Distributed Systems CW1

#### Student Number: 100571823

# Aim

In distributed systems, resource sharing among concurrent processes is a fundamental operation with varying complexity. The model investigated in this report simulates a distributed resource-sharing scenario, where processes request, hold, and release shared resources to complete tasks. This setup is inspired by a real-world two-phase locking system, where efficient resource management and avoidance of deadlocks are crucial for maintaining the system.

The aim of this report is to explore and quantify the impact of the key parameters within the model, focusing on how the:

* **Number of concurrent processes:** The total number of processes actively competing for resources. This parameter tests the model’s scalability, as a higher number of concurrent processes typically increases competition for resources and the likelihood of contention.
* **Availability of shared resources:** The total count of resources within the system that processes can request. This parameter is varied to examine how resource availability influences task completion rates and deadlock events.
* **Likelihood of a process requesting a resource:** This parameter represents the likelihood that a process will request a resource during a given step. Varying this parameter enables observation of how aggressive resource requests impact system throughput and deadlock.
* **Release chance:** This parameter affects how likely a process is to release its acquired resources after completing a task. By adjusting this probability, the study examines whether frequent resource release improves availability and reduces bottlenecks.

Each of these parameters is incrementally adjusted in controlled experimental runs to determine their specific impact on model performance. By systematically testing different levels of each parameter, this study aims to identify optimal configurations that balance resource availability with process demand.

# Methods and Results

## The experimental setup focused on incrementally adjusting four main parameters as stated previously. Each parameter was varied individually while holding the others constant, providing clear insights into the impact of each variable on task completion and deadlock frequency. The experimental runs were divided into four main tests, with each test run for 800 ticks or until a deadlock occurred:

## Process Linear Increase

### Method:

In this first set of tests, the number of processes (i.e., concurrent turtles) was increased incrementally, from a low baseline up to a maximum of 100. By holding the other parameters constant, this test group isolated the effect of process count on both the system’s throughput (tasks completed) and deadlock occurrences. This scalability test aimed to understand how increasing competition among processes influences the likelihood of resource contention and system bottlenecks. The exact task parameters are as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Process Count** | **Resource Count** | **Request Chance** | **Release Chance** |
| 10 | 9 | 0.5 | 0.5 |
| 30 | 9 | 0.5 | 0.5 |
| 50 | 9 | 0.5 | 0.5 |
| 80 | 9 | 0.5 | 0.5 |
| 100 | 9 | 0.5 | 0.5 |

### Results

As the process count increased, task completion rose steadily, peaking at 178 tasks before deadlock with 100 processes. However, efficiency per process decreased at higher counts.

## Process and Resource Linear Increase

Method:  
In the second set of tests, the experiment expanded to examine how the simultaneous increase of both the process count and resource availability would impact task completion and the occurrence of deadlock. By proportionally increasing resources alongside processes, this test aimed to explore how resource sufficiency might mitigate the risk of contention and improve system throughput in a distributed setup. The following parameters were applied across each test increment:

|  |  |  |  |
| --- | --- | --- | --- |
| **Process Count** | **Resource Count** | **Request Chance** | **Release Chance** |
| 10 | 9 | 0.5 | 0.5 |
| 30 | 29 | 0.5 | 0.5 |
| 50 | 49 | 0.5 | 0.5 |
| 80 | 79 | 0.5 | 0.5 |
| 100 | 99 | 0.5 | 0.5 |

Results:  
Increasing both processes and resources proportionally led to greater task throughput, with the highest completion rate of 199 tasks before deadlock at maximum levels, although the gains showed diminishing returns per process.

## Request Chance Increase

### Method:

In this third set of tests, the primary variable was the request chance, which controlled the likelihood of each process requesting a resource at any given moment. By keeping the process and resource counts steady at 50 and 49, respectively, this test isolated the effect of varying request frequency on task completion and deadlock. The purpose was to understand how increased request aggressiveness impacts system performance, potentially leading to higher task throughput or, conversely, increased deadlock risk. The test parameters were as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Process Count** | **Resource Count** | **Request Chance** | **Release Chance** |
| 50 | 49 | 0.1 | 0.5 |
| 50 | 49 | 0.25 | 0.5 |
| 50 | 49 | 0.5 | 0.5 |
| 50 | 49 | 0.75 | 0.5 |
| 50 | 49 | 0.99 | 0.5 |

### Results:

Task completion was optimised at moderate request chances, reaching up to 109 tasks, but dropped slightly at very high request levels due to increased contention.

## Release Chance Increase

### Method:

In this final set of tests, the release chance parameter was the focus, controlling the likelihood of a process releasing a resource after completing a task. The release chance was varied incrementally while maintaining a steady process count of 50 and a resource count of 49, along with a fixed request chance of 0.5. This setup aimed to examine how different release probabilities impact the efficiency of resource turnover and the overall task completion rate before deadlock occurs. The exact parameters are outlined below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Process Count** | **Resource Count** | **Request Chance** | **Release Chance** |
| 50 | 49 | 0.5 | 0.1 |
| 50 | 49 | 0.5 | 0.25 |
| 50 | 49 | 0.5 | 0.5 |
| 50 | 49 | 0.5 | 0.75 |
| 50 | 49 | 0.5 | 0.99 |

### Results:

Lower release chances led to higher task completion rates, achieving up to 643 tasks, while higher release frequencies reduced efficiency as competition for resources intensified.

## Discussion:

The experiments provide essential insights into how each parameter (process count, resource count, request chance, and release chance) affects task throughput and deadlock likelihood in distributed systems, helping to identify an optimal setup for maximum efficiency and stability.

Increasing both process and resource counts together proved more effective than adjusting the process count alone, with balanced increases allowing for significantly higher task throughput before deadlock. This balance reduced resource contention, suggesting that scalability in distributed systems relies on proportionally aligning resources with process demands. However, a limit emerged, as too many concurrent processes still led to bottlenecks, indicating that careful calibration of both process and resource counts is necessary to avoid overloading the system.

Notably, a direct increase in resources did not significantly improve task completion counts, as demonstrated in the chart below:

**Request chance tests** demonstrated that moderate values (around 0.5 to 0.75) enabled the most efficient task completion, likely due to sustained but manageable demand on resources. When the request chance approached 1.0, task completion rates declined slightly, as higher request frequencies amplified competition without yielding a proportional increase in throughput. This indicates an optimal range for request frequency that balances activity without overwhelming the system, ensuring resources are utilised effectively without undue strain.

The **release chance parameter** had the most substantial impact on resource concurrency and deadlock frequency. At lower release chances, processes retained resources longer, resulting in notably higher task completion. This behaviour suggests that holding resources for longer periods temporarily reduces competition, delaying deadlock and increasing overall throughput. However, excessively low release chances risk stalling resource availability in systems where task requirements fluctuate more. Therefore, a moderately low release chance optimises turnover without excessive contention, enabling processes to complete tasks efficiently while maintaining enough resource availability to prevent stagnation.

In summary, the optimal configuration for this distributed system involves balancing process and resource counts, setting request chance within a moderate range (0.5–0.75), and maintaining a slightly lower release chance. This setup enhances performance by enabling high task throughput and minimising deadlock, providing a stable structure well-suited for distributed resource-sharing environments.

# Conclusion

To conclude, this report aimed to evaluate how various parameters affect concurrency and deadlock occurrence in a distributed system. The experiments highlighted the impact of process count, resource count, request chance, and release chance on system performance.

The findings demonstrated that increasing resources and processes proportionally is crucial for efficiency per process, underscoring the importance of balance between the two for optimal scalability.

The most pivotal factor identified during testing was the **request chance variable**. Moderate levels yielded the highest results, showing that systems run best when resources are freed regularly, reducing bottlenecks. Meanwhile, the **release chance** significantly impacted task completion and deadlock frequency. A lower release rate was more effective, reducing competition for resources and delaying deadlock occurrence.

In summary, the findings underscore the need for precise calibration of parameters in distributed systems to achieve high task completion while minimising the risk of deadlocks. By adjusting these parameters effectively, it is possible to optimise configurations within distributed systems, ensuring stable and efficient performance.