Looking at Data

Dror G. Feitelson
School of Computer Science and Engineering
The Hebrew University of Jerusalem
91904 Jerusalem, Israel

Abstract

Collecting and analyzing data lies at the basis of the scientific method: findings about nature usher new ideas, and experimental results support or refute theories. All this is not very prevalent in computer science, possibly due to the fact that computer systems are man made, and not perceived as a natural phenomenon. But computer systems and their interactions with their users are actually complex enough to require objective observations and measurements. We'll survey several examples related to parallel and other systems, in which we attempt to further our understanding of architectural choices, system evaluation, and user behavior. In all the cases, the emphasis is not on heroic data collection efforts, but rather on a fresh look at existing data, and uncovering surprising, interesting, and useful information. Using such empirical information is necessary in order to ensure that systems and evaluations are relevant to the real world.

1. Introduction

Is computer science a science? One may regard this as a philosophical question, whose answer hinges on how one defines science as opposed to say mathematics or engineering. But this question also has a practical side. It is concerned with how we conduct research.

Mathematics is concerned with abstract thought. Engineering is concerned with building things that have a desired function. But science is concerned with learning about the world. Therefore science is built upon observation, measurement, and experimentation.

The above obviously relates to the natural sciences, and to learning about the natural world. But the same can apply equally well to the man-made world of computer systems. Computer systems from microprocessors to the Internet are complex enough so that even their designers cannot claim to have full understanding of how they work. To understand them, we need to observe them in action, and apply scientific methods. Looking at data is part of this.

The results surveyed here were mostly obtained by my students and myself. Naturally, there are many other examples too (e.g. [20, 3, 18, 16, 17, 19, 14, 1]). And I hope that in the future there will be much much more.

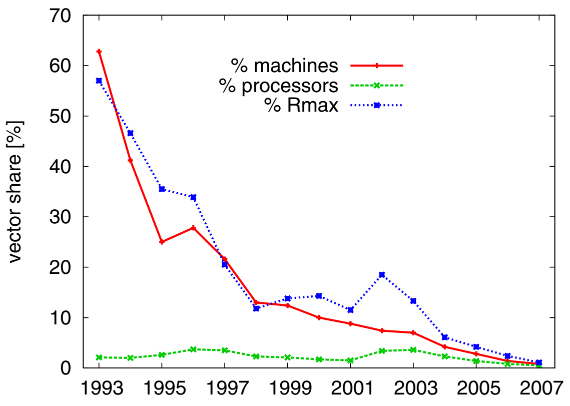


Figure 1. *Shares of vector machines in Top500 lists.*

2. Vector and Scalar Processors in the Top500 list

The title of this section may be surprising to some: aren't vector processors long dead? The answer is that they probably are now, but this is actually recent news. And in any case, it is interesting to look at how things changed over the last 15 years. This is done using the well-known Top500 list [5] (www.top500.org). This analysis extends previous ones from 1999 and 2004 [7, 9].

Fig. 1 shows the relative share of vector machines in the Top500 list. The graph shows that the fraction of vector machines in the Top500 list has steadily declined, from 2/3 of the first list in 1993 to less than 1% in the last list available at the time of writing (2007). However, the share of pro-

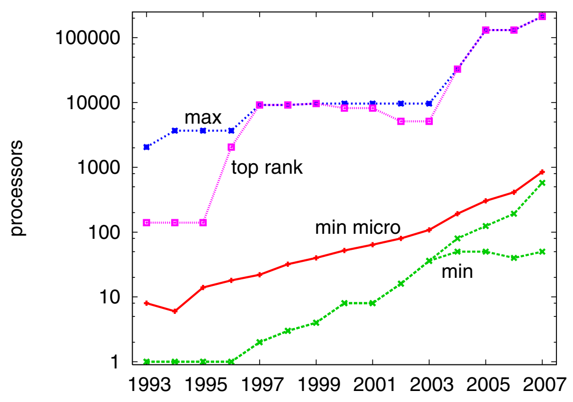


Figure 2. *Machine sizes in Top500 lists.*

cessors and computing power behave somewhat differently. Until recently, the share of processors (that is, what fraction of the total number of processors belonging to all machines on the list were vector processors) fluctuated in the range of 3–5%. And the share of Rmax (the fraction of computing power, as measured by Linpack, that is attributed to vector processors) showed remarkable stability from 1998 to 2003, even peaking at 2002 with the introduction of the Japanese Earth Simulator. This persistence, coupled with the introduction of the Cray X1 line, could be taken as an indication that vector processors would continue to have their niche. But continued data from 2004 and on indicates that this is not the case. So what happened?

Part of the answer may be seen in Fig. 2. Let's focus on the "min" and "min micro" lines. "min micro" indicates the machine with the minimal number of microprocessor-based processors in each year's list. "min" indicates the machine with the minimal number of processors, regardless of technology. Until 2003, this was always a vector processor. Since 2004 the minimal number is achieved by the Hitachi SR11000 machines, where each processor is actually a tightly-coupled, SMP-like aggregate of IBM Power4 processors; these are shown as the bottom branch of the line. The top branch shows the minimal vector-based machine and clearly continues the previous trend. As a result, the gap in favor of vector machines (that is, their ability to achieve equivalent performance with fewer processors) has all but closed. So indeed there is little reason to continue using vector processors (assuming you believe in using Linpack as the benchmark).

There are two other interesting things that can be seen in this graph too. The first is the change in slope of the "min micro" line in 2003. Until 2003, the slope was extremely steady, with the number of processors doubling about every 3 years. Since 2003, the slope has increased. This corresponds to the slower increase in computing power of microprocessors since that time. In particular, it seems that the supercomputer industry continues to deliver the same high rapid increase in performance as it did before, and makes up for the reduced improvements in microprocessors by using ever more of them.

The second interesting feature is the absolutely flat segment of the "max" line from 1997 to 2003. In an industry where practically everything grows exponentially, such an upper limit that persists for 7 years is very rare. The most probable interpretation is the difficulty in managing and utilizing more than 10,000 processors. While this limit was broken in 2004 and the current top machine employs more than 200,000 processors, this is achieved by bundling them into large groups that act as the unit of allocation and management. This may lead to significant unmonitored loss of resources if applications fail to use their full allocation.

3. Parallel Workload Patterns and Their Effect

How many processors do parallel applications actually use? Most application writers will tell you that there are few if any constraints, and that they will be able to use many processors provided the input problem size is big enough [2]. But in practice we find that actual usage is rather restricted.

The data shown in Fig. 3 is based on accounting logs from several large scale parallel machines (we use "cleaned" versions from the Parallel Workloads Archive at www.cs.huji.ac.il/labs/parallel/workload/). Two features immediately stand out. First, the distribution of job sizes is discrete, with a strong preference for powers of two. Second, there are very many serial jobs running on these massively parallel machines. This implies that buying a large-scale machine with a power-of-two number of nodes may be unwise, as this increases the danger for either of two unlucky situations: either that a small job lock out a large job that requires the whole machine, possibly leading to extensive fragmentation, or that a large job occupy the whole machine, preventing any other job from running and thus leading to extensive delays. Rather, it would be more beneficial to buy a machine that is a bit bigger than a power of two, and use the small extra partition for the small jobs.

Two of the machines shown in Fig. 3 do not seem to have serial jobs, possibly implying better utilization. However, this is most probably an erroneous conclusion. The difference between these machines and the others is that they are composed of 8-processor SMP nodes. Thus even a serial job will be allocated 8 processors, inflating the reported utilization, but actually creating significant hidden fragmentation. Again, this implies that special treatment should be given to small jobs.

Another issue that is of importance for parallel job schedulers is the question of correlation between job sizes

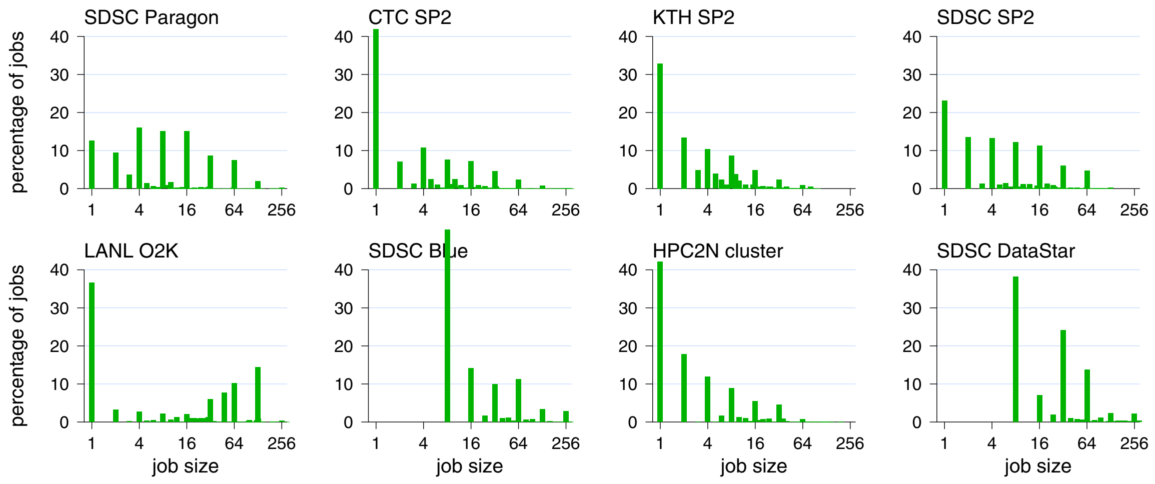


Figure 3. *Distribution of job sizes on various parallel machines.*

system	*CC*	*rank CC*	*dist CC*
NASA iPSC	0.157	0.242	0.884
LANL CM-5	0.178	0.293	0.986
SDSC Paragon	0.280	0.486	0.990
CTC SP2	0.057	0.244	0.892
KTH SP2	0.038	0.250	0.876
SDSC SP2	0.146	0.360	0.962
LANL O2K	-0.096	-0.214	-0.872
OSC cluster	0.021	0.212	0.869
SDSC Blue	0.121	0.411	0.993
HPC2N cluster	-0.046	-0.061	-0.173
SDSC DataStar	-0.012	0.013	-0.208

Table 1. *Correlation coefficients of runtime and size for parallel jobs.*

and runtime. The reason that this is important is that a job's size is typically known when it is submitted, but its runtime is not. Depending on the correlation, scheduling according to job size may unintentionally lead to scheduling by runtime too. For example, if there is a positive correlation, scheduling small jobs first will automatically also tend to schedule short jobs first, leading to improved average response times. But what if the correlation is negative?

Using parallel workload data as above, we can try to answer this question by calculating the correlation coefficient between size and runtime. The results of doing so are shown in Table 1; the correlations are all quite low, with both signs, so it is hard to argue for any real correlation. But some more insight can be gained by using the distributional correlation coefficient [8]. The idea is to partition each data set into two, based on one parameter. For example, we can partition the jobs into small jobs (with a size of up to the median size) and large jobs (larger than the median). Then we compare the distributions of runtimes for these two groups of jobs. If one distribution dominates the other, a distributional correlation exists.

Like the calculation of the correlation coefficient, the results of using this procedure are mixed. On one hand, the distributional correlation coefficient tends to be much more decisive than other coefficients — it is often close to one, meaning that one distribution dominates the other for most of the domain. But still, this is not always the case, and worse, while positive coefficients are much more common, negative ones do appear. However, by looking at the distributions themselves as shown in Fig. 4, one can see that the LANL O2K and HPC2N systems have somewhat abnormal modal distributions, making them less convincing as a source for generalization.

The main implication of such findings is that a single rule does not apply, and workloads may be different from each other. This underscores the need for online adjustments to the characteristics of each individual installation, rather than hoping for a single configuration that is appropriate for all.

4. Cleaning the Data Before Use

A possible source of discrepancies like those shown above is "dirty" data — data that is tainted by abnormal activity that should not be used as generally representative.

Indeed, the HPC2N data provides a striking example of such abnormal activity. More than half of the activity in this log is actually generated by a single user. As shown in Fig. 5, this activity comes in two very long sequences of

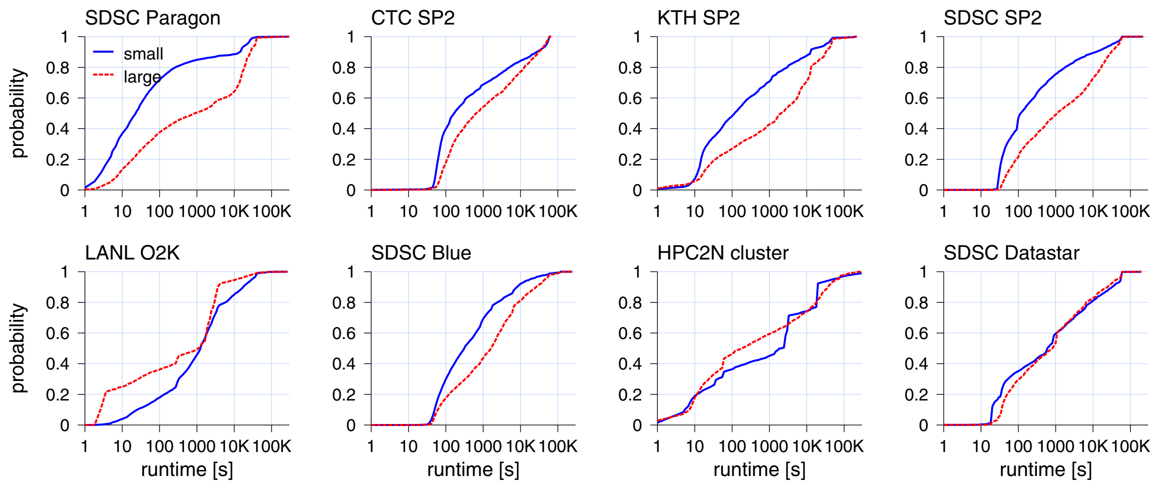


Figure 4. *Distributional correlation of job sizes and runtimes from different parallel machines, excluding serial jobs.*

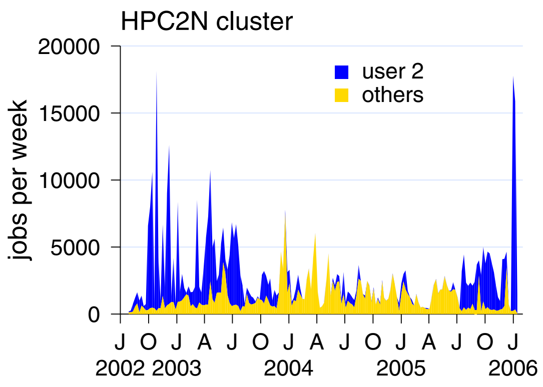


Figure 5. *Weekly activity on the HPC2N cluster, showing abnormal activity by user 2.*

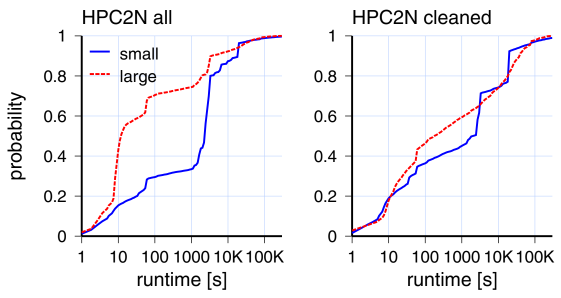


Figure 6. *Distributional correlation of HPC2N data, with and without user 2.*

bursts, that are separated by a year and a half of relatively low activity. The highest peaks of activity by this user reach some 18,000 jobs in a single week — more than one every minute on average for the whole week. The activity of all the other users of the system is much lower in comparison, seldom passing 2000 jobs per week in aggregate.

The problem with such abnormal activity is that it may have a large effect on workload characteristics. In our case, the activity of user 2 has a strong effect on the distributional correlation coefficient. Fig. 6 shows the runtime distributions for small and large jobs, both for the full log and for a cleaned log after removing the activity of user 2. Obviously much of the difference between the distributions may be attributed to the activity of this single user; with him, the distributional correlation coefficient is -0.915, and without him it drops to -0.173.

Other logs also exhibit such flurries of activity by single users, albeit usually they cover much shorter intervals of time. Apart from their effect on workload characteristics, such flurries have also been shown to have an effect on performance evaluations. This happens because a flurry is typically composed of many repeated executions of the same job. Therefore all these repetitions tend to react in the same way to modifications in the system configuration. As a result, these reactions are amplified to the degree that they may determine the evaluation results.

An example of such an effect is shown in Fig. 7. This evaluation concerns the scheduling of parallel jobs, essentially packing them together. To reduce fragmentation backfilling is used; this consists of using small jobs from the back of the queue to fill in holes in the schedule. This scheduling algorithm is evaluated by simulating its performance for the jobs in the CTC SP2 workload trace. The

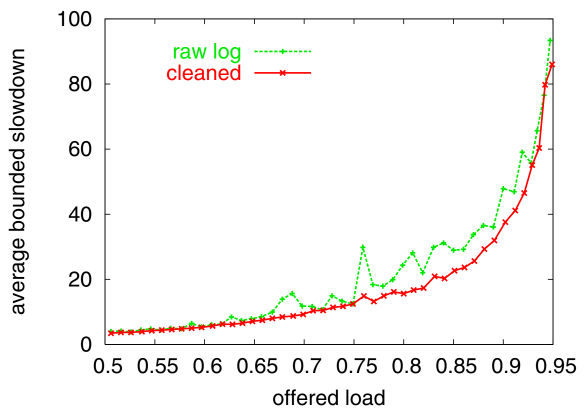


Figure 7. *Performance evaluation results for the CTC SP2 log with and without a flurry.*

interesting thing is that when we condense the log in order to increase the load, erratic behavior is observed [13]. This behavior results from a rather small flurry of only 2080 jobs. When this flurry is removed, scaling the load has the expected smooth effect.

An even more extreme example was found when simulating the same scheduling algorithm using the SDSC SP2 workload trace [25]. On this machine jobs were limited to running for 18 hours, and if they exceeded this limit, they were killed. But killing a jobs takes some time. In the simulation, a single job was truncated from 18 hours and 30 seconds to exactly 18 hours; amazingly, this caused the average bounded slowdown of all the simulated jobs to change by 8%. The effect was traced to a small flurry of 375 jobs that was submitted a full month after the job that was changed. Due to small changes that propagated in the schedule, this whole flurry was delayed by about 30 hours. As the jobs were actually very short, this caused a large change in the slowdown.

The question of when and whether data should be cleaned should not be taken lightly. Inevitably, this decision is not completely technical and includes a subjective element. Intuitively, we don't want the activity of a single user out of a group of hundreds of users to sway our results. But each user has some effect, so the question becomes what metric we use to measure the effect and where we put the threshold beyond which the effect is considered too big.

In any case, cleaning the data is just half the job. In addition, one should conduct a sensitivity analysis to characterize the magnitude of the effect that can be expected due to abnormal activities. This is an important element of the evaluation because we know for a fact that such abnormalities do occur.

5. The Behavior of Human Users

One of the examples in the previous section concerned evaluations in which the offered load is modified by systematically changing the job interarrival times. If we reduce these times, thereby condensing the log, the load increases beyond what it was on the live traced system.

This approach for creating different load conditions is convenient because of two properties: it is simple to apply, and it provides pretty good control over the resulting load. But it is based on a hidden assumption: that the system is open, in the sense that job arrivals are independent of each other and of the system performance. This assumptions may or may not be valid.

A major threat to the validity of the open system assumption is user feedback. If the workload is generated by a closed group of users, there is a high probability that these users will induce a feedback loop: if the system performance is poor, the users will abstain from producing additional work. This throttling effect is a form of negative feedback, and therefore improves system stability (Fig. 8) [21, 12].

In order to perform reliable system evaluations, we therefore need to model user behavior explicitly. But what do users care about? What governs their actions? In particular, what exactly do users perceive as "poor performance", and how does it translate into subsequent behavior?

One way to answer such questions is to conduct large scale user studies. A simpler alternative is to look at available data, and specifically, the same accounting traces we used before. The trick is to extract detailed user actions that can be attributed to specific conditions, rather than looking at overall statistics.

In particular, a basic question we may ask is whether users are more sensitive to a job's response time (the sum of waiting time and runtime) or to its slowdown (the response time normalized by the actual runtime). To answer this we can dissect the accounting logs, and compare each job's performance with the user's reaction. The user reaction is quantified by the think time separating the termination of this job and the submittal of the next job by the same user [22].

The result of doing so is shown in Fig. 9. The top graph shows the reaction to response times, and the bottom one to slowdowns. response times and slowdowns are binned on a logarithmic scale, and only think times of up to 8 hours are included. The results are that there is an obvious relationship between a job's response time and the subsequent think time: as the response time grows from 10 seconds to 3 hours, the average think time grows from about 20 minutes to about 3 hours, and this is very consistent across all the logs. But there is only a very weak relationship between a job's slowdown and the subsequent think time: as the slow-

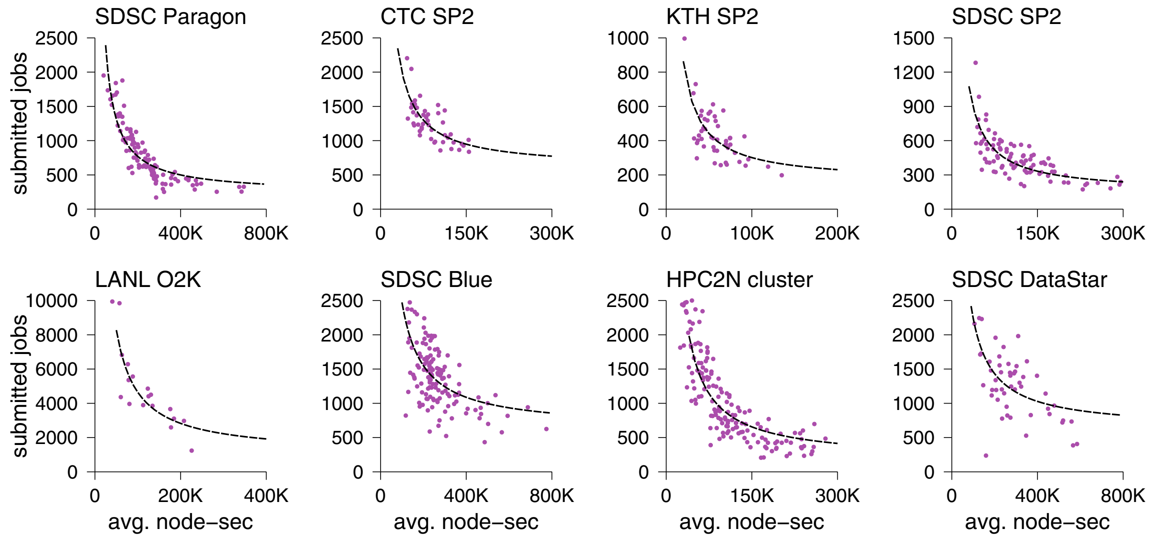


Figure 8. *Evidence of user throttling: there is an inverse relationship between the number of jobs submitted in a week and the average resource requirements of those jobs.*

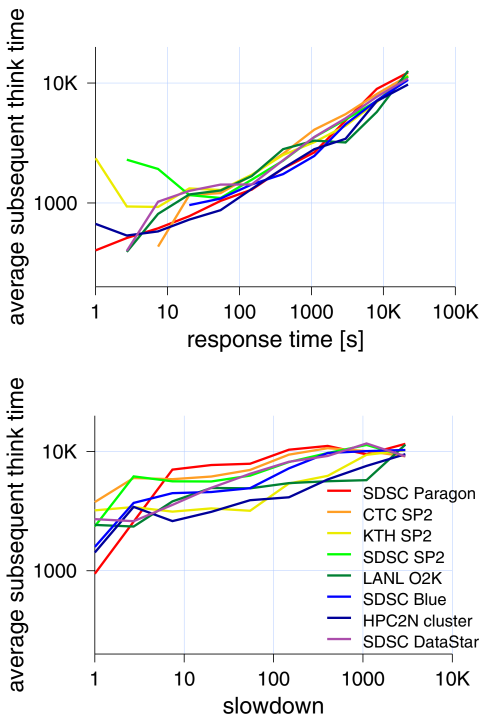


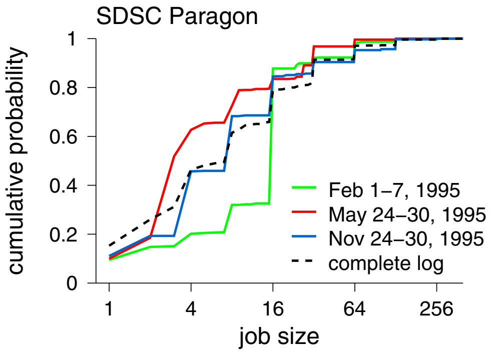
Figure 9. *Think time as a function of performance.*

down grows from 3 to 1000, the average think time grows from about 1–2$\frac{1}{2}$ hours to about 2–3 hours; only for very low slowdowns around 1 we see lower think times in some logs, but even then they are spread over the range of 20 minutes to about two hours. These results seem to indicate that response time is a much better predictor of user reaction than slowdown.

Being able to predict user behavior is important in many contexts. In fact, this is one of the main differences between online and offline algorithms — offline algorithms often achieve better performance by using more information, and specifically, information about the future. And while precise information is typically not available, the ability to make predictions may be good enough.

Luckily, it is often indeed the case that good predictions can be made. This results from the fact that most users tend to be unoriginal, and repeat the same sort of work over and over again. Moreover, they tend to use only a small part of the full space of different configurations and attributes that is available to them [23]. By looking at the historical record, the system may learn about this behavior and predict that it will continue [24]. Moreover, this happens at many levels, including both user behavior and application behavior — the well-known phenomenon of locality [4, 11].

But what does "locality" mean exactly in the context of user behavior? Using statistical workload models as a starting point, user behavior may be characterized by various distributions and the correlations between them. For example, in Section 3 we talked about parallel jobs that have



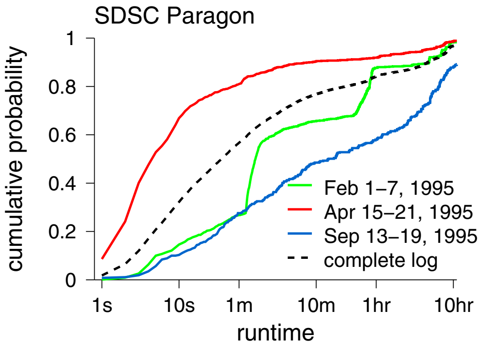
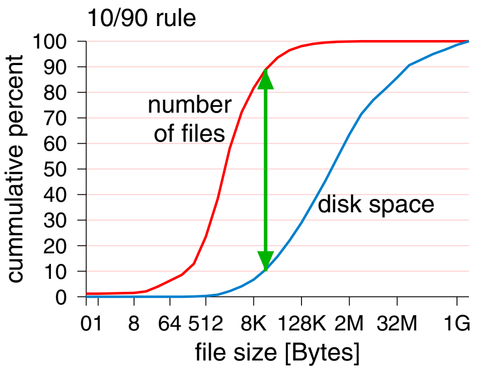


Figure 10. *Locality of sampling: in different weeks, the distribution of job sizes or runtimes is quite distinct from the global distribution of all the activity throughout the year.*



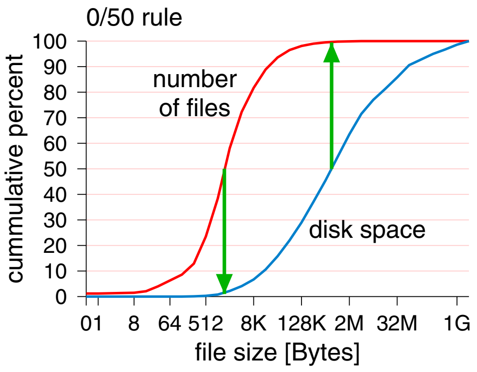


Figure 11. *Mass-count disparity demonstrated on Unix file sizes.*

two main attributes: their size and their runtime. The workload in general may be characterized by the distribution of sizes and the distribution of runtimes (and the correlation between size and runtime, which was one of the topics of Section 3). Locality in this context is manifest as a deviation from the general distribution, as demonstrated in Fig. 10.

Note that many analyses make the opposite assumption: that everything is random, independent, and stationary. For example, this is often the underlying assumption in queueing theory models. This of course makes the analysis considerably easier. However, in many cases it does not reflect reality. One extreme example that we saw above, in Section 4, is that of workload flurries. Flurries create large deviations in workload statistics, that are usually limited in duration. If we use a workload log that includes a flurry to characterize the marginal distributions of a statistical workload model, we get distributions that reflect the mixture of the normal workload and the flurry, and thus do not represent either of them. Such a model is useless, as it cannot be used to evaluate the system under normal conditions, and also cannot be used to evaluate the potential effects of the flurries.

An even worse result of ignoring locality is the inability to evaluate adaptive systems. The whole idea of adaptiveness is that the system adjust itself to match changing conditions. Changing conditions imply locality. If we use random models without locality, there is nothing to adapt to.

6. Sampling and Heavy Tails

Locality is related to the variability in the workload. Another potential characteristic that contributes to variability is heavy tailed distributions. When a workload attribute has such a distribution, some items have a much bigger effect than others. In fact, some items may actually dominate the system. This phenomenon is called mass-count disparity.

Mass-count disparity is a generalization of the proverbial 10/90 rule. Let us explain it through a demonstration based on Unix file sizes [15]. The bulk of the distribution, as shown in Fig. 11, lies between file sizes of about 100 bytes to 10 KB (“number of files” line). But the bulk of the disk space is occupied by files with sizes ranging from about 10 KB to some 100 MB! In particular, 90% of the files (the small ones) occupy only 10% of the disk space, while the other 10% of (large) files are responsible for the remaining 90% of the disk space (of course in other data sets the so-called joint ratio [10] can be different from 10/90, and even in this one it is actually closer to 11/89).

Even more remarkable is the 0/50 rule. The distribution is so skewed, that the bottom half of the files together use a negligible part of the disk space. At the other extreme, half the disk space is occupied by a negligible fraction of the files, which are each very big. This has important implications for file system design. For example, it means that optimizing the storage of very small files is a waste of time, because this will only have a small effect on total disk usage.

An especially interesting situation occurs when the distribution of popularity is heavy-tailed. For example, consider locality of reference in a program's memory accesses. Such locality is often divided into two types: spatial locality and temporal locality. Temporal locality, where the same address is accessed repeatedly, is actually also the product of two distinct phenomena. One is that accesses to a certain address are bunched together, rather than being distributed randomly throughout the execution of the program. The other (and more important) is that some addresses are simply much more popular than others. These addresses are accessed so many times that accesses are never far apart, leading to a semblance of locality,

In analogy with the file sizes example above, when the distribution of popularity is heavy tailed it also exhibits significant mass-count disparity. But here the mass is references to the different addresses. Thus we may find that say 90% of the addresses only receive 10% of the references, while the other 10% of addresses receive 90% of the references. And a relatively small number of addresses, the most popular ones, may be the targets of half of all the references.

The importance of mass-count disparity in the popularity distribution lies in its effect on sampling. Again, let's start with the files example. Selecting a file at random from the list of all files will most probably give us a small file, because most files are small — more than half are smaller than 2 KB, and 97% are smaller than 100 KB. but if we select a byte at random from all the data in the file system, we will most probably find that this byte is part of a very large file. Specifically, nearly half of the disk space belongs to files that are bigger than 1 MB in size, and these are only 0.2% of all the files.

Applying this observation to the popularity of memory addresses, we find that random sampling can effectively identify addresses with different characteristics. This has important implications for caching. Specifically, if we sample memory addresses at random, we are likely to select an address that is seldom accessed. This is why random replacement is a reasonably good cache eviction policy. But if we sample the reference stream, and pick a reference at random, it is much more likely that this is one of many references being made to a very popular address. So sampling references is a good cache insertion policy [6].

7. Conclusions

There's a story about an engineer, a physicist, and a mathematician that were stranded on a desert island with 3 cans of tomato soup but no opener. The engineer was the first to try to solve the problem of getting to the soup, by crushing his can with a big rock; the result was tomato soup all over the island. The physicist tried next, and put his can in a fire until it exploded; boiling tomato soup all over the island this time. "No, you're doing it all wrong", said the mathematician. "Here's how you do it. Assume you have a can opener..."

Undoubtedly the mathematician's solution is the most effective, the most technologically advanced, and the cleanest. It has only one drawback: it is divorced from the actual situation. Collecting and analyzing data is a good way to keep in touch with reality, and to maintain your relevance. Assumptions should be used sparingly, while data is used as the default.

As computer scientists our first step when confronted with a new problem is often to create an abstract version of the problem, that will allow us to focus on the essentials. But this involves a hidden assumption that we can recognize the essentials, and that the parts that we abstract away are indeed not important. Such assumptions should be verified empirically.

To understand your system you must first understand your data. To understand your data you must first have data. So collect data about your system. If you do it right, some of the data will most probably surprise you. Life is full of surprises. That's how progress is made. So keep an open mind.

Acknowledgments

Parts of this work were supported by the Israel Science Foundation (grant no. 167/03).

Most of the results reported here were achieved by my students, as can be seen from the citations made in the text. In particular, Dan Tsafrir, Edi Shmueli, and Yoav Etsion have all enriched me with insightful analyses and observations, for which I am grateful.

References

[1] S. Carmi, S. Havlin, S. Kirkpatrick, Y. Shavitt, and E. Shir, "*A model of Internet topology using k-shell decomposition*". *Proc. Natl. Acad. Sci. USA* **104(27)**, pp. 11150–11154, Jul 2007.

[2] W. Cirne and F. Berman, "*A model for moldable supercomputer jobs*". In 15th *Intl. Parallel & Distributed Processing Symp.*, Apr 2001.

[3] E. G. Coffman, Jr. and R. C. Wood, "*Interarrival statistics for time sharing systems*". *Comm. ACM* **9(7)**, pp. 500–503, Jul 1966.

[4] P. J. Denning, "*The locality principle*". *Comm. ACM* **48(7)**, pp. 19–24, Jul 2005.

[5] J. J. Dongarra, H. W. Meuer, H. D. Simon, and E. Strohmaier, "*Top500 supercomputer sites*". URL http://www.top500.org/. (updated every 6 months).

[6] Y. Etsion and D. G. Feitelson, "*L1 cache filtering through random selection of memory references*". In 16th *Intl. Conf. Parallel Arch. & Compilation Tech.*, pp. 235–244, Sep 2007.

[7] D. G. Feitelson, "*On the interpretation of Top500 data*". *Intl. J. High Performance Comput. Appl.* **13(2)**, pp. 146–153, Summer 1999.

[8] D. G. Feitelson, "*A distributional measure of correlation*". *InterStat*, Dec 2004. URL http://interstat.statjournals.net/.

[9] D. G. Feitelson, "*The supercomputer industry in light of the Top500 data*". *Comput. in Sci. & Eng.* **7(1)**, pp. 42–47, Jan/Feb 2005.

[10] D. G. Feitelson, "*Metrics for mass-count disparity*". In 14th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 61–68, Sep 2006.

[11] D. G. Feitelson, "*Locality of sampling and diversity in parallel system workloads*". In 21st *Intl. Conf. Supercomputing*, pp. 53–63, Jun 2007.

[12] D. G. Feitelson and A. W. Mu'alem, "*On the definition of "on-line" in job scheduling problems*". *SIGACT News* **36(1)**, pp. 122–131, Mar 2005.

[13] D. G. Feitelson and D. Tsafrir, "*Workload sanitation for performance evaluation*". In *IEEE Intl. Symp. Performance Analysis Syst. & Software*, pp. 221–230, Mar 2006.

[14] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "*Measurement, modeling, and analysis of a peer-to-peer file-sharing workload*". In 19th *Symp. Operating Systems Principles*, pp. 314–329, Oct 2003.

[15] G. Irlam, "*Unix file size survey - 1993*". URL http://www.gordoni.com/ufs93.html.

[16] 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.

[17] 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.

[18] J. K. Ousterhout, H. Da Costa, D. Harrison, J. A. Kunze, M. Kupfer, and J. G. Thompson, "*A trace-driven analysis of the UNIX 4.2 BSD file system*". In 10th *Symp. Operating Systems Principles*, pp. 15–24, Dec 1985.

[19] V. Paxson and S. Floyd, "*Wide-area traffic: the failure of Poisson modeling*". *IEEE/ACM Trans. Networking* **3(3)**, pp. 226–244, Jun 1995.

[20] R. F. Rosin, "*Determining a computing center environment*". *Comm. ACM* **8(7)**, pp. 465–468, Jul 1965.

[21] E. Shmueli and D. G. Feitelson, "*Using site-level modeling to evaluate the performance of parallel system schedulers*". In 14th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 167–176, Sep 2006.

[22] E. Shmueli and D. G. Feitelson, "*Uncovering the effect of system performance on user behavior from traces of parallel systems*". In 15th *Modeling, Anal. & Simulation of Comput. & Telecomm. Syst.*, pp. 274–280, Oct 2007.

[23] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "*Modeling user runtime estimates*". In *Job Scheduling Strategies for Parallel Processing*, pp. 1–35, Springer Verlag, 2005. Lect. Notes Comput. Sci. vol. 3834.

[24] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "*Backfilling using system-generated predictions rather than user runtime estimates*". *IEEE Trans. Parallel & Distributed Syst.* **18(6)**, pp. 789–803, Jun 2007.

[25] D. Tsafrir and D. G. Feitelson, "*Instability in parallel job scheduling simulation: the role of workload flurries*". In 20th *Intl. Parallel & Distributed Processing Symp.*, Apr 2006.