acceldata

www.acceldata.io

# Acceldata's ODP Spark with Gluten Velox

## Accelerating Big Data Processing Through Vectorized Execution

# Executive Summary

Organizations today demand faster insights from ever-growing volumes of data. While Apache Spark has become the de facto engine for distributed analytics, its traditional row-based execution model introduces inefficiencies that limit performance at scale.

Acceldata's **ODP Spark with Gluten Velox** represents a breakthrough: by integrating Intel's Gluten plugin and Meta's Velox vectorized execution engine, Spark workloads achieve 1–3x faster performance with significantly improved CPU, memory, and I/O efficiency.

In benchmark testing using the **TPC-DS suite on a 100GB dataset,** executed on a **3-node cluster (64 GB RAM, 12 vCPUs each)**, Acceldata ODP Spark with Gluten/Velox consistently delivered:

**1–3x query acceleration**
across complex aggregations, joins, and window functions

**20–30% reduction**
in CPU cycles per row processed

**15–20% lower**
memory allocation pressure

**Reduced shuffle and I/O overhead**
via optimized columnar storage and execution

## Quantified ROI Example

(1000-core Spark cluster)

| Metric | Without Gluten | With Gluten | Annual Savings |
|---|---|---|---|
| **Compute hours/ day** | 24,000 | 12,000–16,000 | $150K–$300K (AWS EC2 rates) |
| **Failed jobs due to OOM** | 5–10% | 2–3% | $50K–$100K (engineering time + re-runs) |
| **Time-to-insight** | 4 hours | 1.5–2 hours | Intangible (faster decisions) |
| **Infrastructure scaling needs** | Year 2 | Year 3–4 | $500K–$1M (deferred CapEx) |

# Gluten-Velox ROI Analysis

## 1000-core Spark cluster - Annual Comparison

### Compute Hours / Day

| WITHOUT | WITH |
|---|---|
| 24,000 Hours | 12,000 - 16,000 Hours |

**$150K - $300K / Year**

### Failed Jobs (OOM)

| WITHOUT | WITH |
|---|---|
| 5 - 10% | 2 - 3% |

**$50K - $100K / Year**

### Time-to-Insight

| WITHOUT | WITH |
|---|---|
| 4 Hours | 1.5 - 2 Hours |

**Intangible Value**

### Infrastructure Scaling

| WITHOUT | WITH |
|---|---|
| Year 2 | Year 3-4 |

**$500K - $1M Deferred**

For enterprises running large-scale analytics, this translates into **lower infrastructure costs, faster time-to-insight, and improved operational efficiency** — without changing existing Spark applications.

# Spark + Velox Performance Analysis

## TPC-DS Benchmark Results - 100GB Dataset Analysis

### Cluster Configuration

| | | |
|---|---|---|
| **3 Nodes**<br>Hadoop Cluster | **Ubuntu 20.04**<br>Operating System | **750GB**<br>Storage per Node |
| **12 Cores**<br>CPU per Node | **64GB**<br>Memory per Node | **100GB**<br>TPC-DS Dataset |

# 1. Introduction

## 1.1 The Performance Challenge in Big Data Analytics

Modern analytical workloads demand processing of petabyte-scale datasets with sub-second response times. Traditional Spark's row-at-a-time processing model, while flexible and robust, introduces significant computational overhead through:

- Row-based tuple processing with high function call overhead
- JVM garbage collection pressure from object creation
- Inefficient CPU cache utilization
- Limited vectorization opportunities in the JVM runtime

## 1.2 The Vectorization Solution

Vectorized execution engines process data in batches (vectors) rather than individual rows, enabling:

- SIMD (Single Instruction, Multiple Data) instruction utilization
- Improved CPU cache locality
- Reduced function call overhead
- Native code execution performance
- Advanced columnar storage optimizations

# 2. Technical Architecture

## 2.1 Gluten Framework Overview

Gluten serves as an abstraction layer that enables Spark to leverage native execution engines while maintaining API compatibility. The architecture consists of:

### Plugin Interface Layer

- Seamless integration with Spark's Catalyst optimizer
- Rule-based transformation of Spark plans to native execution plans
- Fallback mechanisms for unsupported operations

### Native Execution Engine Integration

- Primary support for Meta's Velox engine
- Extensible architecture supporting multiple backends (Clickhouse, Arrow)
- Memory management bridge between JVM and native heap

### Columnar Data Exchange

- Apache Arrow-based data interchange format
- Zero-copy data transfers where possible
- Optimized serialization/deserialization

## 2.2 Velox Execution Engine

Velox, originally developed by Meta for Presto, provides the native vectorized execution runtime:

### Vectorized Operators

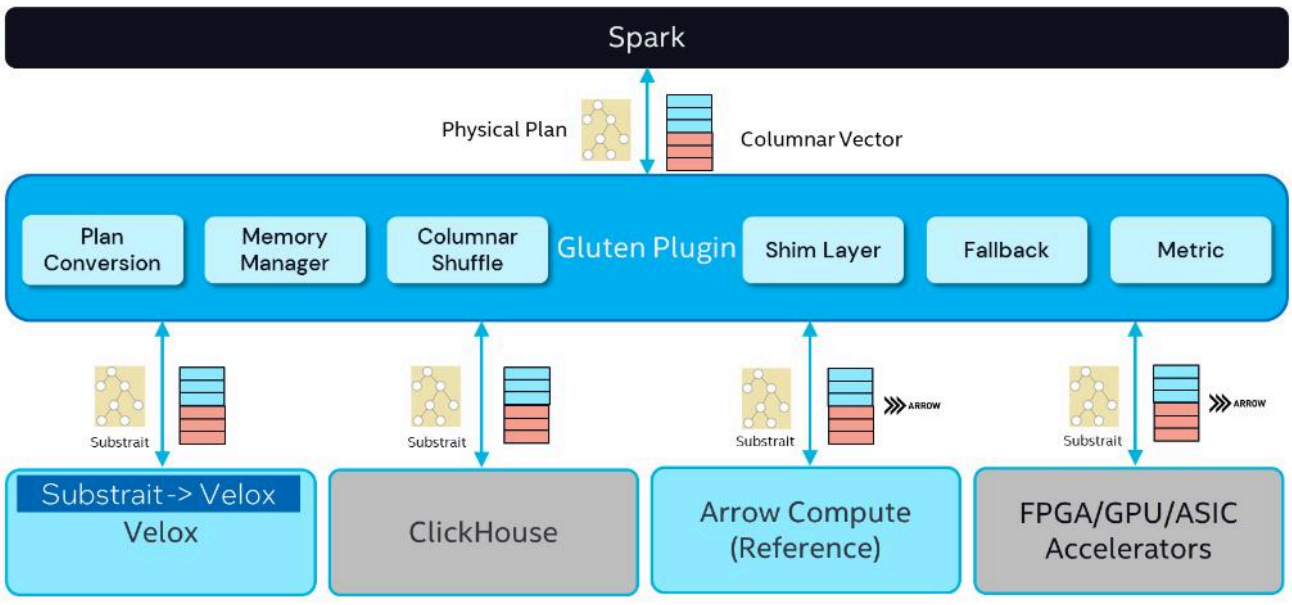
- Columnar batch processing (typically 1024-4096 rows per batch)
- Template-based code generation for type-specific operations
- Adaptive batch sizing based on memory pressure

### Memory Management

- Custom memory pools with NUMA awareness
- Efficient buffer reuse and recycling
- Memory pressure-based execution flow control

### Expression Evaluation

- Compiled expression evaluation with SIMD utilization
- Lazy evaluation and short-circuiting optimizations
- Common subexpression elimination



# 3. Performance Analysis

## 3.1 TPC-DS Benchmark Results

Comprehensive testing on TPC-DS benchmark queries demonstrates consistent performance improvements:

### Query Categories and Performance Gains

**Advanced Analytics (Q37 Q40 Q78 Q90 Q93)**

- Average speedup: 1.46x
- Peak improvement: 3.72x on Q93 (complex window functions)
- Primary benefit: Analytics operations

**Join-Intensive Queries (Q7, Q11, Q17, Q29, Q65)**

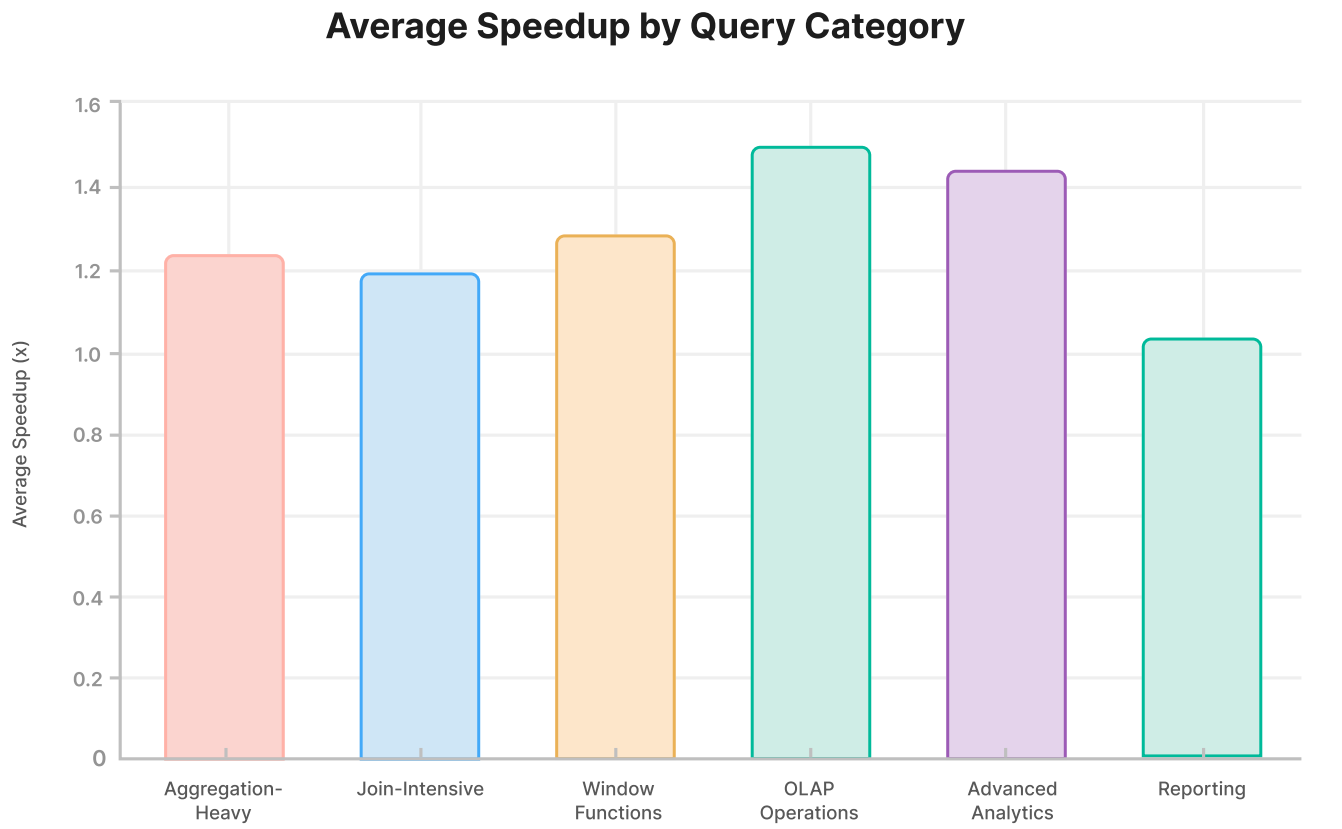
- Average speedup: 1.21x
- Peak improvement: 1.87x on Q29 (multi-table joins)
- Primary benefit: Columnar hash joins with bloom filter pushdown

**Aggregation-Heavy Queries (Q1, Q4, Q5, Q9,Q27, Q84)**

- Average speedup: 1.23x
- Peak improvement: 1.92x on Q5 (complex window functions)
- Primary benefit: Vectorized aggregation with hash table optimizations

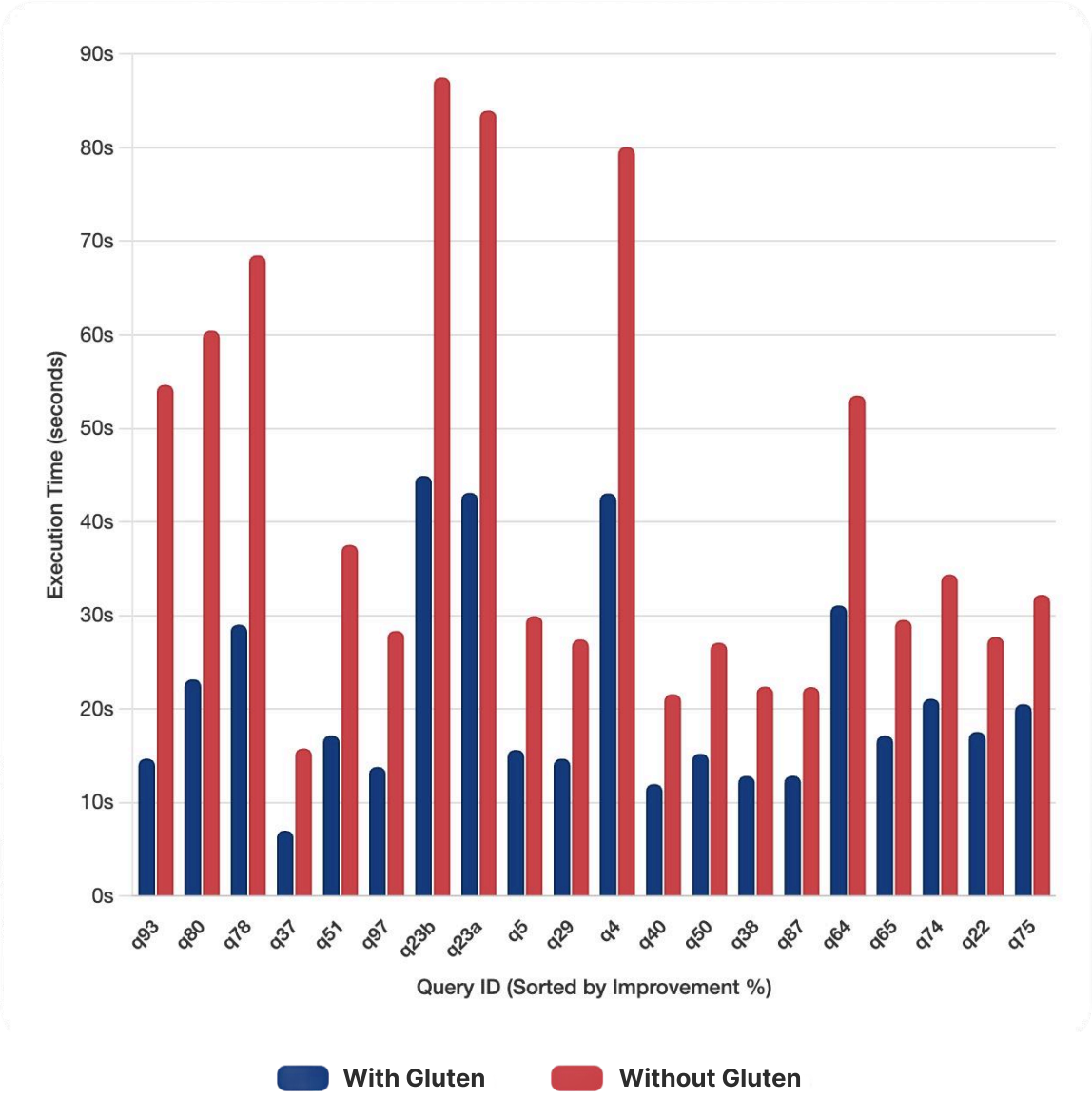
**Window Functions (Q12 Q21 Q51 Q67)**

- Average speedup: 1.27x
- Peak improvement: 2.19x on Q51 (regex operations)
- Primary benefit: SIMD string operations and dictionary encoding



## 🏆 Peak Achievements with Gluten

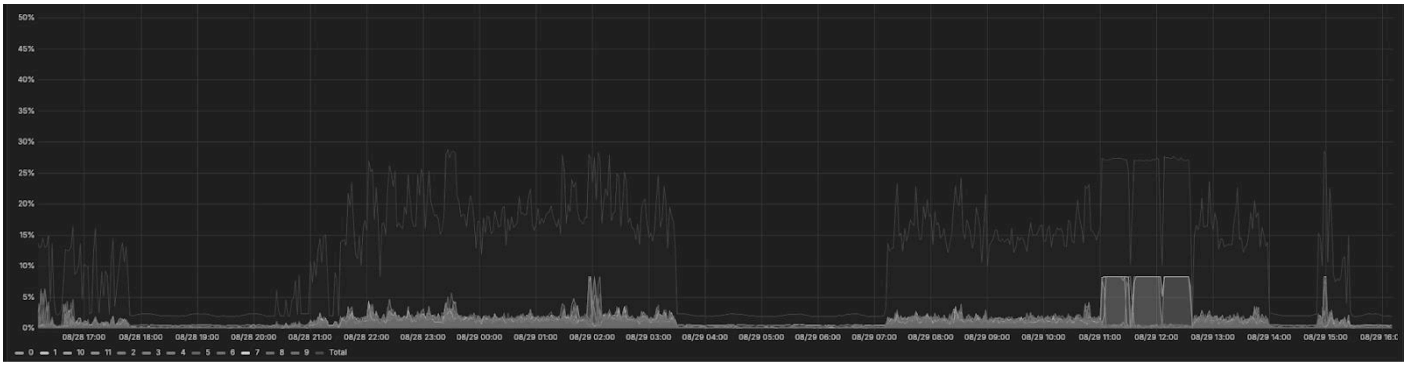
| | | |
|---|---|---|
| **3.72x**<br>**Best Overall Speedup**<br>q93 (Advanced Analytics) | **1.51x**<br>**Best Category Average**<br>OLAP Operations | **552s**<br>**Total Time Saved**<br>Across all 102 queries |
| **1.31x**<br>**Overall Average Speedup**<br>88 queries improved | **6**<br>**Excellent Performers**<br>Queries with 2x+ Speedup | **86%**<br>**Success Rate**<br>Queries showing Improvement |



## 3.2 Resource Utilization Improvements

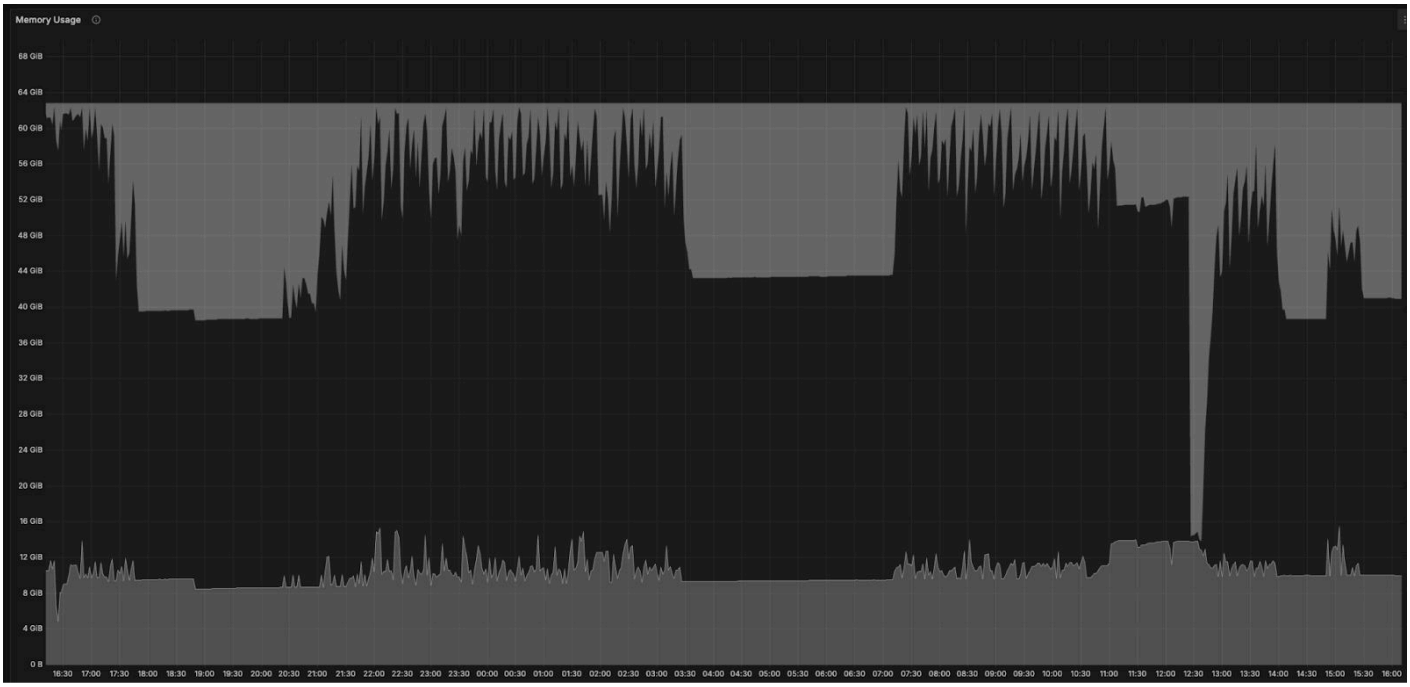
### CPU Efficiency

- 20-30% reduction in CPU cycles per processed row
- Better branch prediction through vectorized control flow
- Improved instruction cache hit rates through code locality



### Memory Performance

- 15-20% reduction in memory allocation pressure
- Reduced GC overhead through native memory management
- Improved memory bandwidth utilization through sequential access patterns



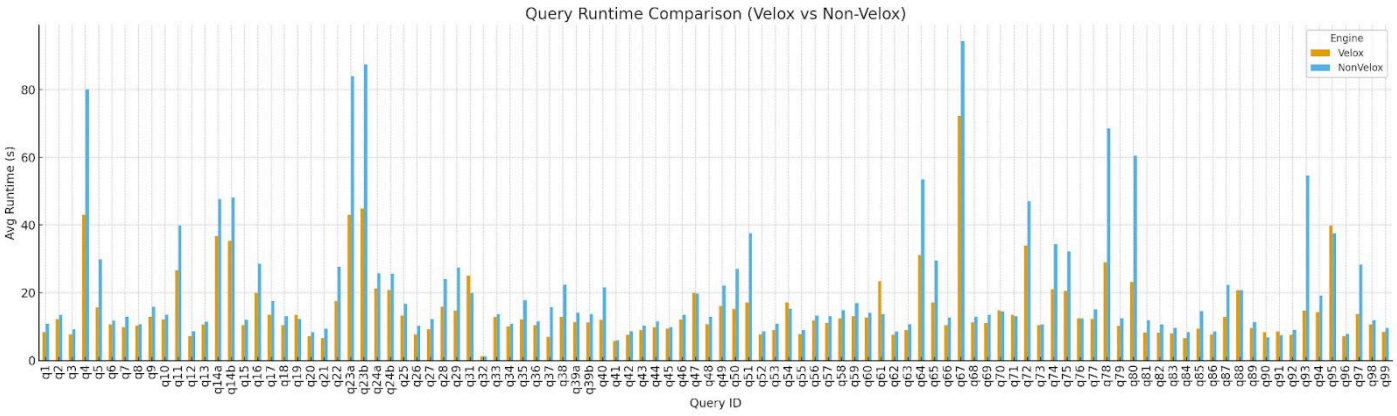

### I/O Optimization

- Enhanced predicate pushdown capabilities
- Reduced network shuffle through better data locality
- Improved compression ratios with columnar layouts

## 3.3 Workload-Specific Performance Characteristics

### OLAP Workloads

- Best suited for analytical queries with large data scans
- Significant gains on aggregation and complex expression evaluation
- Optimal performance on columnar storage formats (Parquet, ORC)



# Spark + Velox Performance Dashboard

TPC-DS Benchmark: Parquet cs ORC Format Comparison

Overview | Detailed Analysis | All Queries

**Overall Winner**

**Parquet**

Wins 95.1% of queries

**Performance Gain**

**11.6%**

Average improvement with Parquet

**Time Saved**

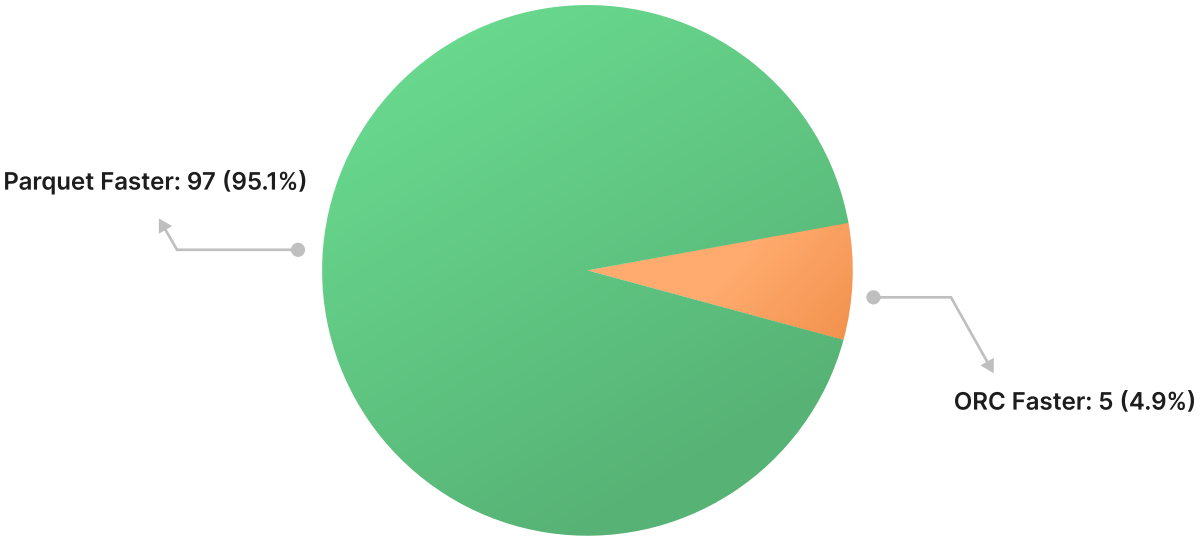
**201.3s**

Total time saved across all queries

**Queries Tested**

**102**

TPC-DS Benchmark queries




## Mixed Workloads

- Performance varies based on operation mix
- Adaptive execution planning based on cost estimates
- Fallback overhead minimal for unsupported operations

# 4. Implementation Considerations

## 4.1 Deployment Architecture

### Cluster Configuration Requirements

- Native library deployment across all executor nodes
- Shared library compatibility (glibc version >= 2.28, libstdc++ version >=11.2)
- Memory management tuning for dual JVM/native heaps

### Storage Format Optimization

- Parquet with optimized column layouts
- Row group sizing aligned with vectorized batch sizes
- Compression codec selection (ZSTD, LZ4) for decompression performance

## 4.2 Configuration Parameters

### Key Gluten Settings:

spark.gluten.enabled=true

spark.gluten.ras.costModel=true

spark.shuffle.manager=org.apache.spark.shuffle.columnar.ColumnarShuffleManager

spark.gluten.memory.overAcquiredMemoryRatio=0.3

spark.gluten.sql.columnar.backend.velox.memCacheSize=4g

**spark.gluten.memory.overAcquiredMemoryRatio :**

*What it does:*

- Controls how much extra memory Gluten can acquire beyond its initial allocation
- Value of 0.3 means Gluten can use up to 30% more memory than initially allocated
- Acts as a buffer for memory spikes during columnar operations

*How it works:*

```
If executor memory = 10GB and initial Gluten allocation = 4GB
With overAcquiredMemoryRatio = 0.3:
Maximum Gluten memory = 4GB + (4GB * 0.3) = 5.2GB
```

*When to adjust:*

**Increase (0.4-0.6) if you see:**

- Memory allocation failures during complex aggregations
- OOM errors in vectorized operations
- Performance degradation in large TPC-DS queries (like Q67, Q95)

**Decrease (0.1-0.2) if you see:**

- Frequent GC pressure
- Memory contention between Spark and Gluten
- Other executors getting OOM due to memory hogging

**spark.gluten.sql.columnar.backend.velox.memCacheSize:**

*What it does:*

- Sets the size of Velox's internal memory cache
- Acts as a buffer pool for vectorized operations
- Used for caching frequently accessed columnar data

*Memory hierarchy:*

```
Spark Executor Memory
├── JVM Heap (Spark operations)
├── Off-heap (Tungsten)
└── Gluten Memory
    ├── Velox Memory Cache (4GB in your config)
    ├── Computation buffers
    └── Intermediate results
```

*When to adjust:*

**Increase (6-8GB) if:**

- Working with large Parquet/ORC files
- High cache hit ratios in your workload
- Sufficient executor memory available (>16GB)
- Complex multi-stage TPC-DS queries

**Decrease (2-3GB) if:**

- Limited executor memory (<8GB)
- Simple queries with low reuse
- Memory pressure from other components

## 4.3 Compatibility Matrix

### Core SQL operations:

- SELECT, WHERE filters, PROJECT
- Joins: Broadcast Hash Join, Shuffle Hash Join, Sort-Merge Join, Nested Loop Join, Null-Aware Anti Join
- Aggregations: GROUP BY, ROLLUP, CUBE, HAVING
- Sorting and ordering (ORDER BY, LIMIT)
- UNION

### Mathematical and string functions:

- Arithmetic (+, -, *, /, %)
- Comparisons and logical expressions (=, <, >, AND, OR, NOT)
- String functions (substring, concat, length, trim, etc.)
- Conditional expressions (CASE, IF)
- Type casting

### Window functions and analytical operations:

- Ranking (RANK, DENSE_RANK, ROW_NUMBER)
- Aggregates over windows (e.g., moving averages, cumulative sums)

### Complex data types:

- Arrays, Maps, Structs
- Nested fields in projections and filters

### File formats and execution:

- Parquet: fully supported
- ORC: partial support
- Native columnar shuffle with Velox
- Spill-to-disk supported (`spark.gluten.sql.columnar.backend.velox.spillEnabled=true`)

# 5. Limitations and Fallbacks

**UDFs:**

- User Defined Functions (Scala/Python/Java UDFs) require JVM execution.
- Gluten automatically falls back to Spark JVM for these.

**Unsupported operators / plan nodes:**

- `ShuffleExchange` (some cases), `Unnest`, `Values`, `Top-N`, `EnforceSingleRow`, `PartitionedOutput`
- These fall back to Spark's JVM engine.

**Advanced SQL features:**

- ANSI mode not fully supported → certain queries will fallback.
- Some JSON/CSV functions and schema evolution scenarios may not be handled natively.
- Complex nested subqueries sometimes fall back.

**Streaming**

- Structured Streaming operations are not yet supported in Gluten+Velox.

**Function semantics mismatches:**

- A few math/string functions behave slightly differently than Spark's JVM version.
- Marked as "partial support" in docs; fallback may occur if strict compatibility is required.

**File formats:**

- Parquet: best support
- ORC: partial support
- CSV/JSON: limited; often fallback

# 6. Operational Deployment

## 6.1 Migration Strategy

Gluten/Velox is designed as a drop-in acceleration layer that requires no application code changes. The "migration strategy" is really about **operational deployment** rather than code migration.

**Infrastructure Deployment:** Gluten requires native libraries (C++ Velox runtime) distributed across all cluster nodes. In enterprise environments with security controls, this isn't trivial - you need to handle native library packaging, distribution, and permissions in Kerberos/SSL environments.

*Install prerequisites:*

```
sudo apt-get update
sudo apt-get install build-essential g++ python3-dev -y
wget https://archives.boost.io/release/1.84.0/source/boost_1_84_0.tar.gz
file boost_1_84_0.tar.gz
tar -xvzf boost_1_84_0.tar.gz
cd boost_1_84_0
./bootstrap.sh --with-libraries=context
sudo ./b2 install
echo "/usr/local/lib" | sudo tee /etc/ld.so.conf.d/boost.conf
sudo ldconfig
ls -l /usr/local/lib/libboost_context*
export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
```

The `boost::context` library is being used here, and other Boost components are not supported in this setup.

Boost is distributed under the **Boost Software License (BSL-1.0)**, which is a permissive, OSI-approved license similar to MIT

## 6.2 RollBack

No rollback plan is required since Gluten Velox supports Spark applications without any code changes. If a particular query is not supported by Velox, it will automatically fall back to Spark's default JVM-based plan.

## 6.3 Monitoring and Observability

### Key Metrics to Track

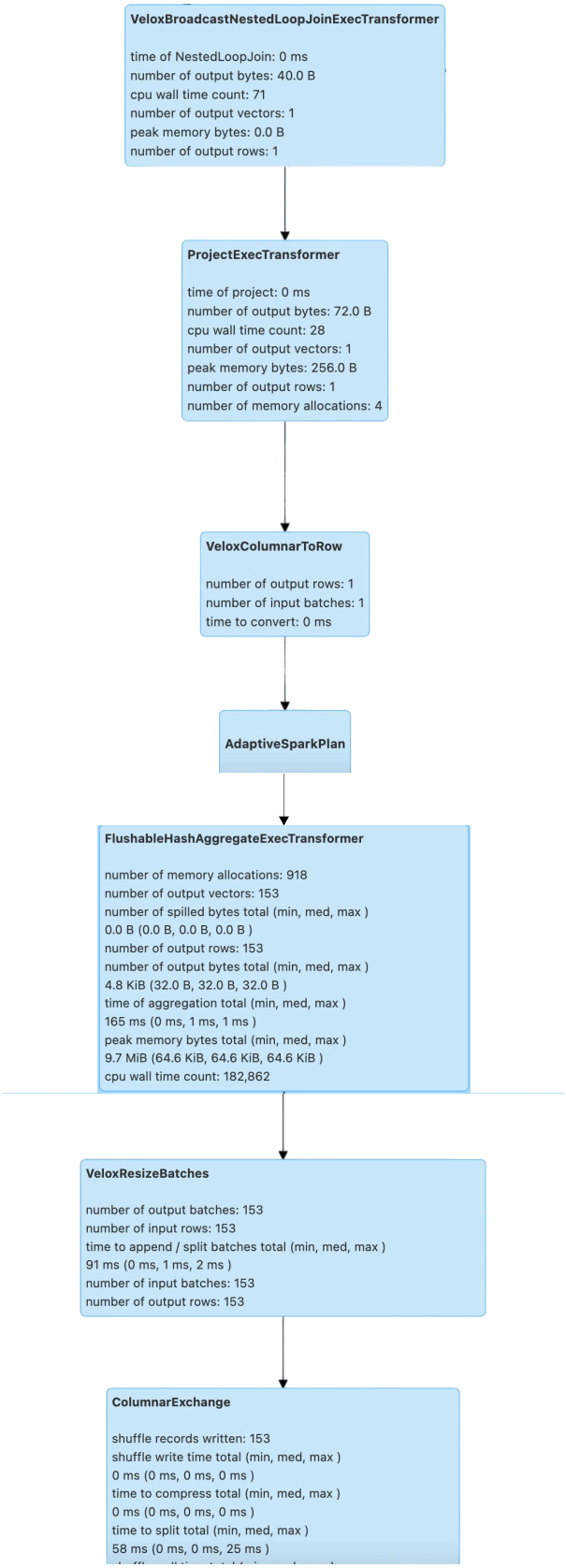
- Query execution time improvements
- Resource utilization (CPU, memory, I/O)
- Fallback operation frequency
- Memory allocation patterns

### Logging and Debugging

- Native execution plan logging
- Memory pool utilization tracking
- Performance regression detection

## Spark UI Metrics

**SQL Tab Plan Details:** Check the physical execution plan in Spark UI. Gluten-accelerated operators show up with distinct names like VeloxColumnarToRowExec, `VeloxHashAggregateExec,` or `VeloxProjectExec` instead of standard Spark operators.



**Stage Details:** Look for native execution indicators in stage metrics. Gluten stages typically show different memory usage patterns and often significantly reduced CPU time per task.

▾ Completed Stages (11)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

| Stage Id ▾ | Description | Submitted | Duration | Tasks: Succeeded/Total | Input | Output | Shuffle Read | Shuffle Write |
|---|---|---|---|---|---|---|---|---|
| 11 | SELECT promotions, total, cast(promotions AS DECIMAL(15, 4)) / cast(total AS DECIMAL(15, 4)... processLine at CliDriver.java:336 +details | 2025/09/12 15:29:05 | 0.4 s | 1/1 | | | 4.9 KiB | |
| 10 | SELECT promotions, total, cast(promotions AS DECIMAL(15, 4)) / cast(total AS DECIMAL(15, 4)... genShuffleDependency at VeloxSparkPlanExecApi.scala:542 +details | 2025/09/12 15:29:00 | 6 s | 153/153 | 2.9 GiB | | | 4.9 KiB |
| 9 | SELECT promotions, total, cast(promotions AS DECIMAL(15, 4)) / cast(total AS DECIMAL(15, 4)... collect at VeloxSparkPlanExecApi.scala:633 +details | 2025/09/12 15:28:59 | 0.3 s | 1/1 | | | 4.9 KiB | |
| 7 | SELECT promotions, total, cast(promotions AS DECIMAL(15, 4)) / cast(total AS DECIMAL(15, 4)... genShuffleDependency at VeloxSparkPlanExecApi.scala:542 +details | 2025/09/12 15:28:46 | 6 s | 153/153 | 2.5 GiB | | | 4.9 KiB |
| 6 | SELECT promotions, total, cast(promotions AS DECIMAL(15, 4)) / cast(total AS DECIMAL(15, 4)... genShuffleDependency at VeloxSparkPlanExecApi.scala:542 +details | 2025/09/12 15:28:46 | 7 s | 153/153 | 2.9 GiB | | | 4.9 KiB |
| 5 | SELECT promotions, total, cast(promotions AS DECIMAL(15, 4)) / cast(total AS DECIMAL(15, 4)... collect at VeloxSparkPlanExecApi.scala:633 +details | 2025/09/12 15:28:43 | 41 ms | 2/2 | 1972.1 KiB | | | |
| 4 | SELECT promotions, total, cast(promotions AS DECIMAL(15, 4)) / cast(total AS DECIMAL(15, 4)... collect at VeloxSparkPlanExecApi.scala:633 +details | 2025/09/12 15:28:43 | 0.1 s | 2/2 | 58.3 KiB | | | |
| 3 | SELECT promotions, total, cast(promotions AS DECIMAL(15, 4)) / cast(total AS DECIMAL(15, 4)... collect at VeloxSparkPlanExecApi.scala:633 +details | 2025/09/12 15:28:43 | 85 ms | 2/2 | 67.5 KiB | | | |
| 2 | SELECT promotions, total, cast(promotions AS DECIMAL(15, 4)) / cast(total AS DECIMAL(15, 4)... collect at VeloxSparkPlanExecApi.scala:633 +details | 2025/09/12 15:28:43 | 0.1 s | 6/6 | 1090.1 KiB | | | |
| 1 | SELECT promotions, total, cast(promotions AS DECIMAL(15, 4)) / cast(total AS DECIMAL(15, 4)... collect at VeloxSparkPlanExecApi.scala:633 +details | 2025/09/12 15:28:43 | 0.1 s | 6/6 | 4.4 MiB | | | |
| 0 | SELECT promotions, total, cast(promotions AS DECIMAL(15, 4)) / cast(total AS DECIMAL(15, 4)... collect at VeloxSparkPlanExecApi.scala:633 +details | 2025/09/12 15:28:43 | 2 s | 9/9 | 15.6 MiB | | | |

**Task Metrics:** Native execution shows different shuffle read/write characteristics, particularly when using ColumnarShuffleManager with columnar shuffle enabled.

Spark 3.5.5.3.3.6.2-1 — Jobs | Stages | Storage | Environment | Executors | Gluten SQL / DataFrame | SQL / DataFrame

## Gluten SQL / DataFrame

▾ Gluten Build Information

| Name | Value |
|---|---|
| Component Velox Branch | <unknown> |
| Component Velox Revision | <unknown> |
| Component Velox Revision Time | <unknown> |
| Components | Velox |
| GCC Version | <unknown> |
| Gluten Branch | branch-1.4 |
| Gluten Build Time | 2025-06-06T13:57:24Z |
| Gluten Repo URL | https://github.com/apache/incubator-gluten |
| Gluten Revision | 50dd117dadb9fe3e9bba5c5002168e1294b2e2bb |
| Gluten Revision Time | 2025-06-06 15:51:55 +0800 |
| Gluten Version | 1.4.0 |
| Hadoop Version | 3.3.4 |
| Java Version | <unknown> |
| Scala Version | 2.12.15 |

# 7. Future Roadmap and Development

## 7.1 Ongoing Enhancements

### Expanded Operation Support

- Additional SQL function coverage
- Streaming workload optimization
- Enhanced UDF integration

### Performance Optimizations

- GPU acceleration integration
- Advanced compression techniques
- Query compilation improvements

## 7.2 Ecosystem Integration

### Storage Format Evolution

- **Apache Iceberg optimization** : Well supported in recent Gluten versions. Vectorized reading of Iceberg tables works with most standard operations like filters, projections, and joins. Schema evolution and time travel queries are generally supported.

- **Delta Lake integration :** Good support for basic read operations. Vectorized execution works well with Delta's Parquet data files, though some Delta-specific features might fall back to JVM execution.

- **Hudi compatibility improvements :** Basic read support exists, particularly for Copy-on-Write tables. Merge-on-Read tables may have limitations since they require merging base and delta files, which can be complex for vectorized execution.

### Cloud Platform Support

- Optimized deployments for AWS EMR, Azure HDInsight
- Containerized deployment patterns
- Serverless execution environments

# 8. Conclusion

Gluten/Velox represents a significant advancement in Spark's execution capabilities, delivering substantial performance improvements for analytical workloads through vectorized processing and native code execution. Organizations processing large-scale analytical workloads can expect 1-3x performance improvements with proper implementation and tuning.

The technology is production-ready for OLAP workloads with strong community support and active development. While some limitations exist around UDF support and streaming operations, the benefits for analytical use cases significantly outweigh the constraints.

Key success factors for deployment include:

- Proper cluster configuration and native library management
- Workload-appropriate tuning of memory and execution parameters
- Comprehensive testing and validation processes
- Monitoring infrastructure for performance tracking

*As the project continues to mature, Gluten/Velox is positioned to become the standard high-performance execution engine for Spark analytical workloads, enabling organizations to extract maximum value from their big data investments.*

# References


- Apache Spark Official Documentation: https://spark.apache.org/docs/
- Gluten Project Repository: https://github.com/oap-project/gluten
- Velox Library Documentation: https://facebookincubator.github.io/velox/
- TPC-DS Benchmark Specification: http://www.tpc.org/tpcds/
- Increase Spark Performance with Intel CPUs and Gluten: https://community.intel.com/t5/Blogs/Tech-Innovation/Cloud/Increase-Spark-Performance-with-Intel-CPUs-and-Gluten/post/1637160
- Velox: Meta's Unified Execution Engine: https://www.vldb.org/pvldb/vol15/p3372-pedreira.pdf



Explore ODP ↗

Free Consultation ↗

acceldata

www.acceldata.io