Byte-Sized

Computer Science for Data People

Part 1: Scalability

Principles of Successful Tech. Products

① **Scalability**

Can the product handle growing data volumes?

② **Maintainability**

Can other people work on the system productively?

③ **Reliability**

Can the product tolerate hardware, software, and human faults?

Framework adapted from Designing Data-Intensive Applications by Martin Kleppmann

Aspects of Scalability

Structure  Ability to handle new requirements without rewriting existing work

Speed Ability to handle larger volumes of data without slowing down

Space  Ability to handle larger volumes of data without breaking

Deep-Dive: Structure

Big Ideas

- **Don’t hardcode; pass your product’s ‘state’ instead**
- **Minimize interdependencies**
- **Handle collections, not single objects**

Copy-pasting code in 20+ different places

Making a single change in your config and everything else ‘just works’

Real-World Example

You’re midway through a forecasting engagement for a large retailer. They initially asked for Category-State-Week forecasts, but are now saying they need SKU-Store-Week forecasts. How many changes do you have to make to satisfy their request?

Related CS Concepts

- Scope & States
- Orthogonality & Coupling
- Iterators

Deep-Dive: Speed

Big Ideas

- **Don’t distribute the small stuff**
- **Minimize for-loops**
- **Double-check if a built-in exists before writing your own function**

Running nested for-loops on a Spark DataFrame

Using a built-in function on a filtered pandas DataFrame instead

Real-World Example

Your modeling pipeline was very fast when running on a small sample of data, but started to hang when you tried the same code on a larger dataframe. What should you do first?

Related CS Concepts

- Costs of Serialization
- Vectorization & Linear Algebra
- Big O Notation

Deep-Dive: Space

Big Ideas

- **Filter first (!!!)**
- **Compress your data by dropping columns, downcasting, and using sparse data structures**
- **Index, chunk, and distribute the big stuff**

Wasting money on expensive compute clusters

Profiling your code and compressing your data so you don't have to

Real-World Example

Your modeling pipeline was working fine when you were filtered on one region, but suddenly throws an out-of-memory error when include all regions in your dataframe. What steps would you take to solve this problem?

Related CS Concepts

- Lossless / Lossy Compression
- Parallelism & Concurrency
- Distributed Computing

