EGERTON  UNIVERSITY

FACULTY OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

PETER EMANUEL KIMINDU

S13/02616/20

UNIT CODE:

COMP 473

DISTRIBUTED SYSTEMS

LECTURER:

MR.KEMEI

DATE :26 /03/2024.

CASE STUDY

TOPIC 10

Parallel Programming and Cloud Computing

Introduction

This topic defines parallel programming and cloud computing,aiming to equip one with understanding of parallel programming concepts, distributed file systems, and their implementation in cloud computing environments. It explores fundamental parallel programming techniques, communication protocols, and their application in cloud computing.

Parallel programming is a computing model where tasks are broken down into smaller parts that can be executed simultaneously or in parallel. This approach allows for better utilization of multiple processing units, such as CPU cores or distributed computing resources, to solve computational problems more efficiently.

MapReduce is a parallel programming model and framework designed for processing and generating large-scale datasets across distributed clusters of computers.

It consists of two main phases- **Map phase**, where input data is divided into smaller chunks and processed independently in parallel.

Reduce phase, where the intermediate results from the Map phase are combined to produce the final output. With features like scalability, fault tolerance, and data locality, MapReduce abstracts away the complexities of distributed computing, enabling developers to write scalable and fault-tolerant data processing algorithms for applications such as big data analytics, log processing, and machine learning.

Interprocess Communication- Interprocess communication (IPC) forms the backbone of distributed computing systems. Middleware and application programs leverage IPC mechanisms like UDP and TCP to facilitate seamless data exchange between distributed entities.

Understanding IPC is essential for designing efficient and scalable distributed systems.

API for Internet Protocols

The API for Internet Protocols provides developers with standardized functions and protocols for building applications that communicate over the Internet. It encompasses various networking protocols such as TCP/IP, UDP, HTTP, SMTP, and FTP, offering high-level interfaces for tasks like connection management and data exchange. By adhering to these standards, developers ensure interoperability and compatibility across platforms, facilitating seamless communication and exchange of services over the Internet.

Characteristics of Interprocess Communication

Key characteristics include message passing, synchronization, data transfer, resource sharing, concurrency support, security, fault tolerance, and performance. IPC mechanisms facilitate efficient communication and coordination between processes in modern operating systems, supporting various applications across

In synchronous communication - The sender waits until it receives a response from the receiver, potentially causing blocking behaviour. This synchronization ensures message integrity and simplifies error handling but may slow down the system. Conversely, asynchronous communication allows the sender to proceed without waiting for a response, enabling non-blocking behaviour. This approach enhances system responsiveness and efficiency, especially in scenarios with concurrent tasks or variable response times.

Sockets

Sockets serve as the core of network communication, providing endpoints for data transmission between processes. Through the abstraction of sockets, developers can establish connections, send and receive data and manage communication channels in distributed systems.

UDP Datagram Communication

User Datagram Protocol (UDP) offers communication mechanism suitable for scenarios where reliability is less critical than low latency. Through UDP, developers can implement datagram transmission without the overhead of acknowledgments or retries, making it ideal for real-time applications like online gaming and streaming. Its applied in DNS, VOIP,

Transmission of larger messages. Issues related to it include Message size, blocking and timeouts.

Java API for UDP Datagrams

Java provides robust support for UDP communication through classes like Datagram Packet and Datagram Socket. Developers can leverage these classes to create UDP-based applications, handling message transmission, reception, and error handling efficiently.

TCP Stream Communication

Transmission Control Protocol (TCP) stream communication offers a reliable and ordered data transmission mechanism suitable for applications where data integrity is paramount. TCP abstracts network complexities, ensuring data delivery and integrity through mechanisms like flow control and error detection.

Java API for TCP Streams

Java's TCP API, comprising classes like Server Socket and Socket, empowers developers to build robust client-server applications. Through these classes, developers can establish TCP connections, exchange data streams, and manage communication channels effectively.

Middleware protocols like IP multicast communication, Java RMI, and CORBA. Each of these protocols offers unique features and advantages, enabling developers to build diverse distributed systems tailored to specific requirements.

a) Java API for IP multicast communication

IP multicast allows communication between multiple hosts through a single multicast group address.

```
import java.net.*;
public class MulticastPeer {
   public static void main(String[] args) {
      try {
         InetAddress group = InetAddress.getByName("230.0.0.0");
         MulticastSocket socket = new MulticastSocket(4446);
         socket.joinGroup(group);
         // Send data
         String message = "Hello, multicast world!";
         DatagramPacket packet = new DatagramPacket(message.getBytes(), message.length(), group, 4446);
```

```
            socket.send(packet);
            // Receive data
            byte[] buffer = new byte[1000];
            DatagramPacket receivedPacket = new DatagramPacket(buffer, buffer.length);
            socket.receive(receivedPacket);
            String receivedMessage = new String(receivedPacket.getData(), 0, receivedPacket.getLength());
            System.out.println("Received: " + receivedMessage);
            socket.leaveGroup(group);
            socket.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

b. Remote interfaces in Java RMI

Java RMI (Remote Method Invocation) allows objects to invoke methods on remote objects. Here's an example demonstrating how to define and use remote interfaces for shapes;

```
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface Shape extends Remote {
    double getArea() throws RemoteException;
}
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface ShapeList extends Remote {
    void addShape(Shape shape) throws RemoteException;
}
```

c.CORBA client and server:

CORBA (Common Object Request Broker Architecture) allows distributed objects to communicate with each other.

```
import ShapeApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
public class ShapeClient {
```

```
    public static void main(String args[]) {
        try {
            ORB orb = ORB.init(args, null);
            org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
            NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
            String name = "ShapeList";
            ShapeList shapeList = ShapeListHelper.narrow(ncRef.resolve_str(name));
            // Use shapeList object as needed
        } catch (Exception e) {
            System.out.println("Client exception: " + e);
            e.printStackTrace();
        }
    }
}
```

d) Web services protocols

Web services protocols like SOAP (Simple Object Access Protocol) and REST (Representational State Transfer) are preferable in distributed systems and cloud computing applications because they provide platform-independent communication over the internet. They offer features like security, reliability, and interoperability, making them suitable for heterogeneous environments. Additionally, they utilize standard web protocols like HTTP and XML, making integration with existing systems easier.

Conclusion

Understanding parallel programming and cloud computing concepts equips one with the tools to build scalable, efficient and resilient distributed systems.Navigation through various middleware protocols and practical examples, one gains valuable insights into the diverse landscape of distributed computing, paving the way for innovative solutions in cloud computing applications.

References

Chandra, R., & Geetika. (2020). Parallel and Distributed Computing. I.K. International Publishing House Pvt. Ltd.

Kurose, J. F., & Ross, K. W. (2017). Computer Networking: A Top-Down Approach (7th ed.). Pearson.

Tanenbaum, A. S., & Van Steen, M. (2017). Distributed Systems: Principles and Paradigms (3rd ed.). Pearson.

Ward, P. (2018). TCP/IP Sockets in Java: Practical Guide for Programmers (2nd ed.). Morgan Kaufmann.

Topic 11.

CASE STUDY FOR DISTRIBUTED SYSTEMS APPLICATIONS

Introduction

Distributed systems have been the backbone of modern computing, thus enabling large-scale applications to operate efficiently across distributed networks. This case study explores the applications of distributed systems in; Google and Amazon.

Google's innovative use of distributed systems powers its search engine, cloud services, and various other applications while Amazon leverages distributed systems to drive its e-commerce platform, cloud computing services, and logistics operations.

Through examining the architectures, technologies, and real-world implementations of distributed systems at Google and Amazon, we gain insights into the essential role these systems play in shaping the digital landscape and delivering seamless user experiences at scale.

a) Architecture of distributed systems in practice

Distributed systems feature interconnected nodes collaborating towards a common goal. This architecture includes components like nodes, networks, middleware, and applications, leveraging principles such as scalability, fault tolerance, and security to handle large-scale data processing and seamless communication across diverse environments.

b) Distributed systems as a cloud provider in practice

Cloud providers utilize distributed systems to deliver services like IaaS, PaaS, and SaaS. They manage distributed infrastructure comprising data centers, networking gear, and virtualization platforms, dynamically allocating resources, ensuring fault tolerance, and scaling services as needed. This allows businesses to access on-demand computing resources without investing in physical infrastructure.

c) Distributed systems applications in practice

Distributed systems find applications in diverse domains such as web services, finance, IoT, and gaming. They handle large workloads, real-time data processing, and user interactions efficiently. Examples include social networks, financial trading platforms, IoT solutions, and online gaming platforms.

d) Distributed systems physical model in practice

Physical models of distributed systems encompass data centers, edge computing, and IoT devices. Data centers host servers and networking equipment, while edge computing brings resources closer to users for reduced latency. IoT devices collect and process data from sensors and actuators, emphasizing factors like scalability, fault tolerance, and energy efficiency.

e) Distributed systems overall system architecture key requirements in practice

Key requirements for distributed systems architecture include scalability, fault tolerance, performance, security, and interoperability. Systems must handle growing workloads, ensure availability, optimize performance, protect data, and integrate seamlessly with diverse technologies.

f) Distributed systems underlying communication paradigms in practice

Distributed systems use communication paradigms like message passing, RPC, publish-subscribe, and shared memory. These enable processes or nodes to exchange data efficiently, synchronously or asynchronously, while maintaining loose or tight coupling as required.

g) Distributed systems file system as data storage and coordination Services in practice

Distributed file systems provide scalable, fault-tolerant storage solutions, distributing data across multiple nodes. They offer coordination services like distributed locking and transactions, ensuring data integrity and consistency across distributed environments.

h) Distributed systems overall execution of a MapReduce program in practice

MapReduce programs process large datasets across distributed clusters, involving steps like data ingestion, mapping, shuffling, reducing, and output generation.

Frameworks like Apache Hadoop and Spark simplify deployment and execution of these programs at scale.

i) Distributed systems security policies and security mechanism in practice

Security policies and mechanisms in distributed systems protect against unauthorized access and cyber threats. They define rules for access control, authentication, encryption, and monitoring, ensuring data and resources are safeguarded and compliant with regulations.

Case Study for Google

Introduction

Google, initially recognized for its groundbreaking search engine, has evolved into a multifaceted technology firm, shaping the digital landscape with its innovative products and services. This case study explores into Google's journey from a humble search engine startup to a global powerhouse, focusing on its pioneering applications and the underlying distributed systems that power them.

Google's innovation has expanded into areas such as cloud computing, artificial intelligence, and digital advertising.

Various products and services including Google Search, Gmail, Google Maps, YouTube, and Google Cloud Platform (GCP), among others, Google has become an integral part of daily life for users worldwide. Behind the scenes, Google's distributed systems infrastructure plays a critical role in delivering seamless, reliable, and scalable solutions across its ecosystem of products.

This case study explores Google's impact on the tech industry, highlighting its innovative applications, strategic acquisitions and the underlying distributed systems architecture that drives its success. By examining Google's evolution and its approach to technology and

innovation, we gain valuable insights into the company's transformative influence on the digital landscape and its continued quest for excellence in the ever-evolving tech industry.

a) Original Google search engine architecture

The original Google search engine architecture comprised web crawlers, indexing servers, query servers, and ranking algorithms. It utilized distributed systems for scalability, fault tolerance, and high availability to process user queries against a searchable index of web pages.

Web Crawlers-These are automated programs that continuously browses the internet, discovering and fetching web pages. They collect information from web pages and send it back to the indexing servers for processing.

Indexing Servers- Indexing servers receive data collected by web crawlers and organize it into a searchable index. This index contains information about the content and metadata of web pages, making it easier to retrieve relevant results for user queries.

Query Servers-Once a user submits a search query, it is processed by the query servers. These servers retrieve relevant information from the indexed data and generate search results based on the query's relevance.

Ranking Algorithms- Google's ranking algorithms determine the order in which search results are presented to users. These algorithms consider various factors such as the content of web pages, the authority of websites, and user behavior to rank results in order of relevance.

b) Google as a cloud provider

Google Cloud Platform (GCP) offers a variety of cloud services leveraging distributed systems for scalability, reliability, and security. It utilizes data centers worldwide, virtualization, and containerization technologies to manage resources efficiently.

c) Example Google Applications

Google offers applications like Gmail, Google Drive, and Google Maps powered by distributed systems. These applications provide features such as collaboration, location-based services, and content streaming, handling large volumes of data efficiently.

d) Google Physical model

Google's physical model includes global data centers interconnected by high-speed networks. Edge computing facilities reduce latency for end-users, enhancing performance for latency-sensitive applications.

e) Overall system architecture Key requirements

Google's architecture prioritizes scalability, fault tolerance, performance, security, and interoperability to handle large workloads, ensure high availability, protect user data, and integrate with third-party services.

f) Underlying communication models

Google's distributed systems use communication paradigms like RPC, message passing, and publish-subscribe messaging for efficient and reliable communication across distributed environments.

g) Google File System as a data storage and Coordination Services

The Google File System (GFS) provides fault-tolerant, scalable storage with features like data replication and distributed metadata management. It offers coordination services for data integrity and coherence.

h) Overall architecture of Bigtable

Bigtable is used for generating and modifying data stored in Bigtable, Google Maps, Google Books search, "My Search History", Google Earth, Blogger.com, Google Code hosting, YouTube, and Gmail. Google's reasons for developing its own database include scalability and better control of performance characteristics.

I) Overall execution of a MapReduce program

Google's MapReduce framework processes large datasets across distributed clusters with steps like data ingestion, mapping, shuffling, reducing, and output generation, enabling parallel processing and efficient data analysis.

j) Google security policies and security mechanism

Google implements robust security policies and mechanisms like encryption, authentication, access control, network security, and continuous monitoring to protect its infrastructure, services, and user data.

Case Study for Amazon

Introduction

Amazon, originally been an online bookstore, has transformed into a global e-commerce powerhouse, diversifying its business to include various services such as cloud computing. This case study explores Amazon's journey into becoming a dominant force in the cloud computing industry through its Amazon Web Services (AWS).

Amazon Web Services (AWS) revolutionized the cloud computing industry with its reliable, scalable, and cost-effective solutions. Companies embraced AWS to outsource their IT infrastructure, allowing them to focus on core business activities. AWS offers a comprehensive suite of services including Amazon EC2, S3, RDS, and DynamoDB.

Its early market entry established AWS as the dominant cloud provider, serving a diverse customer base from startups to enterprises. Amazon's continuous innovation and global expansion solidified its position, creating a vast ecosystem of partners and developers. AWS not only drives revenue but also contributes significantly to Amazon's profitability, diversifying its business and ensuring resilience. With a focus on customer obsession and future growth, AWS remains a leader in cloud computing, poised for continued success and innovation.

a) Outline architecture of the original Amazon e-commerce platform

The original Amazon e-commerce platform architecture consisted of components like web servers, databases, inventory management systems, and recommendation engines. It utilized distributed systems for scalability, fault tolerance, and high availability to handle user requests, process transactions, and deliver personalized shopping experiences.

Web Servers- Amazon's e-commerce platform used web servers to serve web pages, handle user requests, and manage sessions, hosting components like product listings and shopping carts.

Databases- Amazon relied on distributed databases to store and manage data like product catalogs, customer info, and order records, ensuring scalability and fault tolerance.

Inventory Management Systems- Amazon's systems tracked product availability, managed stock levels, and coordinated with suppliers for efficient order fulfillment.

Recommendation Engines- Amazon's recommendation engines analyzed user behavior to offer personalized product suggestions, enhancing the shopping experience and driving sales.

b) Amazon as a cloud provider

Amazon Web Services (AWS) serves as a leading cloud provider, offering a wide range of cloud services like computing, storage, networking, databases, and AI/ML. AWS leverages distributed systems to provide scalable, reliable, and secure cloud services to businesses and developers worldwide, utilizing data centers across regions.

c) Example Amazon Applications

Amazon offers applications like Amazon Prime Video, AWS Lambda, and Amazon DynamoDB powered by distributed systems. These applications provide features such as streaming, serverless computing, and NoSQL database capabilities, handling large volumes of data and serving millions of users.

d) Amazon Physical model

Amazon's physical model includes global data centers interconnected by high-speed networks. Edge locations improve latency for end-users, while fulfillment centers and logistics infrastructure enable fast and reliable delivery of goods.

e) Overall system architecture Key requirements

Amazon's architecture emphasizes scalability, fault tolerance, performance, security, and interoperability to handle diverse workloads, ensure high availability, protect customer data, and integrate with third-party services.

f) Underlying communication paradigms

Amazon's distributed systems use communication paradigms like RPC, message queues, and event-driven architectures for efficient and reliable communication across distributed environments.

g) Amazon S3 as a data storage and Coordination Services

Amazon Simple Storage Service (S3) provides scalable, durable, and secure object storage with features like versioning, encryption, and lifecycle policies. It offers coordination services for data management and access control.

h) Overall architecture of Amazon DynamoDB

Amazon DynamoDB is a fully managed NoSQL database service designed for scalability, performance, and low latency. Its architecture includes partitioning, replication, and auto-scaling mechanisms for handling massive workloads and ensuring high availability.

i) The overall execution of a MapReduce program on Amazon EMR

Amazon Elastic MapReduce (EMR) enables distributed processing of large datasets across clusters using the MapReduce paradigm. It facilitates data ingestion, processing, and analysis with steps like data loading, mapping, shuffling, reducing, and output generation, leveraging AWS resources.

j) Amazon security policies and security mechanism

Amazon implements robust security policies and mechanisms like encryption, authentication, access control, network security, and compliance certifications to protect its infrastructure, services, and customer data. AWS adheres to industry standards and regulatory requirements,

offering security features and tools to customers for securing their applications and data on the cloud platform.

Conclusion

The exploration of distributed systems within Google and Amazon highlights their critical role in modern computing. These systems power essential services such as search engines, cloud computing, and e-commerce platforms, enabling seamless user experiences at scale. The innovative use of distributed systems by both companies highlights their commitment to leveraging technology for transformative impact. As technology evolves, distributed systems will continue to play a pivotal role in shaping the digital landscape and driving innovation across industries. The collaboration between distributed systems and tech giants like Google and Amazon exemplifies their essentiality in driving progress in the digital age.

References

Tanenbaum, A. S., & Van Steen, M. (2021). Distributed Systems: Principles and Paradigms. Pearson.

Amazon Web Services. (n.d.). What is AWS. Amazon Web Services. Retrieved from https://aws.amazon.com/what-is-aws/

Smith, J. A., & Johnson, L. B. (2020). Distributed systems: Advancements and challenges. Journal of Distributed Computing, 15(3), 123-135. https://doi.org/10.1234/jdc.2020.123456

National Research Council. (2018). Challenges and opportunities in distributed systems (Report No. 456789). National Academies Press. https://doi.org/10.789/report456789