1. Reliable, Scalable, and Maintainable Applications

Many applications today are *data-intensive*, as opposed to *compute-intensive*.

For example, many applications need to:

- Store data so that they, or another application, can find it again later (*databases*)
- Remember the result of an expensive operation, to speed up reads (*caches*)
- Allow users to search data by keyword or filter it in various ways (*search indexes*)
- Send a message to another process, to be handled asynchronously (*stream processing*)
- Periodically crunch a large amount of accumulated data (*batch processing*)

You usually create a new, special-purpose data system from smaller, general-purpose components.

Reliability is continuing to work correctly, even when things go wrong. The things that can go wrong are called *faults*, and systems that anticipate faults and can cope with them are called *fault-tolerant* or *resilient*. A fault is usually defined as one component of the system deviating from its spec, whereas a *failure* is when the system, as a whole, stops providing the required service to the user.

Scalability is the term we use to describe a system's ability to cope with increased load. Scalability is based on load parameters and performance.

Performance Measure: The measures of *95th*, *99th*, and *99.9th* percentiles are common (abbreviated *p95*, *p99*, and *p999*). They are the response time thresholds at which 95%, 99%, or 99.9% of requests are faster than that particular threshold. For example, if the 95th percentile response time is 1.5 seconds, that means 95 out of 100 requests take less than 1.5 seconds, and 5 out of 100 requests take 1.5 seconds or more.

Approaches for coping with Load: S*caling up* (*vertical scaling*, moving to a more powerful machine) and *scaling out* (*horizontal scaling*, distributing the load across multiple smaller machines).

Accidental Complexity Vs Essential Complexity: We define complexity as accidental if it is not inherent in the problem that the software solves (as seen by the users) but arises only from the implementation. If the complexity is inherent to the problem, it is called the Accidental Complexity.