Designing Data-Intensive Applications

Chapter 1: Reliable, Scalable, and Maintainable Applications

Book Organization

- This book is organized into three parts
    - Part I is Foundations of Data Systems, and introduces the topic and building blocks
    - Part II is Distributed Data, and discusses the challenges when scaling to multiple machines
    - Part III is Derived Data, and discusses integrating data, including batch and stream processing

Introduction

- Most applications are data-intensive, not compute-intensive
- Data systems are such a successful abstraction that we use them all of the time without thinking about it
- This book will talk about the principles and practicalities of data systems. What do they have in common, what distinguishes them, and how do they achieve their characteristics
- The three main concerns for data systems are reliability, scalability, and maintainability

Reliability

- Simple definition of reliability is continuing to work correctly even when things go wrong
- Hardware Faults
    - Single machines are made more resilient through redundant hardware
    - Larger data and computing demands have driven move to multi-machine redundancy, often using software fault-tolerance techniques
- Software Errors
    - Hard to detect, can lie dormant until unusual set of circumstances
    - Can have a systematic error or cascading failures which cause multiple system failures, unlike hardware faults
    - No quick solution -- a set of small things each help: planning, thorough testing, process isolation, allowing crash & restart, monitoring system behavior
- Human Errors
    - Can combine several approaches to deal with human error, including:
        - Minimize opportunities for human error
        - Decouple places where people make the most mistakes from places where mistakes can cause failures
        - Allow quick & easy recovery from problems
        - Use detailed and clear monitoring
        - Provide good management & training

Scalability

- Even a system working well won't necessarily be reliable with 10x users

- Planning for scalability means asking if the system grows in a particular way, what are the options for coping with that growth
- Describing Load
    - Use metrics called *load parameters*, e.g. post tweet averages 4.6k requests/sec, peak of 12k requests/sec
    - Example of Twitter’s different designs to deal with updating home timelines
- Describing Performance
    - Common concerns are response time and throughput
    - Metrics like response time are reported in percentiles, e.g. median, 95th, 99th
- Approaches for Coping with Load
    - Can scale up or scale out
    - Stateless services are easy to scale out, but scaling out stateful systems can introduce a lot of complexity

Maintainability

- Operability: Making Life Easy for Operations
    - Good operability includes:
        - Providing visibility into the runtime behavior and internals
        - Support for automation and integration with standard tools
        - Self-healing
- Simplicity: Managing Complexity
    - Good to remove *accidental* complexity (not inherent in problem, just the implementation)
    - A good abstraction hides implementation details behind easy to understand interface
- Evolvability: Making Change Easy
    - Agile working patterns provide a framework for adapting to change
    - Evolvability is defined as agility at a large data system level
    - Simplicity and good abstractions can go a long way toward evolvability