Comprehensive Revision Notes for High-Level Design Class

Introduction to High-Level Design (HLD)

High-Level Design (HLD) refers to the architectural design of a system that outlines the core framework and functioning of the overall structure. This approach deals with the architecture of the machines or servers that run the software, as opposed to Low-Level Design (LLD), which focuses on the architecture of the code itself【6:13†source】.

Key Concepts in HLD:

1. **Scaling:**
   - **Vertical Scaling** involves enhancing the capacity of a single server by upgrading its hardware (e.g., adding more RAM or faster CPUs). This improves the server's ability to handle more data but is limited by the server's physical constraints and becomes expensive【6:18†source】.
   - **Horizontal Scaling** distributes the load across multiple servers. This is more complicated to manage but can handle large-scale data more efficiently and cost-effectively【6:4†source】.
2. **Distributed Systems:**
   - In distributed systems, data is spread across numerous machines to utilize computing power efficiently. Sorting tasks, for example, must be carefully managed when data is divided across many servers【6:1†source】.
   - Challenges include data consistency, network latency, and fault tolerance【6:0†source】.
3. **Network and Data Storage:**
   - Systems like those managed by large organizations (e.g., Google and Amazon) have complex network infrastructures with millions of servers worldwide. The data storage architecture considers factors like network issues, data replication, latency management, and machine failures【6:4†source】【6:5†source】.

speed of data access【6:16†source】.

Case Study: Delicious Website

The class discussed a real-world case study based on the Delicious (del.icio.us) website, originally a college project:

1. **Minimum Viable Product (MVP):**
   - Initial features for the Delicious site included user registration, login, and the ability to bookmark URLs and view bookmarks. This demonstrates a simple layer of functionality focusing on critical user needs【6:9†source】【6:15†source】.
2. **Scalability Challenges:**
   - As the number of users grows, from thousands to millions, the infrastructure must adapt to handle increased data loads and user interactions. Strategies include both vertical and horizontal scaling【6:14†source】.
3. **Data Overestimation:**
   - It's crucial to plan for more data than currently needed to create buffer allocations for future growth, ensuring system longevity without constant upgrades【6:12†source】.

Analogies and Key Takeaways:

- **Simple Problems at Scale:**
  - A task like sorting, easy for small datasets, poses significant challenges when scaled to petabyte levels across thousands of servers, requiring distributed sorting algorithms【6:0†source】.
- **Estimation in Design:**
  - Data size estimations, as demonstrated with bookmark storage, teach the importance of planning resources for scalability【6:12†source】.
- **Layered Architecture in Networks:**
  - Using multiple layers of servers (e.g., DNS architecture) helps minimize latency by keeping data closer to end users, thus enhancing overall

These concepts prepared students for developing robust architectures capable of handling significant data loads and understanding the comprehensive roles involved in designing distributed systems.