**Question 1 - Theory**

Please answer **all** of the following questions.

| 1.1.1) | Please explain the differences between the distinct concepts of parallelism and concurrency in computer science.  Concurrency deals with managing multiple tasks and their interactions, while parallelism involves executing tasks simultaneously to achieve faster computation or increased throughput. Both concepts are important in computer science and are often used together to design efficient and responsive systems.   * Concurrency primarily aims to improve the responsiveness and efficiency of systems by allowing tasks to make progress concurrently. * Parallelism focuses on achieving faster computation or increased throughput by executing tasks simultaneously across multiple computing resources. |
| --- | --- |
| 1.1.2) | In what sort of contexts would you want to use parallelism, and in which contexts is concurrency sufficient?  I would use concurrency when I would like improve the responsiveness and efficiency of systems like a web-server.  I would use parallelism when I would like to use resources efficiently to speed up the execution of a single task or to handle multiple tasks simultaneously, like I have done on an HPC I maintained in the past. |
| 1.1.3) | What is Python’s GIL **(Global Interpreter Lock)**? How can one achieve parallelism in Python, given the GIL?  Python's Global Interpreter Lock (GIL) is a mutex (a mutual exclusion lock) that protects access to Python objects, preventing multiple native threads from executing Python bytecodes simultaneously in the same interpreter process.  One could use the multiprocessing module in python.  This module allows you to create separate processes, each with its own Python interpreter and memory space. Since each process has its own GIL, multiprocessing can effectively utilize multiple CPU cores |
| 1.1.4) | Async is a popular concurrent paradigm. When would it be appropriate to use it, and how does it work? (for example, in Python’s async.io library or in Node.js)  Async programming is suitable for scenarios where applications need to handle many I/O-bound tasks concurrently without blocking execution. This paradigm is especially useful in network servers, web applications, and other systems where a single thread can handle multiple simultaneous operations efficiently.  Async programming works by allowing tasks to run concurrently without blocking the execution of other tasks. It uses coroutines and event loops. |
| 1.1.5) | In concurrent and parallel programs, how do concurrently executing tasks communicate? Contrast the paradigms.  In concurrent programs, Communication between concurrently executing tasks often occurs via shared resources such as variables, data structures, or files. Shared Memory: Tasks can communicate by reading from and writing to shared variables or data structures in memory. Message Passing: Tasks can exchange messages through communication channels or message queues. Each task has its own address space, and communication between tasks occurs by sending and receiving messages through predefined channels or queues.  In parallel programs, Communication between parallel tasks is typically achieved through explicit message passing or synchronization mechanisms.  Message Passing: Similar to concurrent programs, parallel tasks can communicate by exchanging messages through communication channels or message queues. However, in parallel programs, message passing is often used to distribute work among processors, share data between parallel tasks, or synchronize their execution  Shared Memory: Some parallel programming models also support shared memory communication, where parallel tasks running on different processors can access shared data structures in memory. However, shared memory communication in parallel programs may require additional synchronization mechanisms to ensure data consistency and avoid race conditions. |
| 1.2.1) | What is a database index and how are they implemented?  A database index is a data structure that improves the speed of data retrieval operations on a database table. It's like an index in a book: it helps you quickly find information without having to read through the entire book. |
| 1.2.2) | Why do databases support more than one type of index implementation (reference examples)? |
| 1.3.1) | What does it mean for a language (or code) to be memory-safe? Why would you want a language to have the potential to not be memory-safe?   * Data Structure: Indexes are typically implemented as specialized data structures optimized for fast lookup operations. The most common type of index is the B-tree (balanced tree) data structure, although other types like hash indexes and bitmap indexes also exist. * Index Creation: Indexes are created using SQL commands like CREATE INDEX. When an index is created, the database system builds the index data structure based on the values in the specified column(s) of the table. * Index Key: Each entry in the index corresponds to a value or a combination of values from the indexed column(s). For example, if you create an index on the "name" column of a table, each entry in the index would store a name value along with a pointer to the corresponding row in the table. |
| 1.4.1) | What is microservice architecture? What are some pros and cons of this system design choice?  Microservice architecture is an architectural style that structures an application as a collection of loosely coupled services. Each service is self-contained, independently deployable, and focused on performing a specific business function.  Some of the pros:   * It is scalable * High Technology Diversity * The system is flexible and agile |
| 1.4.2) | What are a few ways different microservices can communicate with each other?  These services communicate with each other over well-defined APIs, often using lightweight protocols such as HTTP or messaging queues |
| 1.4.3) | What are some different ways to authenticate yourself against a secure service endpoint?   * Token Based authentication * Username and password * Oauth * API Keys |
| 1.5.1) | What is a design pattern? Can you explain the **Model-View-Controller (MVC)** design pattern? What sort of frameworks mostly make use of this pattern?  A design pattern is a reusable solution to a commonly occurring problem in software design. It provides a template for solving specific design problems in a structured and efficient way. Design patterns help improve code readability, maintainability, and scalability  The Model-View-Controller (MVC) design pattern is one of the most widely used architectural patterns in software development. It separates an application into three interconnected components, each with its own responsibility:  Model: The Model represents the application's data and business logic.  View: The View is responsible for presenting the Model's data to the user.  Controller: The Controller acts as an intermediary between the Model and the View. It receives input from the user via the View, processes it (often by updating the Model), and updates the View to reflect any changes in the data.  Here are some frameworks that make use of this pattern. Ruby on Rails, Django and Laravel |
| 1.5.2) | What is the Document Object Model (DOM)? Why use a virtual DOM?  The Document Object Model (DOM) is a programming interface for web documents. It represents the structure of HTML or XML documents as a tree-like data structure, where each node in the tree represents an element, attribute, or piece of text in the document.  Using the DOM can be very inefficient. By using a virtual DOM, frameworks can batch updates and minimize the number of DOM manipulations, leading to better performance.  Virtual DOM libraries like React.js support server-side rendering, allowing web pages to be rendered on the server and sent to the client already populated with data. This can improve initial page load times and support better SEO. |
| 1.5.3) | What is server-side rendering? What are its advantages and disadvantages?  Server-side rendering (SSR) is a technique used in web development where web pages are rendered on the server and sent to the client as fully-formed HTML documents. This is in contrast to client-side rendering (CSR), where web pages are rendered in the browser using JavaScript.  Some advantages:  Improved Initial Page Load Time: With SSR, the server sends fully-rendered HTML to the client, reducing the time it takes for the user to see meaningful content.  Better SEO (Search Engine Optimization): Search engines typically index the content of web pages based on the HTML they receive from the server. SSR ensures that search engines can easily crawl and index the content of the page, improving search engine rankings and discoverability.  Some disadvantages:  Increased Server Load: Server-side rendering requires the server to generate HTML for each request, increasing the computational load on the server. This can lead to higher server costs and scalability challenges, especially for high-traffic websites.  Complexity in Development: Implementing SSR requires additional infrastructure and development effort compared to client-side rendering. Developers need to ensure that server-side rendering is compatible with their chosen web development stack and frameworks, which can introduce complexity and potential compatibility issues. |
| 1.5.4) | What is JSX? How is it interpreted by browser Javascript engines?  JSX (JavaScript XML) is an extension to JavaScript syntax commonly used with React, a popular JavaScript library for building user interfaces. JSX allows developers to write HTML-like code within JavaScript  This is how it is interpreted:  JSX resembles XML/HTML syntax but is embedded directly into JavaScript code.  It allows developers to write UI components more intuitively by combining HTML-like markup with JavaScript logic.  JSX elements look similar to HTML elements but are actually JavaScript objects that represent React elements. |

1.5.5) Ultimately, browsers only have a runtime for Javascript. How is it then that frontend frameworks such as React can be written in Typescript? Or unsupported versions of Javascript such as ES6? Explain with reference to the toolchain.

Frontend frameworks like React can be written in TypeScript or use unsupported versions of JavaScript such as ES6 through the use of a toolchain that includes transpilation and bundling processes.

TypeScript is a superset of JavaScript that adds optional static typing and other features. ES6 (ECMAScript 2015) introduced significant enhancements to JavaScript syntax.Developers write their frontend code using TypeScript or ES6 syntax, taking advantage of features like arrow functions, classes, modules, and more

TypeScript and other modern JavaScript flavors (e.g., ES6, ES7) are transpiled into widely supported JavaScript versions (e.g., ES5) that can run in all browsers.TypeScript, for example, extends JavaScript by adding optional static typing and other features. However, browsers can't directly understand TypeScript code.The TypeScript compiler (tsc) transpiles TypeScript code into plain JavaScript code that browsers can execute**.**