DATACOUCH

CREATING INFINITE POSSIBILITIES

DSAI

Data Science and AI Academy

Piramal Finance

Flink Training for Real-Time Data Engineering

Performance Tuning and Engineering Best Practices for Apache Flink

AGENDA

- Monitoring Flink jobs: Identifying backpressure and bottlenecks.
- Resource management: Allocating CPU, memory, and parallelism effectively.
- Data Serialization and Schema Management (e.g., using Avro).
- Structuring Flink projects for maintainability and testing.

Why Performance Tuning Matters in Flink

Cost Reduction

Pinterest cut streaming costs by 40% and onboarded 40% more jobs through effective tuning strategies

System Stability

Poor tuning causes backpressure, bottlenecks, and unstable clusters that affect entire data pipelines

Operational Excellence

Efficient tuning balances throughput, latency, and resource usage for real-time streaming success

Monitoring Flink Jobs: Spotting Backpressure & Bottlenecks

Understanding Backpressure

Backpressure occurs when CPU starvation or network delays cause task slowdowns throughout your streaming pipeline.

Use Flink UI's **Metrics & Checkpoints** tab to identify heavy operators and lagging subtasks effectively.

Key Warning Indicators:

- Rising checkpoint durations
- Skewed task load distribution
- Network buffer saturation

Visualising Backpressure: The Hot Node Phenomenon

Noisy Neighbours Problem

One job overconsuming CPU resources impacts all other jobs running on the cluster, creating cascading performance issues.

CPU Starvation Effects

CPU starvation leads to cascading delays and complete pipeline stalls, affecting downstream processing and data freshness.

Resource Management: CPU, Memory & Parallelism

01

Autoscaling Triggers

Autoscaling activates when CPU exceeds 75% for 15 minutes; manual scaling needed for other resource spikes

02

Operator Parallelism

Set operator parallelism based on resource intensity, not just application-level parallelism settings

03

Task Slot Ratio

Typical stable ratio: 4 operator subtasks per task slot; adjust based on workload intensity requirements

04

KeyBy Partitioning

Use KeyBy partitioning to evenly distribute workload and avoid data skew issues

Best Practices for Parallelism & Scaling

1

Balance Partitioning

Over-partitioning creates overhead; under-partitioning causes bottlenecks. Find the optimal balance for your workload.

2

Dynamic Tuning

Tune operator parallelism dynamically via runtime properties without requiring complete redeployment cycles.

3

Monitor Distribution

Monitor skewed key distributions causing uneven load and backpressure across your cluster nodes.

4

Task Chaining

Use task chaining to reduce network overhead but watch for resource contention between operators.

Profiling & Tools for Continuous Performance Insight

Async-profiler

CPU cycles, heap allocations, and flame graphs for precise hotspot detection and performance bottleneck analysis.

VisualVM

Live heap and CPU monitoring capabilities for interactive debugging and real-time performance assessment.

jemalloc + jeprof

Memory profiling to detect leaks over time and understand allocation patterns in long-running applications.

Eclipse Memory Analyser

Deep JVM heap dump analysis for comprehensive memory issue diagnosis and resolution strategies.

Why Data Serialization & Schema Management Matter

Performance

Efficient serialization reduces latency and resource consumption in streaming pipelines

Compatibility
Schema management ensures data compatibility and evolution without system downtime

Flink Power

Native support for serialization and schema evolution creates robust streaming applications

Avro & Schema Registry Integration in Flink

Compact Serialization

Avro provides compact, schema-based binary serialization with strong evolution support

Centralized Management

Schema Registry centralizes schema storage, versioning, and compatibility enforcement

Automatic Processing

Flink integrates with Schema Registry for automatic serialization/deserialization of Kafka topics

ⓘ Register Avro schema via REST API, then use ClouderaRegistryAvroKafkaRecordSerializationSchema in Flink

Testing Strategies for Flink Applications

Unit Testing

Test POJO serialization with Flink's PojoTestUtils.assertSerializedAsPojo() for validation

Integration Testing

Use embedded Kafka and Schema Registry to validate end-to-end serialization workflows

State Evolution Testing

Employ savepoint-based tests to verify state schema evolution and migration correctness

CI/CD Automation

Automate schema compatibility checks as part of continuous integration and deployment pipelines

Ready to Build Robust Flink Pipelines?

Schema-Driven Design

Embrace schema-driven design with Avro and Schema Registry for consistency

Clear Structure

Structure your Flink projects for clarity and comprehensive testability

Automated Validation

Automate serialization and schema validation in your CI/CD processes

Production Excellence

Unlock the full power of Flink's serialization and state management for production-grade streaming

ANY SUGGESTIONS?

THANK YOU