**EEX5362 Performance Modelling**

## **Deliverable 01**

**Student Name**: T. Iresh Tharanga

**Registration No**: 121426740

**SID**: S92066740

Contents

[**Deliverable 01**](#_g5uiszsgdkd7) **1**

[1. System Identification](#_bty8ya5w6xd4) 3

[System Name](#_g82iltosdmmk) 3

[System Type](#_i0mf2bdymlke) 3

[System Description](#_psfklm4tbhxu) 3

[System Complexity](#_wdy7lm1of1sc) 3

[2. Performance Objectives](#_fo4isrdzudt) 3

[Primary Goal](#_jf3ysh5hwfn) 3

[Key Performance Indicators (KPIs)](#_vx4rifn8njjd) 4

[3. Data Set](#_qen4zo94fofa) 4

[Input Parameters (from config.json)](#_s3rrvej684pa) 5

[Generated Raw Data (Saved to results/ folder)](#_oogg4hq8cxry) 5

### System Identification

#### System Name

Multi-Threaded Task Scheduler

#### System Type

This is a classic multi-server queueing system (often represented as an M/M/c queue in queueing theory).

#### System Description

A multi-threaded task scheduler manages a finite pool of "worker threads" (resources) to process a continuous, random stream of incoming "tasks" (work).

Its operation is as follows:

1. **Arrival**: Tasks (e.g., HTTP requests, database queries) arrive at the system at random intervals.
2. **Queue**: If all worker threads are busy, the new task is placed in a First-In, First-Out (FIFO) queue.
3. **Service**: As soon as a thread becomes free, it takes the next task from the front of the queue and begins processing it.
4. **Completion**: After a variable amount of time (service time), the task is complete, and the thread becomes free to take another task.

#### System Complexity

The complexity arises from the interaction between the randomness of arrivals and the finite number of resources.

This system is complex enough for performance modeling because its behavior is non-linear. If tasks arrive just slightly faster than the threads can process them, the queue length and wait times do not just get "a little worse", they can grow infinitely, leading to system collapse. This simulation allows us to identify this "tipping point" and model critical performance characteristics:

* **Bottlenecks**: The thread pool itself is the bottleneck.
* **Resource Utilization**: We can measure how busy the threads are.
* **Latency**: We can measure the wait time experienced by each task.
* **Scalability**: We can test how adding more threads affects performance.

### Performance Objectives

#### Primary Goal

The primary objective is to quantify the relationship between system load, resource allocation, and user-perceived performance.

We aim to answer critical business and engineering questions, such as:

* *"If our user traffic increases by 30%, how many more threads do we need to maintain our current response time?"*
* *"What is the maximum number of requests per second our system can handle with 4 threads before performance becomes unacceptable?"*
* *"Is it more effective to make our tasks run faster (decrease service time) or add more threads (increase resources)?"*

#### Key Performance Indicators (KPIs)

To achieve this goal, we will focus on measuring the following four aspects of performance:

1. **Task Wait Time (Latency)**:

Objective: Minimize.

Definition: The time a task spends in the queue before execution begins. This is the most critical metric for user-facing systems, as it directly translates to "lag" or "loading time."

1. **Queue Length**:

Objective: Understand and Stabilize.

Definition: The number of tasks waiting in the queue at any given time. A queue length that fluctuates around a stable average is healthy. A queue length that trends upwards indefinitely is the primary indicator of an unstable, overloaded system.

1. **Thread Pool Utilization**:

Objective: Optimize.

Definition: The percentage of time that threads in the pool are busy processing tasks. While high utilization (e.g., 80-90%) seems efficient, 100% utilization is a sign of a bottleneck, as the system has no "slack" to handle random bursts in arrivals.

1. **Task Turnaround Time**:

Objective: Minimize.

Definition: The total time a task spends in the system (Wait Time + Service Time). This represents the full "end-to-end" time from the task's creation to its completion.

### Data Set

Dataset URL: <https://github.com/CobaltCopperfield/MThreadedTaskSchedulerSim/tree/main/results>

This project uses a model-driven simulation. As such, our "data set" consists of two parts: the Input Parameters that define the simulation and the Generated Output Data that is produced for analysis.

#### Input Parameters (from config.json)

Instead of a static file, our input is a set of parameters that configure the statistical models for the simulation.

* **Task Arrival Process (The Load)**:
  + Model: A Poisson Process. This is a standard model for events arriving independently and at a random-but-predictable average rate.
  + Parameter: arrival\_rate. Defines the average number of tasks arriving per time unit (e.g., 0.7). The time between arrivals is modeled by an Exponential distribution.
* **Task Service Process (The Work)**:
  + Model: A Normal Distribution. This models the natural variation in task complexity (e.g., most tasks take 5 seconds, but some take 4.5 and some take 5.5).
  + Parameter: avg\_service\_time. Defines the average time it takes a single thread to complete one task.
* **System Configuration (The Resources)**:
  + Model: The multi-server queue.
  + Parameter: num\_threads (c). The number of parallel workers available to service the queue.

#### Generated Raw Data (Saved to results/ folder)

The simulation generates its own raw data sets and saves them as CSV files for each scenario, allowing for full transparency and further analysis.

* **Per-Task Transaction Log** (e.g., *tasks\_1\_baseline\_4t\_0\_7\_load.csv*)
  + - Description: A detailed log created for every single task that is processed during the simulation. This is the primary data set for analyzing latency.
    - Schema (Columns):

***task\_name***: A unique identifier (e.g., "Task-123").

***arrival\_time***: The simulation time when the task entered the system.

***start\_time***: The simulation time when the task began execution (i.e., acquired a thread).

***service\_time***: The actual time required to process the task.

***completion\_time***: The simulation time when the task finished.

***wait\_time***: The time spent in the queue (start\_time - arrival\_time).

***turnaround\_time***: The total time spent in the system (completion\_time - arrival\_time).

* **Queue State Time Series** (e.g., *queue\_1\_baseline\_4t\_0\_7\_load.csv*)
  + - Description: A snapshot of the queue length, taken at a regular interval (e.g., every 1.0 time unit). This data is used to analyze queue stability and visualize bottlenecks.
    - Schema (Columns):

***time***: The simulation timestamp when the measurement was taken.

***queue\_length***: The number of tasks waiting in the queue at that exact time.

*End of document*