ESP Benchmark: Default and Modified versions Description

January 3, 2011

0.1 Evaluation based upon ESP Synthetic Workload

Concerning the synthetic workloads, our main interest was to use a validated model that provides easy adaptation upon any size of cluster in order to facilitate our already complex real-scale experimentation procedure.

To evaluate the scheduling performance of Resource and Job Management Systems, we have adopted the Effective System Performance (ESP) model [2, 3]. This is a widely used model, which not only provides a synthetic workload but proposes a specific type of benchmark application that can be executed to occupy resources as simply as possible, without stressing the hardware.

The ESP [2] test was designed to provide a quantitative evaluation of launching and scheduling parameters of a resource and job management system. ESP benchmark is a composite measure that can evaluate the system via a single metric, which is the smallest elapsed execution time of a representative workload. Multiple ESP tests may be performed in order to adjust the various RJMS parameters that can influence the exectuion of the workload (like scheduling policy, or propagation algorithms) and to tune the system for optimal results.

As presented in [2], the overall design goals and choices for the Effective System Performance benchmark can be deduced to the following:

1. Complete independence from the hardware performance (like CPU speed) or compiler improvements on the executed application codes. This is addressed by proposing a simple MPI application with target run-time that can be fixed to a given value.

2. Ability for efficiency evaluation of different scheduling policies (like backfilling,preemption,etc) supported by the resource and job management systems. This is possible through the use of the same particular workload.

3. Ability for scalability evaluation of job scheduling and launching procedures. This is addressed by proposing a dynamically adjusted proportional job mix in order to use the benchmark and provide evaluations upon the same systems of different scales.

4. Capability of repetitions in order to evaluate the RJMS through its improvements over time.

The ESP test has been deliberately constructed to be processor-speed independent with low contention for shared resources (e.g. the file system) and a specific measure of scalability, stability and effectiveness of a system's RJMS. It runs a fixed number of parallel jobs through a manager of resources and jobs. Individually, the jobs have their elapsed run times set to a fixed target run time. The elapsed time of the total test is independent of the different hardware performance and is determined, to a large degree, by the efficiency of the scheduler and the overhead of launching parallel jobs. In ESP, there are 230 jobs derived from a list of 14 job types, which can be adjusted in a different proportional job mix, if needed. The test is stabilized to the number of cores by scaling the size of each job with the entire system size. Table 0.1 shows the job types with their relative size compared to the entire system, instance count and target run time. The ESP test includes two full configuration jobs, called Z-jobs in the test scripts, which are constructed to use the total number of available computational cores. The default ESP execution rules specify that the full configuration jobs cannot run at the beginning or the end of the test period and that no other job is permitted to start running in the interim between the submission of the Z job and its launch. In our case we have slightly simplified this last rule. This was made because only systems that allow preemption or checkpoint/restart would be able to respect this rule and our goal is to be able to use this method for a large spectrum of RJMS. There are two versions of the test the 'throughput' variant without the Z-jobs (228 jobs

Job Type	Fraction of Job Size relative to total system size	Count of the number of job instance	Target Run Time (Seconds)
A	0.03125	75	267
B	0.06250	9	322
C	0.50000	3	534
D	0.25000	3	616
E	0.50000	3	315
F	0.06250	9	1846
G	0.12500	6	1334
H	0.15820	6	1067
I	0.03125	24	1432
J	0.06250	24	725
K	0.09570	15	487
L	0.12500	36	366
M	0.25000	15	187
Z	1.00000	2	100
Total		230	

Table 0.1: ESP benchmarks synthetic workload characteristics [2] .

in total) and the 'multimode' variant where the Z-jobs are submitted at 2400 and 7200 seconds after the start of the test and after all other jobs A, ..., M have been submitted. In our experimentation we use only the 'multimode' variant and we give a higher priority to Z-jobs than the rest of them and let the system decide how these are going to be treated depending on its capabilities and supported parameters. The rules dictate that the first full configuration job is only submitted after a part of the workload has already been scheduled and is running. The jobs in the ESP suite is submitted to the RJMS in a pseudo-random order following a gaussian model of submission. They are separated into two blocks. The first block contains all jobs except the two job type Z jobs. After 40 minutes that the last normal job has been sent, the first Z-job is submitted, and after 2 hours the second Z job is submitted. No manual intervention is permitted once the test has been initiated.

The fractional-size is the size of the job as a fraction of total system size. For example, if the system under test has 10240 cores for computation, then the size of job-type F is 640 (= 0.06250 x 10240) cores. The ESP test can be applied to any system size and has been verified on 64, 512, 1024 2048, 6726 and 19,320 computational core systems. The specific value choices of job sizes fraction along with the jobs target run times were inspired according to the real workloads of a specific computing center [5]. The scientists wanted to construct a synthetic workload that will reflect the characteristics of their everyday workloads in order to make a better selection of the resource management software or the specific scheduling policy they needed to use for improvement of their system utilization. The initial version of the benchmark contained even specific applications from different branches of science to execute [5], whereas the second and current version ESP2 [2] proposed a slightly different model and the execution of a simple MPI application (section 0.2.2).

Given a total amount of work, a hypothetical absolute minimum time (T-BEST) can be computed by dividing the work by the system size. Regardless the system size this hypothetical absolute minimum time is T-BEST = 10,973 seconds (3 hours). T-BEST is independent of the total system size and the processing speed of the system. The Effectiveness ratio, as show in [2], is the time the test actually runs compared to the time the best packing solution indicates, which means T-BEST divided by the observed elapsed time of the ESP test as shown on 0.2, where `x` is the concurrency for test code `i` and `T` is the run time the code `i` runs.

$$E = \frac{T - BEST}{\sum_{i=1}^{I} x_i * T_i} \tag{0.1}$$

`E` is the metric of the ESP test which allows us to make performance evaluations of a singe RJMS by varying some of their parameters and perform comparisons between different RJMS. For increasingly efficient systems, the ratio approaches unity. The T-BEST is simply a convenient definition of a lower bound. It is not possible to obtain T-BEST in a real test even in the optimal case. Therefore, most attainable ESP ratios fall in the range of 0.6 - 0.9.

0.1.1 Modifiying ESP for evaluation of topology aware placement techniques: TOPO-ESP-NAS

The ESP benchmark can be used to compare systems of different scales, regardless the hardware characteristics. But in some cases we want to compare RJMS features (like topology aware placement techniques) using the same system and conditions. In this case the actual application performance and turnaround times of single jobs will provide variations on the final elapsed time of the whole workload. Limited by the fixed target run times of ESP model; the normal ESP benchmark does not allow us to evaluate features like these. Nevertheless, if we execute real applications with real (not-fixed) runtimes then different task topology placement techniques will lead to different application execution speeds, single jobs turnaround times and different whole workload elapsed time. This will finally reveal insights upon the efficiency of the topology aware placement techniques. This is because applications that are sensitive on communications will perform better and have faster turnaround times if placed upon resources that allow faster communication and this will show on the single jobs execution times which will lead to smaller ESP elapsed times. In particular, in our case, we substitute the use of the simple default MPI application of ESP (section 0.2.1) with a NAS MPI application (section 0.2.2) which is sensitive to communications. More details about the particularly adapted ESP benchmark called *TOPO-ESP-NAS* are given in the description of the actual experimentation in section 0.3.3.

Based on the effectiveness ratio of the default ESP benchmark the evaluation metric of the new proposed *TOPO-ESP-NAS* benchmark, is the time the test actually runs compared to the sum of the measured execution times of each job if all jobs where placed upon a selection of nodes that allow their best performance (Ideal Topology for every job).

$$ESPEfficiency = \frac{TheoreticDuration(IdealTopology)}{MeasuredDuration} \tag{0.2}$$

0.1.2 Modified ESP version with smaller execution elapsed time: Light-ESP

The theoretic ideal duration of ESP benchmark is approximately 11000 sec (3 hours). The real-practical duration is between 12000sec and 13500sec. ESP could be also usefull in a context of validating the correct function of the platform after a maintenance session and before returning into production. However, a lighter version of ESP is needed that will execute the workload in smaller elapsed time. For these purposes we have modified ESP by diminishing the individual jobs target runtimes in order to give a theoretic ideal duration of 915 sec (15 min) and a practical duration of about 20 min. The changed target run-times for each class of jobs are shown on table 0.2.

Job Type	Fraction of Job Size relative to total system size	Count of the number of job instance	Target Run Time (Seconds)
A	0.03125	75	22
B	0.06250	9	27
C	0.50000	3	45
D	0.25000	3	51
E	0.50000	3	26
F	0.06250	9	154
G	0.12500	6	111
H	0.15820	6	89
I	0.03125	24	119
J	0.06250	24	60
K	0.09570	15	41
L	0.12500	36	30
M	0.25000	15	15
Z	1.00000	2	20
Total		230	935

Table 0.2: ESP benchmarks synthetic workload characteristics [2] .

A particular detail that needs to be considered is the moment that the Z-jobs are going to be submitted. In the default ESP the 2 Z-jobs are submitted after 2400 and 7200 seconds respectively after the start of ESP. In order to respect the same proportions regarding the whole benchmark's duration in the Light-ESP version the 2 Z-jobs are submitted after 200 and 700 seconds after the start of ESP.

Furthermore the automatic submission of different jobs which is based on a gaussian distribution needs to be adapted to the new time proportions. In the default ESP benchmark the first 50 jobs are submitted without delay between them and after that each job is submitted with an interval from 0 to 30 seconds. In order to respect the new time proportions in the light ESP version this interval becomes from 0 to 3 seconds.

Some first results of *Light-ESP* benchmark upon a 64 cores cluster are shown on table 0.3.

SLURM NB cores	64
Average Jobs Wait time (sec)	290
Total Execution time (sec)	1282
Efficieny for backfill+preemption policy	72.9%

Table 0.3: Light-ESP benchmark results

0.1.3 Executed Applications

Sleep applications

The definition of sleep applications consists of using the simple unix command `sleep` followed by a number which represents the amount of time in seconds that a Core, CPU or Machine will stays idle. The main advantage of using sleep jobs is that they represent the simplest type of application with a predefined steady duration that can be performed in any number of nodes. However, due to their simplicity we can use them only for experiments that the final evaluation measures are not influenced by CPU, bandwidth, or memory stress, or generally when we don't care about computations overheads.

Synthetic applications

Synthetic applications represent a specific type of applications defined by characteristics which are based upon real applications profiles. They can be used for computations with a goal to stress specific parts of the system like CPU, memory, network or even I/O. Furthermore, the fact that they can be parametrized according to the users experimental needs makes them ideal for real-scale experimentation. Their drawback is that even if they implicate real computations they cannot capture the complexity of a real application. The synthetic applications used for the experimentations are parallel versions programmed with MPI message passing implementation and were selected to be the following:

- **NAS NPB3.3 benchmarks** The NAS parallel benchmarks [1], are widely used to evaluate the performance of parallel supercomputers. They exhibit mostly fine-grain exploitable parallelism and are almost all iterative, requiring multiple data exchanges between processes within each iteration. They consist of eight problems (five kernel and three simulated CFD applications) derived from important classes of aerophysics applications. The Benchmarks BT (Block Tri-diagonal), SP (Scalar Penta-diagonal) and LU (Lower-Upper Symmetric Gauss-Seidel) solve a discretized versions of the unsteady, compressible Navier-Stokes equations in three spatial dimensions.

  The NPB-3.3 MPI implementations of the benchmarks have good performance on multiple platforms and are considered as a reference implementation [1].

  Jaspal Subhlok et al. in [4] have made a valuable characterization of all different NAS benchmarks and we use this as a reference for selecting the right NAS benchmark for each experimentation according our needs for CPU, network or memory stress.

- **pchksum benchmark**

  This simple MPI application has been implemented to be used and fit the needs for ESP benchmark [3, 2]. It is a simple parallel application that performs MPI communications. Each task generates a random message string and computes a digest. Each tasks sends the random message and receives a similar message from the next one. The rank of tasks are randomly generated. With each received message, the task does a XOR operation onto a message buffer which initially contained its random message. This operation is continued until a predefined requested time. The next part, repeats the above operation but in reverse order of the tasks ranks. After the last XOR operation, the updated message should be exactly the same as the initial message. Or, equivalently their digests should match. If this is met then the job has been correctly executed.

[1] $http://www.nas.nasa.gov/Resources/Software/npb\_changes.html$

Bibliography

[1] David Bailey, Tim Harris, William Saphir, Rob van der Wijngaart, Alex Woo, and Maurice Yarrow. The NAS parallel benchmarks 2.0. Report NAS-95-020, Numerical Aerodynamic Simulation Facility, NASA Ames Research Center, Mail Stop T 27 A-1, Moffett Field, CA 94035-1000, USA, December 1995.

[2] William TC Kramer. *PERCU: A Holistic Method for Evaluating High Performance Computing Systems*. PhD thesis, EECS Department, University of California, Berkeley, Nov 2008.

[3] NERSC. Effective system performance (esp) benchmark. http://www.nersc.gov/projects/esp.php.

[4] Jaspal Subhlok, Shreenivasa Venkataramaiah, and Amitoj Singh. Characterizing nas benchmark performance on shared heterogeneous networks. In *IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium*, page 91, Washington, DC, USA, 2002. IEEE Computer Society.

[5] Adrian T. Wong, Leonid Oliker, William T. C. Kramer, Teresa L. Kaltz, and David H. Bailey. ESP: A system utilization benchmark. In ACM, editor, *SC2000: High Performance Networking and Computing. Dallas Convention Center, Dallas, TX, USA, November 4–10, 2000*, pages 52–52, pub-ACM:adr and pub-IEEE:adr, 2000. ACM Press and IEEE Computer Society Press.