**Task Forge**

**Development of a Distributed Client-Server Simulation Execution Framework**

**Candidat: Bogdan-Sergiu BOLOȘ**

**Coordonator: ș.l.dr.ing. Alexandru IOVANOVICI**

Sesiunea: Iunie 2024

# **REZUMAT**

În cadrul acestei lucrări va fi prezentat un sistem client-server, intitulat Task Forge, ce are în vedere execuția distribuită a unor simulări, conceput pentru a facilita gestionarea și execuția eficientă a acestora. Sistemul are două componente principale: partea de Client și partea de Server.

Partea de server este alcătuită dintr-o aplicație Flask care gestionează trimiterea sarcinilor, actualizarea de status, încărcarea și descărcarea fișierelor. Sarcinile de simulare sunt procesate folosind fire de lucru care execută sarcinile și utilizează funcții utilitare.

Partea de client oferă o interfață grafică utilizatorului care permite trimiterea sarcinilor de simulare și monitorizarea statusului acestora, la fel descărcarea rezultatelor.

Datele sarcinilor sunt gestionate într-o bază de date MongoDB, asigurând scalabilitatea în ceea ce privește gestionarea sarcinilor. Acest sistem permite execuția asincronă a sarcinilor, distribuită pe mai multe servere, sporind eficiența și fiabilitatea sarcinilor complexe de simulare.

**ABSTRACT**

This paper is going to be focused on the client-server system, titled Task Forge, which focuses on the distributed execution of simulation tasks. It is designed to facilitate efficient task management and their execution. The system is made of two main components: the Client side and the Server side.

The server side represents a Flask application which manages task submission, status updates, as well as file uploads and downloads. These tasks are processed using worker threads which execute the simulations using utility functions.

The client side provides a graphical user interface which allows the user to send simulation tasks as well as monitor them and download the results once the task is ready.

Task data is managed in a MongoDB database, ensuring scalability in task handling. This framework enables the asynchronous execution of the tasks, which is distributed across multiple servers, enhancing the efficiency and reliability of complex simulation tasks.

# **INTRODUCTION**

* 1. **Context**

Task Forge represents a robust client-server application developed to streamline the distributed execution of simulations tasks. The system is designed to handle the complexities associated with task management and execution in a distributed environment by leveraging the strengths of both client and server components.

The server side of the application is built using the *Flask* framework. *Flask* is a popular web framework for Python. The server component is responsible for managing the lifecycle of simulation tasks, from submission and status updates to handling file uploads and downloads. Tasks are processed through the worker threads, which utilize a suite of utility functions to execute simulations. The multi-threaded approach ensures that multiple tasks can be handled concurrently, thus improving the throughput and efficiency of the system.

The client side of the application provides a user-friendly graphical interface. This interface allows users to submit new simulation tasks, monitor their progress, and download the results once they are completed. The graphical user interface is built using *PyQt5*, offering a comprehensive set of tools for developing desktop applications.

Central to Task Forge is its use of *MongoDB* for task data management. *MongoDB* is a NoSQL database known for its scalability and flexibility. This makes *MongoDB* an ideal choice for managing the large volume of data associated with simulation tasks. By storing the task data in *MongoDB*, the application ensures that tasks can be managed and retrieved efficiently, supporting the system’s overall scalability.

The architecture supports asynchronous task execution, distributed across multiple servers. This design enhances the system's efficiency and reliability, ensuring that the complex simulation tasks are processed in a timely manner. The distribution of tasks across several servers makes Task Forge a powerful tool for users needing to execute and manage extensive simulation tasks.

* 1. **Motivation**

The development of this application was driven by the need to automate the process of executing simulation tasks, which previously required significant and frequent user intervention. For instance, users had to manually connect to a powerful computer via a remote desktop connection, initiate simulations, and subsequently transfer the results back to their own devices. This process was not only time-consuming but also prone to human error, leading to inefficiencies and potential inacurracies in simulation outcomes.

Task Forge addresses these challenges by automating and distributing the simulation process. The application eliminates the need for manual remote connections and file transfers, replacing them with a streamlined, user-friendly interface. Users can now submit simulation tasks through a graphical interface, monitor their progress, and download the results upon completion—all without the need for direct interaction with remote machines.

The motivation behind Task Forge is to provide a more efficient, reliable, and scalable solution for executing simulation tasks. By automating the process and distributing tasks across multiple servers, Task Forge significantly improves the workflow for users, allowing them to focus on analysis and interpretation of simulation results rather than managing the execution process. This transformation from a manual to an automated system represents a substantial leap in efficiency and reliability, ultimately enhancing productivity and accuracy in simulation-driven projects.

* 1. **Problem Statement**

Throughout this paper, I address the challenges and inefficiencies in the traditional manual process of executing simulation tasks. I will also delve into the architecture and functionalities of the client-server system by discussing its design, implementation and benefits. This paper aims to demonstrate how the system significantly enhances the efficiency, reliability, and scalability of simulation task management, ultimately transforming the workflow for users.

# **REQUIREMENTS AND SPECIFICATIONS**

**2.1 Theoretical Elements**

Throughout the development of this project, I identified several key functional and non-functional requirements to ensure the system meets the user needs effectively.

**2.1.1 Functional Requirements**

The client-side application needs to offer a graphical user interface that allows users to select the desired *tool* version, load simulation packages, submit simulation tasks, view logs of submitted tasks with their statuses, and download simulation results or receive notifications if results are unavailable.

On the server side, the system should handle task submissions, status updates, file uploads and downloads, track which server executes each task, and resume simulations after any workstation restarts. Additionally, it should allow users to redownload simulation results if needed.

**2.1.2 Non-functional Requirements**

The system must be scalable to handle many simulation tasks at once by distributing them efficiently across multiple servers. It should be reliable, ensuring tasks are completed even if there are system failures or restarts.

Performance is crucial and the system should be able to run multiple simulations in parallel depending on their size. The GUI must be easy to use, providing clear feedback and updates to users.

**2.1.3 System-level Requirements**

The project should include a mechanism to allocate simulation tasks to the least busy server, ensuring tasks are evenly distributed and completed on time. It should also maintain logs of all submitted tasks and track their execution status and locations.

**2.1.4 Hardware-level Requirements**

The system requires multiple servers to distribute the computational load, enhancing performance and reliability. Adequate storage capacity is necessary to manage simulation results, with a system to delete older results when storage is full and the ability to redownload results if needed.

**2.1.5 Software-level Requirements**

The server-side application should be built using Flask, which provides a robust framework for handling HTTP requests. MongoDB should be used to store and manage task data efficiently. The client-side application should be developed using PyQt5 to provide a rich, user-friendly interface. Integration with Artifactory is needed to manage and download different versions of *tool* required for simulations.

**Teorie despre tehnologii?**

**2.2 List of Requirements with Justifications**

For the system-level requirements, Task Forge needs a mechanism to allocate simulation tasks to the least busy server to ensure efficient distribution and timely completion. This mechanism should balance the load based on the size of the tasks and the current workload of each server. Maintaining logs of task submissions and their statuses, as well as tracking where each task is executed, provides transparency and accountability in task management.

At the hardware level, multiple servers are required to handle the distributed task execution enhancing performance and reliability. Adequate storage management is essential to prevent overflow and ensure the availability of simulation results. This includes deleting older results when necessary and allowing users to redownload results if needed.

On the software side, the server application should be developed using Flask, as it offers a solid foundation for managing HTTP requests and task operations. MongoDB integration ensures efficient management of task data, supporting scalability. The client application, built with PyQt5, provides an intuitive and responsive GUI for users. Lastly, integrating with Artifactory allows for easy management and downloading of necessary *tool* versions for simulations.

By meeting these requirements, Task Forge aims to provide an efficient, reliable, and user-friendly solution for managing and executing simulation tasks in a distributed environment, significantly improving the workflow for users.