Multi-Resource Scheduling with Consideration of Differential Services in Multi-Tenant Sharing

Sheng Zhang
Shanghai Jiao Tong University
800 Dongchuan Road
Shanghai, China
zhangsheng@sjtu.edu.cn

Jianguo Yao
Shanghai Jiao Tong University
800 Dongchuan Road
Shanghai, China
jianguo.yao@sjtu.edu.cn

ABSTRACT

Current schedulers in the cloud computing platform are facing the performance requirement. However, due to lack of the theoretical and practical guidance on multi-resource scheduling, algorithms in the schedulers are neglecting the difference between resource demand vectors and resource demand flow, resulting into the large resource waste caused by fragmentation, with all tasks struggling with the same bottleneck resource. We refer the resource demand flow as the relationship between resource demand and time to build a more accurate task model from the perspective of resource consumption. Also, we present the offline sat scheduler, the reduction algorithm based on the sat solver, to give advice on the optimisation on scheduling order of the tasks with various resource demand flow. To make up for the time limit of the sat solver, we provide the price scheduler to optimise the resource utilisation, while promising a certain degree of fairness. The simulation shows that the performance of the sat scheduler beats YARN default scheduler by cutting 30% of the make span, profiting from SAT powerful calculating technique, and the price scheduler gets a shorter make span and higher utilization.

Keywords

Multi-Resource; Scheduling; Multi-Tenant; Cloud

1. INTRODUCTION

In recent years, cloud computing becomes a very hot topic. Nowadays, many well-known sites such as Google, Ali, and Facebook have their own cloud computing clusters, to facilitate management services they provided, which can also be used as the external server rental platform [3]. Cloud computing is facing a high-performance, multi-dimensional high scalability and on-demand services. Tasks with various types of resources request and various service level requirements are needed to run on the cloud platform, which challenges the cloud computing framework, as well as the design for its core multi- resource scheduling algorithm. Cloud computing platforms scheduling algorithms have different optimisation goals, such as make span, efficiency and fairness. The default FIFO scheduling on Hadoop platform only targets at fairness, making it low resource utilisation, long make span. This paper argues that, in weighing the fairness and make span, the make span is a more important indicator [1].

FIFO and dominated resource fairness scheduler policies do not consider the resource utilisation, causing the resource fragmentation and unnecessary waste. Unfortunately, current schedulers don't take tasks resource demand flow into the account, too. The ignorance of the future resource demand of tasks leads to the bump of needs on one resource at a time, which in turns elongate the make span. The multi resource scheduling problem with specified resource demand flow is actually the flow shop problem, tasks requesting the different resources combination in the sequence of phases, which proves to be NPH [5]. To get the short make span, we try to use the SAT solver to suggest the scheduling order. The SAT solver is remarkable for its accuracy, but constrained by the running time. We propose a more feasible scheduler named price scheduler to avoid resource fragmentation by finding the smallest resource-wasted task and peek demand by choosing the quick finished tasks with resource needed.

The main contributions of this paper can be summarized in threefold.

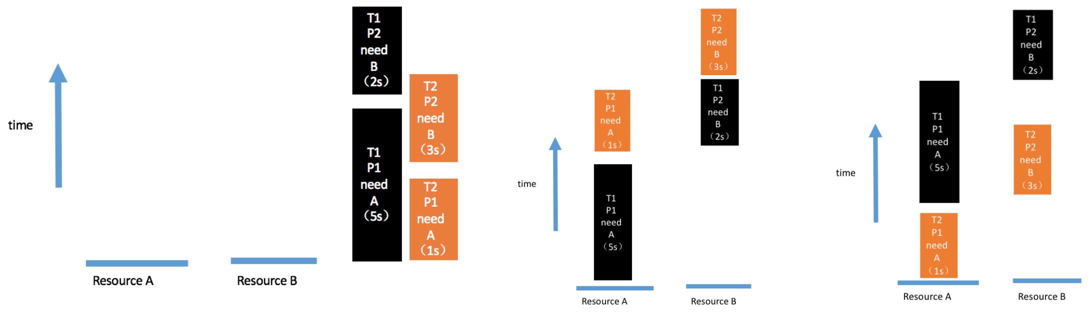
- First we elaborated the task model to formally build problems and analyse its computational complexity.
- Second, faced with different objective functions, we presented the offline SAT scheduler algorithm and the price scheduler algorithm.
- Third, we built experiment to evaluate its performance. The result shows our algorithm outperforms other default algorithm on YARN on the criteria of the make span and balance between the efficiency and utilisation.

The rest of this paper is organized as follows. Sec. 2 discusses the motivation of this paper. Sec. 3 presents the models and scheduler. Sec. 4 conducts the escalations for the proposed scheduling. We conclude this paper in Sec. 5.

© 2016 ACM. ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

2. MOTIVATIONS

2.1 Resource Inefficiency



(a) (b) (c)

Figure 1: A motivation example: (a) Two tasks and two different resources, each with two phases; (b) FIFO takes 10 seconds; (c) The reverse way takes 8 seconds.

The commonly-used schedulers in Hadoop are capacity and fair schedulers. Neither of them highlight the need to concern about the difference demand for the resource of the tasks and the impact brought by multi-resource environment. For example, there are three tasks, namely, tasks 1, tasks 2 and tasks 3, with resource demand $(3CPU, 6GMEM)$, $(8CPU, 5GMEM)$ and $(2CPU, 5GMEM)$, who are competing for the current scheduling chance on the $(10CPU, 10GMEM)$ machine. We note that the default FIFO scheduler will only be able to schedule task 1, since it does not focus on the resource consumption vector, implying the poor resource utilisation. Scheduling the task 2 and task 3 is a better choice in this case. We take the heuristic that picking tasks with bigger resource consumption greedily will lead to a possible satisfied utilisation. So, in the price scheduler, we choose the task that will consume left resources most. To be mentioned, it is also an instance of the multi-resource packing problem analysed in [2] which proposed the cosine function to choose the task with the maximum similarity. But Tetris loses its insight when one resource is so adequate that the weight of it is much bigger than other resources, leading to fragmentation and waste of the bottleneck resource which is insufficient. So, it enlightens us of the necessity to reduce resource waste, especially for those inadequate ones. Our insight is laid on the fact that the resource with high demand should not be wasted. In the price scheduler, we denote Price, the value to reflect the resource shortage degree, to help weigh the waste of different resources caused by scheduling the specific task.

2.2 Resource Demand

Tasks in the cloud platform need different resource in different phases. Map reduce can be viewed as the typical case where the map phase prefers IO resource and the reduce phase biases on the network resource. They have the strict executing order. The scheduling order has hugely impact on the make span. We take 1 as an example. There are two tasks with two phases. Phase 1 of task 1 requests for resource A for 5 unit time, and phase 2 of task 1 requests for resource B for 2 unit time. Phase 1 of task 2 requests for resource A for one unit time, and phase 2 of task 2 requests for resource B for 3 unit time. We observe that we can shorten make span by scheduling task 2 first. This problem belongs to the branch of the flow shop problem, which is best known for its NPH in the multi-resource situation. Few heuristic solution exists on some certain condition. We use SAT solver to solve this kind of problems to analysize how huge the influence is on the make span, and develop the tool called SAT scheduler to offer the sub-optimal scheduling suggestion target at minimising the make span. We observe that the optimisation may root in favouring on the small remaining time tasks and balancing the future workload of different resources. We take this heuristic in the implementation of the price scheduler. In our design, we combine resource demand with time to express the concept of workload. By choosing the phase characterised by maximising the rare resource utilisation and short remaining time, we achieve the utilisation and a good scheduling order.

3. MODEL AND SCHEDULER

In this section, we will describe the problem model, assuming that we know the complete information of the tasks resource demand and its demand flow. In practically, these variables can be approximated closely by a lot of heuristic methods.

3.1 Modeling

We construct the cloud platform with n machines. The wth machine can be expressed by a resource vector:

$$r_w =< k_1, k_2, \ldots, k_m > \quad w \leq n, \tag{1}$$

where m is the number of the resource categories. k_i is the total capacity of the resource i.Also, the total cloud resource information can be briefly denoted as r.

Define the task v_a can be formed as:

$$v_a =< p_a, s_a >, \tag{2}$$

where p_a is its priority. s_a is the collection of v_a phases. The set of all phases is briefly denoted as $\{phase\}$ or s. Each phase can be denoted by:

$$q_l^a =< d_1^{a,l}, d_2^{a,l}, d_m^{a,l}, t^{a,l}, a, l >, \tag{3}$$

where $d_i^{a,l} 1 \leq i \leq m$ is demand for resource i of q_l^a, and $t^{a,l}$ is time needed by the phase q_l^a.

Phases have strict execute order. We define the relation $Before$ on phases as $<$:

$$q_i^a < q_j^a \quad q_i^a, q_j^a \in s_a. \tag{4}$$

$Before$ meaning that q_i^a must be executed and done before q_j^a. In addition, we need to define the execution functions:

$$e : (r, q_l^a, \{phase\}) \mapsto (t, n) \tag{5}$$

where t is the start execution time of q_l^a, n is the id of the machine to run the phase on.

To judge whether a phase is running, we have:

$$isRunning(e, r, q_l^a, s, t) = \begin{cases} 1 & t' < t \ \wedge \ (t' + t^{a,l} > t) \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where $t' = (e(r, q_l^a, s))^{(1)}$. The tag (1) is the reference to the correspondent number of the content in the brace. In the above case, it means referring to the t in (t, n). To define whether the phase is done, we have:

$$isDone(e, r, q_l^a, s, t) = \begin{cases} 1 & t' + t^{a,l} \leq t \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

Finally, we come to the scheduling function:

$$h : (t, n) \mapsto (r, q_l^a, \{phase\}) \forall l, a, t \in N \tag{8}$$

$$s.t. \sum_a \sum_l isRunning(h, r, q_l^a, \{phase\}, t)\, d_i^{a,l} \leq k_i \tag{9}$$

$$h(r, q_i^a, \{phase\})^{(1)} + t^{a,l} < h(r, q_j^a, \{task\})^{(1)} \ \forall i, j : q_i^a \leq q_j^a \tag{10}$$

Equation 9 is the constraint of the capacity of all resource in the given time. Equation 10 is the constraint of the phase scheduling order. We note the problem is NPH, due to the following reason. First, the problem is the multi dimensional bin packing problem obviously, if we consider machines are bins and phases are objects. Second the problem is also the flow shop problem, if we consider all resource is unit resource, then the machine is like operating tasks in the certain order. Both of them is NPH.

3.2 The SAT Scheduler

Faced with two thorny problems, we want to take advantage of general NPC solution to reduce the make span. Because the sat solver is an advanced technique which is widely used, we choose it as our powerful tool [4]. However, a problem occurs when we want to allocate the resource to phases of tasks. The sat solver can only express the boolean value, which means the allocation should be based on the unit resource. If a phase of a task needs 10 cores, while 100 cores are available, it need to choose 10 out of 100, that means there are around C_{100}^{10} probabilities. To calculate it is not practical. We avoid it by averaging the workload among the all the same kind of resource. The allocation algorithm is simple but useful. Every time, we choose the machine with the least workload on it and distribute the demand among the unit resource which have less workload. After the resource distribution, we have to give constraints to cut down impossible answers. We denote Q_l^a as the ID of q_l^a, and R_l^a is the set of the resource ID the phase q_l^a demanding for.We use $A[r, t, Q_l^a]$ to indicate whether allocating resource r to phase q_l^a at time t.

Then we get the first constraint, Task Demand Flow Rule (pre-phases must be finished first):

$$A[r, t, q_l^a] \rightarrow \vee_{t'<t} \wedge_{max\, l' : q_{l'}^a < q_l^a} \wedge_{r' \in R_{l'}^a} A[r', t' - t^{a,l'}, Q_{l'}^a] \tag{11}$$

Equation 11 describes that if the succeed phase is scheduled, the prerequisites must be scheduled before.

Next, we get the Resource Not Allocated Twice Rule:

$$A[r, t, Q_l^a] \rightarrow (\wedge_{a' \neq a} \wedge_{l' \neq l} \not{A}[r, t, Q_{l'}^{a'}] \wedge (\wedge_{l' \neq l} \not{A}[r, t, Q_{l'}^a])) \tag{12}$$

Equation 12 describes that if one phase is scheduled, other phases demanding for the same unit resource must not be scheduled at that time.

Application All finished Rule:

$$\vee_{t \leq TIME} \wedge_a \wedge_{max\, l} \wedge_{r \in R_l^a} A[r, t, Q_l^a] \tag{13}$$

Task Not Interrupted Rule:

$$\neg A[r, t-1, Q_l^a] \wedge A[r, t, Q_l^a] \rightarrow \wedge_{1 \leq \delta t < t^{a,l}} A[r, t+\delta t, Q_l^a] \tag{14}$$

Equation 14 means once we allocate the resource to one phase, we will allocate that resource to it until its end. We use the binary search method to find the minimum $TIME$, satisfying all rules above, which should be the shortest make span to the problem. We notice that the reduction method takes $O((|phase||resource| * TIME)^2)$ time to generate all the clause.

3.3 Price Scheduler

We use the sat solver to analyse the influence of the resource demand flow on the make span and suggest the schedule order offline. However, it become extremely slow when the scale of the problems is large. So, we develop a kind of the heuristic method to make up for this shortcoming. We present the price scheduler, which is based on the idea that reducing resource waste and balancing the workload of different resources can help shorten the make span. Reducing resource waste can increase the resource utilisation, which will in turn shorten the make span. However, maximising utilisation currently is not the necessary and sufficient condition for yielding the shortest make span in the multi-resource environment, while tasks conform with problem model in 3.1. We need to balance the workload of resource demand now and future, too. We refer this kind of resources for which tasks have huge demand as rare resources. We prefer choosing tasks needing rare resources to balance the their workloads. We denote the value of scheduling a task as u_a, which may be the reflection of its priority or the its urgency to show the different service requirement of the tasks in the cloud. We want to distribute its value into different phases. Normally, each phase is given the same amount value denoted as $U_{a,l}$, if all phases are equally important. Then we need to define the price of the task at the certain time t:

$$Price(r, t) = \frac{\sum_{l \in \{S\} d_r^{a,l} > 0} U_{a,l}}{DISCOUNT(l) * k_r} \tag{15}$$

{S} is the set of phases needed to be scheduled. We derive the function to score the scheduling of a phase. First, we consider the gain of the scheduling a phase is its value $U_{a,l}$. Next, we want to maximise the utilisation. We take the heuristic to prefer the phase which could consume the maximum amount of resource. We have:

$$Waste(r, t, a, l) = \sum_{T \leq t < T + t^{a,l}} Price(r, t)(k_r - d_r^{a,l}) \tag{16}$$

T is the current time. Last, to prevent the job hunger, we give the compensations to those phases losing the scheduling chance:

$$Welfare(a,l) += welfareRate * \bar{U}/NUM\_OF\_TASKS \quad (17)$$

The $welfareRate$ is used to adjust the fairness level, if the $welfareRate$ is high, the scheduler will focus on the fairness factor. The $\bar{U}$ is the chosen phase value. So the score function will be:

$$score(r,t,a,l) = U_{a,l} - Waste(r,t,a,l) + Welfare(a,l) \quad (18)$$

It takes $O(NUM\_OF\_TASKS^2)$ to calculate the price. Then we use the $O(NUM\_OF\_TASK)$ to choose the maximum score phase. So it takes $O(NUM\_OF\_TASK^2)$ time to choose next scheduling phase.

4. EXPERIMENTS

The test focuses on the performance in different situations, as resources are relatively abundant or scare. We will use our algorithm again FIFO algorithm which is carried out by YARN itself. The result of the efficiency tests shows that our sat solver scheduler outperforms it by 20%, in the most situation, mainly due to the high computation ability of sat solver. The running time suggests the range of feasible problem size. Similarly, we deploy the price scheduler in this environment. The result shows that the sat scheduler beats FIFO scheduler by 30% make span.

4.1 Experimental Setup

We use a certain amount of random data to simulate the real world situation, to highlight the performance of algorithm on various possible condition. The kinds of data to simulate are: 1)the number of machines in the cloud platform, 2)the resource amount on the certain machine, 3)the number of tasks and their values symboling for their service levels, 4)the phase of tasks and the amount of resources and time required for each phase.

4.2 Comparison Baselines

We use the FIFO algorithm as the comparison baseline. The FIFO scheduler is a simple algorithm, which schedules the first coming task. If it can't be scheduled due to the lack of available resource, it will wait. We generate another scheduler, the capacity scheduler, which schedules the phase which has the smallest resource consumption. The priority scheduler which schedules the largest service level task first is also used in our comparison.

4.3 SAT Scheduler Results

We set the background environment as follows. There are two kinds of resource denoted as bandwidth and IO. Both of them are 100 units, distributed equally in 2 machines. The number of tasks is set to 10. The task randomly gets one to three phases. The maximum resource demand of each phases is limited to ten percents of the total available resource. The longest phase is four times longer than the shortest phase. Obviously the resource is abundant. The result showed by Fig. 2 is quite trivial. In this case, our sat solver is better than the FIFO algorithm by 20%, in term of the make span. We think the method benefits from two factors. First we distribute workloads equally into unit resources. Second, we make use of the powerful sat solver to generate the reasonable scheduling order that can maximise utilisation in the whole make span rather than currently.

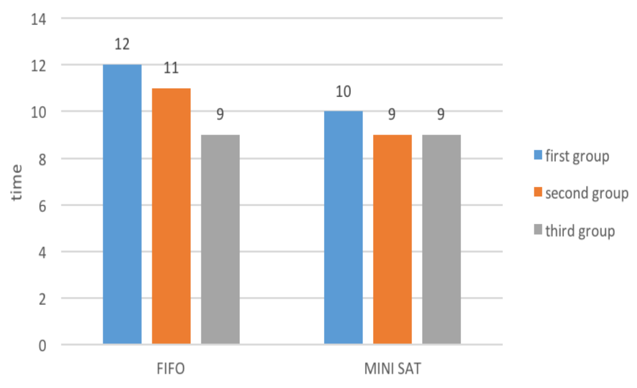


Figure 2: Makespan comparisons under sufficient resources.

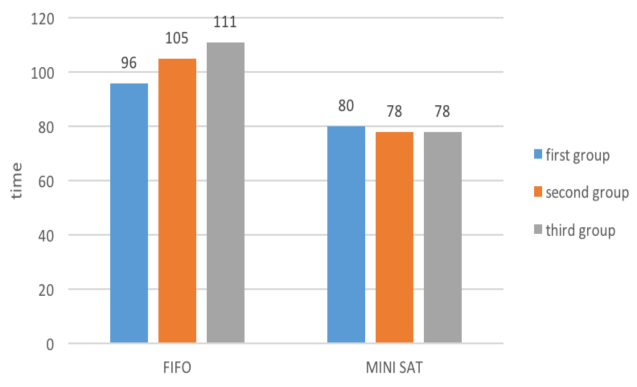


Figure 3: Makespan comparisons under limited resources.

In another background environment setting, the resource amount available in the cloud keeps the same. The task size becomes 30, and every of them gets 1 to 5 phases. The maximum resource demand of each phases is limited to thirty percents of the total available resource. The longest phase is nine times longer than the shortest phase. The scheduling becomes a little harsh for the sat solver, because the resource is not enough sometimes and the tasks differ from each other largely. In this case, our scheduler outperforms FIFO by 30% as shown in Fig. 3. It is the right choice of the sat solver to bias on the time or the resource demand in the different situation that harvests such good performance.

4.4 Discussions on Overhead of SAT Solver

The sat solver becomes helpless when the problem size is large. We need to test the sat solver overhead to indicate the right problem size. In our model, the problem size is direct ratio to $nm * \max k_m * NUM\_OF\_TASKS^2 * MAX\_TIME\_LIMIT\_OF\_TASK$. The relation is depicted in Fig. 4. We find that the time tends to be exponential when the input setting is large. In our case, the second setting costs us around 30 minutes to get the answer. The feasible problem size may be limited within several tasks, hundreds of unit resources and the executing time differs at most 5 times.

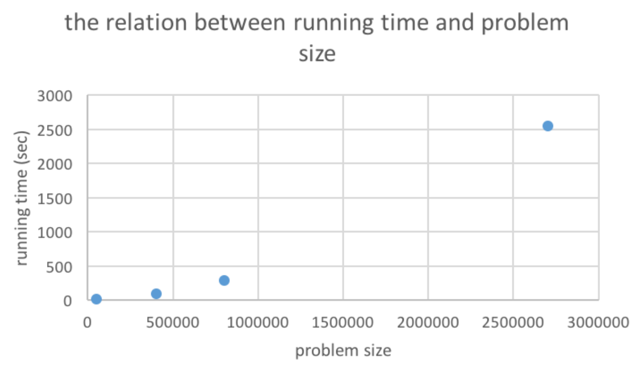


Figure 4: The running time with increasing problem size.

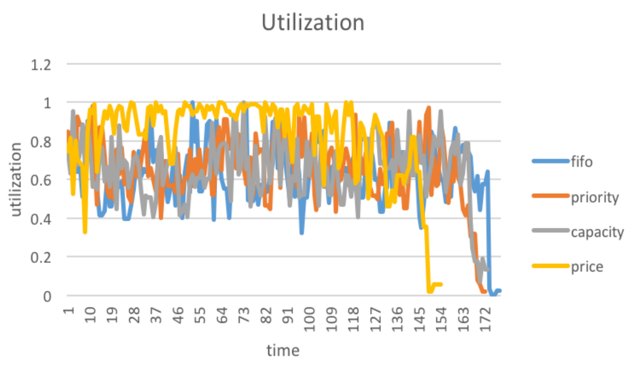


Figure 6: Price scheduler has the shortest makespan and maximum utilisations due to its least waste policy.

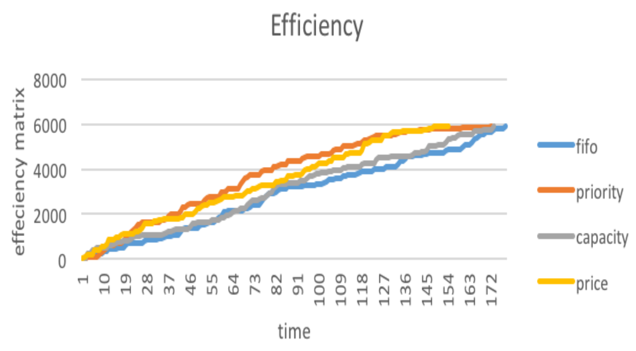


Figure 5: Compared with others, our scheduler achieves high throughput and has similar guarantee for service levels of different tasks.

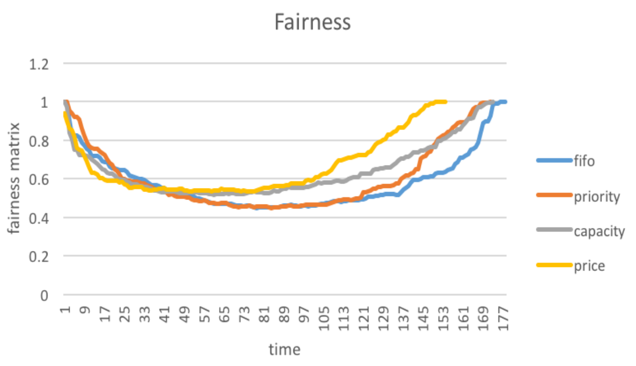


Figure 7: Price scheduler has the highest fairness due to the welfare factor to compensate for those losing scheduling chance tasks.

4.5 Price Scheduler Results

We set up the test environment similarly with the sat solver scheduler, but in a large scale. There are two kinds of resource denoted as bandwidth and IO. Both of them are 100 units, distributed equally in 2 machines. The number of tasks is set to 100. The task randomly gets one to five phases and their service levels is uniform distribution in one to five. The maximum resource demand of each phases is limited to ten percents of the total available resource. The longest phase is four times longer than the shortest phase.

The efficiency in Fig. 5 means total service levels finished at the time t. Utilisation metric is the ratio of used resource accounting for total resource. The fairness denotes the variance of the tasks progress. In our perspective, if the scheduling policy is fair, tasks have the same chance to be scheduled, which means their variance should be close to zero.

Fig. **??** and 10 show that the price scheduler outperforms other schedulers such as FIFO, priority and capacity in terms of efficiency, utilisation and fairness matrix. From efficiency matrix, the price scheduler is much better than the priority scheduling algorithm and other algorithms due to his preference on high service level tasks . For comparison with the capacity scheduler, the price scheduling algorithm outperforms by 12 percents through avoiding the resource fragmentation and balancing different resources workloads. From the equity point of view, the presence of welfare factors ensures that the price scheduler will not cause the task starvation and its choice based on the adaption to the left resource offers every task a chance to be scheduled. The highest point of the price scheduler is higher than other scheduling algorithms, which is also a proof to its fairness.

We test the price scheduler in a more harsh situation, and the results are shown in Fig. 8-10. The maximum resource the task requires consumes three-fifths of the total resource. From the experimental results, resource scheduling algorithm based on pricing , is far better than other scheduling algorithms. From an efficiency point of view, the price scheduler better than the priority scheduling method lies in the fact that priority-based scheduling algorithm will cause some fragmentation of resources, leading to the block of further scheduling, while the price scheduler reasonably trade-off each task service levels and waste of resources, to a greater extent, increase the efficiency of the unit resources . From the perspective of resource utilisation , resource scheduling algorithm based pricing has been ahead of other scheduling algorithms , thanks to its advanced scheduling strategies,

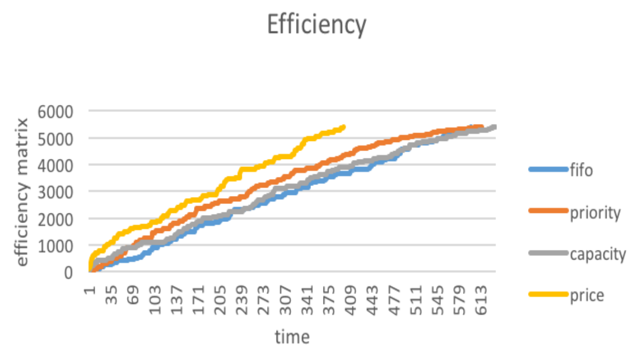


Figure 8: Our price scheduler outperforms others schedulers in terms of throughput and guarantee for service levels.

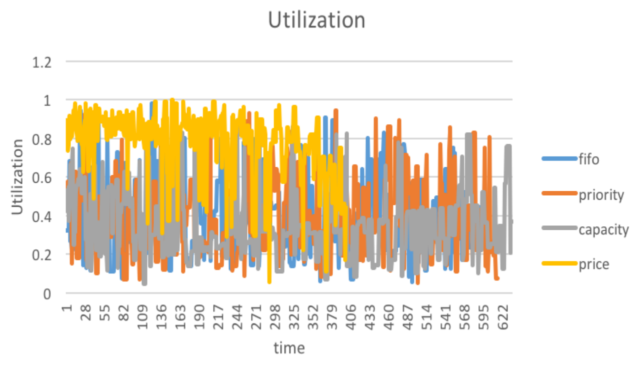


Figure 9: Price scheduler has the shortest makespan and maximum utilisations due to its least waste policy.

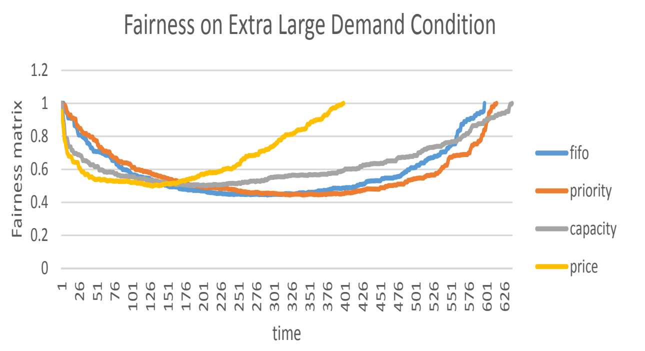


Figure 10: Price scheduler has the highest fairness due to the welfare factor to compensate for those losing scheduling chance tasks.

by selecting the largest remaining resources tasks , ensuring high availability. From the equity point of view, the presence of the welfare factor promises each tasks will be scheduled eventually.

5. CONCLUSION

This paper discusses the scheduling policy in multi-resource environment. We find that tasks in the cloud platform have the resource consumption pattern, which we denoted as resource demand flow of tasks. We observe there are two problems theoretically existing this model. One is the bin packing problem. The other is the flow shop problem. We take the heuristic solution to bin packing problem, which is to choose the maximum fit object. For the multi-resource condition, we use the price to function as the weight of different resource. Price is used to measure the level of rarity. We note that the resource with huge demand on should be more cherished than the sufficient resource. For the flow shop problem, our insights are laid on to balance workloads of different resource and preference on the short needing time task first. We combine these heuristic method and insights together to generate the price scheduler. It can avoid task starvation and reflect service levels. In addition, to get the possible optimal makespan, we take advantage of the sat solver to generate the sat scheduler to give the suggested scheduling order. The random simulation shows that our schedulers beats the default YARN scheduler by around 20%, in the term of make span, proving their superiority.

6. REFERENCES

[1] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: a flexible data processing tool. *Communications of the ACM*, 53(1):72–77, 2010.

[2] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. Multi-resource packing for cluster schedulers. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 455–466. ACM, 2014.

[3] Shen Li, Shaohan Hu, Shiguang Wang, Lu Su, Tarek Abdelzaher, Indranil Gupta, and Richard Pace. Woha: Deadline-aware map-reduce workflow scheduling framework over hadoop clusters. In *Proceedings of the 2014 IEEE 34th International Conference on Distributed Computing Systems*, ICDCS'14, pages 93–103, Washington, DC, USA, 2014. IEEE Computer Society.

[4] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving sat and sat modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll (t). *Journal of the ACM (JACM)*, 53(6):937–977, 2006.

[5] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 5:1–5:16, New York, NY, USA, 2013.