Scalability Considerations

Current Performance Overview

My current pipeline efficiently processes a single folder with 10 sensor data files, totalling approximately 769,752 rows, in just 3 minutes. This translates to about 4,276 rows per second. However, scaling this to handle millions of files per day—such as 1 million files (around 38.475 billion rows) or 10 million files (384.75 billion rows)—reveals a challenge. Without optimization, it would take long time, far exceeding practical needs.

Proposed Scalable Pipeline Design

To address this, I envision a horizontally scalable pipeline that grows with demand by adding more resources. Here's my design:

- **Ingestion Layer**: I would implement Apache Kafka to manage the influx of millions of files daily. Kafka would act as a robust messaging system, receiving file events from a cloud storage solution like AWS S3. With 100 or more partitions, it would distribute the workload across multiple consumer nodes, ensuring smooth handling even at peak loads.
- **Processing Layer**: I'd use Apache Spark to process the data in parallel. Spark would read batches of files from Kafka, perform validation and transformation tasks (like checking for nulls or temperature ranges), calculate aggregates, and prepare data for storage. A cluster with 50 to 500 executors could handle 38.475 million to 384.75 million rows per day per node, scaling as needed.
- **Storage Layer**: I'd continue using PostgreSQL, hosted on AWS RDS, but enhance it with yearly partitions and data compression to manage the vast dataset efficiently. This setup would support billions of rows without performance drops.
- **Orchestration**: Apache Airflow would schedule and monitor the pipeline, ensuring timely execution and triggering auto-scaling when the workload increases.

This design allows the system to scale horizontally by adding more Kafka brokers, Spark nodes, and database instances, adapting to millions of files daily.

Optimizations for High-Volume Data Handling

To ensure the pipeline can ingest and process such large volumes effectively, I'd implement the following improvements:

- **Batching**: Process files in batches of 10,000, aiming for each batch to complete in under 10 minutes. This reduces overhead and speeds up handling.
- **Parallel Processing**: Use 100 to 1,000 Spark partitions to divide the workload, allowing multiple tasks to run simultaneously and scale with the cluster size.

- **Data Compression**: Compress CSV files before ingestion and use PostgreSQL's TOAST feature to shrink storage needs, potentially reducing 384.75 billion rows to 50-100 terabytes.
- **Efficient Inserts**: Employ asynchronous batch inserts with a connection pool, processing thousands of rows at once to maintain high throughput.
- **Optimized Validation**: Leverage Spark's DataFrame capabilities for parallel data validation, skipping slower full DataFrame operations on large datasets.
- **Monitoring and Scaling**: Integrate tools like Prometheus and Grafana to track performance metrics (e.g., CPU usage), with auto-scaling rules to add resources when usage hits 80%.
- **Fault Tolerance**: Ensure data integrity with Kafka's replication and Spark's fault recovery, minimizing risks during failures.

These optimizations would boost throughput to handle 4.4 million rows per second for 10 million files daily and ensure the system remains reliable under heavy load.