Accepted Manuscript.
Article accepted for publication in the International Journal of Grid and Utility Computing (IJGUC), 09/08/2019.

Improving the MXFT Scheduling Algorithm for a Cloud Computing Context

Paul Moggridge
Na Helian
Yi Sun
Mariana Lilley
Vito Veneziano
Martin Eaves

School of Computer Science,
University of Hertfordshire
Hatfield, Hertfordshire, UK
p.moggridge@herts.ac.uk

Abstract: In this paper the Max-min Fast Track (MXFT) scheduling algorithm is improved and compared against a selection of popular algorithms. The improved versions of MXFT are called Min-min Max-min Fast Track (MMMXFT) and Clustering Min-min Max-min Fast Track (CMMMXFT). The key difference is using Min-min for the fast track. Experimentation revealed that despite Min-min's characteristic of prioritising small tasks at the expense of overall makespan, the overall makespan was not adversely effected and the benefits of prioritising small tasks were identified in MMMXFT. Experiments were conducted using a simulator with the exception of one real world experiment. The real world experiment identified challenges faced by algorithms which rely on accurate execution time prediction.

Keywords: Cloud Computing, Scheduling Algorithms, Max-min

1 Introduction

Cloud computing is described as a disruptive technology (1) and is well established commercially with over 80% of companies utilising cloud based services. However, there are still challenges to be met (2). These include resource pooling/server consolidation, Service Level Argeements (SLAs), Quality of Service (QoS), energy management, traffic management, stability and fault tolerance (3)(4). These challenges are underpinned by the NP-Complete problem of scheduling. There are two important areas of scheduling, allocating virtual machines (VMs) to hosts (servers) and brokering cloudlets (tasks) onto VMs (5)(6). This paper is focused on the later.

One host can contain many VMs. When choosing the host for a VM, an allocator must ensure a host has adequate resources to support the VM. Resources considered can be one or any combination of cores, processing speed, RAM and bandwidth. Consolidating the VMs (packing them onto few hosts) means fewer hosts are required to be active.

Brokering cloudlets onto VMs is the process of deciding which VMs should run which user applications. One VM can process one cloudlet at a time. By effectively scheduling cloudlets onto VMs, the makespan (execution time) of metatasks (groups of tasks) can be reduced.

Scheduling in cloud computing presents an important opportunity for improving the efficiency and performance of data centers. Server utilisation in the cloud has been found to be about 30% to 40% (7) with some sources citing utilisation low as 6% (8). Through improved scheduling, tasks can be consolidated on fewer hosts, which increases utilisation and saves power. Fully relieving hosts is important, because servers consume 50% of their peak power at idle, which makes it detrimental to run servers with a light work load (9). Globally, datacenters consume approximately 1.5-2% of global electricity and this is predicted to grow at a rate of 12% annually (10). Based on that prediction from 2011, datacenters may now be consuming as much as 3%-3.9% of global electricity. In 2014 another estimate was made - datacenters in the United States consume aproximately 1.8% of US electricity (11). In particular, our paper looks at reducing the overall makespan of a collection of tasks on a fixed number of VMs (consumers) by improving Max-min Fast Track (MXFT) (12).

Copyright © 2018 Inderscience Enterprises Ltd.

Copyright © 2018 Inderscience Enterprises Ltd.

Accepted Manuscript.
Article accepted for publication in the International Journal of Grid and Utility Computing (IJGUC), 09/08/2019

This paper makes three main contributions by proposing

- MMMXFT algorithm which improves on MXFT with superior small task prioritisation.
- CMMMXFT algortihm which is a further improvement on the MMMXFT by automatically adapting to a wider range scenarios than MMMXFT.
- TAWRR algorithm an online alternative to CMMMXFT

As well as the proposed algorithms, this paper is an example of using the ScheduleSim (19) simulator and offers insight into the performance of commonly known algorithms.

This paper is structured as follows. Section 2 presents the related work. Section 3 discribes our proposed novel algorithms. Section 4 is the methodolgy and experimental results as well as discussion of our findings. Finally, section 5 draws conclusions from our findings and presents our reccomendation for future work.

2 Related Work

2.1 Commonly Known Algorithms

Scheduling is a complex NP-Complete problem and there are many papers proposing, surveying and comparing scheduling algorithms. In this paper we compared our proposed algorithms against commonly know algorithms with well understood performance.

Random (R)

This algorithm assigns tasks to consumers randomly, using Java's built in random function "java.util.Random".

Round Robin (RR)

This algorithm assigns tasks to consumer in repeating cycle. The first task is assigned to the first consumer, the second task is assigned the second consumers and so on. When the algorithm runs out of consumers to assign tasks to, it loops back around.

Weighted Round Robin (WRR)

Building on RR, WRR assigns weights to the consumers based on their speed - slower consumers are more likely to be skipped, instead of assigned being a task.

Task Alternating Weighted Round Robin (TAWRR)

Building on WRR, TAWRR (proposed in this paper) alternates between assigning big and small tasks. For example, if a consumer last had a big task, next cycle it will be assigned a small task.

Grouping Round Robin (GRR)

Building on plain RR, GRR breaks the consumers and tasks into groups before applying the round robin cycle to each group. GRR groups the slowest consumers with the smallest tasks, and the fastest consumers with the biggest tasks.

Shopping Execution Time (SET)

For each task this algorithm randomly selects several consumers to consider, it then assigns the task to fastest (lowest execution time) of the selected consumers.

Shopping Completion Time (SCT)

This algorithm is similar to SET. However, it considers completion time, which takes into account the tasks already assigned to the consumer. Completion time is the time it will take the given task to complete on the consumer in question. The more tasks assigned to the consumer the less likely it is to have the earliest completion time for the given task.

Minimum Completion Time (MCT)

This algorithm is also similar to SET however MCT considers the completion time of every consumer rather than just a randomly selected subset.

Max-min (MX)

Max-min assigns the biggest tasks first, placing them onto the fastest consumers. Max-min is a popularly researched scheduling algorithm. Numerous papers found that Max-min produces the best overall makespan, but has poor average task makespan due to placing numerous small tasks on slower VMs. Max-min is the opposite of Min-min which assigns the task earliest (Min) possible completion time to the fastest (Min) VM. The result of this is the biggest tasks get prioritised.

Min-min (MM)

The opposite of MX, MM assigns the smallest tasks first and onto the fastest consumers. This prioritises small tasks.

Max-min Based Algorithms

Underpinning Max-min Fast Track (MXFT) is MX. MX is the base for many scheduling algorithms. It is an advantageous choice because MX produces the shortest overall makespan.

Accepted Manuscript.
Article accepted for publication in the International Journal of Grid and Utility Computing (IJGUC), 09/08/2019

Max-min Improved (MX)

This algorithm improves and adapts MX for the cloud (13). By default, MX assigns the tasks with the latest (Max) possible completion time to the fastest (Min) VM. Max-min Improved assigns tasks with shortest execution time to the VM that can complete it earliest. This is the version we will refer to simply as Max-min or MX going forwards.

Selective Max-min Min-min

This algorithm proposed by Etminani and Naghibzadeh is for scheduling in Grid environment (14). The algorithm is based on the premise that in certain scenarios MM can outperform MX. It selects between MM and MX using standard deviation of the expected completion times of tasks on resources. The standard deviation is used to decide whether this is a small or big task relative to the list. This allows the algorithm to select the VM which MX would have chosen or the VM MM would have chosen. In the scenarios chosen, the algorithm always performed as well as either MX or MM.

Duplex/Greedy Max-min Min-min

This algorithm is similar to the Selective algorithm above. Duplex considers both MX and MM, then uses the better solution (15)(16). Duplex executes both MX and MM, then chooses the solution that achieves the smaller sum of the predicted run-time - minimised over all VMs (17). The difference to selective is that this algorithm chooses an algorithm for the whole metatask rather than an algorithm for each task.

RASA

RASA considers the distribution and scalability of VMs (18). Like Selective and Duplex mentioned above, RASA also combines MX and MM. RASA uses MM to execute small tasks before the large tasks and uses MX to support concurrency. The RASA algorithm alternates between MX and MM task by task. If task one was assigned by MX, task two would be assigned by MM. This provides a balance between the characteristics of MX and MM. By balancing the characteristics of MX and MM, RASA is more applicable for a cloud computing context as small tasks can be completed in reasonable amounts of time. This is a quality MXFT strives for and that we aim to improve in this paper.

Max-min Fast Track (MXFT)

This is the algorithm this paper aims to improve. MXFT uses two instances of MX, one designated as a fast track for small tasks using the fastest consumers expediting the processing of small tasks and another designated as a normal track processing the rest of the tasks (12).

2.2 Comparison of Algorithms

Scheduling algorithms also have many properties with some are particularly important in a cloud computing context. For instance, can the algorithm reschedule tasks? Does it need to know priorities such as speed of the consumer/servers? Does the algorithm need to know the size of tasks? Can the algorithm operate on a task by task basis (online) mode? Can the algorithm ensure small tasks are completed in a timely fashion? Can the algorithm prioritise VIP users? Is the algorithm energy consumption aware? Can the algorithm work to a target SLA and switch off unnecessary machines? These are just some of many properties that could be important to consider in a cloud context. This paper uses makespan for comparison across various algorithms.

3 Proposed Methods

3.1 Min-min Max-min Fast Track

In this paper we propose an improvement to MXFT, Min-min Max-min Fast Track (MMMXFT). The below broadly describes the steps in the algorithm. The steps are almost identical to MXFT with the only small but important change being that MMMXFT uses MM instead of MX for the the fast track. An implementation is available in ScheduleSim.

- For each consumer, find their delay times - as they may already have tasks waiting.
- Sort new tasks - biggest tasks first.
- Place 60% of the number of tasks in the normal track biggest tasks first.
- Place the remaining smaller task in the fast track.
- Add up the amount of work in the normal and fast track.
- Work out the ratio of work in the fast track to work in the normal track.
- Sort the consumers - fastest consumers first.
- Using the ratio, place consumers into the fast tack, fastest first (skipping every other).
- Place the remaining consumer into the normal track.
- Perform the normal Min-min algorithm to place the fast tracked tasks onto the fast track.
- Perform the normal Max-min algorithm to place the normal tracked tasks onto the normal track.

Accepted Manuscript.
Article accepted for publication in the International Journal of Grid and Utility Computing (IJGUC), 09/08/2019

Algorithm 1 shows the detailed algorithm. Table 1 defines the variables used in the algorithm. The value seen in algorithm 1 of 0.6 (60/40)% split of normal track to fast track can be adjusted.

Table 1 Algorithm notation used in algorithm descriptions 1.

R	Resources, (Consumers or VM).
R^p	Combined resources speed.
R_i^p	A resources speed (processing speed).
R_i^d	A resources delay, the time until it is free.
T	A metatask (tasks to schedule).
T^l	Number of tasks
T_i	A task.
T_i^s	A tasks size.
NT	Normal track tasks.
FT	Fast track tasks.
NT_j	A task in normal track.
FT_j	A task in fast track
NT^s	Normal track tasks combined size.
FT^s	Fast track tasks combined size.
FR	Fast track resources.
NR	Normal track resources.
FR_j	A fast track resource.
NR_j	A normal track resource.
E_{ij}	Execution time of task on a resource.
C_{ij}	Completion time of a task on a resource.

3.2 Clustering Min-min Max-min Fast Track

Another algorithm proposed in this paper is CMMMXFT which extends MMMXFT. CMMMXFT aims to improve MMMXFT by dynamically setting the size of the fast track using clustering. The advantage of this is that the size of the fast track can automatically suit the tasks to schedule. To achieve this, CMMMXFT uses K-Means clustering to find one cluster of small tasks and one cluster of large tasks. CMMMXFT also has the ability to decide not to use a fast track at all if the variance of the tasks is below a specified value. An implementation is available in ScheduleSim.

- Work out the range and standard deviation of the tasks.
- Decide whether to use a fast track - is the standard deviation as a percentage of the range smaller than the threshold if so, do not create a fast track.
- If not using a fast track, perform standard Max-min.
- If using a fast track perform the K-Means algorithm:
- Randomly position two points as centroid in range of the task sizes.
- Get the tasks nearest to each centroid.
- Calculate the mean of the task sizes nearest to each centroid.
- Those mean values become the new positions of centroids.

Algorithm 1 Min-min Max-min Fast Track

```
1: for all R do
2:     {Accumulate total speed for all resources.}
3:     R^p += R_i^p
4: end for
5: sort tasks T biggest execution time first
6: for all T do
7:     {Is index in first 60% of the number of tasks?}
8:     if i < T^l * 0.6 then
9:         append T_i to NT
10:        {Accumulate total size of normal track tasks.}
11:        NT^s += T_i^s
12:    else
13:        append T_i to FT
14:        {Accumulate total size of fast track tasks.}
15:        FT^s += T_i^s
16:    end if
17: end for
18: {Calculate ratio of speed to size.}
19: α = R^p/(NT^s + FT^s)
20: {Calculate the size the fast track should have.}
21: λ = R^p * FT_s
22: sort R fastest first
23: for all R do
24:    if FR^p < λ then
25:        append R_i to FR
26:        skip R_{i+1}
27:    else
28:        append R_i to NR
29:    end if
30: end for
31: for all FT_i do
32:    for all FR_j do
33:        {Find completion time.}
34:        C_ij = E_ij + R_j^d
35:    end for
36: end for
37: while FT not empty do
38:    find task T_i costs minimumexecutiontime
39:    assign T_i to FR_j which gives minimumcompletiontime
40:    remove T_i from T
41:    update R_j^d
42:    for all i do
43:        update C_ij
44:    end for
45: end while
46: for all FT_i do
47:    for all FR_j do
48:        {Find completion time.}
49:        C_ij = E_ij + R_j^d
50:    end for
51: end for
52: while NT not empty do
53:    find task T_i costs maximumexecutiontime
54:    assign T_i to NR_j which gives minimumcompletiontime
55:    remove T_i from T
56:    update R_j^d
57:    for all i do
58:        update C_ij
59:    end for
60: end while
```

Accepted Manuscript.
Article accepted for publication in the International Journal of Grid and Utility Computing (IJGUC), 09/08/2019

- Recalculate the nearest tasks - if tasks where re-assigned to new a centroid repeat previous two steps.
- See which centroid is associated with the smallest tasks by calculating the average task size for each cluster. The cluster with the smallest tasks is designated for fast tracking, the cluster with the larger tasks is designated for the normal track.
- Perform the normal Min-min algorithm to place the fast tracked tasks onto the fast track.
- Perform the normal Max-min algorithm to place the normal tracked tasks onto the normal track.

3.3 Task Alternating Weighted Round Robin

The third algorithm proposed in this papers is TAWRR. This algorithm adapts WRR by attempting to improve the distribution of tasks onto consumers. TAWRR algorithm does this by alternating between assigning big and small tasks to consumers, i.e. if a consumer last had a small task, next it will have a big task. By doing this, TAWRR inter-disperses small among big tasks, better spreading the tasks over the consumers and avoiding a single consumer receiving several large tasks by chance and becoming over utilised. An implementation is available in ScheduleSim.

- Sort the task smallest first.
- Divide the task into two equally sized groups.
- Assign a weighting to each consumer based on its speed relative to the fastest consumer.
- Cycle over consumers assigning tasks using the weighting and random number to occasionally skip slower consumers (Weighted Round Robin)
- If assigning a task to consumer, that last had a big task, this time assign a small task and vice versa.

4 Experiments and Results

4.1 Implementation of Experiments

Five experiments were conducted to look at the impact of different variables on the chosen algorithms. In these experiments, all 14 algorithms were compared including the three algorithms proposed in this paper and the aforementioned 11 commonly known scheduling algorithms.

Two further experiments were conducted to search for optimal values with the fast track algorithms: MXFT and MMMXFT. This experiment is not required with CMMMXFT since that version dynamically configures it's fast track size.

In addition, one more experiment was conducted to compare real world results against results from the simulator.

All experiments were repeated 20 times. The reason for this is that the poisson distribution of tasks produces slightly different sets of consumers and tasks on each run. The results presented are the average of the 20 runs to mitigate the effects of noise.

All experiments create tasks and consumers from normal distribution. The continuous Gaussian formula (below) was used to create a discrete (Poisson) distribution.

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

When:
μ (Mu) determines the position of the normal distribution.
σ (Sigma) determines width of the normal distribution.

Consumers and tasks were randomly created in the proportions of the Gaussian distribution. When adding a task or a consumer to a collection of tasks or consumers causes the target value to be exceeded, execution stops. This means there is always less total consumer speed or task size than the target value.

For all experiments except the "real world" experiment, ScheduleSim was used. ScheduleSim is an open source scheduling simulator. ScheduleSim is not designed to suit a specific domain, instead it uses generic terminology (producer, tasks, units, steps, schedulers and consumers) which prevents association of ScheduleSim to a specific context. ScheduleSim focuses on making it easy to simulate and prove scheduling algorithms in a simple environment.

ScheduleSim is implemented as a Java API. To use ScheduleSim, researchers can download the ".jar" and reference it in their Java projects. The ScheduleSim source code can also be downloaded using Git. The advantage of downloading code (as was done for this paper) is that it makes it easy to add new schedulers.

The simulator operates by using discrete time steps instead of an event driven design. This allows fine grain control and simplifies design. Furthermore, visualising these steps by producing PNGs and animated GIFs showing simulation at each step (20). Figure 1 is an annotated example output image from ScheduleSim.

As ScheduleSim is not domain specific, when using to simulate algorithms for cloud computing it must be noted that some aspects specific to cloud computing are not simulated.

Accepted Manuscript.
Article accepted for publication in the International Journal of Grid and Utility Computing (IJGUC), 09/08/2019

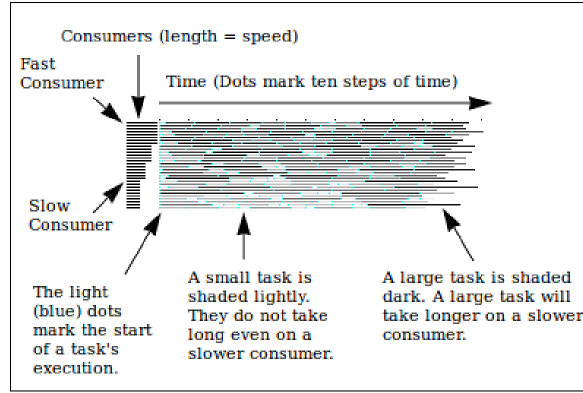

Figure 1 An ScheduleSim image with annotations explaining the meaning.

- *Networking* It should be noted that network latency is not modelled - meaning that a task can fully transverse the network from the producer through any number of schedulers to a consumer in a single step. Neither packet loss nor bandwidth are not modelled.
- *Virtualisation* ScheduleSim has consumers of fixed sizes and these cannot change during the simulation. In a real cloud environment resources can be resized (within limitations of the physical hosts).
- *Scheduler Processing Time* Outside of the scope of the simulator is modelling scheduler processing time. Schedulers are allowed to do as much processing as required without time advancing.
- *Application Complexity* ScheduleSim tasks have no interdependency and the size of tasks can be calculated ahead of execution. Furthermore, ScheduleSim tasks have defined starts and finishes. However, in a real world cloud environment, an end-user may use an IaaS instance indefinitely.
- *Service Levels* ScheduleSim schedulers are free to assign any consumer to a task. In cloud computing, often a user selects the size of the resource they require.

Due to these limitations, we recommended that after prototyping and proving an algorithm with ScheduleSim, further experimentation is completed with a simulator such as, CloudSim (6) and/or with experiments in the real world.

All experiments were measured using the following metrics:

Overall Makespan in verbose, the max/latest finishing-time of the simulation for given tasks.

$$m = Max\{T_j^f | \forall j \in T\}$$

Where, m is the makespan, T_j^f is finishing time of task j and T is all submitted tasks from all metatasks. Simply it is the time it has taken all the tasks to complete.

Task Makespan is the time it has taken the task to execute counting from when it was submitted to the scheduler. This metric highlights which size tasks are getting completed first either by being placed on fast resource or simply by being processed first.

4.2 Results and Discussion

Experiment A

The objective of experiment A is to test how each algorithm performs with different sets of consumers. Sets of consumers vary from many small consumers through to sets containing fewer but larger consumers. In a cloud context, we are testing how the algorithms would perform operating with lots of small VMs servers through to fewer but more capable VMs. Table 2 shows the parameters used in ScheduleSim. In the table the variable being adjusted is consumer μ. The consumer μ variable changes the position of the peak of the normal distribution. This changes speed of the consumers created.

Table 2 Experiment A Parameters

Variable	Value(s)
Consumer Min Size	2UPS
Consumer Max Size	32UPS
Consumer μ	5UPS, 15UPS and 25UPS
Consumer σ	6UPS
Consumer Target Speed	4000UPS
Task Min Size	20U
Task Max Size	240U
Task μ	110U
Task σ	44U
Task Target Size	200,000U

Figure 2 shows the performance (makespan) of the algorithms compared to each other with three different Mu (μ) values. Running the experiment with a μ of 5 results in most consumers having this value for their UPS (speed).

Almost all algorithms tested found scheduling more challenging with the lower μ value. As the total combined UPS of the consumers is a controlled variable and held

Accepted Manuscript.
Article accepted for publication in the International Journal of Grid and Utility Computing (IJGUC), 09/08/2019

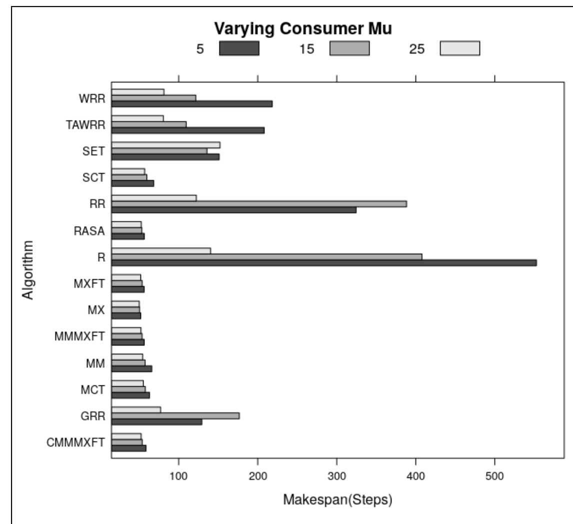

Figure 2 Experiment A results, μ (mu) is the most common speed of consumer.

consistent a lower μ value leads to more consumers being created. Having more consumers makes the problem space larger, leading to a higher makespan (worse performance) of the algorithms. Figure 2 does show some exceptions to this trend. SET was one such exception to this trend. The reason for this may be SET does not utilise a percentage of the slower consumers at μ 5 and over utilises the faster consumers at μ 25. In Figure 3 on the schedule presented on the left is the results from μ 5. Here SET is under utilising numerous consumers. In the middle of the figure is the results from μ 15, here SET is utilising more consumers. On the right is the results from μ 25, here SET is over utilising the fastest consumers. Another exception to the trend of worse performance with lower μ values (meaning, more but slower consumers) is RR and GRR. They are interesting because while they are effected, WRR and TAWRR are not effected despite of them having RR behaviour in common. In Figure 4, a possible cause can be seen. On the left of the figure is the results from μ 5, here the slowest resources are being often over utilised (there is several "spikes" where a slow consumer has been assigned a large task). On the right is the results from μ 15 there are fewer consumers but they are more powerful when compared to each task and thus less likely to be over utilised. On the right of Figure 4, there are only three over utilised servers. This problem is not seen in WRR and TAWRR since these algorithms weight slower consumers with a lower probability of receiving a task, thus partially mitigating this problem.

Focusing on the best performing algorithms, in Figure 5 the smaller performance differences in the best algorithms can be seen. MX produces the shortest makespan. In almost joint second across all μ values is MXFT, MMMXFT, CMMMXFT and RASA, while these algorithm are second by a clear margin when considering overall makespan. As these algorithms do not lead to excessive waiting time for small tasks, they may be advantageous when deployed in a cloud computing context. CMMMXFT marginally performed worse than MMMXFT in this experiment. However, the difference in performance is minimal.

Comparing CMMMXFT with RASA across the different μ values reveals that CMMMXFT is effected more by consumer μ. CMMMXFT responded worse to the lower μ value of 5 (more but smaller consumers) than RASA. However CMMMXFT performed slightly better with the higher μ value. It should be noted that the margin that CMMMXFT outperforms RASA by may not be statistically significant.

Experiment B

The objective of Experiment B is to investigate how the algorithms perform with different sets of tasks. This experiment tests the algorithms with lots of small tasks through to fewer but larger tasks. In Table 3 the task μ value is varied. Just like in Experiment A where the consumer size changes, this experiment changes the size of the tasks.

Table 3 Experiment B Parameters

Variable	Value(s)
Consumer Min Size	2UPS
Consumer Max Size	32UPS
Consumer μ	17UPS
Consumer σ	6UPS
Consumer Target Speed	4,000UPS
Task Min Size	20U
Task Max Size	240U
Task μ	20U, 120U and 220U
Task σ	44U
Task Target Size	200,000U

Different to Experiment A where the algorithms performed better with less complexity/fewer options to chose conversely, in this experiment the general trend is the lower the μ the better the algorithms performed, which is shown in Figure 6. This is maybe because with smaller pieces the algorithms can pack the tasks together tighter. A real world metaphor is that, with

Accepted Manuscript.
Article accepted for publication in the International Journal of Grid and Utility Computing (IJGUC), 09/08/2019

Figure 3 ScheduleSim visualisation, SET algorithm in Experiment A with μ 5, 15 and 25

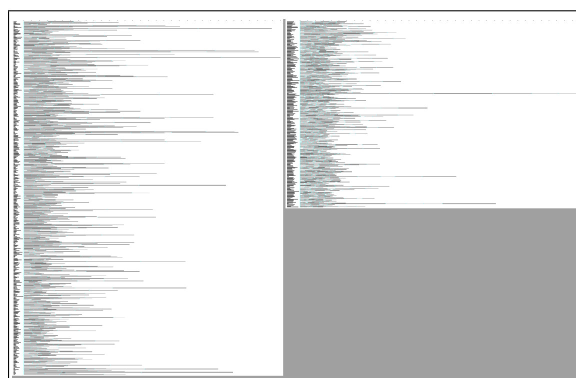

Figure 4 ScheduleSim visualisation, comparing the RR algorithm for Experiment A with μ 5 and 15. Note the difference in the number of over utilised consumers.

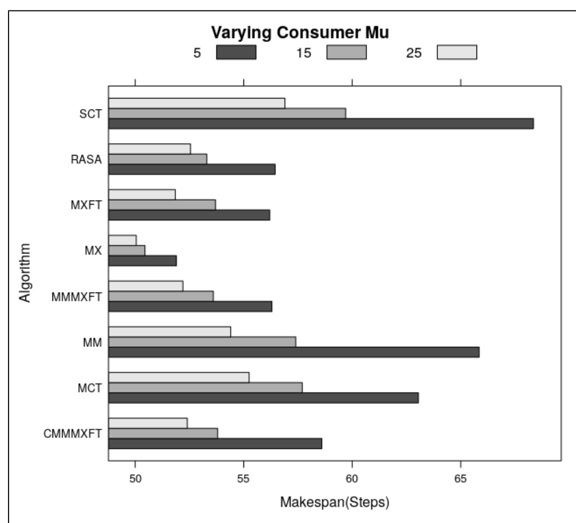

Figure 5 Experiment A results, showing only the best performing algorithms.

sand you can more completely fill a volume than with golf balls.

R and RR performed consistently badly for all task μ values, unlike in Experiment A where adjustment of the consumer μ made conditions favourable for them.

TAWRR's performance in this experiment is interesting as it outperforms WRR at μ 20 and is outperformed by WRR at μ 220. The difference

Accepted Manuscript.
Article accepted for publication in the International Journal of Grid and Utility Computing (IJGUC), 09/08/2019

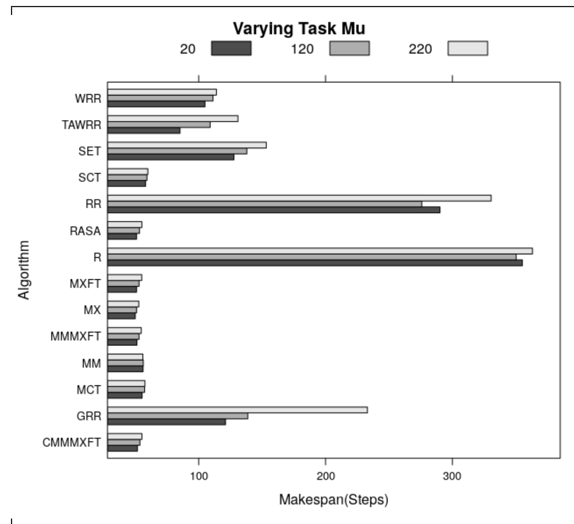

Figure 6 Experiment B results, μ (mu) is the most common size of task.

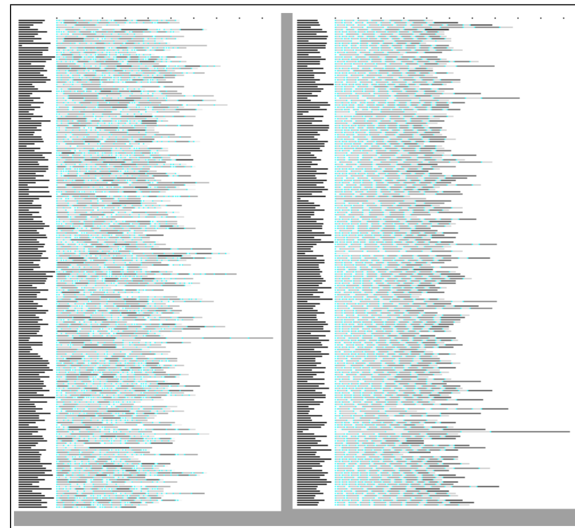

Figure 7 ScheduleSim visualisation, WRR and TAWRR algorithms in Experiment B with μ 20

between the WRR and TAWRR is a small. TAWRR (Task Alternating WRR) simply states that when the algorithm assigns a task to a consumer if a small task was assigned last, then the next task to be assigned is large, and vice-versa. In this experiment, this led to TAWRR spreading the tasks more evenly when compared to WRR seen in Figure 7 on the left with TAWRR on the right. It is possible to see TAWRR's alternation of small (lightly shaded) and big tasks (darkly shaded) in Figure 7. The particular schedule shown in Figure 7 WRR outperformed TAWRR. However, on average with μ 20 TAWRR outperformed WRR.

Zooming into the best performing algorithms, in Figure 8 the results look similar to Experiment A.

Interestingly, in this experiment, MMMXFT slightly outperforms MXFT and RASA with the higher μ values of 120 and 220. However, this is a small difference and may not be significant.

Uniquely to MM in Figure 8, MM appears to be unaffected by varying the task μ. MM is used in MMMXFT and this possibly explains why MMMXFT slightly outperforms MXFT with higher task μ values and then underperforms compared to MXFT with lower taks μ values.

Experiment C

The objective of this experiment is to access how the algorithms respond to different amounts of load. This experiment tests the algorithms with increasing amounts of tasks. In Table 4 you can see the increasing load the algorithms were trailed with.

In Figure 9 the results are very predictable. The more the total units the longer the overall makespan. A subtle feature of the results is the rate the performance gets worse as more tasks are added. As the load rises linearly over the three intervals, algorithms that are not effected by the scale of the load will exhibit a linear rise in makespan over the three experiments. However, not all

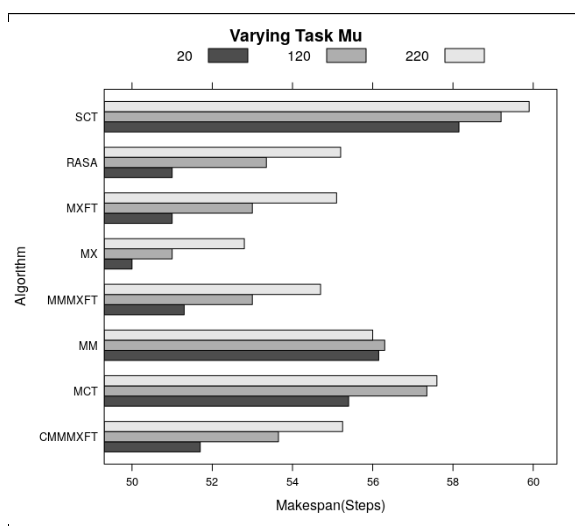

Figure 8 Experiment B results, showing only the best performing algorithms.

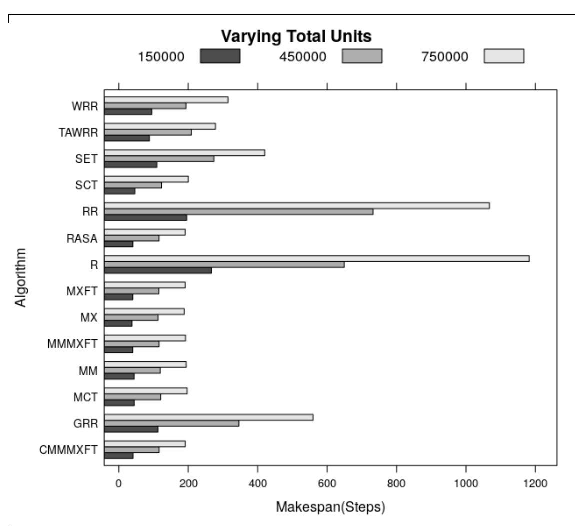

Figure 9 Experiment C results, this experiment change the total combined units of the tasks sent.

Table 4 Experiment C Parameters

Variable	Value(s)
Consumer Min Size	2UPS
Consumer Max Size	32UPS
Consumer μ	17UPS
Consumer σ	6UPS
Consumer Target Speed	4,000UPS
Task Min Size	20U
Task Max Size	240U
Task μ	110U
Task σ	44U
Task Target Size	150,000U, 450,000 and 750,000U

the algorithms exhibited this. With RR, TAWRR and GRR, the increase in makespan seems to be non-linear across this experiment. As only three values where tested the relationship is not fully clear. However, it is clear that the difference between 150,000U to 450,000U is greater than the distance between 450,000U to 750,000U for RR,TAWRR and GRR. Furthermore, R has the opposite relationship - adding more tasks makes the makespan worse. In Figure 10 you can see that a lot of the R makespan comes from a few over utilised consumers.

The results of Experiment C were further analysed in Figure 12 which shows the makespan of different groups of tasks based on size. The x axis shows the tasks grouped into 10 groups. This is based on their units (size). Group 0 represents the smallest tasks and Group 10 represents the largest tasks. The y axis shows the average makespan for each group. The objective of this graph is to assess the makepsan experienced by different sized tasks.

MX and MM take opposite paths in Figure 12. MX places the biggest tasks on the fastest consumers. In the Figure 12 you can see the result of this. Small tasks are being processed slowly and thus suffering a high makepsan, while the big tasks are being processed quickly and thus have low makespans. In contrast, MM places the smallest tasks on the fastest consumers. In Figure 12 you can see the result of this. Small tasks are being processed almost instantly and large tasks are being processed very slowly.

Accepted Manuscript.
Article accepted for publication in the International Journal of Grid and Utility Computing (IJGUC), 09/08/2019

Figure 10 ScheduleSim visualisation, R algorithm in Experiment C with total units 450,000

Figure 12 highlights the important improvement made to MXFT. MXFT proposed previously to this paper (12) uses two instances of MX. One for the fast track and one for the normal track. The consequence of this can be seen in Figure 12. The smallest tasks (Group 0) on the left are not be prioritized. MXFT requires a margin value to be specified, in this experiment that value was 0.4, meaning 40% of the number of tasks will be sent to the fast track. The fast track instance of MX running on the smallest 40% of the tasks can be seen in Figure 12, in the stepper downwards slope acting on the first 4 groups before MXFT spikes upwards for the second instance of MX acting on the normal track.

The proposed improvement to MXFT called Min-min Max-min Fast Track (MMMXFT) uses an instance of MM instead of MX for the fast track. As MM prioritises small tasks, this gives MMMXFT a useful advantage in the cloud for meeting SLAs of small tasks. In Figure 12 the improved profile of MMMXFT can be seen as small task are prioritised fully. The RASA algorithm has very similar profile to MMMXFT. Both the proposed algorithm (MMMXFT) and RASA prioritise both small and large tasks. Prioritising small tasks to prevent them from having to wait excessively for large tasks to finish and prioritising the largest tasks to achieve a good overall makespan.

TAWRR has a very interesting profile. This can be explained by how it works. TAWRR sorts tasks from smallest to biggest. The first half of the tasks are designated as mall tasks and the second half are designated as large tasks. TAWRR then performs WRR with the addition that if the algorithm previously assigned a large task to a consumer then it will assign a small task. Because the tasks are sorted by size (smallest first), the smallest of the small tasks and the smallest of the large tasks are assigned first causing them to finish earliest. Figure 12 shows the profile this produces.

GRR also has a very a interesting profile. The peaks and troughs seen in Figure 12 are caused by it's grouping behaviour. The grouping sorts the tasks smallest first and sorts the consumers slowest first. Then, in that order the tasks and consumers are broken into equal sized groups, both the consumers and tasks are broken into the same number of groups. For instance, if there was 50 tasks and 20 consumers and the number of groups was set to 5, the grouping process would create 5 groups of tasks each with 10 consumers, and 5 groups of consumers each with 4 consumers. The troughs are where the tasks at the start of a group are finishing first. Each peak is at the end of a group. Although in Figure 12 GRR appears to favour small tasks, this may not always the case. Considering how GRR works, it does nothing to ensure that the task groups are matched to the consumer groups other than the sorting. The experiment setup from which these results are taken have greater variation in size of the tasks than the consumers which could lead to groups not optimally matching. The ScheduleSim visualisation in Figure 11 shows such a mismatch. Notice that despite the smallest group of tasks being assigned to slowest group of consumers the slowest consumers were still too slow and as result they massively extend what could have been a very effective schedule.

Another insight that can be drawn from Figure 12 is that MCT, SCT, SET, WRR, RR and R all treat tasks fairly, not favouring any particular size of task. This was expected as their implementations do not sort on or consider task size. MCT and SCT simply consider completion time of each task on consumers. While SET and WRR consider the UPS (speed) of the consumers, R and RR consider nothing at all.

Again in Figure 13, there is similar ranking of the best algorithms to the previous experiments. The ranking of the algorithms and difference between their performance at different load levels suggests that the algorithms seen in Figure 13 do not degrade under high load. This experiment did not test very low load levels (where there are fewer tasks than consumers), so the inverse should not be assumed.

Experiment D

Experiment D assesses how the algorithms handle different distributions of consumers, from all consumers being quite uniform through to the consumers varying greatly in size. To achieve this, the σ value of the normal distribution was adjusted. In table 5, the experiment can be seen with the varying σ value.

Figure 14 shows that a lower consumer σ (sigma), (lower variation in the size of consumers) leads to a lower makespan (better performance). There are several exceptions to this, with one example being SET. With SET, the trend is in fact reversed. SET performed best with high consumer σ (high variation in the size of

Accepted Manuscript.
Article accepted for publication in the International Journal of Grid and Utility Computing (IJGUC), 09/08/2019

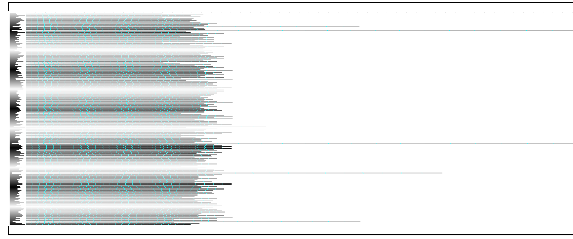

Figure 11 ScheduleSim visualisation, GRR algorithm in Experiment C with at the highest load 750000

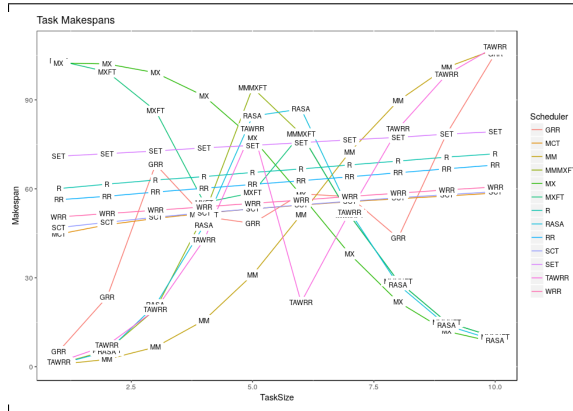

Figure 12 Experiment C results, showing the finishing time of tasks of different sizes. CMMMXFT is missing but would perform very similar to MMMXFT.

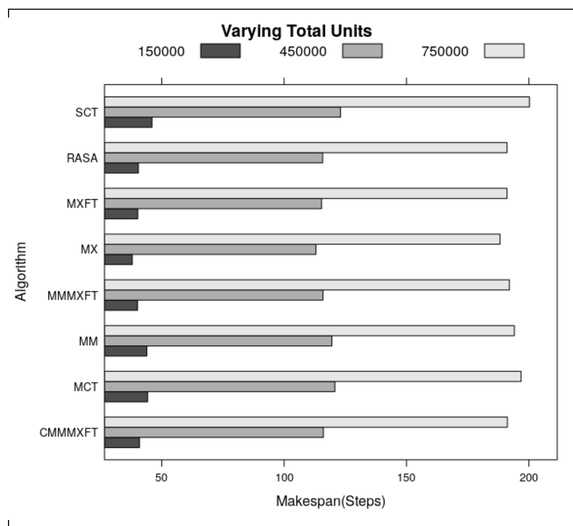

Figure 13 Experiment C results, showing only the best performing algorithms.

Table 5 Experiment D Parameters

Variable	Value(s)
Consumer Min Size	2UPS
Consumer Max Size	32UPS
Consumer μ	17UPS
Consumer σ	4UPS, 10UPS and 16UPS
Consumer Target Speed	4,000UPS
Task Min Size	20U
Task Max Size	240U
Task μ	110U
Task σ	44U
Task Target Size	200,000U

consumers). A possible explanation for this is that the lower amount of choice (due to less variation in size of consumers) negated the algorithm's logic of selecting faster consumers. In Figure 14 with σ at 4 (low variation) the performance of SET is very poor and similar to R. However, in Figure 15 with high consumer σ, SET avoids choosing the slowest consumers, leading to much better performance.

Another exception to the trend is the GRR algorithm. GRR as implemented in ScheduleSim groups the sorted consumers and tasks and then pairs the groups together. This does not take into consideration the UPS (speed) of the groups of consumers or the total units of the groups tasks. In Figure 16 in the left of the figure is σ 4 where GRR is working well with the large tasks (dark) being placed on the fastest consumers (longest bars on the left). However, on the right of the figure

Accepted Manuscript.
Article accepted for publication in the International Journal of Grid and Utility Computing (IJGUC), 09/08/2019

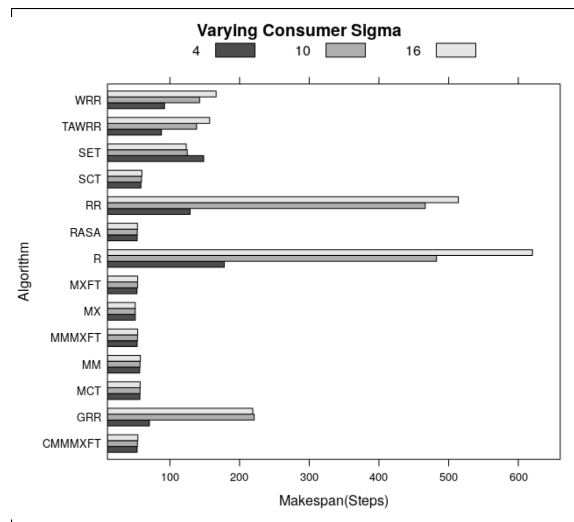

Figure 14 Experiment D results, higher σ (sigma) means more variation in the speed of the consumers.

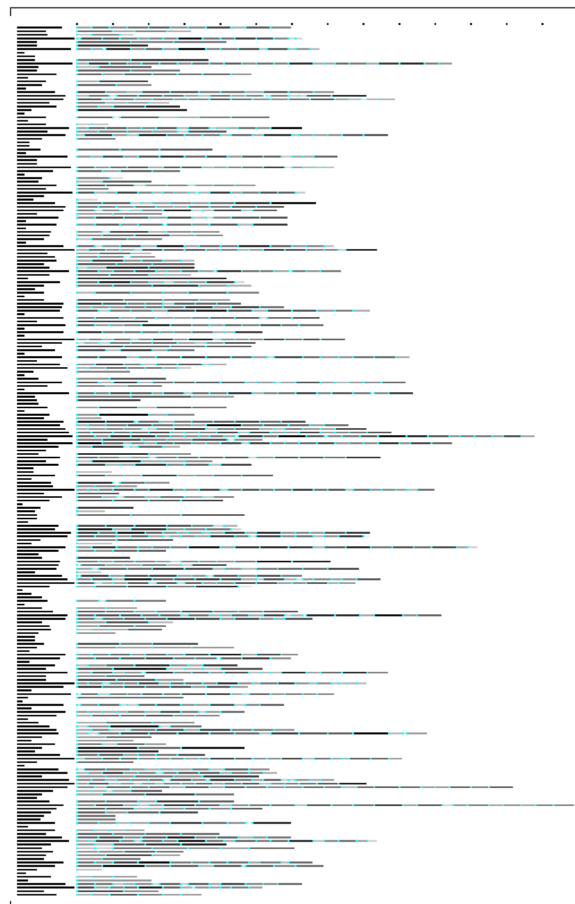

Figure 15 ScheduleSim visualisation, SET algorithm in Experiment D with σ 16

with σ 10 GRR is performing poorly by over utilising the slowest consumers. This over utilising can be explained, the greater variation in the size of the consumers causes a group of very small consumers to form, which is easily over utilised. This echoes Figure 11 from Experiment C.

Figure 17 shows MX performing very well compared to the other algorithms. All algorithms were not heavily impacted by consumer σ (compared with difference seen in Figures 8 and 5). A slight trend favouring not much variation of the consumers is perceptible, not significant. In this experiment, RASA, MXFT, MMMXFT and CMMMXFT all performed comparably.

Experiment E

Experiment E assesses how the algorithms handle different distributions of tasks, from all tasks being quiet

Accepted Manuscript.
Article accepted for publication in the International Journal of Grid and Utility Computing (IJGUC), 09/08/2019

Figure 16 ScheduleSim visualisation, GRR algorithm in Experiment D with σ 4 and 10

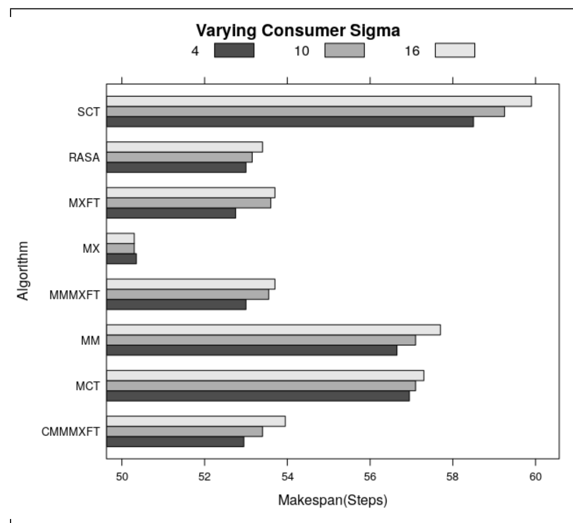

Figure 17 Experiment D results, showing only the best performing algorithms.

uniform through to tasks varying greatly in size. Table 5 shows experiment setup with the the varying task σ value.

Table 6 Experiment E Parameters

Variable	Value(s)
Consumer Min Size	2UPS
Consumer Max Size	32UPS
Consumer μ	17UPS
Consumer σ	6UPS
Consumer Target Speed	4,000UPS
Task Min Size	20U
Task Max Size	240U
Task μ	110U
Task σ	40U, 140U and 240U
Task Target Size	200,000U

The Experiment E results show how the variation of the size of tasks impacts the algorithms. The most interesting result in Figure 18 concerns the RR based algorithms. RR suffers a severe loss of performance when there is a large variety of tasks. However, WRR and TAWRR using their consumer weighting mechanism largely counter the loss of performance seen with RR. GRR improves further on the performance of WRR and TAWRR. GRR outperforms WRR and TAWRR with the high σ values, this may be because GRR always places big tasks on the fastest consumers. Whereas WRR and TAWRR leave this to probability, and thus occasionally a large tasks is placed on a slow consumer and this extends the overall makespan.

In Figure 19, again the rank of algorithms is fairly consistent with the other experiments. MX producing the best overall makespan, RASA, MXFT, MMMXFT and CMMMXFT all performing comparably.

Experiment F

The purpose of Experiment F is to explore how the size of the fast track (controlled by the margin parameter) effects overall makespan and task makespan. The margins trailed where 0.2 to 0.7, in increments of 0.05.

Table 7 Experiment F Parameters

Variable	Value(s)
Consumer Min Size	2UPS
Consumer Max Size	32UPS
Consumer μ	17UPS
Consumer σ	6UPS
Consumer Target Speed	8,000UPS
Task Min Size	20U
Task Max Size	240U
Task μ	110U
Task σ	44U
Task Target Size	800,000U

Accepted Manuscript.
Article accepted for publication in the International Journal of Grid and Utility Computing (IJGUC), 09/08/2019

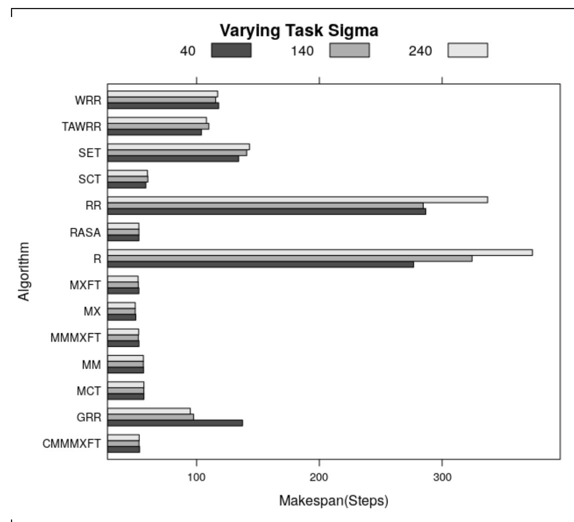

Figure 18 Experiment E results, higher σ (sigma) means more variation in the size of the task.

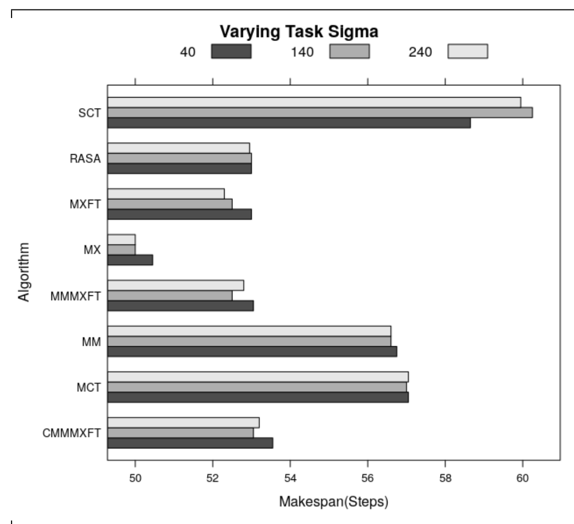

Figure 19 Experiment E results, showing only the best performing algorithms.

Experiment F varies the margin value supplied to MXFT. Figure 20 shows the trend of the higher margin values (size of fast track) the higher (worse) the overall makepsan. A possible hypothesis is that performance is lost due to breaking the problem into two. Each instance of MX is not aware of the other, and cannot utilise each others spare capacity. Furthermore, the MX instance processing the normal track has less small tasks to fill space in schedule with. It would be fair to assume that after 0.5 the trend would be mirrored (i.e. would take a downward turn). However, Figure 20 shows that this is not true, as 0.55 onwards continues the upward trend. A possible explanation is that the tasks assigned to the fast track were previously being completed in time that would have otherwise have been left empty around the normal track's larger tasks. To make an analogy to real life imagine filling two jars with golf balls and sand. Place a mixture of golf balls and sand in one jar and just sand in the other. Taking sand from the mixed jar placing it in the sand jar may cause an overflow. Whilst not reducing the level of mixed jar, since the sand was previously fitting into space in between the golf balls.

MXFT's weakness is that it does not prioritise small tasks as much as is possible. As it runs two instances of MX, inside each instance of MX, small tasks are not prioritised and get executed last. In Figure 21 this is clear, regardless of the margin. The group 1 tasks (the smallest tasks) are always executed latest out of the first four groups. In Figure 21, we can see that 0.25 - 0.35 are around optimal for the tasks in the experiment for trying to optimise makepsan of the smallest Groups (1 to 4 out of 10).

Experiment G

This has the same purpose as Experiment F. However, experiment G investigates the impact of the size of the fast track with MMMXFT rather than MXFT. Again the margins trailed were 0.2 to 0.7, in steps of 0.05. The experiment parameters are the same as in Experiment

Accepted Manuscript.
Article accepted for publication in the International Journal of Grid and Utility Computing (IJGUC), 09/08/2019

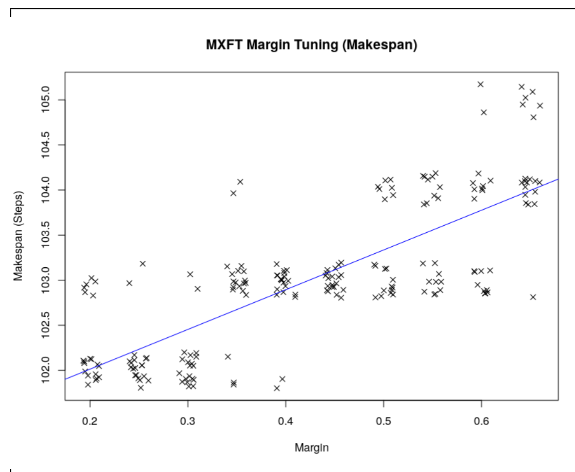

Figure 20 Experiment F results, varying the MXFT margin value, a higher margin value places more tasks in the fast track.

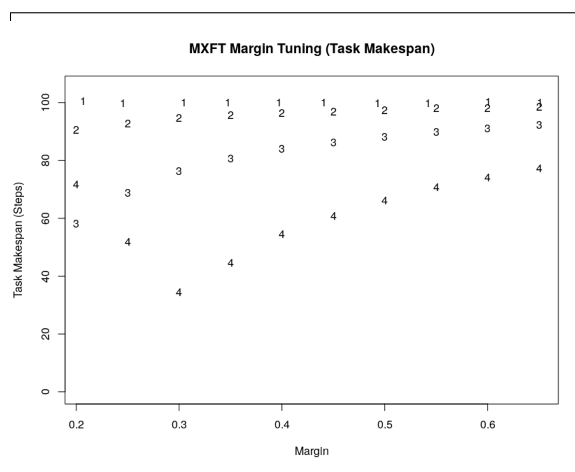

Figure 21 Experiment F results, task makespan results showing the impact of varying the MXFT margin on task size groups 1 to 4.

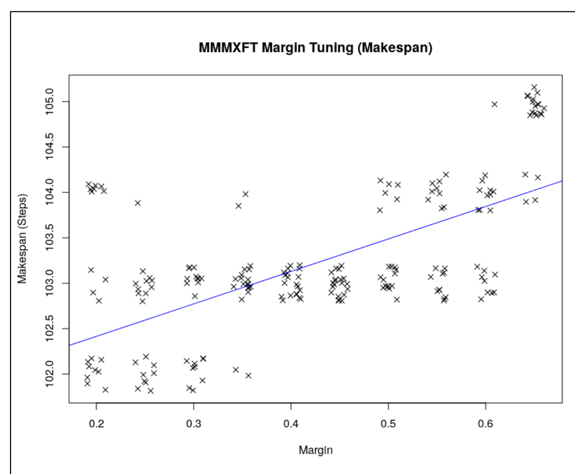

Figure 22 Experiment G results, varying the MMMXFT margin value, a higher margin value places more tasks in the fast track.

F, see Table 7.

Again, the same trend between the size of the margin and the overall makespan can be observed.

MMMXFT is simpler to use than MXFT. The bigger the fast track, the more small tasks are prioritised. The smallest tasks (Group 1 tasks) are always prioritised first. As the fast track gets larger, the fast track prioritises bigger groups more and more. Figure 23

Accepted Manuscript.
Article accepted for publication in the International Journal of Grid and Utility Computing (IJGUC), 09/08/2019

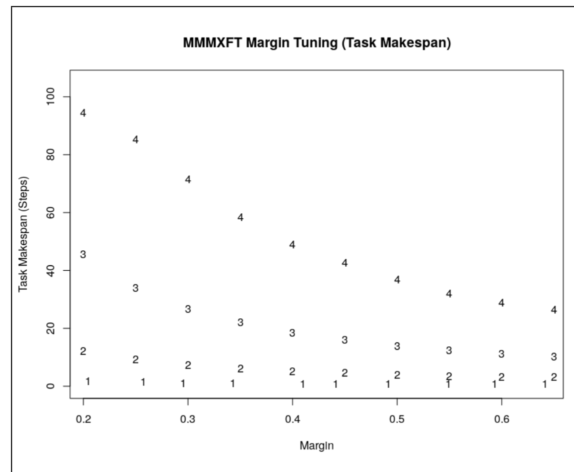

Figure 23 Experiment G results, task makespan results showing the impact of varying the MMMXFT margin on task size groups 1 to 4.

shows the result of this. In the figure, the relationship is not linear between the amount of prioritisation (i.e. lower task makespan) and increasing the size of the fast track (i.e margin). Thus there is a trade off between the prioritisation of small tasks against the importance of creating a schedule with good overall performance that must be balanced. In summary, the bigger the margin the linearly worse the makespan however, the more small tasks are prioritised.

Experiment H

The purpose of this experiment is to ascertain if the ScheduleSim findings can be reproduced in the real world. For this only R, RR, MX and MMMXFT were trailed since the corresponding code had to be produced for the real world experiment. Rather than using the normal distribution to generate the consumers and tasks in this experiment they were supplied in list, an overview is in Table 9.

Table 8 Experiment H (ScheduleSim) Parameters

Item	Count	Value(s)
Consumers	4	24UPS
Consumers	4	25UPS
Consumers	4	26UPS
Tasks	100	234U
Tasks	100	255U
Tasks	100	264U

Table 9 Experiment H (Real World) Parameters

Item	Count	Description
Server	4	Azure BS1 (single core)
Server	4	Azure BS2 (dual core)
Server	4	Azure B4MS (quad core)
TSP Problem	100	9 cities
TSP Problem	100	10 cities
TSP Problem	100	11 cities

Figure 24 shows performance of the real world server used in the comparison experiment. In particular, Figure 24 shows the performance of the fastest servers used the experiment. These are Microsoft Azure B4MS cloud IaaS instances, each with 4 (Intel(R) Xeon(R) CPU E5-2673 v4 @ 2.30GHz) cores. The workload for the comparison experiment was brute force solving instances of the travelling salesman problem. Three different task sizes were tested with, 9 city problems, 10 city problems and 11 city problems.

The real world server differed from ScheduleSim consumers in two way. Firstly, the real servers perform inconsistently, and this is not currently modelled in ScheduleSim. In Figure 24 you can see the range of results collected. The smallest tasks (9 city problems with 9! iterations to try) vary the most. Because of their short time span, they can be greatly effected by the server being busy with other background tasks. Secondly, in the real world, larger tasks benefited from an economy of scale that was not taken into consideration in ScheduleSim. In Figure 24 the slight downward trend can be seen, favouring the large 11 city problems.

Another challenge for ScheduleSim with modelling the real world experiment was the lack of variation in the performance of different servers. The real world fastest servers performed only 12% faster than the slowest overall server. The workload (brute force solving travelling salesman problems) was multithreaded and a bigger benefit was expected. This lack of variation is similar to Experiment D, indicated in Figure 14 where the σ value was 4. As ScheduleSim models the speed of consumers with low integer values, the real world IPS was adjusted to suit ScheduleSim, 2359468 (real world) Iteration Per Second (IPS) became 24 (ScheduleSim) Units Per Step (UPS), 2549294 (real world) IPS became 25 (ScheduleSim) UPS and 2635902 (real world) IPS became 26 (ScheduleSim) UPS.

Figure 25 shows the same trends seen with the fastest servers. Massive variation in the small 9 city problems and a slight economy of scale with the larger problems.

Accepted Manuscript.
Article accepted for publication in the International Journal of Grid and Utility Computing (IJGUC), 09/08/2019

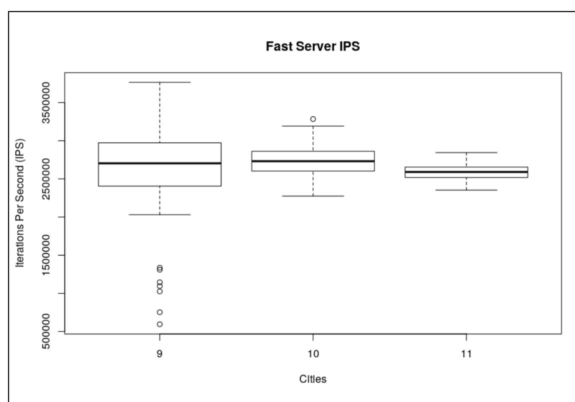

Figure 24 Experiment H fast server testing.

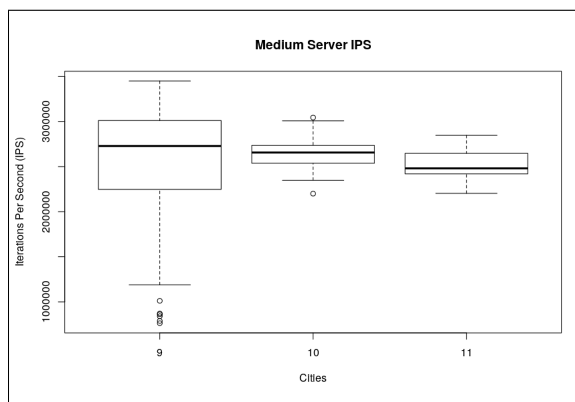

Figure 25 Experiment H medium server testing.

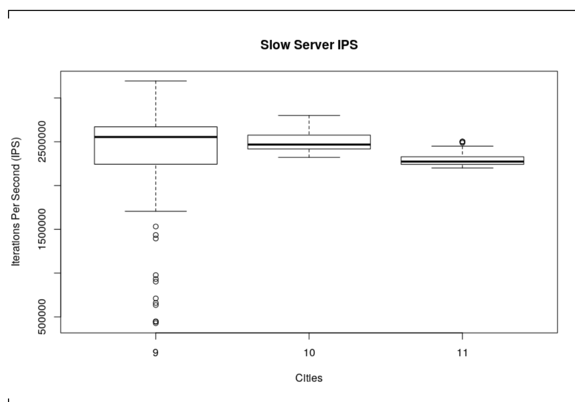

Figure 26 experiment H slow server testing

Moreover Figure 26 shows the trends again.

Figure 27 shows the results of the experiment conducted outside of ScheduleSim. Notice that rather than the "steps" ScheduleSim uses, in the real world experiments the makespan is measured in seconds. Of the 4 algorithms trailed for this experiment, MX produced the shortest makespan. This was expected as MX is very rarely surpassed by other algorithms in terms of makespan. Interestingly and unexpectedly RR came second. A possible explanation for this is that the very little variation in the speed of consumers meant that RR's choices were fairly optimal. Furthermore, the tasks were sent to the scheduler pre-sorted smallest first. Had these conditions not arisen, RR may have performed much worse. In joint third was R and MMMXFT, R often performs poorly which was expected. However, MMMXFT underperformed in this experiment.

Both R and RR have $O(n)$ complexity. In Figure 28, the execution time of the scheduling algorithm can be seen. This is the amount of time that it took the scheduling algorithm to decide where to assign the tasks. As expected R and RR executed very quickly whereas MX and MMMXFT took much longer.

As in the real world experiment, MX performed best and was closely followed by RR. However, unlike in the real world experiment where MMMXFT and R performed the same, MMMXFT performed much better

Accepted Manuscript.
Article accepted for publication in the International Journal of Grid and Utility Computing (IJGUC), 09/08/2019

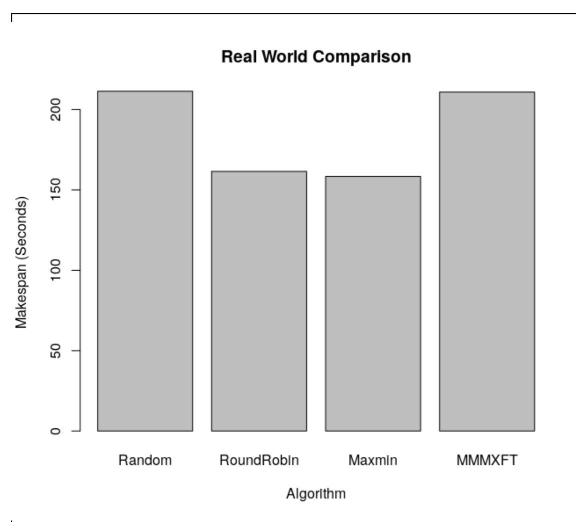

Figure 27 Experiment H, real world experiment results.

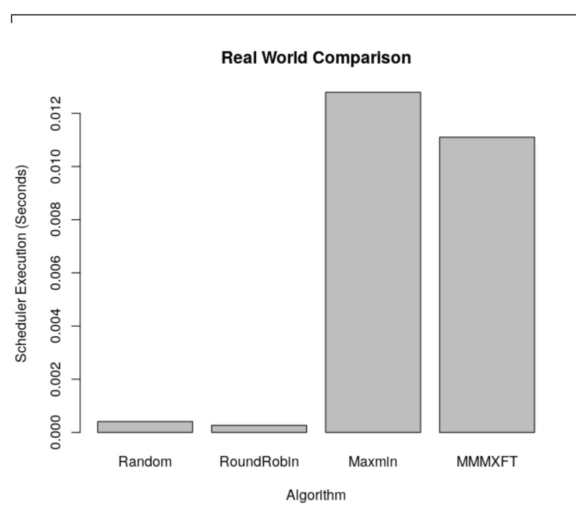

Figure 28 Experiment H, real world experiment results (scheduler execution time)

than R. This is likely related to the variation seen in Figures 24, 25 and 26. In the real world experiment, MMMXFT could have performed poorly because some small tasks were taking much longer than expected, causing the fast track to be executing long after the normal track had finished. In ScheduleSim there is no noise, so schedulers can accurately predict makespans.

5 Conclusion and Future Work

Summarising our findings, MMMXFT improves on MXFT as it has a similar overall makespan but suprior small task prioritisation. This makes it more suitable in a cloud computing context where small tasks may have SLAs that imply they cannot wait for large tasks to complete.

Adjusting the margin of MXFT and MMMXFT does not have a big impact on the overall makespan, but can make a big difference to the finishing time of small tasks. With MMMXFT, the larger the margin the less impact it has. In our experiments, it appeared that it was not very beneficial for small tasks if the margin exceeds 0.5 for MMMXFT. For MXFT a margin of between 0.25 and 0.35 appeared optimal for our experiments.

A benefit of using CMMMXFT was not seen in the experiments and graphs produced. However, this could be because the method of creating tasks for the experiments used a single normal distribution. Thus the clustering could not perform optimally since in every experiment there was really only one significant cluster. If the task patterns allowed for the superposition of two normal distributions then a clear benefit to using CMMMXFT may emerge.

With some small adjustments to TAWRR, the algorithm could be a good online alternative to RASA and the Fast Track algorithms. TAWRR was third best online algorithm tested.

Accepted Manuscript.
Article accepted for publication in the International Journal of Grid and Utility Computing (IJGUC), 09/08/2019

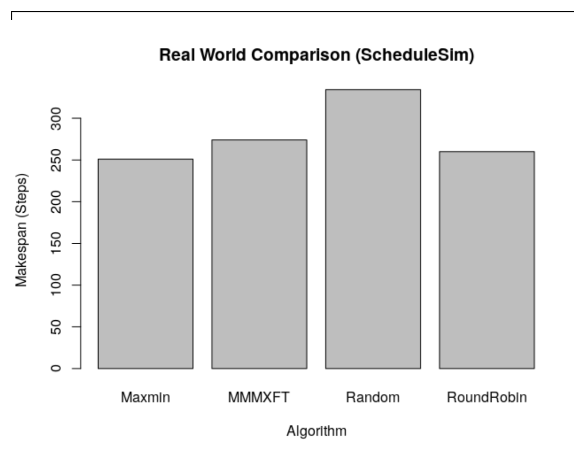

Figure 29 Experiment H (part 2), the real world experiment replicated in ScheduleSim.

Our work leads on to several opportunities for future research. The following is are some examples.

One such opportunity is investigating the impact of partial information. For instance, where task sizes and consumer speeds are only partially correct - as if they were estimated. This would be worthwhile as there is likely numerous services where the makespan can only be predicted or is possibly not know at all, or only known at runtime.

Furthermore, this work considers scheduling algorithms operating in batch mode. The algorithms operate on batches (metatasks) creating a schedule first then deploying it. While this is advantageous as the algorithms are operating with more information, but for some use cases, an online approach is simply required to instantly process tasks. In online mode, tasks are assigned to conumers individually as they arrive. Redesigning and testing the algorithms to work under these conditions would expand the applicability of this work. TAWRR, MCT and SCT could easily be adapted to operate in online mode.

TAWRR could be adapted to use the clustering seen in CMMMXFT. The implementation of TAWRR simply divides tasks in half after sorting them based on size. However, substituting this for clustering may prove more robust given an interesting profile of tasks.

References

[1] CIF Team, "`https://www.cloudindustryforum.org/content/uk-cloud-adoption-trends-2016`," 2016.

[2] Y. Wei and M. B. Blake, "Service-oriented computing and cloud computing: Challenges and opportunities," *IEEE Internet Computing*, vol. 14, no. 6, pp. 72–75, 2010.

[3] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of internet services and applications*, vol. 1, no. 1, pp. 7–18, 2010.

[4] R. Buyya, C. Vecchiola, and S. T. Selvi, *Mastering cloud computing: foundations and applications programming.* Newnes, 2013.

[5] M. Katyal and A. Mishra, "A comparative study of load balancing algorithms in cloud computing environment," *arXiv preprint arXiv:1403.6918*, 2014.

[6] Melborne CLOUDS Lab, "`http://www.cloudbus.org/cloudsim/`," 2016.

[7] C. Longbottom, *The Evolution of Cloud Computing: How to plan for change.* British Computer Society, 2017.

[8] J. Kaplan, W. Forrest, and N. Kindler, "Revolutionizing data center energy efficiency," 2008.

[9] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *ACM SIGARCH Computer Architecture News*, vol. 35, pp. 13–23, ACM, 2007.

[10] G. Cook and J. Horn, "How dirty is your data," 2011.

[11] Julie Chao, "`http://newscenter.lbl.gov/2016/06/27/data-centers-continue-proliferate-energy-use-plate`" 2016.

[12] P. Moggridge, N. Helian, Y. Sun, M. Lilley, V. Veneziano, and M. Eaves, "Revising max-min for scheduling in a cloud computing context," in *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2017 IEEE 26th International Conference on*, pp. 125–130, IEEE, 2017.

Accepted Manuscript.
Article accepted for publication in the International Journal of Grid and Utility Computing (IJGUC), 09/08/2019

[13] O. Elzeki, M. Reshad, and M. Elsoud, "Improved max-min algorithm in cloud computing," *International Journal of Computer Applications*, vol. 50, no. 12, 2012.

[14] K. Etminani and M. Naghibzadeh, "A min-min max-min selective algorihtm for grid task scheduling," in *Internet, 2007. ICI 2007. 3rd IEEE/IFIP International Conference in Central Asia on*, pp. 1–7, IEEE, 2007.

[15] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, *et al.*, "Scheduling resources in multi-user, heterogeneous, computing environments with smartnet," in *Heterogeneous Computing Workshop, 1998.(HCW 98) Proceedings. 1998 Seventh*, pp. 184–199, IEEE, 1998.

[16] T. D. Braunt, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswarans, A. I. Reuthert, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgeno, *et al.*, "A comparison study of eleven static heuristics for mapping a class of independent tasks onto ileterogeneous distributed computing systems," 2000.

[17] R. Armstrong, D. Hensgen, and T. Kidd, "The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions," in *Heterogeneous Computing Workshop, 1998.(HCW 98) Proceedings. 1998 Seventh*, pp. 79–87, IEEE, 1998.

[18] S. Parsa and R. Entezari-Maleki, "Rasa: A new task scheduling algorithm in grid environment," *World Applied sciences journal*, vol. 7, no. Special issue of Computer & IT, pp. 152–160, 2009.

[19] Paul Moggridge, "`https://bitbucket.org/paulmogs398/schedulesim`," 2017.

[20] Paul Moggridge, "`https://www.youtube.com/watch?v=o9kb04qztlc`," 2017.