ECE 1779 Assignment4

Junhong Chen

Q1. Sawzall

Explain what is Sawzall?

Sawzall is a programming language developed by Google primarily for processing large-scale data sets in distributed computing environments. It is designed to handle vast amounts of data efficiently and is particularly useful for tasks such as log analysis, data mining, and other data processing tasks commonly encountered in big data analytics.

Using https://github.com/google/szl, how is szl is related to Sawzall?

Szl is implementation of Google's Sawzall language for statistical aggregation of log data.

Identify the parallel processing requirements for processing a dataset.

1. Scalability: the system should be able to scale accordingly to handle the increased workload

2. Partitioning: The dataset needs to be divided into smaller chunks that can be processed independently in parallel.

3. Load balancing: Each processing unit should receive a fiar share of the workload.

4. Fault Tolerance: The system should be resilient to failures, or other unexpected events.

Identify how job scheduling is performed.

Job scheduling involves the allocation of computing resources and the sequencing of tasks to efficiently execute jobs in a computing environment. This process typically begins with the submission of jobs to a scheduling system, which evaluates factors such as job priority, resource requirements, and system load. The scheduler then assigns resources to jobs based on predefined policies, such as first-come-first-served, priority-based, or fair-share scheduling. In distributed environments, job schedulers may also consider factors like data locality and network topology to optimize performance. Once resources are allocated, the scheduler orchestrates the execution of tasks within each job, ensuring that dependencies are met and resources are utilized effectively. Job scheduling plays a crucial role in maximizing resource utilization, minimizing job wait times, and improving overall system efficiency in various computing environments, including data centers, cloud platforms, and high-performance computing clusters.

What is the role of the MapReduce s/w.

MapReduce is a software framework designed to simplify and parallelize the processing of large-scale datasets across distributed computing clusters. Its primary role is to enable efficient and scalable data processing by dividing tasks into map and reduce phases.

Identify a system lifecycle for processing a dataset using Sawzall.

1. Requirement Analysis: Define the objectives and requirements for data processing.

2. Data Preparation: Gather, clean, and preprocess the dataset for analysis.

3. Sawzall Scripting: Write Sawzall scripts to analyze and process the dataset.

4. Execution: Execute the Sawzall scripts on a distributed computing infrastructure.

5. Monitoring: Monitor the progress and performance of the data processing tasks.

6. Result Analysis: Analyze the output generated by Sawzall for insights and conclusions.

7. Optimization: Fine-tune the Sawzall scripts and processing pipeline for improved efficiency and performance.

8. Maintenance: Maintain and update the processing pipeline as needed to accommodate changes in data or requirements.

What would be the purpose of integrating a relational database with the system components.

Integrating a relational database with system components serves several purposes. Firstly, it provides a structured and efficient storage mechanism for structured data, enabling fast and reliable access to critical information. Secondly, it facilitates seamless data integration and sharing between different system components, ensuring consistency and coherence across the entire system. Additionally, a relational database supports complex querying and analytics, empowering users to derive valuable insights from the data. Overall, integrating a relational database enhances data management, accessibility, and analysis capabilities, thereby improving the overall performance and effectiveness of the system.

Q2. Developer Ecosystems for Software Safety:
Identify how you would find and fix implementation defects.

As a software engineer, I would employ a systematic approach to find and fix implementation defects. This involves rigorous testing methods such as unit testing, integration testing, and system testing to identify bugs and issues. Additionally, code reviews and peer collaboration help catch errors early in the development process. Once defects are identified, I would use debugging tools and techniques to isolate and understand the root cause of the issue. Finally, I would implement fixes, retest the affected code, and ensure that the solution meets quality standards before deploying it to production. Regular maintenance and monitoring further ensure ongoing reliability and quality assurance.

Explain what a safe system is.

A safe system is a system that mitigates risks of relevant harms and adverse outcomes for its users and stakeholders.

Identify how can you achieve safe deployment for microservices and web services.

Implement automated testing at multiple levels (unit, integration, and end-to-end) to detect issues early in the development cycle. Next, adopt a blue-green or canary deployment strategy to gradually roll out changes and monitor performance in a controlled environment before full deployment. Additionally, utilize containerization and orchestration tools like Docker and Kubernetes for consistency and scalability. Implementing comprehensive monitoring and logging helps detect and mitigate issues quickly. Finally, establish rollback procedures and perform regular disaster recovery drills to ensure resilience and continuity in case of failures.

Q3. The Chubby Lock Service
What is Chubby?

Chubby is a distributed lock service developed by Google to provide coarse-grained locking for distributed systems. It offers a reliable, replicated, and fault-tolerant storage service for managing small amounts of critical configuration and coordination data. Chubby uses a master-slave architecture with multiple replicas to ensure high availability and durability. It provides features such as advisory locks, file system-like directories, and notifications, making it suitable for coordination and synchronization tasks in large-scale distributed systems.

Why Chubby is needed?

Chubby is needed to provide a reliable and fault-tolerant distributed lock service for coordinating access to shared resources in large-scale distributed systems. It ensures consistency and coordination by managing locks and storing critical configuration data across multiple replicas. Chubby's features, such as advisory locks and file system-like directories, enable applications to synchronize access to shared resources and coordinate distributed operations effectively. Its fault-tolerant design and high availability make it essential for maintaining system integrity and reliability in complex distributed environments.

What is coarse-grained synchronization? Explain how it differs from fine-grained synchronization.

Coarse-grained synchronization involves locking large sections of code or resources, reducing concurrency but simplifying coordination. It locks entire data structures or critical sections, limiting access to multiple threads or processes simultaneously. In contrast, fine-grained synchronization locks smaller, more granular sections, allowing for greater concurrency but requiring more complex coordination. Fine-grained locks provide more precise control over shared resources but can lead to higher contention and synchronization overhead. Coarse-grained synchronization sacrifices concurrency for simplicity, while fine-grained synchronization optimizes concurrency at the expense of complexity.

What are three functionalities that Chubby Service implements?

1. Distributed Locking: Chubby provides a distributed lock service, allowing clients to acquire locks on resources to coordinate access in distributed systems. This ensures mutual exclusion and prevents race conditions by allowing only one client to hold a lock at a time.

2. File System-like Directories: Chubby offers a hierarchical namespace similar to a file system, allowing clients to organize and access data in a structured manner. This enables the storage and retrieval of configuration data and other critical information.

3. Notifications: Chubby supports notifications, allowing clients to register interest in specific events or changes to data. When these events occur, Chubby notifies interested clients, enabling them to react accordingly and maintain consistency in distributed applications.

What is the duration of the Chubby service lease time?

The duration of the Chubby service lease time is typically set to a relatively short period, often in the range of seconds to minutes. This lease time determines how long a client can hold a lock or lease on a resource before it expires. When a lease expires, the lock or lease is automatically released, allowing other clients to acquire it.

What is the purpose of the KeepAlive messages.

KeepAlive messages are periodically sent by clients to the Chubby servers to renew their lease and prevent it from expiring prematurely. By sending KeepAlive messages, clients indicate to the Chubby servers that they are still actively using the resources and wish to extend their lease duration. This helps ensure the liveness and continuity of distributed operations by preventing locks from being prematurely released due to client inactivity or network issues, thereby maintaining system integrity and consistency.

Q4. Bigtable

What is BigTable?

Bigtable is a distributed storage system developed by Google for managing structured data. It is designed to handle large-scale data sets across thousands of commodity servers, providing high availability, scalability, and fault tolerance. Bigtable organizes data in tables with rows identified by a unique row key, allowing for efficient storage, retrieval, and querying of data. It is widely used within Google for various applications, including web indexing, analytics, and data warehousing.

How does BigTable achieves concurrency?

It uses a distributed architecture, partitioning data into tablets, which can be independently managed. Each tablet can be accessed and modified concurrently by multiple clients. BigTable employs distributed locks to coordinate access to shared resources, ensuring consistency while allowing parallel operations. Additionally, it utilizes a distributed commit protocol to manage transactions across multiple nodes, enabling atomicity and isolation. These mechanisms enable BigTable to support high concurrency while maintaining data integrity and consistency in a distributed environment.

What happens when a client’s session expires?

any ongoing operations associated with that session are aborted, and the client's connection to the server is terminated. Any locks held by the client are released, allowing other clients to access the affected resources. The client needs to re-establish its session by reconnecting to the BigTable server to resume its operations.

List four functions that BigTable uses by deployment of Chubby.

1. Master Election: Chubby helps BigTable nodes elect a master node responsible for coordinating cluster operations, ensuring high availability and fault tolerance.

2. Configuration Management: Chubby stores and distributes configuration data across BigTable nodes, enabling dynamic adjustment of cluster settings without downtime.

3. Lock Service: Chubby provides distributed locking, ensuring mutual exclusion for critical operations across BigTable nodes, preventing conflicts and maintaining data integrity.

4. Health Checking: BigTable leverages Chubby for health monitoring, enabling nodes to detect failures and initiate failover procedures to maintain system reliability and performance.

What are Bloom filters? How are they used in Chubby?

Bloom filters are probabilistic data structures used for membership testing. They efficiently determine whether an element is likely present in a set or not, with a small probability of false positives. In Chubby, Bloom filters are employed to reduce the number of unnecessary disk reads when searching for files or directories. They help quickly determine if a file or directory might exist on a Chubby server, allowing the system to avoid costly disk accesses for non-existent entries. This optimization improves lookup performance, especially in scenarios where Chubby manages a large number of files or directories.

Q5

the research studies performance interference between high priority accelerated ML tasks and low priority CPU tasks. Identify the four production ML workloads that research experimented with. Identify the three accelerated platforms that were utilized.

Image classification, object detection, language translation, speech recognition.

GPU, FGPAs, TPUs

What are TPUs and GPUs?

TPUs (Tensor Processing Units) and GPUs (Graphics Processing Units) are specialized hardware accelerators used in machine learning and other computationally intensive tasks. TPUs are custom-designed by Google specifically for deep learning tasks, optimizing for matrix operations common in neural network computations. GPUs, originally developed for rendering graphics, have evolved into powerful parallel processors suitable for a wide range of tasks, including machine learning.

Use https://cloud.google.com/tpu/docs/intro-to-tpu to compare between TPUs and GPUs.

GPUs

Models with a significant number of custom TensorFlow/PyTorch/JAX operations that must run at least partially on CPUs

Models with TensorFlow ops that are not available on Cloud TPU (see the list of available TensorFlow ops)

Medium-to-large models with larger effective batch sizes

TPUs

Models dominated by matrix computations

Models with no custom TensorFlow/PyTorch/JAX operations inside the main training loop

Models that train for weeks or months

Large models with large effective batch sizes

Bandwidth in ML processing problems is considered as a bottleneck explain how the problem was mitigated.

Bandwidth limitations in ML processing were mitigated by optimizing data movement and storage strategies. Techniques such as data compression, minimizing redundant transfers, and utilizing high-speed interconnects like InfiniBand or RDMA improved data transfer efficiency. Additionally, distributed computing frameworks like TensorFlow and PyTorch distributed training helped parallelize computation, reducing the reliance on bandwidth.