Quantifying the Brown Side of Priority Scheduler: Lessons from Big Clusters

Derya Çavdar
IBM Research
Zurich, Switzerland
avd@zurich.ibm.com

Andrea Rosa
University of Lugano
Lugano, Switzerland
andrea.rosa@usi.ch

Lydia Chen
IBM Research
Zurich, Switzerland
yic@zurich.ibm.com

Fatih Alagöz
Bogazici University
Istanbul, Turkey
alagoz@aboun.edu.tr

ABSTRACT

Scheduling is a central operation to achieve "green" datacenters , i.e., distributing diversified workloads across heterogeneous resources in energy efficient manners. Taking an opposite perspective from most of the related work, this paper reveals the "brown" side of the scheduling, i.e., wasted CPU core resources (so called brown resources), using field analysis and trace-driven simulation of Google cluster trace. First, based on the trace, we pinpoint the dependency between priority scheduling and task eviction that causes brown resources and present a characterization study of different workload priorities. Next, to better understand and further improve the resource "inefficiency" of priority scheduling, we develop slot based scheduler with various tunable parameters, using simulation. Our key finding is that tasks of low priorities suffer greatly in terms of response times as well as CPU core resources because of high probability of being evicted and resubmitted. We develop a simple threshold-based policy that considers the trade-off between task drop rates and wasted CPU cores due to task resubmission. Our experimental results show that we are able to effectively mitigate brown resources without sacrificing the performance advantages of priority scheduling.

1. INTRODUCTION

Big datacenter clusters became the standard IT delivery platforms, whose energy-efficiency is one of the foremost optimization criteria in daily operations. On the one side, several scheduling studies aim to achieve "green" datacenters by intelligently managing workloads [3] so that the energy consumption of IT resources and thermal equipment is minimized and the usage of green energy supply, e.g., renewal energy, is maximized. On the other side, due to ever increasing complexity of systems and dynamicity of workloads [4], scheduling policies employ techniques, e.g., duplicative executions, to mitigate the impact of unexpected events and further avoid the degradation of tail response times [1]. As a result, computation resources are unavoidably spent on such executions. While a vast amount of studies provide the "green" evidences of today's datacenters, such as PUE, little is known about how the computing resources are given away, the so called brown resources, to boost the quality of services of such clusters.

In this paper, our objective is to better understand the brown side of datacenter scheduling and further mitigate the resource inefficiency of the scheduling, using a field trace provided by Google and trace driven simulation. To such an end, we focus on a particular scheduling event in Google trace – task eviction – which is triggered by the scheduler due to tasks congestion, reservation excess or hardware failure. The first step of our study is to qualitatively identify its dominant root cause by analyzing sequence of tasks co-executed on the same server. Our finding that priority scheduling is the main cause for eviction leads us to conduct a workload characterization analysis of different priorities, including inter-arrival times, CPU and memory demands of tasks. To quantify the degree of brown resources due to the priority scheduling, we further develop a simulation-based analysis that models the cluster as parallel and heterogeneous servers configured with a fixed number of slots and cores. We explore various what-if scenarios, i.e., no priority, resumable priority scheduling, and non-resumable priority scheduling, in terms of wasted CPU core resources and response time degradation. Moreover, we propose a simple threshold policy to regulate the eviction process, motivated by experimental results that show a significant improvement in brown resources with a slight increase in task failure rates.

1.1 Google Trace

The Google cluster trace [7] represents a rich heterogeneous workload mix, particularly MapReduce, on a large heterogeneous cluster for 29 days. For more details about the trace, we direct the interested readers to characterization studies [6, 2], whose foci are on providing an overall view of statistical properties about all scheduling events. In particular, our analysis is based on eviction events, which are recorded in the tasks Events table, the tasks Usages table and the Machines events table of the trace, dated between the midnight

of Monday 09-05-11 and the midnight of Monday 16-05-11. As evicted tasks are automatically resubmitted to the scheduler, tasks can experience multiple evictions before leaving the system with (un)successful finish. Combining aforementioned tables, we know the CPU and memory demand[1]of each task, as well as the execution sequence of tasks on the same server. Tasks have also a priority, ranging between [0, 11], where high values represent important tasks.

2. ANALYSIS OF EVICTION

Motivated by the fact that eviction events are non-negligible and are highly controlled by the scheduler, we first try to answer the question – what the dominant root cause of eviction are. Furthermore, we analyze workload characteristics, in terms of arrival processes and resource demands, with respect to the root-causes.

Cause	Tolerance (τ)	Percentage
Machine failure	$\{0, 15, 30, 45\}$ s	0.67%
Memory excess	0	1.74%
Disk excess	0	0.0002%
Higher priority task	1s	93.69%
Higher priority task	5s	93.76%
Higher priority task	10s	93.94%
Higher priority task	20s	94.59%
Higher priority task	30s	95.36%

Table 1: Causes of eviction and their percentages.

2.1 Cause of Eviction

Although the traces provide textual explanation of why eviction takes place, there is no ready information or explanation for individual eviction, i.e., single eviction event are not motivated. As a result, we categorize every eviction based on certain causes, summarized by the trace document and our assumptions, and identify the most critical one.

The following event can cause eviction: (a) machine failure; (b) arrivals of higher priority tasks; (c) disk failure; (d) reservation excess. In addition to them, we inspect a fifth event, resource excess, since some eviction events have correlation with it. As the traces don't contain information related to disk failures, we are unable to analyze their impact on eviction. Analysis of the reservation excess requires complete information about the requested resources of all the co-executed[2]tasks at eviction time, which we are unable to find for all the evictions due to the particular traces format and the truncated observation window. We choose to not show this analysis, believing that the strong results obtained on the other causes are enough to highlight the main cause of eviction.

For each task eviction, we try to inspect if each cause withstands under multiple thresholds. For example, we count how many eviction records are present in the traces within 0,15, 30 and 45 seconds from a machine failure. Note that a single eviction can be caused by multiple events. In the rest of the section, we analyze, in order, the relevance of following causes: machine failure, resource excess and preemption due to higher priority tasks. We summarize our results in Table 1.

2.1.1 Machine Failure

To correlate machine failures with eviction, we compute the downtime interval of each machine, and compare them with each eviction time stamp. We identify their dependency by a time threshold τ, to account for potential profiling delay. As long as the time stamp of an eviction is τ seconds after the start of machine downtime interval, we consider it as caused by machine failure. We apply multiple values for τ, i.e., 0, 15, 30, and 45 seconds, and obtain the same results for all of them, i.e., only 0.67% of eviction is caused by machine failure. Therefore, we conclude that machine failure is not a main cause for eviction.

2.1.2 Resource Excess

In this section we aim to understand if tasks are evicted due to consuming more resources than the ones they request at submission time. We note that, according to the traces document [5], tasks with resource excess should be terminated by the scheduler using kill event. Surprisingly, we find that some eviction events can also be related to resource excess. In particular, we focus on memory and disk and overlook CPU, which is allowed to use spare capacity as clearly stated by the documentation. For each eviction, we determine whether the maximum memory or mean disk usages are greater than the requested ones, by extracting this information from the Events and Usages tables. We use the mean usage of disk because the maximum disk usage is not provided by the traces.

Results summarized in Table 1 show that only a very low percentage of eviction is related to memory excess, roughly 1.74%, and nearly negligible percentage of eviction is related to disk excess. Our results lead us to conclude that resource excess is not a dominant reason to cause eviction of tasks.

2.1.3 Preemption by Higher Priority

In situation of reservation excess, high priority tasks may preempt the execution of other tasks of lower priority. We propose a simple approach to see if higher priority tasks cause eviction, by comparing the time stamp of eviction and the scheduling time of higher priority tasks on the same machine within a time threshold of 1, 5, 10, 20 and 30 seconds. To better mark the relationship of tasks in an eviction event, we name "kick-in" the task that preempts lower priority tasks, while we use the term "kicked-out" for the preempted ones. We first query the Events Table and extract all the eviction records, then, for each of them, we search for potential "kick-in" tasks considering three criteria, i.e., time stamp, priority and the machine ID in the Events table. In particular, a task is classified as "kick-in" when the the following condition holds: $t_{KO} \leq t_{KI} \leq t_{KO} + \tau$, where t_{KO} denotes the time stamp of eviction, t_{KI} is the scheduling time of the kick-in task and $\tau \in \{1, 5, 10, 20, 30\}s$. Machine ID of kick-in and kicked-out tasks must coincide.

As shown in Table 1, we can identify that roughly 93% of eviction has corresponding kick-in tasks with higher priorities, for all the threshold values considered. In contrast to

[1]All resource demands are normalized between 0 and 1 by the largest machine resource capacy in the trace.
[2]With this term, we refer to all the tasks in execution in the same machine of the evicted tasks at eviction time.

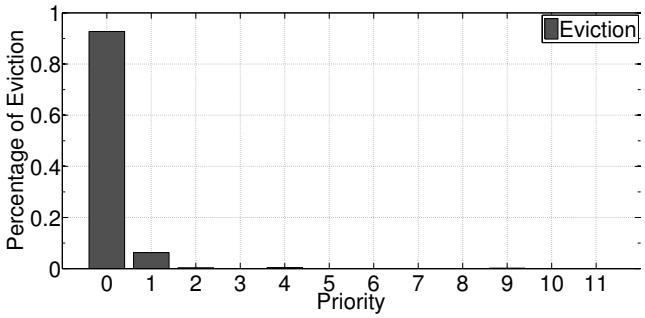


Figure 1: Distribution of eviction across priorities.

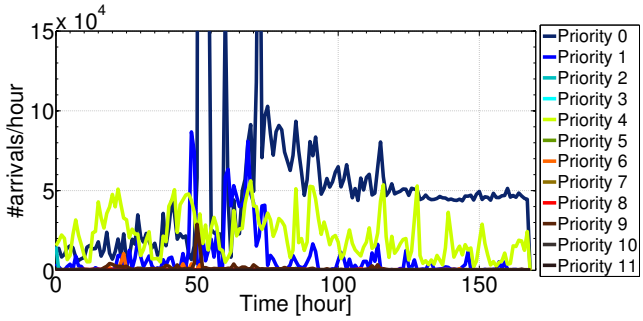


Figure 2: Hourly task arrival rate, breakdown by priority.

lower percentages of other causes for eviction, scheduling of high priority tasks on the same machine appears to be the most relevant and dominant cause. Such an observation leads us to conduct an in-depth analysis on the relationship between kick-in and kicked-out tasks.

2.2 Eviction per Priority

Verified that tasks are evicted mainly due to priority preemption, as highlighted in Section 2.1, here we aim to study how different priorities affect the eviction process. To this end, we analyze the relationship between kick-in and kicked-out tasks with a special focus on their priority.

Figure 1 shows the distribution of kicked-out tasks across priorities. Priority 0 alone includes the 92.73% of evicted tasks, while the 99% of eviction affects the two lowest priorities. Number of evicted tasks decreases exponentially when the priority increases.

Preemption is mainly caused by priority 4 tasks. Indeed, when looking at correlations among scheduled and descheduled priorities, the 70% of eviction events is composed by priority 4 tasks preempting priority 0 ones.

Overall, distribution of eviction across priorities clearly emphasizes the central role of priority in the preemption process, where low priorities can be descheduled with very high probability, whereas high priorities are rarely preempted.

2.3 Workload Analysis of Priorities

Our aim in this section is to study the main characteristics of different priorities in the cluster. We consider individually each priority along two dimensions: (1) arrivals pattern and (2) resource demands. Particularly, we only consider resource demands of tasks starting and ending in our one week observation window. For the arrival patterns, we study the task interarrival time of each priority in the system. To better quantify the impact on physical resources, we use a metric called *resource demand*, defined as the product of task service demand and the amount of resources requested by tasks at arrival time. By doing so, we include in a single metric both the quantity of requested resources and the time for which they're reserved for the task. We consider two different types of resources, i.e. number of cores and memory, and use separate units of measurement for each resource demand, i.e., $CPU \cdot s$ and $RAM \cdot s$ respectively.

We summarize the mean values of task interarrival time and resource demand for each priority in Table 2. Interarrival time shows large difference among different priorities. Arrivals at priorities 0, 1 and 4 are very frequent, as multiple tasks are submitted to the cluster every second. Indeed, these priorities cover the 97.35% of the task schedulings observed in one week. On the contrary, priorities 3 and 5 rarely experience new arrivals, resulting in high interarrival times. Higher priorities do not necessarily correspond to higher interarrival times: for example, this time is low for priorities 8 and 9 and very high for priorities 3 and 5.

Conversely, high priority tasks clearly impact more than lower priorities on physical resources. As one can see by Table 2, priority 11 tasks have, on average, a CPU demand which is 29 times higher than the overall mean (reported at the bottom of the table) and their average RAM demand is 42 times higher than the global mean RAM demand. Low priorities, instead, show lower and more balanced resource demands. Overall, tasks request more CPU demand than RAM, especially at low priorities.

Finally, we report the hourly arrivals pattern for each priority in Figure 2. The arrival rate is clearly highly fluctuating regardless of the priority, as the number or task submissions in consecutive hours can easily vary of 40000 units. In particular, priority 0 experiences strong and swift peaks in the hourly arrival rate, reaching up to 1.1 millions of arrivals per hour.

In conclusion, our analysis shows that high priorities have higher impact on physical resources than lower priorities, whereas arrival pattern and interarrival time are not affected by the task priority.

3. SYSTEM MODEL

To explore various what-if scenarios of various settings of priorities, we develop a detailed systems model that capture the energy consumption and response times when executing complex workload on heterogeneous systems. The system consists of three parts: the central queue, multiple machines, and the scheduler. Each machine is equipped with certain CPU and memory, and configured with a fixed number of slots based on number of cores. Moreover, as only the normalization values are given in the trace, we further assume 1 unit CPU have homogenous 16 cores and 16 slots, and scale accordingly to other unit, e.g., 0.5 unit CPU has 8 cores/slots. In the following, we describe in detail the basic

Priority	Interarrival time [s]		CPU demand [$CPU \cdot s$]		RAM demand [$RAM \cdot s$]	
	Mean	SD	Mean	SD	Mean	SD
0	0.059153	0.40471	133	544.88	78.673	358.3
1	0.43635	3.6045	79.858	293.98	66.002	657.69
2	14.586	59.572	535.81	3045.1	286.77	1533.7
3	17876	48617	0.38941	0.4447	0.039666	0.032372
4	0.16401	2.2769	55.258	404.09	30.781	194.23
5	30601	44586	0.15221	0.10224	0.023401	0.020716
6	7.5633	92.299	14.305	41.019	5.3376	22.769
7	2243.9	3610.2	NaN	NaN	NaN	NaN
8	9.0927	21.976	36.742	666.53	56.074	607.48
9	2.9024	5.4195	1034.6	3459.2	965.34	2925.7
10	86.006	97.878	2307.9	6333.4	2979.6	7305.1
11	1144.3	1320.1	2497.7	4826.2	2260.1	2835.1
All	0.0385	0.3128	85.481	589.15	54.52	451.27

Table 2: Per-task mean interarrival time and resource demand, breakdown by priority. No demand value is given for priority 7 as no task of such priority terminated in the observation window.

discipline of the scheduler and response time model.

3.1 Response Time Model

In addition to the detailed model on energy consumption, we also model the task response time defined as time between task arrival and departure. Response time of a task, $T_{response}$ is composed of two terms: $T_{execution}$ is the successful execution time of the task and the rest is called T_{waste} which is the time spend other than useful computation.

The first contributor of $T_{response}$ is the execution time of the task, $T_{execution}$. In our system we use multicore property for servers. We also allow concurrent execution of multiple tasks (up to a limit) on a single cpu core of a server. A task can not use on more than one core, then the maximum cpu rate that a task can run is limited with by core capacity. Therefore, minimum execution time of a task is bounded by cpu demand of the task divided by core capacity. Concurrently running tasks equally share the cpu capacity of the core. As a result of these properties $T_{execution}$ of a task is tied to both core capacity and number of concurrently running tasks on that core.

We calculate the assigned CPU rate of the task with the following equation at each time slot. The idea is equally sharing CPU processing capacity among tasks running on the same core.

$$\Omega_i^{cpu}(t) = \frac{\Delta_{s,j}^{cpu}}{N_{s,j(t)}} \tag{1}$$

where $\Omega_i^{cpu}(t)$ is the assigned CPU rate of task i at time slot t. $N_{s,j}(t)$ is the number of concurrently running tasks on core j of server s at time slot t. $\Delta_{s,j}^{cpu}$ is the cpu processing capacity of core j of server s, it is independent of time. In the later experiments we limit the concurrency level of that server(φ_s), i.e. $N_{s,j} \leq \varphi_s$ where $\varphi_s \geq 1$.

We can give an illustrative example here. Assume $\Delta_{s,j}^{cpu}$ cpu processing capacity of core j is 1. φ_s is 4. At time t; $N_{s,j}(t)$ is 2. Hence $\Omega_i^{cpu}(t)$ where $i = 1, 2$ is 0.5, as equally shared. At time t+1; two new tasks are assigned to core j then $N_{s,j}(t+1)$ is 4. Therefore, $\Omega_i^{cpu}(t)$ where $i = 1, 2, 3, 4$ is 0.25. Assigned cpu rate of a task is updated at the beginning of each slot.

$$CPU_{demand}^{i} = \sum_{t} \Omega_i^{cpu}(t) \tag{2}$$

The second term of $T_{response}$ is T_{waste} may composed of several terms like T_{wakeup}, the other terms T_{queue}, and $T_{execution}^{wasted}(P)$. $T_{execution}^{wasted}(P)$ strictly depends on scheduling algorithms, system properties and system load. We investigate the effects on this terms in the following sections.

4. QUANTITATIVE BROWN ANALYSIS

As the priority scheduling governs eviction, there exists a performance trade-off among priorities. This lead us to explore its advantages via what-if simulation, in combination with simple assumptions. Using a slot-, and core-based simulator driven by Google trace, we compare the average response times and wasted core seconds of tasks execution across priorities, under different priority scheduling schemes and no-priority scheduling.

In particular, we evaluate a datacenter of 100 server nodes. The aim of our work is two-fold. First, we show how the brown resources and response times are effected by the introduction of basic priority schemes, namely resume v.s. non-resume. Second, we apply several simple threshold policies to minimize the wasted CPU core seconds and improve the response times due to repetitive evictions of the low priorities.

4.1 Scheduling Policies

In particular, we consider preemptive priority scheduling, i.e., high priority tasks can preempt any execution of low priority tasks. Upon preemption, low priority tasks immediately rejoin the central queue. As for tasks that are evicted, we specifically consider two types of resuming schemes for their executions: resume and non-resume. In the former, all evicted tasks resume from the point of interruption, whereas in the non-resume case all evicted tasks start from the beginning. When there are multiple tasks are from the the same priorities, we adopt two types of policies to select particular tasks, i.e., random (RND), and most recently start (MRS). Moreover, motivated by the high number of repetitive eviction of low priorities, we propose to apply a threshold on

the number of eviction a task can experience. Once such a threshold is reached, a task is dropped.

In summary, to quantitatively compare the impact of introducing priority scheduling, we benchmark the following scheduling policies against the no priority (NOP) where every task waits in the central queue in a first-come-first-serve manner until an available.

- Priority-RND(RND). Every task waits in queue in decreasing order of their priorities. At every slot, the scheduler checks, for all tasks in the central queue, if there're lower priority tasks in execution in the cluster and randomly kicks out one having the lowest priority, if exists.
- Priority-RND-drop$N$(RNDN). Similar to Priority-RND, with the difference that the scheduler drops tasks whose number of resubmissions is greater than N.
- Priority-MRS(MRS). Similar to priority-RND, with the difference that high priority tasks in the central queue kick out lowest priority tasks in machine with the most recent starting time.
- Priority-MRS-drop$N$(MRSN). Similar to Priority-MRS, with the difference that the scheduler drops tasks whose number of resubmissions is greater than N.

Note that we study both resume and non-resume scenarios. However due to the lack of space, we first focus on the high level comparison between resume and non-resume and the detailed analysis on different priorities of non-resume case.

4.2 Resume vs. Non-resume

To evaluate the pros and cons of using priority scheduling, we implement different policies for a non-resume system. In non-resume systems the execution of the task before eviction is lost. However, in resume systems when an eviction occurs, the evicted task execution is only suspended and no execution cost because of eviction is experienced. As a result, non-resume systems are more utilized than resume systems. We show the details of comparison of resume and non-resume systems in Table 3.

We compare resume and non-resume systems in terms of average response time, wasted core seconds and probability of drop. Since evicted tasks continue their execution without any loss, wasted execution due to eviction is not experienced. On the other hand in non-resume system the system utilization is increased due to due to restarted executions. Moreover, the non-resume system has higher response time and higher drop probability. In terms of response time among policies, policies employing drops has lower average response time.

4.3 Threshold-based Priority Scheduling

Wasted CPU Core Seconds. In Figure 3(a), we investigate the wasted core seconds for different policies under non-resume system. When MRS policy is not adopted we see more wasted execution, since the evicted task is selected randomly. Since MRS evicts the youngest task among the lowest priority, we observe less high priority class execution waste than RND policies. And if we allow *drops* in addition to MRS policy, we achieve less execution wasted hence a greener system. By allowing drops, the system get rid of the repetitively evicted tasks. Comparing MRS and MRS3, the wasted time of MRS3 is significantly lower than MRS by dropping 5851 tasks, and resulting average response times is lower.

Response Time. In Figure 3(b), we look deeper into better see the tradeoff among priorities, we further break down the response time by priority under different policies. As expected when priority scheduling is applied, the higher priority tasks finish in smaller time on the other hand the low priority tasks stays in the system longer due to waiting and evictions. As RND incurs a significant performance disadvantages on priority 0, the resulting average response times are visibly higher than NOP. From our results, we advocate to apply priority scheduling, in combination with age and drop limits, so that the best trade-off between high and low priorities can be achieved. Furthermore, we also suggest the drop limits can affectively mitigate negative performance impact for repetitive unsuccessful executions, such as fail in the trace. In terms of response time, MRS3 gives best results by providing small response time for high priority classes and also reasonable response time increase in the lowest priority class.

Percentage of Task Drop. In Figure 3(c), we compare resume and non-resume systems in terms of number of drops. In the resume system, the evicted tasks continue its execution hence there is no wasted execution cost incurred. As a result the system load for resume system is lower than non-resume system. Since the system is more loaded for non-resume, we expect higher number of evictions hence higher number of drops.

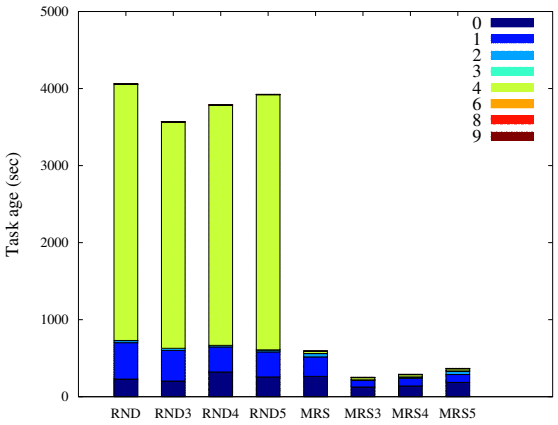
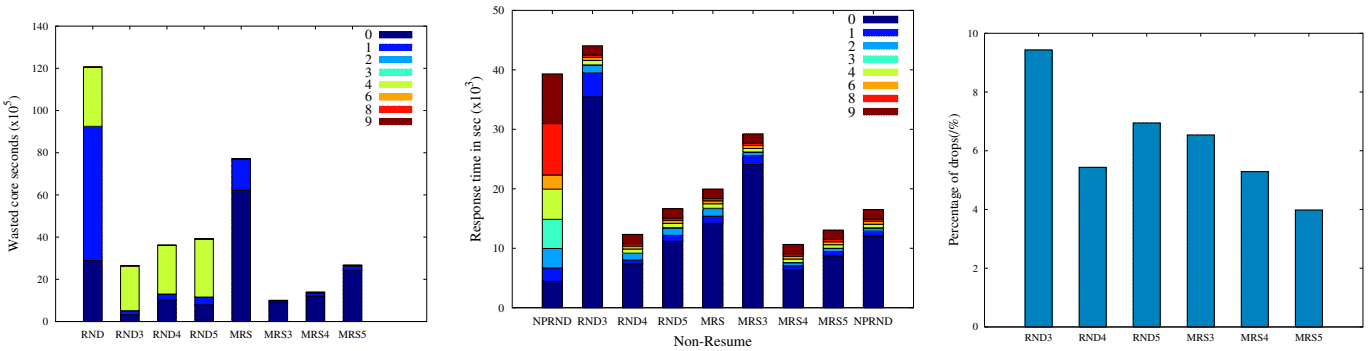


Figure 4: Task age

Task Age. In Figure 3(c), we look at how the wasted core seconds are changing with different policies, in Figure 4 we look at the wasted execution part is distributed among the priority classes with different policies. In other words, how long the tasks are executed before experiencing eviction, termed as task age. Since the MRS policy is always selecting the youngest(most recently started) task among the

	Resume				Non-Resume			
Metrics	RAND	RAND3	MRS	MRS3	RAND	RAND3	MRS	MRS3
wasted core sec(10^3)	0	0	0	0	12048	2645	7711	998
average response time	4199	1467	4207	2060	11460	2357	7765	2143
probability of drop	0	0.067	0	0.055	0	0.094	0	0.065

Table 3: Resume versus Non-resume



(a) Wasted core seconds (b) Response time by class (c) Percentage of drops

Figure 3: Analysis with different policies

lowest priority class, it achieves significant improvement on the class 4's evictions compared to randomly selecting the evicted task.

We summarize the simulation results of applying threshold on the number of eviction, i.e., RNDN, and MRSN can, effectively mitigate the brown resources and improve the response times of all the priorities.

5. CONCLUSION

In this paper, we provide an quantitative analysis on the resource waste due to the scheduling policy, in particular the priority scheduling. Our analysis is based on a large scale cluster trace provided by Google and the trace driven simulation that has detailed modeling of system scheduler and response times of different priority tasks. Our key findings are that priority scheduling causes a non-significant number of evictions. Moreover, such the large scale systems are dominated by lower priority tasks and high priority tasks require higher resource demands. Using a trace driven simulation, we show that there is a significant amount of brown resources associated with basic priority scheduling. To mitigate the wasted resources and response time degradation, we proposed to impose a threshold on the number of evictions and our results show a significant improvement on the resource efficiency of priority scheduling.

6. REFERENCES

[1] L. Barroso, J. Dean, and U. Hölzle. Web Search for a Planet: The Google Cluster Architecture. *IEEE Micro*, 23(2):22–28, Mar. 2003.

[2] S. Di, D. Kondo, and C. Franck. Characterizing Cloud Applications on a Google Data Center. In *ICPP*, 2013.

[3] Z. Liu, Y. Chen, C. Bash, A. Wierman, D. Gmach, Z. Wang, M. Marwah, and C. Hyser. Renewable and cooling aware workload management for sustainable data centers. In *Proceedings of SIGMETRICS*, pages 175–186, 2012.

[4] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *ACM SoCC*, pages 7:1–7:13, 2012.

[5] C. Reiss, J. Wilkes, and J. L. Hellerstein. Google cluster-usage traces: format + schema. Technical report, Google Inc. Revised 2012.03.20. Posted at URL `code.google.com/p/googleclusterdata/wiki/TraceVersion2`.

[6] C. Reiss, J. Wilkes, and J. L. Hellerstein. Obfuscatory obscanturism: making workload traces of commercially-sensitive systems safe to release. In *IEEE CLOUDMAN*, pages 1279–1286, 2012.

[7] J. Wilkes. More Google cluster data. Google research blog. `https://code.google.com/p/googleclusterdata/wiki/ClusterData2011_1`, 2011.