Distributed Filesystems

Tags	CompSci Low-Level
Status	In progress
Type	Topic

Goals

1. **Build a Mini Distributed File Store Simulator** in Rust within 10 consecutive days.
2. **Implement key features**: chunking, SHA-256 hashing, lossless compression (LZ4 or Huffman), configurable replication, XOR parity recovery, and a CLI.
3. **Exercise Rust fundamentals**: ownership, borrowing, lifetimes, `Arc` / `Mutex`, threads or async tasks, Send/Sync traits.
4. **Benchmark and document** throughput, storage overhead, and areas for optimisation.
5. **Produce artifacts**: public Git repo, release binary, README with usage examples, and a short retrospective.

Resources

Type	Title / Tool	Purpose
Paper	**Google File System (2003)**; **HDFS Architecture Guide**	Storage design reference
Paper	**Colossus/Spanner Papers (summaries)**	Modern object-store inspiration
Book	*Designing Data-Intensive Applications* – ch. 2, 10	Distributed-file-system theory
Book	*The Rust Programming Language* – ch. 4, 15, 16	Ownership, lifetimes, concurrency

Crate	`clap`, `sha2`, `lz4_flex`, `reed-solomon-erasure`	CLI, hashing, compression, parity
Crate	`tokio` or `std::thread` + `crossbeam`	Async or thread pools
Tool	`cargo flamegraph`, `hyperfine`	Profiling and benchmarking
Blog	MinIO Erasure Coding Docs	Parity layout ideas

Project

Day-by-Day Outline (25–30 min slots scale-able to 3–4 h)

Day	Focus & Checkpoints
1	Read GFS & HDFS papers; decide chunk size, replication, parity; initialise Git/Cargo project
2	Scaffold CLI (`dfs-sim`) with commands: `put`, `get`, `node ls`
3	Implement file-to-chunk splitter and SHA-256 hashes; unit tests
4	Add in-memory `Node` and round-robin replication; basic catalog
5	Integrate compression crate; measure ratios on sample files
6	Implement XOR parity per block set; simulate node failure & recovery
7	Build download path: verify checksum, decompress, reassemble file
8	Add simple benchmarks and logging; profile hot paths
9	Polish code, error handling, parallel transfers (threads or Tokio)
10	Write README usage section, performance results, retrospective, tag v0.1.0

Stretch Goals (fit as time allows)

- Swap internal calls to gRPC streaming APIs.
- Prototype consistent-hash placement.
- Replace XOR with Reed-Solomon erasure coding.
- Compare your simulator's API to MinIO's S3 layer.

How it should look like?

```
dfs-sim/
├─ Cargo.toml
├─ README.md      – quick-start + design notes
└─ src/
   ├─ main.rs      – CLI
   ├─ cli.rs
   ├─ catalog.rs
   ├─ node.rs
   ├─ chunker.rs   – split + SHA-256
   ├─ compressor.rs
   ├─ parity.rs
   └─ util.rs
```

Run a local cluster

```
# three storage nodes
dfs-sim node --id n1 --listen 127.0.0.1:4001
dfs-sim node --id n2 --listen 127.0.0.1:4002
dfs-sim node --id n3 --listen 127.0.0.1:4003

# coordinator
dfs-sim coord \\
  --nodes 127.0.0.1:4001,127.0.0.1:4002,127.0.0.1:4003 \\
  --replication 2 --parity xor
```

Basic CLI

```
dfs-sim put  ./large.iso  --name iso/ubuntu.iso
dfs-sim get  iso/ubuntu.iso  --out ./restore.iso
```

If a node is offline, download still succeeds via XOR rebuild.