**CASE STUDY**

**Case 1: Live Journal**

The case study describes how Livejournal engineers solved the problem of managing RAM resources on their web servers to handle increasing website traffic. Here's a summary of the main points:  
  
The Challenge: As website traffic increased, LiveJournal needed to install more web servers to keep up with the demand. However, each server stored a duplicate of its most frequently used SQL queries in RAM, resulting in wasteful memory utilization.   
  
Solution: To boost efficiency and minimize database load, LiveJournal’s creators created Memcache, a method for caching frequently performed queries. They generated a unique signature or hash for each SQL query to detect duplicates across the web servers.  
  
Memcache: The Memcache system enables web servers to query each other for stored query results, eliminating the requirement for SQL queries and reducing server overhead.   
  
Open source: The LiveJournal creators made Memcache available as open-source software letting other developers utilize and contribute to its development. They democratized the technique of controlling RAM resources and decreasing database load, which benefited the whole software development community.

Impact: Memcache was extensively adopted by other firms dealing with similar database issues and performance challenges. Its availability as open-source software and a standardized communication protocol makes it accessible and simple to install for developers.

**Case 2: Google MapReduce**

Google uses a tool to generate search indexes, one of the most significant examples of NoSQL. In this article, google described its technique for converting vast volumes of online data into search indexes utilizing low-cost production processors. The two major stages of Google MapReduce are map operation and the reduce operation. The map operation performs data extraction, transformation, and filtering whereas the reduce operation sorts, combines, and summarizes the findings.

Scalability: Google built up the map and reduction procedure to handle massive amounts of network data spread over thousands of low-cost core processors. This approach enabled Google to analyze data more efficiently and cost-effectively, laying the groundwork for the big data revolution.

Impact on programming: Google’s success with MapReduce encouraged a rethinking of functional programming concepts and their scalability in distributed environments. The capacity of operational software systems to expand to thousands of low-cost processors became clear.

Open-Source Adaptation: Google's usage of MapReduce prompted engineers from other firms or organizations, such as Yahoo, to develop an open-source version of the MapReduce framework. This raised awareness of the limits of traditional procedural programming, encouraging the use of functional programming techniques for computer jobs.

**Case 3: Google’s Big Table – A table with a Billion Rows and a Million Columns.**

When Google issued a white paper on the Bigtable system titled A Distributed Storage System for Structured Data, it prompted many developers to use the million-column table

Tables were created to hold the results of crawlers that retrieve HTML pages, and multimedia information for the web. The resulting data set was too huge to fit into a single relational database, so Google created a storage system that could be scaled without requiring expensive equipment, which they dubbed a distributed storage system.

Success and Impact: The Bigtable project was extremely successful, providing Google with a scalable and economical storage solution to store massive volumes of data. Its revolutionary design and architecture inspired developers across the world, laying the groundwork for the future development of distributed storage systems and NoSQL databases.

**Case 4: Amazon’s Dynamo – Accept an order 24 hours.**

Challenges: At first Amazon depended on a relational database. Shopping cart and checkout system. However, they quickly realized that the relational model was insufficient for their future business demands. Despite having limitless RDBMS software licenses and outstanding staff, they discovered that relational databases could not match the scalability and reliability needs of their 24/7 business model.

Impact On the NoSQL: The publication of the Amazon Dynamo article represented a watershed moment in the NoSQL moment. Dynamo challenged the current quo and existing best practices while relational database models dominated the database environment. Its emphasis on simplicity, scalability, and dependability opened the path for the adoption of key-value stores and other NoSQL solutions by enterprises wishing to construct high system availability and scalability.

Significance: Amazon’s Dynamo implementation has proven strong and effective, enabling the company to become one of the world’s most successful commercial enterprises. By creating a resilient and scalable infrastructure that can support their business model around the clock.

**Case 5: MarkLogic**

The MarkLogic case study describes the growth and profitability of a firm started in 2001 that specializes in handling enormous volumes of XML documents.

MarkLogic designed a cluster architecture made up of query nodes and document nodes that send queries to distant servers containing indexes XML.

Efficiency: Marklogic’s solution, which moves queries to documents rather than documents to query servers, allows for linear scalability with large bytes of documents. This design enabled them to efficiently process massive datasets.

Market Demand: MarkLogic’s solutions find interest in a variety of industries, including US federal government systems that store terabytes of intelligence data and huge publishing companies that want to store and search XML content.

Utilized Technologies: Initially Marklogic engineers utilized Xquery with REST to develop apps. However, later versions also offer Java and other language interfaces allowing them to service a larger developer base.