Reducing the number of miscreant tasks executions in a multi-use cluster

A. Stephen McGough, Matthew Forshaw
Computing Science
Newcastle University
Newcastle upon Tyne, UK
{*stephen.mcgough, m.j.forshaw*}*@newcastle.ac.uk*

Clive Gerrard[1]
Information Systems and Services
Newcastle University
Newcastle upon Tyne, UK

Stuart Wheater
Arjuna Technologies Ltimited
Newcastle upon Tyne
stuart.wheater@arjuna.com

***Abstract*—Exploiting computational resources within an organisation for more than their primary task offers great benefits – making better use of capital expenditure and provides a pool of computational power. This can be achieved through the deployment of a cycle stealing distributed system, where tasks execute during the idle time on computers. However, if a task has not completed when a computer returns to its primary function the task will be preempted, wasting time (and energy), and is often reallocated to a new resource in an attempt to complete. This becomes exacerbated when tasks are incapable of completing due to excessive execution time or faulty hardware / software, leading to a situation where tasks are perpetually reallocated between computers – wasting time and energy. In this work we investigate techniques to increase the chance of 'good' tasks completing whilst curtailing the execution of 'bad' tasks. We demonstrate, through simulation, that we could have reduce the energy consumption of our cycle stealing system by approximately 50%.**

***Keywords*-Energy; cluster; task completion; cycle stealing;**

I. Introduction

Many organisations exploit the computational power of their existing computing resources in order to perform large amounts of computational work through the use of cycle stealing technologies such as Condor [1] or BOINC [2]. Making the best use of existing capital investment for the minimal additional cost and energy incurred by using these computers for a secondary purpose when otherwise unused. Companies often use workers' desktops whilst universities can also exploit computers made available for students. Figure 1 illustrates an architecture for an organisation in which interactive users have direct access to computers whilst cycle stealing users interact through a pool management system adhering to policy. Both groups are able to wake up sleeping computers which enter this state after a period of inactivity.

One of the major problems of such an approach is ensuring that all 'good' tasks complete. Where a 'good' task is defined as one which given enough time on a dedicated resource would run to a natural completion. Computers within the cluster can appear and disappear arbitrarily, the computers may be heterogeneous (or broken) making it difficult for tasks to execute correctly, or computers may have to preempt tasks in order to return to their primary

[1]Work carried out whilst working for ISS.

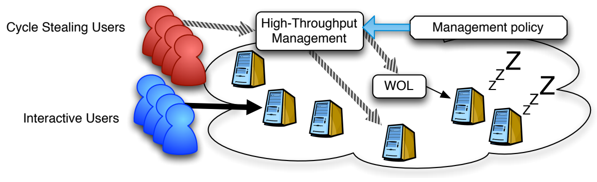


Figure 1. Multi-use cluster architecture

role. Thus if a task fails to complete on a resource we cannot assume that it is a 'bad' task.

To alleviate the effects of the system on task execution an approach is adopted in which tasks that do not reach a natural completion are reallocated to a new resource. This leads to potential wasted energy from tasks repeatedly allocated to resources either because the task will never complete or the resource is incapable of satisfying task requirements (e.g. appropriate environment or long enough period for execution). This can be alleviated by limiting the number of resubmissions, though if the value is too low 'good' tasks, unfortunate in their allocation, will fail to complete whilst if the value is too high 'bad' tasks will waste time and energy. We define a *miscreant* task as one which exhibits multiple reallocations attempts and seek to minimise energy consumption by reducing the number of reallocations of 'bad' tasks whilst increasing the chance that 'good' tasks are reallocated to resources capable of servicing their needs. It should be noted that miscreant does not imply 'good' or 'bad' just that a task has required multiple reallocations.

Traditionally this has led to a two-way compromise between the number of failed 'good' tasks and the overhead (defined as the time in excess of the tasks run-time incurred by the system), with each organisation selecting a local optimal – an open question which received significant discussion at Condor Week 2012 [3]. However, due to energy conservation – now a more important criteria – this has become a three-way problem. Reducing reallocation attempts reduces energy consumption through removal of 'bad' executions, though increases 'good' tasks failures.

Tasks can be allocated to resources whilst they are idle or sleeping (through Wake on LAN) executing until the resource is required for its primary purpose (interactive user, system maintenance or reboot). This would suggest that the ‘best’ option is to have tasks shorter than the intervals between primary use and / or only run task during expected long periods of primary inactivity (overnight). However, such a policy leads to excessively short execution times and significant delays in task execution.

In this paper we investigate a number of policies for curtailing ‘bad’ executions whilst still minimising the number of ‘good’ task terminations and the average task overhead – allowing us to minimise energy consumption. The rest of the paper is set out as follows. In Section II analysis of an existing multi-use cluster is performed. Section III describes policies aimed at identifying which miscreant tasks should be re-run and which should be terminated. Related work is presented in Section IV. Section V describes our simulation model, whilst Section VI presents the simulated results for these different polices, before concluding in Section VII.

II. Analysis of the Cycle Stealing architecture

Condor [1] is a high-throughput computing system used for cycle stealing or dedicated resources. Condor attempts to successfully complete all submitted tasks irrespective of issues with the computers that it is running on. Tasks which are deallocated from a resource are reallocated to a different resource unless they have been allocated too many times.

A. *Task Deallocation*

Tasks may become deallocated from the resource they were previously allocated to for several reasons:

- **Task preemption:** Condor has decided to deallocate the task. Condor [4] identifies four preemption cases: i) Higher priority task is identified which will start once this task has been preempted; ii) Policy of the resource – this can include an interactive user logging in or a pre-defined time during when tasks can’t run; iii) Resource ranking – the resource determines a more appropriate task to execute (e.g. a maths department owned computer preempts non-maths tasks for maths tasks); iv) Condor is shutting down – during shut-down Condor will preempt running tasks. Many managed clusters have a regular shutdown policy allowing updates and resetting. These preemptions will mark a task as miscreant though none indicate the tasks is ‘bad’.
- **Hardware / Software failures:** If a resource becomes unreachable by the system for an appropriate interval it will be deemed no longer part of the pool. This can be for a myriad of reasons including hardware failure, Operating System failure, catastrophic software failure (including the running task) or network failure. Note that these may be transient in nature. Again none of these issues implies that the running task was ‘bad’.

Although the above indicate under what circumstances a task is deallocated from a resource they don’t distinguish whether the task could complete on a subsequent execution. In all cases the task is deallocated before it has reached its own natural exit point. The reason for this can be:

- **Execution time longer than time available:** The time between allocation and deallocation, t_r, is less than the task execution time. If t_r is small the task is likely to complete on a new allocation, whilst if t_r is close to the maximum time available then it is most likely to be deallocated again. Note that the task may be ‘good’ but require more time than the system can provide.
- **Code has malfunctioned:** the code crashes but does not terminate (infinite loop, awaiting user interaction) remaining active until deallocated. Re-running the task is unlikely to change this scenario. Reducing the chance of a re-run here is highly desirable.
- **Hardware / Software malfunction:** A fault in the environment causes the task to fail to terminate (broken library, CPU failure). Reallocating to a different resource is likely to allow the task to complete.
- **Task requirements not satisfied:** Although many failures in task requirements would prevent the task from starting or fail upon starting, there are circumstances where an apparent code malfunction would occur. However, in this case allocation to a new computational resource could resolve these requirements.

This problem becomes exacerbated by the fact that it is not possible to distinguish easily these cases from each other. A piece of code which malfunctions and is deallocated after only a few minutes exhibits the same properties as a ‘good’ task which is also evicted after only a few minutes. Hence the use of the term ‘miscreant’ indicating that, although not definitively ‘bad’ tasks, the task is behaving in a manner which is not desirable. An assumption could be taken in which any task failing to complete on the first attempt is abandoned by the system, however, this will lead to a significant number of ‘good’ tasks being terminated, though this will reduce energy consumption and the overheads on those tasks which complete – as ‘bad’ tasks will not be consuming resources. Alternatively allowing miscreant tasks to be re-run an arbitrary number of times allows ‘bad’ tasks to consume significant amounts of energy and increase the overheads for all tasks due to bad tasks consuming resources. Historically, this problem has only been considered in terms of the metrics of overhead and number of ‘good’ tasks being terminated with sites selecting a value for the number of retries which keeps the number of ‘good’ tasks terminated to an acceptable level and keeps the overheads to a reasonable level. Although desirable to obtain the right balance for these metrics there is little penalty for not getting the balance right. Including energy as the third metric thus imposes a significant penalty for wasting computational resources.

B. *Analysis of the Newcastle Condor System*

Here we investigate the two implicit policy assumptions made by many high-throughput cluster managers. In general

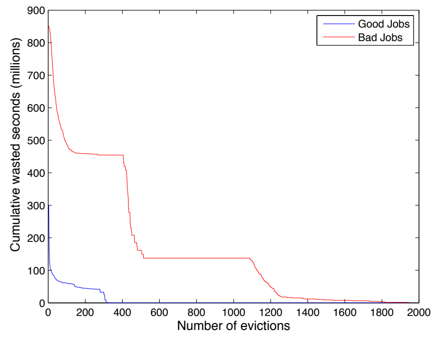


Figure 2. Decreasing graph of total wasted time vs evictions

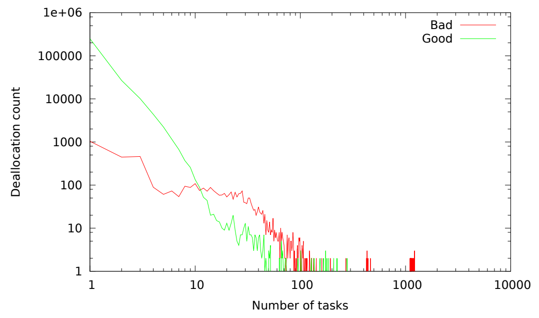


Figure 3. Histogram of good task evictions

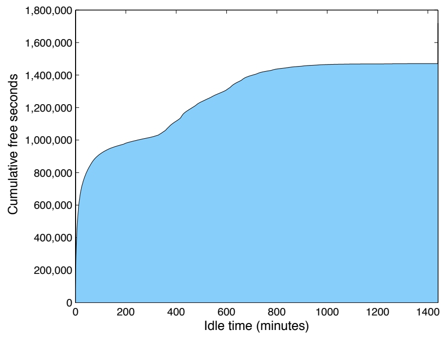


Figure 4. Cumulative idle time

it is assumed that a (fairly low) value for reallocations will allow the vast majority of 'good' tasks to be completed and that choosing a small enough task duration will allow the majority of 'good' tasks to complete without reallocation.

Newcastle University has been running a largely unmanaged Condor pool since October 2005 [5]. We are fortunate to possess the Condor history files for this period. We analyse the tasks from 2010 in order to exemplify the effects of miscreant tasks on the cluster and to address the two assumptions. In total 561,851 tasks were submitted through Condor consuming 1,684,940,087 seconds ($\sim$53 years), of which 1,218,729,685 ($\sim$39 years) was wasted. This wasted time comprised 851,989,414 seconds ($\sim$27 years) for tasks which were subsequently killed by the user – 'bad' tasks – and 366,740,271 seconds ($\sim$12 years) wasted on tasks which did complete – 'good' tasks. Although it is not possible to determine, from the history, the time consumed by each unsuccessful allocation of a task terminated by the user the total time for tasks with at least one deallocation is relatively close to the total wasted time for killed tasks (849,725,325 seconds $\sim$27 years). Thus indicating most 'bad' tasks accrued at least one reallocation. For 'good' tasks this is only the wasted time, thus all of these tasks have accrued at least one reallocation.

Although a maximum number of reallocations can be specified in Condor this property was not activated. Figure 2 shows the maximum number of retries for 'bad' tasks was 1946, whilst the maximum for 'good' tasks was 360. Figure 2 also illustrates that the majority of wasted time is associated with low eviction counts. It should be noted that Condor history does not explicitly record the number of times a task ran on a resource but the number of times that the task was allocated, thus resource state changes could cause a task to be deallocated before execution starts. However, as we are interested here in the number of times a task is allocated these rapid deallocations can simply be ignored as fortuitous in terms of energy consumption. Thus to ensure all 'good' tasks are successful we need a maximum reallocation count of 360. Reducing wasted 'bad' task time to 395,373,483 seconds ($\sim$13 years). It should be noted that this does not take into account the effect that changing the policy would have on the operation of the cluster or the way users would interact with the cluster.

Figure 3 illustrates the number of 'good' and 'bad' deallocations. In both cases the average number of deallocations is relatively low (1.38 and 44.89 respectively). In order to ensure 95% 'good' task completion we need a reallocation maximum of 3, whist for 99% we need a threshold of 6 – this matches nicely with the intuitive value quoted by many cluster managers. However, a maximum of 6 reallocations would mean 2,022 'good' tasks failures, though reducing wasted time on 'bad' tasks to 7,534,050 seconds ($\sim$87 days).

Figure 4 illustrates the idle intervals for computers, defined as the time between a user logging off and the next user logging in. The average idle length is 371 minutes. A task would need to be no longer than 2 minutes to ensure that 95% of idle intervals are long enough, whilst it would have to be no longer than 1 minute to ensure that 99% of intervals are long enough. This is clearly unobtainable.

We can clearly reduce the wasted time in a cluster by reducing the number of reallocations at the expense of failing to complete 'good' tasks. Hence we need a better approach in which good tasks which have been unfortunate in their allocations are reallocated whilst 'bad' tasks are curtailed.

III. Policy for handling miscreant tasks

We first outline policies which have previously been identified for detecting miscreant tasks and for the selection of resources before introducing newly identified policies.

A. Existing Policies

X0: There is no limit on the number of times a task can be reallocated with termination only occurring if the task is removed by the submitter or an administrator.

N1(*n*): Termination after *n* reallocations. If a user still believes that the task is good they can resubmit it. This represents the Condor default policy for reallocation.

C1: Tasks are allocated to resources at random favouring awake resources. This represents the Condor default policy.

C2: Targeting less used computers [6]. By selecting resources with longer idle times between users reduces the chance that a task will be deallocated due to preemption.

The remaining policies are proposed as part of this work:

B. Computer selection policy

C3: tasks are allocated to computers in clusters with least amount of time used by interactive users. This reduces the chances of task preemption and exploits the less popular clusters around campus. Computers can be ranked using:

$$Rank(c) = \frac{\sum_{s}^{s \in c} s_{idle}/s_{total}}{|c|}$$

where c is the set of computers in a cluster, s is a computer in c, s_{idle} is the total idle time on computer s, and s_{total} is the total time for computer s.

C. Dedicated resources

D1(*m,d*): Tasks identified as miscreant are permitted to continue executing on a dedicated set of m computers (without interactive users or reboots), A maximum duration d prevents the task from running indefinitely.

D. Miscreant task identification

We evaluate a number of policies to identify and handle miscreant tasks. These policies govern the circumstances under which deallocated tasks are abandoned or reallocated.

Conventional *n* reallocation policies do not distinguish the causes of deallocation, thus are poorly suited to the multi-use cluster context. Evictions due to the arrival of interact users and planned machine reboots do not in any way imply a task to be miscreant. We propose two variations on **N1** which discount these evictions from a task's reallocation count:

N2(*n*): A task will be abandoned if it deallocated *n* times ignoring deallocations due to interactive users.

N3(*n*): A task will be abandoned if it is deallocated *n* times ignoring deallocation due to computer reboots.

Random policies are presented to allow comparison:

R1(*p*): a task is abandoned with probability *p* ($0 \leq p \leq 1$).

A deallocated task *j* is retried according to exponential function $P(f) = (1 - e^{-kf}), 0 \leq k \leq 1$, where k is a scaling factor:

E1(*f*=*n*): Exponential decay on deallocation count *n*.

E2(*f*=*t*): Exponential decay on the total *accrued time* from all executions.

Tasks are subject to an upper bound *t* on their cumulative execution time, and are abandoned if deallocated and over this bound. Furthermore, we investigate the impact of discounting deallocations due to interactive users and reboots from a task's accrued execution time:

A1(*t*): Abandon if accrued time $> t$ and task deallocated.

A2(*t*): Abandon if accrued time $> t$ and task is deallocated discounting deallocations due to interactive users.

A3(*t*): Abandon if accrued time $> t$ and tasks is deallocated discounting deallocations due to reboots.

I1(*t*): Abandon if individual time $> t$. Nightly reboots bound this to 24 hours. We investigate the impact of lowering this threshold.

By leveraging historical information it is possible to more closely identify 'bad' tasks by looking at the percentile values for different properties:

P1(*n,p*): Abandon if $percentile(n) > p$.

P2(*t,p*): Abandon if $percentile(accruedtime) > p$.

IV. Related work

The most common and default policy for handling task failures in an unreliable environment is resubmission. Berten and Jeannot [7] performed a numerical analysis of resubmissions in a fault prone Grid environment. Their approach studies the effect of bounded and unbounded reallocation polices (equivalent to X0 and N1(n)). However, energy consumption is not considered and tasks are assumed not to be faulty – something we see here as a significant factor.

Checkpointing and migration [8] does not reduce task reallocation but removes the need to re-start the task after each reallocation. However, to allow checkpoint and migration the task and the environment needs to support this process, something which is currently unavailable in the Windows implementation of Condor which makes up the majority of the Newcastle pool. Users can 'roll' their own checkpoint and migration mechanism, however this is often a non-trivial task to perform. We see that the use of checkpoint and migration is complementary to our proposed policy, though the values of the parameters for a number of our policies would need to be modified in order to accomodate.

The use of task redundancy [9] in which multiple copies of each task are deployed increasing the chance that at least one will complete in the first attempt, helps to reduce overheads for the task and ensure that 'good' tasks complete. However, these multiple runs will in general consume more energy than running resubmissions. We therefore see this as an alternative approach for scenarios in which overhead and 'good' task completion are the primary concerns.

Hwang and Keselman [9] present an architecture in which extra tasks are run alongside the main task in order to more closely identify the state of the main task. This we see as complementary to our work and could be used to

Table I
COMPUTER TYPES

Type	Cores	Speed	Power Consumption		
			Active	Idle	Sleep
Normal	2	~3Ghz	57W	40W	2W
High End	4	~3Ghz	114W	67W	3W
Legacy	2	~2Ghz	100-180W	50-80W	4W

Table II
BASELINE RESULTS

Policy	Overheads	Power	Good tasks killed
X0 C1	20.03 minutes	137.54 mWh	0
X0 C2	15.05 minutes	123.58 mWh	0
X0 C3	15.77 minutes	117.43 mWh	0

help aid 'good' and 'bad' task detection. Haider *et. al.* [10] provide a literature review for the different fault tolerance mechanisms provided by different distributed systems along with an argument for the need for such techniques.

Estimates of task execution times can be used as a criteria for selecting when to abandon a task. However, the use of estimates, provided by users at submission, have been widely criticised by the scheduling community for their inaccuracy [11]. With many papers reporting the majority of task taking less than 30% of their requested allocation [12], [13], [14]. This may be due to tasks misconfiguration causing immediate termination [15] but is often due to wide variation in execution times [16] – especially if the cluster is heterogeneous – or since tasks are often terminated at the end of their estimated time interval users 'pad' their estimate to increase the chance of completing. If estimates are provided by the system (equivalent to I1(t)) then these need to be generous – often the maximum available. This in general does not help here as reallocations will happen before this time limit is reached.

V. SIMULATION MODEL

Our simulations are based on trace logs from the Condor high-throughput cluster at Newcastle University along with user cluster access logs for 2010 [5], [6]. The 1359 student access desktops, which were running Microsoft Windows XP, were distributed around the University Campus in 35 different clusters. Several clusters may share the same room, each room having its own opening hours. These hours vary between clusters that are predominantly for teaching purposes (normally 9am till 5pm) through to 24-hour access computer clusters. Computers are rebooted nightly around 5am unless an interactive user is logged in. The location of clusters has a significant impact on throughput of interactive users. From clusters buried deep within a particular school to those within busy thoroughfares such as the University Library. Computers are replaced on a four year rolling cycle. This leads to three main computer categories as illustrated in Table I. As the main focus of our work is the comparison of different polices for reducing energy, 'good' task terminations and overheads we ignore differences between the performance of these computers and assume the execution time will match the original execution time.

We have extended our cluster based simulation for Condor [6] to take account of the data transfer times. The *iperf* bandwidth testing software [17] was used to compute the maximum bandwidths available between computers for different payload sizes. Although bandwidth for small (less than 1Kb) of data exceeded 100MBits/s this quickly capped out at 94.75MBits/s. It should be noted that these are maximum bandwidth potentials, real use is likely to be less. Thus these are lower estimates of transfer times.

VI. SIMULATION RESULTS

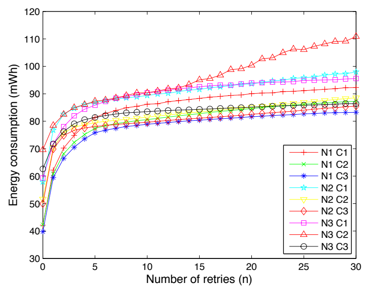
Table II depicts the results for running the base-line case of no abandonment policy (X0) against the three resource selection policies (C1, C2, C3) with the two usage-based selection policies providing a better overhead for tasks though providing alternative optimality's for energy or overhead between themselves. It should be noted that as all of these policies have no limit on the number of reallocations of tasks this leads to zero 'good' tasks being killed.

In the rest of this section we compare the energy, good tasks killed and overheads for all policies. Although not illustrated here the cost for these polices can easily be derived by multiplying energy by the cost per unit. In some cases the key has been omitted from a graph for clarity, for these the key on the other graphs in the set can be used.

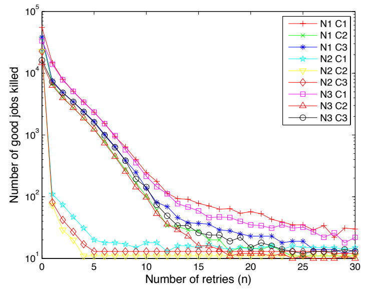
Figures 5(a), 5(b) and 5(c) illustrate the differences between abandonment polices N1, N2 and N3 along with selection polices C1, C2 and C3. Policy N2 (maximum deallocation count ignoring user preemption) gives a significant improvement for 'good' tasks terminated when the value of n is small. Selection policy C2 and C3 work well with this by keeping the energy levels and overheads low.

The individual time accrued policy I1 is explored in Figures 6(a), 6(b) and 6(c). This policy gives better energy performance and overheads in comparison with N1. However, this is at significant impact on the number of 'good' tasks which are terminated. It should be noted that the steep step in energy in Figure 6(a) corresponds with an individual execution time of 24 hours. This effectively allows the task to run indefinitely.

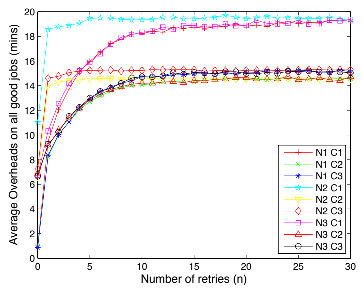
Exponential abandonment polices E1 and E2 are shown in Figures 7(a), 7(b) and 7(c). Note the scaling factors for these have been adjusted to allow both data sets to be drawn on the same graph E1 needs to be scaled by 10^{-3} and E2 by 10^{-10}. Although for this policy energy and overheads fall as the growth factor increases the number of good tasks terminated increases and from a high initial value ($\sim$4500). Likewise for the random selection policy R1 – Figures 8(a), 8(b) and 8(c) the number of 'good' tasks killed is high and increases despite offering good overheads and energy results.



(a) Energy

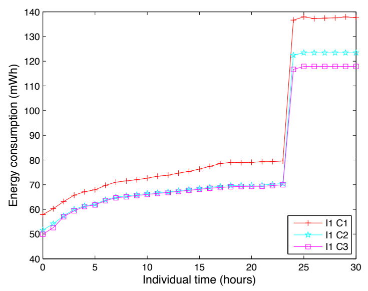


(b) Good kills

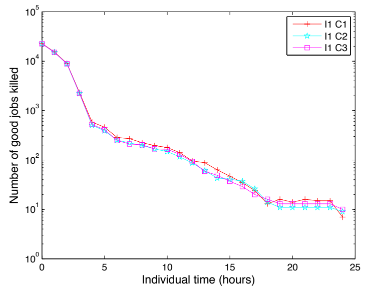


(c) Overhead

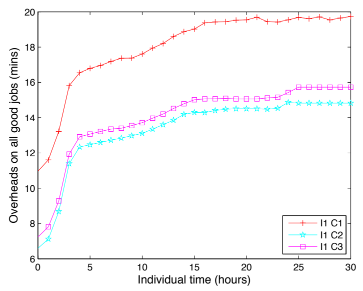
Figure 5. Policy : Terminate after N



(a) Energy

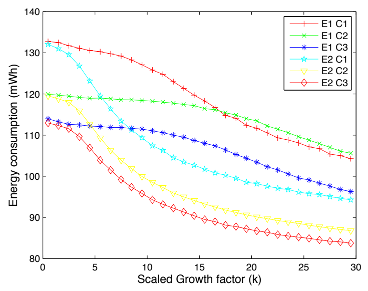


(b) Good kills

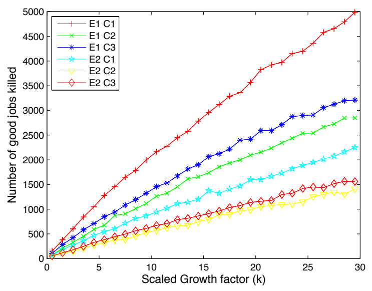


(c) Overhead

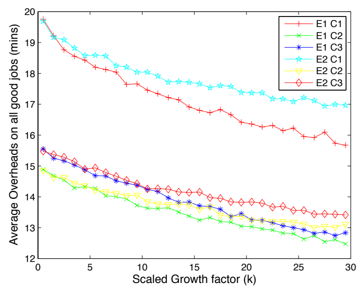
Figure 6. Policy : Individual



(a) Energy



(b) Good kills



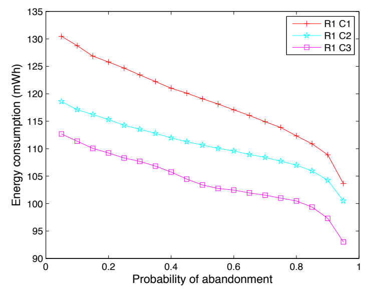
(c) Overhead

Figure 7. Policy : Exponential

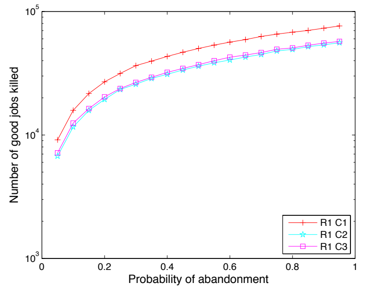
The policy of dedicated resources D(m, d) is explored in Figures 9(a), 9(b) and 9(c). There is a significant advantage here for energy in keeping the number of retries (n) low the other factors (dedicated resources and maximum dedicated time) have relatively small impact on the energy consumed increasing as the maximum run time increases on the dedicated resources. Only the maximum dedicated time has an impact on the number of good tasks killed. A dedicated maximum execution time of ∼90 hours then allows for zero 'good' task terminations with little effect on the overall overheads. Though the overheads are in general poor. Note that dedicated resources are assumed to use the same energy as our top-end computers.

Accrued policy A1, A2 and A3 are explored in Figures 10(a), 10(b) and 10(c). Low accrued times offer lower energy consumption at the expense of 'good' tasks killed. Apart from combination A3,C2 there is no significant advantage in selecting an accrued total over ∼40 hours.

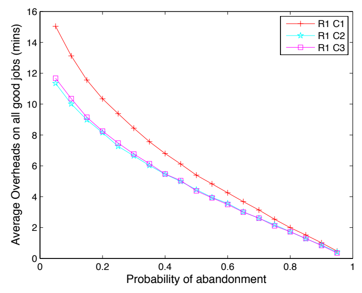
The percentile policies depicted in Figures 11(a), 11(b) and 11(c) show that the consumed energy comes down to an equivalent level as the other polices, however, only as the percentile tends to 100% do the number of 'good' tasks terminated reduce significantly. In order to get benefit from using policy P2 the percentile needs to be almost exactly



(a) Energy

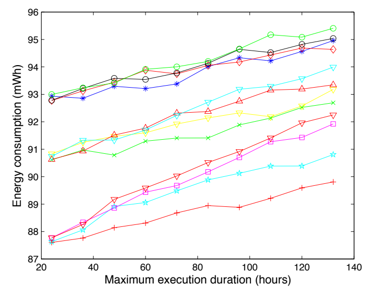


(b) Good kills

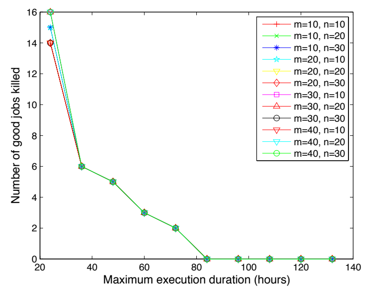


(c) Overhead

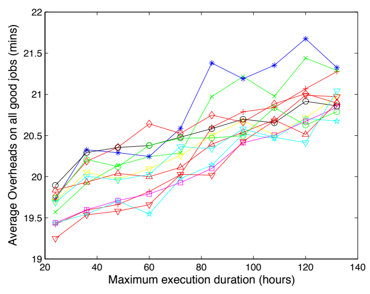
Figure 8. Policy : Random



(a) Energy

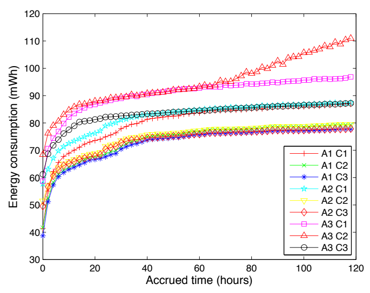


(b) Good kills

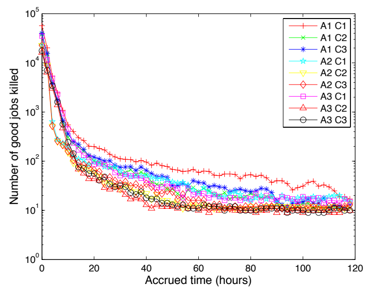


(c) Overhead

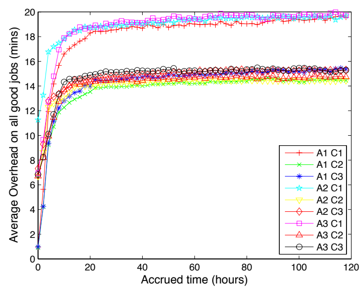
Figure 9. Policy : Dedicated



(a) Energy



(b) Good kills



(c) Overhead

Figure 10. Policy : Accrued

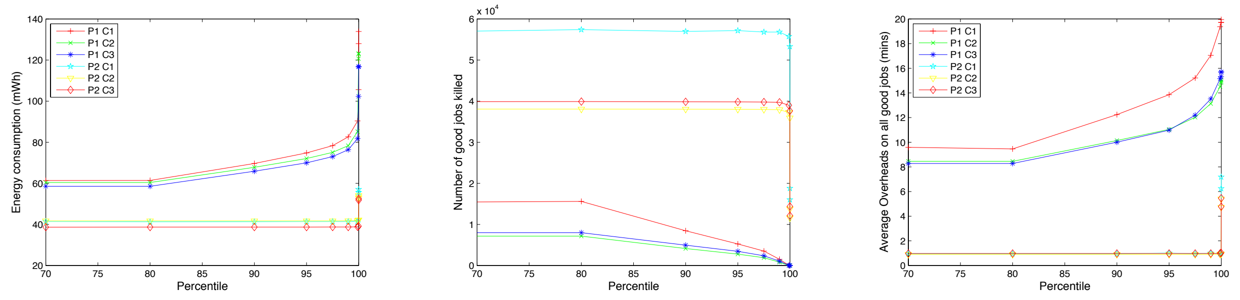
100% giving little advantage over policy N1. Whilst policy P1 benefit as low as 90%.

VII. Conclusion

In this work we demonstrate, through simulation, a number of policies which can be used to reduce the effect of miscreant tasks in a multi-use cycle stealing cluster. Each policy is capable of dramatically reducing the energy consumption for tasks in the system – to around $\frac{1}{2}$ of the original. This is largely attributed to reducing the amount of effort wasted on tasks that will never complete, but also by ensuring that tasks are placed onto computers which are less likely to be required for their primary task.

Although we are able to reduce the energy consumption significantly in all cases this can often be to the detriment of the users of the high-throughput system. Choosing a policy such as N2 (total deallocation count ignoring interactive user preemption) allows a significant decrease in energy consumption without loosing a significant number of good tasks and a minor increase in average task overhead, albeit still significantly less than the values for the baseline results. By using dedicated computers we are able to reduce the number of good tasks lost to zero for a relatively small increase in energy consumption.

The policy D(m, d) with a low value for reallocations and a value of ∼90 for maximum dedicated resource time would



(a) Energy (b) Good kills (c) Overhead

Figure 11. Policy : Percentile

appear to give a 'good' solution. However, this policy could easily be adapted to incorporate the advantages of policy N2 or even the individual or accrued polices. In fact most of the policies presented in this work could be combined with each other to maximise the potential energy savings.

Although we have developed our work around the Condor high-throughput system we see that our approach could as easily be applicable to other cycle-stealing architectures in which the tasks being executed are not necessarily guaranteed to complete. While we do not consider fully the effects of computer faults within this work we do see this as an important area for future research.

REFERENCES

[1] M. Litzkow, M. Livney, and M. W. Mutka, "Condor-a hunter of idle workstations," in *8th International Conference on Distributed Computing Systems*, 1998, pp. 104–111.

[2] D. P. Anderson, "Public Computing: Reconnecting People to Science," *Presented at the Conference on Shared Knowledge and the Web, Residencia de Estudiantes, Madrid, Spain*, Nov. 2003.

[3] The Condor Project, "Condor week 2012," http://research.cs.wisc.edu/condor/CondorWeek2012/.

[4] The Condor Project, "The condor project manual," http://research.cs.wisc.edu/condor/manual/.

[5] A. S. McGough, P. Robinson, C. Gerrard, P. Haldane, S. Hamlander, D. Sharples, D. Swan, and S. Wheater, "Intelligent power management over large clusters," in *International Conference on Green Computing and Communications (GreenCom2010)*, 2010.

[6] A. S. McGough, C. Gerrard, J. Noble, P. Robinson, and S. Wheater, "Analysis of power-saving techniques over a large multi-use cluster," in *International Conference on Cloud and Green Computing (CGC2011)*, 2011.

[7] V. Berten and E. Jeannot, "Modeling Resubmission in Unreliable Grids: the Bottom-Up Approach," in *Seventh International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks - heteroPar'09*, Delft, Netherlands, 2009.

[8] M. Litzkow, T. Tannenbaum, J. Basney, and M. Livny, *Checkpoint and migration of UNIX processes in the Condor distributed processing system.* Computer Sciences Department, University of Wisconsin, 1997.

[9] S. Hwang and C. Kesselman, "A flexible framework for fault tolerance in the grid," *Journal of Grid Computing*, vol. 1, pp. 251–272, 2003.

[10] S. Haider, N. Ansari, M. Akbar, M. Perwez, and K. Ghori, "Fault tolerance in distributed paradigms."

[11] C. Bailey Lee, Y. Schwartzman, J. Hardy, and A. Snavely, "Are user runtime estimates inherently inaccurate?" in *Job Scheduling Strategies for Parallel Processing*. Springer, 2005, pp. 253–263.

[12] W. Cirne and F. Berman, "A comprehensive model of the supercomputer workload," in *Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop*, ser. WWC '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 140–148.

[13] W. A. Ward, Jr., C. L. Mahood, and J. E. West, "Scheduling jobs on parallel systems using a relaxed backfill strategy," in *Revised Papers from the 8th International Workshop on Job Scheduling Strategies for Parallel Processing*, ser. JSSPP '02. London, UK, UK: Springer-Verlag, 2002, pp. 88–102.

[14] S.-H. Chiang, A. C. Arpaci-Dusseau, and M. K. Vernon, "The impact of more accurate requested runtimes on production job scheduling performance," in *Revised Papers from the 8th International Workshop on Job Scheduling Strategies for Parallel Processing*, ser. JSSPP '02. London, UK, UK: Springer-Verlag, 2002, pp. 103–127.

[15] A. Mu'alem and D. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 12, no. 6, pp. 529 –543, jun 2001.

[16] J. P. Jones and B. Nitzberg, "Scheduling for parallel supercomputing: A historical perspective of achievable utilization," in *Proceedings of the Job Scheduling Strategies for Parallel Processing*, ser. IPPS/SPDP '99/JSSPP '99. London, UK, UK: Springer-Verlag, 1999, pp. 1–16.

[17] Sourceforge project, "The iperf project," http://iperf.sourceforge.net/.