Starburst

Spooling client protocol

Trino Community Broadcast

Mateusz Gajewski
@wendigo

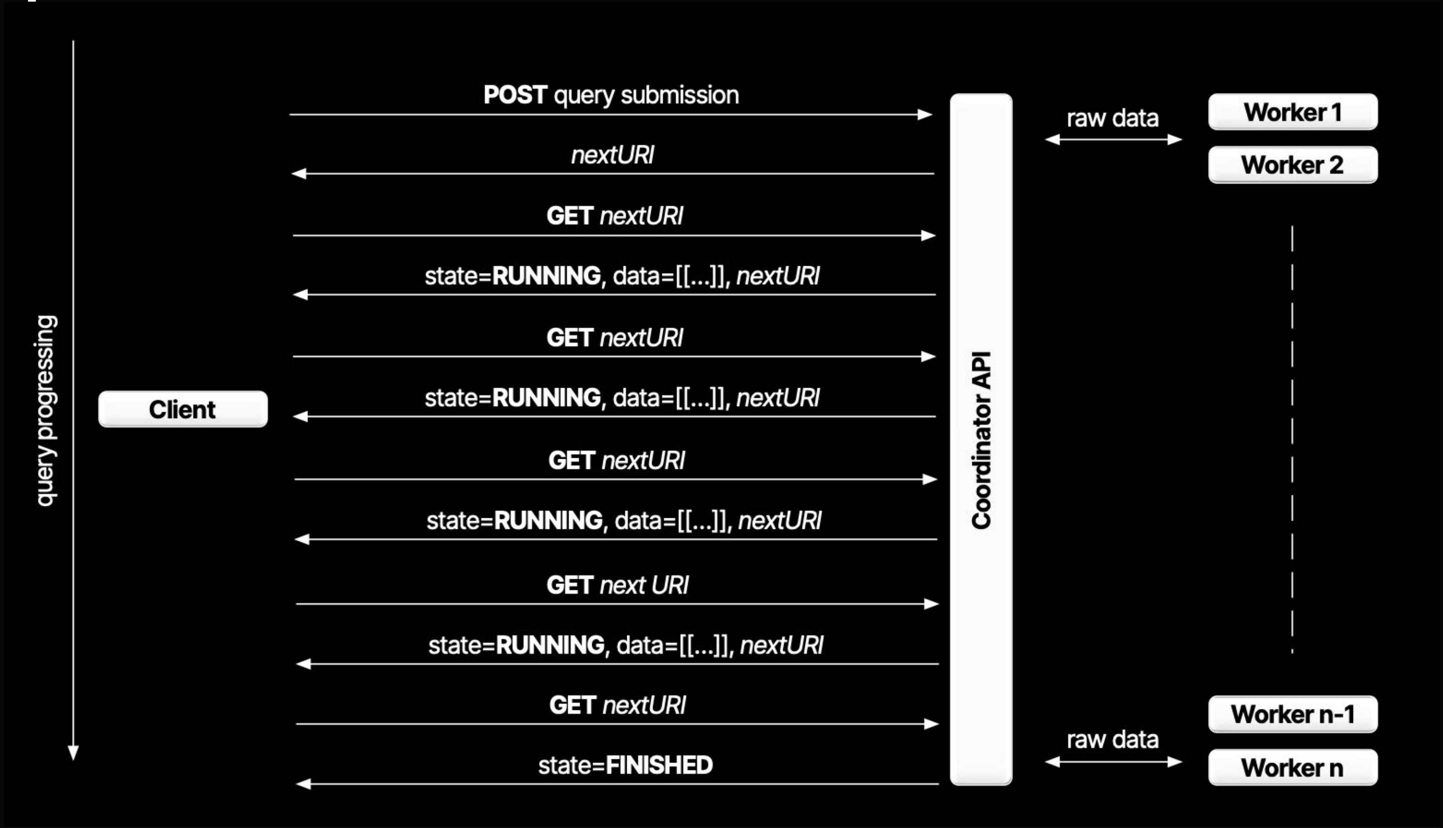
Agenda

- Direct protocol overview
- Spooling protocol design
- Direct vs spooling
- Server/client configuration
- Live demo 🎉
- Final thoughts

Direct protocol

- Stable for the last 10 years, since the early days of Presto,
- “Streaming” data retrieval semantics,
- Works out of the box:
    - Doesn’t need any configuration,
    - Single deployment architecture - coordinator-oriented,
- JSON format only (we will get to it),
- Low latency, but also mediocre throughput:
    - Works best for highly-selective/DML queries,
    - Not so good for getting large datasets out of the cluster,
- Non-extensible and impossible to change.

Direct protocol flow



query progressing

Client

POST query submission

nextURI

GET *nextURI*

state=**RUNNING**, data=[[...]], *nextURI*

GET *nextURI*

state=**RUNNING**, data=[[...]], *nextURI*

GET *nextURI*

state=**RUNNING**, data=[[...]], *nextURI*

GET *next URI*

state=**RUNNING**, data=[[...]], *nextURI*

GET *nextURI*

state=**FINISHED**

Coordinator API

raw data

Worker 1

Worker 2

raw data

Worker n-1

Worker n

Spooling protocol objectives

- Much higher throughput,
  - Traded for some latency,
- Multiple deployment architectures possible,
  - Configurable to support diverse range of use-cases,
- Reuse existing framing and protocol semantics,
- Easy to implement on the client side,
  - Backward and forward compatibility for existing clients,
- Extensible encoding formats,
  - Negotiated between the client and the server,
  - Adding Arrow support is “easy” now.

How we did that?

- Spooling protocol extension (https://github.com/trinodb/trino/issues/22662),
- One **extra header** and one **extra shape** for the *data* field,
  - **inline** (*byte[]*) and **spooled** (*URI*) "segments",
- Ships with **existing JSON encoding**,
  - Rolling out support in the existing clients,
- **Protocol and encoding negotiation** with **graceful fallback**,
  - Forward and backward compatibility for the existing clients and deployments,
- New ***SpoolingManager*** SPI,
  - Ships with native file system based manager with support for S3, ABFS, GCS,
- Allows **adding new encoding** schemes seamlessly.

On the wire format comparison

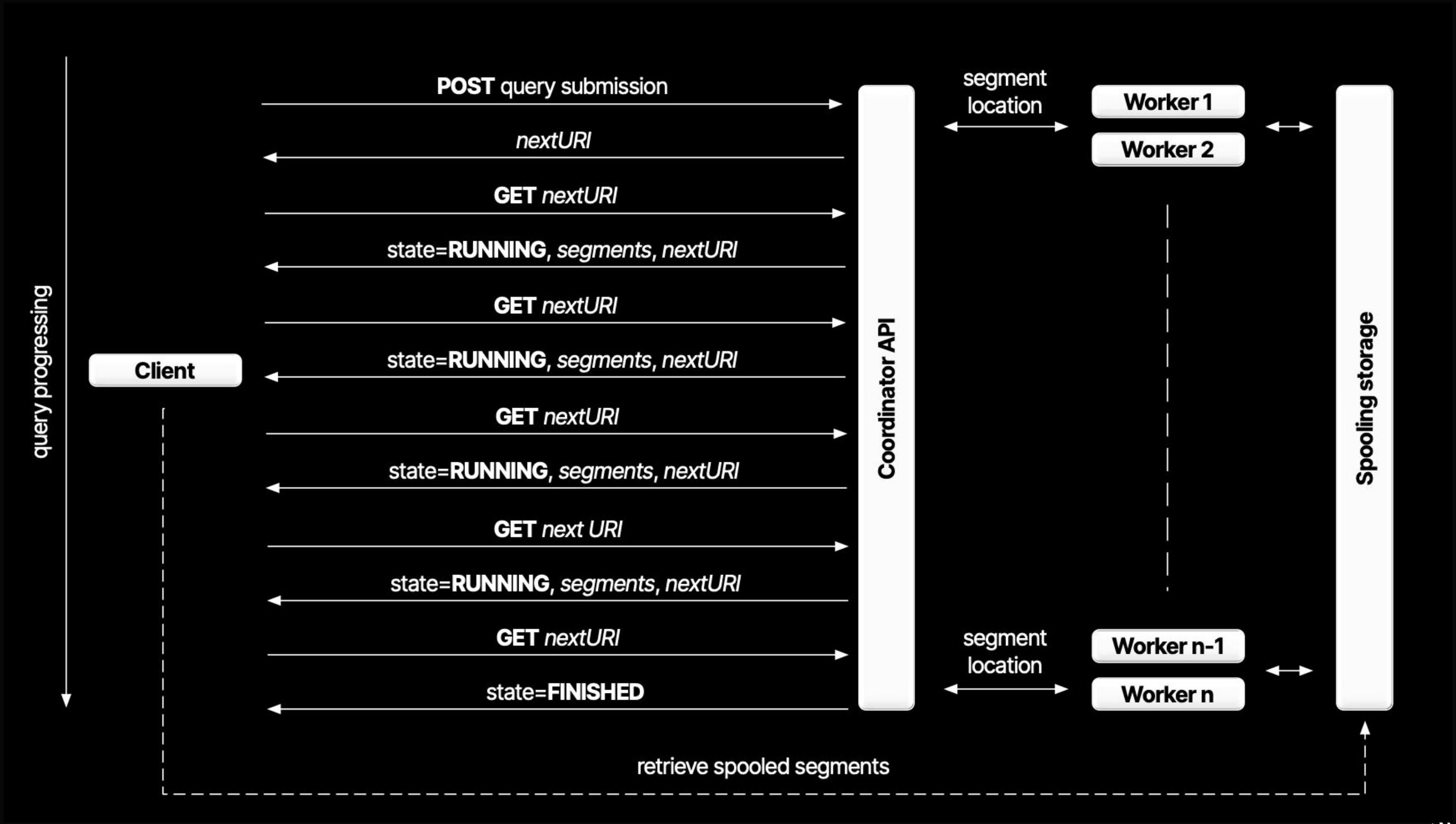
Direct protocol

```
“data”: [
        [row1:col1, row1:col2],
        [row2:col1, row2:col2],
        [row3:col1, row3:col2],
        …
]
```

Spooling protocol (X-Trino-Query-Data-Encoding)

```
"data": {
   "encoding": "json+zstd",
   "segments": [{
        "type": "inline",
        "data": "c3VwZXI=",
        "metadata": {...}
    }, {
        "type": "spooled",
        "uri": "http://location",
        "ackUri": "http://location"
        "headers": {...},
        "metadata": {...}
    }]
}
```

Spooling protocol flow



Starburst

Implications

- Data encoding (*CPU*) moved from coordinator to the workers,
- Data handoff (I/O) moved from the nodes to the spooling storage,
- Segments spooled to the storage as fast as possible with little or no buffering*,
- Segments retrieval can happen outside of the query lifetime,
- Client gets more “data” (multiple segments) in a single *nextURI* call,
- Larger data chunks (1 MB in the direct vs up to the 128 MB per segment),
  - Configurable compression (Zstd, LZ4),
  - Improved throughput with small initial latency penalty.

* Small pages are coalesced to bigger segments to avoid creating too many small objects on the spooling storage

Configuration

- protocol.spooling.enabled=true
- spooling-manager.properties file:
    - spooling-manager.name=filesystem
    - fs.s3.enabled=true
    - fs.location=s3://spooling/
    - fs.segment.ttl=12h
    - ...
- Support in the JDBC, ODBC*, CLI, Java and Python clients,
- SSE-C encryption enabled by default,
- Compressed JSON variants preferred

protocol.spooling.retrieval-mode

- **STORAGE** (1 hop)
    - Client goes directly to the spooling storage (presigned URI)
- **COORDINATOR_STORAGE_REDIRECT** (2 hops)
    - Client goes to the coordinator, gets redirected to the spooling storage (presigned URI)
- **COORDINATOR_PROXY** (1 hop)
    - Client retrieves the data through the coordinator (acts as an I/O proxy)
- **WORKER_PROXY** (2 hops)
    - Client goes to the coordinator, gets redirected and retrieves data through one of the workers (acts as an I/O proxy)

Demo time

- Spooling-enabled Starburst Galaxy dev/prod cluster (~120 ms latency across the pond),
- Retrieval mode: STORAGE
- Trino CLI v469 run with:

  -
    ```
    time trino --server https://wendigo-wendigo-spooling.trino.galaxy-dev.io \
    --user 'mateusz.gajewski@starburstdata.com/accountadmin' --password \
    --execute 'SELECT * FROM tpch.sf10.lineitem LIMIT 1_000_000' \
    --network-logging=BASIC \
    --output-format=null [--encoding='json+lz4']
    ```
- Direct protocol: **~35s**
- Spooled protocol: **~9s**

Spooling protocol summary

- Non-experimental since 466 (Nov, 2024),
- Coexists with the direct protocol (for forward, backward compatibility),
- “Streaming” segment location retrieval,
- Requires storage configuration to use:
  - Extensive configuration options, session properties and deployment architectures,
- Only JSON format supported (for now),
  - Compressed variants
- Server side encryption with ephemeral per-segment encryption keys,
- Higher latency than direct but also much higher throughput.

Direct vs spooling comparison

Direct protocol

Encoding: only JSON,

Optimized for: latency for small queries,

Data retrieval: streaming in inlined chunks,

Data chunk size: 1 MB,

Bottleneck: coordinator (CPU, I/O),

Requirements: none,

Client support: all Trino and Starburst clients.

Spooling protocol

Encoding: extensible, json, json+lz4, json+zstd,

Optimized for: large data set retrieval,

Data retrieval: streaming in spooled segments,

Data chunk size: 2-128MB pre compression,

Bottleneck: workers (CPU, I/O), storage (I/O),

Requirements: spooling storage configuration,

Client support: CLI, JDBC, ODBC*, Python.

Some extra things

- Lots of code refactors on the client and server side,
- Most of the optimizations ported to direct protocol:
  - New streaming JSON decoding/encoding,
  - Faster decimal/floats parsing (Jackson),
  - Direct memory to on-the-wire write for the Slice-backed types (char, varchar, varbinary, etc),
- Session properties to control inlining and segment sizes (469),
- TODO:
  - Experimental Arrow encoding in the works,
  - Support in the other client libraries (javascript, c#, golang, etc)

Starburst

Thank you!

Mateusz Gajewski

01/31/2025