Uni Distributed Systems Notes

Notes for the distributed systems course at HdM Stuttgart

Felicitas Pojtinger

2022-10-07

Contents

1 Introduction **2**
1.1 Contributing . . . 2
1.2 License . . . 2

2 Overview **3**
2.1 Goals . . . 3
2.2 Definition of Distributed Systems . . . 3
2.3 Emergence . . . 3
2.4 Emergent Failure Modes . . . 3
2.5 Why are Distributed Systems difficult to understand? . . . 3
2.6 Why Distribute? . . . 4
2.7 Scale and Distributions: Power Laws . . . 4
2.8 Characteristics of Distributed Systems . . . 4
2.9 The Eight Fallacies of Distributes Computing . . . 4
2.10 Programming Languages and Distributes Systems . . . 4
2.11 History of Distributed Systems . . . 5
2.12 Metcalfe's Law (Network Effects) . . . 5
2.13 Security in Distributed Systems . . . 5
2.14 Theoretial Foundations of Distributed Systems . . . 5
2.15 Distributed System Design . . . 6
2.16 Compnents of Distributed Operating Systems . . . 6
2.17 The Transparency Dogma . . . 7
2.18 Distribution Transparencies . . . 7
2.19 Classification . . . 7
2.20 RPC type Middleware . . . 8
2.21 Distributed Objects . . . 8
2.22 Distributed Computing Frameworks . . . 8

1 Introduction

1.1 Contributing

These study materials are heavily based on professor Kriha's "Verteilte Systeme" lecture at HdM Stuttgart and prior work of fellow students.

Found an error or have a suggestion? Please open an issue on GitHub (github.com/pojntfx/uni-distributedsystems-notes):

Figure 1: QR code to source repository

If you like the study materials, a GitHub star is always appreciated :)

1.2 License

Figure 2: AGPL-3.0 license badge

2 Overview

2.1 Goals

- Basic concepts
- Different programming models
- Theoretical foundation of computability
- Design of distributed systems
- Hardware and failure constraints
- How to build middleware in distributed systems

2.2 Definition of Distributed Systems

> Independent agents repeatedly interacting in a way that a coherent behavior ("system") **emerges**. Events happen concurrently and parallel.

Why emerges? You haven't watched the movie if you looked at every frame!

2.3 Emergence

- **Strong**: We cannot predict what will emerge (game of life)
- **Weak**: Things are combined by simple principles but the result suprises (flock of birds)
- **Evolutionary**: Complex but robust (egg to human)
- **Constructed**: Complex but often not robust (distributed systems; emergent Failure Modes: Cascading failures in constructed emergence)

2.4 Emergent Failure Modes

See Laura Nolan (black swans) on YouTube!

2.5 Why are Distributed Systems difficult to understand?

> See the slides for details on these first principles.

- Emergence
- We have a single machine view
- Errors are the fabric
- There is no free lunch
- Is total end-to-end system engineering

2.6 Why Distribute?

- **Robustness/Resilience**: Avoid single points of failures with replication
- **Performance**: Split processing into independent parts
- **Scalability/Throughput**: Allow millions of requests/sec
- **Security**: Create different security domains
- **Price per Request**: Use cheaper horizontal scaling or free resources

2.7 Scale and Distributions: Power Laws

There is a tendency that the big ones (e.g. Google) will become even bigger!

2.8 Characteristics of Distributed Systems

- Influence of distribution topology and remoteness
- Emergent behaviors, concurrent events
- Few analytics solutions, few model-based approaches
- Heterogeneous components
- No global time
- A strong need for security
- Concurrency, parallelism and replication
- Failure models define everything!

2.9 The Eight Fallacies of Distributes Computing

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn’t change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

2.10 Programming Languages and Distributes Systems

Message Camp

- Simple CRUD interface. Message content is interface.
- Coarse grained messages (documents)
- Programmers deal with remoteness directly
- Events based or REST architectures

Transparency Camp

- Hide remoteness from programmer
- Create type-safe interfaces and calls
- Hide security, storage and transactions behind frameworks (.NET, EJB etc.)
- Think distributed systems as a programming model

2.11 History of Distributed Systems

- 50s-80s: Basic papers on time, consensus, computability etc.
- 90s: Connecting intranet applications (CORBA, COM), programming models dominate
- 2000s: P2P, large social sites emerge, message passing, batch processing, eventual consistency & RAM replaces disk
- 2010s: Warehouse scale, fan out architecutres, realtime stream processing, flash memory, network performance, microservices & serverless, applications that run on the network adapter etc.

2.12 Metcalfe's Law (Network Effects)

- The usefulness of a network grows by the square of the number of users (one single fax machine is useless - if there are two fax machines it becomes important!)
- There can be only one!

2.13 Security in Distributed Systems

> "The company end where I don't have control over the cryptography"

2.14 Theoretial Foundations of Distributed Systems

- No global time (logical clocks, vector clocks)
- FLP theorem of asynchronous systems
- The problem of failure detection and timeout
- Concurrency and deadlocks

- CAP theorem: Consistency, availability and partitioning: Choose only two!
- End-to-end argument: Where in the application do we put the logic?
- Consensus, leader selection etc.

2.15 Distributed System Design

- Common Problems (performance, fail-over, maintenance, policies, security integration)
- Information Architecture (define and qualify the information fragments and flows)
- Distribution Architecture (create a map of all participating systems and their quality of service)

2.16 Compnents of Distributed Operating Systems

From top to bottom:

First layer

- Data Analysis and Request Processing Applications

Second layer

- APIs

Third layer

- Scheduler
- Queue
- Log service
- Notification services
- Locking services
- Fragment handler
- Memory cache
- Key/value store
- Distributed file system
- Load balancer
- IP service realocator
- Membership service

- Failure detector

Fourth layer

- Map reduce
- Consistent hashing
- Consensus algorithms
- Optimistic replies

Fifth layer

- Failure models

2.17 The Transparency Dogma

Middleware is supposed to hide remoteness and concurency by hiding distribution behind local programming language constructs.

2.18 Distribution Transparencies

- **Access**: Mask differences in languages and data representtation
- **Failure**: Mask failures to enable fault tolerance
- **Scalability**: Intelligent load-balancing of requests
- **Redundancy**: Transparent replication of data
- **Location**: Use logical, not physical names to access services
- **Migration**: Hide the true liocation of a service
- **Persistence**
- **Sharding**
- **Transactions**
- **Security**
- **Monitoring**

2.19 Classification

- Sockets
- RPC
- Object Request Brokers (CORBA, RMI)
- Message Oriented Middleware
- Web-Services

- Frameworks
- P2P
- Agent based (Jini)
- Tuple-Spaces
- Warehouse-Computing Architectures

2.20 RPC type Middleware

- E.g. Sun-RPC, Apache Thrift, gRPC
- Main idea: Allow function calls across languages with concurrent and parallel processing of requests
- Has generators to create specific glue code
- Directories, file systems etc. can be built on it

2.21 Distributed Objects

CORBA

- Object request broker
- Multi-language support
- Has an IDL
- Wire protocol: IIOP, GIOP

RMI

- Java only
- Lightweight method call semantics
- Java implementations
- Wire protocol: IIOP, GIOP

2.22 Distributed Computing Frameworks

- Objects are to granular
- Seperation of concerns and contet
- EJB, COM+ etc.