Evaluation of Energy Consumption of Replicated Tasks in a Volunteer Computing Environment

A. Stephen McGough
School of Computing
Newcastle, UK
stephen.mcgough@newcastle.ac.uk

Matthew Forshaw
School of Computing
Newcastle, UK
matthew.forshaw@newcastle.ac.uk

ABSTRACT

High Throughput Computing allows workloads of many thousands of tasks to be performed efficiently over many distributed resources and frees the user from the laborious process of managing task deployment, execution and result collection. However, in many cases the High Throughput Computing system is comprised from volunteer computational resources where tasks may be evicted by the owner of the resource. This has two main disadvantages. First, tasks may take longer to run as they may require multiple deployments before finally obtaining enough time on a resource to complete. Second, the wasted computation time will lead to wasted energy. We may be able to reduce the effect of the first disadvantage here by submitting multiple replicas of the task and take the results from the first one to complete. This, though, could lead to a significant increase in energy consumption. Thus we desire to only ever submit the minimum number of replicas required to run the task in the allocated time whilst simultaneously minimising energy. In this work we evaluate the use of fixed replica counts and Reinforcement Learning on the proportion of task which fail to finish in a given time-frame and the energy consumed by the system.

CCS CONCEPTS

• **Computing methodologies** → *Sequential decision making*; *Simulation evaluation*; • **Hardware** → *Enterprise level and data centers power issues*;

KEYWORDS

Trace-Driven; Simulation; Machine Learning, Energy

ACM Reference Format:
A. Stephen McGough and Matthew Forshaw. 2018. Evaluation of Energy Consumption of Replicated Tasks in a Volunteer Computing Environment. In *ICPE '18: ACM/SPEC International Conference on Performance Engineering Companion , April 9–13, 2018, Berlin, Germany*. ACM, New York, NY, USA, Article 4, 6 pages. https://doi.org/10.1145/3185768.3186313

ICPE '18, April 9–13, 2018, Berlin, Germany

ACM ISBN 978-1-4503-5629-9/18/04…$15.00
https://doi.org/10.1145/3185768.3186313

1 INTRODUCTION

Our desire to perform high volumes of computational work often outstrips the capability of what a single computer can perform in an acceptable time-frame. Thankfully, many computational problems consist of repeated actions (referred to hereafter as tasks without loss of generality) which can be performed on many computational units (referred hereafter as resources) at the same time – parallel computing. If the resources require regular inter-communication this is referred to as fine grain parallelism. At the other extreme is course grained parallelism where tasks do not communicate at all – High Throughput Computing (HTC). Each task is a single run of a piece of software. A central service acts as master which hands out task to resources and collects the output on compleation.

High Throughput Computing is well suited to making use of heterogeneous resources as each task runs at its own pace. However, this can lead to complications. A resource may crash, or be lost, meaning that the task will never complete or the task may be allocated to a particularly slow resource causing significant delays. This problem is compounded further when the resources used are not owned by the task owner but are offered on a voluntary (best effort) basis by others – volunteer computing. The two most popular volunteer HTC environments are HTCondor [10], often used within an organisation where the resources (computers) are purchased for some other purpose, and BOINC [1], where the desire is to make use of resources (computers) provided by members of the general public. In both of these cases when a resource is required for its primary use the HTC system will have to relinquish the resource. This may require the termination, suspension or movement of the running task. In all cases leading to a delay in the completion time of the task and potentially extra energy consumption.

In many cases the users of a HTC system will be happy with the delay to their tasks – HTC systems focus on ensuring that tasks will eventually complete. However, these delays can be significant – in 2010 for the HTCondor system at Newcastle University 52.4% of the task executions encountered a delay greater then their own runtime. Efforts can be made to reduce the chance of a task being allocated to a resource which will be required for primary use [4, 12], however, it is not possible to determine this with certainty.

An alternative approach to identifying the resource most likely to be available for the duration of a task is to perform task replication. In this case a task will be replicated a number of times and each replica is submitted to the HTC system and treated like other tasks. However, on the completion of the first replica all other replicas are terminated. Increasing the probability of a task completing quickly, where naively it may be assumed that as the replication count increases so too does the probability of quick completion. The downside is twofold. In the first case the system can become

overloaded through the extra replicas, leading to fewer tasks finishing within their desired deadline. In the second case, the additional replicas, although terminated at the first replica completion, will consume resources and hence energy of the system.

Task replication has been used extensively for volunteer computing systems such as BOINC, albeit more for validating resources aren't providing invalid results. Here we seek to identify if more centralised HTC systems, with multiple concurrent task submitters, can make efficient use of task replication. In order to achieve this we have extended the HTC-Sim [4] simulation system with the ability to handle task replication. We have developed two forms of task replication: i) Fixed replication count, and, ii) Replication counts determined through Reinforcement Learning [14]. We use HTC-Sim to determine the practicality of using task replication both in terms of energy consumed and impact on the number of task which fail to complete within a QoS bound (time-frame).

The rest of this paper is set out as follows. In Section 2 we provide a background and motivation to the problem of task replication. We discuss related work in Section 3. Our Reinforcement Learning based Replication approach is discussed in detail in Section 4. We provide details of our experimental setup in Section 5 before providing simulation results in Section 6. We conclude the paper and present future directions in Section 7.

2 BACKGROUND AND MOTIVATION

We present an overview of HTC-Sim [4], along with the adaptations for task replication, before providing motivation for why task replication could be of benefit by analysis of a HTC system deployed at Newcastle University.

2.1 HTC-Sim

We have developed a general purpose HTC simulation system capable of modelling the behaviour of a HTC system based around a collection of both dedicated and volunteer resources. This system takes, as input, trace data from an existing system – representing both primary and HTC users. Interested readers in the full capability of this system are referred to previous papers, most notably [4].

Figure 1 illustrates the high-level view of HTC-Sim. Two types of user can interact with the system – High-Throughput users who provide descriptions of the tasks they want executing and Interactive users who can use the resources for their primary purpose.

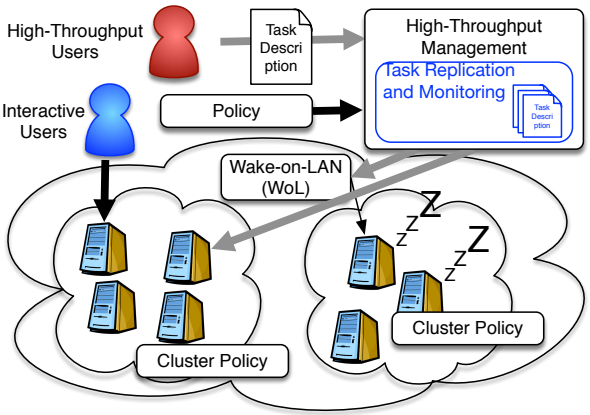

Figure 1: Overview of the HTC-Sim model

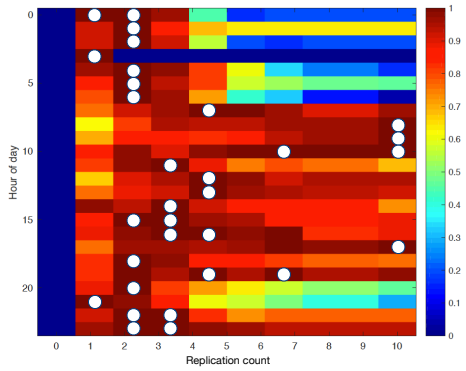

Figure 2: Heatmap: replication count per hour

Resources are organised into a number of 'clusters' (defined as a set of resources which share similar attributes such as hardware type or management policy). Thus we can handle both homogeneity within a cluster and heterogeneity between clusters. Clusters are available to interactive users, or not, depending on the cluster policy with policy changes allowed at any time. Resources, which may be put to sleep when idle, may be woken up by the HTC system, based on policy, using Wake On Lan [11]. The HTC management is under the control of a time-varying policy, controlling such things as how long after a primary user has logged out can the resource be used.

Our extension is built into the HTC management system and is highlighted in blue (Figure 1). Each task, for replication, is first replicated N times and tagged with the id of the original task before being submitted as a 'regular' task. These replica tasks can cease execution due to three cases:

Successful task completion: On first replica completion all other replicas are no longer needed and are wasting resources (energy). They can be terminated. The task which completed will assume the identity of the original task – allowing the user to access output.

Task eviction due to primary use case: Providing it is allowed by policy the task will be re-allocated to a new resource (if available). As the overall task has not completed no actions against other replicas is required.

Task termination by the task submitter: This will cause a cascade of terminations across all replicas. This can also be triggered by the HTC administrator.

We do not consider here such cases as task suspension (execution starvation) or task checkpointing and migration [13] as these do not affect the execution of the other replicas.

2.2 HTC Condor At Newcastle

The HTCondor [15] setup at Newcastle University in 2010 comprised of ~1500 desktop computers distributed over 37 clusters spread around the university. Clusters had widely varying policies – from clusters only available during teaching hours to clusters which were open 24/7 for any use.

Figure 2 illustrates the probability of a task completing given that it was submitted during a given hour of the day and that the system replicated each task a given number of times. For ease the highest probabilities for each hour are highlighted with a white dot – note that for some hours there are multiple hours with the highest probability. The modal 'best' replica count is two – one

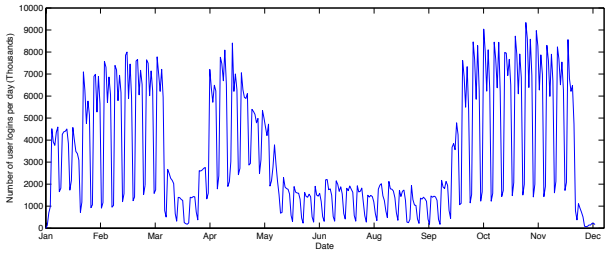

Figure 3: Number of interactive logins per day

could therefore set this as the replica count and proceed no further. However, between 8am and 10am the best option is ten replicas, and there is no account of energy usage. The special case is 3am where all computers are regularly rebooted.

Figure 2 is a static snapshot of Newcastle University, and almost certainly incorrect for any other point in time or other site. The process could be repeated at regular intervals (on other sites). However, this would not dynamically update as tasks are run. Nor take into account seasonal effects such as weekly / termly interactive user patterns – Figure 3.

3 RELATED WORK

Task replication for volunteer computing has been studied extensively for BOINC [1] style systems in which there is effectively only one HTC user – e.g. SETI@Home [2]. However, the primary focus for using replication has been task output validation as opposed to QoS or energy consumption.

Heien et al. [5] proposed the use of replica batches to handle unreliable workers (resources) in volunteer computing systems. They simulate their approach for a BOINC-style volunteer system, and don't take energy into consideration.

Kondo et al. [7] produce a simulation of a HTC volunteer computing system in order to compare it to running workload in the cloud. However, they only allow a fixed replica count of three and don't consider QoS or energy.In other work Kondo et al. [6] simulate the running of task on a desktop grid. However, their model has just one single HTC user and they only consider makespan.

BOINC itself uses replication, though only for validating results from new resources – only trusting new resources once they have provided a number of 'valid' results [8].

Litke et al. [9] use task replication in mobile grid environments to overcome the problems of fault tolerance. Though they take neither energy nor QoS into account.

4 TASK REPLICATION

In this section we initially present the metrics which will be used to evaluate our approach before presenting details of the Reinforcement Learning approach used approach.

4.1 Metrics

Before we define metrics to evaluate our approach we must first define the criteria for a task completion within Quality of Service (QoS) bounds. As the intention here is to increase the chances of tasks finishing quickly we define bounds on the turnaround time of a task, where turnaround is defined as the time between task submission and results being staged back to the submitter. We define two QoS bounds:

Within Hour: A task has achieved its QoS bound if $t \leq \lfloor e \rfloor + 1$, where t is the turnaround time (in hours) and e is the execution on a dedicated resource (in hours). As many tasks have an execution time significantly less than an hour this can be a good QoS bound.

Percentage overhead: A task is considered to be successful if its turnaround time is no more than $p\%$ greater than its execution time on a dedicated resource.

In order to evaluate the effectiveness of our approach we define the following metrics:

- **Number of tasks which fail to meet the QoS bound**.
- **Energy consumption from replicas**. As the sum of good and bad energy:

$$bad = \sum_{r \in R} \sum_{i \in I_r} \begin{cases} t_i E_i & \text{if } G_{r,i} \neq 1 \\ 0 & \text{otherwise} \end{cases}, \quad (1)$$

where R is the set of all replicas, I_r is the set of all invocations of replica r, t_i is the execution time for invocation i, E_i is the energy consumption rate of the resource used for invocation i and $G_{r,i}$ evaluates to one for the good replica, else zero. Likewise: $good = t_g E_g$, where, t_g is the execution time of the successful replica (i.e. $G_{g,r} = 1$), and E_g is the energy consumption rate for the good replica.

4.2 Reinforcement Learning

We wish to determine for a given task t, submitted at time s_t, the number of replicas which should be submitted into the HTC system. We include here one as a valid number of replicas (indicating no replication). In order to tailor our approach to an individual HTC system we use Reinforcement Learning [14] (RL), to train an agent to estimate the replica count which is expected to give the greatest reward (chance that the task will complete within the QoS bounds). RL has the advantage that it is an unsupervised machine learning approach which can learn the 'best' action to perform given a particular state of the system. This can be achieved without the need for training data – with the training coming from rewards (positive feedback) given for choosing the right action and punishments (negative feedback) for choosing the wrong action. Thus RL can, not only, adapt itself to any given environment but also, as it continually trains, adapting to environment changes. RL has been previously used to solve control problems (elevator scheduling), resource allocation within a data centre [3], and reduction of energy consumption in volunteer computer systems [12].

In order to use RL to optimise the number of replicas to run we use the approach of an n-armed bandit [14]. Under this assumption each action – the number of replicas to run – is independent of all other actions performed.

Each task $t \in \{1, 2, \ldots\}$ which is to be replicated will observe the system in a given state $s \in S$. Our state space here represents those characteristics of the system over which decisions should be made – e.g. the time of day at which the task is submitted. As these need to be discrete values we round the time of submission to the nearest hour, thus giving 24 states for time. In order to maintain our n-armed bandit model we assume here that a task which has to relinquish a resource becomes a new task within the system when it is re-allocated. The set of actions ($a \in A$) is the number of replicas

which should be submitted to the system. We can then determine the action a to perform as:

$$a = f(Q(s, A)), \tag{2}$$

where $Q(s, A)$ is the set of all reward values for the actions A available when the system is in state s and $f()$ is a selection policy. The true reward values $Q(s, A)$ are unknown, however, we can estimate $Q'(s, A)$ from the prior decisions which have been made and the associated rewards. Thus this becomes an estimator for $Q(s, A)$:

$$Q'_t(s, A) = \{q'_t(s, a)\} \quad \forall a \in A,$$

and:

$$q'_t(s, a) = \overline{R_i(s, a')} \quad \forall i \leq t, a' = a,$$

where $R_t \in [-k, k]$ is the reward function for task t. A value, for $R_t(s, a)$, of $-k$ indicating that this was the worst possible choice of action whilst $+k$ indicates the best. The value of k can be chosen arbitrarily, however, it is normally chosen to be a small number to prevent buffer overflows.

We can define the reward function as follows for task t:

$$R_t(s, a) = \begin{cases} +k - \sigma_t & t \text{ completed within QoS bound} \\ -k & t \text{ failed QoS bound} \end{cases}$$

where the first term in the reward function is used to indicate that the chosen action was good or not and the second term (if present) helps to steer the replication task towards the minimum value. The QoS bound is as defined above. We define two approaches for determining σ_t:

Wasted Energy: We set the value of $\sigma_t \in [0, k]$ to be proportional to the wasted work performed by replicas. We compute σ_t using the energy consumption of the 'bad' replicas as energy is an easy to compute proxy for the wasted work:

$$\sigma_t = \delta \min(1, \frac{W_t}{a d_t \Xi}),$$

where a is the number of replicas, d_t is the execution time of task t, W_t is the energy wasted running task t (Equation 1), $\delta \in [0, k]$ is the impact we want wasted energy to have on σ_t and Ξ is the average energy consumption rate for the selected resource when performing computational work – for simplicity we assume that this is the average energy consumption rate when the resource is running at 100% utilisation.

Contender: We evaluate here the number, c of other replicas which could have completed within the QoS bound and penalise for each: $\sigma_t = \delta c / a$, where a is the number of replicas.

We can now define the selection policy $f()$ which is used to evaluate the action to perform given the prior history reward set $Q'(s, A)$. We define two approaches here, those of a greedy selection (often referred to as exploitative) and an explorative selection policy:

$$f(Q'(s, A)) = \begin{cases} max_a(Q'(s, A)) & \text{with probability } 1 - \epsilon \text{ (exploitative)} \\ random(A) & \text{with probability } \epsilon \text{ (explorative)} \end{cases},$$

$max_a()$ selects action a with the greatest expected reward, whilst $random(A)$ selects an action uniformly from A.

By selecting the greedy policy we are exploiting prior knowledge to use the action with the greatest expected reward, whilst an exploitive policy allows us to search for potentially better actions. Both exploitative and explorative policies are required. This is particularly important due to the dynamic and changing nature of our system. Being too greedy can lead to poor choice of replica counts as the agent will keep using sub-optimal actions, whilst being too explorative can lead to the use of sub-optimal actions which are known to be bad. Careful selection of ϵ is therefore required.

4.3 Comparison Cases

In this work we will compare three different mechanisms for task enactment:

Single task execution Here each task is submitted only once to the HTC system – the default case. This provides a baseline for how long the task will take to execute within the HTC system. Effectively a replication count of 1.

Fixed replica execution: In this case the number of replicas which are submitted to the HTC system is fixed. Although this can lead to reduced execution time it has two main disadvantages. Firstly, tasks will be needlessly replicated at times of the day when this is not required. Secondly, the extra replicas running within the system can end up overloading the system. This will lead to tasks which would have completed no longer being able to due to contention for the limited resources.

Reinforcement Learning for replica selection: Here we dynamically select at submission time the number of replicas to run. This increases the chance of the task finishing within the defined contingency, whilst minimising the effort to achieve this and reducing the chance of overloading the HTC system in the process.

5 EXPERIMENTAL SETUP

In order to evaluate the benefits of task replication for a multi-HTC user based system we have extended the HTC-Sim simulation system [4]. We use here our trace logs for the use of the HTCondor [10] system and interactive users at Newcastle University during 2010. These trace logs represent some 1,229,820 interactive user logins – these were the primary users of the resources which made up the cluster of ~1,500 computers. During the year a total of 561,851 HTC tasks were submitted by 19 different HTC users. Full analysis of these trace logs can be found in [4].

As well as having the fixed replica counts (1, 2, 3, 4) we also evaluate six different RL state spaces: DAY – one state for each hour of the day, DAY_LENGTH – a 2D state space of hour of the day and job length (in hours), DAY_LOAD – a 2D state space of hour of the day and HTC system load (in 10 buckets each of 10%), LOAD - the load of the HTC system (in 10 buckets each of 10%), SINGLE – just one state, WEEK – 168 states with one for each hour of the week.

As our intention here is to identify the impact of task replication, which did not exist in the original trace log, we randomly select a proportion of the original tasks to be replicated tasks. This value has been varied in the results. Likewise we choose the value of k (reward / punishment value) to be one. The value of δ is varied in order to determine its impact on learning.

We have limited the maximum number of replicas to ten for Reinforcement Learning as experiments have shown that values

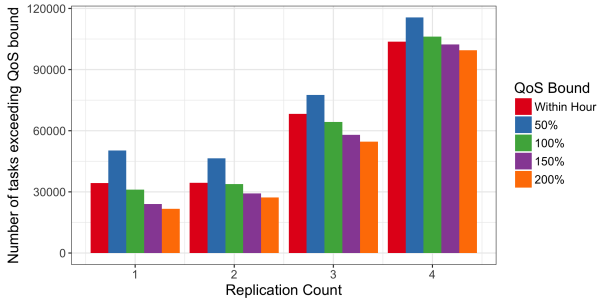

Figure 4: Impact of replica count and QoS bound on failed QoS tasks

greater than this give no advantage and often cause the cluster to become overloaded.

All simulations were performed using parameter sweeps over the parameter space and were run across the HTCondor system at Newcastle University. Ten repetitions were used to minimise the effect of random variations.

6 RESULTS

We present here the results of our simulations of the HTC-Sim system using replicated tasks. In Figure 4 we show the impact on the number of tasks which fail the QoS bound for different fixed replication counts. The lowest failed QoS value is observed for a replication of one (in this case only the original task is run with no replicas). This would suggest that for this system the 'best' approach is not to perform replications. Likewise, for the least QoS bound failures happens for the highest percentage value – a consequence of the fact that any task which satisfies a QoS bound of $n\%$ will also satisfy a QoS bound of $m\%$ where $n \leq m$. We do not reproduce the corresponding Energy graph here as no interesting results can be derived over i) QoS bounds have no impact on energy consumption (as the energy consumed is not affected by QoS bounding), and, b) The energy consumption increases linearly with the increase in replicas.

Figures 5 and 6 illustrate the impact of σ policy and RL state space on failed QoS tasks and energy respectively. The first thing worth noting here is that the Contender σ function outperforms the Wasted function. This seems to be worst for energy consumption - which might be considered to be where this function would work best. Out of the state spaces for the Contender cases the SINGLE state space (only one state) gives, on average, the 'best' performance for both failed QoS tasks and energy. However, it does have the greatest variability. Looking at the RL action space $max_a(Q'(s, A)) = 1$ for these cases. This concurs with the result of Figure 4 where not performing replication is the best choice.

In Figures 7 and 8 we investigate the impact on failed QoS tasks and energy consumption of varying the proportion of tasks which are considered for replication. In all cases using no replication (replication count of 1) gives the 'best' performance. For RL the LOAD state space (bucketing by percentage of the cluster which is in use) gives the 'best' performance – though this is still inferior to fixed replication cases. One may assume that the SINGLE state space would give better performance here – especially considering that it recommends not performing replicas. However, this benefit is lost due to the explorative nature of RL where it randomly chooses

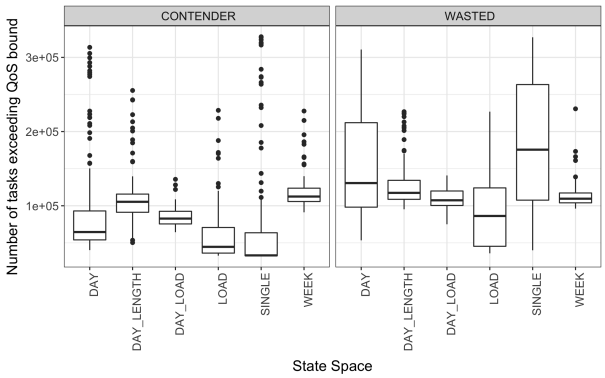

Figure 5: Impact of state space and σ policy on failed QoS tasks

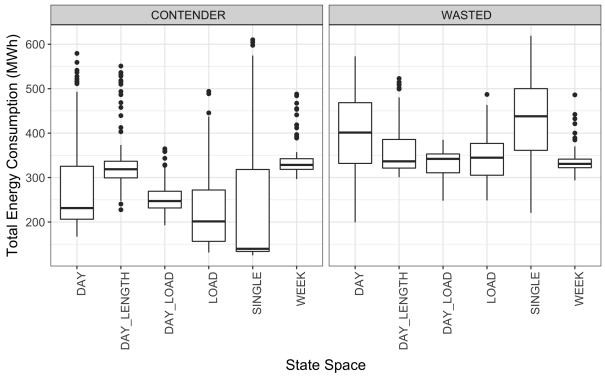

Figure 6: Impact of state space and σ policy on Energy consumption

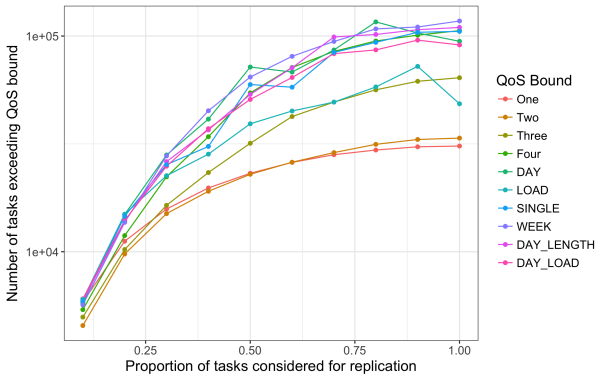

Figure 7: Impact of replica task load on failed QoS tasks

to use a sub-optimal choice in the hope of finding a better scenario.

In Figure 9 we illustrate $Q'(s, A)$ for one run of the simulation – to aid reading the 'best' replica count has been marked with a white circle. It can be seen that a replica count of 0 (no replication) is by far the most preferred choice for most hours of the day. The interesting exceptions to this are 4am, 7am 11am, 8pm and 9pm. The 4am slot is just after the nightly reboot of the system in which case lots of task will be re-started and be contending for resources. 7am is just before many people come to the university, thus running many replicas helps with reducing failed QoS. 11am, although having a 'best' choice of 0 doesn't have any real strong winner – a consequence of

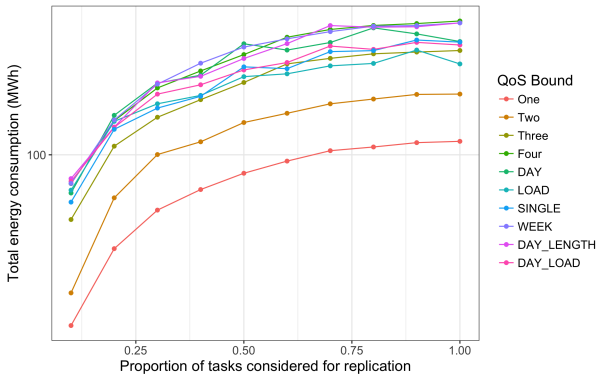

Figure 8: Impact of replica task load on Energy

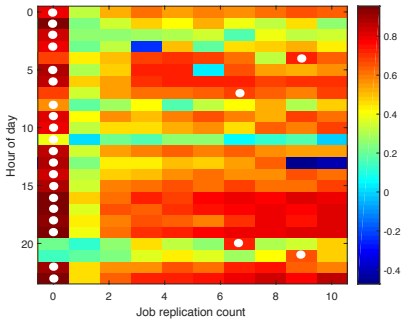

Figure 9: RL rewards for one day

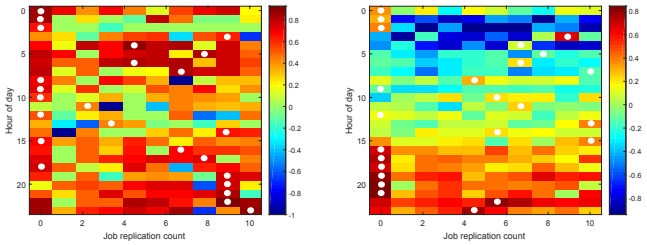

Figure 10: RL rewards split on job size

lots of users coming in at lunchtime. For 8pm and 9pm, this could be a consequence of students coming in to do work in the evening after eating.

We break Figure 9 up into short jobs (less than one hour) and long jobs (more than one hour) in Figure 10 – left and right side respectively. Longer jobs have a greater tendency to go for no replication while shorter jobs seem to favour more replication. This could be a consequence of the greater potential energy wastage when a longer job fails. Between 4am and 3pm there is no clear winner for replication count of longer jobs – with all actions having poor potential rewards. This would suggest that not even trying to run the task would be the best option. You would automatically fail the QoS bound, though this is likely to happen anyway due to the large number of interactive users.

7 CONCLUSIONS

Motivated by the success of replicated tasks in single HTC applications the purpose of the paper here was to evaluate if replicating tasks in a university HTC cluster could save energy whilst at the same time improve the number of tasks which completed within a QoS bound. Unfortunately, due to the nature of the workload here this did not turn out to be the case. Not performing replications was observed to be the preferred option in both fixed replications and when using Reinforcement Learning (RL).

In prior work the focus of replication has not been to increase the number of tasks achieving a QoS bound for multiple users, but rather to tolerate faults or dishonest users, or reduce the turnaround time for individual tasks. Although turnaround could be reduced here for individual tasks this is often at the cost of other tasks in the system. If we limited the number of tasks in the system such that it would not overload the cluster with replications it would be assumed we could achieve this goal.

The use of RL has provided insights into how we could better handle tasks within the cluster – such as not submitting long tasks during the core working day. Further analysis of this could help in maximising throughput – though at the expense of some HTC users. Likewise, analysis of different values of ϵ and when to change between them could help improve the performance of the RL approach.

REFERENCES

[1] David P Anderson. 2004. Boinc: A system for public-resource computing and storage. In *Grid Computing, 2004.* IEEE, 4–10.

[2] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. 2002. : An Experiment in Public-resource Computing. *Commun. ACM* 45, 11 (Nov. 2002), 56–61. https://doi.org/10.1145/581571.581573

[3] Peter Bodík, Rean Griffith, Charles Sutton, Armando Fox, Michael Jordan, and David Patterson. 2009. Statistical Machine Learning Makes Automatic Control Practical for Internet Datacenters. In *USENIX HotCloud*. Article 12. http://dl.acm.org/citation.cfm?id=1855533.1855545

[4] M. Forshaw, A.S. McGough, and N. Thomas. 2016. HTC-Sim: a trace-driven simulation framework for energy consumption in high-throughput computing systems. *Concurrency and Computation: Practice and Experience* 28, 12 (2016), 3260–3290. https://doi.org/10.1002/cpe.3804 cpe.3804.

[5] E. M. Heien, N. Fujimoto, and K. Hagihara. 2008. Computing low latency batches with unreliable workers in volunteer computing environments. In *2008 IEEE International Symposium on Parallel and Distributed Processing*. 1–8. https://doi.org/10.1109/IPDPS.2008.4536442

[6] Derrick Kondo, Andrew A Chien, and Henri Casanova. 2004. Resource management for rapid application turnaround on enterprise desktop grids. In *ACM/IEEE Supercomputing*.

[7] D. Kondo, B. Javadi, P. Malecot, F. Cappello, and D. P. Anderson. 2009. Cost-benefit analysis of Cloud Computing versus desktop grids. In *2009 IEEE International Symposium on Parallel Distributed Processing*. 1–12. https://doi.org/10.1109/IPDPS.2009.5160911

[8] Eric J. Korpela. 2012. SETI@home, BOINC, and Volunteer Distributed Computing. *Annual Review of Earth and Planetary Sciences* 40, 1 (2012), 69–87. https://doi.org/10.1146/annurev-earth-040809-152348 arXiv:https://doi.org/10.1146/annurev-earth-040809-152348

[9] Antonios Litke, Dimitrios Skoutas, Konstantinos Tserpes, and Theodora Varvarigou. 2007. Efficient task replication and management for adaptive fault tolerance in Mobile Grid environments. *Future Generation Computer Systems* 23, 2 (2007), 163 – 178. https://doi.org/10.1016/j.future.2006.04.014

[10] M. Litzkow, M. Livney, and M. W. Mutka. 1988. Condor-a hunter of idle workstations. In *ICDCS*.

[11] A.S. McGough, C. Gerrard, J. Noble, P. Robinson, and S. Wheater. 2011. Analysis of Power-Saving Techniques over a Large Multi-use Cluster. In *IEEE DASC*. https://doi.org/10.1109/DASC.2011.78

[12] A. Stephen McGough and Matthew Forshaw. 2014. Reduction of wasted energy in a volunteer computing system through Reinforcement Learning. *Sustainable Computing: Informatics and Systems* 4, 4 (2014), 262 – 275. https://doi.org/10.1016/j.suscom.2014.08.014

[13] Shuangcheng Niu, Jidong Zhai, Xiaosong Ma, Mingliang Liu, Yan Zhai, Wenguang Chen, and Weimin Zheng. 2013. Employing checkpoint to improve job scheduling in large-scale systems. In *Job Scheduling Strategies for Parallel Processing*. Springer, 36–55.

[14] R.S. Sutton and A.G. Barto. 1998. *Reinforcement Learning: An Introduction*. Bradford Book.

[15] The Condor Team. 2010. Condor Manual. http://www.cs.wisc.edu/condor/manual/. (Oct 2010). University of Wisconsin.