Project Overview: Genomic Data Analysis for Precision Medicine

Personalized medicine is a new approach to healthcare that customizes treatments for each person. Instead of a one-size-fits-all approach, doctors would use a person's unique genes, lifestyle, and environment to decide the best treatment.

The biggest challenge is handling all the genetic data. A single person's genetic code is huge—more than 100 gigabytes of data. This is way too much for standard computer systems to manage, so we need special, powerful technology to store and analyze it all.

Methodology: A Hybrid Big Data Engineering Pipeline

This project proposes developing a comprehensive data engineering pipeline based on a **Lambda Architecture**. This hybrid approach combines the strengths of both batch and real-time stream processing, providing a unified framework for managing and analyzing diverse data at scale.

The pipeline is structured into distinct stages—Ingestion, Storage, Processing (Batch & Streaming), and Serving—each leveraging open-source, academic-friendly technologies:

1. **Ingestion:**
    - **Problem:** The need to ingest both large historical genomic datasets and continuous, high-velocity clinical data streams from various sources.
    - **Dataset** (**1000 Genomes Project):** This was a real research project that aimed to create a detailed catalog of human genetic variation. The project sequenced the genomes of more than 2,500 people from various populations worldwide. The sheer volume of data generated by this project is why the abstract uses it as an example of the "big data" problem in precision medicine.
    - **Methodology: Apache Kafka** is employed as a high-throughput, fault-tolerant messaging system for real-time clinical data (e.g., patient vitals, lab results). For massive genomic files (e.g., VCF, BAM).
2. **Storage:**
    - **Problem:** Securely and scalably persisting petabyte-scale raw and processed data while ensuring fault tolerance and privacy compliance.
    - **Methodology:** The **Hadoop Distributed File System (HDFS)** serves as the primary distributed storage layer.
3. **Processing (Batch & Streaming):**
    - **Problem:** Cleaning, normalizing, transforming, and enriching both static genomic data and dynamic clinical streams for various analytical and machine learning applications.

- **Methodology: Apache Spark**, with its Python API **PySpark**. Spark is chosen for its speed, in-memory processing capabilities, and versatility across different workloads.
    - **Batch Processing (Genomic):** PySpark jobs read genomic data from HDFS, perform transformations such as filtering variants by quality scores or allele frequencies, and enrich them by joining with external annotation data (e.g., gene functions, disease associations). The processed data is then stored in optimized columnar formats like Parquet for efficient querying.
    - **Streaming Processing (Clinical):** PySpark Structured Streaming jobs continuously consume real-time clinical data from Kafka. These jobs will apply immediate transformations, such as calculating moving averages of vital signs, and can be configured with watermarking to handle late-arriving data while managing state efficiently.

4. **Serving:**
    - **Problem:** Making processed data and model outputs readily available for researchers, clinical decision support systems, and business intelligence tools.
    - **Methodology:** The processed data is made accessible for various downstream applications:
        - **Machine Learning Model Training:** The combined genomic and clinical data from HDFS is used to train predictive models (e.g., using Python's Scikit-learn) to forecast patient responses to drugs or identify disease risks.
        - **Real-Time Insights:** Processed streaming data can be used to generate immediate alerts for critical patient health statuses, enabling proactive care.
        - **Research Insights:** Spark SQL allows researchers to run ad-hoc queries on the integrated datasets to identify correlations between genetic variants and clinical traits, accelerating medical discoveries.
5. **Containerization & Orchestration:**
    - **Problem:** Ensuring the scalability, portability, and reprehensibility of the entire pipeline, without incurring cloud costs.
    - **Methodology: Docker** is used to containerize individual services (Hadoop, Spark, Kafka),.

      Kubernetes then orchestrates these containers, automating their deployment, scaling, and management.