Presto Training Series, Session 4: Cluster Sizing & Performance Tuning

Try Presto: www.prestosql.io

Dain Sundstrom and Manfred Moser
9 September 2020
Today’s Speakers

Manfred Moser
Developer, author, and trainer at Starburst

Manfred is an open source developer and advocate. He is an Apache Maven committer, co-author of the book Presto: The Definitive Guide, and a seasoned trainer and conference presenter. He has trained over 20,000 developers for companies such as Walmart Labs, Sonatype, and Telus.

Dain Sundstrom
Co-creator of Presto and CTO at Starburst

Dain is a co-creator of Presto, co-founder of the Presto Software Foundation, and CTO at Starburst. Prior to Starburst, Dain was a Software Engineer at Facebook, A Software Architect at Proofpoint, founded the Apache Geronimo project, and was one of the original JBoss authors.
Agenda

- Presto overview / review
- Cluster configuration
- Machine sizing
- Five minute break and Q&A
- Workload tuning
- Making queries faster
- Sharing resources
- Q&A
Questions

• Ask any time
• Use the meeting Questions feature
• Manfred screens, collects and interjects
• Dedicated Q&A in break and at end
Some advice for attendees

• This is a fast-paced overview – don't try to follow along during class
• Instead focus and pay attention
• Use the demo video after class to setup Presto and CLI locally
• Learn at your own pace
• Use video recording and slides from class as reference to learn more
• Apply skills for your own use case
Presto overview

… probably just a recap for you
What is Presto?

High performance ANSI SQL engine
- SQL support for any connected data source - SQL-on-anything
- Cost-based query optimizer
- Proven horizontal scalability

Open source project
- Very active, large community
- User driven development
- Huge variety of users
- Prestosql.io

Separation of compute and storage
- Scale query processing and data sources independently
- Query storage directly
- No ETL or data integration necessary

Presto everywhere
- No cloud vendor lock-in
- No storage engine vendor lock-in
- No Hadoop distro vendor lock-in
- No database lock in
Why use Presto?

- High performance query processing
- Low barrier of entry for users
- Massive scalability
- High concurrency
- Direct access to storage

- Reduced need to copy and move data
- Avoid complex data processing
- Scale storage and compute independently
- Only run computes when processing queries
- One data consumption layer

- No more data silos, departmental copies
- Query data with the existing skills and tools - SQL + BI tools
- Query any data source
- Move data
- Create optionality
Cluster sizing and performance tuning
Agenda

• Sizing
  • How big should my cluster be?
  • What machine size should I use?
• Break and Q&A
General strategy

• Create a big cluster
  • Bigger than you think you need

• Verify everything works and is stable
  • Don’t try to stabilize and tune at the same time

• Tune
  • Performance and efficiency are follow up tasks
Presto is unlike most systems

- Query can, and will, use all computers for a single query
  - Most expect one computation per HTTP request
- Presto will use all available resources
  - Memory, CPU, and network allocated
- Presto uses one process per machine
- Presto uses multiple threads for a single query per machine
  - Query data structures are shared across threads for efficiency
Baseline advice

- Disable OS spilling (JVM is not designed for spilling)
  - Spilling = OS swap memory to disk
  - Java uses a compacting GC, so there aren’t cold sections to spill
- Disable “runaway” process detector
  - Presto uses all available resources
  - Presto should be the only active process on the machine
- Upgrade regularly
  - Improvements all the time - active healthy community
  - Security fixes
Cluster sizing
CPU and memory

CPU: Process data for the query
- More CPUs == shorter queries (generally)

Memory: data structures required to run the query
- Memory limit concurrency
- Either you have enough or you don’t

If you have 2x the CPU, you can run 2x the workload with same memory
... just not at the same time.
CPU

- Presto queries requires a certain amount of CPU
  - Run query a few times to see the stable cost
- Generally:
  - Double the CPU and the query takes half the time
  - Run two concurrent copies, and query takes twice as long
Memory

Memory is needed for JOIN, GROUP BY, ORDER BY, and window functions

- Hash tables and sorting

Peak memory is generally stable

- JOIN: build hash table from one table
- GROUP BY: hash table of group by keys and aggregate values
- ORDER BY: sort the results
- Window: partitioned sorted window frames

Query plan matters

- Join order: smaller table in memory
- Phases: How many joins are in memory
Cluster sizing

- What is the workload?
  - It all *depends*!
- What queries are run?
  - Operations: JOIN, GROUP BY, ORDER BY, window
  - Table sizes
  - Filter selectivity
- Expected latency? (more CPU)
- Expected consistency? (readily available CPU)
- What is the concurrency?
  - Do big memory queries run at the same time?
  - What is the peak?
What if I don’t know?

- Engage a vendor
  - Vendors have expertise in sizing
- Build a big cluster (bigger than you think you need)
  - Limit query memory and runtime
  - Onboard one team at a time
  - Seek out diversity in teams
  - Measure and resize
So we are done!

... with the first steps.
Machine sizing
Memory

- OS has limits
  - Must leave spaces for others
- Other processes
- JVM is not just heap size (Xmx)
- You MUST have free space

<table>
<thead>
<tr>
<th>OS</th>
<th>Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Disk and network buffers</td>
</tr>
<tr>
<td></td>
<td>Other process</td>
</tr>
<tr>
<td></td>
<td>Other process</td>
</tr>
<tr>
<td>Presto JVM</td>
<td>Thread stacks</td>
</tr>
<tr>
<td></td>
<td>Code cache</td>
</tr>
<tr>
<td></td>
<td>GC data structures</td>
</tr>
<tr>
<td></td>
<td>Native buffers (off heap)</td>
</tr>
<tr>
<td>Heap (Xmx)</td>
<td>Untracked (shared buffers)</td>
</tr>
<tr>
<td></td>
<td>General</td>
</tr>
<tr>
<td></td>
<td>Query 1 User System</td>
</tr>
<tr>
<td></td>
<td>Query 2 User System</td>
</tr>
</tbody>
</table>
Memory allocations

- How much memory does the OS really have available?
  - Disk/network buffers are used when needed
- Allocate 80% of machine to Java heap (-Xmx)
  - 20% is for OS overhead, JVM overhead, and shared Presto stuff
- Presto max query memory 70% of heap
  - Only memory accountable to a single query (major structures)
  - Does not include temporary or shared data structures

128 GB machine = 100 GB JVM heap = 70 GB query memory
Shared JOIN hash

- JOIN performs lookup into hash table for each row
- Hash table is shared for all JOIN threads
- More CPUs can finish query faster… freeing up the memory!
Distributed JOIN

\[
\{ \text{hash(key)} \% n \}
\]
Skew

Each machine has $1/m$ keys

\[
\begin{align*}
\text{SELECT} & \text{ * FROM views JOIN user USING (user_id)} \\
\text{SELECT} & \text{ count(*) FROM views GROUP BY user_id}
\end{align*}
\]

Some keys will have more rows than others

*Active users see more pages*

What happens when one key has half the rows?

*User id 0 is the anonymous user*
Skew impact

JOIN build
  Uneven memory distribution across machines
  Likely more output data (depends on matches)

JOIN probe
  Some machine will process way more rows
  Likely more output data (depends on matches)

GROUP BY
  More effective partial aggregations on workers*
  Likely less data to distribute
  Possibly more partials for a machine to process
Use bigger machines

Fewer bigger machines is better than more smaller machines

- Less memory overhead per machine
- Mitigate problems with scheduler
  - Scheduler makes decisions with very little information
- Mitigate the problems from skew
  - More cores to for parallel JOIN
  - More memory for hash table
- Less overhead for broadcast joins
- Less coordination work
What machine type should I use

Start with bigger balanced machines:

<table>
<thead>
<tr>
<th>Type</th>
<th>vCPU</th>
<th>Cores</th>
<th>Memory GiB</th>
<th>Network Gbps</th>
</tr>
</thead>
<tbody>
<tr>
<td>m5.16xl</td>
<td>64</td>
<td>32</td>
<td>256</td>
<td>20</td>
</tr>
<tr>
<td>r5.8xl</td>
<td>32</td>
<td>16</td>
<td>256</td>
<td>10</td>
</tr>
<tr>
<td>m5.8xl</td>
<td>32</td>
<td>16</td>
<td>128</td>
<td>10</td>
</tr>
<tr>
<td>r5.4xl</td>
<td>16</td>
<td>8</td>
<td>128</td>
<td>5</td>
</tr>
<tr>
<td>c5.24xl</td>
<td>96</td>
<td>48</td>
<td>192</td>
<td>25</td>
</tr>
</tbody>
</table>

CPU kind

AMD: ~10% less $

Graviton: 64 cores, 256 mem, 25 net for ~20% less $ than m5.16xl
Additional thoughts
Hash join vs. (sort) merge join

- Hash join loads one table into memory as a hash table and the does a lookup for each row
  - One table in distributed memory
  - Memory cache unfriendly
- Merge join takes two sorted tables worked on join key, advance “lower” table until a match is found
  - Stream buffer for each table
  - Memory cache friendly

- What if tables are not sorted?
  - Sort them… which is very expensive

- What about parallelism?
What about spilling?

Spilling can reduce memory… at the cost of latency.

JOIN

- Part of hash table and matching probe rows are written to disk
- Later each part is loaded and probe continues

Aggregation

- Dump partial aggregation results (sorted)
- Merge back together
Spilling: Don’t do it

- Disks are slow and SSD is expensive
- CPU cost for read/write occurs (and maybe compress and encrypt)
- Completely changes workload profile when triggers
- Queries become much slower
  - May become IO bound (idle CPUs)
  - May cause workload queueing
  - Users find the system “unreliable” (is it down?)
- Rather than buying disks, buy some more machines
- Change workload to not need spilling
- Scale up cluster before big queries
Small clusters

- A small cluster that needs most resources for a single query
- Workload can easily exceed cluster limits
  - Can be any resource (CPU, memory, network)
  - Bigger clusters get a mix of queries to even out usage
  - Skew kills here
- Can feel unreliable to users
  - Single big query can cause queuing (is it down?)
Sizing summary
Summary

- Don’t treat Presto like Hive, Spark, micro-services, ...
- Upgrade regularly - each release has performance improvements
- Determine workload
  - Consider concurrency and workload mix
  - CPU requirements
  - Peak memory required
- Divide required resources into machines
  - Favor fewer bigger machines
- Never spill (not in the OS, not in Presto)
Plan for growth!

- Plan for growth because Presto is awesome!
  - Presto is easy to use, so people use it more
  - Presto is fast, so people run more queries
  - Presto can access all data, so people will use it for new things
- It is not unusual to see demand double (or triple) every year
- Put this in the budget (don’t surprise your boss)
5 minute break

And if you stick around:

- Browse prestosql.io
- Join us on Slack
- Submit questions
Tuning the workload

Doing more with less
Agenda

• Workload tuning
  • How do I reduce work?
  • How do I make queries faster?
  • How do I share resource?

• Q&A
Query plan

- Watch **Martin’s training**
  - Join order is critical
  - Join type is important
  - Stats are required: ANALYZE

- Watch **David’s advanced SQL training**
  - GROUPING SETS, CUBE, ROLLUP
  - Top N row_number (top by group)
  - Aggregate with filter
    - count(name) FILTER (WHERE name < 10)
  - Approximate functions
    - approx_distinct
    - approx_most_frequent
    - approx_percentile

- Train your users!
Precompute

• For a regular or expected queries
  • Commonly dashboards or daily reports

• Easy:
  • Store full precomputed results
  • Copy source tables into a faster store or format (ZSTD ORC)

• Medium:
  • Store expensive subquery

• Hard:
  • Store partial aggregations (for example, daily counts by user)
  • Requires application changes

• Very, very hard:
  • Sample data - any non-trivial query requires deep statistics knowledge
Connectors

- Hive connector is normally biggest CPU user
  - Reading data is parallel distributed across all machines
  - Typically data is highly compressed on network
  - Typically CPU bound, but maybe network bound
- JDBC based connectors
  - MySQL, PostgreSQL, SQL Server, Oracle, Redshift, etc.
  - Single reader per table
  - Starburst has parallel versions of many connectors
  - Typically not compressed on network
  - Expect ~5 MBps per connection
- For others, check the docs
Hive data organization
Organize the data for the Hive connector

- Presto can take advantage of physical data organization
- Partition
  - Each value written to a different directory (date is common)
- Bucket (a.k.a, hash partitioning)
  - Data is hashed on some columns and divided into N buckets
  - hash(column) % n
  - Can be sorted on a column (sorted bucket)
- A table can only be organized one way, but both partitioning and bucketing can be used
Hive partitioning

- One directory per value (e.g., each day in a different directory)
- Filters can be applied directly
  - `WHERE ds >= DATE '2020-09-08`
  - `WHERE month(ds) = 9`
- All values are known
  - Powerful in inference and predicate move around
  - For JOIN, possible matches can be known
- Reduce scan size significantly
- Mostly used for data management (possibly already taken)
- Only apply this to lower cardinality values
Hive bucketing

- Data is hashed on some columns and divided into N buckets
- Reduces reads for equality filters
  - WHERE x = 42
  - WHERE x in (42, 55, 99)
- One file per bucket per INSERT
- Can reduce scan size significantly
- Planner can take advantage of bucketing for planning
  - Session property `bucket_execution_enabled`
  - Node local GROUP BY or JOIN which removes redistribution costs
- Sorting can help reduce scans for some file formats (more later)
- Only apply this to high cardinality columns
ORC and Parquet

- Use ORC or Parquet… ORC is faster in Presto

- Always compress
  - High compression: ZSTD over ZIP
  - Low compression: LZ4 over Snappy
  - Don’t use LZO

- Both are read optimized
  - Writes are expensive in CPU and memory
  - Columns can be read independently
  - Lots of columns (or nested columns) result in small IOs

- Both have min and max per column (and nested columns)
  - Stats are used to skip sections that won’t match filters
  - Sorting helps narrow the range
File Size

- File size in Hive is a big deal
- Any file less than 8 MB is considered small
- Small files result in small IOs
  - Object store: increase latency and can get throttled
  - HDFS: disks can run out of IO capacity
- Lots of small files make file listing slow
- Each file becomes a job, which increases scheduling time and cost
- For ORC and Parquet, whole file is loaded into memory
  - No lazy loading, but lazy decode still works
Bad Parquet Files

Be wary of Parquet files written non-Hadoop systems

• A common bug (feature?) is tiny row groups
• Typical size is 4k for a bad row group, when default is 128 MB
• Writing good Parquet files is CPU and memory intensive
• We have seen this from Snowflake and Greenplum
• Use Parquet dump tools to check sizes
• Tiny row groups compress poorly
• Tiny row groups cause bad IO patterns
• Increase network usage, IO latency, and costs
Rewrite table with Presto ORC writer

- Presto ORC writer has optimizations not in Hive ORC
  - Focuses on writing files that are easy to read
- Automatically collects stats
  - This is true for any insert operation in Presto
- Can switch to more efficient ZSTD compression algorithm
- Can change partitioning and bucketing
- Writes well sized files
  - Consider using `scale-writers` option
Making queries faster
Faster queries

Advice from earlier:

• Tune workload: query plan, approx functions
• Precompute if possible
• Use parallel connectors
• Organize data in Hive
• Use ZSTD ORC (or at least Parquet)
• Watch out for small files
Faster queries

Obvious, but worth asking:

- Are you out of CPU at peak times?
- Is the network saturated at peak times?
- Is your storage IO (HDFS, etc) saturated at peak times?
- Is your HMS or the database out of CPU at peak times?
What to look for in a query

Read the query

• If query is huge, you will need to narrow it down
• Does it have tons of distincts? Use approx distinct
• Does it have tons of unions?
  • Did they use UNION ALL? If not use, UNION ALL
  • Is this just an unrolled grouping set? Use grouping sets
• Are there a million regular expressions or other expensive functions?
• Is there a lot of JSON processing?
What to look for in a query

Run EXPLAIN?

- Do you get what you want?
- Do you have tons of mark distinct? Use approx distinct and union all
- Do the JOINs look right (order and type)? Check for stats

And don’t forget to revisit the training material from Martin!
What to look for

• Run EXPLAIN ANALYZE on your query
  • Find the most expensive stage (largest CPU time)
    • Find the most expensive operation

• If table scan (expected):
  • Is it reading too much data?
  • Did pushdown work? Did they forget partition filters?

• If JOIN:
  • Is it an expanding join? Did you want an expanding join?
  • Is the filter really expensive?

• Everything else
  • Check out row count and look for expanding joins
  • Look for very expensive functions
  • Look for raw data parsing (regex, json, etc)
More hardware

- Can I throw more hardware at it?
  - Easiest way is to just try it!
- Most effective at speeding up scans
  - Make sure the connector is parallel
  - Do you have more splits than cores?
Underutilization

- There is plenty of work to do but CPU is idle
- Common causes:
  - Hive metastore is slow
    - Check metastore and database load
    - Check Presto JMX stats for metastore operations (list partitions)
  - File listing is slow
    - Check for tiny files in tables
    - Check HDFS name node is load
    - Check for S3 throttling
  - Reads are slow
    - Check for tiny files in tables
    - Check for network overload
    - Check Presto JMX stats for read performance
  - Skew
Hive caching

• NOT a silver bullet
• Cache Hive metastore data:
  • See hive.metastore-cache*
  • Possibly miss new table and partitions
• Cache file listings:
  • See hive.file-status-cache*
  • Possibly miss new files
• File data (new):
  • Cached on Presto nodes local disks
  • Uses network between Presto nodes
  • Uses some CPU and memory on Presto nodes
  • Doesn’t help much with S3
Sharing resource
Resource groups

- Define resource and concurrency limits (a.k.a., queues)
- Focus on maximizing user happiness
  - Psychology not computer science
- User experience should match expectations
  - Small, fast, trivial queries should run immediately
- People hate waiting in line
  - Allow everyone to run one query
  - Add more hardware at peak times if necessary
- Lean on social dynamics
  - Divide users into groups of people they know
  - Let everyone know those in their group using the resources
  - Let users kill any query in their group
Wrapping up
Presto Training Series

Review our past sessions:

• Advanced SQL in Presto with David - recording available
• Presto Query Processing with Martin - recording available
• Securing Presto with Dain - recording available
Presto Summit series

Diverse information about Presto and real world usage

• State of Presto - recording available
• Presto as Query Layer at Zuora - recording available
• Presto Migration at Arm Treasure Data - recording available
• Presto for Analytics at Pinterest - recording available
And finally …

• Learn more from our website and documentation at prestosql.io

• Join us on slack at prestosql.io/slack

• Get a free digital copy of Presto: The Definitive Guide

• Thank you for hanging out with us

• See you next time
Your question
Our answers ...