays again towards the log function (Figure 4).

The definition (1) can also be used to determine if a given data set is heavy tailed. Taking the log from both sides we observe that

$$\log \bar{F}(x) = \log x^{-a} = -a \log x \tag{2}$$

So plotting $\log \bar{F}(x)$ (the log of the fraction of observations larger than x) as a function of $\log x$ should lead to a straight line with slope $-a$ (this is sometimes called a "log-log complementary distribution plot", or LLCD, see Figure 3).

This technique can be further improved by aggregating successive observations (that is, replacing each sequence of k observations by their sum). Distributions for which such aggregated random variables have the same distribution as the original are called stable distributions. The Normal distribution is the only stable distribution with finite variance. Heavy tailed distributions (according to definition (1)) are also stable, but have an infinite variance. Thus the central limit theorem does not apply, and the aggregated random variables do not have a Normal distribution. Rather, they have the same heavy-tailed distribution. This can be verified by creating LLCD plots of the aggregated samples, and checking that they too are straight lines with the same slope as the original [19, 18]. If the distribution is not heavy tailed, the aggregated samples will tend to be Normally distributed (the more so as the level of aggregation increases), and the slopes of the LLCD plots will increase with the level of aggregation.

Using these and other procedures, the following have been argued to be heavy tailed:

– Process runtimes on general purpose workstations [51, 37]. Note that this only applies to the tail of the distribution, i.e. to processes longer than a certain threshold. Measurements show the power to be close to 1.

$$\Pr[T > t] = t^{-k}$$

model	tail	k
[51] ('86)	> 3s	1.05–1.25
[37] ('96)	> 1s	0.78–1.29

- File sizes on a general purpose system (Figure 3), again limited to the tail of the distribution. There has been some discussion on whether this is best modeled by a Pareto or a lognormal distribution, but at least some data sets seem to fit a Pareto model better, and in any case they are highly skewed [22].
- Various aspects of Internet traffic, specifically [62, 69]
  - Flow sizes
  - FTP data transfer sizes
  - TELNET packet interarrival times
- Various aspects of web server load, specifically [18, 6]
  - The tail of the distribution of file sizes on a server
  - The distribution of request sizes
  - The popularity of the different files (this is a Zipf distribution — see below)
  - The distribution of off times (between requests)
  - The distribution of the number of embedded references in a web page
- The popularity of items (e.g. pages on the web) is often found to follow Zipf's Law [77], which is also a power law [7]. Assume a set of items are ordered according to their popularity counts, i.e. according to how many times each was selected. Zipf's Law is that the count y is inversely proportional to the rank r according to

  $$y \approx r^{-b} \qquad\qquad b \approx 1 \tag{3}$$

  This means that there are r items with count larger than y, or

  $$\Pr[Y > y] = r/N \tag{4}$$

  where N is the total number of items. We can express r as a function of y by inverting the original expression (3), leading to $r \approx y^{-1/b}$; substituting this into (4) gives a power-law tail

  $$\Pr[Y > y] = C \cdot y^{-a}$$

  moreover, $b \approx 1$ implies $a \approx 1$ [2].

The problem with procedures such as plotting $\log \bar{F}(x)$ as a function of $\log x$ and measuring the slope of the line is that data regarding the tail is sparse by definition. When applying an automatic classification procedure, a single large sample may sway the decision is favor of "heavy". But is this the correct generalization? The question is one of identifying the nature of the underlying distribution, without having adequate data. Claiming a truly heavy tailed distribution is almost always unfounded, because such a claim means that unbounded samples should be expected as more and more samples are generated. In all real cases, samples must be bounded by some number (a process cannot run for longer than the uptime of the computer; a file cannot be larger than the total available disk space).

One simple option is to postulate a certain upper bound on the distribution, but this does not really solve the problem because the question of where to place the bound remains unanswered. Another option is to try fitting alternative distributions for which all moments converge. For example, there have been successful attempts to model file sizes using a lognormal distribution rather than a Pareto distribution [22]. This has the additional benefit of fitting the whole distribution rather than just the tail.

A more general approach is to use phase-type distributions, which employ a mixture of exponentials. Consider a simple example, in which N samples are drawn from an exponential distribution, and one additional sample is a far outlier. This can be modeled as a hyperexponential distribution, with probability $N/(N+1)$ to sample from the main exponential, and probability $1/(N+1)$ to sample from a second exponential distribution with a mean equal to the outlier value. In general, it is possible to construct mixtures of exponentials to fit any observed distribution [9]. This is especially important for analytical modeling, as distributions with infinite moments cause severe problems for such analysis. For simulation the exact definition is somewhat less important, as long as significant mass is concentrated in the tail.

4.2 The Phenomena of Self Similarity

Self similarity refers to situations in which a phenomenon has the same general characteristics at different scales [56, 67]. In particular, parts of the whole may be scaled-down copies of the whole, as in well known fractals such as the Cantor set and the Sierpiński triangle. In natural phenomena we cannot expect perfect copies of the whole, but we can expect the same statistical properties. A well known natural fractal is the coast of Britain [56]. Workloads often also display such behavior.

The first demonstrations of self similarity in computer workloads were for Internet traffic, and used a striking visual demonstration. A time series representing the number of packets transmitted during successive time units was recorded. At a fine granularity, i.e. when using small time unites, this was seen to be bursty. But the same bursty behavior persisted also when the time series was aggregated over several orders of magnitude, by using larger and larger time units. This contradicted the common Poisson model of packet arrivals, which predicted that the traffic should average out when aggregated.

Similar demonstrations have since been done for other types of workloads. Figure 5 gives an example from jobs arriving at a parallel supercomputer. Self similarity has also been shown in file systems [36] and in web usage [18].

The mathematical description of self similarity is based on the notion of long-range correlations. Actually, there are correlations at many different time scales: self similarity implies that the workload at a certain instant is similar to the

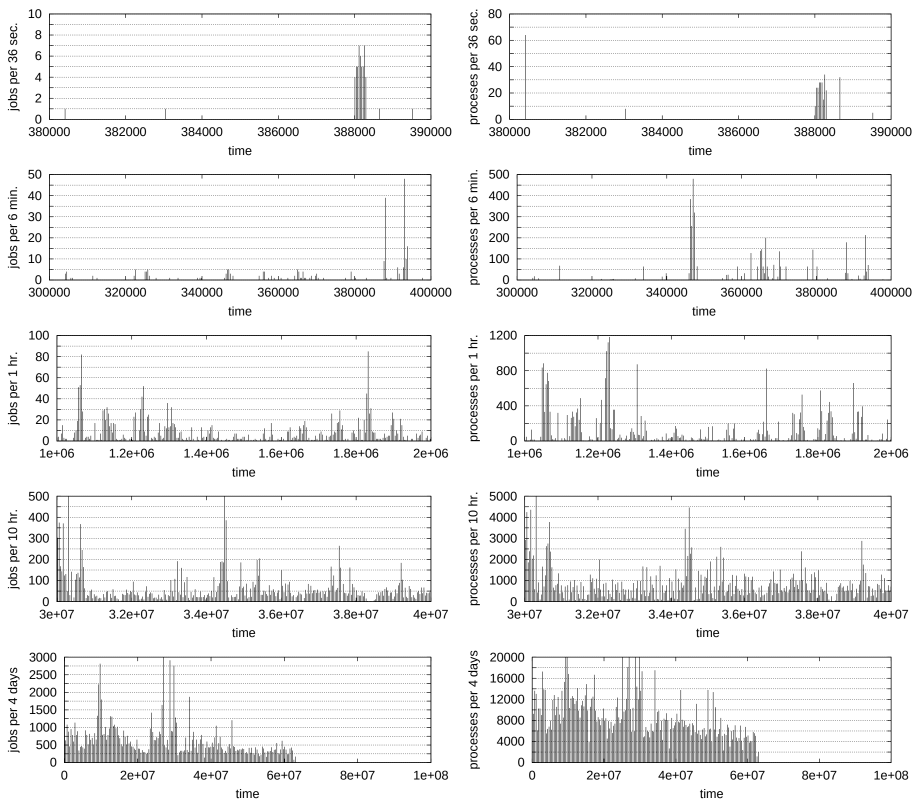


Fig. 5. *Burstiness of job arrivals to the SDSC Paragon parallel supercomputer at different time scales. Left: jobs per time unit. Right: processes per time unit (each parallel job is composed of multiple processes). In all the graphs time is in seconds; the duration of the log is two years, which is about 63 million seconds.*

workload at other instants at different scales, starting with a short time scale, through medium time scales, and up to long time scales. But the strength of the correlation decreases as a power law with the time scale.

A model useful for understanding the correlations leading to self similarity is provided by random walks. In a one-dimensional random walk, each step is either to the left or to the right with equal probabilities. It is well known that after n steps the expected distance from the origin is $\sqrt{n}$, or $n^{0.5}$. But what happens if the steps are correlated with each other? If each step has a probability higher than $\frac{1}{2}$ of being in the same direction as the previous step, we can expect slightly longer stretches of steps in the same direction. But this is not enough to change the expected distance from the origin after n steps — is stays $n^{0.5}$. This remains true also if each step is correlated with all previous steps with exponentially decreasing weights. In both these cases, the correlation only has a short range, and the effect of each step decays to zero very quickly.

But if a step is correlated with previous steps with polynomially decreasing weights, meaning that the weight of the step taken k steps back is proportional to k^{-a}, stretches of steps in the same direction become much longer. And the expected distance from the origin is found to behave like n^H, with $0.5 < H < 1$. H is called the Hurst parameter [63]. The closer it is to 1, the more self-similar the walk.

One way of checking whether a process is self similar is directly based on the above: measure the range covered after n steps, and check the exponent that relates it to n. Assume you start with a time series $x_1, x_2, \ldots$. The procedure is as follows [63]:

1. Normalize it subtracting the mean $\bar{x}$ from each sample, giving $z_i = x_i - \bar{x}$. The mean of the new series is obviously 0.
2. Calculate the distance covered after j steps:
$$y_j = \sum_{i=1}^{j} z_i$$
3. The range covered after n steps is the maximum distance that has occurred:
$$R_n = \max_{j=1\ldots n} y_j - \min_{j=1\ldots n} y_j$$
4. Rescale this by dividing by the standard deviation of the original data.
5. The model is that the rescaled range, R/s, should grow like cn^H. To check this take the log leading to
$$\log\left(\frac{R}{s}\right)_n = \log c + H \log n$$
If the process is indeed self similar, we expect to see a straight line, and the slope of the line gives H.

If a long time series is given, the calculation for small values of n is repeated for non-overlapping sub-series of length n each, and the average is used. An example of the results of doing so is given in Figure 6, based on the data shown graphically in Figure 5.

Other ways of checking for self similarity are based on the rate in which the variance decays as observations are aggregated, or on the decay of the spectral density, possibly using wavelet analysis [1]. Results of the Variance-time method are also shown in Figure 6. This is based on aggregating the original time series (that is, replacing each m consecutive values by their average) and calculating the variance of the new series. This decays polynomially with a rate of $-\beta$, leading to a straight line with this slope in log-log axes. The Hurst parameter is then given by
$$H = 1 - (\beta/2)$$

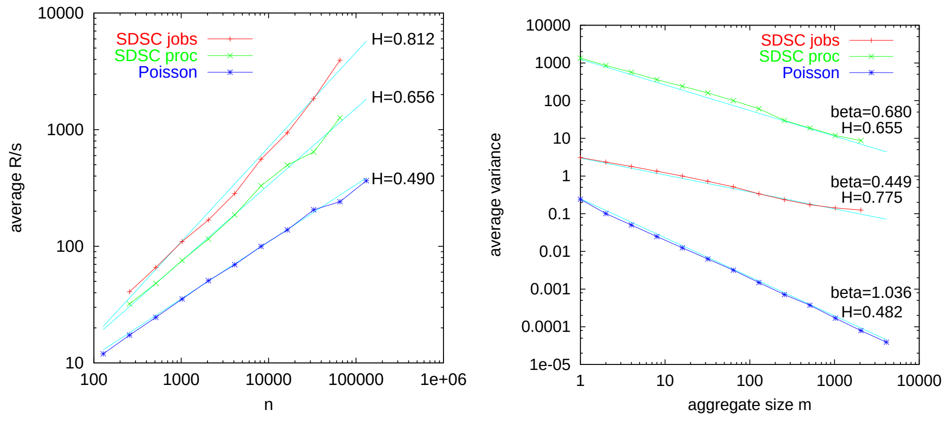


Fig. 6. *The* $(R/s)_n$ *and variance-time methods for measuring self similarity, applied to the data in Figure 5. A Poisson process with no self-similarity is included as reference, as well as linear regression lines.*

4.3 Modeling Self-Similarity

Heavy tailed distributions and self similarity are intimately tied to each other, and the modeling of self-similar workloads depends on this. As noted above, self similarity is a result of long-range correlation in the workload. By using heavy tailed distributions to create a workload model with the desired long range correlation, we get a model that also displays self similarity.

The idea is that the workload is not uniform, but rather generated by multiple on-off processes [75, 18, 36]. "On" periods are active periods, in which the workload arrives at the system at a certain rate (jobs per hour, packets per second, etc). "Off" periods are inactive periods during which no load is generated. The complete workload is the result of many such on-off processes.

The crux of the model is the distributions governing the lengths of the on and off periods. If these distributions are heavy tailed, we get long-range correlation: if a unit of work arrives at time t, similar units of work will continue to arrive for the duration d of the on period to which it belongs, leading to a correlation with subsequent times up to $t + d$. As this duration is heavy tailed, the correlation created by this burst will typically be for a short d; but occasionally a long on period will lead to a correlation over a long span of time. As many different bursts may be active at time t, what we actually get is a combination of such correlations for durations that correspond to the distribution of the on periods. But this is heavy tailed, so we get a correlation that decays polynomially — a long range dependence.

In some cases, this type of behavior is built in, and a direct result of the heavy tailed nature of certain workload parameters. For example, given that web server file sizes are heavy tailed, the distribution of service times will also be heavy tailed (as the time to serve a file is proportional to its size). During the time a file is served, data is transmitted at a constant rate. This is correlated with later transmittals according to the heavy-tailed distribution of sizes and transmission times, leading to long range correlation and self similarity [18].

5 Workload Dynamics and Structural Modeling

The on-off process used for modeling self-similar workloads has another very important benefit. It provides a mechanism for introducing locality into the workload, so that not only the statistics will be modeled, but also the dynamics.

5.1 User Behavior

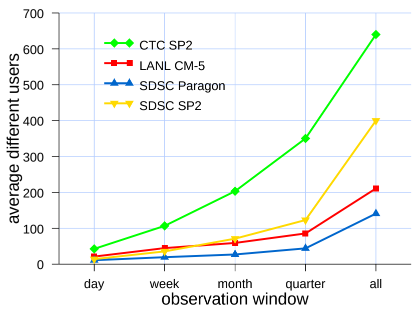
The procedure for workload modeling outlined in Section 3.2 was to analyze real workloads, recover distributions that characterize them, and then sample from these distributions. The main problem with this procedure is that is loses all structural information.

A real workload is not a random sampling from a distribution. For example, the load on a server used by students at a university changes from week to week, depending on the assignments that are due each time. In each week, everybody is working on the same task, so the workload is composed of many jobs that are statistically similar. The next week all the jobs are similar to each other again, but they are all different from the jobs of the previous week. Over the whole year we indeed observe a wide distribution with many job types, but at any given time we do not see a representative sampling of this distribution. Instead, we only see samples concentrated in a small part of the distribution (Figure 7). The workload displays a "locality of sampling"[3].

The common way to model workload dynamics is with a user behavior graph [31]. This is a graph whose nodes represent states. In each state, the user executes a certain job with characteristics drawn from a certain distribution. The arcs denote the probability of moving from state to state. The graph therefore encodes a Markovian model of the workload dynamics. A random walk on the graph, subject to the model's transition probabilities, creates a random workload sequence such that the probability of each job matches the limiting probability of that job's state, but it also abides by the model of which jobs come after each other, and how many times a job may be repeated (using self-looping arcs in the graph) [64]. However, this needs to be adjusted in order to create heavy tailed distributions.

In a university it may be plausible to argue that all students should be modeled using the same user behavior graph. But in a production environment one would expect different users, with different levels of activity and different behaviors. In addition, the active population changes with time (Figure 7) [23].

[3] The existence of such local repetitiveness in workloads was suggested to me by Larry Rudolph over ten years ago.



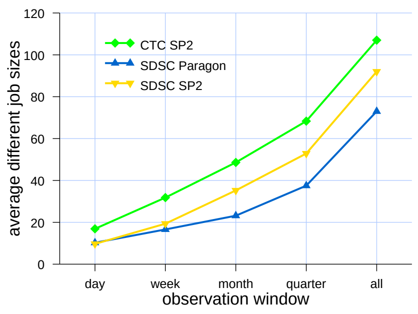


Fig. 7. *The dynamics of workloads. Left: the active set of users grows with the observation window. Right: so does the diversity of the workload, in this case represented by the number of different job sizes observed. Note that the x scale is not linear.*

Thus what we actually need is not one user behavior graph, but a model of the user population as a whole: how the population of users changes, and what user behavior graph each one should have. Using such a model has two important advantages. First, it has built-in support for generating self-similar workloads (assuming users have long-tailed on and off activity times). Second, it provides a good way to control load without modifying the underlying distributions: simply change the number of users [6].

Another aspect of user behavior, which is not captured by the user behavior graph, is the feedback from the system performance to the generation of new work. Real users are not oblivious to the system's behavior: They typically submit additional work only when existing work is finished. Thus, if the user population is bounded, the system's current performance modulates the offered load, automatically reducing it when congestion occurs, and spreading the load more evenly over time. But adding this integrates the workload model with the system, and prevents the use of an independent workload model.

5.2 Internal Structure

User modeling implants a structure on the workload. But it does not by itself define the basic building blocks of the workload — the jobs that are submitted to the system.

One approach is to use a descriptive model. For example, modeling of parallel applications requires a functional relationship between the number of processors and the runtimes — in short, a speedup function of the application. A model of speedups based on the average parallelism and its variance was proposed by Downey [21]. Another model, based on the parallel and sequential parts of the application and on the overheads of parallelization, was proposed by Sevcik [68].

An alternative is to model the application's internal structure. It is common practice to measure systems using parameterized synthetic applications [8].

Such applications typically involve several nested loops that mimic the behavior of iterative applications, and perform different amounts of computations, I/O operations, and memory accesses. The number of iterations, types of operations, and spread of addresses are all parameters, thus allowing a single simple and generic benchmark to mimic many different applications.

A similar approach can be used to generate a synthetic workload: use a parameterized program, selecting the parameters from suitable distributions in order to create the desired mix of behaviors. For example, Rudolph and Feitelson have proposed a model of parallel applications with relatively few parameters, including the total work done, the average size of work units and its variability, the way in which these work units are partitioned into threads, and the number of barriers by which they are synchronized [30].

The question is what distributions to use. While there has been some work done on characterizing specific applications [20, 74, 65], there has been little if any work on characterizing the mix of application characteristics in a typical workload. A rather singular example is the Charisma project, in which a whole workload was measured [59]. Interestingly, this requires the same statistical techniques described in Section 3.2, just applied to a different level. Indeed, such hierarchical structuring of workloads has been recognized as an important workload structuring tool [64].

Naturally, all this applies to practically all types of workloads, and not only to jobs on (parallel) machines. For example, web workloads can be viewed as sessions that each include a sequence of requests for pages that each have several embedded components; database workloads include transactions that contain a number of embedded database operations, and so on.

6 Conclusions

Performance evaluation depends on workload modeling. We have outlined the conceptual framework of such modeling, starting with simple statistical characterization, continuing with the handling of self similarity, and ending with the need to also model user behavior. But all this is useless without real measured data from which distributions and parameters can be learned. One of the most important tasks is to collect large amounts of high resolution data about the behavior of workloads, and to share this data to facilitate the creation of better workload models.

Apart from collecting data, there are also many methodological issues that beg for additional work. These include techniques to analyze and characterize workloads, evaluations of the relative importance of different workload parameters, and demonstrations of how workloads affect system performance. In all of these, emphasis should be placed on the dynamics of workloads. And as with the workload data, it is important to share the programs that perform the analysis and implement the models — both to facilitate the dissemination and use of new techniques, and to help ensure that researchers use compatible methodologies.

Acknowledgement

This research was supported by the Israel Science Foundation (grant no. 219/99).

References

1. P. Abry and D. Veitch, "*Wavelet analysis of long-range-dependent traffic*". *IEEE Trans. Information Theory* **44(1)**, pp. 2–15, Jan 1998.
2. L. A. Adamic, "*Zipf, power-laws, and Pareto – a ranking tutorial*". 2000. http://www.hpl.hp.com/shl/papers/ranking/.
3. R. J. Adler, R. E. Feldman, and M. S. Taqqu (eds.), *A Practical Guide to Heavy Tails: Statistical Techniques and Applications.* Birkhäuser, 1998.
4. A. K. Agrawala, J. M. Mohr, and R. M. Bryant, "*An approach to the workload characterization problem*". *Computer* **9(6)**, pp. 18–32, Jun 1976.
5. M. F. Arlitt and C. L. Williamson, "*Web server workload characterization: the search for invariants*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 126–137, May 1996.
6. P. Barford and M. Crovella, "*Generating representative web workloads for network and server performance evaluation*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 151–160, Jun 1998.
7. L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "*Web caching and Zipf-like distributions: evidence and implications*". In *IEEE Infocom*, pp. 126–134, Mar 1999.
8. W. Buchholz, "*A synthetic job for measuring system performance*". *IBM Syst. J.* **8(4)**, pp. 309–318, 1969.
9. W. Bux and U. Herzog, "*The phase concept: approximation of measured data and perfrmance analysis*". In *Computer Performance*, K. M. Chandy and M. Reiser (eds.), pp. 23–38, North Holland, 1977.
10. M. Calzarossa, G. Haring, G. Kotsis, A. Merlo, and D. Tessera, "*A hierarchical approach to workload characterization for parallel systems*". In *High-Performance Computing and Networking*, pp. 102–109, Springer-Verlag, May 1995. Lect. Notes Comput. Sci. vol. 919.
11. M. Calzarossa, L. Massari, and D. Tessera, "*Workload characterization issues and methodologies*". In *Performance Evaluation: Origins and Directions*, G. Haring, C. Lindemann, and M. Reiser (eds.), pp. 459–482, Springer-Verlag, 2000. Lect. Notes Comput. Sci. vol. 1769.
12. M. Calzarossa and G. Serazzi, "*A characterization of the variation in time of workload arrival patterns*". *IEEE Trans. Comput.* **C-34(2)**, pp. 156–162, Feb 1985.
13. M. Calzarossa and G. Serazzi, "*Workload characterization: a survey*". *Proc. IEEE* **81(8)**, pp. 1136–1150, Aug 1993.
14. S-H. Chiang and M. K. Vernon, "*Characteristics of a large shared memory production workload*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 159–187, Springer Verlag, 2001. Lect. Notes Comput. Sci. vol. 2221.
15. W. Cirne and F. Berman, "*A comprehensive model of the supercomputer workload*". In 4th *Workshop on Workload Characterization*, Dec 2001.
16. E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, "*Approximation algorithms for bin-packing — an updated survey*". In *Algorithm Design for Computer Systems Design*, G. Ausiello, M. Lucertini, and P. Serafini (eds.), pp. 49–106, Springer-Verlag, 1984.

17. M. E. Crovella, "*Performance evaluation with heavy tailed distributions*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 1–10, Springer Verlag, 2001. Lect. Notes Comput. Sci. vol. 2221.
18. M. E. Crovella and A. Bestavros, "*Self-similarity in world wide web traffic: evidence and possible causes*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 160–169, May 1996.
19. M. E. Crovella and M. S. Taqqu, "*Estimating the heavy tail index from scaling properties*". *Methodology & Comput. in Applied Probability* **1(1)**, pp. 55–79, Jul 1999.
20. R. Cypher, A. Ho, S. Konstantinidou, and P. Messina, "*A quantitative study of parallel scientific applications with explicit communication*". *J. Supercomput.* **10(1)**, pp. 5–24, 1996.
21. A. B. Downey, "*A parallel workload model and its implications for processor allocation*". In 6th *Intl. Symp. High Performance Distributed Comput.*, Aug 1997.
22. A. B. Downey, "*The structural cause of file size distributions*". In 9th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, Aug 2001.
23. A. B. Downey and D. G. Feitelson, "*The elusive goal of workload characterization*". *Performance Evaluation Rev.* **26(4)**, pp. 14–29, Mar 1999.
24. A. Erramilli, U. Narayan, and W. Willinger, "*Experimental queueing analysis with long-range dependent packet traffic*". *IEEE/ACM Trans. Networking* **4(2)**, pp. 209–223, Apr 1996.
25. D. G. Feitelson, *Analyzing the Root Causes of Performance Evaluation Results.* Technical Report 2002–4, School of Computer Science and Engineering, Hebrew University, Mar 2002.
26. D. G. Feitelson, "*The forgotten factor: facts*". In *EuroPar*, Springer-Verlag, Aug 2002. Lect. Notes Comput. Sci.
27. D. G. Feitelson, "*Memory usage in the LANL CM-5 workload*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 78–94, Springer Verlag, 1997. Lect. Notes Comput. Sci. vol. 1291.
28. D. G. Feitelson, "*Packing schemes for gang scheduling*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 89–110, Springer-Verlag, 1996. Lect. Notes Comput. Sci. vol. 1162.
29. 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*, D. G. Feitelson and L. Rudolph (eds.), pp. 337–360, Springer-Verlag, 1995. Lect. Notes Comput. Sci. vol. 949.
30. D. G. Feitelson and L. Rudolph, "*Metrics and benchmarking for parallel job scheduling*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 1–24, Springer-Verlag, 1998. Lect. Notes Comput. Sci. vol. 1459.
31. D. Ferrari, "*On the foundation of artificial workload design*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 8–14, Aug 1984.
32. D. Ferrari, "*Workload characterization and selection in computer performance measurement*". *Computer* **5(4)**, pp. 18–24, Jul/Aug 1972.
33. K. Ferschweiler, M. Calzarossa, C. Pancake, D. Tessera, and D. Keon, "*A community databank for performance tracefiles*". In *Euro PVM/MPI*, Y. Cotronis and J. Dongarra (eds.), pp. 233–240, Springer-Verlag, 2001. Lect. Notes Comput. Sci. vol. 2131.
34. R. Gibbons, "*A historical application profiler for use by parallel schedulers*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 58–77, Springer Verlag, 1997. Lect. Notes Comput. Sci. vol. 1291.

35. S. D. Gribble and E. A. Brewer, "*System design issues for internet middleware services: deductions from a large client trace*". In *Symp. Internet Technologies and Systems*, USENIX, Dec 1997.
36. S. D. Gribble, G. S. Manku, D. Roselli, E. A. Brewer, T. J. Gibson, and E. L. Miller, "*Self-similarity in file systems*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 141–150, Jun 1998.
37. M. Harchol-Balter and A. B. Downey, "*Exploiting process lifetime distributions for dynamic load balancing*". *ACM Trans. Comput. Syst.* **15(3)**, pp. 253–285, Aug 1997.
38. J. K. Hollingsworth, B. P. Miller, and J. Cargille, "*Dynamic program instrumentation for scalable performance tools*". In *Scalable High-Performance Comput. Conf.*, pp. 841–850, May 1994.
39. S. Hotovy, "*Workload evolution on the Cornell Theory Center IBM SP2*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 27–40, Springer-Verlag, 1996. Lect. Notes Comput. Sci. vol. 1162.
40. G. Irlam, "*Unix file size survey - 1993*". http://www.base.com/gordoni/ufs93.html.
41. J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, and J. Riodan, "*Modeling of workload in MPPs*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 95–116, Springer Verlag, 1997. Lect. Notes Comput. Sci. vol. 1291.
42. R. E. Kessler, M. D. Hill, and D. A. Wood, "*A comparison of trace-sampling techniques for multi-megabyte caches*". *IEEE Trans. Comput.* **43(6)**, pp. 664–675, Jun 1994.
43. D. N. Kimelman and T. A. Ngo, "*The RP3 program visualization environment*". *IBM J. Res. Dev.* **35(5/6)**, pp. 635–651, Sep/Nov 1991.
44. D. L. Kiskis and K. G. Shin, "*SWSL: a synthetic workload specification language for real-time systems*". *IEEE Trans. Softw. Eng.* **20(10)**, pp. 798–811, Oct 1994.
45. E. J. Koldinger, S. J. Eggers, and H. M. Levy, "*On the validity of trace-driven simulation for multiprocessors*". In 18th *Ann. Intl. Symp. Computer Architecture Conf. Proc.*, pp. 244–253, May 1991.
46. G. Kotsis, "*A systematic approach for workload modeling for parallel processing systems*". *Parallel Comput.* **22**, pp. 1771–1787, 1997.
47. P. Krueger, T-H. Lai, and V. A. Dixit-Radiya, "*Job scheduling is more important than processor allocation for hypercube computers*". *IEEE Trans. Parallel & Distributed Syst.* **5(5)**, pp. 488–497, May 1994.
48. M. Krunz and S. K. Tripathi, "*On the characterization of VBR MPEG streams*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 192–202, Jun 1997.
49. A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis.* McGraw Hill, 3rd ed., 2000.
50. E. D. Lazowska, "*The use of percentiles in modeling CPU service time distributions*". In *Computer Performance*, K. M. Chandy and M. Reiser (eds.), pp. 53–66, North-Holland, 1977.
51. W. E. Leland and T. J. Ott, "*Load-balancing heuristics and process behavior*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 54–69, 1986.
52. W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "*On the self-similar nature of Ethernet traffic*". *IEEE/ACM Trans. Networking* **2(1)**, pp. 1–15, Feb 1994.
53. V. Lo, J. Mache, and K. Windisch, "*A comparative study of real workload traces and synthetic workload models for parallel job scheduling*". In *Job Scheduling*

Strategies for Parallel Processing, D. G. Feitelson and L. Rudolph (eds.), pp. 25–46, Springer Verlag, 1998. Lect. Notes Comput. Sci. vol. 1459.

54. U. Lublin and D. G. Feitelson, *The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs.* Technical Report 2001-12, Hebrew University, Oct 2001.
55. A. D. Malony, D. A. Reed, and H. A. G. Wijshoff, "*Performance measurement intrusion and perturbation analysis*". *IEEE Trans. Parallel & Distributed Syst.* **3(4)**, pp. 433–450, Jul 1992.
56. B. B. Mandelbrot, *The Fractal Geometry of Nature.* W. H. Freeman and Co., 1982.
57. A. W. Mu'alem and D. G. Feitelson, "*Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling*". *IEEE Trans. Parallel & Distributed Syst.* **12(6)**, pp. 529–543, Jun 2001.
58. T. D. Nguyen, R. Vaswani, and J. Zahorjan, "*Parallel application characterization for multiprocessor scheduling policy design*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 175–199, Springer-Verlag, 1996. Lect. Notes Comput. Sci. vol. 1162.
59. N. Nieuwejaar, D. Kotz, A. Purakayastha, C. S. Ellis, and M. L. Best, "*File-access characteristics of parallel scientific workloads*". *IEEE Trans. Parallel & Distributed Syst.* **7(10)**, pp. 1075–1089, Oct 1996.
60. *Parallel workloads archive.* http://www.cs.huji.ac.il/labs/parallel/workload/.
61. K. Park and W. Willinger, "*Self-similar network traffic: an overview*". In *Self-Similar Network Traffic and Performance Evaluation*, K. Park and W. Willinger (eds.), pp. 1–38, John Wiley & Sons, 2000.
62. V. Paxon and S. Floyd, "*Wide-area traffic: the failure of Poisson modeling*". *IEEE/ACM Trans. Networking* **3(3)**, pp. 226–244, Jun 1995.
63. E. E. Peters, *Fractal Market Analysis.* John Wiley & Sons, 1994.
64. S. V. Raghavan, D. Vasukiammaiyar, and G. Haring, "*Generative workload models for a single server environment*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 118–127, May 1994.
65. E. Rosti, G. Serazzi, E. Smirni, and M. S. Squillante, "*Models of parallel applications with large computation and I/O requirements*". *IEEE Trans. Softw. Eng.* **28(3)**, pp. 286–307, Mar 2002.
66. R. V. Rubin, L. Rudolph, and D. Zernik, "*Debugging parallel programs in parallel*". In *Workshop on Parallel and Distributed Debugging*, pp. 216–225, SIGPLAN/SIGOPS, May 1988.
67. M. Schroeder, *Fractals, chaos, Power Laws.* W. H. Freeman and Co., 1991.
68. K. C. Sevcik, "*Application scheduling and processor allocation in multiprogrammed parallel processing systems*". *Performance Evaluation* **19(2-3)**, pp. 107–140, Mar 1994.
69. A. Shaikh, J. Rexford, and K. G. Shin, "*Load-sensitive routing of long-lived IP flows*". In *SIGCOMM*, pp. 215–226, Aug 1999.
70. A. Singh and Z. Segall, "*Synthetic workload generation for experimentation with multiprocessors*". In 3rd *Intl. Conf. Distributed Comput. Syst.*, pp. 778–785, Oct 1982.
71. D. Thiébaut, "*On the fractal dimension of computer programs and its application to the prediction of the cache miss ratio*". *IEEE Trans. Comput.* **38(7)**, pp. 1012–1026, Jul 1989.
72. D. Thiébaut, J. L. Wolf, and H. S. Stone, "*Synthetic traces for trace-driven simulation of cache memories*". *IEEE Trans. Comput.* **41(4)**, pp. 388–410, Apr 1992. (Corrected in *IEEE Trans. Comput.* **42(5)** p. 635, May 1993).

73. J. J. P. Tsai, K-Y. Fang, and H-Y. Chen, "*A noninvasive architecture to monitor real-time distributed systems*". *Computer* **23(3)**, pp. 11–23, Mar 1990.
74. J. S. Vetter and F. Mueller, "*Communication characteristics of large-scale scientific applications for contemporary cluster architectures*". In 16th *Intl. Parallel & Distributed Processing Symp.*, May 2002.
75. W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson, "*Self-similarity through high-variability: statistical analysis of Ethernet LAN traffic at the source level*". In *ACM SIGCOMM*, pp. 100–113, 1995.
76. K. Windisch, V. Lo, R. Moore, D. Feitelson, and B. Nitzberg, "*A comparison of workload traces from two production parallel machines*". In 6th *Symp. Frontiers Massively Parallel Comput.*, pp. 319–326, Oct 1996.
77. G. K. Zipf, *Human Behavior and the Principle of Least Effort.* Addison-Wesley, 1949.