Sparkle: Adaptive Sample Based Scheduling for Cluster Computing

Chunliang Hao[1,3], Jie Shen[2], Heng Zhang[1,3], Xiao Zhang[1,3], Yanjun Wu[1], Mingshu Li[1]

[1]Institute of Software, Chinese Academy of Science, Beijing, China
[2]Department of Computing, Imperial College London, London, UK
[3]University of Chinese Academy of Science, Beijing, China
{chunliang}@nfs.iscas.ac.cn,{js1907}@imperial.ac.uk,{zhangheng,zhangxiao}@nfs.iscas.ac.cn,
{yanjun,mingshu}@iscas.ac.cn

Abstract

Traditional centralised scheduling has becoming unsuitable to analytics clusters with ever growing workload. As a promising alternative, sample based scheduling is highly scalable and, due to its decentralised nature, immune from becoming potential system bottleneck. However, the existing design could only function well in very specific application scenarios. Specifically, we argue that the performance of the baseline sample based scheduling method is sensitive to workload heterogeneity and the cluster's individual worker strength. In this work, we propose a novel method to reduce these sensitivities. We implement our method in the Sparkle scheduler and demonstrate our scheduler is capable of adapting to a much wider range of scenarios. Instead of introducing extra system costs, Sparkle's improved performance is gained by cutting unnecessary wastes and reducing the number sub-optimal scheduling decisions. Hence it could also serve as a foundation model for further studies in decentralised scheduling.

Categories and Subject Descriptors C.2.4 [*Cloud computing*]: Software architecture; K.6.4 [*Management of computing and information systems*]: System Management–Centralization/decentralization

Keywords Cluster computing, sample based method

1. Introduction

In the near future, the clusters are expected to expand further in both size and workload[6][7]. This trend poses an enormous challenge to the widely used centralised schedulers. As the sole bridge between jobs and workers in a cluster, a centralised scheduler may become a potential system bottleneck. Although the scheduler can be continuously enhanced, each upgrade may also increase its complexity and make it harder, and eventually, impractical to maintain. Hence, adopting decentralised schedulers would be a more effective solution in the long run[8].

One way to build decentralised schedulers is to use a sample based design [3]. The core idea is that whenever a scheduler needs to make a decision about task placement, it no longer decides among the complete set of workers in the cluster, but only chooses from a small randomly sampled subset (known as the 'sample cluster'). To maintain overall load balance, each scheduler redraws its sample cluster for every new task. Following this strategy, there is no longer a need to maintain the complete logic graph connecting all jobs to all workers. Hence multiple schedulers may co-exist in the same cluster, functioning independent from each other [5]. Our DataOS project in ISCAS aims at building next-generation data-centric cloud operating system, of which decentralised scheduling method is an integral part.

However, the aforementioned method only fits into a narrow scope of usage. We identify its two weaknesses, namely, the sensitivities to workload heterogeneity and individual worker strength. Specifically, its performance (in terms of job delay) deteriorates quickly when the workload becomes more heterogeneous and when the individual workers are weak (i.e., each containing fewer parallelisable slots). Although the two issues have been long existed for centralised scheduling as well and there have been a number of solutions [1][9][13][15] proposed by the community, these solutions cannot be easily adapted to sample based schedulers.

In this paper, we propose two novel methods to tackle these problems in sample based scheduler. Firstly, we use a worker-end prediction method called "traffic light" to better support heterogeneous workload. The traffic light gives a quick and rough estimation about the worker's availability during the scheduling process, providing scheduler with evidences toward better decision. It can also serve as the basis for extra policies such as delayed scheduling. Secondly, to reduce the impact of weak workers, each scheduler in our de-

CloudDP '15, April 21–24, 2015, Bordeaux, France.
Copyright © 2015 ACM 978-1-4503-3478-5/15/04…$15.00.
http://dx.doi.org/10.1145/2744210.2744212

sign maintains a worker blacklist. To be used in conjunction with traffic light, the blacklist enables the scheduler to make full use of information acquired from previous scheduling sessions, thus further reduce the number of sub-optimal decisions.

We implemented our methods in an adaptive scheduler named Sparkle, which has been tested on a heavy weight simulated cluster. The result shows that comparing to the baseline method Sparkle could reduce up to 40% of the extra job delay caused by workload heterogeneity. It also better maintains its performance while switching to a cluster with weaker workers, reducing up to 20% of delay in weak worker cluster.

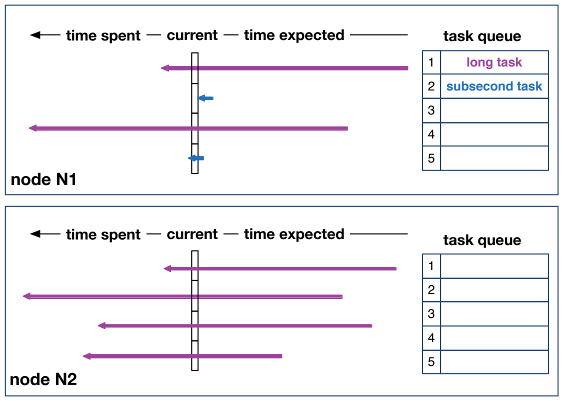
The remainder of this paper is organised as follows. Section 2 presents background and related works. Section 3 presents the design of Sparkle. In Section 4, experiment results are discussed. Finally, Section 5 concludes this paper with some remarks on future work.

2. Related Work

Centralised scheduling is neither the only nor the ideal method for cluster computing[8]. Many efforts have been invested to explore alternative, decentralised, methods. Standalone resource management like in Mesos[2] and Yarn[10] splits the centralised structure into layered structure, so that the load of scheduler could be shared. These methods, however, are still not easily scalable. Auction based method[12] offers dynamic and decentralised structure based on attract economy model. Omega[8] provides flexible and scalable design based on shared state mechanism and optimistic conflict expectations. Sparrow[5] optimize sample (probe) based design for sub-second parallel jobs while keeping the inherited high scalability. Our work extends sample based design, enabling it to better handle heterogeneous workload. The impact of workload heterogeneity has been widely investigated for centralised (monolithic) scheduling[13]. Reorganisation of job and task[11], invoke intentional delay[14], implement priorities and preemptive action[4], execution estimation[9], etc, could all, to some extend, reduce such impact. In this paper we focus on how such improvement could be made for sample based structure.

3. The Sparkle Scheduler

The high scalability of sample based scheduling is a clear advantage over the more traditional centralised scheduling strategies in today's ever-expanding clusters. However, in practice, the performance of sample based scheduling can be hindered by a number of issues, preventing the method to reach its full potential. In Sparkle, we try to tackle the two most prominent ones, namely, the method's sensitivity to workload heterogeneity and individual worker strength. The principle of our design is to reduce the impact by cutting unnecessary waste and the number of sub-optimal scheduling decisions.



time spent — current — time expected
task queue
long task
subsecond task
1
2
3
4
5
node N1
time spent — current — time expected
task queue
1
2
3
4
5
node N2

Figure 1. The availability status of two workers (running on nodes N1 and N2, respectively) in a sample cluster. Both workers have 4 slots for parallelised execution of tasks.

To keep this work concentrated, a number of simplified assumptions have been made. Firstly, following the "power of two" law[3], we fix the sample cluster size to two workers. Secondly, since we are focusing on task-level scheduling in this study, we assume each job only contains one task. Nonetheless, existing methods about how to efficiently break jobs into tasks can be directly applied on top of Sparkle in practice.

3.1 Traffic Light

Traffic light is designed to mitigate the impact of heterogeneous workload in sample based scheduling.

Because sample based scheduler allocates tasks to a small subset of workers, it is more likely to need to choose between two busy workers, hence is more prone to making poor decisions because of workload heterogeneity. An example is shown in Figure 1. There are two workers on node N1 and N2, respectively, and both are busy. When these workers form a sample cluster, according to queue length, N2 should be selected for the new task because it has an empty task queue. In this case, the waiting time of the new task would be proportional to the execution time of long tasks. In comparison, although there are two tasks queued in N1, the waiting time for a new task would be only in proportion to the length of sub-second tasks. Therefore, although N2 has a shorter task queue, allocating the new task to N2 is actually a sub-optimal decision.

We aim at reducing these sub-optimal scheduling decisions by utilising a worker-end prediction method called traffic light. In our method, in addition to queue length, the sampled workers also provide a more informative discrete indicator (the traffic light) about their availability. The indicator can be in one of the following three states. A green light represents that there are resources (i.e., empty slots) immediately available in this worker, hence choosing it is recommended. A red light means the worker is busy and the waiting time for the new task is expected to be unacceptably long, thus the scheduler should avoid placing the new task on this worker. A yellow light tells that the worker is busy but will have resources available in the near future. The pseudo code for determining traffic light colour is given in Algorithm 1. As illustrated in the code, the difference between red light and yellow light is that in the former case, the new task would need to wait for the completion of at least one task having an expected execution time an order of magnitude longer than its own. Choosing a yellow coloured worker does not guarantee low responding latency, but it does make sure the new task's waiting time is in proportion to its own length.

Algorithm 1 Calculate Traffic Light Colour

```
Require: The new task being scheduled, $task$; The worker
    running this calculation, $worker$;
Ensure: The traffic light colour, $TL_{worker}$
1: $m$ = the number of tasks running in $worker$ with a
    duration order of magnitude longer than $task$;
2: $n$ = the number of tasks in $worker$'s queue with a
    duration order of magnitude longer than $task$;
3: $sc$ = the total number of slots in $worker$;
4: If $worker$ is not busy, $TL_{worker}$ =GREEN;
5: Else If $m+n < sc$, $TL_{worker}$ =YELLOW;
6: Else $TL_{worker}$ =RED;
```

Figure 2 illustrates how traffic light is used in Sparkle. When a sample cluster is formed, the scheduler sends a probe with task description to all sampled workers. The calculation of traffic light then takes place on the worker-side. After receiving the status information (which contains the traffic light colour) from all workers, the scheduler allocates the task based on the following policies:
P1. Choose any green coloured worker if at least one exists;
P2. If all workers are either yellow or red, choose the yellow worker with shorted task queue length;
P3. If all workers are red, choose the one with shortest task queue length;
This policy set is also configurable in order to meet the specific needs of different application scenarios. Moreover, the simplicity of the traffic light also allows it to serve as a basis for other existing methods, such as delayed scheduling.

3.2 Worker Blacklist

The effectiveness of the traffic light method alone can be less significant when the cluster is constructed from a large number of weak workers. With both the workload and overall cluster strength stay unchanged, a scheduler would still have a higher probability to encounter only red-coloured workers in its sample cluster when each worker contains fewer slots. When the traffic light can no longer provide discriminative information, tasks would be scheduled in the same way (i.e., only based on worker's task queue length) as

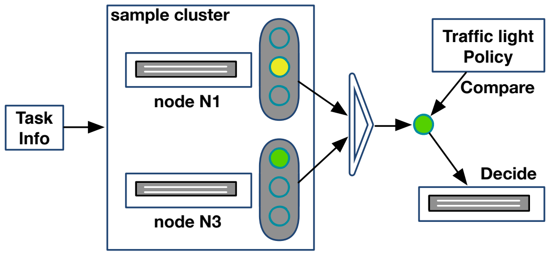


Figure 2. Using traffic light in Sparkle.

in the baseline sample based scheduling method, thus obtaining no improvement. Aiming at eliminating these degenerative situations, we propose the worker blacklist mechanism as a further refinement.

The blacklist mechanism reduces the chance of drawing red-colour workers by reusing the worker status information collected from previous scheduling sessions. The baseline sample based scheduler is stateless, meaning that no worker status statistics is accumulated over time. After each scheduling session, the information received from sampled workers is discarded completely. This approach is based on the assumption that worker status can change extremely fast. However, with traffic light introduced, it can be deduced that this assumption does not hold for red-coloured workers. Since these workers are executing tasks at least an order of magnitude longer than the task being scheduled, their status (in terms of traffic light colour) is unlikely to change when a subsequent task of the same type arrives. Therefore, by recording these workers in a blacklist and avoiding choosing them when forming a sample cluster for the subsequent task, the chance of drawing red-coloured workers can be reduced.

Specifically, the worker blacklist mechanism works as follows. We use a separate worker blacklist for each type of tasks and each scheduler maintains its own set of blacklists. When a red-coloured worker is identified during scheduling a task of type T, it is put into the blacklists for type T and all other task types of a shorter length. These entries would have an expiration time set to T's length. When a new task of type T arrives, the scheduler would first mask all workers in blacklists for type T and all other task types of a longer length and then form a sample cluster from the remaining workers.

Under high task throughput scenarios, the blacklists would mask a considerable amount of unsuitable worker before the scheduling starts, thus lead to much better scheduling performance. This improvement is especially noticeable in clusters with many weak workers, which are more susceptible to becoming red workers.

3.3 Sparkle Architecture

Illustrated in Figure 3, Sparkle combines traffic light and worker blacklist with the sample based process. The proce-

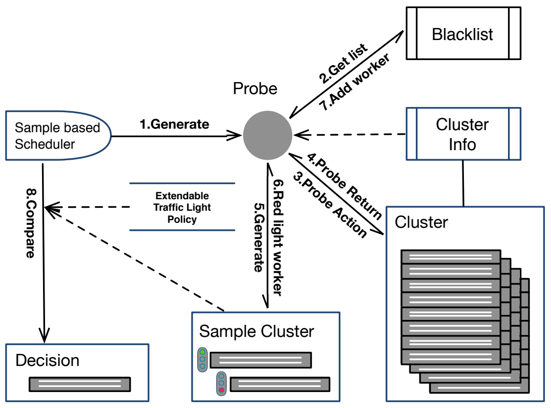


Figure 3. Sparkle's scheduling dynamics.

s	Number of slots per worker
t	Mean task service time
z	Sample size per task
h	Number of schedulers functioning
q	Queue length of a busy worker
λ	Mean request arrival rate
ρ	Overall Cluster load (fraction non-idle slots)

Table 1. Summary of notation.

dure of Sparkle's scheduling dynamics is labelled as eight steps in the figure. The procedure starts when a task is about to be scheduled. In step 1, the scheduler generates probes for information gathering. Step 2 sees the scheduler checking the blacklists to make sure all target workers in the probe are likely to be either green or yellow. In step 3, the probe action requests all target worker to response. When receiving the probe in step 4, workers calculate their traffic light colour and return their status. All information then gathered from the sample cluster in step 5 and red-coloured workers are identified in step 6. These workers are recorded in the blacklists in step 7. Finally, in step 8, a scheduling decision is made according to predefined policies. Note that when the policy is default to delayed scheduling, step 8 may also actively start a re-scheduling process.

3.4 Analysis

In this section, we give a brief analysis about traffic light. Notations used are listed in Table 1. When the cluster is under load ρ, the probability of a size z sample cluster containing at least one idle slot is: $1-\rho^{z*s}$. This also means for each scheduling session, there is a ρ^{z*s} chance to encounter only busy workers in the sample cluster. Without traffic light, this is when a sub-optimal scheduling decision may be made, as comparison based on queue length only makes sense with homogeneous workload.

Although traffic light does not change the proportion of available workers, it categorises busy workers a step further, thus facilitates better informed decision making. Consider the situation when the scheduler is working with heterogeneous workload, for every particular task to be scheduled, the workload can be divided into two set: set A for tasks being the same or order(s) of magnitude shorter than the new task and set B for all other tasks. In this case, for any busy slot, the probability of running a task in B is: $\lambda_B * t_B/(\lambda_A * t_A + \lambda_B * t_B)$ (denoted by φ_B). Hence in a busy worker with q queued tasks, the probability that x task(s) from B exist in this worker is: $\binom{x}{s+q} * (\varphi_B{}^x) * ((1-\varphi_B)^{s+q-x})$ (denoted by $\psi_B(x)$). The probabilities of different traffic light colours are shown as in equation (1).

$$\text{P}(light) = \begin{cases} 1-\rho^s & light \text{ is GREEN} \\ \rho^s * \sum_{x=0}^{s-1} \psi_B(x)) & light \text{ is YELLOW} \\ \rho^s * \sum_{x=s}^{s+q} \psi_B(x)) & light \text{ is RED} \end{cases} \tag{1}$$

With traffic light enabled, we have a P(RED) chance of declaring one worker in the sample cluster as temporarily inappropriate for the given task type. This enables the scheduler to reduce misjudgements between multiple busy workers, hence increase the cluster performance. More importantly, it allows us to take future measures on these red light workers. At high rate of $\lambda * z/h$, traffic light is evaluated frequently and red-coloured workers are put into blacklists. Hence during each scheduling session, the number of workers to choose from is reduced. Since the number of 'good' (non-red-coloured) workers stays the same, this leads to a decrease in the expected averaged waiting time.

4. Experimental Evaluation

We performed two experiments to examine the performance of Sparkle. The first experiment evaluated how Sparkle compares to the baseline sample based scheduling method under different levels of workload heterogeneity. The second experiment tested Sparkle's adaptability to weak workers. We implemented Sparkle and the baseline scheduler in a custom-built heavy-weight simulation platform. All schedulers and the simulation platform is available online at: `https://github.com/chunliang-hao/Sparkle`.

In our experiments, we used two different metrics for the evaluation of scheduling performance. One is the average relative delay in task response, which reflects the ratio between a task's actual response time and its expected execution time. This metric is used (instead of absolute delay) because it is normalised for different of task types in heterogeneous workload so that the measurement would be unbiased. The second metric we chose is the proportion of tasks with a relative delay exceeding 0.1. This metric is chosen because, in practice, one common requirement in SLA is to have a maximum relative delay of 0.1.

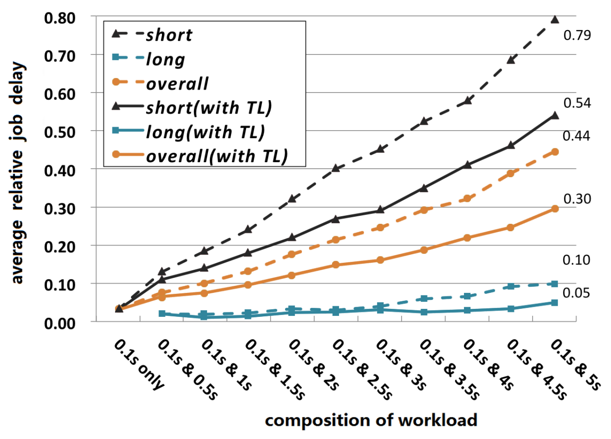


Figure 4. Average relative response delay under different workload heterogeneity levels when using the schedulers with and without traffic light.

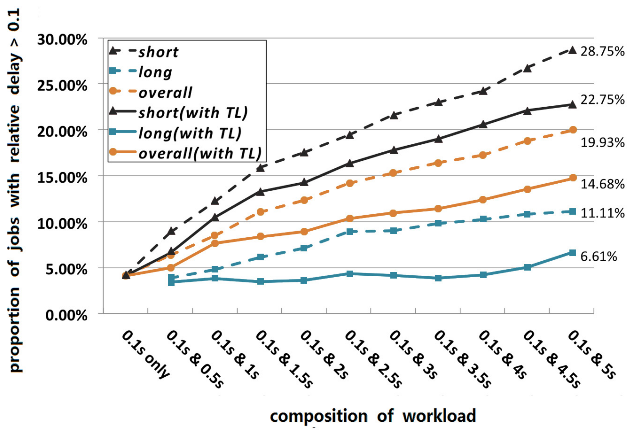


Figure 5. Proportion of unqualified jobs under different workload heterogeneity levels when using the schedulers with and without traffic light.

4.1 Sparkle with Heterogeneous Workload

We compared sample based schedulers with and without traffic light on a simulated 200 worker cluster, with each worker containing 8 slots. Sample size was fixed to 2 and one-task-jobs were used for simplification, as discussed earlier. We started with a homogeneous workload with only 0.1s jobs. We then ran 10 subsequent tests, each having a workload containing both short (0.1s) job and long (0.5s 5s) job. The arrival intervals for both types of jobs followed an identical Poisson distribution. In all tests, the overall workload was adjusted so that cluster was stable at 80% utilisation.

The performance of the two schedulers in terms of relative response delay is illustrated in Figure 4. As shown in the figure, the schedulers performed almost the same under homogeneous workload. In this case, the small difference between their performances could be attributed to the randomness in job arrival and worker sampling. When the workload gradually became more heterogeneous, the scheduler with traffic light (labelled as 'with TL' in the figured) consistently outperformed the baseline method. The figure shows that traffic light eliminated more than one third of delays caused by workload heterogeneity in this experiment. The improvement is especially noticeable at high heterogeneity levels. In particular, Sparkle with traffic light outperformed the baseline scheduler by more than 30% in the last test with a workload composed from 0.1s and 5s jobs.

Figure 5 shows the proportion of jobs with relative delay exceeding 0.1 (referred to as unqualified jobs) in the same experiment. As can be observed, enabling traffic light reduced the amount of unqualified jobs by around 15% under high workload heterogeneity scenarios. Through this experiment, we show that traffic light allowed the sample based scheduler to better handle heterogeneous workload, alleviating about one third of the performance loss caused by workload heterogeneity with only small computation cost and no extra communication cost.

4.2 Sparkle with Weak Workers

In this experiment, we tested Sparkle on clusters with different level of worker strength to observe how it could help reducing the loss in scheduling performance on clusters built from weak workers. In a series of tests, we gradually reduced the amount of resources (number of slots) available to each worker while maintaining the overall resource level in the cluster unchanged (through increasing the number of workers). During each test, we injected the workload with equal amounts of 0.1s and 1s jobs until the cluster stabilised at 60%, 70% and 80% overall load level, respectively. Figure 6 shows that as individual workers became weaker, the overall average job delay increased accordingly. However, the increase was at a slower pace when Sparkle was used. In particular, comparing to the baseline, Sparkle achieved 40% improvement in terms of relative delay on the cluster built from 2-slot workers, under 80% cluster load level.

Figure 7 shows that the baseline sample based scheduler produced much more unqualified jobs on clusters with weak workers. Specifically, in the cluster constructed from 2-slot-workers, the proportion of unqualified jobs reached 50% under 80% cluster load level. In comparison, up to a third of these unqualified jobs were eliminated when Sparkle was used.

5. Conclusions and Future Work

In this paper, we proposed the traffic light and worker blacklist mechanisms to extend the scope of sample based scheduling by reducing the number of sub-optimal scheduling decisions caused by workload heterogeneity and weak workers. Our methods only introduce few steps of computation and incur no extra communication cost. We imple-

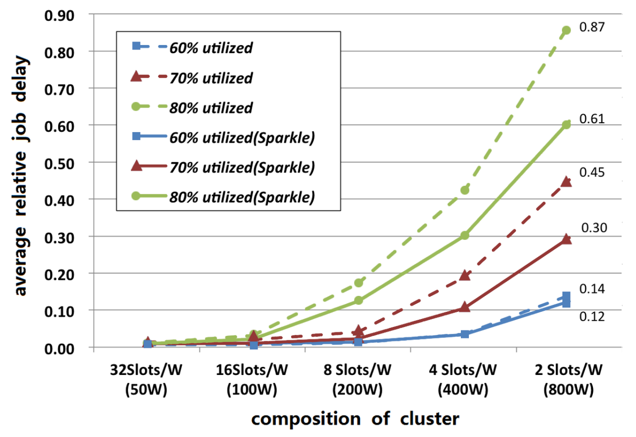


Figure 6. Average relative response delay on clusters with different worker strength levels when using Sparkle and the baseline scheduler.

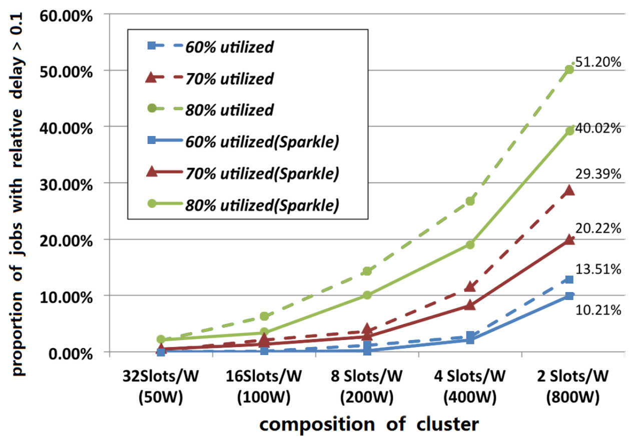


Figure 7. Proportion of unqualified jobs on clusters with different worker strength levels when using Sparkle and the baseline scheduler.

mented these refinements in the Sparkle scheduler, which can also be easily extended to incorporate delayed scheduling, batch scheduling, late binding and other existing methods due to its simplicity.

6. Acknowledgements

This work was financially supported by the Strategic Priority Research Program of the Chinese Academy of Science (No. XDA06010600), as part of the DataOS project. The work of Jie Shen has also been funded in part by the European Community Horizon 2020 [H2020/2014-2020] under grant agreement no. 645094 (SEWA).

References

[1] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. Vijaykumar. Tarazu: optimizing mapreduce on heterogeneous clusters. In *ACM SIGARCH Computer Architecture News*, volume 40, pages 61–74. ACM, 2012.

[2] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, volume 11, pages 22–22, 2011.

[3] M. Mitzenmacher. The power of two choices in randomized load balancing. *Parallel and Distributed Systems, IEEE Transactions on*, 12(10):1094–1104, 2001.

[4] A. C. Murthy, C. Douglas, M. Konar, O. O'Malley, S. Radia, S. Agarwal, and K. Vinod. Architecture of next generation apache hadoop mapreduce framework. *Apache Jira*, 2011.

[5] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica. Sparrow: Scalable scheduling for sub-second parallel jobs. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2013-29*, 2013.

[6] 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 *Proceedings of the Third ACM Symposium on Cloud Computing*, page 7. ACM, 2012.

[7] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Towards understanding heterogeneous clouds at scale: Google trace analysis. *Intel Science and Technology Center for Cloud Computing, Tech. Rep*, page 84, 2012.

[8] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: flexible, scalable schedulers for large compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 351–364. ACM, 2013.

[9] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer. Scheduling heterogeneous multi-cores through performance impact estimation (pie). *ACM SIGARCH Computer Architecture News*, 40(3):213–224, 2012.

[10] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, et al. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 5. ACM, 2013.

[11] A. Verma, L. Cherkasova, and R. H. Campbell. Slo-driven right-sizing and resource provisioning of mapreduce jobs. *Proc. LADIS*, 11, 2011.

[12] M. P. Wellman, W. E. Walsh, P. R. Wurman, and J. K. MacKie-Mason. Auction protocols for decentralized scheduling. *Games and economic behavior*, 35(1):271–303, 2001.

[13] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. In *OSDI*, volume 8, page 7, 2008.

[14] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European conference on Computer systems*, pages 265–278. ACM, 2010.

[15] Q. Zhang, M. F. Zhani, R. Boutaba, and J. L. Hellerstein. Harmony: Dynamic heterogeneity-aware resource provisioning in the cloud. In *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*, pages 510–519. IEEE, 2013.