Table of Contents

1. Introduction

1.1. Background and Motivation

A report by [1], suggests that IT downtime costs top companies $26.5 Billion per year. This problem will continue to grow as the number of IT systems increases and current systems begin to age. The magnitude of downtime stretches further than a company's bottom line such as Delta Airlines 2019 systems outage [2], which caused international travel delays. Therefore, a reduction in response and turnaround times with respect to IT support incidents is an important step in reducing IT incurred costs and the consequences of downtime globally.

An IT incident is *"an unplanned interruption to an IT service or reduction in the quality of an IT service or a failure of a Configurable Item (CI) that has not yet impacted an IT service (for example failure of one disk from a mirror set)"* [3, p. 72]. These incidents arise from hardware failures, software failures, monitoring alerts and customer or user queries.

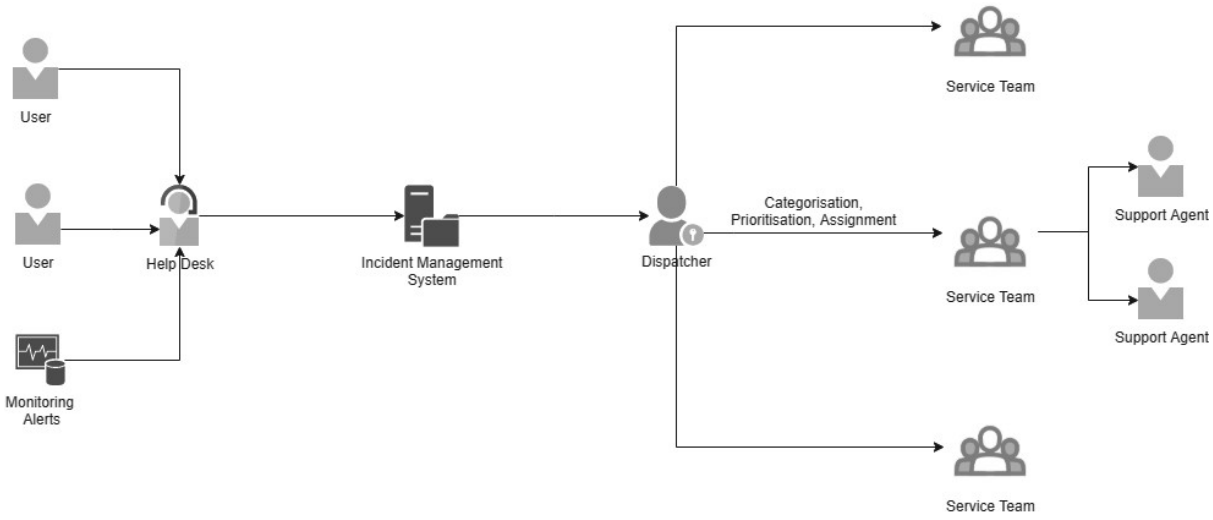

Figure 1 - Structure of Incident Management System

Figure 1 illustrates the typical structure of an IT support desk incident management system. These systems are typically designed around the IT Infrastructure Library (ITIL) [3] that describes a set of best practices and processes, specifically an 8-step incident life cycle process [3, p. 77]. When an incident is logged it leads to the creation of a ticket in the incident management system. A ticket describes the nature of the incident, any potential causes and optionally steps to reproduce it. Service Level Agreements (SLA) are also associated with an individual ticket. An SLA defines the first response and/or turnaround time due dates for the ticket. Failure to comply with these dates can incur penalties for the IT support desk. Tickets also have a form of categorisation applied and priority levels. Priority levels are determined by how severe the impact is on operations. A dispatcher uses a scheduling policy for assigning tickets to a team or individual agents.

Due to the real time and dynamic nature of the problem, it becomes difficult to identify an optimal schedule for tickets. This is because tickets arrive in a non-deterministic manner as time progresses.

1.2. Aims and Objectives

The main aim of this project is to research the effectiveness of existing and novel algorithms from the Operating System scheduling paradigm for use in the context of a dynamic IT support incident environment. The secondary aim of this project is to investigate machine learning and prediction techniques to increase the effectiveness of these algorithms.

The key objectives of this project are:

1. Design a simulation platform capable of producing a range of different ticket types to test the algorithms.
2. Implement and evaluate a selection of the algorithms found in the Operating System scheduling paradigm.
3. Investigate the information failure gap between static and dynamic environments.
4. Design and implement improvements to algorithm(s) based on the findings in the research and evaluation stage.
5. Evaluate the performance of the improved algorithm(s).
6. Design and implement a model capable of predicting future support incidents on a rolling window basis.
7. Implement integration between the machine learning model and scheduling algorithms.
8. Evaluate the performance of the algorithm(s) with the machine learning model.
9. Develop an interface to display the queues for the algorithms under study during the simulation. This will allow users to see the performance of the algorithms in real time.

2. Related Work

In this section, we present literature related to ticket scheduling, alternative Operating System methods and their performance. We will not review the algorithms as they will be described and explained in a later section.

2.1. Pre-emptive Scheduling

Pre-emptive algorithms are now the most widely used for scheduling processes in Operating Systems. Before them were simpler algorithms such as First Come First Serve (FCFS) that did not consider task switching.

There has been some investigation into the effectiveness of pre-emptive algorithms in context of ticket scheduling. One such example introduced an algorithm for high severity tickets, called High Severity Earliest Deadline First (HSEDF) [4] as well as a pre-emptive version, HSEDF-P. The authors of [4], identified the pre-emption version of HSEDF was beneficial in reducing SLA violations providing the overhead of context switching was below 45% [4]. These results align with work outlined in [5] which identifies the cost of task switching in humans to be between 0-40%. This shows that pre-emption can improve on SLA violations with data from the real world from two different departments spanning 4 months.

Despite the improvements proposed in [4], the approach fails to provide any measures to reduce the cost of context switching. The implementation of HSEDF-P can lead to excessive switching where the cost of switching is greater than the remaining time on the current ticket. Pre-emption Threshold Scheduling (PTS) could be used to reduce the overhead of context switching by reducing the number of times a context switch occurs. Work outlined by [6], highlights a reduction of context switches by a factor of 4.5 when

using PTS in the context of Real Time Operating Systems (RTOS). Although the effects of PTS look promising it wouldn't add any value to the implementation proposed by [4]. The problem lies in the datasets used, which only have two severity levels. For PTS to work a range of priority values need to be available, for instance the Linux scheduler, O(1) [7] has 140 priority levels by default.

2.2. Heuristic Based Scheduling

Work outlined by [8] illustrates the use of heuristics within this problem set offers performance improvements of up to 60%, the metric been weighted completion time. Compared to previous work, this approach considers more domain specific information in the scheduling process such as weights reflecting priority, staff skills and queue sizes.

The scenario used assumes tickets are available from time zero, which isn't reflective of the dynamic nature of the problem. The delay the authors introduced to counter this is somewhat superficial. Naturally, when there are more tickets than agents there is a delay between arrival and scheduling. By explicitly introducing a fixed waiting time it introduces the possibility of unnecessary delays because the wait time does not reflect current or future workloads.

One potential problem in this approach is no pre-emption of tasks under any circumstances. Initially, when referring to the performance improvements this does not seem to be a problem. To illustrate the potential costs, consider the following example: Assume that there are 4 staff members capable of handling tickets. Each member of staff is currently working on a medium priority ticket, the effort to complete the tasks are 30, 40, 45 and 40 minutes respectively. The amount of effort already expended on each task is 2, 5, 3, 10 minutes respectively. Now suppose a system outage leads to 3 system failures, thus 3 critical tickets are created. Using the approaches outlined in either of the two algorithms means there will be 3 SLA response time violations, according to the evaluation criteria set out in [8]. The violations would be by approximately 29, 35, 42, 30 minutes respectively.

3. Model

To enable reproducibility of the results, this section outlines the data and workload characteristics that will be used to analyse and evaluate the algorithms.

3.1. Dataset

Because of the sensitive nature of the problem there were no suitable datasets to drive the simulations. This led to the creation of a data generator. The initial analysis did find open data sets from [9] and [10]. Both datasets lacked time stamps for creation date and length of the tickets. However, both datasets were analysed for ticket type and severity, for the initial dataset the averages were taken between both sets of data. To make the data more realistic, the levels of tickets produced should change with respect to the hour of day, day of week and month of year. The results from [11] were used when designing the data generator and initial data set. The benefit of this approach allows for more granular control whilst keeping suitable levels of variation that make datasets more realistic, although if necessary, the parameters can be changed later. The data generator produces tickets with the following attributes:

Attribute	Description
Id	Uniquely identifies each ticket in the system.
Created At	Time stamp for when the ticket was created.
Type	Identifies if ticket is an incident or request.
Severity	Severity / impact of the ticket.
Length	How much time to complete the ticket.
Response Time At	Time stamp for when the ticket must be responded to.
Turn Around Time At	Time stamp for when the ticket must be resolved by.

Table 1 – Ticket attributes

3.2. Evaluation Criteria

The number of SLA violations is the primary metric for evaluating the performance of scheduling algorithms in this project. Although response and turnaround times are a byproduct and reducing these average times should reduce SLA violations. As such, SLA criteria was developed using data from [8], adjusted appropriately to account for two different types of ticket, work performed in [4] and [8] only considers one type. The SLA times outlined in Table 2 must respect the environments work hours. For example, if work hours are Monday to Friday and a critical ticket arrives on Saturday, it's response and turnaround time would be 9AM and 10AM on the Monday respectively.

Ticket Type	Critical	High	Medium	Low
Incident	-	-	-	-
Response Time	1	30	60	120
Turnaround Time	60	120	240	480
Request	-	-	-	-
Response Time	-	120	240	480
Turnaround Time	-	240	480	720

Table 2 – Ticket Service Level Agreements (Minutes)

3.3. Data Generation Results

A breakdown of the data created from the data generator is presented in this subsection. The breakdown considers most aspects of the 1549 tickets created from 2019-11-19T22:00 to 2020-02-11T22:00 (12 weeks). The data was generated based on a company of 500 employees.

	Total	Critical	%	High	%	Medium	%	Low	%
Incident	515	14	2.7%	54	10.5%	258	50.1%	189	36.7%
Request	1034	0	0%	126	12.2%	548	53.9%	360	34.8%
Total	1549	14	0.9%	180	11.6%	806	52%	549	35.4%

Table 3 – Dataset 1 breakdown

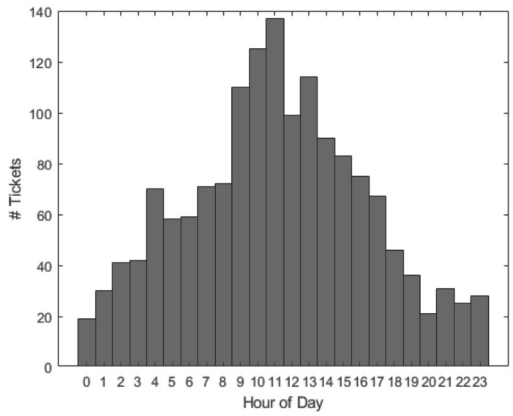

Figure 2 – Ticket creation per hour

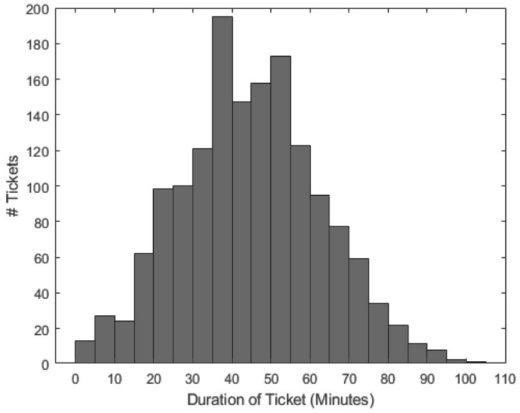

Figure 3 – Ticket duration

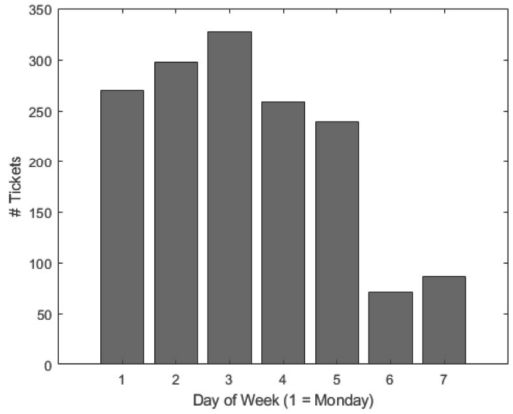

Figure 4 – Ticket creation per day

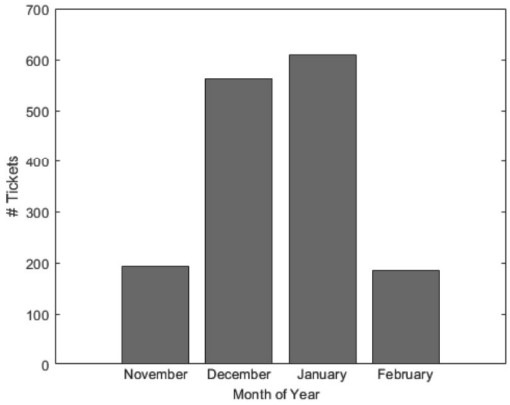

Figure 5 – Ticket creation per month

3.4. Simulation

To analyse the results of the scheduling algorithms on the generated data, a simulation was to closely reflect an incident management system. The reason for not using generic modelling tools such as Simulink [12] was due to the learning curve and to provide greater flexibility with tools and frameworks for later aims and objectives.

3.5. Experimental Setup

Algorithms will be tested against 3 different datasets that outline 3 different workload and scenarios. The remainder of this section will outline the parameters for the first dataset. Table 3 outlines the ticket information by type and severity over a 3-month period. Support agents have work hours from 9AM-6PM, Monday to Friday. Each agent has a 1-hour break that can be taken between 12-2PM but at least 1 agent must remain available. The setup cost when finishing a task and starting a new task is 5 minutes. This should not be confused with a context switch overhead cost; this cost is present regardless of context switching. The test will be run 3 times on the dataset with 2, 3 and 5 support agents. The reason for this is to factor resource utilization into the analyses of the algorithms.

4. Design

At the start of the project Java was chosen to implement the majority of the solution. This decision was made because of the extensive support and libraries available for building a User Interface (UI) such as JavaFx [13] and machine learning models. Although Java offers lower performance when considered against languages like C++, it does offer greater

portability between Operating Systems and platforms. With minimal changes the solution could run on serverless environment such as Azure Functions [14] or Google Cloud Functions [15] and be able to process real time data.

Figure 6 outlines the high-level architecture of the solution. The solution was split into 3 different components initially, to help separate responsibilities, concerns and encourage reuse. The inclusion of an intermediary queue naturally decouples the simulator and data generator. The benefit of this approach would allow for data generator to be swapped out for real time data, where the data could reside in a persistent queue such as Azure Service Bus [16].

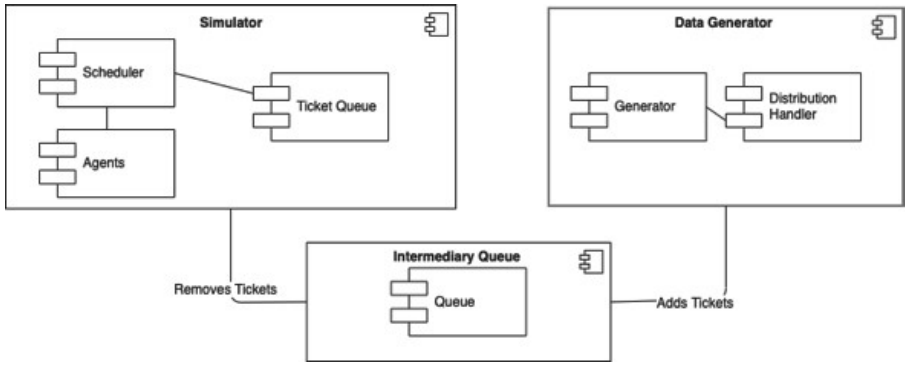

Figure 6 - High level architecture diagram

5. Implementation

Artifacts relating to the design and implementation of the tasks thus far are outlined in this section. Specifically, details relating to the algorithms, underlying architecture of the data generator, simulator and their corresponding user interfaces.

5.1. Component Construction

During the development of the data generator and simulator the main goal was to produce maintainable and reliable solutions, whilst keeping development time to a minimum. This is because these components are not the main focus of this project. To accomplish this these components share some core codebase and attempt to apply appropriate Software Development practices. Figures 7 and 8 outlines the general structure of the generator and simulator.

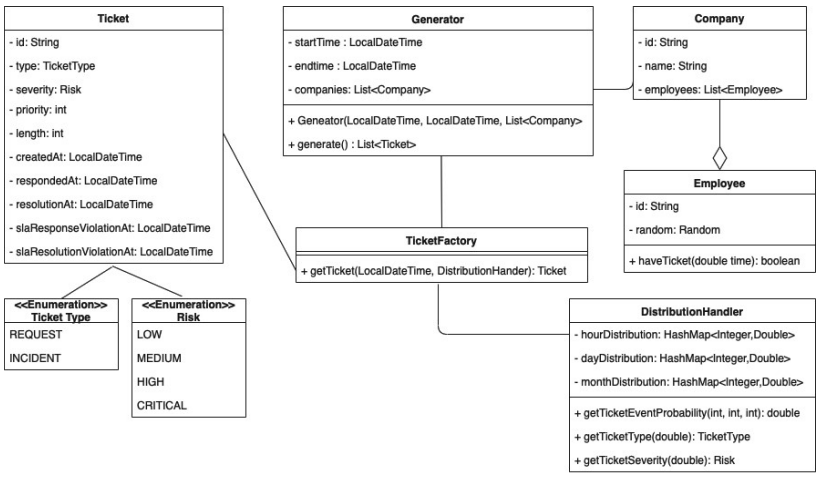

Figure 7 - Data generator class diagram

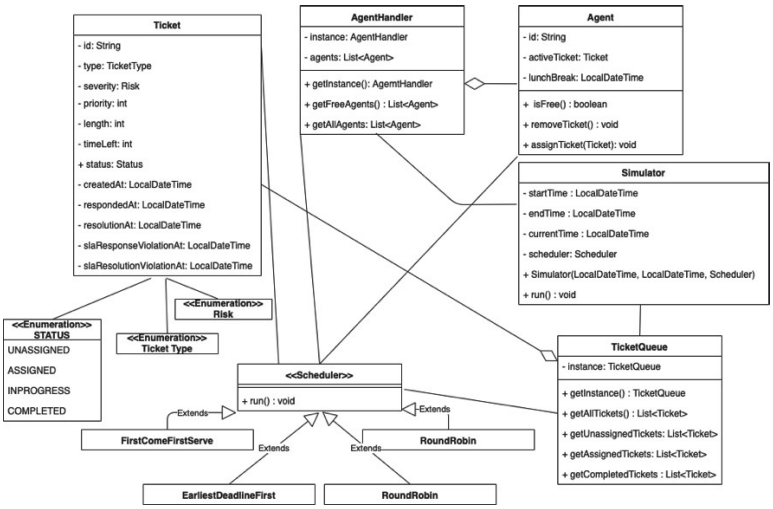

Figure 8 - Simulator class diagram

5.2. User Interface

The data generator user interface is still under construction and is only capable of showing graphs of the data produced. Figure 9 illustrates a high-fidelity version of the structure the user interface will follow once complete. The main screen tries to focus on the user provider parameters that will be taken into account when producing the data.

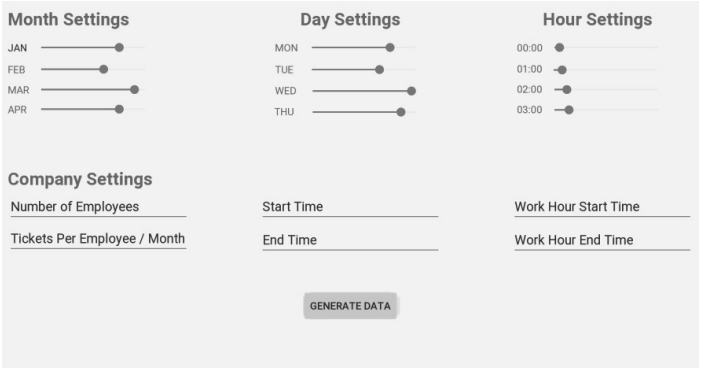

Figure 9 - Data generator UI

The simulator user interface is yet to be started, as these tasks planned much later in the project plan. Figure 10 outlines the structure and controls available to the user to control the state of the simulation. The design attempts to emphasis the state of the simulation i.e. the ticket queues and violations by allocating a large portion of the screen to display this information.

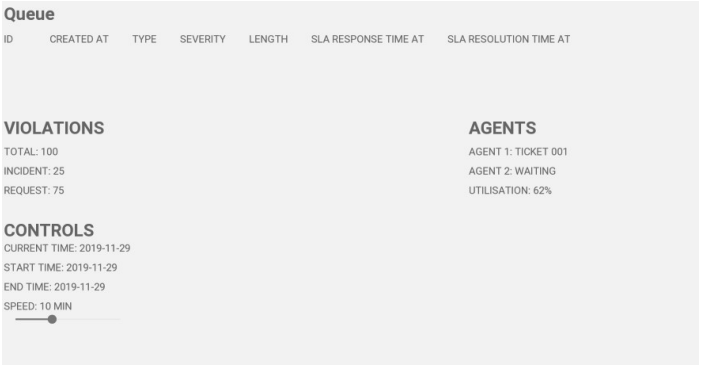

Figure 10 – Simulator UI

5.3. Algorithms

This sub section outlines the initial algorithms to be analysed from the Operating System scheduling paradigm. First Come First Serve First, Earliest Deadline First and High Severity Earliest Deadline First with Pre-emption were also implemented in [4] and are included to provide a baseline comparison with previous work.

5.3.1. First Come First Serve (FCFS)

FCFS is perhaps the simplest algorithm that we're going to implement. Tickets are assigned to agents based on their arrival time only and are worked on until completion. We decided to include this algorithm because it is relatively simple to implement and is a common algorithm used in environments such as checkout queuing. Another reason for its inclusion is that it helps establish a baseline with work performed by [4], which can help validate that our generated datasets are realistic.

5.3.2. Earliest Deadline First (EDF)

Unlike FCFS, EDF assigns tickets to agents based on the closest deadline. The deadline for tickets can be calculated based on the created time and the corresponding SLA that was outlined in section 3.2. Unlike the implementation in [4], this version does consider the severity of tickets indirectly, when the SLA is calculated. This is because a tickets SLA depends on the type of ticket and its severity. As a result, our implementation of EDF matches the ideas proposed in [4] for HSEDF.

5.3.3. High Severity Earliest Deadline First with Pre-emption (HSEDF-P)

The reasoning to include HSEDF-P was because with some small amendments to the implementation of EDF in section 5.3.2, we're able to produce the same scheduling behavior as High Severity Earliest Deadline First with Pre-emption (HSEDF-P) [4]. This is because of the same reasons highlighted in section 5.3.2. The results of HSEDF-P in [4] show a significant improvement in performance if overheads are low.

5.3.4. Round Robin (RR)

RR is a pre-emptive version of FCFS and uses a time slice known as a quantum to facilitate task switching. In the context of the problem this would involve working on a ticket for the length of the quantum and then switching to the next ticket in the queue. RR can reduce response time, although this does introduce an overhead. Our reasons for including RR in our analysis is that it is a trivial pre-emptive algorithm that can highlight the benefits and drawbacks of pre-emption in the context of this problem. One of our aims for its inclusion is to illustrate how sensitive the problem is to different quantum values and overheads.

5.3.5. Multi-level Feedback Queues (MLFQ)

MLFQ was included to further evaluate pre-emption and built on some of the suspected drawbacks RR may introduce. Much like RR, MLFQ uses a quantum however if a ticket exhausts its quantum it is moved down to a lower priority queue. This should reduce response times for newly created tickets. The drawback of this approach is resolution times can increase significantly because if a number of new tickets arrive, work cannot resume to older tickets until the new ones are dealt with. However, the effects of this could be reduced by increasing the priority of a ticket if it hasn't been worked on for a specified period of time. Due to multiple queues and the opportunity to use different algorithms in each queue the implementation of MLFQ will occur after the initial analyses.

5.3.6. Multi-level Feedback Queues with Thresholds (MLFQT)

MLFQT uses most of the same ideas as MLFQ but introduces Pre-emption Thresholds to reduce the amount of context switching but still maintain the same responsiveness as MLFQ. Much like MLFQ this will be implemented after the initial analyses.

6. Initial Analysis

During the initial analysis the following algorithms were considered; First Come First Serve (FCFS), Earliest Deadline First (EDF), High Severity Earliest Deadline First with Pre-emption (HSEDF-P) [4] and Round Robin (RR). Some algorithms were tested with different parameters, hence the reason for 6 results in the below figures. These include, HSEDF-P with an overhead cost of 5 minutes (HSEDF-P C5), HSEDF-P with an overhead cost of 25 minutes (HSEDF-P C25) and Round Robin with a 20-minute quantum (RR Q20).

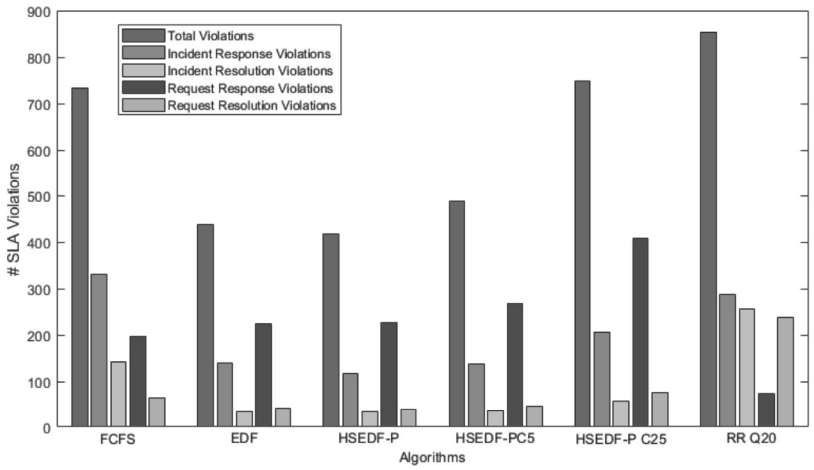

Figure 11 – Total SLA violations

The results in this section are with 2 agents, this is because when increasing the number of agent's, utilization dropped significantly, although violations did also drop. Increasing from 2 agents with a utilization of 62% to 3 agents reduced utilization to 27%. Because of this, it's clear that 2 agents are sufficient to represent the specific workloads outlined in dataset 1. When comparing the algorithms in Figure 11 its quite clear that EDF and HSEDF-P perform significantly better across both types of tickets. Pre-emption did not offer any significant improvements overall even when the cost of switching tasks was 0, this was the case for HSEDF-P. Upon further investigation of Figures 12 and 13, pre-emption does reduce response and resolution violations of tickets with critical and high severities.

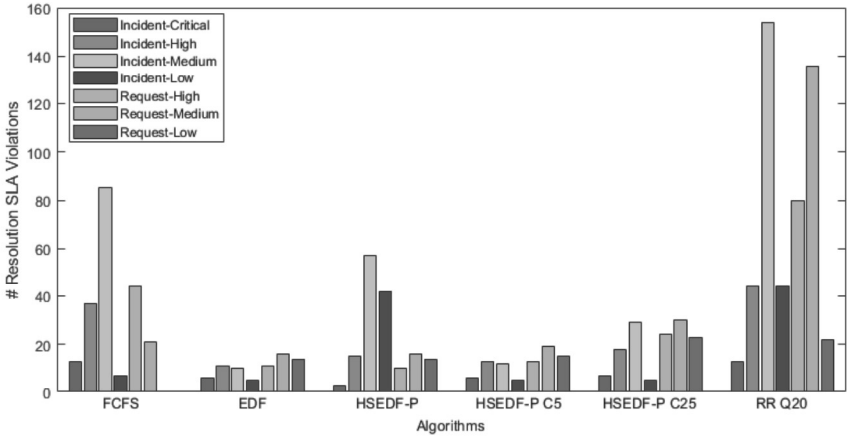

Figure 12 - Response SLA Violations

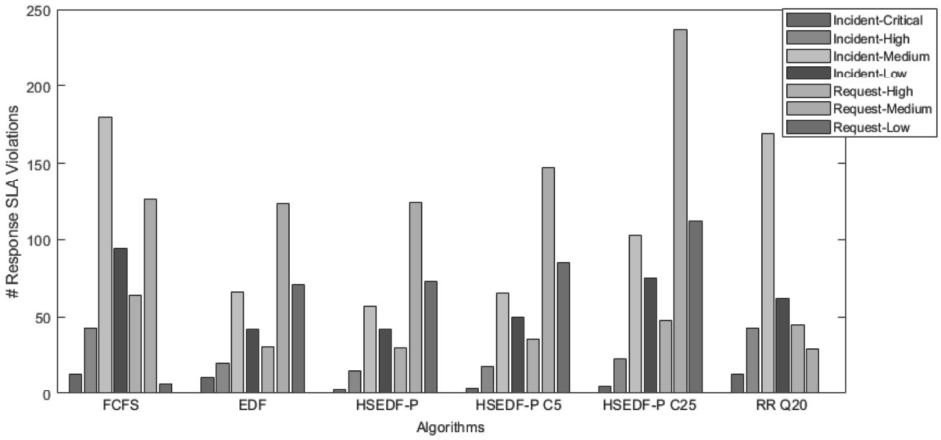

Figure 13 - Resolution SLA Violations

These results do reflect similarities with work performed by [4]. However, the results outlined in this section provide a somewhat are different view. This is because the authors in [4] aimed to reduce the number of violations of critical tickets, where there were only critical and low severity tickets. In the scenario behind the results of this section, and as outlined in section 3, there are 2 types of tickets, each with at least 3 different severity levels. Although EDF & HSEDF-P do reduce SLA violations, this is typically for higher severity tickets. The problem with this is critical and high severity tickets only account for 13% of all tickets in the generated dataset.

These results provide an insight into optimization areas as the project continues to progress. Scheduling policies for Multi-level Feedback Queues and Multi-level Feedback Queues with Thresholds (MLFQT) can be devised using the information gained from this analysis.

7. Progress

This section outlines the progress against the initial plan, including the obstacles, risks and a critical evaluation of how the project has been handled.

7.1. Project Management

Management of tasks has been done via an agile methodology which includes sprints and a Kanban style board, this was done through the use of Trello [17]. The length of sprints are two weeks, with a mid-sprint review with the project supervisor. Each task has a unique id in the form TASK-XXX, a label for categorisation and a due date. Tasks start off in the 'To-Do' column and move through 'In-Progress', 'In-Review' and 'Done', all of which are visible in Figure 14. The structure of the workflow allowed for the project to be broke up and structured into tasks that fit within the sprint size.

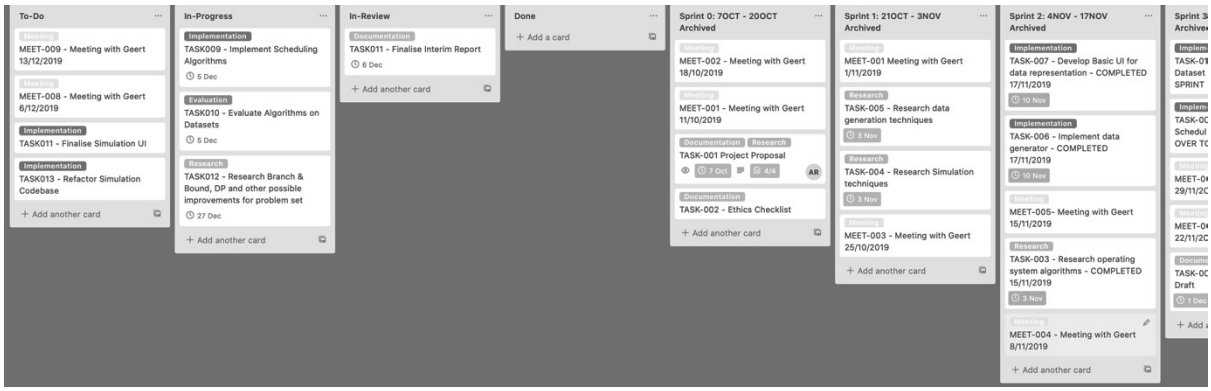

Figure 14 - Trello Board Task Management

In Sprint 0, it was identified that there was a risk where the initial Gantt chart underestimated how much time would be required for the interim and final reports. This was based on experience Initially 2 weeks were allocated for the interim report and 5 weeks for the final report. Before submitting the proposal, the project was slightly restructured so that work on the interim report could begin 1 week earlier and the final report could begin 3 weeks earlier whilst concurrently working on other tasks.

At the beginning of the project it was quickly identified that the available datasets were not appropriate for the aim's and objectives of the project. As a result, the time allocated to design and implement the simulator was underestimated. The addition of data generation added two weeks to the completion date of the task. This failure was due to insufficient analysis of the datasets found at the start of the project. Despite the setback, the plan was altered and some UI tasks, using JavaFx, were brought forward which were meant to be completed much later. The change was justified on the grounds that it provided the opportunity to get familiar with JavaFx and provide a foundation for later UI work. The added benefit of the change provided a visual representation of the data produced by the data generator.

Reflections on the project plan were not completed as often as they should've been during the first 4 sprints. During the initial planning, workload and coursework loads were taken into account between semesters. However, the plan was never updated to reflect actual coursework deadlines that were released after the initial plan. The initial plan only considered coursework weighting per module. Moving forward with the project the plan will be updated as soon as coursework deadlines are released.

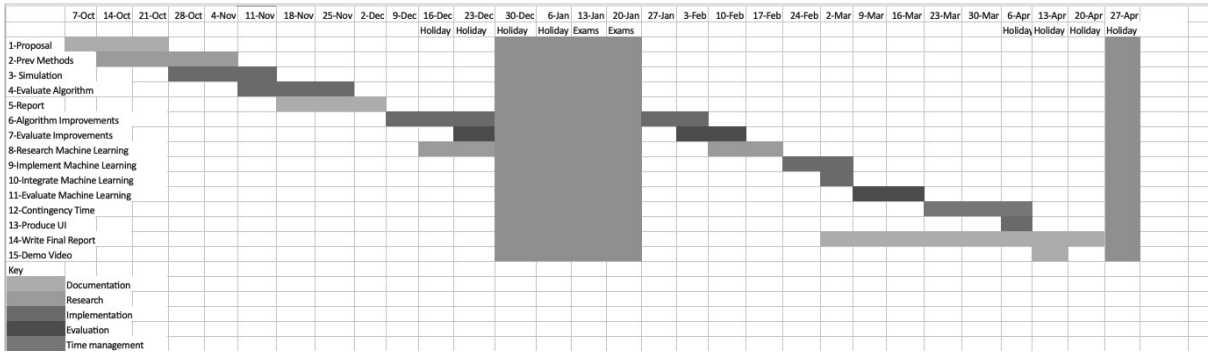

Figure 15 - Gantt chart produced with proposal

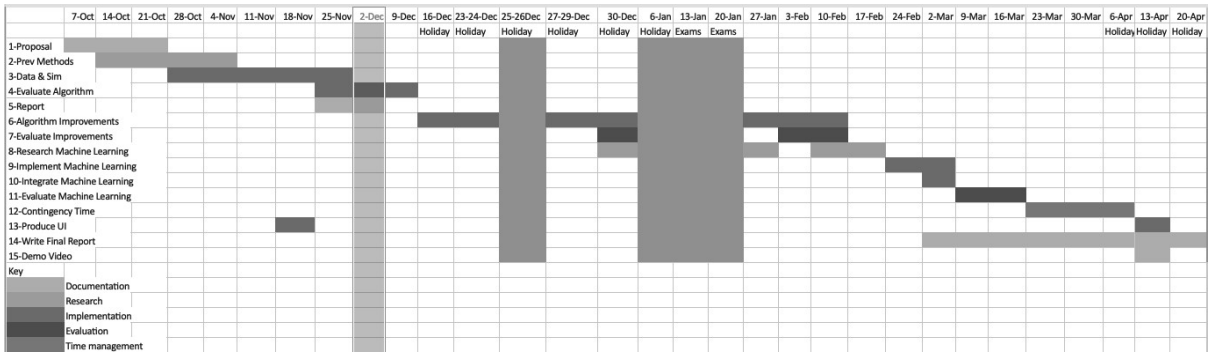

Figure 16 – Gantt chart with current progress

7.2. Contributions & Reflections

The progress made thus far highlight interesting properties of the scheduling problem proposed in the initial project proposal and the potential benefits of offering improved performance. Despite a setback of 2 weeks, the current contributions do provide an architecture to test and implement more novel algorithms going forward with significantly less effort.

The speed at which the project has progressed is unsatisfactory from a creative perspective. The project hasn't proposed any novel work as of yet. However, taking into consideration the proposed changes to management outlined in section 7.1 and the architecture that is now in place, the project can begin looking at more optimal techniques such as Branch and Bound and Dynamic Programming.

If the opportunity arose to restart the project, data analyses and generation would start during the proposal stages. Basic implementation and analyses of the algorithms would follow without the context of the simulator. This is because although there is a drawback in terms of efficiency to have to refactor the algorithms to work in the simulator, the knowledge gained about the problem could come to light much earlier in the project.

8. References

[1] C. Harris, "IT Downtime Costs $26.5 Billion In Lost Revenue," 24 05 2011. [Online]. Available: https://www.informationweek.com/it-downtime-costs-$265-billion-in-lost-revenue/d/d-id/1097919.

[2] D. Airlines, "January systems outage | Delta News Hub," 2019. [Online]. Available: https://news.delta.com/tags/january-systems-outage.

[3] R. Steinberg, ITIL Service Operation, 2nd Edition, TSO (The Stationery Office), 2011.

[4] M. Assuncao, V. Cavalcante, M. Gatti de Bayser , M. Netto, C. Pinhanez and C. Souza, "Scheduling with preemption for Incident Management: When interrupting tasks is not such a bad idea," in *Proceedings - Winter Simulation Conference*, 2012.

[5] B. Bailey, J. Konstan and J. Carlis, "Measuring the effects of interruptions on task performance in the user interface," in *IEEE International Conference on Systems, Man, and Cybernetics*, 2000.

[6] E. Logic, "Preemption-Threshold Scheduling Enables Real-Time Systems to Achieve Higher Performance," 10 2017. [Online]. Available: https://rtos.com/wp-content/uploads/2017/10/EL_PTS_Improves_Performance.pdf.

[7] M. T. Jones, "Inside the Linux Scheduler," June 2006. [Online]. Available: https://www.ibm.com/developerworks/library/l-scheduler/index.html.

[8] G. Rauchecker, E. Yasasin and G. Schryen, "A Decision Support System for IT Security Incident Management," 2014.

[9] SudhanshuSharma, "Analyze Helpdesk tickets | Kaggle," September 2018. [Online]. Available: https://www.kaggle.com/sudhanshu746/analyze-helpdesk-tickets.

[10] Microsoft and Endava, "support-ticket-classification," [Online]. Available: https://github.com/karolzak/support-tickets-classification.

[11] M. A. Farakh, "Lessons learned from analyzing 7 million customer support tickets," April 2019. [Online]. Available: https://www.jitbit.com/news/255-lessons-learned-from-analyzing-7-million-customer-support-tickets/.

[12] "Simulink," [Online]. Available: https://www.mathworks.com/products/simulink.html.

[13] "JavaFX," [Online]. Available: https://openjfx.io.

[14] "Azure Functions," [Online]. Available: https://azure.microsoft.com/en-gb/services/functions/.

[15] "Google Cloud Functions," [Online]. Available: https://cloud.google.com/functions/.

[16] "Azure Service Bus," [Online]. Available: https://azure.microsoft.com/en-gb/services/service-bus/.

[17] "Trello," [Online]. Available: https://trello.com.