- Task 2: Software Architecture:

Part A: How would you design a web crawler for a cluster of crawlers?

- Helpful questions you should try to address in your solution:
- How do you ensure that many crawlers don't crawl the same URLs?
- Where do you store the results?
- How would you design the system to be efficient and robust?
- What are the advantages and limitations of your proposed solution?

- Ensuring that many crawlers don't crawl the same URLs:

Utilizing a distributed lock is one approach to do this. Only one process may hold a lock on a certain resource at once thanks to a distributed lock mechanism. In this situation, a distributed lock may be used to guarantee that only one crawler at a time can access a specific URL.
We could use a distributed lock implementation such as Redis or ZooKeeper to ensure that many crawlers don't crawl the same URLs.

Utilizing a bloom filter is another approach to make sure that several crawlers do not access the same URLs. A space-effective probabilistic data structure called a bloom filter may be used to determine if an element is a part of a set. In this instance, we may track the URLs that have previously been crawled using a bloom filter.
We could use a bloom filter implementation such as Redis or Apache Guava to track the URLs that have already been crawled.

- Where to store the results:

The crawl's findings can be kept in a database, distributed file system, or key-value store, among other locations. The optimal option will rely on the application's particular requirements.
We could use a database such as PostgreSQL or MySQL to store the results of the crawl

- How to design the system to be efficient and robust:

The usage of a distributed queue is one technique to improve the system's effectiveness. A distributed queue is one that allows numerous programs to access it simultaneously. As a result, the crawlers are able to take URLs from the queue and crawl them simultaneously.

Using a fault-tolerant design is another technique to strengthen the system. This implies that even if any of the crawlers malfunction, the system should still be able to function. Replication can be used to accomplish this. We may duplicate the distributed queue and the bloom filter, for instance, across other servers.

We could use a distributed queue implementation such as Kafka or RabbitMQ to distribute URLs to the crawlers.

- **The advantages and limitations of my proposed solution:**

The benefits of my suggested course of action are:

- **Scalability**: The number of crawlers in a cluster may be increased or decreased to suit the demands of the application.
- **Performance**: A group of crawlers can search the web far more quickly than one crawler can.
- **Robustness**: Compared to a single crawler, a cluster of crawlers is more resilient to failures.

The following are my suggested solution's drawbacks:

-
- **Complexity**: The design and implementation of a cluster of crawlers is more difficult than that of a single crawler.
- **Operating costs**: Running a group of crawlers may be more expensive than running a single crawler.

Part B: The data collected by the web crawler you designed above is to be used for analytics purposes. There are multiple use cases, but most of the analytics will be done on the page content itself. Some of the analysis will be complex and long-running. Analysis should not affect the performance of the crawler itself. Your task is to suggest a high level architecture of such a system.

Questions you should try to address:

- How would the crawler communicate with the analytics system?
- What tradeoffs between consistency and availability do you feel you can make?
- What are the benefits and drawbacks of keeping all the data in one database?
- What are the benefits and drawbacks of keeping the data in separate databases?
- If the system has its own database, what precautions would you need to take to make sure the data is consistent across the two databases?

- **Communication:**

  A message queue can be used to facilitate communication between the web crawler and analytics system. The analytics system can consume messages from the queue to perform analytics on the pages, and the crawler may post messages to the queue containing the crawled pages.
  The web crawler and the analytics system can communicate with each other using a variety of message queues, such as Kafka or RabbitMQ. The message queue should be

able to handle a high volume of messages, as the web crawler will be crawling the web continuously.
The analytics system can use any database to store the results of the analytics. However, it is important to choose a database that is scalable and reliable. The database should also be able to handle complex and long-running analytics queries.

- **Tradeoffs between consistency and availability:**

  The utilization of synchronous or asynchronous communication between the crawler and the analytics system is one trade-off to take into account. When using a synchronous communication channel, the crawler would stop when the analytics system had finished analyzing the page. Consistency would be guaranteed, but the crawler's performance would suffer as a result.

  The crawler wouldn't wait for the analytics system to finish analyzing the page before moving on if there was an asynchronous communication method in place. This would enhance the crawler's performance, but if the analytics system malfunctions, it may also cause discrepancies.
  In this case, I would recommend using an asynchronous communication mechanism. This is because the performance of the crawler is more important than consistency for most analytics use cases.

- **Benefits and drawbacks of keeping all the data in one database:**

  The advantages of maintaining all data in a single database include:

  - **Simplicity**: Managing a single database is easier than managing several databases.
  - **Performance**: Analyzing a single huge database could be quicker than analyzing several smaller datasets.

  The disadvantages of storing all the data in a single database include:

  - **Scalability**: A high-volume web crawler may require more resources than a single database can provide.
  - **Reliability**: All of the data will be inaccessible if the database crashes.

- **Benefits and drawbacks of keeping the data in separate databases:**

  The advantages of maintaining the data in distinct databases include:

  - **Scalability**: It is simpler to scale several databases than a single database.
  - **Reliability**: The other databases will still be accessible if one database fails.

  Keeping the data in distinct databases has the following disadvantages:

- **Complexity**: Managing several databases is more difficult than managing a single database.
- **Performance**: Analyzing numerous smaller datasets may take longer than doing so on a single large database.

- **Precautions to take to make sure the data is consistent across two databases:**

  The following measures can be done to ensure that the data is consistent between the two databases if the system has its own database:

  Use a two-phase commit to guarantee that all changes to a distributed system are either made or not made. A two-phase commit is a transaction mechanism.
  Use a replication mechanism to verify that the data is the same across all databases in the system.

- **Conclusion:**The best way to design a web crawler with analytics depends on the specific needs of the application. However, the general architecture outlined above should be a good starting point for most applications.