A Critique of ESP

Dror G. Feitelson

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

Abstract. This note is an elaboratin of a panel presentation, and is meant as a constructive critique of ESP. It should be remembered that the bottom line is that ESP is a big step in an important direction — otherwise we wouldn't bother with this discussion...

1 Introduction

The evaluation of parallel systems is very important, mainly for costly acquisition decisions, and has therefore been practiced for a long time. However, such evaluations typically focus on the computational aspects of the system. They typically use a small set of benchmark applications, and measure the performance of these applications in isolation [2, 3, 7, 1]. The results reflect the performance of the processor, the memory hierarchy, the interconnection network, and the relationship between these factors.

ESP is different — it targets the *system-level* performance rather than the hardware [8]. Issues include the efficiency of the scheduling, its flexibility, and mundane details such as booting time. While this is a very welcome shift in focus, there are some potential problems that have to be addressed. The purpose of this note is to point them out.

2 Good Points

The most important point in ESP is the objective of including the system in the evaluation. This should not be underestimated. Many large-scale parallel systems, costing tens of millions of dollars, suffer from very low utilization (e.g. [6]). This is at least partly due to the fact that vendors emphasize single-job performance, and that is where they invest most of their development effort. It is high time that system performance receive similar treatment.

The way to measure system performance is to measure how the system performs under a representative workload. The choice of workload is crucial, as different workloads can lead to very different performance results. It seems that ESP has made a very reasonable choice in this respect. The proposed workload conforms to various features of workloads observed in production installations [4], including

- The distribution of job sizes, which is mostly powers of two, but not only
- The existence of a large variance in the distribution of runtimes
- The repetition of certain jobs

The size of the test — 82 jobs — is also a reasonable compromise between the desire to have enough jobs to exercise the system scheduler, and the need to complete the test in a reasonable time (a few hours).

3 Debatable Points

While ESP is in general a very promising system-level benchmark, the details of its definition contain some potential problems. Some of these originate from the attempt to bundle everything into a result that is expressed as a single number.

3.1 The arrival pattern

The ESP benchmark is composed of a set of 82 jobs. 80 of these jobs arrive in three batches 10 minutes apart. While the set of jobs is fixed, their division into these three batches is randomized. The other two jobs are so-called "full configuration jobs" that require all the processors in the system, and arrive later.

There are two problems with this arrival pattern. One is that essentially all the jobs arrive at the beginning, within 20 minutes of a test that takes several hours. In particular, there is a distinct possibility that towards the end of the test the system will start to drain and utilization will drop. In a real setting, where jobs continue to arrive, this would not happen.

The second problem is that the user feedback cycle is missing. In real systems, users often do not submit additional work until their previous work is done. This tends to automatically spread out the load, and reduces the risk that the system will saturate; it is part of the on-line nature of real scheduling work. EPS, on the other hand, is closer to off-line scheduling, with all jobs available (nearly) at the outset.

While these problems are very disturbing, there is no obvious way to solve them. Changing the arrival pattern so as to spread the jobs out throughout the test risks all sorts of interactions with the load, especially considering that the jobs have different runtimes on different platforms. A partial solution might be to ignore idle time at the end of the test, and not include it in the utilization measure. This at least reduces the effect of the system drainage towards the end. This idea is elaborated upon in Section 3.4.

3.2 Including booting the system

The ESP metric of efficiency calculates the useful computation time as a fraction of the total runtime including a system boot. This implies an expectation that the system will be booted every 82 jobs, which is unreasonable and puts too much emphasis on booting. It also gives vendors an oppportunity to improve their score significantly by reducing boot time, without any modifications to the scheduler, which is more important for normal use.

The correct way to incorporate booting time would be to estimate the MTTB — the mean time to boot. Systems with a high MTTB would add less of the boot time to the denominator of the metric formula. However, there is no easy way to estimate the MTTB of a system. It may therefore be preferable to leave booting time as a separate metric, rather than trying to incorporate it into the efficiency metric.

3.3 Features and restrictions

The definition of ESP makes special provisions for the full-configuration jobs: they are given higher priority, and must be run as soon as possible after being submitted. This favors computers with the capability to checkpoint current jobs, because they can then make room for the full configuration jobs immediately. Computers lacking this capability are forced to idle their nodes as they collect them when the current jobs terminate. Backfilling is deemed undesirable, as it may actually delay the high-priority full-configuration jobs (even if the conservative version is used). As a side issue, even if backfilling is used, it is not clear what runtime estimates should be given, as accurate estimates are not necessarily the best, but normal user estimates are worse [5].

While favoring machines with checkpoint and restart is reasonable for a system-level metric, the effect in this case may be too extreme. A more balanced approach would be to first run the workload with no special requirements, and allowing all the features that exist in the scheduler. Then a second test would be administered to see how well the system handles special requirements, such as the need to run certain jobs immediately. This would then be able to use specialized metrics, such as the waiting time of the high-priority jobs. With the current efficiency metric, the advantage of checkpointing jobs in order to run the full configuration jobs immediately only has a secondary effect on the score, and might have practically no effect if backfilling is used. A system without checkpointing or preemption that causes a high-priority job to wait should be penalized by more than some added idle time.

Finally, submitting the first full configuration job after all the others, and requiring the second to terminate within 90% of the test duration, are artificial mechanisms to ensure that the full configuration jobs are not placed at the ends. The cause of this problem is that the test length, only 82 jobs, may be too short. Alternative solutions are therefore to either enlarge the test, or use only a single full configuration job.

3.4 Calculating the score

The ESP efficiency metric is supposed to lie in the range $[0, 1]$, with a value of 1 representing the perfect score. However, due to the impossibility of a perfect packing, a score of 1 is unattainable (even if the booting time is not included).

In fact, the top possible score is unknown, partly because this also depends on the runtimes of the jobs on the measured system and on the randomized order in which they are submitted.

One way to ensure that a top score of 1 is attainable is to ignore idle time at the end of a schedule. Thus the denominator of the efficiency metric will not be $P \cdot T$, but rather $\sum_{i=1}^{P} T_{p_i}$, where T_{p_i} is is the time from the start of the test to the last instant in which processor p_i is used. With this definition, a largest-job-first algorithm that does not reuse any processor once it is idled can create a perfect packing, with all the idle time at the end. The problem is that this is a very wasteful schedule, and is based on avoiding any attempts for dense packing — exactly the opposite of what we want!

To prevent such situations, it is possible to use a predefined scheduling algorithm as the comparison point. For example, we can choose "most work first" (MWF), which sorts the jobs according to the product of their runtime and number of processors. Using this reference algorithm, we calculate the IAE (idle at end) time as follows:

$$IAE^{MWF} = \sum_{i=1}^{P} T^{MWF} - T_{p_i}^{MWF}$$

where T^{MWF} is the total time for the schedule generated by MWF, and $T_{p_i}^{MWF}$ is the last instant processor p_i is used under MWF. The efficiency metric of ESP is then calculated as

$$\frac{\sum_{j=1}^{82} p_j \cdot t_j}{P \cdot T - IAE^{MWF}}$$

i.e. the IAE time calculated for the reference algorithm is deducted from the denominator. This avoids the unfair penalty due to drainage at the end to some degree. while it may happen that a superb scheduling algorithm would create a dense packing leading to a metric larger than 1, this is not very probable, and even if it happens there is a good interpretation: the algorithm is better than MWF.

3.5 Using real applications

The question of using real vs. synthetic applications is perhaps the hardest to settle. Real applications have the advantage of being good for comprehensive system valuation, including the processor, the communications infrastructure, the memory hierarchy, and so on. They really evaluate how all the system components work together, and foster a very wide interpretation for the word "system".

However, due to the way in which they interact with the hardware, real applications are problematic if you want to evaluate the "system" in a more narrow interpretation, namely the operating system components such as the scheduler. For example, the runtimes of different jobs may be totally different on different platforms, and even the relations between the jobs (which is longer than the other) may change. Therefore schedulers on different platforms are actually faced with a different workload when scheduling the same real applications! If you want to compare *only the schedulers*, this will not do.

It seems that in the case of ESP the choice of using real applications is the correct one. This is so for two reasons:

- ESP is designed to evaluate full system performance. This indeed *includes* the scheduler, but is not *limited* to the scheduler. Thus if the scheduler on a certain platform benefits from the fact that certain jobs run faster on that platform, so be it — the platform is indeed better.
- Using synthetic applications requires the benchmark designer to give answers to many hard questions, that are implicitly hidden by the choice of job mix. These include
  1. What should the memory requirements of the jobs be?
  2. What degree of locality should they display?
  3. What should be the granularity of communications?
  4. What should be the patterns of communications?
  5. How much I/O should the applications perform?
  6. What should be the correlations among the above parameters?

  These questions should be answered based on a detailed workload analysis, which is extremely hard to do. No real data about these issues exists to date.

However, different installations may opt to use different sets of applications that are more representative of their local workload. If this happens, ESP will become more of a framework than a well-defined metric.

4 Conclusions

It seems that there are a number of points in the current ESP definition that should be addressed. However, this critique should not be understood as saying that ESP is bad. On the contrary, it is a very significant first step in a very important direction. It is just that the problem of how to evaluate a complete systems is a hard one, and cannot be expected to be solved in one step.

References

1. D. H. Bailey, E. Barszcz, L. Dagum, and H. D. Simon, "*NAS parallel benchmark results*". *IEEE Trans. Parallel & Distributed Syst.* **1(1)**, pp. 43–51, Feb 1993.
2. G. Cybenko, L. Kipp, L. Pointer, and D. Kuck, "*Supercomputer performance evaluation and the Perfect Benchmarks*". In *Intl. Conf. Supercomputing*, pp. 254–266, Jun 1990.
3. J. J. Dongarra, "*Performance of various computers using standard linear equations software*". *Comput. Arch. News* **18(1)**, pp. 17–31, Mar 1990.
4. A. B. Downey and D. G. Feitelson, "*The elusive goal of workload characterization*". *Perf. Eval. Rev.* **26(4)**, pp. 14–29, Mar 1999.
5. D. G. Feitelson and A. Mu'alem Weil, "*Utilization and predictability in scheduling the IBM SP2 with backfilling*". In 12th *Intl. Parallel Processing Symp.*, pp. 542–546, Apr 1998.

6. J. P. Jones and B. Nitzberg, "*Scheduling for parallel supercomputing: a historical perspective of achievable utilization*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), pp. 1–16, Springer-Verlag, 1999. Lect. Notes Comput. Sci. vol. 1659.
7. J. P. Singh, W-D. Weber, and A. Gupta, "*SPLASH: Stanford parallel applications for shared-memory*". *Comput. Arch. News* **20(1)**, pp. 5–44, Mar 1992.
8. A. Wong, L. Oliker, W. Kramer, T. Kaltz, and D. Bailey, "*System utilization benchmark on the Cray T3E and IBM SP2*". In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), p. 58–70, Springer Verlag, 2000. Lect. Notes Comput. Sci. vol. 1911.