University of Hertfordshire
School of Computer Science

Modular BSc Honours in Computer Science

6COM1055 – NETWORKS PROJECT

Final Report
April 2017

Load Balancing using Hierarchical Structure for Cloud Computing

S.O Omotayo
SRN – 13071467

Supervised by – Dr. Na Helian

Table of Contents

Abstract

Efficient task scheduling is an important aspect in any cloud computing environment. It directly affects resource utilization, makespan, and job allocation. In a hierarchical cloud setup, if the scheduling algorithms can perform efficiently, then the cloud environment will be able to provide adequate quality of service. In this project, the opportunistic algorithm alongside other algorithms are combined with other algorithms such as max-min and min-min with the aim of ensuring good resource utilization, makespan, and job allocation in the hierarchy. This hierarchical method is demonstrated using a batch mode vector bin packing simulator implemented in java which allows multiple layers of load balancing. The performance data of the hierarchical load balancing proves that the opportunistic algorithm was a good job distributor and the max-min algorithm remains a very efficient scheduling algorithm for cloud environments regardless of the layer it is used in.

Keywords – Makespan, nodes, response time, execution, algorithm, job, tasks

Acknowledgements

Special thanks to my supervisor Na Helian who has guided me throughout the whole project process and advised me along the way. Another special thanks to Paul Moggridge who has been very instrumental regarding the ScheduleSim simulator used to make this project a success.

List of Figures

List of tables

1 Introduction

1.1 Literature review

This chapter provides a literature review on the topics related to the domain of hierarchical load balancing and the algorithms used. It also provides an insight into the research work that has been done in this domain and how the same idea has been implemented in other areas like cloud computing. It will include cloud computing, scheduling, load balancing algorithms and the hierarchical structure model.

1.1.1 Cloud Computing

Cloud computing is an ever-evolving paradigm that enables on-demand network access to a shared pool of configurable computing resources such as networks, servers, storage, applications and services that can be rapidly provisioned and released with minimal management effort (Mell, Grance, 2011). Resource pooling is one essential characteristic out of many that the cloud is comprised of (Kaur, Luthra, 2012). Computing resources are pooled to serve multiple consumers and physical and virtual resources are dynamically assigned or re-assigned according to demand.

1.1.2 Load balancing

In a cloud computing environment, load balancing is used to distribute load accordingly, ensure adequate resource usage, maximise throughput and minimize response time. However, with the increasing size and complexity of the internet and the resources within it, load balancing has become a challenge in the cloud computing environment (Kaur, Luthra, 2012). Because of this load balancing problem, some work has been done to improve load balancing structure to produce better results. Task Scheduling is one of the most important factors that affect the efficiency of load balancing in the cloud computing environment and other environments or domains, Services provided by the cloud are heavily reliant on task scheduling which in turn affects user satisfaction. Scheduling plays a role in improving the flexibility and reliability of systems in a cloud environment (Singh, Paul, Kumar, 2014). An efficient cloud environment is one that finds the complete and best sequence in which various tasks can be executed and resources fully utilized appropriately to give the best satisfaction to users.

The cloud generally consists of computational resources that have different capabilities which makes scheduling so important. (Singh, Paul, Kumar, 2014) discusses two scheduling categories which are static and dynamic type scheduling and (Salot, 2013) shows that there are two job scheduling classifications which are batch mode heuristic scheduling and online mode heuristic scheduling.

Batch mode heuristic algorithms and online mode heuristic algorithms can be static or dynamic. The heuristic of the algorithm will determine what category the algorithm falls into.

1.1.3 Batch mode heuristic Algorithms

Jobs are grouped and collected into a set when they arrive in the system and the scheduling algorithm starts after a period of time (Salot, 2013). Examples of Batch mode heuristic algorithms are sufferage, Max-min, and min-min. The min-min heuristic begins with unmapped tasks and then the set of minimum completion time is computed for each task is computed. The algorithm uses the overall minimum execution time information to select the machine to allocate tasks. The process continues until all the tasks have been mapped (Laxmi, Kaur, 2012). Min-min works on the expectation that a smaller Makespan can be obtained if more tasks are assigned to the machines that execute them the earliest and fastest. The Max-min algorithm is very similar to the min-min algorithm such that it begins with unmapped or unscheduled tasks and the set of minimum execution times for each available resource is found

(Laxmi, Kaur, 2012). However, it is the task with the overall maximum completion that is selected and assigned to the corresponding machine. The Max-min algorithm is designed with the aim of increasing throughput, resource utilization, reducing overhead and attaining good performance and response times.

1.1.4 Online mode heuristic Algorithms

Jobs are scheduled as they arrive in the system (Laxmi, Kaur, 2012). Online mode heuristic algorithms are more suitable for cloud environments because of the heterogeneity of the cloud environment. Computational resource capability is likely to differ, or go through breakdown. Basic examples of online mode heuristic algorithms are opportunistic load balancing (OLB) and most fit task scheduling algorithm (Salot, 2013; Kaur, Luthra, 2012). NUCOLB (Non-Uniform Communication Costs Load Balancer) is another online mode heuristic algorithm used by (Pilla *et al.*, 2012). One main feature of the algorithm is that it dynamically reviews task distribution in the system to make better load balancing decisions.

1.1.5 Static type Scheduling

The static scheduling approach works by scheduling tasks in a known environment which means it already has information about the tasks and the resources available before execution as well as the execution time. Examples of static scheduling algorithms are max-min, min-min, opportunistic load balancing (OLB), minimum execution time (MET), minimum completion time (MCT), duplex, and Genetic Algorithms (GA).

1.1.6 Dynamic type Scheduling

Dynamic type scheduling not only depends on the submitted task but also the cloud environment such as number of resources, computational power, bandwidth, and the current state of the system to make load balancing decisions. Examples of dynamic scheduling programs are opportunistic load balancing (OLB), and ESCE (Equally Spread Current Execution). Equally Spread current execution works by equally spreading load to the available and adequate resource according to the job size. The expected impact of the algorithm is to improve the response time and the processing time (Daryapurkar, Deshmukh 2013). Dynamic scheduling algorithms have higher performance than static algorithms because of the dynamic nature of cloud environments. However, it has a lot of overhead in comparison to its counterpart (Salot, 2013). Such overhead can come in the form of communication costs between resources in the cloud environment or communication overhead from real-time communication with the networks. The dynamic load balancing approach can be split into two further classifications; Distributed and Non-Distributed approach (centralized) (Kaur, Luthra, 2012; Kaur, Luthra, 2014).

1.1.6.1 Non-distributed approach (centralized)

Load balancing in the non-distributed approach is done by one or many resources. This approach is split into two further categories; centralized and semi-distributed approach. The centralized approach works by executing the load balancing algorithm on one single node, this node is solely responsible for all operations and controlling in the system such as distribution of workload.

The semi-distributed approach works by dividing the nodes of the system into clusters where there is a central controller controlling all the operations of load balancing. The central controller is selected dynamically during load balancing. (Zheng *et al.*, 2011) proposes a load balancing hierarchical strategy similar to the semi-distributed approach, the strategy divides processors into autonomous groups where each processor has a controlling root that makes load balancing decisions across a group of processors. The Centralized approach generally

makes excellent load balancing decisions but suffers from scalability issues. Hence why the distributed approach is very suited to cloud computing environments which tend to be dynamic and vast (Kaur, Luthra, 2014).

1.1.6.2 Distributed approach

Nodes in the cloud execute the algorithm together, dividing the work between themselves (Kaur, Luthra, 2014). In addition, no one single node is responsible for load balancing decisions or job scheduling, instead all domains are responsible for making load balancing decisions by monitoring the cloud network. (Kaur, Luthra, 2014) explains the distributed approach of dynamic load balancing as nodes independently building its own load vector where each vector collects load information of other nodes and decisions are made locally using the information from the load vector. (Zheng *et al.*, 2011) employs a distributed approach where nodes in the system collect information about the CPU speeds to make good load balancing decisions allowing nodes under its control to receive work proportional to its size. The distributed approach is more suitable for widely heterogeneous systems which can be found in a cloud computing environment.

1.1.6.2.1 Cooperative

Nodes work together in the environment to achieve a better response time of a system (Kaur, Luthra, 2014). (Pilla *et al.*, 2012) proposes the NUCOLB (Non-Uniform Communication Costs Load Balancer) algorithm to improve the performance of applications based on parallel multi-core systems. The algorithm used is an online mode heuristic algorithm that dynamically reviews the way tasks are being distributed on the processing units. The algorithm can successfully maximize use of cores and minimize communication costs which is very good for dynamic environments.

1.1.6.2.2 Non-cooperative

Nodes work independently to achieve a better response time of the tasks (Kaur, Luthra, 2014). (Nandagopal and Uthariaraj, 2010) utilizes a hierarchical load balancing algorithm approach to improve the average response time of tasks in a grid application with minimum communication overhead. All nodes in this grid all work autonomously to improve the response time of individual tasks.

1.1.7 Hierarchical Structure

Tasks can be rendered to datacentres in a geographical region and can be organized in a hierarchy with each level utilizing one or more load balancing algorithm(s). In conclusion, there is load balancing on each layer of the hierarchy. (Pilla *et al.*, 2012) use of a hierarchical algorithm approach to solve the load balancing problem of distribution of application load on parallel NUMA (Non-uniform memory access) machines can be applied to solving the problem of load balancing in cloud computing. A similar idea is implemented with large supercomputers in (Zheng *et al.*, 2011) where the application data and machine topology information is used for hierarchical load balancing. Processors in the supercomputers can be organized in a hierarchical structure with each level of the structure utilizing a load balancing algorithm. The lowest level of this structure utilizes a greedy-based load balancing algorithm that works by assigning tasks that have higher execution times to the most capable processors and a refinement-based load balancing algorithm is used at the high level (Zheng *et al.*, 2011) that

makes global load balancing decisions at each level of the hierarchy. Hierarchical load balancing is used overcome the scalability issue that a centralized scheme would have without sacrificing the quality of load balance achieved (Zheng *et al.*, 2011). A three phases scheduling is utilized to minimize the load balancing problem in a hierarchical cloud computing network (Wang *et al.*, 2011), algorithms are combined for use in the 3-level hierarchy to discover which combination best utilizes the resources in the shortest time. The hierarchy heuristic is a request manager at the top level, a service manager on the next lower level, and service nodes at the lowest level which deal with the actual task execution and is also the final load balancing step. (Wang *et al.*, 2011) uses different combinations of EOLB (enhanced opportunistic load balancing), OLB (opportunistic load balancing), EMM (Enhanced min-min) and MM (min-min) algorithms in the hierarchy in different orders to find the optimum combination. The combination of EOLB and EMM is the most efficient because the performance of the system is enhanced. (Nagaty, 2014) suggests a hierarchical cloud approach where datacentres are distributed geographically provides an added layer of redundancy and load balancing. Dynamic algorithms have been used throughout the hierarchy and most structures have utilized the same algorithms on each level of the hierarchy. The Hierarchical structure will be a cooperative one as the aim is to improve response time of system not the individual tasks.

This project aims to utilize a hierarchical structure for a cloud computing environment to ensure good resource utilization, efficient job allocation, and higher job execution rate.
The Hierarchical structure will be a cooperative as the aim is to distribute the jobs to be executed more appropriately in the system.

1.2 Research Gaps

Common load balancing algorithms such as max_min and min_min have not yet been used in hierarchical environments. Because these algorithms are very efficient algorithms, it could perform efficiently when combined with other algorithms, and can take advantage of the benefits of the hierarchical approach to cloud computing.

1.3 Aim of Project

The aim of the project is to develop an in-depth understanding of how the structure of cloud environments and scheduling algorithms in use can affect the performance and quality of service expected from cloud computing setups.

1.4 Objectives of Project

- Conduct Literature Review on the areas of cloud computing such as load balancing algorithms, scheduling, and structure of cloud.
- Design Hierarchical Approach.
- Have a clear understanding of the ScheduleSim simulator and why it is so useful.
- Implement Algorithms for the different layers of the hierarchy.
- Implement Opportunistic service broker algorithm
- Evaluate usage of algorithms.
- Discover which Algorithm combination yields the best results.

1.5 Advanced objectives

- Implement Stable marriage algorithm.
- Evaluate feasibility and suitability of stable marriage algorithm the Load Balancing Domain with the result of the implementation.

1.6 Brief overview of upcoming chapters

Chapter 2: Design of Load Balancing
It is focused on the design of the hierarchical approach that is designed to utilize multiple algorithms in each layer so that tasks can be properly scheduled.

Chapter 3: Simulator choice
This chapter discusses the choice of simulator, what kind of output to expect from the chosen simulator and the advantages of using such a simulator.

Chapter 4: Implementation and Experiments
This chapter is focused on how the experiments are implemented to follow the design and provides some scenarios that will be tested.

Chapter 5: Results and Discussion
Here the results of the experiment are discussed and analysed.

Chapter 6: Conclusion and Future work
In this chapter, the best algorithm combination is identified and the opportunistic algorithm is critically analysed and improvements are suggested. The project is also evaluated and future work proposals are included.

2 Design of Load Balancing

Key Terms: Jobs, Producer, UserBase, Scheduler, datacentre, Consumer, virtual machine

2.1 Hierarchical approach

The hierarchical approach is essentially modelled as a tree structure where there are parent-child relationships. Adopting the model introduced in (Wang *et al.*, 2011), the Hierarchical structure strategy in this project is to divide the scheduling process into a hierarchy with three independent layers. The layers are Producer, Scheduler and Consumer. Each layer will utilize different load balancing algorithms with the aim of choosing the best possible resource for job execution. The hierarchical structure is shown here in **Figure 1.** The approach will not be one where the nodes within the tree are distributed geographically but instead, the hierarchical structure will be within a geographical region.

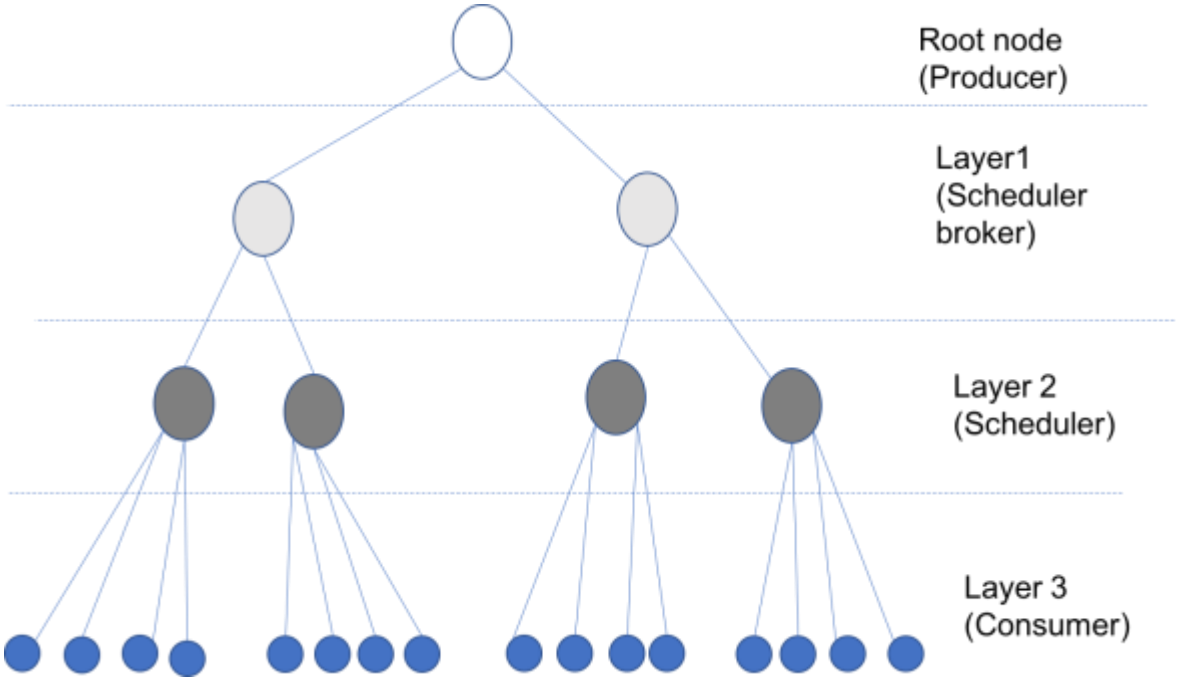

Figure 1 – Example Hierarchical Structure

A. Producer (Root node)

This is the highest and root layer of the hierarchy. The producer in the hierarchical model is responsible for generating and delivering traffic (load) to be delivered to the scheduler broker layer for further load balancing. This is the first point of contact of the scheduling environment to the outside network, a practical example of this layer is where a group of users in a geographical region submit internet requests to perform a specific action. The producer layer will collate these requests and sort them in some order depending on the algorithm in use and it will then deliver it to the Scheduler Broker.

B. Scheduler Broker (Layer 1)

The schedule broker layer is the first layer of load balancing in the hierarchy. This layer is responsible for the initial distribution of jobs after receiving jobs from the producer. It makes use of different sets of algorithms to ensure optimal distribution of jobs to proper schedulers for more load balancing. This is essentially the most important layer of the hierarchy because further load balancing decisions to be made will depend on the distribution of jobs from this layer. This layer also operates in a restriction of one child to one parent, In addition, this layer

can have multiple schedulers of different capabilities i.e. consumers capabilities under each scheduler are collectively higher or lower than its counterpart. Information can also be sent from the consumers to the schedulers, an example is the consumer capability (Units per Step). This information can be used by the scheduler to make load balancing decisions. Load Balancing decisions must be correct and suitable for the type of job received, although no actual execution of jobs happens in this layer, a bad decision in distributing the jobs at this layer can result to unwanted situations such as bottleneck, high makespan values, and latency.

C. Scheduler (Layer 2)

The Scheduler layer is a sub-layer of the scheduler broker. It is a representative of datacentres in a cloud environment. The scheduler layer receives tasks or jobs to be executed from the producer and then applies a load balancing algorithm within the layer to choose the best consumer to execute the specific type of job(s). There can be any number of schedulers under one scheduler broker but each scheduler can must have just one parent. All the schedulers are all in the same geographical region.

D. Consumer (Layer 3)

The consumer is the lowest and last layer in the hierarchy. It is responsible for execution of jobs or tasks that are sent from the producer in the top most layer to the scheduler broker and then to the schedulers. There can be any number of consumers under a scheduler and each consumer can only be a child of one scheduler. Each consumer can have different task execution capabilities or uniform execution capabilities, and the number of consumers under each scheduler do not have to be uniform. Having a non-uniform number of consumers with varied task execution capabilities is best for modelling a cloud environment as they are heterogeneous in application.

2.2 Algorithms to be used in hierarchy

A. Round-Robin

The Round-Robin algorithm is one of the basic algorithms used in distributed environments. The concept of the algorithm forms much of the foundation that other algorithms are created to fulfil. round-robin maintains a queue of incoming requests and allocates them to resources in a cyclic time scheduling manner meaning that time slices are given to individual jobs when they are assigned to resources. However, because of the nature of the simulator (ScheduleSim) used, the round-robin algorithm will distribute the jobs evenly among the resources but ignore time scheduling. The simulator works using discrete time steps rather than an event driven heuristic. The Round-Robin algorithm can be applied in both the scheduler broker layer and the scheduler layer.

B. Max-min

The Max-min algorithm is a commonly used algorithm in distributed environments. The Max-min algorithm is a batch mode heuristic algorithm because it starts with a set of unscheduled tasks and then moves to calculating the execution matrix and expected completion times of each tasks on the available resources. The next step of the Max-min algorithm is to choose the task with the overall maximum expected completion time and assign the job to the resource with the minimum overall execution time. This process is repeated until task-set becomes empty.

C. Max-min Fast-track

The Max-min fast-track algorithm is a combination of the Max-min and min-min algorithm that aims to use the advantages of both algorithms and avoid the drawbacks of both. An example is where the min-min algorithm suffers from starvation. Jobs that require more capable resources are neglected and the smaller jobs that do not need highly capable resources have higher priority. The Max-min fast-track algorithm works by working out the delay of the resources which is computed from the completion times of the consumers. The jobs are then sorted in a max first order. Based on a margin, it selects jobs to be fast tracked and jobs to be on the normal track, the fastest consumers are then placed on the fast track to sort the fast-tracked jobs and consumers remain there until the target is met. The remaining consumers are placed in the normal track to execute the jobs placed on the normal track.

Table 1 - Notation used for Algorithm. (Moggridge et al.., 2017)

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 meta-task (Tasks to schedule).
T^l	Number of tasks
Ti	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.

Max-min Fast-track Algorithm Pseudo code (Moggridge et al.., 2017)

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_i biggest execution time first
6: for all T do
7: {Is index in first 60% of the number of tasks?}
8: if $I < T^s * 0.6$ then
9: append T_i to NT_i
10: {Accumulate total size of normal track tasks.}
11: $NT^s += T_i^s$
12: else
13: append T_i to FT_i
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: $\alpha = R^p / (NT^s + FT^s)$
20: {Calculate the size the fast track should have.}
21: $\lambda = R^p * FT_s$
22: sort R_i fastest first
23: for all R do
24: if $FR^p < \lambda$ 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_k costs maximumexecutiontime
39: assign T_k to F R_j which gives minimumcompletiontime
40: remove T_k from T
41: update R_j^d
42:
43: for all i do
44: update C_{ij}
45: end for
46: end while
47: for all FT_i do
48: for all FR_j do
49: {Find completion time.}
50: $C_{ij} = E_{ij} + R_j^d$
51: end for
52: end for
53: while NT not empty do
54: find task T_k costs maximumexecutiontime
55: assign T_k to NR_j which gives minimumcompletiontime
56: remove T_k from T
57: update R_j^d
58:
59: for all i do
60: update C_{ij}
61: end for
62: end while

D. Min-min

The min-min algorithm is very similar to the Max-min algorithm, it is also a batch-mode heuristic algorithm that begins with a set of unscheduled tasks and then the minimum expected completion time for the all the tasks in the set is calculated. After the completion times are calculated, the task with the minimum expected completion time is selected and assigned to the resource with the corresponding execution time. Each task that is completed is removed from the set and the process is repeated until all the tasks are completed or the task-set is empty.

E. Opportunistic Algorithm

The opportunistic algorithm is an algorithm that aims to find the most idle consumer out of all the possible consumers in the hierarchy and allocate a job to the scheduler that is a parent (scheduler) of that consumer. The algorithm is a combination of Opportunistic load balancing and round-robin, it takes the advantages of the two algorithms and combines them to ensure greater effect. Round-robin is used for the first step of the algorithm so that every consumer receives a job of random size, this is so that each consumer can compute their initial delay times. Delay times are then further updated as jobs are allocated to schedulers and then to consumers in the subsequent steps. The remaining run of the algorithm is the opportunistic algorithm where jobs are allocated to schedulers (datacentres) with the smallest delay times. The criteria used to make decisions in allocating jobs to schedulers is the delay of the consumer, the remaining units of the task to be executed, and the computational ability (units per step) of the consumer. The opportunistic algorithm does not consider the size of the job to be allocated but it is still able to attain good load distribution since it is not used on any of the layers below layer 1. In addition, any jobs allocated by layer 1 will still have to go through further load balancing which essentially makes load balancing more efficient. This algorithm is not used on any of the lower layers because it is poor in terms of job execution which leads to very high MakeSpan values which in turn can have an adverse effect on the cloud quality of service.

Table 2 – Notation for Opportunistic Algorithm

R	Resources (Consumers or Vms)
R_i	A Consumer
D	Resources (Schedulers or Datacentres)
D^C	Children of Datacentre
D^C_i	A child of a Datacentre
R^S	Combined Resource Speed
R^S_i	Speed of a consumer
R^P_i	A resources speed (processing speed)
R^D_i	A resources delay (time until it is free)
R^P	Consumers parent
T	Meta-task to schedule
T^W	Waiting tasks
T^W_i	Task in waiting tasks
T_i	A Task
T^S_i	A tasks size
T^R_i	Remaining units of a task

Opportunistic Pseudo code

```
if count = 0 then
        for all D^C do
                for all R_i in D^C do
                        {submit T_i to R_i}
                end for
```

```
        end for

        for all D^C do
                for all R_i in D^C do
                        {accumulate R^D_i}
                end for
        end for

        increment count by 1
end if

if size D^C > 0 then
        for all D^C do
                for all R_i in D^C do
                        {accumulate all units per step of consumer}
                end for
        end for

        while size of T^W > 0 do
                assign T^W_i to variable
                while variable has next value do
                        assign R_i to a variable
                        assign R^P_i of R_i to a variable
                        updated delay = T^R_i – (R^S_i + R^D_i)
                        assign updated delay to variable
                        if updated delay < start delay then
                                assign updated delay to value
                                update variable holding R^P_i to variable
                        end if
                end while
        submit T^W_i to datacentre in variable holding R^P_i
        remove T^W_i from T^W
        end while
end if
```

3 Simulator choice

3.1 ScheduleSim Simulator

The ScheduleSim simulator is used to model and execute the hierarchical structure in this project. ScheduleSim is a batch mode bin packing simulator implemented in java, the scheduling algorithms used in the simulator are batch mode heuristic type algorithms that group a number of tasks when they arrive and then allocates jobs to resources based on the algorithm in use. For the simulator run, instead of an event driven design the simulator uses discrete time steps. The usage of discrete time steps is very appropriate for this project because it allows for a simple design and more control where features can be added or removed without too much effect.

The terminology used in the simulator are producers, tasks, units, steps, schedulers and consumers. Schedulers and Consumers keep task waiting to be processed in buffers and consumers are rated with a unit per step where the rating is how many units they can decrement per step from the tasks they are actively processing. Producers can submit tasks of different units at any time. The size and number of tasks can be specified in the simulator. Below Table 3 shows what the terminology used in ScheduleSim corresponds to in the cloud computing environment.

Table 3 - ScheduleSim Terminology

ScheduleSim	Cloud Computing
Task(s)	Load, cloudlet (if more than 1)
Producer	User Base
Scheduler	Datacentre / service broker
Consumer	Virtual Machine (VM)
Units	load size
Units per Step	Computational ability of entity (scheduler or consumer)

An example run of the ScheduleSim simulator is where a cloudlet consisting of a number of tasks grouped together enters the cloud. The Producer is responsible for generating cloudlets of different units and resource requirements, the tasks are then routed to the schedulers where scheduling algorithms are executed and job allocation to consumers occur. Lastly, the final step is where the tasks are routed from the schedulers to the consumers which are responsible for executing each individual task submitted.

Because of the simulators discrete time step design, tasks are routed from the producer to the consumer in one time step. This means there is no latency between the transfer of tasks from the producer to the consumers. Although the simulator does not fully represent how fast a task can be routed and executed on a consumer in a cloud computing environment, the simulator is able to give a visual representation of job execution and allocation and provide the number of successfully executed tasks by a consumer. The output of the simulator gives adequate information regarding the scope of the project. Figure 2 shows an example output that can be provided by ScheduleSim where the length horizontal bars to the most left of the figure before the first black rectangle depict the executing capability of a consumer where the longer the rectangle is, the more capable it is. The colour of the bars after the cyan squares depict the size

and start of the task to be executed where the darker the bar is, the more difficult the task is to execute.

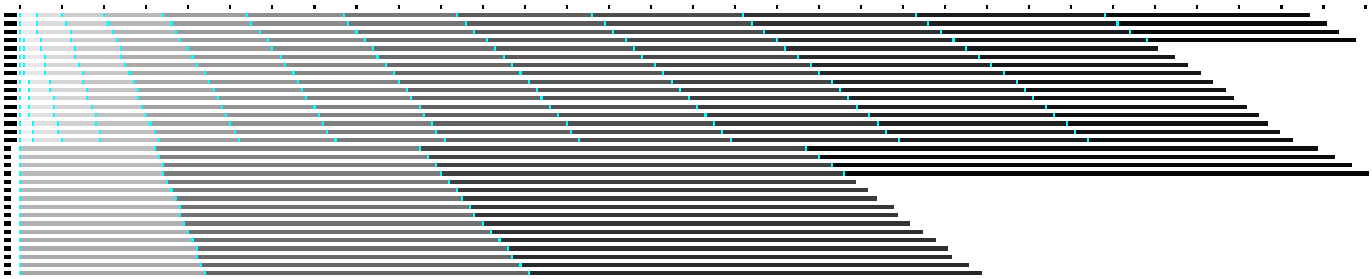

Figure 2 – Visualization of results by ScheduleSim

3.2 Other Simulators considered

Cloud Analyst

Cloud Analyst is a simulator built on top CloudSim which is a very popular tool for simulating large scale heterogeneous cloud environments. Cloud analyst allowed setup of various datacentres with different capabilities such as CPU processing power, memory, number of possible virtual machines and hardware units (servers). The Hierarchical model was already setup in CloudSim by default in the form of the user base, service broker and the datacentre broker. Cloud Analyst is a very good tool to use to model the structure because it provides well detailed results together with graphical representations of the events of the simulation e.g. a graph of response times per region, hourly processing times and cost of using the virtual machines. Figure 3 provides an example Cloud Analyst output.

Response Time by Region

Userbase	Avg (ms)	Min (ms)	Max (ms)
UB1	300.06	237.06	369.12

User Base Hourly Response Times

UB1

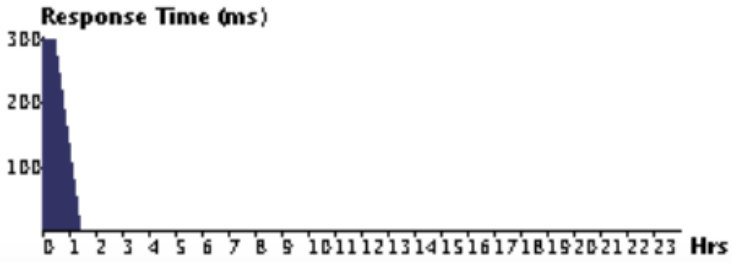

Figure 3 - cloud analyst example output

However, Cloud Analyst is not used in this project because the kind of output needed for the project is very different in comparison to what the Cloud Analyst simulator provides. The Cloud Analyst simulator output consists of variables like Datacentre processing time,

Datacentre Request servicing times, data transfer cost, and virtual machine cost whereas the output of ScheduleSim provides variables such as overall makespan, makespan of group of different sized tasks, utilization and number of complete tasks. Another reason why cloud analyst is not used is because the individual virtual machines in the simulator execute tasks concurrently and one of the main factors for the hierarchical model in this project is that all tasks are independent of each other.

4 Implementation and Experiment

4.1 Methodology

Several experiments will be conducted to compare the impact of different types of scheduling algorithms on different layers of the hierarchical model. The Generation of tasks are designed to be dynamic, the reason for this is because, in a real cloud environment, tasks are most likely to be of different sizes, be of different amounts, and will require different types of consumers to execute them. For this reason, a Gaussian, an Incrementing pattern, and a flat pattern type generation of jobs are used where the producer can generate different sized jobs of different amounts, increment the jobs size by a fixed amount and create a steady stream of fixed size jobs. This is used because it best models a cloud environment and its dynamic characteristics. A random job generation would have been used instead of the incrementing pattern to model the dynamic nature of cloud environments but because of the random nature of the jobs created by the random pattern, the results of the experiments will not be consistent as you would want all experiments to be fair. Six algorithm are compared where two are static (max-min and min-min) and the remaining are dynamic (Fast track and Opportunistic). Each experiment is run 5 times and the average of the results is used as the final set of results. Below Table 4 shows the six algorithm combinations that will be used for different test scenarios, and Table 5 shows the task bin meaning for experiments.

Table 4 - Experiment Design

Hierarchical Design Script	Sub-Manager of Second layer	Executive node of third layer
1	Opportunistic	Max-min
2	Opportunistic	Min-min
3	Opportunistic	Max-min Fast Track
4	Max-Min	Max-Min Fast Track
5	Max-Min Fast Track	Max-min Fast Track
6	Max_Min	Max_Min

Table 5 - Experiment Design 2

Task Bin	Meaning
0	Contains average task makespan for smallest tasks
5	Contains average task makespan for medium sized tasks
10	Contains average task makespan for large sized tasks

Why the combinations of algorithms were chosen?

As the aim of the hierarchical design was to ensure adequate resource utilization, job allocation, and higher job execution rate (Makespan), the algorithms chosen were chosen with that criteria in mind.

Design Script 1 – The opportunistic algorithm is generally good for utilizing resources in a cloud environment and the max_min algorithm is good for ensuring good makespan values so a combination of the two algorithms is proposed to fulfil the aims of using the hierarchical design.

Design Script 2 – The min_min algorithm works like the max_min algorithm but the complete opposite. The combination with the opportunistic algorithm is used as a benchmark for comparison, and the aim of this combination is to also meet the aims of the first script.

Design Script 3 – This algorithm combination is used to evaluate the concept that if jobs are distributed adequately, then it should result to good job execution rates.

Design Script 4 – This combination is used because the max_min is a very efficient performing algorithm in cloud computing and the fast track algorithm is good for splitting big and small jobs and allocating jobs to consumers. This combination will be used to evaluate if their advantages can be combined to provide good job allocation and job allocation rate.

Design Script 5 – This combination is used to evaluate if the modified max_min algorithm (max_min fast track) can be both a good a good service broker algorithm and a good job execution algorithm. The combination should be able to attain good job execution rates and utilize the hierarchy efficiently.

Design Script 6 – As max_min is one of the most efficient, best performing and simplest algorithms in cloud computing, it is used to observe if it can fulfil the target of using the hierarchy design for load balancing.

4.2 Assumptions

The Assumptions for the experiments are:

- Tasks are independent off each other and there is no communication between the nodes within a layer.
- There is also no migration of tasks between nodes in a layer and nodes in different layers. The reason why migration is not possible is because it is very complex to model in a simulator and is best simulated when experimenting on an actual datacentre or supercomputer. However, the scheduler on layer 1 can collect information about the scheduler below it as well as the consumers. The reason why there is no communication between the nodes (consumer) within a layer is that it is out of the scope of this project and is not possible to model with the simulator as the model works with a discrete time step design explained earlier.
- None of the consumers can go offline, this is also because of the limitation of the simulator.

4.3 Limitations

- Simulator did not allow dynamic changing of consumer capability so at every start of the experiment, the number of consumers and their capabilities were constant. This makes a fully dynamic environment harder to attain.
- Deciding which of load balancing algorithm to use depending on the type of consumers available within a scheduler was not possible using the simulator.

4.4 Experiment Scenarios

A. Overview

The experiment scenarios for this project are to evaluate:

- How the algorithms will perform in a non-uniform cloud environment where a scheduler can hold a number of consumers with low computational capability while the other scheduler will hold considerably more consumers with larger computational capabilities.
- How the algorithms will perform when the hierarchy is uniform. i.e. where the schedulers in the hierarchy have consumers that are of the same amount and total computational capability.
- How the algorithms will perform in a non-uniform hierarchy where the total amount of consumers are high but not equally distributed to each scheduler, and there is a considerable significant difference between the computational abilities of the consumers collectively.

B. Experiment one setup

The first experiment is building multiple non-uniform hierarchies where every layer is making use of a combination of the six algorithms mentioned earlier (see Table 4) and the tasks generated are also varied. The number and the speed of the consumers are not similar to each other and consumers are labelled incrementally from left to right so the first consumer on the left is consumer 0 and the last on far right is consumer 9. Jobs are allocated depending on the algorithm in use on the layer, and job generation will be of type Gaussian and Incrementing pattern. Figure 4 shows the structure of the non-uniform hierarchy where the layer two schedulers have different amounts of consumers with different capabilities as well. The consumers are also labelled from left to right where scheduler 1 has 2 consumers and scheduler 2 has 8 consumers.

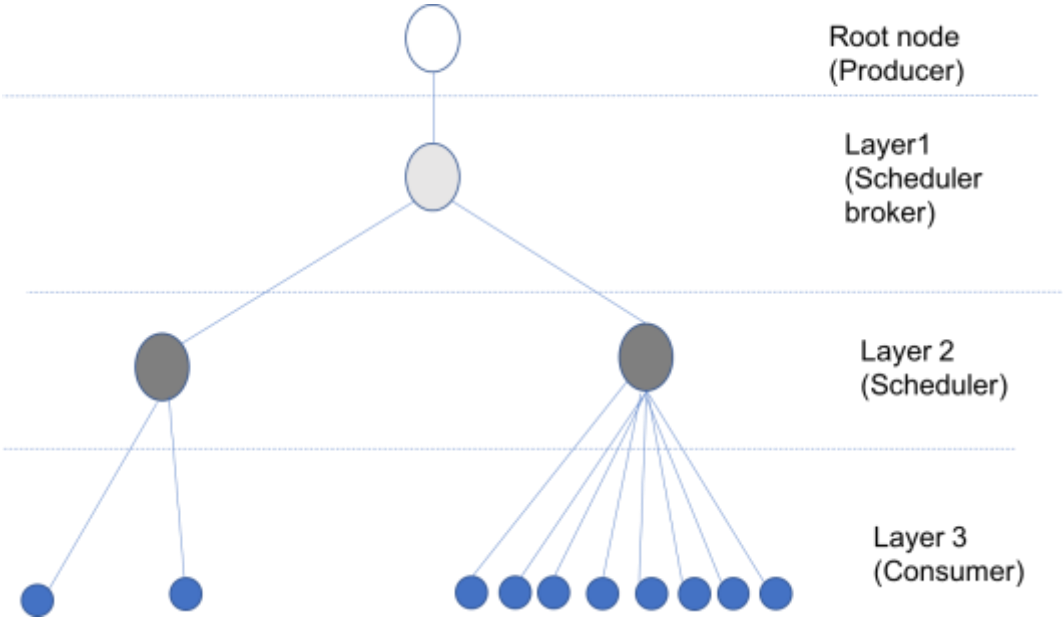

Figure 4 - Hierarchy to utilize different algorithms

Table 6 - Experiment one parameters

Type	Variable	Value(s)
Producer	Job Min Size	20

	Job Max Size	150
	Job μ	20 to 150 increments of 1
Scheduler	Number of Schedulers	Layer 1 -> 1 Layer 2 -> 2
Consumer	Consumer Min units	10
	Consumer Max units	30
	Number of Consumers	10

The reason for such a structure is to model the difference of datacentres in cloud environments. For example (Wang *et al.*, 2011) employs a tree system where each group have nodes of different capabilities. This is very useful to elicit accurate results.

C. Experiment two Setup

Table 7 - Experiment two parameters

Type	Variable	Value(s)
Producer	Job Min Size	20
	Job Max Size	240
	Job μ	20 to 250 increments of 5
Scheduler	Number of Schedulers	Layer 1 -> 1 Layer 2 -> 2
Consumer	Consumer Min units	10
	Consumer Max units	60
	Number of Consumers	50

In this experiment, the number of consumers is increased, and the minimum consumer speed is on 10 and the maximum speed is now at 60. The same six algorithm combinations in Table 4 will be used for the experiments and the hierarchy will be a uniform one in terms of the number of consumers on each scheduler. In this experiment, the generation of tasks will include the Gaussian, flat and Incrementing generation. The combination of job generation is used to attempt to model the heterogeneity of requests in cloud environments. At certain points, requests to datacentres are often random in size or grow incrementally over a period and then falls to a certain value. Such generation of incoming requests is very useful in modelling hierarchical environments as it is an advantage of using a hierarchical design that it can adapt to drastic changes in the environment. The consumers are shared equally in numbers but not in capability, consumers are allocated to the schedulers subsequently after each other and the units per step of the consumer is not considered before allocating a consumer to a scheduler.

D. Experiment three Setup

Table 8 – Experiment three parameters

Type	Variable	Value(s)
Producer	Job Min Size	20
	Job Max Size	240
	Job μ	20 to 250 increments of 5
Scheduler	Number of Schedulers	Layer 1 -> 1 Layer 2 -> 2
Consumer	Consumer Min units	10
	Consumer Max units	60
	Number of Consumers	26

In this experiment, the number of consumers is halved but the number of schedulers in the layer above kept the same. The aim of this experiment is to compare the performance of the hierarchy when there is a uniform hierarchy and a non-uniform hierarchy. Job generation will remain the same and as experiment two and the values compared will be the make span and the utilisation values.

4.5 Performance Metrics

1. **Overall Makespan:**

Makespan is the total time taken to completely process all jobs, i.e. where all the jobs have been processed completely. The makespan is calculated for separate task bins where bin 0 will contain the average task makespan for all the smallest tasks, bin 5 will contain the average task makespan for all the medium size tasks, and bin 10 will contain the average task makespan for all the largest tasks.

2. **Hierarchy Utilization:**

Hierarchy utilization is the usage (%) of all the Consumers in the hierarchy after all the jobs have been completely processed.

5 Results and Discussion

A. Experiment one

In experiment one, the aim was to build a non-uniform hierarchy where multiple load balancing algorithms will be used.

Table 9 - Experiment One Results

Algorithm	Makespan	Utilization
OPP + Max_min	224	66%
OPP + min_min	225	66%
OPP + FastTrack	322	46%
Max_min + FastTrack	226	66%
FastTrack + FastTrack	178	83%
Max_min + Max_Min	162	91%

It is visible from the table of results above that the best performing combination is Max_min+Max_min and OPP+Max_min coming a close second. The algorithm combination can properly utilize the hierarchy and distribute jobs appropriately given the non-uniform nature of the hierarchy and the uneven distribution of consumers with high executing capability. Below Figure 5 shows a visual representation of how the tasks have been distributed in the hierarchy for Max_min+Max_min.

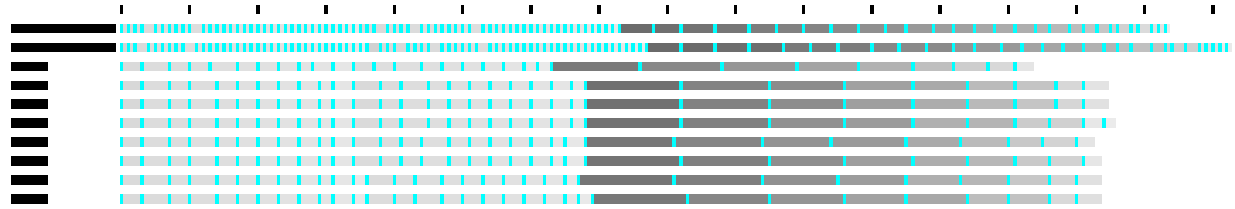

Figure 5 - Max_Min visual representation

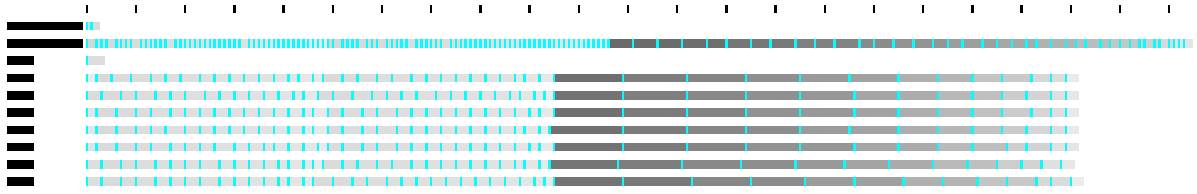

Figure 6 - OPP + Max_min visual representation of experiment

Figure 5 clearly shows the hierarchy being utilized efficiently irrespective of the hierarchy being non-uniform. It is important to note that the makespan can be lower if the execution ability of the consumers were higher. In addition, a striking result was the makespan value for the FastTrack+FastTrack combination. The expected makespan for the algorithm combination was very high because the FastTrack algorithm generally struggles when there are not enough consumers with high and low computational ability in a hierarchy as there is in this experiment. But in this experiment, the Fast Track algorithm can perform reasonably well. The FastTrack algorithm generally performs well with a consumer count of at least 20 in the hierarchy. Below Figure 7 shows that the FastTrack algorithm also utilizes the hierarchy efficiently. The

following experiment results should be able to point out FastTrack's strengths more adequately. Figure 6 also shows how badly jobs can be distributed and how underutilized the hierarchy can be when the number of consumers are small and the hierarchy is non-uniform.

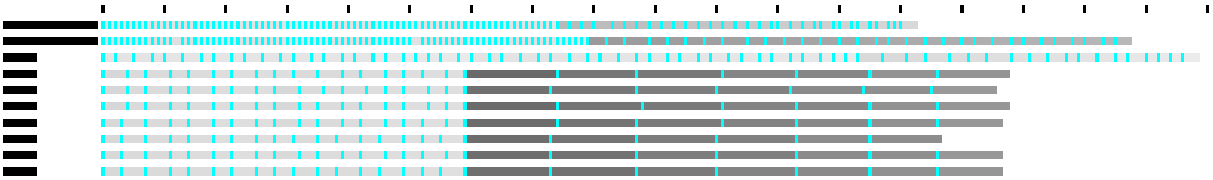

Figure 7 - FastTrack + FastTrack visual

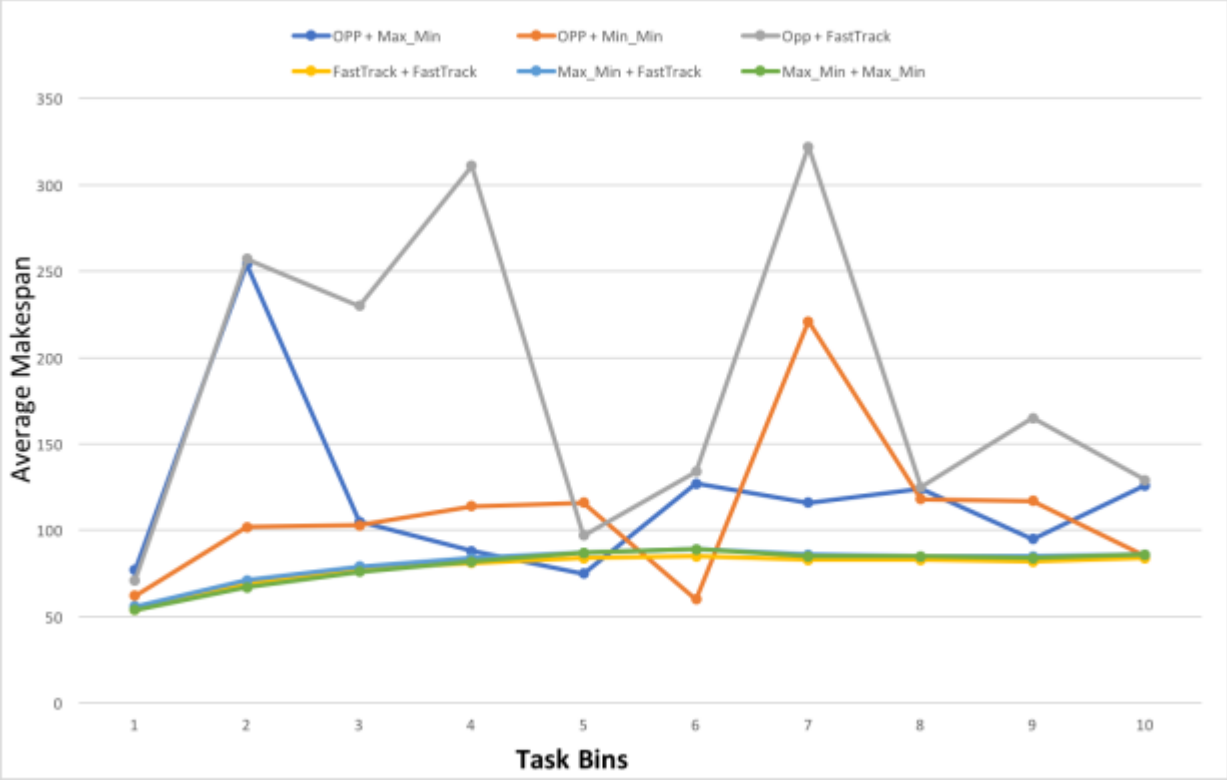

Figure 8 – Makespan for each task bin. Experiment 1

B. Experiment two

The aim in experiment two was to test the performance of the hierarchy where the computational ability of the consumers had been increased. The hierarchy tested was uniform and job generation was higher in terms of size and difficulty compared to the job generation in experiment one.

Table 10 - Experiment Two Results

Algorithm	Makespan	Utilization
OPP + Max_min	64	45%
OPP + min_min	65	45%
OPP + FastTrack	65	44%
Max_min + FastTrack	62	47%
FastTrack + FastTrack	63	46%

Max_Min + Max_Min	62	47%

The experiment results depicted in Table 10 are very similar but upon close analysis of the results, when the job size gets higher, there starts to be a clear difference between the performance of the algorithms see Figure 10. The Max_min algorithm seems to work very well regardless of the layer it operates in, and the algorithm it is paired with. The experiment was tested with starting job sizes of 170 units and then increased to 200 and 250 respectively. It was observed that the Max_min + FastTrack could keep its makespan value consistent as the job sizes increased, the algorithm was able to successfully do this by utilizing more of the resources of the hierarchy, this is shown in the table as it was able to use 47% of the hierarchy for job execution. It is also clear that if the job size was to increase further, the Max_min+FastTrack combination will be able to keep its makespan value lower for a longer period compared to other combinations. A possible contributing factor to the similarities of the results is the uniformity of the hierarchy, the consumers are distributed evenly between the schedulers and their collective capabilities within their parent schedulers are not far off each other at all. Essentially, each scheduler has the same number of consumers as well as consumers with enough computational power. If there is to be a more drastic increase in job quantity and size, there will be a clearer distinction between the performance of the different algorithms. Therefore, the same experiment will be tested in a non-uniform environment where the number of consumers under each scheduler is not the same and there is a distinct difference between their capabilities.

Figure 9 Shows a visual representation of the job distribution of the Max_min+FastTrack algorithm and how it manages to utilize the resources available to it.

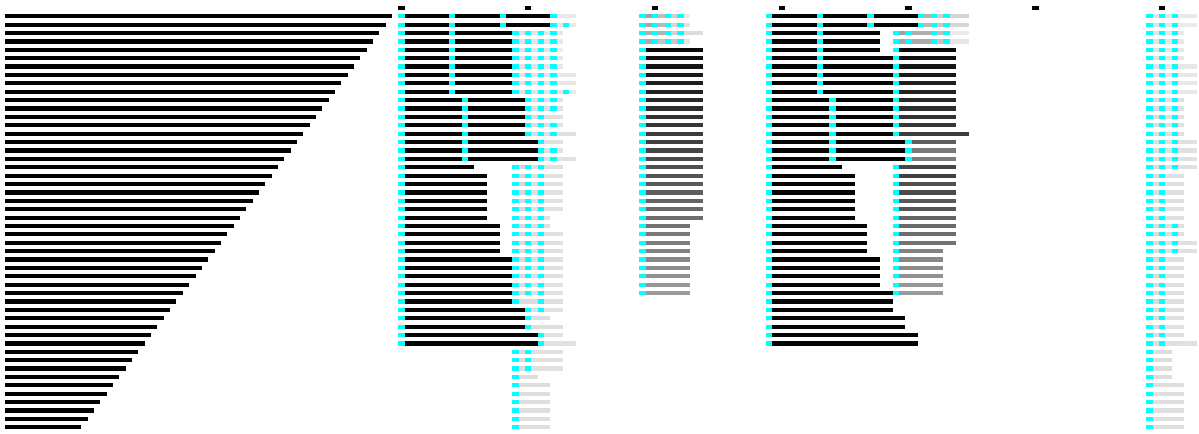

Figure 9 - Max_min + FastTrack Visual Representation

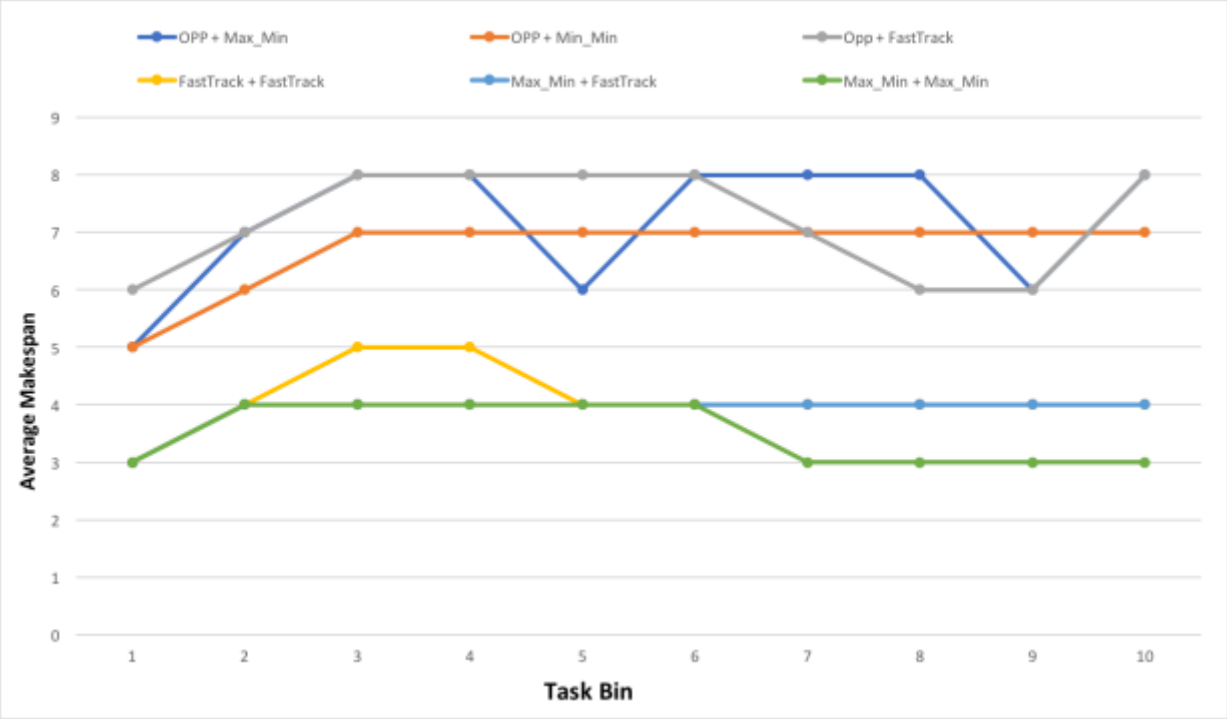

Figure 10 - Makespan for each task bin. Experiment 2

C. Experiment three

The aim of this experiment was to perform a comparison with the previous test (experiment two). The previous experiment was carried out with a uniform hierarchy where consumers were evenly distributed in size and capability between the schedulers. This experiment looks at the differences when the hierarchy is non-uniform.

Table 11 - Experiment Three Results

Algorithm	Makespan	Utilization
OPP + Max_min	82	69%
OPP + min_min	83	68%
OPP + FastTrack	94	60%
Max_min + FastTrack	82	70%
FastTrack + FastTrack	99	57%
Max_Min + Max_Min	64	88%

Table 11 shows a more varied difference in results that depicted in Table 10 of experiment two. It is observed that in this non-uniform hierarchy, the best performing algorithm combination is Max_Min+Max_Min with a makespan value of 64 and utilization value of 88%. The algorithm efficiently uses the hierarchy even though it is non-uniform. It is expected that in a cloud environment, most datacentres (schedulers) will not all have the same capabilities and such algorithms are needed to utilize the hierarchy efficiently to produce good service. One issue Datacentres will like to avoid is over utilization of resources which can cause bottlenecks. However, this can be countered by adding more consumers into the hierarchy, consumers are essentially virtual machines in a cloud environment and they are very easy to setup. Below Figure 11 shows a visual representation of how Max_Min+Max_min can utilize he hierarchy and efficiently distribute jobs to the appropriate consumers. Figure 12 also shows how

opportunistic can achieve high utilization and a reasonable makespan value which is second after Max_Min+Max_Min.

Figure 11 - Max_Min + Max_Min visual representation

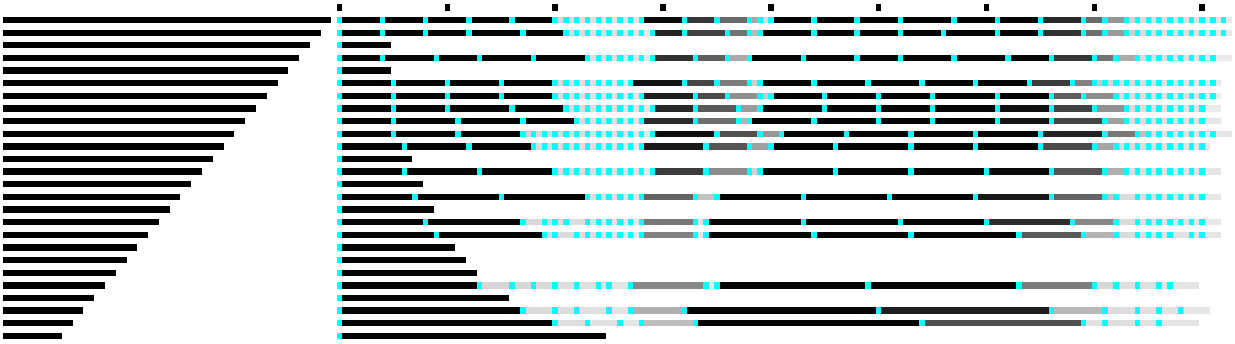

Figure 12 - Opportunistic + Max_Min

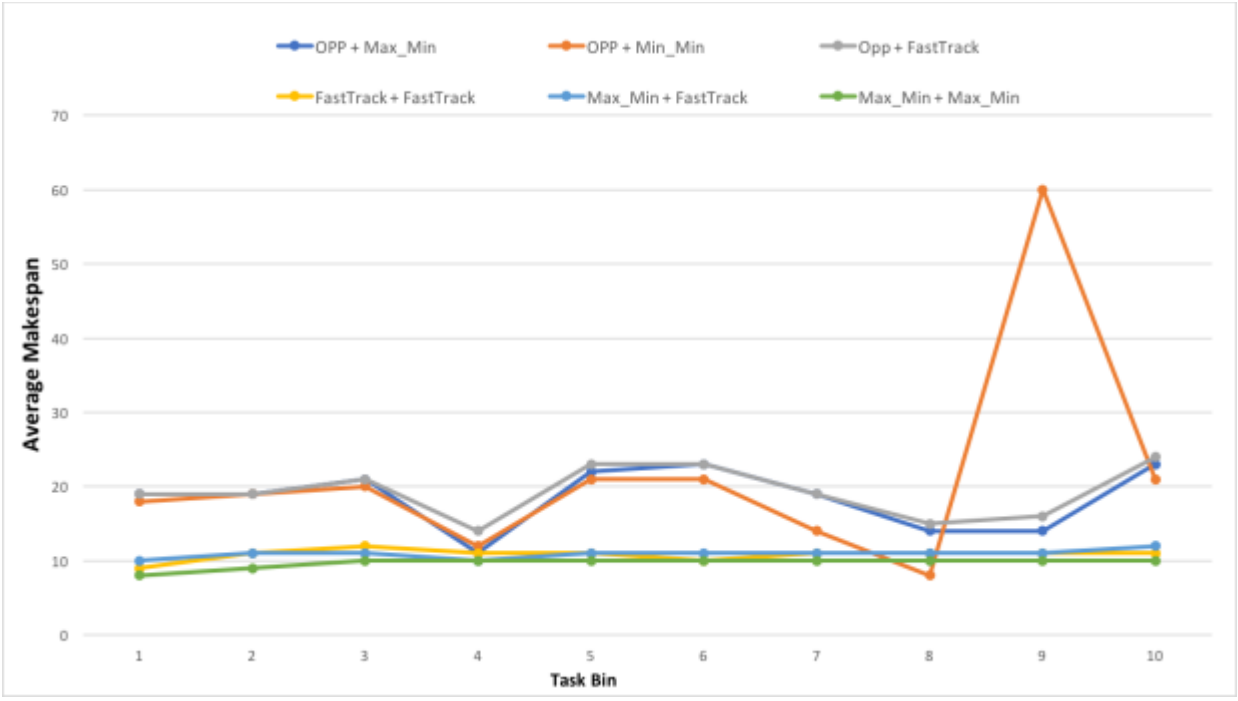

Figure 13 - Makespan for each task bin. Experiment 3

6 Conclusion and Future Work

A. Performance of Algorithm

The Max_Min algorithm performs exceptionally well in every scenario it was tested under, it was able keep job makespan quite low for every circumstance and when combined with other algorithms it performs well see Figure 8, Figure 10, and Figure 13. Algorithms to note that has performed well with Max_Min is Opportunistic and Max_Min_FastTrack algorithm. The Opportunistic algorithm has also proved to be a good service broker algorithm regardless of its flaws, it is an algorithm that can easily be improved to ensure the flaws do not hinder its performance. The Max_Min_Fast Track algorithm is not suitable for situations where the cloud environment only receives a small amount of jobs and the consumers available are not up to a threshold. An example of this result is the result from Table 9 where the Fast Track performs badly with the Opportunistic and Max_Min algorithm. Below Figure 14 and Figure 15 show how jobs were not so efficiently distributed among consumers in their respective hierarchies.

B. Drawback of opportunistic Algorithm

After analysing the results from each experiment, a drawback of the opportunistic algorithm was identified. The drawback was that if certain conditions were present, such as running the algorithm in a non-uniform hierarchy, i.e. in a hierarchy where the ability of the consumers under a scheduler is not similar to those of other schedulers. This problem can be seen in Figure 12 where the small consumers are also receiving big jobs to execute. This results to poor job distribution where there is allocation of jobs to schedulers with little computational power which will certainly cause a bottleneck. However, this effect is minimized because the opportunistic algorithm only runs on one layer and there is further load balancing after jobs are routed using the algorithm. Other issues can be countered by one of the following:

- Migration of jobs from consumers which are overloaded or are executing jobs too slowly. A reason for this can be seen in Figure 14 where the tenth consumer is overloaded with jobs and other consumers are either idle or executing jobs with small sizes e.g. the first and third consumer.
- A mechanism within the cloud system that picks the algorithms to use for each layer especially as the incoming jobs are supplied in batches.
    - This can be implemented by analysing the incoming jobs and setting a threshold for big jobs for each algorithm.

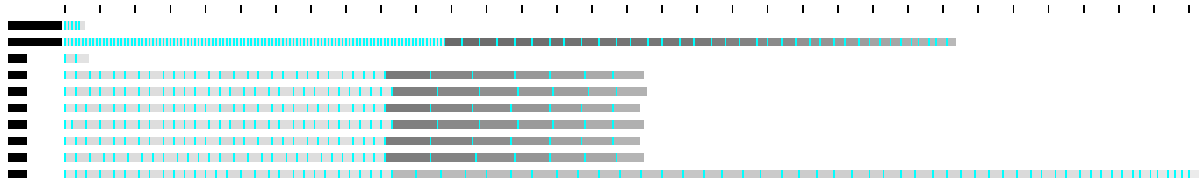

Figure 14 - Opp + Fast Track

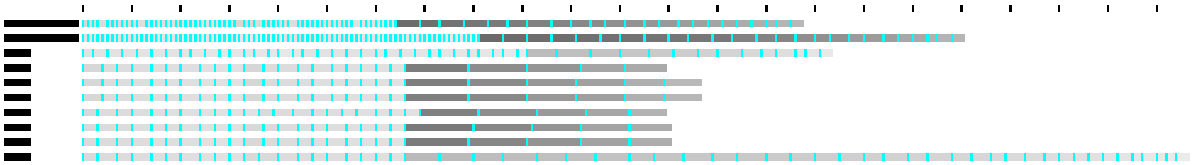

Figure 15 - Max_Min + FastTrack

C. Findings

The Max_min+Max_min algorithm combination is the best performing algorithm from all the experiments, it is able to attain low makespan values and utilize the consumers in the hierarchy efficiently. Table 10 and Table 11 are examples of how well the Max_min+Max_min combination has performed. The performance of the other algorithms has been overshadowed by the max-min combination but they have also performed very well with reasonable makespan values and utilization values.

On the other hand, regarding the performance of the algorithms with groups of different sized tasks, three algorithm combinations have produced the best results. The Max_min+Max_min, Max_min_FastTrack+Max_min-FastTrack, and Max_min+Max-min_FastTrack. Figure 10 and Figure 13 show how well these algorithms have performed with groups of large jobs. The other combinations have also performed well but are far off the best three in terms of grouped jobs.

Future Work

The future work of this project is to implement the stable marriage algorithm and evaluate its suitability in a cloud environment. In addition, another objective is to:

- Make use of more algorithms in the hierarchy where there can be job migration and child nodes can communicate more efficiently with their parent nodes by sending more information such as number of tasks executed, and the size of task to execution time ratio.
- Investigate the effect of migration of tasks and communication of nodes in the hierarchy and determine if the advantages overshadow the disadvantages. Regarding total efficiency, an algorithm that is fast and does not add a lot to the overall latency will be the most ideal but the added effects of job migration and communication within each tree can slow down the whole execution process.

7 Project Evaluation

Evaluation of Project

The project has been very challenging especially with the limited amount of time available to complete it. A concrete understanding of the cloud computing paradigm has been gained, and the importance of efficient algorithms has been understood. Conducting the literature review was not easy because of the vast amount of information on cloud computing, there was a lot of information on cloud computing and it was difficult to find literature for this project specifically. Not a lot of work had been done on hierarchical load balancing in cloud computing, regardless of the small amount of literature available on the project topic, information was gathered by reading relevant journals and articles which was then used as a foundation of this project.

Secondly, understanding the simulator to be used in this project was a worthwhile task. The simulator originally intended for use in this project was CloudSim. After trying to understand how the simulator could be used and evaluating the output needed from the simulator, attention was shifted elsewhere to other simulators. The CloudAnalyst simulator which is built on top CloudSim was next to be used but after running into some problems described in chapter 3, the ScheduleSim simulator which provided a simple design and provided output suitable for the project was used. Understanding the workings of the ScheduleSim simulator and how to implement further algorithms into it took most of the time allocated but was very crucial to the success and any future work of this project.

Lastly, regarding time management, this project has been time consuming and has required strict time allocation and discipline to finish objectives that were set at the beginning. The most time-consuming elements of the project has been the literature review and gaining an understanding of the simulator to be used and how to use it efficiently. However, with the difficulty of conducting the literature review, understanding the simulator, designing experiments, and implementing the algorithms to be tested the objectives set have been able to be achieved except for the advanced objectives which has now been included in the future work (see Chapter 6).

The table below shows what objectives have been successfully achieved, and the comments about their completion.

Objective (Normal and Advanced)		Comment
Conduct Literature Review on the areas of cloud computing such as load balancing algorithms, scheduling, and structure of cloud.	✓	N/A
Design Hierarchical Approach.	✓	N/A
Have a clear understanding of the ScheduleSim simulator and why it is so useful.	✓	N/A
Implement Algorithms for the different layers of the hierarchy.	✓	N/A

Implement Opportunistic service broker algorithm	✓	N/A
Evaluate usage of algorithms.	✓	N/A
Discover which Algorithm combination yields the best results.	✓	N/A
Implement Stable marriage algorithm.	✗	In future work
Evaluate feasibility and suitability of stable marriage algorithm the Load Balancing Domain with the result of the implementation.	✗	In future work

8 References

Daryapurkar, A. and Deshmukh, M.V., 2013. Efficient load balancing algorithm in cloud environment. *International Journal Of Computer Science And Applications*, *6*(2), pp.308-312.

Kaur, R. and Luthra, P., 2012, December. Load balancing in cloud computing. In *Second Symposium on Cloud computing*.

Kaur, R. and Luthra, P., 2014. Load Balancing in Cloud System using Max-min and Min-min Algorithm. *International Journal of Computer Applications (0975–8887)*.

Laxmi, V. and Kaur, N., 2012. Batch Mode Scheduling-Mid_Max Algorithm. *International Journal of Computer Applications*, *49*(15).

Mell, P. and Grance, T., 2011. The NIST definition of cloud computing.

Moggridge, P., Helian, N., Sun, Y., Lilley, M., (2017) 'Revising Max-Min for Scheduling in a Cloud Computing Context'. *School of Computer Science*.

Nagaty, K.A., 2014. Cloud Tree: A Hierarchical Organization as a Platform for Cloud Computing. *International Journal of Computer and Electrical Engineering*, *6*(1), p.16.

Nandagopal, M. and Uthariaraj, R.V., 2010. Hierarchical load balancing approach in computational grid environment. *International Journal of Recent Trends in Engineering and Technology*, *3*(1), pp.19-24.

Pilla, L. L., Ribeiro, C. P., Cordeiro, D., Mei, C., Bhatele, A., Navaux, P. O. A., Broquedis, F., Méhaut, J. F. and Kale, L. V. (2012) 'A hierarchical approach for load balancing on parallel multi-core systems', in *Proceedings of the International Conference on Parallel Processing*, pp. 118–127. doi: 10.1109/ICPP.2012.9.

Salot, P., 2013. A survey of various scheduling algorithm in cloud computing environment. *International Journal of Research in Engineering and Technology*, *2*(2), pp.131-135.

Singh, R.M., Paul, S. and Kumar, A., 2014. Task scheduling in cloud computing: Review. *International Journal of Computer Science and Information Technologies*, *5*(6), pp.7940-7944.

Malarvizhi, N. and Uthariaraj, V. R. (2009) 'Hierarchical load balancing scheme for computational intensive jobs in grid computing environment', in *2009 1st International Conference on Advanced Computing, ICAC 2009*, pp. 97–104. doi: 10.1109/ICADVC.2009.5378268.

Pilla, L. L., Ribeiro, C. P., Cordeiro, D., Mei, C., Bhatele, A., Navaux, P. O. A., Broquedis, F., Méhaut, J. F. and Kale, L. V. (2012) 'A hierarchical approach for load balancing on parallel multi-core systems', in *Proceedings of the International Conference on Parallel Processing*, pp. 118–127. doi: 10.1109/ICPP.2012.9.

Wang, S. C., Yan, K. Q., Wang, S. S. and Chen, C. W. (2011) 'A three-phases scheduling in a hierarchical cloud computing network', in *Proceedings - 2011 3rd International Conference on Communications and Mobile Computing, CMC 2011*, pp. 114–117. doi: 10.1109/CMC.2011.28.

Zheng, G., Bhatelé, A., Meneses, E. and Kalé, L. V. (2011) 'Periodic hierarchical load balancing for large supercomputers', *International Journal of High Performance Computing Applications*, 25(4), pp. 371–385. doi: 10.1177/1094342010394383.

9 Bibliography

Dave, S. and Maheta, P., 2014. Utilizing Round Robin Concept for Load Balancing Algorithm at Virtual Machine Level in Cloud Environment. *International Journal of Computer Applications*, *94*(4).

Kanani, B. and Maniyar, B., 2015. Review on Max-Min Task scheduling Algorithm for Cloud Computing. *Journal of Emerging Technologies and Innovative Research*, *2*(3).

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

Sran, N. and Kaur, N., 2013. Comparative analysis of existing load balancing techniques in cloud computing. *International Journal of Engineering Science Invention*, *2*(1), pp.60-63.

10 Appendix

10.1 APPENDIX I – OPPORTUNISTIC ALGORITHM

```
package schedulesim.scheduler;
/**
 *
 * @author Samuel Omotayo
 */

import java.util.*;
import schedulesim.Consumer;
import schedulesim.ConsumingEntity;
import schedulesim.SimEntity;
import schedulesim.Task;
import schedulesim.Scheduler;
import schedulesim.scheduler.*;

public class Opportunistic extends Scheduler{
   //HashMap to use to determine which child gets allocated job
   private HashMap<ConsumingEntity, Integer> results = new HashMap<>();
   private HashMap<ConsumingEntity, Double> delay = new HashMap<>();
   private int i =0, min_util, value = 0, count = 0;
   private Map.Entry entry;
   private ConsumingEntity child;
   private Task task;
   private SimEntity scheduler, decision;
   private boolean check = true;
   private double updatedDelay;
   private ConsumingEntity bestScheduler;


   public Opportunistic(){
      super();
   }

   @Override
   public void step(){
      super.step();
      if (count == 0){//round robin one task to all consumers
         for (ConsumingEntity entity: super.getChildren()){
            for (ConsumingEntity consumer: entity.getChildren()){
               consumer.submitTask(super.getWaitingTasks().remove(0));
            }
         }
         for (ConsumingEntity entity: super.getChildren()){
            for (ConsumingEntity consumer: entity.getChildren()){
               delay.put(consumer, consumer.getDelay());
            }
         }
         count++;
      }
      if (super.getChildren().size() > 0){ //as long as there are children run this code block

         for (ConsumingEntity sch: super.getChildren()){
            for (ConsumingEntity consume: sch.getChildren()){
               results.put(consume,consume.getUnitsPerStep());// units per step of all consumers will be here.
            }
         }

         //choose scheduler to allocate job
         while (super.getWaitingTasks().size()>0){
            task = super.getWaitingTasks().get(0);

            Set set = results.entrySet();
            Iterator iterator = set.iterator();

            while (iterator.hasNext()){//iterate amount of units per step per map
```

```
                entry = (Map.Entry)iterator.next();
                child = (ConsumingEntity)entry.getKey();//consumer is here
                scheduler = child.getParent(); //parent is here

                updatedDelay = task.getRemaingUnits() - ((double)child.getUnitsPerStep() + (double)delay.get(child));
                min_util = (int)updatedDelay;

                if (min_util < value || check ){
                   value = min_util;
                   check = false;
                   decision = scheduler; //scheduler with least delay is chosen
                }
                delay.put(child, (double)min_util);//update delays
             }

             bestScheduler = (ConsumingEntity)decision;
             bestScheduler.submitTask(super.getWaitingTasks().remove(0));
          }
       }
    }
}
```

10.2 APPENDIX II – MAX-MIN CODE

```
package schedulesim.scheduler;

import java.util.Collections;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import schedulesim.ConsumingEntity;
import schedulesim.Task;
import schedulesim.TaskMaxFirstComparator;
import schedulesim.Scheduler;
//line below added by tosin
import java.util.ArrayList;

/**
 *
 * @author Paul
 */
public class MaxminScheduler extends Scheduler {
    public HashMap<ConsumingEntity, Integer> list =new HashMap<>();

    public MaxminScheduler(){
        super();
    }

    @Override
    public void step() {
        super.step();
        if(super.getChildren().size()>0){

            // This will store the delays on children below.
            HashMap<ConsumingEntity, Double> childDelay = new HashMap<>();


            // Find any exsisting delay on child from previous waves
            for (ConsumingEntity child : super.getChildren()) {
                // Create entry for child
                childDelay.put(child, child.getDelay());
            }

            // Sort the task max first, biggest tasks first
            Collections.sort(super.getWaitingTasks(), new TaskMaxFirstComparator());

            while(super.getWaitingTasks().size()>0){
                Task task = super.getWaitingTasks().get(0);


                // Which ConsumingEntity can finish it first? i.e in the min. time,
                // takes into account the UnitPerStep of the ConsumingEntity and tasks
                // already scheduled to it.
                double minChildDelay = 0.0;
                ConsumingEntity minChild = null;

                // Which child can complete task first
                for(ConsumingEntity child : super.getChildren()){
                    double taskMakespanWithChildDelay = ((double)task.getRemaingUnits() / (double)child.getUnitsPerStep()) +
childDelay.get(child);
                    // Can this child finish the task faster
                    if(taskMakespanWithChildDelay < minChildDelay || minChild == null){
                        minChild = child;
                        minChildDelay = taskMakespanWithChildDelay;
                    }
                }

                // Update child delays map
                childDelay.put(minChild, minChildDelay);

                // Submit task to child
```

```
                minChild.submitTask(super.getWaitingTasks().remove(0));
            }
        }
    }
}
```

10.3 APPENDIX III – MIN-MIN CODE

```
package schedulesim.scheduler;

import java.util.Collections;
import java.util.HashMap;
import schedulesim.ConsumingEntity;
import schedulesim.Task;
import schedulesim.TaskMinFirstComparator;
import schedulesim.Log;
import schedulesim.Producer;
import schedulesim.Scheduler;

/**
 *
 * @author Paul
 */
public class MinminScheduler extends Scheduler {

    public HashMap<ConsumingEntity, Integer> list =new HashMap<>();

    public MinminScheduler(){
        super();
    }

    @Override
    public void step() {
        super.step();
        if(super.getChildren().size()>0){

            // This will store the delays on children below.
            HashMap<ConsumingEntity, Double> childDelay = new HashMap<>();

            // Find any exsisting delay on child from previous waves
            for (ConsumingEntity child : super.getChildren()) {
                // Create entry for child
                childDelay.put(child, child.getDelay());
            }

            // Sort the task min first, smallest tasks first
            Collections.sort(super.getWaitingTasks(), new TaskMinFirstComparator());

            while(super.getWaitingTasks().size()>0){
                Task task = super.getWaitingTasks().get(0);

                // Which ConsumingEntity can finish it first? i.e in the min. time,
                // takes into account the UnitPerStep of the ConsumingEntity and tasks
                // already scheduled to it.
                double minChildDelay = 0.0;
                ConsumingEntity minChild = null;

                // Which child can complete task first
                for(ConsumingEntity child : super.getChildren()){
                    double taskMakespanWithChildDelay = ((double)task.getRemaingUnits() / (double)child.getUnitsPerStep()) +
childDelay.get(child);
                    // Can this child finish the task faster
                    if(taskMakespanWithChildDelay < minChildDelay || minChild == null){
                        minChild = child;
                        minChildDelay = taskMakespanWithChildDelay;
                    }
                }

                // Update child delays map
                childDelay.put(minChild, minChildDelay);

                // Submit task to child
                minChild.submitTask(super.getWaitingTasks().remove(0));
            }
        }
    }
```

```
}
```

10.4 APPENDIX IV – MAX MIN FAST TRACK ALGORITHM

```
package schedulesim.scheduler;

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import schedulesim.ConsumingEntity;
import schedulesim.ConsumingEntityMinFirstComparator;
import schedulesim.Task;
import schedulesim.TaskMaxFirstComparator;
import schedulesim.Log;
import schedulesim.Scheduler;

/**
 *
 * @author Paul
 */
public class MaxminFastTrackScheduler extends Scheduler {

  private float margin;
  //added by sam
  public HashMap<ConsumingEntity, Integer> list =new HashMap<>();


  public MaxminFastTrackScheduler(){
    super();
    margin = 0.4f;
  }

  @Override
  public void step() {
    super.step();
    if(super.getChildren().size() > 0 && super.getWaitingTasks().size() > 0){

      // This will store the delays on children below.
      HashMap<ConsumingEntity, Double> childDelay = new HashMap<>();

      // Find any exsisting delay on child from previous waves
      for (ConsumingEntity child : super.getChildren()) {

        // Create entry for child
        childDelay.put(child, child.getDelay());
      }

      // Sort the task max first, smallest tasks first. The task that come first here will
      // be palced in the normal track
      Collections.sort(super.getWaitingTasks(), new TaskMaxFirstComparator());

      // Decicde which tasks are small and eligible for fast tracking using margin
      // if fast track margin 0.3, this means the smallest 30% of tasks are fast
      // tracked the rest, 70% are processed on the normal track.
      ArrayList<Task> normalTrackTasks = new ArrayList<>();
      ArrayList<Task> fastTrackTasks = new ArrayList<>();
      int taskUnitsInNormalTrack = 0; // The amount of work in the normal track
      int taskUnitsInFastTrack = 0;  // The amount of work in the fast track

      // Number of tasks to go in Normal
      int countTasksToNormalTrack = Math.round(super.getWaitingTasks().size() * (1.0f-margin));

      // Using the sorted task list (biggest length first), put the biggest tasks into the normal track,
      while(super.getWaitingTasks().size() > 0){
        if(normalTrackTasks.size() <= countTasksToNormalTrack){
          taskUnitsInNormalTrack += super.getWaitingTasks().get(0).getRemaingUnits();
          normalTrackTasks.add(super.getWaitingTasks().remove(0));
        } else {
          taskUnitsInFastTrack += super.getWaitingTasks().get(0).getRemaingUnits();
          fastTrackTasks.add(super.getWaitingTasks().remove(0));
        }
      }
```

```
    if (super.getWaitingTasks().size() != 0) {
      Log.println("Error, all tasks should have been assigned a track in the above");
    }

//     Log.println("Fast Track Tasks: ");
//     for(Task task : fastTrackTasks){
//       Log.println(task.getTid() + "," + task.getRemaingUnits() + "u");
//     }
//     Log.println("Normal Track Tasks: ");
//     for(Task task : normalTrackTasks){
//       Log.println(task.getTid() + "," + task.getRemaingUnits() + "u");
//     }

    // Count total Unit Per Step
    int totalUPS = 0;
    for(ConsumingEntity child : super.getChildren()){
      totalUPS+= child.getUnitsPerStep();
    }

    // Based on the distribution of fast trackable task units to normal task units
    // calculate percentages of consumer units per step for the normal and fast tracks.
    double consumerUPSToTaskUPS = totalUPS / ((double) taskUnitsInNormalTrack + (double) taskUnitsInFastTrack);
    // Below is used later to decide how many UPS to put in the fast track
    double fastTrackTargetMips = consumerUPSToTaskUPS * taskUnitsInFastTrack;

    // Sort the child fastest (min. time) first (for building the FastTrack) i.e.
    // The ConsumingEntities with biggest UPS go first and thus into the fast track
    Collections.sort(super.getChildren(), new ConsumingEntityMinFirstComparator());

    // Assign VMs to fast or normal track
    ArrayList<ConsumingEntity> copyChildren = new ArrayList<>(super.getChildren());
    ArrayList<ConsumingEntity> fastTrack = createFastTrack(copyChildren, fastTrackTargetMips);
    ArrayList<ConsumingEntity> normalTrack = createNormalTrack(copyChildren);

    if (copyChildren.size() > 0) {
      Log.println(copyChildren.size() + " Consuming Entities where not assigned a track.");
    }

    // MaxMin tasks onto FastTrack
    Collections.sort(fastTrackTasks , new TaskMaxFirstComparator());

    while(fastTrackTasks.size()>0){
      Task task = fastTrackTasks.get(0);

      // Which ConsumingEntity can finish it first? i.e in the min. time,
      // takes into account the UnitPerStep of the ConsumingEntity and tasks
      // already scheduled to it.
      double minChildDelay = 0.0;
      ConsumingEntity minChild = null;

      // Which child can complete task first
      for(ConsumingEntity child : fastTrack){
        double taskMakespanWithChildDelay = ((double)task.getRemaingUnits() / (double)child.getUnitsPerStep()) +
childDelay.get(child);
        // Can this child finish the task faster
        if(taskMakespanWithChildDelay < minChildDelay || minChild == null){
          minChild = child;
          minChildDelay = taskMakespanWithChildDelay;
        }
      }

      // Update child delays map
      childDelay.put(minChild, minChildDelay);

      // Submit task to child
      minChild.submitTask(fastTrackTasks.remove(0));
    }

    // MaxMin tasks onto NormalTrack
    Collections.sort(normalTrackTasks , new TaskMaxFirstComparator());
```

```
      while(normalTrackTasks.size()>0){
        Task task = normalTrackTasks.get(0);

        // Which ConsumingEntity can finish it first? i.e in the min. time,
        // takes into account the UnitPerStep of the ConsumingEntity and tasks
        // already scheduled to it.
        double minChildDelay = 0.0;
        ConsumingEntity minChild = null;

        // Which child can complete task first
        for(ConsumingEntity child : normalTrack){
          double taskMakespanWithChildDelay = ((double)task.getRemaingUnits() / (double)child.getUnitsPerStep()) +
childDelay.get(child);
          // Can this child finish the task faster
          if(taskMakespanWithChildDelay < minChildDelay || minChild == null){
            minChild = child;
            minChildDelay = taskMakespanWithChildDelay;
          }
        }

        // Update child delays map
        childDelay.put(minChild, minChildDelay);

        // Submit task to child
        /*for (ConsumingEntity ent: super.getChildren()){
            System.out.println(ent.getDelay());
        }*/

        minChild.submitTask(normalTrackTasks.remove(0));
      }
    }
  }

  public ArrayList<ConsumingEntity> createFastTrack(ArrayList<ConsumingEntity> allConsumingEntities, double
fastTrackTargetUPS){
    ArrayList<ConsumingEntity> fastTrack = new ArrayList<>();
    int fastTrackUPS = 0;

    // Loop over biggest ConsumingEntities first, if it fits in the fast track place in there
    while((fastTrackTargetUPS - fastTrackUPS) > allConsumingEntities.get(0).getUnitsPerStep()) {
      // Add VM to fast track
      fastTrackUPS += allConsumingEntities.get(0).getUnitsPerStep();
      fastTrack.add(allConsumingEntities.remove(0));
    }


    // Check we have at least one VM in fast track
    if (fastTrack.size() <= 0) {
      // If we not, add one smallest VM
      fastTrackUPS += allConsumingEntities.get(allConsumingEntities.size() - 1).getUnitsPerStep();
      fastTrack.add(allConsumingEntities.remove(allConsumingEntities.size() - 1));
    }

    return fastTrack;
  }

  public ArrayList<ConsumingEntity> createNormalTrack(ArrayList<ConsumingEntity> remainingConsumingEntities){

    // Create the NormalTrack and FastTrack as close to the target MIPS as possible
    ArrayList<ConsumingEntity> normalTrack = new ArrayList<>();

    // Add all remaining VMs
    while (remainingConsumingEntities.size() > 0) {
      normalTrack.add(remainingConsumingEntities.remove(0));
    }

    return normalTrack;
  }

}
```

10.5 APPENDIX V – Experiment one code

```
private static boolean expr_1() {
// Create simulation
// expr 1
for (int i =1; i <=10; i++){
ScheduleSim sim = new ScheduleSim();

// Create Producer, this will dispatch waves of jobs
Producer producer = new Producer("Opp");

// Create Job Pattern(s), this will create the wave of jobs. There several
int taskCount = 20;
int taskSize = 16;
FlatPattern flat = new FlatPattern(taskCount, taskSize);

int startSize = 20;
int endSize = 150;
double mu = 11;
double sigma = 4;
int combinedTargetSize = 5000;

GaussianPattern gaus = new GaussianPattern(startSize,endSize,mu,sigma, combinedTargetSize);

//incrementing pattern
int start = 20;
int end = 150;
IncrementingPattern incrementing = new IncrementingPattern(start,end);

producer.addMetatask(1,gaus);
producer.addMetatask(10,gaus);
producer.addMetatask(20, incrementing);


// Create Architecture
Architecture architecture = new Architecture("Min_Min");

// Create Scheduler(s)
Opportunistic robinTopScheduler = new Opportunistic();
MinminScheduler robinSubOneScheduler = new MinminScheduler();
MinminScheduler robinSubTwoScheduler = new MinminScheduler();

// Create Consumer(s)
Consumer[] consumers = new Consumer[10];
consumers[0] = new Consumer(10); // A consumer with a speed of 10 units per step
consumers[1] = new Consumer(30);
consumers[2] = new Consumer(10);
consumers[3] = new Consumer(10);
consumers[4] = new Consumer(10);
consumers[5] = new Consumer(10);
consumers[6] = new Consumer(10);
consumers[7] = new Consumer(30);
consumers[8] = new Consumer(10);
consumers[9] = new Consumer(10);

// Build Architecture Tree
try {
// Build up tree
// producer > robinTopScheduler |> robinSubOneScheduler |> consumers[0]
//                              |                       |> consumers[1]
//                              |
//                              |> robinSubTwoScheduler |> consumers[2]
//                                                      |> consumers[3]
//                                                      |> consumers[4]
//                                                      |> consumers[5]
//                                                      |> consumers[6]
//                                                      |> consumers[7]
//                                                      |> consumers[8]
//                                                      |> consumers[9]
architecture.addEntity(producer, robinTopScheduler);
```

```
architecture.addEntity(robinTopScheduler, robinSubOneScheduler);
architecture.addEntity(robinTopScheduler, robinSubTwoScheduler);
architecture.addEntity(robinSubOneScheduler, consumers[0]);
architecture.addEntity(robinSubOneScheduler, consumers[1]);
architecture.addEntity(robinSubTwoScheduler, consumers[2]);
architecture.addEntity(robinSubTwoScheduler, consumers[3]);
architecture.addEntity(robinSubTwoScheduler, consumers[4]);
architecture.addEntity(robinSubTwoScheduler, consumers[5]);
architecture.addEntity(robinSubTwoScheduler, consumers[6]);
architecture.addEntity(robinSubTwoScheduler, consumers[7]);
architecture.addEntity(robinSubTwoScheduler, consumers[8]);
architecture.addEntity(robinSubTwoScheduler, consumers[9]);

} catch (BadParentException bpe) {
Log.println(bpe.getMessage());
Log.println("Parent Id: " + bpe.getParent().getId());
Log.println("Child Id: " + bpe.getChild().getId());
}

sim.addExperiment(architecture);

// Define what output
OutputOptions outputOptions = new OutputOptions(); // default is makespan and utilisation
outputOptions.setCountBins(i); // Display job makespans group on 10 bins of job size
outputOptions.setRenderSchedule(true); // Render schedule images
outputOptions.setMakespan(true);
outputOptions.setUtilisation(true);

// Run the experiment
try {
sim.runExperiments(outputOptions);
} catch (BadStepsException bse) {
Log.println(bse.getMessage() + " SimEntity id:" + bse.getSimEntity().getId());
} catch (BadTaskCompletionException bjce) {
Log.println(bjce.getMessage() + " Sent:" + bjce.getTasksSent() + " Completed:" + bjce.getTasksCompleted());
}
System.out.println("\n");

}//for
return true;
}
```

10.6 APPENDIX VI – Experiment two code

```
private static boolean expr_2(){
// Create simulator
for (int x =1; x <=10; x++){
ScheduleSim sim = new ScheduleSim();

// Create Producer
Producer producer = new Producer("Opp");

// Create Task Pattern
int start = 20;
int stop = 250;
IncrementingPattern pattern = new IncrementingPattern(start, stop);

int taskCount = 60;
int taskSize = 250;
FlatPattern flat = new FlatPattern(taskCount, taskSize);

int startSize = 20;
int endSize = 250;
double mu = 11;
double sigma = 4;
int combinedTargetSize = 5000;

GaussianPattern gaus = new GaussianPattern(startSize,endSize,mu,sigma, combinedTargetSize);

producer.addMetatask(1, flat);
producer.addMetatask(10, gaus);
producer.addMetatask(20,pattern);
producer.addMetatask(30, flat);
producer.addMetatask(40, pattern);
producer.addMetatask(60, gaus);


// Create Architecture
Architecture architecture = new Architecture("Min");

// Create Scheduler
Opportunistic broker = new Opportunistic();
MinminScheduler scheduler1 = new MinminScheduler();
MinminScheduler scheduler2 = new MinminScheduler();

// Create Consumers
ArrayList<Consumer> consumers = new ArrayList<>();
for (int i = 60; i > 10; i--) {//create consumers with minimum value of 10 and max 50
consumers.add(new Consumer(i));//done this to ensure that every test will have the same values for the consumers
//consumers.add(new Consumer(rand.nextInt(60-10)+10));
}

// Build Architecture Tree
try {
architecture.addEntity(producer, broker);
architecture.addEntity(broker, scheduler1);
architecture.addEntity(broker, scheduler2);
int c =0;
while(c < consumers.size()){//equally distribute consumers

architecture.addEntity(scheduler1, consumers.get(c));
c++;

architecture.addEntity(scheduler2, consumers.get(c));
c++;
}

} catch (BadParentException bpe) {
System.out.println(bpe.getMessage());
System.out.println("Parent Id: " + bpe.getParent().getId());
System.out.println("Child Id: " + bpe.getChild().getId());
return false;
```

```
}

// Give experiment to ScheduleSim
sim.addExperiment(architecture);

// Chose output options
OutputOptions outputOptions = new OutputOptions(); // default is makespan and utilisation
outputOptions.setCountBins(x); // Display job makespans group on 10 bins of job size
outputOptions.setRenderSchedule(true); // Render schedule images
outputOptions.setMakespan(true);
outputOptions.setUtilisation(true);

// Run the experiment
try {
sim.runExperiments(outputOptions);
} catch (BadStepsException bse) {
Log.println(bse.getMessage() + " SimEntity id:" + bse.getSimEntity().getId());
} catch (BadTaskCompletionException bjce) {
Log.println(bjce.getMessage() + " Sent:" + bjce.getTasksSent() + " Completed:" + bjce.getTasksCompleted());
}
System.out.println("\n");
}
return true;

}
```

10.7 APPENDIX VII – Experiment three code

```
private static boolean expr_3(){
// Create simulator
for (int x =1; x <=10; x++){
ScheduleSim sim = new ScheduleSim();

// Create Producer
Producer producer = new Producer("Opp");

int start = 20;
int stop = 250;
IncrementingPattern pattern = new IncrementingPattern(start, stop);

int taskCount = 60;
int taskSize = 250;
FlatPattern flat = new FlatPattern(taskCount, taskSize);

int startSize = 20;
int endSize = 250;
double mu = 11;
double sigma = 4;
int combinedTargetSize = 5000;

GaussianPattern gaus = new GaussianPattern(startSize,endSize,mu,sigma, combinedTargetSize);

producer.addMetatask(1, flat);
producer.addMetatask(10, gaus);
producer.addMetatask(20,pattern);
producer.addMetatask(30, flat);
producer.addMetatask(40, pattern);
producer.addMetatask(60, gaus);


// Create Architecture
Architecture architecture = new Architecture("Min_Min");

// Create Scheduler
Opportunistic broker = new Opportunistic();
MinminScheduler scheduler1 = new MinminScheduler();
MinminScheduler scheduler2 = new MinminScheduler();
//MaxminFastTrackScheduler scheduler3 = new MaxminFastTrackScheduler();

Random rand = new Random();
// Create Consumers
ArrayList<Consumer> consumers = new ArrayList<>();
for (int i = 60; i >= 10; i-=2) {//create consumers with minimum value of 10 and max 60
consumers.add(new Consumer(i));//done this to ensure that every test will have the same values for the consumers
}
//System.out.println("size is " + consumers.size());

int [] one = {0,1,3,5,6,7,8,9,10,12,14,16,17,21,23,24}; // indexes for picking consumer out of array list
int [] two = {2,4,11,13,15,18,19,20,22,25};

// Build Architecture Tree
try {
architecture.addEntity(producer, broker);
architecture.addEntity(broker, scheduler1);
architecture.addEntity(broker, scheduler2);
//architecture.addEntity(broker, scheduler3);
for (int i = 0; i <one.length; i++){
architecture.addEntity(scheduler1,consumers.get(one[i]));
}
for (int i =0; i< two.length; i++){
architecture.addEntity(scheduler2,consumers.get(two[i]));
}


} catch (BadParentException bpe) {
System.out.println(bpe.getMessage());
```

```
System.out.println("Parent Id: " + bpe.getParent().getId());
System.out.println("Child Id: " + bpe.getChild().getId());
return false;
}

// Give experiment to ScheduleSim
sim.addExperiment(architecture);

// Chose output options
OutputOptions outputOptions = new OutputOptions(); // default is makespan and utilisation
outputOptions.setCountBins(x); // Display job makespans group on 10 bins of job size
outputOptions.setRenderSchedule(true); // Render schedule images
outputOptions.setMakespan(true);
outputOptions.setUtilisation(true);

// Run the experiment
try {
sim.runExperiments(outputOptions);
} catch (BadStepsException bse) {
Log.println(bse.getMessage() + " SimEntity id:" + bse.getSimEntity().getId());
} catch (BadTaskCompletionException bjce) {
Log.println(bjce.getMessage() + " Sent:" + bjce.getTasksSent() + " Completed:" + bjce.getTasksCompleted());
}
System.out.println("\n");
}

return true;
}
```

10.8 APPENDIX VIII – Experiment 1 visuals and Results

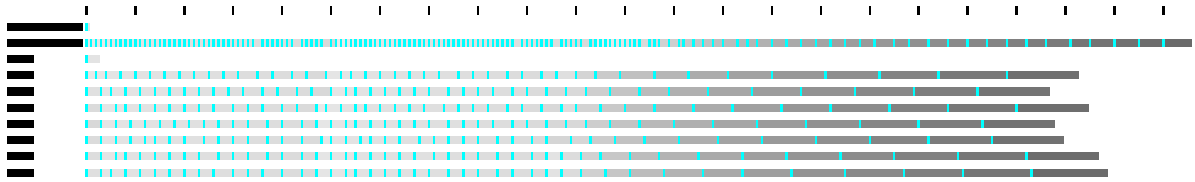

Figure 16 - Opp+min_min

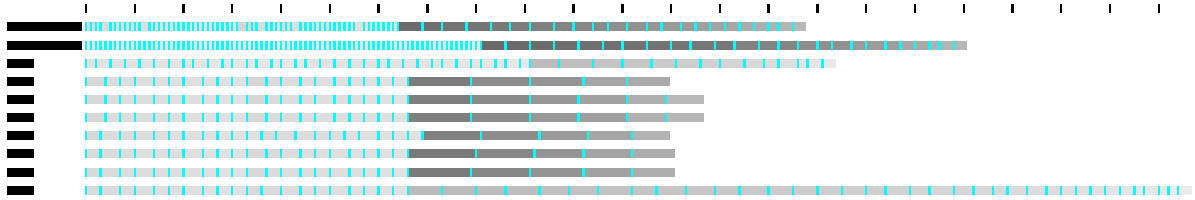

Figure 17 – Max_min+FastTrack

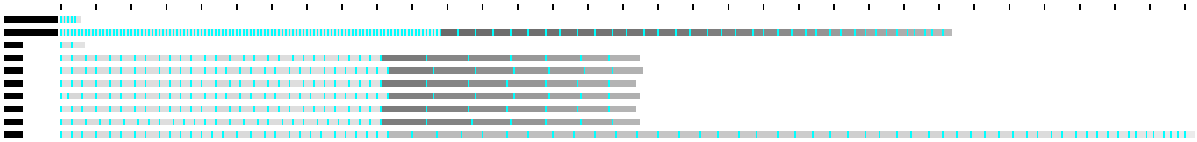

Figure 18 Opp+FastTrack

Opp+min_min

run:
ScheduleSim 1.3.0
Running self test...

Running Tests:

testing EXPR_1
ScheduleSim Experiment Results
Min_Min,Opp,219 <-make, 0.6797077922077922 <-,util 61.927472527472524,

ScheduleSim Experiment Results
Min_Min,Opp,221 <-make, 0.674002574002574 <-,util
51.68848167539267,151.63076923076923,

ScheduleSim Experiment Results
Min_Min,Opp,216 <-make, 0.6902896642527979 <-,util
42.74104683195592,110.0,157.27906976744185,

ScheduleSim Experiment Results
Min_Min,Opp,226 <-make, 0.659565764631844 <-,util
48.079320113314445,104.09375,132.96875,172.63636363636363,

ScheduleSim Experiment Results
Min_Min,Opp,224 <-make, 0.6647936507936508 <-,util
47.07183908045977,97.3076923076923,117.11538461538461,143.30769230769232,176.69230769230768,

ScheduleSim Experiment Results
Min_Min,Opp,244 <-make, 0.6106997084548105 <-,util
101.21470588235294,13.772727272727273,28.19047619047619,47.27272727272727,71.0952380952381,99.18181818181819,

ScheduleSim Experiment Results
Min_Min,Opp,438 <-make, 0.34080702896192644 <-,util
52.36,126.09375,186.1578947368421,222.77777777777777,265.2105263157895,317.22222222222223,380.1578947368421,

ScheduleSim Experiment Results
Min_Min,Opp,224 <-make, 0.6644126984126983 <-,util
43.77258566978193,82.69444444444444,98.5,108.8125,124.3125,141.0,160.3125,182.647058823529 42,

ScheduleSim Experiment Results
Min_Min,Opp,221 <-make, 0.6746782496782497 <-,util
40.93511450381679,67.25287356321839,95.26666666666667,105.5,116.85714285714286,131.33333333333334,147.78571428571428,164.07142857142858,184.06666666666666,

ScheduleSim Experiment Results
Min_Min,Opp,182 <-make, 0.8176424668227947 <-,util
25.074235807860262,46.86776859504132,56.92307692307692,66.3076923076923,74.53846153846153,86.53846153846153,99.76923076923077,114.92307692307692,131.23076923076923,149.3846153846154,

SUCCESS

BUILD SUCCESSFUL (total time: 2 seconds)

Opp+Max_min

run:
ScheduleSim 1.3.0
Running self test...

Running Tests:

testing EXPR_1
ScheduleSim Experiment Results
Max_Min,Opp,228 <-make, 0.6541796631316282 <-,util 77.3546255506608,

ScheduleSim Experiment Results
Max_Min,Opp,533 <-make, 0.28027019796682723 <-,util
176.96915167095116,331.83076923076925,

ScheduleSim Experiment Results
Max_Min,Opp,217 <-make, 0.686435124508519 <-,util
56.868852459016395,150.7674418604651,107.04651162790698,

ScheduleSim Experiment Results
Max_Min,Opp,173 <-make, 0.8601395730706075 <-,util
49.73446327683616,129.78125,104.21875,67.60606060606061,

ScheduleSim Experiment Results
Max_Min,Opp,167 <-make, 0.8903486394557824 <-,util
61.554597701149426,111.96153846153847,94.46153846153847,69.73076923076923,39.5
3846153846154,

ScheduleSim Experiment Results
Max_Min,Opp,223 <-make, 0.6682397959183674 <-,util
56.748520710059175,157.56,166.33333333333334,149.77272727272728,127.8095238095
2381,101.13636363636364,

ScheduleSim Experiment Results
Max_Min,Opp,213 <-make, 0.6994659546061416 <-,util
48.42727272727273,112.39285714285714,161.89473684210526,149.88888888888889,132
.05263157894737,113.61111111111111,91.21052631578948,

ScheduleSim Experiment Results
Max_Min,Opp,226 <-make, 0.6592825676526117 <-,util
56.21405750798722,87.69047619047619,177.375,162.0,152.3125,137.4375,116.75,100.88
235294117646,

ScheduleSim Experiment Results
Max_Min,Opp,185 <-make, 0.8040706605222735 <-,util
45.35036496350365,37.98684210526316,144.93333333333334,137.42857142857142,126.

14285714285714,114.13333333333334,99.28571428571429,82.14285714285714,65.53333333333333,

ScheduleSim Experiment Results
Max_Min,Opp,228 <-make, 0.654086088583905 <-,util
60.62,50.12621359223301,182.0,170.76923076923077,162.3846153846154,157.92307692307693,139.15384615384616,126.07692307692308,114.61538461538461,97.0,

SUCCESS

BUILD SUCCESSFUL (total time: 4 seconds)

Opp+FastTrack

run:
ScheduleSim 1.3.0
Running self test...

Running Tests:

testing EXPR_1
ScheduleSim Experiment Results
FastTrack,Opp,311 <-make, 0.4800137362637364 <-,util 71.44370860927152,

ScheduleSim Experiment Results
FastTrack,Opp,566 <-make, 0.2642101284958428 <-,util
181.74742268041237,333.2307692307692,

ScheduleSim Experiment Results
FastTrack,Opp,469 <-make, 0.31834346504559274 <-,util
104.92602739726027,354.48837209302326,232.65116279069767,

ScheduleSim Experiment Results
FastTrack,Opp,563 <-make, 0.26540020263424513 <-,util
160.0280112044818,415.40625,397.65625,271.72727272727275,

ScheduleSim Experiment Results
FastTrack,Opp,290 <-make, 0.5146048109965635 <-,util
50.174418604651166,123.73076923076923,131.57692307692307,104.38461538461539,76.03846153846153,

ScheduleSim Experiment Results
FastTrack,Opp,323 <-make, 0.4619929453262787 <-,util
61.733333333333334,179.2608695652174,154.76190476190476,152.13636363636363,136
.52380952380952,120.4090909090909,

ScheduleSim Experiment Results
FastTrack,Opp,565 <-make, 0.264449772841999 <-,util
150.37951807228916,309.75,395.3157894736842,446.6666666666667,389.210526315789
5,320.3888888888889,239.94736842105263,

ScheduleSim Experiment Results
FastTrack,Opp,323 <-make, 0.4619929453262786 <-,util
62.68976897689769,89.37254901960785,142.25,160.1875,158.6875,141.3125,125.6875,11
7.0,

ScheduleSim Experiment Results
FastTrack,Opp,353 <-make, 0.42332526230831313 <-,util
50.527272727272724,60.921052631578945,172.93333333333334,216.57142857142858,29
8.92857142857144,254.8,203.85714285714286,147.85714285714286,82.8,

ScheduleSim Experiment Results
FastTrack,Opp,323 <-make, 0.46155202821869484 <-,util
66.08597285067873,58.86507936507937,189.76923076923077,120.46153846153847,183.
0,147.46153846153845,158.07692307692307,124.07692307692308,119.46153846153847,
125.76923076923077,

SUCCESS

BUILD SUCCESSFUL total time: 2 seconds

Max_min+Max_min

run:
ScheduleSim 1.3.0
Running self test...

Running Tests:

testing EXPR_1
ScheduleSim Experiment Results
Max_Min,Max_Min,162 <-make, 0.9187116564417178 <-,util 53.59777777777778,

ScheduleSim Experiment Results
Max_Min,Max_Min,164 <-make, 0.9058874458874459 <-,util
47.932467532467534,85.87692307692308,

ScheduleSim Experiment Results
Max_Min,Max_Min,163 <-make, 0.9128484320557492 <-,util
44.26027397260274,107.74418604651163,77.32558139534883,

ScheduleSim Experiment Results
Max_Min,Max_Min,162 <-make, 0.9167397020157757 <-,util
41.75852272727273,116.40625,98.28125,73.54545454545455,

ScheduleSim Experiment Results
Max_Min,Max_Min,164 <-make, 0.9074891774891777 <-,util
40.328530259366,120.61538461538461,108.96153846153847,92.0,70.73076923076923,

ScheduleSim Experiment Results
Max_Min,Max_Min,163 <-make, 0.9133710801393727 <-,util
39.21449275362319,124.13636363636364,112.71428571428571,103.04545454545455,85.52380952380952,69.68181818181819,

ScheduleSim Experiment Results
Max_Min,Max_Min,162 <-make, 0.9178790534618756 <-,util
39.13939393939394,84.78571428571429,117.73684210526316,108.88888888888889,96.52631578947368,82.61111111111111,68.36842105263158,

ScheduleSim Experiment Results
Max_Min,Max_Min,163 <-make, 0.9120644599303136 <-,util
39.43037974683544,62.60526315789474,118.0,112.9375,103.625,93.0,82.25,66.05882352941177,

ScheduleSim Experiment Results
Max_Min,Max_Min,163 <-make, 0.9126742160278746 <-,util
40.34146341463415,42.44776119402985,123.0,115.42857142857143,108.78571428571429,99.46666666666667,89.35714285714286,77.92857142857143,65.93333333333334,

ScheduleSim Experiment Results

Max_Min,Max_Min,164 <-make, 0.9069264069264071 <-,util
45.28699551569507,31.376,122.3076923076923,119.0,112.46153846153847,104.5384615
3846153,96.84615384615384,87.07692307692308,73.61538461538461,67.692307692307,

SUCCESS

BUILD SUCCESSFUL total time: 1 second

Max_min+FastTrack

run:
ScheduleSim 1.3.0
Running self test...

Running Tests:

testing EXPR_1
ScheduleSim Experiment Results
FastTrack,Max_Min,226 <-make, 0.6589993706733794 <-,util 56.07743362831859,

ScheduleSim Experiment Results
FastTrack,Max_Min,226 <-make, 0.65921963499056 <-,util 49.75835475578406,92.6,

ScheduleSim Experiment Results
FastTrack,Max_Min,227 <-make, 0.656046365914787 <-,util
46.561307901907355,107.81395348837209,82.6046511627907,

ScheduleSim Experiment Results
FastTrack,Max_Min,226 <-make, 0.6591881686595342 <-,util
45.11581920903955,103.9375,106.46875,79.60606060606061,

ScheduleSim Experiment Results
FastTrack,Max_Min,227 <-make, 0.6567982456140351 <-,util
43.540462427745666,102.61538461538461,115.73076923076923,95.8076923076923,78.9
6153846153847,

ScheduleSim Experiment Results
FastTrack,Max_Min,226 <-make, 0.6587476400251729 <-,util
42.328445747800586,108.54166666666667,103.0952380952381,109.81818181818181,92.
52380952380952,75.86363636363636,

ScheduleSim Experiment Results
FastTrack,Max_Min,227 <-make, 0.6564223057644111 <-,util
42.694189602446485,85.62068965517241,98.05263157894737,111.83333333333333,104.57894736842105,84.11111111111111,77.05263157894737,

ScheduleSim Experiment Results
FastTrack,Max_Min,226 <-make, 0.6590308370044053 <-,util
42.381107491856675,60.75,91.6875,116.25,114.1875,98.25,79.5625,76.23529411764706,

ScheduleSim Experiment Results
FastTrack,Max_Min,226 <-make, 0.6597860289490246 <-,util
44.36363636363637,42.93589743589744,107.13333333333334,106.42857142857143,114.78571428571429,103.26666666666667,91.92857142857143,78.42857142857143,78.53333333333333,

ScheduleSim Experiment Results
FastTrack,Max_Min,226 <-make, 0.658936438011328 <-,util
47.28636363636364,36.388888888888886,119.38461538461539,85.3076923076923,120.53846153846153,110.46153846153847,101.3076923076923,89.84615384615384,76.0,77.53846153846153,

SUCCESS

BUILD SUCCESSFUL total time: 3 seconds

FastTrack+FastTrack

---

run:
ScheduleSim 1.3.0
Running self test...

Running Tests:

testing EXPR_1
ScheduleSim Experiment Results
FastTrack,FastTrack,176 <-make, 0.845318805488297 <-,util 53.99777282850779,

ScheduleSim Experiment Results
FastTrack,FastTrack,178 <-make, 0.835634477254589 <-,util
47.6839378238342,90.6923076923077,

ScheduleSim Experiment Results

FastTrack,FastTrack,177 <-make, 0.8400882825040129 <-,util
44.42739726027397,102.02325581395348,88.0,

ScheduleSim Experiment Results
FastTrack,FastTrack,177 <-make, 0.8408105939004815 <-,util
43.04829545454545,99.375,102.5,79.6969696969697,

ScheduleSim Experiment Results
FastTrack,FastTrack,174 <-make, 0.8542448979591837 <-,util
41.497126436781606,83.88461538461539,112.26923076923077,107.1923076923077,73.1
5384615384616,

ScheduleSim Experiment Results
FastTrack,FastTrack,175 <-make, 0.8511363636363636 <-,util
41.11695906432749,85.04166666666667,109.04761904761905,96.86363636363636,107.8
5714285714286,69.86363636363636,

ScheduleSim Experiment Results
FastTrack,FastTrack,178 <-make, 0.8364325618515563 <-,util
41.44311377245509,72.38461538461539,89.63157894736842,112.33333333333333,102.1
5789473684211,100.27777777777777,67.15789473684211,

ScheduleSim Experiment Results
FastTrack,FastTrack,175 <-make, 0.851055194805195 <-,util
41.78328173374613,54.529411764705884,76.75,123.4375,90.3125,116.375,94.8125,65.23
529411764706,

ScheduleSim Experiment Results
FastTrack,FastTrack,179 <-make, 0.8318650793650795 <-,util
44.11913357400722,37.12328767123287,86.46666666666667,102.21428571428571,112.2
1428571428571,93.33333333333333,112.92857142857143,89.5,62.733333333333334,

ScheduleSim Experiment Results
FastTrack,FastTrack,173 <-make, 0.8607963875205253 <-,util
45.99047619047619,34.85401459854015,92.84615384615384,74.92307692307692,131.0,9
4.76923076923077,107.15384615384616,108.15384615384616,85.92307692307692,61.53
846153846154,

SUCCESS

BUILD SUCCESSFUL total time: 1 second

10.9 APPENDIX IX – Experiment 2 visuals and Results

Figure 19 - FastTrack+FastTrack

Figure 20 - Max_min+Max_min

Figure 21 - Opp+FastTrack

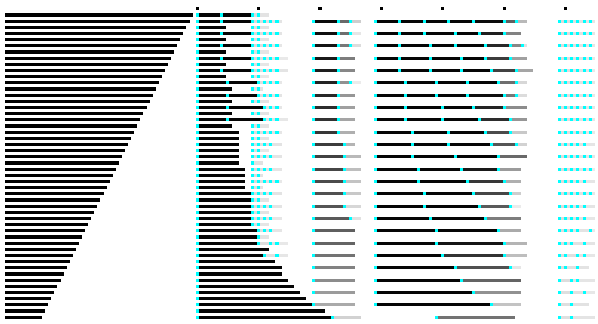

Figure 22 - Opp+Max_min

Figure 23 - Opp+min_min

Opp+min_min

run:
ScheduleSim 1.3.0
Running self test...

Running Tests:

testing EXPR_2
ScheduleSim Experiment Results
Min,Opp,65 <-make, 0.4452838241570638 <-,util 5.026415094339622,

ScheduleSim Experiment Results
Min,Opp,65 <-make, 0.4453179684165603 <-,util 3.032520325203252,9.939024390243903,

ScheduleSim Experiment Results
Min,Opp,65 <-make, 0.44524967989756753 <-,util
2.8929577464788734,7.1,10.113333333333333,

ScheduleSim Experiment Results
Min,Opp,65 <-make, 0.4452326077678192 <-,util
2.6063218390804597,5.75,8.636363636363637,10.140845070422536,

ScheduleSim Experiment Results
Min,Opp,65 <-make, 0.4452411438326931 <-,util
2.2915451895043732,5.166666666666667,7.388888888888889,9.944444444444445,10.442
028985507246,

ScheduleSim Experiment Results
Min,Opp,65 <-make, 0.4450704225352116 <-,util
2.817910447761194,4.25,6.25,8.071428571428571,10.1875,10.104477611940299,

ScheduleSim Experiment Results

Min,Opp,65 <-make, 0.445224071702945 <-,util
2.5861027190332324,3.4166666666666665,5.083333333333333,6.416666666666667,7.8,9.
333333333333334,10.916666666666666,10.069230769230769,

ScheduleSim Experiment Results
Min,Opp,65 <-make, 0.44513871105420416 <-,util
2.56508875739645,3.3,4.6,6.0,6.9,8.4,9.8,11.0,10.069230769230769,

ScheduleSim Experiment Results
Min,Opp,65 <-make, 0.445394793000427 <-,util
2.8384146341463414,3.5833333333333335,4.25,5.5,6.5,7.625,9.0,10.125,11.3,10.0390625,

SUCCESS

BUILD SUCCESSFUL (total time: 3 seconds)

Opp+Max_min

run:
ScheduleSim 1.3.0
Running self test...

Running Tests:

testing EXPR_2
ScheduleSim Experiment Results
Max,Opp,64 <-make, 0.45193499458288194 <-,util 5.2781954887218046,

ScheduleSim Experiment Results
Max,Opp,64 <-make, 0.4520736728060673 <-,util 4.14247311827957,9.341463414634147,

ScheduleSim Experiment Results
Max,Opp,64 <-make, 0.45214301191766004 <-,util
3.906515580736544,9.233333333333333,9.373333333333333,

ScheduleSim Experiment Results
Max,Opp,64 <-make, 0.45203033586132174 <-,util
2.924198250728863,9.75,9.090909090909092,9.67605633802817,

ScheduleSim Experiment Results
Max,Opp,64 <-make, 0.45196966413867806 <-,util
3.2682215743440235,6.055555555555555,6.0,4.888888888888889,9.072463768115941,

ScheduleSim Experiment Results
Max,Opp,64 <-make, 0.45206500541711825 <-,util
3.6339285714285716,9.5625,9.4375,9.0,7.8125,9.559701492537313,

ScheduleSim Experiment Results
Max,Opp,64 <-make, 0.45204767063922 <-,util
2.625748502994012,9.666666666666666,9.583333333333334,9.333333333333334,9.2,8.25,
7.5,9.623076923076923,

ScheduleSim Experiment Results
Max,Opp,64 <-make, 0.4521170097508125 <-,util
2.5150602409638556,6.6,6.0,6.0,6.0,5.7,4.7,4.0,9.592307692307692,

ScheduleSim Experiment Results
Max,Opp,64 <-make, 0.45214301191766 <-,util
3.513595166163142,9.7,9.625,9.6,9.3,9.25,8.5,7.875,7.4,9.65625,

SUCCESS

BUILD SUCCESSFUL (total time: 7 seconds)

Opp+FastTrack

run:
ScheduleSim 1.3.0
Running self test...

Running Tests:

testing EXPR_2
ScheduleSim Experiment Results
Opp,FastTrack,65 <-make, 0.4451899274434485 <-,util 5.584269662921348,

ScheduleSim Experiment Results
Opp,FastTrack,65 <-make, 0.4450277422108409 <-,util
4.053763440860215,9.487804878048781,

ScheduleSim Experiment Results
Opp,FastTrack,65 <-make, 0.44527528809218964 <-,util
3.8477011494252875,9.733333333333333,9.46,

ScheduleSim Experiment Results
Opp,FastTrack,65 <-make, 0.44531796841656 <-,util
3.4770114942528734,8.5,9.363636363636363,9.528169014084508,

ScheduleSim Experiment Results
Opp,FastTrack,65 <-make, 0.4452240717029451 <-,util
3.415204678362573,7.333333333333333,10.222222222222221,8.833333333333334,9.5652
17391304348,

ScheduleSim Experiment Results
Opp,FastTrack,65 <-make, 0.4450874946649598 <-,util
3.6451612903225805,7.3125,9.6875,9.785714285714286,8.375,9.58955223880597,

ScheduleSim Experiment Results
FastTrack,Opp,65 <-make, 0.4451643192488264 <-,util
3.1921921921921923,5.5,3.1666666666666665,7.0,7.0,5.666666666666667,4.58333333333
3333,9.438461538461539,

ScheduleSim Experiment Results
FastTrack,Opp,65 <-make, 0.4453521126760563 <-,util
2.5545722713864305,5.8,3.0,5.8,7.0,6.7,5.3,4.5,9.646153846153846,

ScheduleSim Experiment Results
FastTrack,Opp,65 <-make, 0.4450533504054633 <-,util
3.5424242424242425,9.6,7.5,7.2,10.4,10.0,9.3,8.25,7.7,9.6875,

SUCCESS

BUILD SUCCESSFUL (total time: 2 seconds)

Max_min+Max_min

run:
ScheduleSim 1.3.0
Running self test...

Running Tests:

testing EXPR_2
ScheduleSim Experiment Results
Max,Max,62 <-make, 0.4664520456069751 <-,util 3.166355140186916,

ScheduleSim Experiment Results
Max,Max,62 <-make, 0.46637156270959074 <-,util
1.7903225806451613,6.286585365853658,

ScheduleSim Experiment Results
Max,Max,62 <-make, 0.46656829868097494 <-,util
1.725212464589235,3.3333333333333335,6.546666666666667,

ScheduleSim Experiment Results
Max,Max,62 <-make, 0.4664252179745138 <-,util
1.710144927536232,3.0416666666666665,3.5,6.71830985915493,

ScheduleSim Experiment Results
Max,Max,62 <-make, 0.46664878157835893 <-,util
1.7055393586005831,2.7777777777777777,3.3333333333333335,3.388888888888889,6.82
6086956521739,

ScheduleSim Experiment Results
Max,Max,62 <-make, 0.46646993069528275 <-,util
1.6923076923076923,2.5625,3.1875,3.5,3.25,6.940298507462686,

ScheduleSim Experiment Results
Max,Max,62 <-make, 0.4666219539458976 <-,util
1.685459940652819,2.3333333333333335,2.9166666666666665,3.1666666666666665,3.5,3
.5,3.0,7.061538461538461,

ScheduleSim Experiment Results
Max,Max,62 <-make, 0.4663536776212833 <-,util
1.6736526946107784,2.4,2.6,3.2,3.3,3.5,3.4,3.0,7.061538461538461,

ScheduleSim Experiment Results
Max,Max,62 <-make, 0.46654147104851335 <-,util
1.6838905775075987,2.090909090909091,2.375,3.1,3.2,3.5,3.5,3.25,3.2,7.109375,

SUCCESS

BUILD SUCCESSFUL (total time: 8 seconds)

Max_min+FastTrack

run:
ScheduleSim 1.3.0
Running self test...

Running Tests:

testing EXPR_2
ScheduleSim Experiment Results

FastTrack,Max_Min,62 <-make, 0.46635367762128327 <-,util 3.292134831460674,

ScheduleSim Experiment Results
FastTrack,Max_Min,62 <-make, 0.4664073328862062 <-,util
1.8556149732620322,6.439024390243903,

ScheduleSim Experiment Results
FastTrack,Max_Min,62 <-make, 0.46654147104851335 <-,util
1.8356940509915014,3.5,6.673333333333333,

ScheduleSim Experiment Results
FastTrack,Max_Min,62 <-make, 0.46660406885759 <-,util
1.8401162790697674,2.5833333333333335,3.9545454545454546,6.823943661971831,

ScheduleSim Experiment Results
FastTrack,Max_Min,62 <-make, 0.4665057008718981 <-,util
1.8411764705882352,2.2222222222222223,3.5,4.055555555555555,6.898550724637682,

ScheduleSim Experiment Results
FastTrack,Max_Min,62 <-make, 0.46630002235636037 <-,util
1.8023598820058997,2.125,3.125,3.9285714285714284,4.0625,6.985074626865671,

ScheduleSim Experiment Results
FastTrack,Max_Min,62 <-make, 0.4663894477978986 <-,util
1.8053892215568863,3.0,2.0,3.1666666666666665,3.9,4.0,4.083333333333333,7.07692307
6923077,

ScheduleSim Experiment Results
FastTrack,Max_Min,62 <-make, 0.46646993069528275 <-,util
1.8036253776435045,3.1,1.7,3.0,3.5,4.0,4.1,4.0,7.076923076923077,

ScheduleSim Experiment Results
FastTrack,Max_Min,62 <-make, 0.466219539458976 <-,util
1.774390243902439,3.4,2.25,2.2,3.2,3.875,4.0,4.125,4.0,7.125,

SUCCESS

BUILD SUCCESSFUL (total time: 10 seconds)

FastTrack+FastTrack

run:
ScheduleSim 1.3.0
Running self test...

Running Tests:

testing EXPR_2
ScheduleSim Experiment Results
FastTrack,FastTrack,63 <-make, 0.45893485915492965 <-,util 3.507518796992481,

ScheduleSim Experiment Results
FastTrack,FastTrack,63 <-make, 0.45916373239436625 <-,util
1.9811320754716981,6.951219512195122,

ScheduleSim Experiment Results
FastTrack,FastTrack,63 <-make, 0.4589348591549297 <-,util
1.839541547277937,4.866666666666666,7.166666666666667,

ScheduleSim Experiment Results
FastTrack,FastTrack,63 <-make, 0.45907570422535227 <-,util
1.8517441860465116,3.75,5.454545454545454,7.183098591549296,

ScheduleSim Experiment Results
FastTrack,FastTrack,63 <-make, 0.45910211267605644 <-,util
1.8727810650887573,2.3333333333333335,4.055555555555555,6.833333333333333,7.217
391304347826,

ScheduleSim Experiment Results
FastTrack,FastTrack,63 <-make, 0.4592077464788733 <-,util
1.894578313253012,1.0625,5.0625,4.642857142857143,6.3125,7.268656716417911,

ScheduleSim Experiment Results
FastTrack,FastTrack,63 <-make, 0.4591901408450705 <-,util
1.8885542168674698,1.0,2.0833333333333335,5.416666666666667,3.6,7.0,5.75,7.3153846
15384615,

ScheduleSim Experiment Results
FastTrack,FastTrack,63 <-make, 0.4592253521126762 <-,util
1.8821752265861027,1.0,1.1,5.3,4.1,5.2,6.7,5.7,7.315384615384615,

ScheduleSim Experiment Results

FastTrack,FastTrack,63 <-make, 0.45938380281690144 <-,util
1.9181818181818182,0.75,1.125,3.3,5.2,2.625,7.2,6.375,5.6,7.34375,

SUCCESS

BUILD SUCCESSFUL (total time: 2 seconds)

10.10 APPENDIX X – Experiment 3 visuals and Results

Figure 24 - FastTrack+FastTrack

Figure 25 - Max_min+fastTrack

Figure 26 - Opp+FastTrack

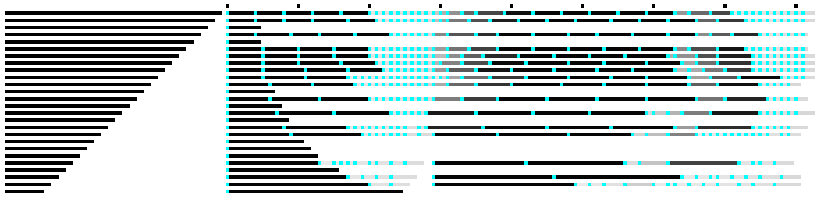

Figure 27 - Opp+min_min

Max_min+Max_min

run:
ScheduleSim 1.3.0
Running self test...

Running Tests:

testing EXPR_3
ScheduleSim Experiment Results
Max_Min,Max_Min,64 <-make, 0.8815384615384615 <-,util 8.318352059925093,

ScheduleSim Experiment Results
Max_Min,Max_Min,64 <-make, 0.8817413355874891 <-,util
7.08130081300813,11.060975609756097,

ScheduleSim Experiment Results

Max_Min,Max_Min,64 <-make, 0.8820794590025357 <-,util
6.9602272727272725,10.833333333333334,11.086666666666666,

ScheduleSim Experiment Results
Max_Min,Max_Min,64 <-make, 0.8815553677092137 <-,util
6.860058309037901,10.75,10.590909090909092,11.154929577464788,

ScheduleSim Experiment Results
Max_Min,Max_Min,64 <-make, 0.8817244294167372 <-,util
6.79646017699115,11.055555555555555,10.833333333333334,10.0,11.217391304347826,

ScheduleSim Experiment Results
Max_Min,Max_Min,64 <-make, 0.8817751479289941 <-,util
6.748502994011976,10.8125,10.625,11.0,9.75,11.26865671641791,

ScheduleSim Experiment Results
Max_Min,Max_Min,64 <-make, 0.8817244294167372 <-,util
6.738601823708207,10.666666666666666,11.0,10.666666666666666,11.1,10.083333333333334,9.583333333333334,11.3,

ScheduleSim Experiment Results
Max_Min,Max_Min,64 <-make, 0.8818089602704987 <-,util
6.728915662650603,9.909090909090908,11.2,10.8,10.6,11.2,9.8,9.4,11.315384615384616,

ScheduleSim Experiment Results
Max_Min,Max_Min,64 <-make, 0.8818089602704989 <-,util
6.753012048192771,10.3,11.125,11.2,10.4,11.125,10.5,9.5,9.2,11.3828125,

SUCCESS

BUILD SUCCESSFUL (total time: 8 seconds)

Opp+Max_min

run:
ScheduleSim 1.3.0
Running self test...

Running Tests:

testing EXPR_3
ScheduleSim Experiment Results
Max_Min,Opp,83 <-make, 0.6821690214547355 <-,util 18.81902985074627,

ScheduleSim Experiment Results
Max_Min,Opp,83 <-make, 0.6825091575091569 <-,util
18.341463414634145,19.890243902439025,

ScheduleSim Experiment Results
Max_Min,Opp,83 <-make, 0.682378335949764 <-,util
18.022662889518415,25.066666666666666,19.433333333333334,

ScheduleSim Experiment Results
Max_Min,Opp,76 <-make, 0.7445982588839731 <-,util
7.344827586206897,13.208333333333334,10.363636363636363,14.732394366197184,

ScheduleSim Experiment Results
Max_Min,Opp,83 <-make, 0.6821428571428566 <-,util
17.723529411764705,26.555555555555557,24.666666666666668,23.77777777777778,19.0
5072463768116,

ScheduleSim Experiment Results
Max_Min,Opp,83 <-make, 0.6823783359497642 <-,util
17.65680473372781,25.875,25.4375,24.714285714285715,23.375,18.98507462686567,

ScheduleSim Experiment Results
Max_Min,Opp,75 <-make, 0.7542943898207053 <-,util
7.105421686746988,16.666666666666668,16.666666666666668,15.75,15.3,14.25,12.25,15.
946153846153846,

ScheduleSim Experiment Results
Max_Min,Opp,72 <-make, 0.7849465602890261 <-,util
7.213855421686747,16.6,16.8,16.2,15.6,14.9,13.9,12.0,15.946153846153846,

ScheduleSim Experiment Results
Max_Min,Opp,83 <-make, 0.6822998430141284 <-,util
17.473053892215567,26.0,26.0,26.6,25.0,25.125,24.1,23.375,21.6,18.859375,

SUCCESS

BUILD SUCCESSFUL (total time: 8 seconds)

FastTrack+FastTrack

run:
ScheduleSim 1.3.0

Running self test...

Running Tests:

testing EXPR_3
ScheduleSim Experiment Results
FastTrack,FastTrack,78 <-make, 0.7252608151342329 <-,util 9.080223880597014,

ScheduleSim Experiment Results
FastTrack,FastTrack,89 <-make, 0.6370329670329671 <-,util
8.764075067024129,12.371951219512194,

ScheduleSim Experiment Results
FastTrack,FastTrack,99 <-make, 0.5729670329670329 <-,util
10.575070821529746,12.633333333333333,12.96,

ScheduleSim Experiment Results
FastTrack,FastTrack,100 <-make, 0.5676096181046677 <-,util
10.741279069767442,11.166666666666666,11.0,13.056338028169014,

ScheduleSim Experiment Results
FastTrack,FastTrack,89 <-make, 0.636947496947497 <-,util
8.576470588235294,9.722222222222221,12.5,11.555555555555555,12.623188405797102,

ScheduleSim Experiment Results
FastTrack,FastTrack,89 <-make, 0.6370085470085468 <-,util
8.789317507418398,5.25,14.0625,10.642857142857142,10.9375,13.208955223880597,

ScheduleSim Experiment Results
FastTrack,FastTRack,100 <-make, 0.5673920139266675 <-,util
11.012012012012011,6.0,7.083333333333333,15.0,10.0,11.75,9.916666666666666,13.3384
61538461539,

ScheduleSim Experiment Results
FastTrack,FastTRack,89 <-make, 0.6368498168498168 <-,util
8.605970149253732,7.0,7.0,15.5,12.7,10.7,11.6,9.8,12.792307692307693,

ScheduleSim Experiment Results
FastTrack,FastTRack,99 <-make, 0.5731538461538462 <-,util
10.736363636363636,12.4,5.75,9.7,14.8,9.625,11.8,11.25,9.4,13.421875,

SUCCESS

BUILD SUCCESSFUL (total time: 6 seconds)

Max_min+FastTrack

run:
ScheduleSim 1.3.0
Running self test...

Running Tests:

testing EXPR_3
ScheduleSim Experiment Results
FastTrack,Max_Min,82 <-make, 0.6904011650999603 <-,util 10.00945179584121,

ScheduleSim Experiment Results
FastTrack,Max_Min,82 <-make, 0.6905468025949953 <-,util
9.232432432432432,11.774390243902438,

ScheduleSim Experiment Results
FastTrack,Max_Min,81 <-make, 0.6990753149289735 <-,util
9.127478753541077,10.566666666666666,11.9,

ScheduleSim Experiment Results
FastTrack,Max_Min,82 <-make, 0.6903746855554084 <-,util
9.078260869565218,10.708333333333334,10.0,12.02112676056338,

ScheduleSim Experiment Results
FastTrack,Max_Min,81 <-make, 0.6989145001340124 <-,util
8.96774193548387,12.166666666666666,10.166666666666666,10.0,12.115942028985508,

ScheduleSim Experiment Results
FastTrack,Max_Min,82 <-make, 0.6904276446445122 <-,util
8.954682779456194,15.1875,10.6875,10.428571428571429,9.0625,12.26865671641791,

ScheduleSim Experiment Results
FastTrack,Max_Min,82 <-make, 0.6907453991791342 <-,util
8.874251497005988,17.833333333333332,11.083333333333334,10.5,10.4,9.666666666666
666,8.5,12.407692307692308,

ScheduleSim Experiment Results
FastTrack,Max_Min,82 <-make, 0.6908248378127896 <-,util
8.886904761904763,17.4,13.3,10.6,10.6,10.4,9.0,8.6,12.407692307692308,

ScheduleSim Experiment Results
FastTrack,Max_Min,82 <-make, 0.6907321594068583 <-,util
8.637462235649547,20.7,14.125,11.0,10.5,9.875,10.3,9.5,8.2,12.453125,

SUCCESS

BUILD SUCCESSFUL (total time: 7 seconds)

Opp+FastTrack

run:
ScheduleSim 1.3.0
Running self test...

Running Tests:

testing EXPR_3
ScheduleSim Experiment Results
FastTrack,Opp,93 <-make, 0.6094926350245504 <-,util 19.149812734082396,

ScheduleSim Experiment Results
FastTrack,Opp,94 <-make, 0.603262001156738 <-,util
18.84054054054054,19.902439024390244,

ScheduleSim Experiment Results
FastTrack,Opp,94 <-make, 0.6035049161364952 <-,util
18.519662921348313,25.3,19.486666666666668,

ScheduleSim Experiment Results
FastTrack,Opp,83 <-make, 0.6823129251700679 <-,util
8.296511627906977,16.041666666666668,15.227272727272727,16.014084507042252,

ScheduleSim Experiment Results
FastTrack,Opp,94 <-make, 0.6032272990167725 <-,util
18.12684365781711,27.666666666666668,25.055555555555557,24.22222222222222,19.16
6666666666668,

ScheduleSim Experiment Results
FastTrack,Opp,94 <-make, 0.6032620011567379 <-,util
17.97032640949555,29.9375,26.1875,24.142857142857142,23.3125,19.067164179104477,

ScheduleSim Experiment Results

FastTrack,Opp,140 <-make, 0.40650767672044275 <-,util
9.3003003003003,39.25,15.5,13.583333333333334,11.9,10.0,7.333333333333333,15.84615
3846153847,

ScheduleSim Experiment Results
FastTrack,Opp,140 <-make, 0.4064297404722937 <-,util
9.24702380952381,40.9,20.7,12.4,12.7,11.5,9.2,7.1,15.830769230769231,

ScheduleSim Experiment Results
FastTrack,Opp,93 <-make, 0.6099251812017774 <-,util
17.790419161676645,31.0,28.25,24.6,26.6,24.125,25.1,22.75,22.2,18.7890625,

SUCCESS

BUILD SUCCESSFUL (total time: 10 seconds)

Opp+min_min

run:
ScheduleSim 1.3.0
Running self test...

Running Tests:

testing EXPR_3
ScheduleSim Experiment Results
Min_Min,Opp,83 <-make, 0.6824568288853998 <-,util 17.796992481203006,

ScheduleSim Experiment Results
Min_Min,Opp,83 <-make, 0.6825353218210356 <-,util
16.502732240437158,20.701219512195124,

ScheduleSim Experiment Results
Min_Min,Opp,83 <-make, 0.6824175824175818 <-,util
16.229461756373937,21.933333333333334,20.46,

ScheduleSim Experiment Results
Min_Min,Opp,74 <-make, 0.7643516483516483 <-,util
6.667630057803469,10.833333333333334,14.272727272727273,17.12676056338028,

ScheduleSim Experiment Results
Min_Min,Opp,83 <-make, 0.6819204604918886 <-,util
16.179411764705883,19.333333333333332,21.88888888888889,25.27777777777778,19.96
3768115942027,

ScheduleSim Experiment Results
Min_Min,Opp,83 <-make, 0.682378335949764 <-,util
16.1910447761194,18.5,21.125,23.214285714285715,25.8125,19.78358208955224,

ScheduleSim Experiment Results
Min_Min,Opp,77 <-make, 0.7345026768103691 <-,util
6.13595166163142,1.4166666666666667,3.1666666666666665,5.333333333333333,7.5,10.0,13.666666666666666,16.853846153846153,

ScheduleSim Experiment Results
Min_Min,Opp,162 <-make, 0.3515067754331557 <-,util
52.14242424242424,61.63636363636363,58.1,60.6,61.3,63.3,69.5,67.7,44.73076923076923,

ScheduleSim Experiment Results
Min_Min,Opp,84 <-make, 0.6744537815126049 <-,util
16.157575757575756,16.3,18.375,19.8,20.8,23.5,24.2,25.875,28.4,19.375,

SUCCESS

BUILD SUCCESSFUL (total time: 5 seconds)