Project Presentation
CS744 Autumn 2024

SmartScheduler

TEAM NAME: BETTER CALL SAUL

MEMBERS: FARHAN JAWAID (24M0801), RAVI SAH (24M0854)

GIT LINK: HTTPS://GITHUB.COM/FARRU2610/CS744_PROJECT

Context

- **What is the area/domain of the project?**

  This project revolves around task scheduling in a multi-client environment, involving efficient task distribution among workers, ensuring load balancing, and processing multiple requests in parallel.

- **Why?**

  The need for distributed task scheduling arises in high-demand environments where jobs/tasks need to be processed efficiently, especially when tasks are computationally intensive which takes more cpu computations or other mathematical operations.

Problem Description

- **Statement:**

  We aim to build a distributed task scheduler system that can handle multiple clients submitting tasks. The scheduler needs to distribute tasks to available workers, monitor their progress, return correct output to each client, ensuring load balancing.

- **Scope/Goals/Deliverables:**

  - Task Queue Management: Design an efficient queue to hold tasks and distribute them to workers.

  - Worker Availability: Create a system for monitoring worker availability periodically and distributing tasks in a round-robin manner.

  - Client-Server Communication: Implement communication between clients, workers, and the scheduler.

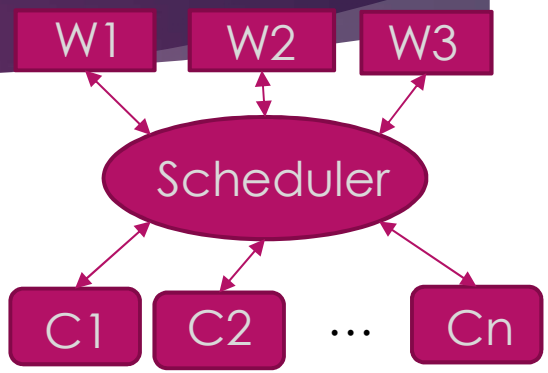
  - Task Execution: Workers will compute the correct results and send them back to clients.

  - Performance Monitoring: Evaluate system performance by simulating multiple client requests with different workers.

Components of the Project

- **Scheduler (Server):** Manages task queue, assigns tasks to workers, handles worker failures, and sends results back to clients.
- **Worker:** Executes tasks (for now, factorial calculation) and sends the result back to the scheduler.
- **Client:** Sends tasks (for now, factorial request of a given number) to the scheduler and receives results.
- **Queue Management:** A queue system to manage the tasks when workers are down.
- **Load Balancer:** Balances the load by distributing tasks to available workers in a round-robin fashion.
- **Performance Monitor:** A mechanism to evaluate system performance under different loads.

Design



- **1. Task Flow:**

  - Clients submit tasks to the scheduler.

  - Scheduler checks if any workers are available.

  - If workers are available, the scheduler assigns the task to a worker.

  - Worker computes the result and sends it back to the client via the scheduler.

- **2. Components Interaction:**

  - Clients ↔ Scheduler (Via TCP Stream) : Clients send tasks, scheduler returns results.

  - Scheduler ↔ Worker (Via TCP Stream) : Scheduler assigns tasks to workers; workers compute and send back results.

  - Queue Management: Task queue stores tasks until a worker is available.

Implementation Details

- **Scheduler:**

  - Implemented using **epoll** for efficient connection handling.

  - Load Balances between workers using Round Robin Algo.

  - Task queue management with a circular queue to store tasks.

  - Each task is binded to its Client's socket ID and Task to uniquely identify request and response associated with a task.

  - Differentiates between Worker and Clients by their type 'W' or 'C'.

  - Periodic worker status check to ensure active workers.

  - Removes worker when no longer responding.

Implementation Details Contd...

- **Workers:**

  - Workers connect to the scheduler and await task assignments.

  - Workers process tasks using **multi-threading** to handle **concurren**t requests efficiently.

  - Each worker processes tasks from the queue and sends results back to the scheduler.

- **Clients:**

  - Clients send tasks (for now, factorial calculation) and send them to the scheduler using its IP Address and Port via TCP Stream.

  - Clients are unaware about the internal relationship between Scheduler and Worker.

Evaluation

- **Setup:**

  - Test Environment: Multiple clients simulating load on the system.
  - Workloads: Factorial calculation tasks submitted by clients.

- **Parameters/Configuration:**

  - Number of Clients: Varying number of clients to simulate different loads. (5 to 2000)
  - Number of Workers: Limited number of workers to process the tasks. (eg. At max 5)

- **Metrics:**

  - Throughput: Number of tasks processed per second.
  - Latency: Time taken to complete the tasks of entire batch.

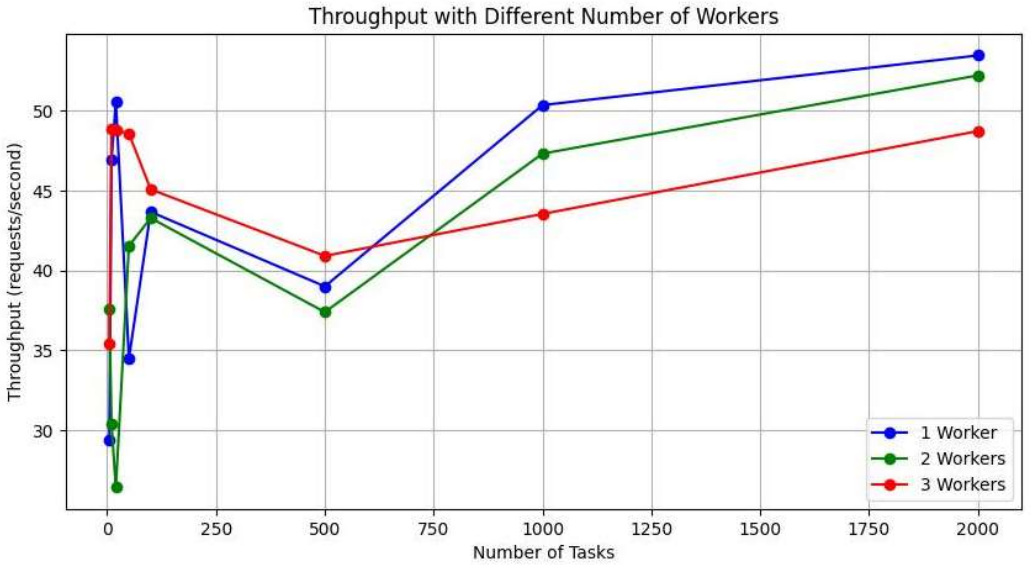
Task Processing Performance Under Different Loads

Experiment 1: Finding Throughput with Different workers

Observations:

For a small number of tasks, the throughput across different numbers of workers remains almost the same.

However, as the number of clients increases, the throughput initially improves with the addition of more workers. After a certain point, the throughput for higher worker counts begins to decrease compared to fewer workers.

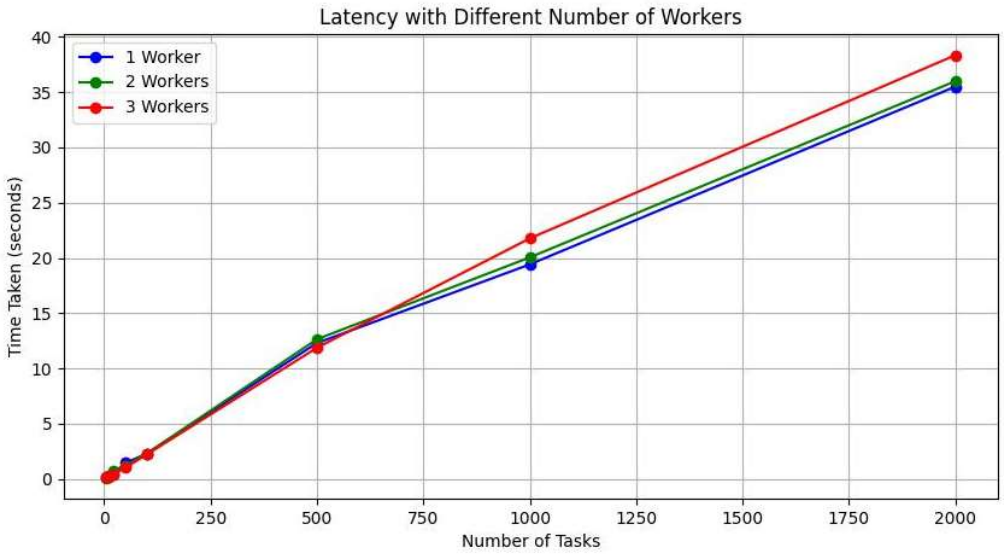


Experiment Contd…

Experiment 2: Finding Latency with Different workers

Observations:

-The response time for entire batch increases in proportion to the number of clients for any number of workers.



Summary of Results

- Ideally, with increase in the number of worker nodes, Throughput should increase and Latency should decrease. But in our case the behavior is slightly different than ideal case. The reason for this anomaly can be associated with:

- **Network Overhead**

  - As the number of workers increases, communication between the scheduler and workers intensifies. This can create network bottlenecks, where the time spent transmitting task data grows, leading to delays in task assignment and result reporting. This reduces the overall throughput as workers wait for tasks and results, causing lag in the system's performance.

- **Scheduler Bottleneck**

  - With a higher number of clients, the scheduler's task allocation speed may not keep up with the rate of incoming tasks, causing delays in task processing and impacting throughput. When too many tasks are queued up, the scheduler becomes a bottleneck. It can only assign tasks to workers as they become available.

Unfinished Scope

- **Scalability:** Improve the system's ability to handle more clients and workers.
- **Fault Tolerance:** Enhance worker failure recovery mechanisms.
- **Task Prioritization:** Add support for prioritizing certain tasks over others based on client needs.
- **Bottleneck:** Decentralize the Scheduler as it can become bottleneck for entire system.

Challenges

- **Worker Availability:** Managing worker disconnections and ensuring new workers are added dynamically.
- **Task Queue Management:** Ensuring efficient queue management with availability or non-availability of workers.
- **Load Balancing:** Balancing the load effectively to prevent any worker from being overwhelmed.

Reflection

- **Interesting Aspects:**

  - Understanding task scheduling algorithms and optimizing them for multi-client systems.

  - Implementing real-time monitoring and worker management with various workloads.

- **What I Would Have Done Differently (If had more time):**

  - Would have added a more sophisticated load balancing mechanism **(Least Loaded)** to handle more complex tasks and higher loads.

  - Prioritization of complex tasks.

  - More detailed performance analysis.

Conclusions

- The SmartScheduler system successfully demonstrates a multi-client, multi-worker environment where tasks are efficiently scheduled and processed.
- With improvements in scalability and fault tolerance, the system can be extended for more complex workloads.

References

- https://github.com/farru2610/cs744_project
- Wikipedia - Task Scheduling Algorithms: https://en.wikipedia.org/wiki/Scheduling_(computing)
- Last but not least, one of our dearest friend: https://chatgpt.com/