Research Archive

Citation for published version:
Paul Moggridge, Na Helian, Yi Sun, Mariana Lilley, Vito Veneziano, and Martin Eaves, 'Revising Max-Min for Scheduling in a Cloud Computing Context', 2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2017), Poznan, August 2017.

DOI:
https://doi.org/10.1109/WETICE.2017.58

Document Version:
This is the Accepted Manuscript version.
The version in the University of Hertfordshire Research Archive may differ from the final published version.

Copyright and Reuse:
© 2017 IEEE.
Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Enquiries
If you believe this document infringes copyright, please contact the Research & Scholarly Communications Team at rsc@herts.ac.uk

Revising Max-min for Scheduling in a Cloud Computing Context

Paul Moggridge, Na Helian, Yi Sun, Mariana Lilley, Vito Veneziano
School of Computer Science. University of Hertfordshire
Hatfield, United Kingdom
p.moggridge@herts.ac.uk
and
Martin Eaves
Advanced Collection Systems
Hatfield, United Kingdom

Abstract*—Adoption of Cloud Computing is on the rise[1] and many datacenter operators adhere to strict energy efficiency guidelines[2]. In this paper a novel approach to scheduling in a Cloud Computing context is proposed. The algorithm Max-min Fast Track (MXFT) revises the Max-min algorithm to better support smaller tasks with stricter Service Level Agreements (SLAs), which makes it more relevant to Cloud Computing. MXFT is inspired by queuing in supermarkets, where there is a fast lane for customers with a smaller number of items. The algorithm outperforms Max-min in task execution times and outperforms Min-min in overall makespan. A by-product of investigating this algorithm was the development of simulator called "ScheduleSim"[3] which makes it simpler to prove a scheduling algorithm before committing to a specific scheduling problem in Cloud Computing and therefore might be a useful precursor to experiments using the established simulator CloudSim[4].

Keywords*—*Cloud Computing, Scheduling, Max-min.

I. Introduction

A. Cloud Computing

Cloud Computing is the latest evolution from a history of technologies that dominate our life today. The technologies include Distributed Systems, Virtualisation, Web 2.0, Service Orientated Computing and Utility Computing. Cloud Computing is increasingly being adopted to provide solutions to today's demand for information and connectivity. As an evolving technology, Cloud Computing has many open challenges[5]. These include resource pooling/server consolidation, SLAs, Quality of Service (QoS), energy management, traffic management, stability and fault tolerance[6][7]. These challenges are underpinned by the NP-Complete problem of scheduling. There are two important areas of scheduling which are allocating virtual machines (VMs) to hosts (servers) and brokering cloudlets (tasks) onto VMs [8][4].

One host can contain many VMs. When choosing the host for a VM, the allocator must ensure the host has adequate resources to support the VM. Resources to consider 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) take to run can be reduced.

These areas of scheduling in the Cloud present important opportunities for improving efficiency and performance. Server utilisation has been cited as low as 6%[9]. Through improved scheduling, tasks can be consolidated on fewer hosts to save power. Fully relieving hosts is important because servers consume 50% of their peak power at idle, which make it detrimental to run servers with a light work load[10]. Globally, datacenters consume approximately 1.5-2% of global electricity and this is predicted to grow at a rate of 12% annually[11]. Based on that prediction from 2011, datacenters may now be consuming as much as 3%-3.9% of global electricity.

B. Scheduling Algorithms

Scheduling algorithms are a variation of the NP-Complete problem of fitting x objects into y space. There are many algorithms proposed to solve this problem. An important difference to note is whether an algorithm acts on metatasks (batched) or individual tasks (online). Another important distinction to make is whether an algorithm behaves statically or dynamically - does the algorithm dynamically adapt it's behaviour in response to it's environment or does it always do the same thing? Another difference to note is how much information a algorithm requires. For instance, whether the algorithm needs to know execution time for each take and/or the performance of each VM.

C. Aim and Structure

The aim of this paper is to propose a novel algorithm which considers the Cloud scheduling problem of brokering cloudlets on VMs. This paper is structured as follows. Related Work, this section describes existing adaptions to the Max-min algorithm. Proposed Algorithm, this section describes the proposed novel algorithm. Description of ScheduleSim, this section details the novel simulator created for experimentation. Methodology, the methodology section describes the experiments carried out. Results and Discussion, the section presents the results and remarks of the findings. Conclusion and Future Work, this section reflects on the findings and identifies some research opportunities opened up by MXFT and ScheduleSim.

II. Related Work

A. Max-min

The Max-min is popularly researched scheduling algorithm. It assigns the tasks with the latest (Max) possible completion time to the fastest (Min) VM. Numerous papers find 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.

B. Improved

The Max-min Improved algorithm improves and adapts the Max-min algorithm for the Cloud[12]. By default the Max-min 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.

C. Selective

Etminani and Naghibzadeh proposed the Min-min Max-min Selective algorithm for scheduling in Grid environment[13]. The algorithm is based on the premise that in certain scenarios Min-min can outperform Max-min. It selects between Min-min and Max-min 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 Max-min would have chosen or the VM Min-min would have chosen. In the scenarios chosen, the algorithm always performed as well as either Max-min or Min-min.

D. Duplex/Greedy

Similar to the selective algorithm above, Duplex considers both Max-min and Min-min and uses the better solution[14][15]. Duplex executes both Max-min and Min-min, then chooses the solution that achieves the smaller sum of the predicted run-time - minimised over all VMs[16]. The difference to selective is that this algorithm chooses an algorithm for the whole metatask rather than an algorithm for each task.

E. RASA

RASA considers the distribution and scalability of VMs[17]. Like the Selective and Duplex mentioned above RASA also combines Max-min and Min-min. RASA uses the Min-min to execute small tasks before the large tasks and uses the Max-min algorithm to support concurrency. The RASA algorithm alternates between Max-min and Min-min task by task. If task one was assigned by Max-min, task two would be assigned by Min-min.

Considering the existing research into the Max-min algorithm, this paper builds on the Max-min Improved algorithm. However, unlike Selective, Duplex and RASA (rather than integrating the Min-min algorithm), this paper takes a novel approach by using a fast track instead. The hypothesis is that extending the Max-min algorithm in this way will improve task execution makespan for small tasks.

III. Proposed Algorithm

In this paper we propose a new algorithm, Max-min Fast Track (MXFT). MXFT extends Max-min Improved[12]. The below broadly describes the steps in the algorithm.

- 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 units of the task in the normal and fast track.
- Work out the ratio of the units in the fast track to the units 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 Max-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.

Furthermore, Algorithm 1 shows the detailed algorithm. Table I defines the variable used in the algorithm.

TABLE I. Algorithm notation used in algorithm 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.

IV. Description of ScheduleSim

A. Design Features

ScheduleSim is an open source scheduling simulator implemented in Java[3]. The simulator operates by using

Algorithm 1 Max-min Fast Track

```
for all $R$ do
    {Accumulate total speed for all resources.}
    $R^p += R_i^p$
end for
sort tasks $T$ biggest execution time first
for all $T$ do
    {Is index in first 60% of the number of tasks?}
    if $i < T^l * 0.6$ then
        append $T_i$ to $NT$
        {Accumulate total size of normal track tasks.}
        $NT^s += T_i^s$
    else
        append $T_i$ to $FT$
        {Accumulate total size of fast track tasks.}
        $FT^s += T_i^s$
    end if
end for
{Calculate ratio of speed to size.}
$\alpha = R^p/(NT^s + FT^s)$
{Calculate the size the fast track should have.}
$\lambda = R^p * FT_s$
sort $R$ fastest first
for all $R$ do
    if $FR^p < \lambda$ then
        append $R_i$ to $FR$
        skip $R_{i+1}$
    else
        append $R_i$ to $NR$
    end if
end for
for all $FT_i$ do
    for all $FR_j$ do
        {Find completion time.}
        $C_{ij} = E_{ij} + R_j^d$
    end for
end for
while $FT$ not empty do
    find task $T_i$ costs $maximumexecutiontime$
    assign $T_i$ to $FR_j$ which gives $minimumcompletiontime$
    remove $T_i$ from $T$
    update $R_j^d$
    for all $i$ do
        update $C_{ij}$
    end for
end while
for all $FT_i$ do
    for all $FR_j$ do
        {Find completion time.}
        $C_{ij} = E_{ij} + R_j^d$
    end for
end for
while $NT$ not empty do
    find task $T_i$ costs $maximumexecutiontime$
    assign $T_i$ to $NR_j$ which gives $minimumcompletiontime$
    remove $T_i$ from $T$
    update $R_j^d$
    for all $i$ do
        update $C_{ij}$
    end for
end while
```

discrete time steps instead of an event driven design. This allows fine gain control and simplifies design.

ScheduleSim uses generic terminology (producer,tasks, units, steps, schedulers and consumers) which prevents association of ScheduleSim to a specific context. Producers are where tasks are created. In Cloud terms this is where the cloudlets from users enter the Cloud. Tasks represent load on the Cloud, for instance, user applications to be ran. Schedulers are where tasks are routed - these execute the scheduling algorithms. Consumer are where tasks are processed, in Cloud these could be seen as VMs.

Outside of scope is modelling scheduler processing time. Schedulers are allowed to do as much processing as required without time advancing. It should be noted that network latency is not modelled - meaning that a task can fully transverses the network from the producer through any number of schedulers to a consumer in a single step. ScheduleSim simulates a single producer which makes the simulation inherently centralised. ScheduleSim is however capable of modelling hierarchical tree structures with the limitation that schedulers and consumers can only have only one parent. Schedulers and consumers have buffers in which they store tasks waiting to be processed. Consumers can only actively process one task at a time. Tasks are measured in units, the higher units the larger the task. Consumers are rated with a units per step, this is how many units they can decrement per step from the task they are actively processing. Producers can submit a metatask at any time step. The size and number of tasks in the metatask can be specified.

V. Methodology

Three experiments were conducted to look at the impact of different variables. Five algorithms were compared - two static algorithms (Random and Round Robin) and three dynamic algorithms (Min-min Max-min and MXFT). Each experiment was ran 10 times.

A. Gaussian Generation of Experiments

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

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

When:
μ Determines the position of the normal distribution.
σ Determines width of the normal distribution.

Consumers and tasks are created to follow the distribution. When creating either a task or consumer cause the target value to be exceeded, execution stops.

When adding a task or a consumer to the collection of tasks or consumers, would causes the target value to be exceeded, execution stops. This means there is always less total consumer speed or task size than the target.

B. Experiment One Setup

Experiment one varies the consumer μ. The consumer μ variable changes the position of the peak of the normal distribution. This changes speed of consumers created. II details the variables and there corresponding values for this experiment.

Fig. 1 shows created consumers of different sizes at the start of the experiment. The Y axis shows number of consumers and the X axis shows the speed of the consumers. The arrow shows how the normal distribution is shifted as the μ is altered. A normal distribution is used to weight the

TABLE II. EXPERIMENT ONE PARAMETERS

Variable	Value(s)
Consumer Min Size	2
Consumer Max Size	32
Consumer μ	2 to 32, increments of 1
Consumer σ	6
Consumer Target Speed	4000
Job Min Size	20
Job Max Size	240
Job μ	110
Job σ	44
Job Target Size	200000

creation of consumers of different speeds. Note that although the most common consumer speed varies over the course of the experiments, the total combined speed stays at a consistent level. The result of this is that in the first simulations there are many lower speed consumers and in the later experiments there are fewer but faster consumers.

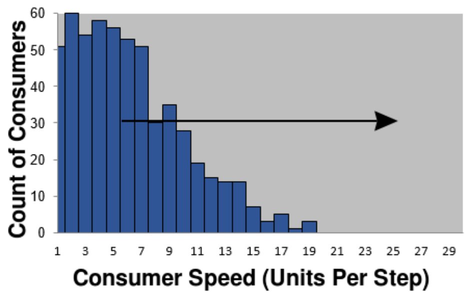

Fig. 1. Experiment One Variable - showing the most common consumer speed for the first simulations.

C. Experiment Two Setup

Experiment two varies the task μ - changing the size of the tasks created. Table III shows the variables and there corresponding values for this experiment.

TABLE III. EXPERIMENT TWO PARAMETERS

Variable	Value(s)
Consumer Min Size	2
Consumer Max Size	32
Consumer μ	17
Consumer σ	6
Consumer Target Speed	4,000
Job Min Size	20
Job Max Size	240
Job μ	20 to 240, increments of 5
Job σ	44
Job Target Size	200,000

Again note that the total combined size of the tasks stay at a consistent level. In Fig. 2 the Y axis shows number of task and the X axis shows the size of the tasks.

D. Experiment Three Setup

Changes the count of tasks. This is achieved by increasing the target for the total units of all tasks. This increases the workload without changing the distribution. The experiment runs the algorithm from with a few tasks to with many tasks.

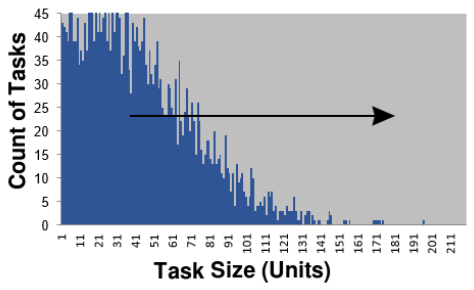

Fig. 2. Experiment Two Variable - showing the most common task size for the first simulation.

TABLE IV. EXPERIMENT THREE PARAMETERS

Variable	Value(s)
Consumer Min Size	2
Consumer Max Size	32
Consumer μ	17
Consumer σ	6
Consumer Target Speed	4,000
Job Min Size	20
Job Max Size	240
Job μ	110
Job σ	44
Job Target Size	20,000 to 800,000, increments of 5,000

Table IV shows the variables and there corresponding values for this experiment.

In Fig. 3 normal distribution does not change, instead the counts of task changes. Fig. 3 (like in 2) the Y axis shows number of tasks and the X axis shows the size of the tasks.

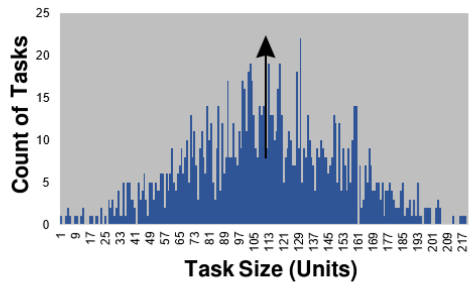

Fig. 3. Experiment Three Variable - showing the count of tasks for the first simulation.

E. Performance Metrics

1) Overall Makespan: In verbose, the max/latest finishing-time into 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.

2) Task Makespan: This is simply the time the task took to execute on a given consumer. E_{ij} represents the execution time of task j on i. This is the time the task took to run. In verbose, $T_j^f - T_j^s$, where T_j^s is the start of execution for task j and T_j^f is the finish time.

VI. Results and Discussion

A. Experiment One

In experiment one, the variable investigated was Consumer μ. The results in Fig. 4 show that the overall makespan reduces with fewer but faster consumers. This could be because the scheduling problem becomes less combinatorially complex - supporting this is that gap between the algorithms becomes narrower.

Note that in all of the result graphs the Random and Round Robin Algorithms are omitted because performance was significantly worst than Min-min, Max-min and MXFT algorithms. This result was expected as the Random and Round Robin algorithms are static algorithms and do not consider consumer load like the other algorithms investigated. In addition MXFT can seen to outperform Min-min.

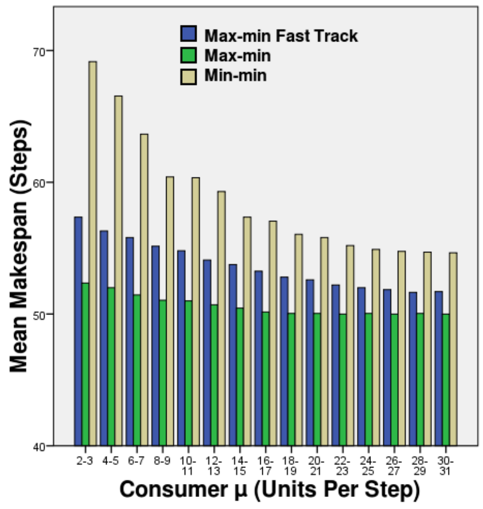

Fig. 4. Experiment One Results - mean overall makespan of the algorithms when given different distributions of consumers.

B. Experiment Two

The results in Fig. 5 show that the Max-min and MXFT algorithms both are able to achieve a lower overall makespan when provided with many small tasks opposed to a few large tasks. An explanation for this is that the Max-min algorithm by executing the few big tasks first, is left with the many smaller tasks towards the end of the simulation that it can compact into gaps created by the bigger tasks. Again MXFT can be seen to outperform Min-min. Seeing Fig. 8 may help understanding.

C. Experiment Three

The of experiment 3 showed linear relationship between the combined number of task units and the overall makespan. As Fig. 4 and 5 Min-min has the smallest mean overall makespan followed by Max-min and MXFT.

D. All Experiments

Tasks that fall in the fast track-able margin show reduced makespans (execution times). Data collected over all experiments shows that MXFT reduces task execution time for

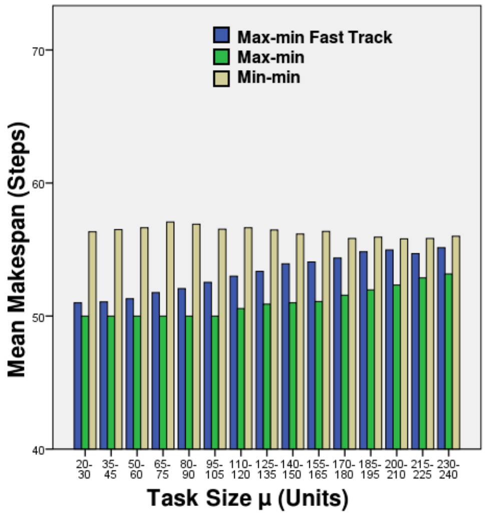

Fig. 5. Experiment Two Results - mean overall makespan of the algorithms when given different distributions of tasks.

small tasks up to the point where the tasks are too big to be submitted to the fast track. This attribute of MXFT is useful in the scenario that small task have stricter SLA agreements. Fig. 7 and 6 show that MXFT outperforms Max-min in task makespan.

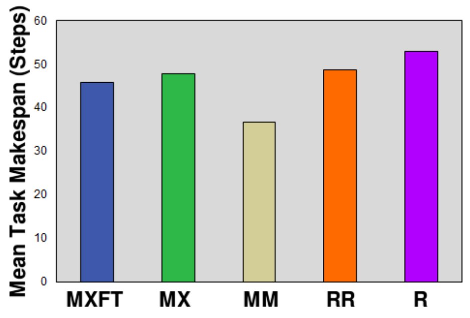

Fig. 6. Mean task makespan of equi-width bins based on task size.

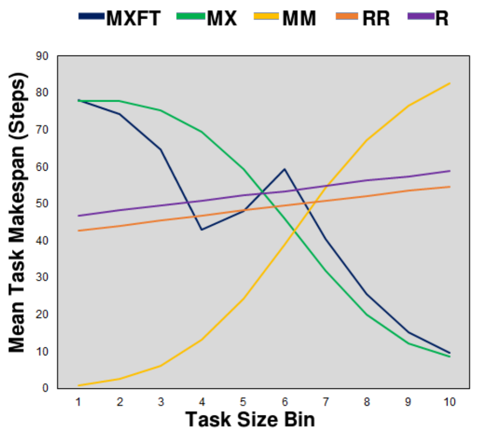

Fig. 7. Mean tasks makespans for each algorithm.

E. Reflection on Simulation

A unique benefit ScheduleSim is the ability to visualise the Scheduling. This novel way of interpreting the data allows for rapid and deep understanding of the implications of an algorithm. In Fig. 8 you can see the scheduling of the MXFT algorithm.

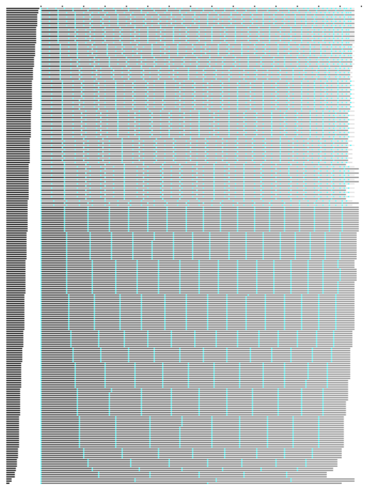

Fig. 8. Max-min Fast Track Visualised using ScheduleSim.

In Fig. 8, consumers (VMs) are represented as the black horizontal lines on the left, the longer the line the higher the units per step (MIPS) of Consumer (VM). Tasks are represented over time starting on the left going towards the right by the grey scale horizontal lines. The dashes along the top are at intervals of ten steps. The start of the processing of a task is marked by a cyan dot, the proceeding colour of the task represents the size of the task, the darker the task is harder it is to complete.

In Fig. 8 our algorithm can be seen to be performing reasonably optimally. There is very little wasted throughput. This is expected because the algorithm we extended (Max-Min) is within 20% of optimal. Although it should be noted this number was computed with small simulations as the solution had to be brute forced to confirm the best solution[14].

Better than Max-min, our algorithm can be seen prioritising small tasks. Every other fast server is elected as fast-track, and can be seen running small tasks on the fastest consumers.

VII. Future Work and Conclusion

This paper modifies the Max-min algorithm, however optimising the proposed algorithm was not explored. For instance, in this paper the “margin” value (the size of the Fast Track) was fixed to 40% of the count of tasks. However it is likely that adjusting this margin to suit variations in the tasks could yield better results. In addition, we can improve performance by using a more sophisticated approach to decide how to allocate consumers to the fast track once the tasks have been divided.

In conclusion, MXFT offers a compromise between task execution makespan of small tasks and overall makespan, which could be beneficial in a Cloud Computing context. The algorithm outperforms Max-min in task execution times and outperforms Min-min in overall makespan. Another outcome of this paper is the simulator ScheduleSim which provides a simple environment for testing algorithms.

References

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

[2] L. Newcombe, M. Acton, P. Bertoldi, J. Booth, S. Flucker, and A. Rouye, “Best practice guidelines 2016,” 2016.

[3] Paul Moggridge, “https://bitbucket.org/paulmogs398/schedulesim,” 2017.

[4] Melborne CLOUDS Lab, “http://www.cloudbus.org/cloudsim/,” 2016.

[5] 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.

[6] 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.

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

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

[9] J. Kaplan, W. Forrest, and N. Kindler, “Revolutionizing data center energy efficiency,” 2008.

[10] 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.

[11] G. Cook and J. Horn, “How dirty is your data,” 2011.

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.