SECTION 9:: BIG DATA ARCHITECTURE PATTERNS

This lecture introduces big data and its significance, focusing on the challenges of processing large datasets. It defines big data as datasets too large, complex, or fast-moving for traditional applications. Three key characteristics are highlighted:

1.Volume: This refers to the enormous amounts of data that organizations handle, such as terabytes or petabytes from various sources like search engines, medical systems, and weather prediction models.

2.Variety: This addresses the different types of data, particularly unstructured data. Organizations need to integrate diverse data sources to uncover hidden insights, with social media as an example.

3.Velocity: This pertains to the speed of data generation and processing. High-traffic online stores and IoT devices generate continuous data streams that require quick ingestion and analysis.

The lecture also touches on the complexities and costs involved in storing and processing big data, while emphasizing the competitive advantages gained from insights like visualizations and predictive analytics. It concludes by previewing architectural styles for effective big data processing and analysis.

The "Big Data Processing - Introduction" section focuses on two main strategies for processing big data using event-driven architecture: Batch Processing and Real-Time Processing.

1.Batch Processing: This method involves storing incoming data in a distributed database or file system without modification and processing it in batches at scheduled intervals. It allows for efficient handling of multiple records simultaneously and is exemplified by applications such as online learning platforms that analyze user engagement data or search engines that periodically index large datasets. Benefits include ease of implementation, high availability, efficiency, and fault tolerance. However, a significant drawback is the delay between data input and output, which can affect real-time decision-making.

2.Real-Time Processing: Here, each incoming event is processed immediately as it enters a queue, enabling instant analysis and feedback. This approach is ideal for situations requiring immediate response, like log analysis or stock trading. Nevertheless, it has limitations in performing complex analyses since it mainly focuses on recent data.

Overall, this section equips learners with a foundational understanding of these big data processing strategies, along with their specific use cases, advantages, and disadvantages.

The "Batch Processing Model" section delves into the mechanics and advantages of batch processing in big data systems. It highlights the key features and benefits, including:

1.Data Handling: In the batch processing model, data is collected and stored in a distributed system without immediate modification. This approach allows for the processing of large volumes of data at scheduled intervals rather than in real-time.

2.Historical Insights: One of the primary advantages of batch processing is the ability to analyze complete datasets over time. This facilitates in-depth insights into historical data and allows for pattern recognition, making it easier to generate reliable predictive models.

3.Data Fusion: Batch processing enables the integration of data from various sources, effectively creating enriched datasets that contribute to enhanced analytical capabilities and business intelligence.

4.Efficiency: The model is built for efficiency, allowing multiple records to be processed simultaneously, which can be particularly advantageous for certain business use cases where immediate results are not critical.

5.Trade-offs: Despite its benefits, batch processing presents a trade-off in terms of data latency. There is typically a delay between data input and output, which can be a significant factor in scenarios requiring timely responses.

The section emphasizes the applicable scenarios for batch processing while also recognizing its limitations, providing a foundation for understanding when to choose this model over real-time processing strategies.

The "Real Time Processing Model" section explains how real-time processing handles incoming data by putting each new event into a queue or message broker, allowing for immediate processing. Key aspects include:

1.Immediate Processing: As each data record enters the system, it is processed without any delay. This facilitates instant analysis and enables rapid responses to events, making it suitable for scenarios like online stock trading, where trades must be matched promptly.

2.Real-Time Visualization: Once an event is processed, the system updates the database, providing real-time querying capabilities for visualization and analysis. This ensures that users and systems can react to data as soon as it arrives.

3.Advantages: The primary advantage of this model is the ability to analyze and respond to data instantaneously, significantly enhancing operational efficiency and decision-making speed.

4.Limitations: Despite its benefits, real-time processing is challenging for complex analyses. It focuses primarily on recent data, making historical analysis difficult and limiting the insights available from broader datasets.

5.Use Cases: The model is ideal for applications that require quick responses without deep historical analysis, contrasting with batch processing, which excels at comprehensive data analysis but suffers from delays.

This section provides a foundational understanding of the real-time processing model, its operational dynamics, advantages, and typical scenarios for application.

The "Challenges of Choosing Big Data Processing Strategy" section addresses the complexities involved in selecting the most suitable big data processing approach. Key challenges highlighted include:

1.Data Volume and Velocity: The huge volume and high-velocity nature of big data often exceed traditional processing capabilities. Organizations must evaluate their infrastructure and decide whether their existing systems can handle increased loads. This may involve investing in more advanced computing resources or selecting a processing strategy that aligns with their capacity.

2.Variety of Data: Organizations often deal with diverse types of data, including structured, semi-structured, and unstructured formats. This variety complicates integration and analysis, leading to challenges in data fusion and requiring thoughtful strategies to process and combine disparate data sources effectively.

3.Use Cases and Requirements: Different applications necessitate different processing strategies, from real-time analytics to batch processing. Understanding specific business needs, such as the urgency of data analysis and the type of insights required, is crucial for making the right choice.

4.Cost-Effectiveness: Budget constraints often play a significant role in decision-making. Organizations must assess the costs associated with implementing various processing strategies while ensuring they meet performance and scalability needs.

5.Scalability and Future Growth: Selecting a strategy that is scalable is vital, as companies must anticipate future data growth. This foresight helps in avoiding the need for frequent overhauls of data processing systems.

6.Complexity of Implementation: The complexity involved in deploying various big data technologies can pose a significant barrier. Adequately trained personnel and streamlined processes are needed to ensure successful implementation and maintenance.

Making informed decisions about the most appropriate big data processing strategy requires careful consideration of these challenges, balancing immediate needs with long-term goals.

The "Lambda Architecture" section presents a robust framework for handling big data processing by merging the strengths of batch and real-time processing. Key aspects include:

1.Definition and Purpose: Introduced by Nathan Mars, the Lambda Architecture is designed to balance high fault tolerance and comprehensive data analysis from batch processing with the low latency features of real-time processing.

2.Architecture Layers:

•Batch Layer: Acts as the system of records, managing immutable master data and pre-computing batch views. It processes all data periodically to create a comprehensive dataset while ensuring data consistency by appending new records rather than modifying existing ones.

•Speed Layer: Handles real-time data processing, focusing on immediate insights and allowing for swift operational responses to newly incoming data.

•Serving Layer: Provides a unified interface for querying processed data from both the batch and speed layers, catering to the diverse needs of applications.

3.Data Handling: Data enters the system simultaneously in both layers, ensuring that recent insights from the speed layer complement the extensive historical analysis provided by the batch layer.

4.Use Case Examples: The architecture allows for various queries, such as current user engagement with ads and historical ad performance, demonstrating how combined data enables effective decision-making.

By integrating these layers, Lambda Architecture addresses the limitations of using either batch or real-time processing alone, making it suitable for applications needing both timely responsiveness and in-depth analysis.

The "Lambda Architecture in Action" section outlines the practical application of the Lambda Architecture framework in processing big data. Here are the key points:

1.Hybrid Approach Necessity: The lecture revisits the challenges posed by batch and real-time processing methods. Batch processing provides in-depth insights into historical data but suffers from high latency. In contrast, real-time processing enables immediate querying but lacks comprehensive historical analysis. Many use cases require both immediate insights and historical context.

2.Use Case Examples:

•Log Monitoring: In systems that monitor logs and performance metrics, real-time visibility is crucial for troubleshooting, while historical data is vital for performance comparisons and anomaly detection.

•Ride-Sharing Services: Real-time data is required to quickly match drivers with clients, while historical data helps analyze trends for optimizing service delivery.

3.Architecture Layers:

•Batch Layer: Manages the master dataset and performs comprehensive processing, ensuring accuracy while storing results in a read-only format.

•Speed Layer: Handles incoming data in real time, providing quick insights without analyzing historical data.

•Serving Layer: Merges outputs from both layers, facilitating flexible querying that integrates real-time metrics with historical performance data.

4.Practical Application: The lecture uses an example from the ad tech industry, demonstrating how Lambda Architecture efficiently processes various event types (ad views, clicks, purchases) through both layers. It enables advertisers to access real-time metrics and historical performance, illustrating the architecture's effectiveness in balancing immediate and long-term analytical needs.

In summary, the Lambda Architecture serves as a crucial solution for scenarios requiring both batch and real-time processing, forming a comprehensive framework for big data analysis.

SECTION 10: SOFTWARE ARCHITECTURE AND SYSTEM DESIGN PRACTISE

a. Highly Scalable Public Discussion Forum (like Reddit, or Stackoverflow)

This lecture focuses on the design of a highly scalable public discussion forum from scratch, emphasizing key system design principles. Here are the main points:

1.Requirements Gathering: Both functional and non-functional requirements are identified through targeted questions. Functional requirements include user registration, creating posts with titles, tags, multimedia content, commenting, and voting. Non-functional requirements emphasize scalability, performance, fault tolerance, and high availability.

2.Designing the REST API: Key entities such as users, posts, images, comments, and votes are identified and mapped to URIs. JSON format is used for representations of posts and comments, and HTTP methods are allocated to the appropriate API operations.

3.Performance Considerations: Pagination is highlighted as an important technique for managing large volumes of posts and comments, which helps improve performance and user experience.

4.Systematic Approach: The lecture reinforces the importance of clarity in requirements and thoughtful API design, enabling the creation of a scalable discussion forum.

In this lecture, we focus on designing a highly scalable software architecture for a discussion forum by addressing both functional and non-functional requirements. Here are the key points:

1.Functional Requirements: We outline essential features such as user sign-up/login, post creation, commenting, voting, and generating a newsfeed of popular posts.

2.Microservice Architecture: We adopt a microservice approach to handle different functions:

•User Service: Manages user accounts and stores data in a SQL database.
•Post Service: Handles post creation and uses a NoSQL database for flexible data structures including topic tags and images.
•Commenting Integration: Comments are integrated into the Post Service for easier data retrieval, with a dedicated collection for comments.

3.Voting Service: Dedicated to tracking votes, ensuring users can vote only once per post/comment and maintaining timestamps for a 24-hour count.

4.Ranking Service: Manages the newsfeed by pulling data from the Voting and Post Services, performing batch processing to update and rank posts based on votes, and storing results in a read-optimized database.
5.Next Steps: We have established a clear architecture that meets the functional requirements, with an upcoming focus on non-functional aspects to ensure scalability, performance, and reliability.

In this lecture, we finalize the architecture for a highly scalable discussion forum by focusing on non-functional requirements. Here are the main points:

1.Scalability: We introduce a load balancer to manage traffic and an API gateway to separate frontend from backend services. Sharding is proposed for the posts and comments database to efficiently distribute data across multiple instances.
2.Sharding Strategies: We explore two strategies: applying a hash function on post IDs for posts and using a compound index of post ID and comment ID for comments, balancing performance with scalability.
3.Performance Enhancements: To address image loading times, we suggest utilizing a global CDN and edge servers for static content, along with API caching during peak traffic to improve response times and reduce latency.
4.Voting Mechanics: A message broker is proposed between the voting service and the post/comment service to manage vote counts efficiently, helping to buffer spikes in traffic without affecting user experience.
5.Fault Tolerance & High Availability: We discuss the importance of database replication and operating services across multiple data centers to prevent data loss and ensure resilience, supported by global load balancing for enhanced performance during disasters.
6.Overall Strategy: We ensure scalability through load balancing and sharding, enhance performance via CDN and indexing, maintain high availability with replication, and ensure durability through regular backups.
This comprehensive approach ensures the discussion forum can efficiently handle user demands with reliability and responsiveness.

b. Scalable Ecommerce Marketplace (like Amazon, Ebay)

In this lecture, we explore the design of a scalable eCommerce marketplace platform tailored for merchants and buyers.

1.Functional Requirements:

•Merchants: Need a system for product management where they can create, update, and analyze their products' performance.

•Buyers: Require a storefront for browsing and searching products without needing to register, along with a detailed checkout process that informs them of costs and order status.

2.Project Scope:

•Emphasizes the need to distinguish between in-scope and out-of-scope features to understand the project's boundaries better.

3.Visualization with Sequence Diagrams:

•A sequence diagram is used to illustrate interactions among merchants, buyers, and third-party services for payment and delivery. This aids in organizing requirements and clarifying operational flows like product creation, inventory updates, and the checkout process.

4.Quality Attributes:

•Highlighted that scalability is vital for high user traffic, while merchants can handle some latency. Performance metrics with response time goals for storefront and checkout processes are outlined.

5.Conclusion:

•Recaps essential steps in clarifying requirements, creating sequence diagrams, and defining quality attributes for the project, which sets the foundation for subsequent design steps.

In this lecture, we advance the design phase of our e-commerce marketplace platform, emphasizing functional requirements and the architecture diagram.

1.Merchant Functional Requirements:

•We aim to create a web-based product management system for merchants, involving the development of a product management service. This service allows merchants to register and log in, using a SQL database for secure merchant information storage.

•A product service will enable merchants to add or update products, with product details stored in a NoSQL database to accommodate varying attributes.
•An object store will handle product images, and an inventory service will manage product inventory via a high-performance key-value store.
2.User Functional Requirements:
•For users, we design a storefront with separate UIs for web and mobile. A storefront web service will serve HTML pages, while an API gateway will manage requests from the mobile app.
•We address the conflicting needs of merchants and users concerning product data through a Product Search Service optimized for read-heavy workloads, ensuring availability for users and maintaining consistency for merchants.
3.Checkout Process:
•We introduce a tax service for calculating taxes based on product type and user location, along with order, payment, and shipping services for processing user orders.
•An asynchronous approach to order processing is implemented to provide immediate user confirmation while allowing billing and shipping to be handled in the background, supported by an event sourcing pattern for order data that improves auditing and recovery.
4.Analytics Requirements:
•We discuss the requirements for merchants to collect data on product page visits and sales, proposing a Lambda architecture for real-time and historical analytics. This involves a speed layer for immediate data and a batch layer for aggregated insights.
In summary, we have established a microservices architecture to distribute responsibilities effectively, addressed conflicting requirements through strategic choices, and laid the groundwork for a robust e-commerce platform. The next lecture will focus on nonfunctional requirements to refine our architecture further.

In this lecture, we finalize the software architecture for our e-commerce marketplace by addressing nonfunctional requirements for both merchants and storefront users.
1.Architecture Review:

•We revisit the architecture diagram developed previously, which illustrates system features and quality attributes.

2.Merchant Requirements:

•Focus is on performance, consistency, and availability. Since merchants update infrequently, a few instances of the product management system suffice. We implement load balancing and ensure high availability with a primary database and replicas.

•A CP (Consistency and Partition tolerance) model is adopted for databases to maintain consistency while ensuring high availability.

3.Storefront User Requirements:

•Given the high user volume and potential traffic spikes, we emphasize high availability and performance. Load balancing and auto-scaling are implemented for all services, particularly focusing on availability for the product search service, which uses replicated databases.

4.Performance Optimization:

•To enhance user experience, we optimize image loading by generating smaller thumbnails and introduce an in-memory cache for popular product search queries, significantly speeding up response times.

5.Checkout Process:

•An event-driven architecture supports asynchronous order processing, allowing efficient management of high purchase volumes. The inventory service utilizes a key-value store for improved performance.

6.Global Resilience:

•Deploying across multiple data centers with global load balancing ensures traffic is directed efficiently and high availability is maintained, even during data center failures.

By the end of the lecture, all nonfunctional requirements have been addressed, resulting in a scalable and highly available e-commerce platform capable of accommodating tens of millions of users.