SPARK I/O AND FILE LAYOUT: PARTITIONING BEST PRACTICES & PARQUET INTRODUCTION

AGENDA

SPARK I/O BASICS

Apache Spark I/O handles efficient data read/write across distributed systems.

Key Aspects:

- Distributed cluster processing
- Formats: CSV, JSON, Parquet, ORC
- Storage: HDFS, S3, local
- In-memory optimization

Core Components:

- DataSource API: Schema inference
- Partitioning: Parallelism
- Shuffling: Data redistribution

SPARK I/O PIPELINE

Principles:

- Partitioning: Parallel chunks
- Locality: Compute near data
- Fault Tolerance: Lineage
- Caching: Memory storage

Operations:

- Read/Write: With partitioning
- Streaming: Real-time
- Connectors: Kafka, etc.

PARTITIONING ESSENTIALS

What & Why?
Divides data for parallel processing, boosting performance.

Types:

- Input: File splits (~128MB)
- Shuffle: Transformations
- Output: By columns/keys

Strategies:

- Range/Hash/File/Custom

Mechanics:

- Repartition/Coalesce
- Bucketing for joins
- Pruning: Skip irrelevant data

PERFORMANCE IMPACT

Benefits:

- Parallelism: Full cluster use
- Resource Efficiency: Even load

Risks:

- Too Many: Overhead
- Skew: Imbalance
- Memory: Overload

Mitigation:

- Query alignment
- Monitor skew

BEST PRACTICES: SIZE & STRATEGIES

Optimal Size: 100MB–1GB per partition (balance overhead/memory).

Strategies:

- Filter columns (date/region)
- Avoid high cardinality
- File example: year/month/day

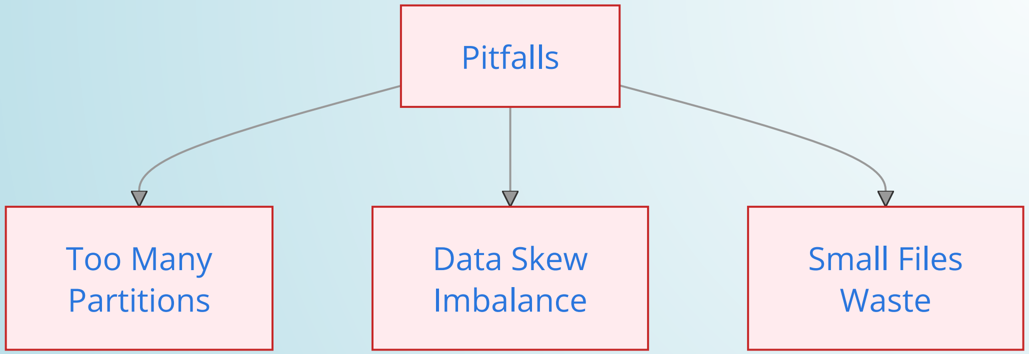
BEST PRACTICES: PRUNING & PITFALLS

Pruning:

- Pushdown filters for I/O savings (50-90% reduction)
- Use in WHERE clauses on partition columns

Pitfalls:

- Too many: Metadata overhead
- Skew: Use salting
- Small files: Coalesce
- Deep nesting: Limits effectiveness



BEST PRACTICES: SUMMARY & CONFIG

Core Guidelines:

- Select columns wisely: Frequent filters
- Size appropriately: 100MB–1GB
- Keep shallow: 2-3 levels max
- Query-driven: Match patterns
- Monitor: Skew, files, times

Key Configurations:

Parameter	Purpose	Rec.
maxPartitionBytes	Input size	100MB–1GB
shuffle.partitions	Shuffle	200–2000
adaptive.enabled	Dynamic	true

PARQUET INTRODUCTION

What is Parquet?
Columnar format for Spark/big data analytics.

Why Use It?:

- Columnar: Read specific columns
- Compression: 50-95% savings
- Pushdown: File-level filters
- Schema Evolution: Parquet supports backward/forward compatibility, allowing addition, deletion, or renaming of columns without rewriting entire datasets, ideal for evolving data pipelines.

vs. Row Formats:

Feature	CSV/JSON	Parquet
Storage	Large	Smaller
Speed	Slow	Fast
Schema	Loose	Strong
Opt.	Basic	Advanced

PARQUET STRUCTURE: LAYERS

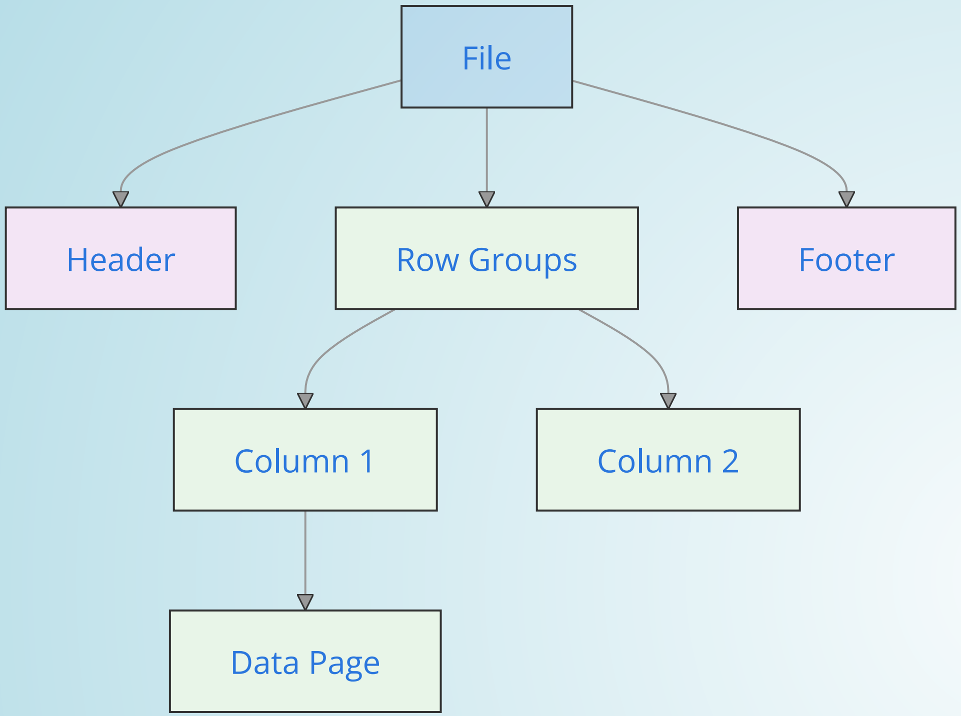
Key Layers:

1. Header: Magic bytes
2. Row Groups: 128MB–1GB chunks
3. Column Chunks: Per column
4. Pages: ~1MB units
5. Footer: Schema/stats

Features:

- Encoding: RLE/Dictionary
- Stats: Min/max for pruning
- Types: Nested support

PARQUET STRUCTURE: DIAGRAM



KEY TAKEAWAYS

Partitioning:

- Size: 100MB–1GB
- Prune: Filter columns
- Avoid: Skew/high cardinality
- Monitor: Query patterns

Parquet:

- Columnar efficiency
- Compression gains
- 10-100x faster queries
- For analytics/large data

Checklist:

- Analyze patterns
- Optimize sizes/compression
- Test pruning
- Monitor performance

WHEN TO USE & NEXT STEPS

Use Cases:

- Analytics/reporting
- Data lakes
- ETL with aggregations
- Evolving Schemas: Ideal for datasets where structure changes over time, as Parquet handles schema modifications efficiently without data loss or full reprocessing.

Implementation:

- Low-cardinality partitions
- Snappy compression
- Adaptive Spark enabled
- Iterate on metrics

CONCLUSION

- **Partitioning**: Aim for 100MB–1GB partitions, use low-cardinality columns for pruning, monitor for skew.
- **Parquet**: Columnar storage for efficient analytics, compression, and schema evolution.

Apply these practices to optimize your Spark workflows and achieve better performance!