Robust Mouldable Scheduling Using Application Benchmarking for Elastic Environments

Ibad Kureshi
University of Huddersfield
Queensgate, Huddersfield, HD1 3DH
United Kingdom
+44 (0) 148447 1855
i.kureshi@hud.ac.uk

Violeta Holmes
University of Huddersfield
Queensgate, Huddersfield
United Kingdom
+44 (0) 148447 3588
v.holmes@hud.ac.uk

David Cooke
University of Huddersfield
Queensgate, Huddersfield
United Kingdom
+44 (0) 148447 2703
d.j.cooke@hud.ac.uk

ABSTRACT

In this paper we present a framework for developing an intelligent job management and scheduling system that utilizes application specific benchmarks to mould jobs onto available resources. In an attempt to achieve the seemingly irreconcilable goals of maximum usage and minimum turnaround time this research aims to adapt an open-framework benchmarking scheme to supply information to a mouldable job scheduler. In a green IT obsessed world, hardware efficiency and usage of computer systems becomes essential. With an average computer rack consuming between 7 and 25 kW it is essential that resources be utilized in the most optimum way possible. Currently the batch schedulers employed to manage these multi-user multi-application environments are nothing more than match making and service level agreement (SLA) enforcing tools. These management systems rely on user prescribed parameters that can lead to over or under booking of compute resources. System administrators strive to get maximum "usage efficiency" from the systems by manual fine-tuning and restricting queues. Existing mouldable scheduling strategies utilize scalability characteristics, which are inherently 2-dimensional and cannot provide predictable scheduling information.

In this paper we have considered existing benchmarking schemes and tools, schedulers and scheduling strategies, and elastic computational environments. We are proposing a novel job management system that will extract performance characteristics of an application, with an associated dataset and workload, to devise optimal resource allocations and scheduling decisions. As we move towards an era where on-demand computing becomes the fifth utility, the end product from this research will cope with elastic computational environments.

Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: Organization and Design – *batch processing systems, distributed systems*

General Terms

Economics, Management, Performance

Keywords

Scheduler, mouldable, benchmarking, elastic system, cloud, HPC, batch processing, cluster, grid

BCI'12, September 16–20, 2012, Novi Sad, Serbia.

Local Proceedings also appeared in ISBN 978-86-7031-200-5, Faculty of Sciences, University of Novi Sad.

1. INTRODUCTION

Benchmarking Schemes have historically been used as marketing and administration tools. Some schemes like Standard Performance Evaluation Corporation (SPEC) [10] and Perfect Benchmark [8] used "real" applications with generic datasets to test a systems performance. This way a scientist looking for a cluster computer could ask questions such as "How well will my software run?" rather than "How many FLOPS can I get out of this system?" If adapted to include an API to plug in any software to benchmark and to pass results to other software, these toolkits can be used for purposes other than sales and marketing. Benchmarking schemes are inherently 2-dimentional as scalability and performance is calculated using only cores versus dataset sizes. To be truly useful for a scheduler a 3rd dimension – workload size needs to be factored in. If a job scheduler can get access to performance characteristic curves for every application on the system, optimal resource allocation and scheduling/queuing decisions can be made at submit time by the system rather than the user. This would further improve the performance of mouldable schedulers that currently follow the Downey model [9]. Along with the decision-making regarding resource allocation and scheduling, if the scheduler is able to collect a historic record of simulations by the particular users, then further optimization is possible. This would lead to better and safer utilization of the system. Currently AI is used in some decision making in mouldable schedulers. Given a user-inputted variance of resources required, the scheduler makes a decision on resource allocation by selecting from the available range. If the user supplied range is incorrect, the scheduler is powerless to adapt, and on a next run cannot learn from previous mistakes or successes.

This research aims to overcome shortcomings of the existing solutions, by proposing a design of a job scheduler that uses artificial intelligence techniques, heuristic information and application specific characteristics. Utilizing results from an adapted open-framework benchmarking scheme, the scheduler will be able to provide a mouldable and intelligent interface to a batch queuing system, helping to increase the usage efficiency of the system and provide a high quality of service to all end users. The gathered heuristic data will support scheduling decisions to optimize the resource allocation and the system utilization. This work will be further expanded to include elastic or even shared resource environments where the scheduler can expand the size of its world based on either financial or SLA driven decisions.

2. BACKGROUND

2.1 Benchmarking Schemes

Benchmarking Schemes, especially kernel benchmarks, are mostly utilised by manufacturers to sell their products. For many users of high-end High Performance Computer (HPC) systems, predicting how their code or application will run is of more pressing concern. Another aspect of evaluating machines is to attempt a performance prediction for a family of code. These predictions are based on the scalability models for the code. Efforts by Allen B. Downey, outlined in a report titled "A model for speedup of parallel programs" [9] have led to the establishment of what others have referred to as the Downey Model. The Downey model establishes "a family of speedup curves that are parameterized by the average parallelism of a program, A, and the variance in parallelism". The performance characteristic of a particular algorithm, like a CFD code, is modelled to predict the run time if the cluster size is changed but the dataset is kept the same. As this model utilizes observable and measurable program characteristics, like run times for a particular section of code and then creates an average, it at times falls short when attempting to calculate averages. If the timings are linear or near linear on any range of system size, for the particular dataset size, the model cannot calculate an average. There is no guarantee that at a different dataset size the algorithm keeps the same speed up profile. While the Downey model has laid the foundations for analysing the scalability of algorithms, it does not however answer the question "If I buy this system how fast will my particular code for my particular problem run?"

Further efforts, like that done by the San Diego Supercomputer Center [sic] [19], attempt to create a multidimensional benchmarking scheme which provides more information than Cycle-accurate models (similar to the Downey model, only more elaborate and detailed) and run times from real applications. Their approach "attempts to see how much of the factors that affect performance can be attributed to few parameters only adding complexity as needed to explain observed phenomena." This approach sits between kernel benchmarks and real world application testing. The authors are averse to benchmarking schemes that collect many kernel benchmarks and algorithm specific characteristics to generate a performance model. They have found that this (very accurate) method is not feasible to model full application packages on large-scale systems due to time and monetary considerations.

The natural evolution from kernel benchmarks and then algorithm performance modelling is real world application benchmarks. In their paper "Measuring High-Performance Computing with Real Applications" Sayeed et al. [17] make a case for the development of a benchmark that uses actual applications to grade systems. The paper is aimed at establishing an application-based benchmark to assess performance of machines to influence "buying time" decisions. Aside from the financial factors limiting the development of real world application benchmarks, there are also some technical limitations. The benchmarks discussed above do provide developers with figures to help steer improvement of the hardware infrastructure. Real applications are not suitable to do this. Fine-tuning of components in a system, to improve the performance of a particular application, is possible but if the system runs more than one application then this approach is detrimental. For manufacturers and hardware designers real applications are not the best yardstick as "today's real applications might not be tomorrow's." Propriety applications also cannot be fairly utilized as manufacturers and developers can work together thus tainting the results. Real application benchmarks appear to be in the same category as its predecessors like LINPACK – just marketing devices sometimes leading to fine tuning monitoring and measuring devices.

Work carried out by Simon et al. and Alam et al. [1] creates a foundation for benchmarking HPC Systems and new generation processors with real world applications. Their work outlines how best to use such benchmarks to assess the new hardware effectively before deployment. Alam et al. [1] work also includes bringing together kernel benchmarks and application specific benchmarking to create a complete picture of a systems performance.

For the purpose of assisting in the procurement of a new system, real application benchmarks are very effective. Our interest in real application benchmarks stems from the question "For a given application on a given system is it possible to create a metric of performance versus dataset and problem size?" Benchmarking suites like Perfect Benchmarks [8] and the Stanford Performance Evaluation Corporation (SPEC) [10] are two real application benchmarks that have withstood the test of time and are still up to date, unlike many others from that era. The SPEC benchmarks are a compilation of popular applications from each domain of scientific research with associated workloads and datasets. Further work has been carried out to include some level of kernel benchmarks, though these are limited to the application. When benchmarking, along with the turn-around-time (TaT) gathered by SPEC, add-on components also provide information about the inter-processor communication, instructions and flops per cycle and the I/O calls and performance. Thus meeting some of the shortfalls outlined above.

As a result of testing and evaluation of the above-mentioned benchmarking suites, it was concluded that while the benchmarks are able to provide information about an applications performance the benchmarks are lacking an "open-framework" to allow a user to put in his/her own workload and dataset into the mix. The benchmarks, probably due to their maturity, are rigid in nature.

2.2 Scheduling Techniques

2.2.1 Traditional Scheduling Strategies

In parallel environments a job scheduler has two parts to consider – the selection of the machine and the scheduling of the jobs over time. Within the selection strategies there are two constants: a number of nodes available in the execution environment and, at the time of calculation, the number of free or available nodes/end points/slots. A limited list of strategies a scheduler can follow for resource allocation is Biggest Free, Random, Best Fit and Equal Utilization [13].

While there are many popular algorithms for job scheduling First Come First Served (FCFS), Backfill (Conservative, Aggressive) and Pre-emptive are most commonly used [12, 13].

Schedulers can also take into account a jobs profile. There are four types of profiles of jobs; these are Rigid Jobs, Mouldable Jobs, Evolving Jobs and Malleable Jobs [11, 12].

Because most applications cannot accommodate Malleable jobs and current production level schedulers have their own offline mechanisms to handle Evolving jobs this paper will henceforth only address the requirements of Rigid and Mouldable jobs.

2.2.2 Mouldable Scheduling

To further expand the definition of a mouldable job given in Traditional Scheduling Strategies, a mouldable job is where the user provides some "recommendations" for resources required, and possibly a deadline for the job. This is against the concept of a rigid job where a user prescribes the requirements for the job and the scheduler attempts to match the parameters exactly, ignoring scheduling or other benefits, and fails if it cannot. The scheduler then allocates resources to best decrease the turnaround time (TaT) of the job. The quantity of allocated resources is adjusted based on a constraint to help maximize the throughput. This constraint can exist in the form of a variance provided by the user or in some cases, if available, the decision can be based on the scalability profiles of the application being run [4–6, 14, 16, 18, 20, 21].

These studies have shown that in comparison to rigid jobs in a first come first served queue with backfilling (aggressive or otherwise), mouldable jobs have lower average queued times. Under other scheduling strategies, mouldable schedulers almost always give the best result for the contradictory demands of lowest response times and highest utilization rates [5, 6, 14].

Where most algorithms rely on users giving a recommendation of required resources and then a variance, Srinivasan et al.'s [20] approach uses algorithm scalability characteristics. Adopting the Downey model as a framework, Srinivasan is able to show how letting the scheduler decide the variance, and then allocating the corresponding resources, improves the turn-around times for jobs in a FCFS queue with conservative backfilling and a Fair Use policy. The results also show improved usage efficiency and more robustness on the part of the system.

3. CURRENT SYSTEM & LIMITATIONS

The following section analyses the University of Huddersfield HPC cluster Eridani. Eridani is a 200-core Beowulf cluster that caters to all research groups within the University. Primarily used as an "entry" system, this machines' main aim is to introduce researchers to utilizing HPC in their research. This system services all e-Science research groups within the University. Quality of service is of the utmost importance as any lapse could result in driving away a research group. Being an entry system means this system gets a large workload from inexperienced users who are not fully aware of the nuances of fine-tuning a job submission script. The system utilizes TORQUE [7] as the batch queuing system and MAUI [15] as the scheduler.

3.1 Fair Use Policies

Quality of service, as has been observed, is not always the fastest throughput but the illusion that users jobs are running though rather than sitting in a queue. With multiple users and a plethora of applications, the system administrators formulate a fair use policy (FUP) that dictates the scheduling decisions made by the jobs management system. These fair-use policies tend to override any other scheduling strategies.

The first constraint on jobs comes from restricting *Job sizes.* When creating the fair use policy an administrator could make the following decision:

If the queue is full, no user should be allowed to run more than:

- 5 large sized jobs, or
- 10 medium sized jobs, or
- 100 small sized jobs at a time.

This way the system would never appear to be completely locked by a few users. Other users with jobs in a queue get the impression that the system is working on their simulations as well.

This brings in the need to define a *problem size*. Problem size for a system of this configuration can be quantified in two variables – resources required and time required. TORQUE has a switch (flag) in the job file, which allows for both to be defined by users. The users usually define resources but not time requirement. From the system administrator's perspective, it is important not to rely on the users to provide all the information required, hence default values need to be in place. Implementing the queues with the factors hard coded in the script will force users to select an appropriate queue. While this will not give the specific requirements accurately it will provide enough information for a job scheduler to make basic calculations.

Evidently, any queue requiring large resources or time will have a lower priority weighting. This weighting is also considered in the decisions making process by the scheduler, when planning out of order runs. Beyond these factors, if two jobs are in the same queue, no licensing factors are affect them, the submitting users have not manually specified a run time, and the submitting user has not exceeded his/her run quota, the jobs will be run on a first come first served basis.

3.2 Scheduling Strategies

As the user base increases and the profiles of applications on the system change proper scheduling based on the established fair use policy has become essential. To force all jobs to provide and adhere to a "ceiling" time multiple queues are deployed. Each queue has a different set of Min, Default and Max Resources and a different Max Wall Time. The queues themselves have a priority weighting, which is factored in when making scheduling decisions. Jobs requiring less resources and less time are given a higher priority as outlined in the table of queues below. (srt is for short; std is for standard; 'ul' is unlimited)

Table 1: A Sample Configuration of Typical Queue Values

Queue Name	Min. N/C	Default N/C	Max N/C	Default Run Time	Max Run Time	Weight
Serialsrt	1P	1P	1Nx4P	6 hr.	6 hr.	100
Serialstd	1P	1Nx4P	1Nx4P	48 hr.	48 hr.	70
Serialul	1Nx4P	1Nx4P	1Nx4P	14 d	416d	30
Parasrt	2Nx1P	2Nx4P	4Nx4P	12 hr.	12 hr.	50
Parastd	2Nx4P	2Nx4P	4Nx4P	48 hr.	48 hr.	40
Paraul	2Nx4P	2Nx4P	32Nx4P	14 d	416d	0

Once jobs are in the respective queues the MAUI scheduler makes the run time scheduling decisions by including clauses from the fair use policy (FUP). For the systems within the QGG the FUP stipulates that when the resources are full:

1. Only 10 jobs per user will run (counting those that were running before the system was full). If a user already has more than 10 running no more can be scheduled until the total running jobs by that user drop to 9,
2. Queue weightings will factor into the scheduling decision,
3. If two jobs exist in the same queue and have been submitted within +/- 1 hour of each other, then

whichever jobs resource requirement is met first will run. Otherwise it is first come first served,

4. If a job is de-prioritised in favour of another job, after 24 hours it will become the highest priority job. If multiple jobs meet this criteria then the jobs with the earliest submit time will be served first,
5. The de-prioritisation rule does not apply in the 'ul' queues if the user has requested more than the default upper time limit,
6. All other conflicts and scheduling decisions will utilise the first come first served rule.

3.3 Limitations

As the previous sections have shown, system administrators and IT managers are constantly trying to maintain a balance between system efficiency and quality of service. To meet the various requirements of the user community running jobs on the system, the system parameters have to be loosely configured to allow flexibility. To get the maximum efficiency out of a system it must be governed with stringent rules that would result in maximum utilization. With rules too lax - wide jobs get stuck leading to systems appearing to be idle. Also a single user can completely consume the system with their jobs. With rules too strict - large backlogs begin to be created as shorter and narrower jobs are preferred leading to sometimes never ending wait times. Users only see quality of service and find the system useful when the right balance is struck. This balance depends on the applications behaviour and characteristics.

As outlined in 2.2, scheduling techniques have some weaknesses that are exposed by the performance characteristic of a particular application or dataset. Whether the system is scheduled with an aggressive backfilling strategy or a FCFS policy, based on user prescribed resource parameters, the goals of the system administrator are not fully met. In the first case, as identified previously, under certain conditions the Quality of Service afforded to the end user drops, with wide jobs getting stuck; in the later case wide jobs can lock queues leading to poor system utilization. These conditions are further exasperated if the users prescribed parameters are incorrect. Incorrect parameters can lead to over booking – leading to under utilization and large queues, and under booking could lead to hardware failure.

Adopting a heuristic scheduler will make the scheduler intelligent and then the system would be able to correct itself in future runs. This configuration can also be broken if the user is using multiple classes of workloads and datasets for the same application. A user who has pre-processing, simulation and post-processing jobs will still need to be careful when classifying the job. Misclassification leads to bad heuristic information.

However, the benefits of a system that utilizes heuristic data and aggressive backfilling stand. These systems fail when there is a lack of information on application and dataset performance characteristics, and when they rely on the users to properly assess and reassess the jobs' needs as they change the problem.

4. PROPOSED SYSTEM

To overcome the above limitations we are proposing the job management system architecture as shown in Figure 1. This system will request from users the following information at submit time: an application name, job family name (simulation name), dataset size and workload size. The job management system includes not just the scheduler and batch queuing system but also an open-framework real-application benchmarking suite. The output of this suite will provide the workload manager with a better understanding of the capabilities of the underlying system. When installing a new applications on the systems, the system administrator will run the benchmarking suite and the results of the performance characteristics will be stored in the database.

The workload manager will use the benchmarking information to determine a range of optimum resources to allocate to the job. From the metadata it will also identify the estimated run times for each combination of allocated resources. The limits estimated from the application profile will determine the scheduling of the job and the eventual position of the job in the queue.

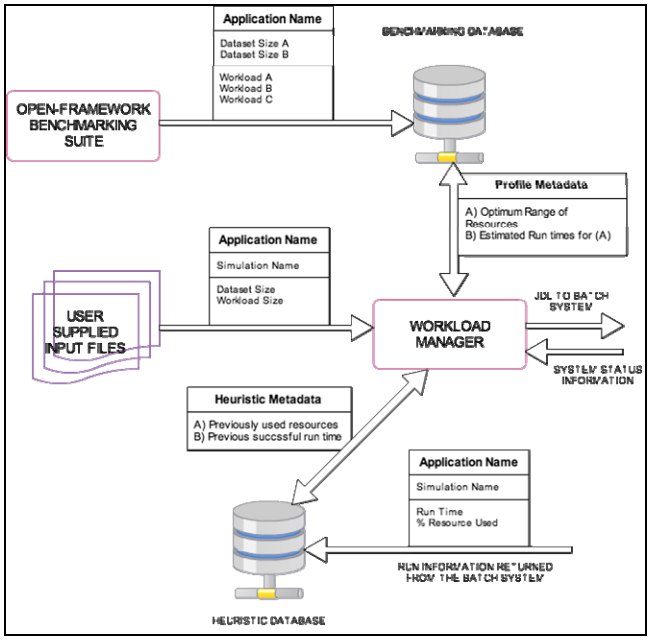

Figure 1: The Proposed Mouldable Scheduler based on Heuristic and Application Benchmarking information in a fixed execution environment.

To make a decision on the run-time allocation the workload manager takes into account two sources of information. The first source of information is the current system status. The second source is obtained from the heuristic database. This database is populated with run time information returned from the batch system log. The workload manager will use the simulation name provided by the user to compare it with the heuristic information, for fine-tuning the resource allocation.

AI techniques such as fuzzy logic and rules based systems will be used to arrive at the resource allocation decisions. All resource allocation decisions will be made keeping minimum Turn-around-Time as the primary goal and maximum system utilization as the secondary. With the estimated run time information also available the scheduler can provide the user with the time to completion at point of submission.

Though this proposed system architecture queries the user for initial job parameters, it is capable of recovering from bad input. Each user prescribed parameter has a counter check or a correction mechanism on future runs. The user has to provide the correct application name otherwise the job will fail immediately. If the user gives the wrong dataset or workload size, cumulative heuristic information will correct the resource allocation over

time. If a user incorrectly specifies the simulation name and the heuristic information is beyond the range specified by the application profiler, the system could auto-correct itself.

4.1 Open-Framework Benchmarking

To provide information of the application performance on a system we have designed a toolkit named Application and System Performance Characterization toolkit (ASPC) [2]. Built using Python and Bash script ASPC is itself very modular. The program, at the time of writing, has two wrappers that allow it to be used over Torque or SGE job management systems (see Figure 1 - OPEN-FRAMEWORK BENCHMARKING SUITE).

A system administrator needs to provide ASPC with a range of models for a particular application and corresponding workloads. In a configuration file the administrator needs to list the file names of the models, where the file name matches the variant variable between each model, and give an arbitrary classification of the model. Rather than using standardized test models given by manufacturers and developers, the administrator has the ability to plug in models that a typical to their own institutions work load. Figure 2 shows a sample configuration file for the application Fluent, developed by ANSYS. Here ASPC gets valuable metadata such as the fact that the users requirements will vary based on two factors – iterations and elements. Both these labels maybe specific to Fluent only, but it will make the job submission mechanism more native to the user. For the mouldable scheduler these are the two parameters against which it must search the database for adequate resource allocation information and run times for scheduling purposes. The additional 'classification' information only helps ASPC to narrow resource allocations for the benchmarking. For example, for the model classified as VVLARGE, ASPC will not attempt to test it against one node. Similarly ASPC will not test a VSMALL model against the whole system. This helps to save time when benchmarking as the number of combinations can be unmanageable in large systems.

The ranges that correspond to these 'classification' keywords are all set in a separate configuration file. Keeping inline with the philosophy of being an open-framework package, the administrator can set any words as keywords. Additional configuration files are available to pass any other parameters to ASPC that may be specific to the runtime environment, e.g. license information, library paths or environment modules.

Four nodes from the Eridani cluster were isolated to run ASPC for the results presented here. Each node has a 4-core 2.33Ghz Intel processor with 4GB of RAM. The nodes are configured as thin clients. Upon execution, ASPC generated the multiple jobs as described in Table 2: ASPC Job Breakdown for Benchmarking.

Once the jobs have terminated, ASPC goes through the job logs and determines runtime information and whether any job failed due to lack of resources. This information is then stored in a MySQL database for the Job Schedule module to call on (see Figure 1 – BENCHMARKING DATABASE). A visually representation of the results from the above performance characterization are shown in Figure 3.

From Figure 3 it can be seen that the LARGE dataset has only two data points. This is because when the LARGE dataset is run on 1 and 2 nodes it failed due to memory limitations. The other two datasets, VLARGE and VVLARGE could not run on the nodes utilised due to similar memory limitations. This data defines the lower limit for the system and can warn the user against overloading the nodes. This provides robustness in the system.

```
application: fluent

number-of-workloads: 2
workload-meta: iteration
workload: 5000
workload: 10000

number-of-models: 6
model-meta: elements

model: 312500
classification: VSMALL

model: 496190
classification: SMALL

model: 1014429
classification: AVERAGE

model: 2193408
classification: LARGE

model: 4093656
classification: VLARGE

model: 8109956
classification: VVLARGE
```

Figure 2: Sample of ASPC benchmark configuration file for the application fluent.

Table 2: ASPC Job Breakdown for Benchmarking

MODEL	WORKLOADS	MIN RES.	MAX RES.	NUMBER OF JOBS
VSMALL	2	1	1	2
SMALL	2	1	3	6
AVERAGE	2	1	4	8
LARGE	2	1	4	8
VLARGE	2	2	4	6
VVLARGE	2	4	4	2

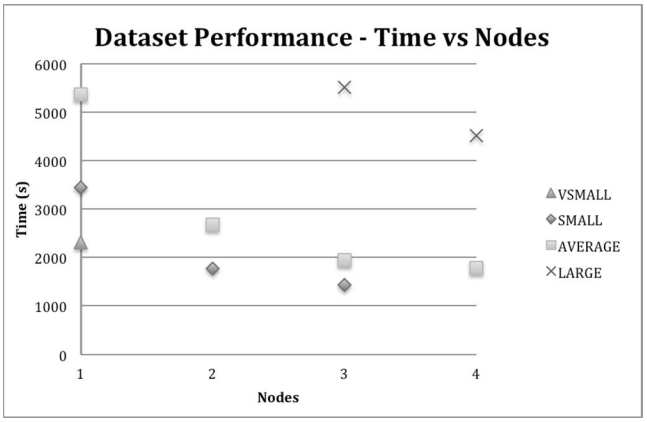

Figure 3: Performance curves for the different datasets.

4.2 Resource Allocation

With the information stored in the database the workload manager can begin to make resource allocation decisions. For the user this feature is a major shift from the traditional mechanism of submitting jobs. However this new process is more intuitive as the user needs to be concerned only about the parameters that are relevant to their domain. Figure 4 illustrates an example of a submission script where a user provides a job name, selects the application and defines relevant parameters. The parameters are based on the metadata provided when the benchmarks were done and the system was configured. The 'jobname' field is used to track this particular job and, on its termination, the heuristics module will use the job name with the run time information to generate heuristic records.

```
$ jobname=mytestjob
$ application=fluent
$ iteration=800
$ elements=800000
$ input=~/simulation/input.trn
```

Figure 4: Sample job file under the mouldable scheduler.

The workload manager will then decide on the resource allocation for the job based on the status of the system. To evaluate the mouldable allocation, currently the allocation component of the workload manager translates the above custom job file into a TORQUE specific job file as shown in Figure 5.

```
#!/bin/bash
#PBS -l nodes=3:ppn=4
#PBS -N mytestjob

module load fluent

cd $PBS_O_WORKDIR
fluent 3d -g -ssh -mpi=openmpi -t12 -cnf=$PBS_NODEFILE \
        -i ~/simulation/input.trn
```

Figure 5: Resulting TORQUE submission script.

In initial trials this system performed well. Decisions on resource allocation have been made opaque to the user. This makes the system more adaptable and robust.

5. CONCLUSION AND FURTHER WORK

In this paper we have presented a review of existing benchmarking schemes and scheduling techniques. We have identified their strengths and limitations and proposed a new robust mouldable scheduling system to address those limitations. In addition, we have considered mapping the mould-ability of jobs within the finite execution environment to elastic systems.

With distributed computing infrastructure turning towards "on demand" elastic systems, the job schedulers have to be adapted to cope with another changing variable. This variable, the size of the cluster, has always remained the constant 'K' in scheduling equations. While clouds are not generally utilized for HPC applications they are being used more and more for high throughput applications – especially in the commercial sector. Within the realm of academia, sites like the University of Huddersfield are moving towards shared resources, which are scalable to a certain degree. The UK Government's new approach to a three-tiered infrastructure for eScience computing [3] also sees many stakeholders coming together to utilize one system. The well-defined prior demand notification mechanism, offered by traditional service level agreements, cannot cope in such settings. This mechanism makes the same oversight as all computer workload schedulers do – they rely on the user to predict their needs for the next accounting cycle. In an academic world this put more stress on the "human workload manager" or the user himself. Consequently, system administrators are given more rescheduling tasks, as rules have to be adjusted and then restored to meet the scalability requests.

An "intelligent" scheduling systems that utilizes application performance data and a competitive algorithm, will be able to maximize the efficiency of the system as well provide the users with a quality of service. Adding a learning component, to help fine tune user specific application and dataset characteristics, the scheduler can be adapted to cope with elastic environment based on the load. This will enforce new types of SLA's without human intervention. These are the requirements of workload management systems as 'Supercomputing' moves into an era of austerity and a Green IT compliant computing systems.

6. ACKNOWLEDGMENTS

The authors would like to acknowledge the use of the University of Huddersfield computational grid known as the Queensgate Grid in carrying out this work.

7. REFERENCES

[1] Alarm, S.R., Barrett, R.F., Kuehn, J.A., Roth, P.C. and Vetter, J.S. 2006. Characterization of Scientific Workloads on Systems with Multi-Core Processors. *2006 IEEE International Symposium on Workload Characterization* (Oct. 2006), 225–236.

[2] ASPC Home: 2011. *http://hpc.hud.ac.uk/projects/ASPC/index.php/Main_Page*. Accessed: 2012-05-31.

[3] BIS 2011. *Delivering the UK's e-Infrastructure for Research and Innovation*. RCUK.

[4] Carroll, T.E. and Grosu, D. 2010. Incentive Compatible Online Scheduling of Malleable Parallel Jobs with Individual Deadlines. *Parallel Processing (ICPP), 2010 39th International Conference on* (2010), 516–524.

[5] Cirne, W. and Berman, F. 2001. A model for moldable supercomputer jobs. *Parallel and Distributed Processing Symposium., Proceedings 15th International* (2001), 8–pp.

[6] Cirne, W. and Berman, F. 2002. Using moldability to improve the performance of supercomputer jobs. *Journal of Parallel and Distributed Computing*. 62, 10 (2002), 1571–1601.

[7] Cluster Resources:: Products: *http://www.clusterresources.com/pages/products.php*. Accessed: 2012-02-07.

[8] Cybenko, G., Pointer, L. and Kuck, D. 1990. Supercomputer Performance Evaluation and the Perfect Benchmarks. *IN PROCEEDINGS OF THE 1990 ACM INTERNATIONAL CONFERENCE ON SUPERCOMPUTING*. (1990), 254–266.

[9] Downey, A.B. 1997. A model for speedup of parallel programs. *Computer*. (1997).

[10] Eigenmann, R. and Hassanzadeh, S. 1996. Benchmarking with real industrial applications: the SPEC High-Performance Group. *IEEE Computational Science & Engineering*. 3, 1 (Spring. 1996), 18–23.

[11] Feitelson, D. and Nitzberg, B. 1995. Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860. *Job Scheduling Strategies for Parallel Processing* (1995), 337–360.

[12] Feitelson, D., Rudolph, L., Schwiegelshohn, U., Sevcik, K. and Wong, P. 1997. Theory and practice in parallel job scheduling. *Job Scheduling Strategies for Parallel Processing* (1997), 1–34.

[13] Hamscher, V., Schwiegelshohn, U., Streit, A. and Yahyapour, R. Evaluation of Job-Scheduling Strategies for Grid Computing. *Grid Computing — GRID 2000*. R. Buyya and M. Baker, eds. Springer Berlin Heidelberg. 191–202.

[14] Hungershofer, J. 2004. On the combined scheduling of malleable and rigid jobs. *Computer Architecture and High Performance Computing, 2004. SBAC-PAD 2004. 16th Symposium on* (2004), 206–213.

[15] Jackson, D.B. 2001. Maui scheduler: a multifunction cluster scheduler. *Beowulf Cluster Computing with Windows* (2001), 345–362.

[16] Saule, E., Bozdağ, D. and Catalyurek, U. 2010. A moldable online scheduling algorithm and its application to parallel short sequence mapping. *Job Scheduling Strategies for Parallel Processing* (2010), 93–109.

[17] Sayeed, M., Hansang Bae, Yili Zheng, Armstrong, B., Eigenmann, R. and Saied, F. 2008. Measuring High-Performance Computing with Real Applications. *Computing in Science & Engineering*. 10, 4 (Aug. 2008), 60–70.

[18] Shih, P.C. and Chung, Y.C. 2009. Adaptive Processor Allocation for Moldable Jobs in Computational Grid. *International Journal of Grid and High Performance Computing*. 1, 1 (2009), 10–21.

[19] Snavely, A., Carrington, L., Wolter, N., Labarta, J., Badia, R. and Purkayastha, A. 2002. A Framework for Performance Modeling and Prediction. *Supercomputing, ACM/IEEE 2002 Conference* (Nov. 2002), 21–21.

[20] Srinivasan, S., Krishnamoorthy, S. and Sadayappan, P. 2003. A robust scheduling technology for moldable scheduling of parallel jobs. *2003 IEEE International Conference on Cluster Computing, 2003. Proceedings* (Dec. 2003), 92–99.

[21] Trystram, D. 2001. Scheduling parallel applications using malleable tasks on clusters. *Proceedings of the 15th International Parallel & Distributed Processing Symposium* (2001), 199.