Deep Dive on Amazon Redshift

Eric Ferreira, Principal Database Engineer

August 11, 2016

Agenda

- Service overview
- Migration considerations
- Optimization:
  - Schema/table design
  - Data ingestion
  - Maintenance and database tuning
- What’s new?
- Q & A (~15 minutes)

Service overview

Amazon Redshift

a lot faster
a lot simpler
a lot cheaper

Relational data warehouse

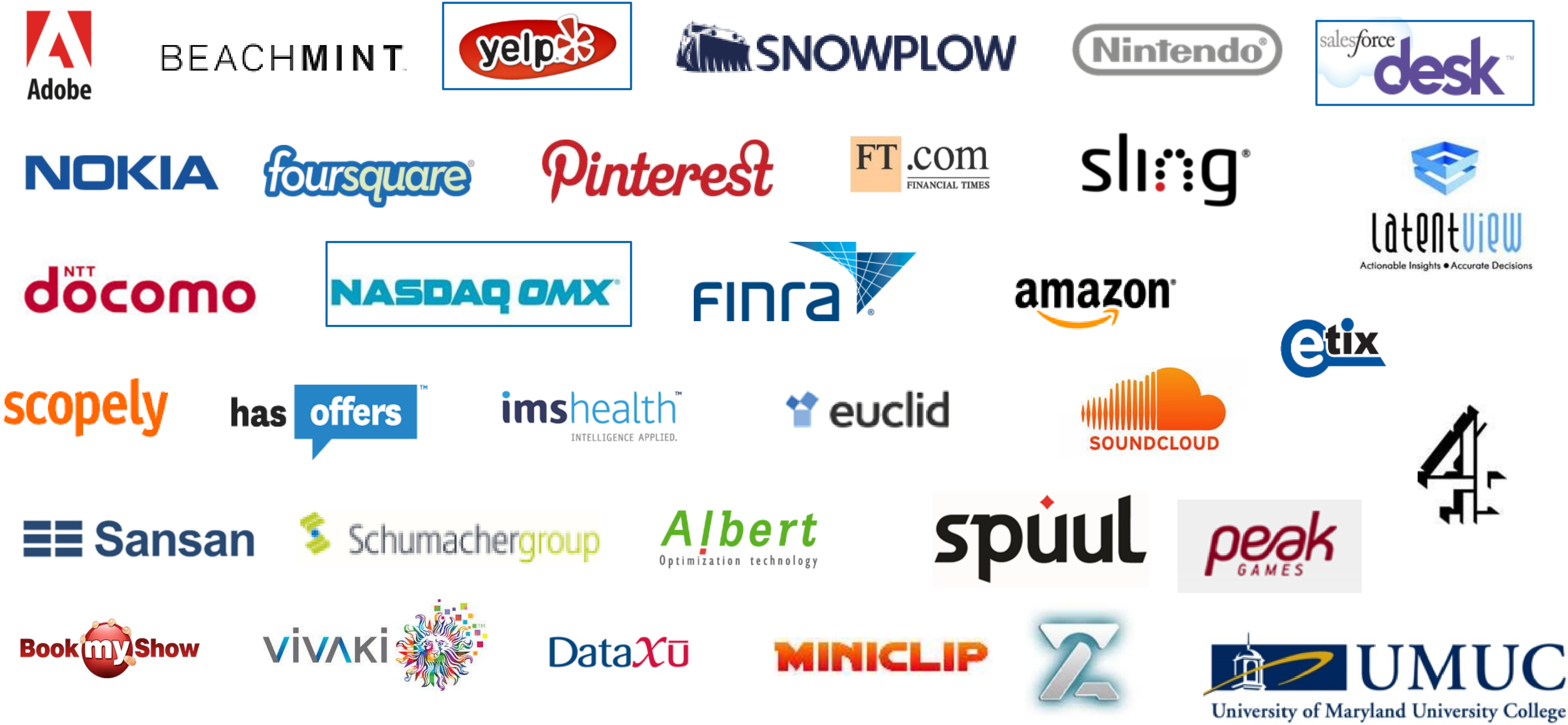
Massively parallel; petabyte scale

Fully managed

HDD and SSD platforms

$1,000/TB/year; starts at $0.25/hour

Selected Amazon Redshift customers



Amazon Redshift system architecture

Leader node

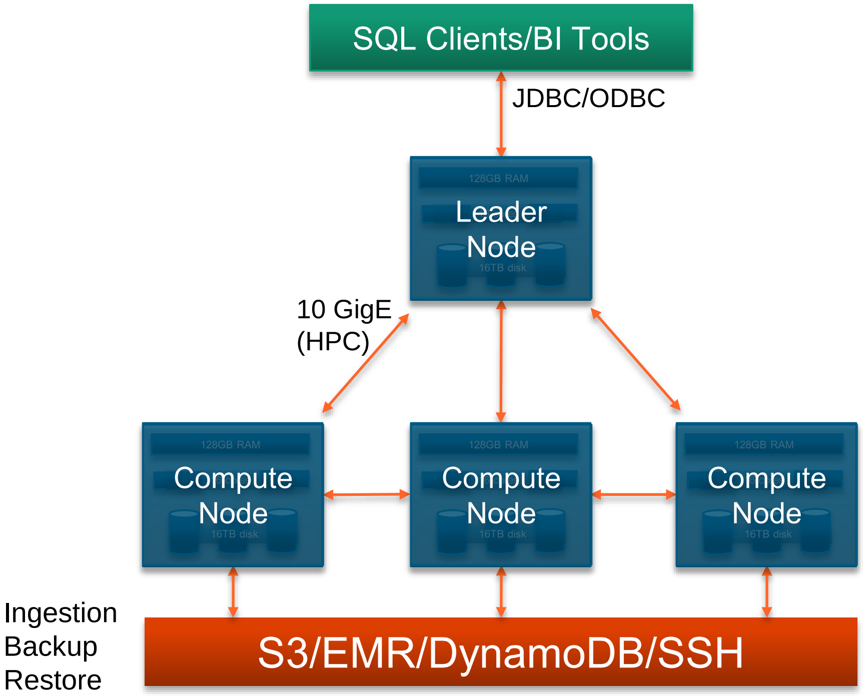
- SQL endpoint
- Stores metadata
- Coordinates query execution

Compute nodes

- Local, columnar storage
- Executes queries in parallel
- Load, backup, restore via Amazon S3; load from Amazon DynamoDB, Amazon EMR, or SSH

Two hardware platforms

- Optimized for data processing
- DS2: HDD; scale from 2 TB to 2 PB
- DC1: SSD; scale from 160 GB to 326 TB



SQL Clients/BI Tools

JDBC/ODBC

128GB RAM
Leader Node
16TB disk

10 GigE (HPC)

128GB RAM
Compute Node
16TB disk

128GB RAM
Compute Node
16TB disk

128GB RAM
Compute Node
16TB disk

Ingestion
Backup
Restore

S3/EMR/DynamoDB/SSH

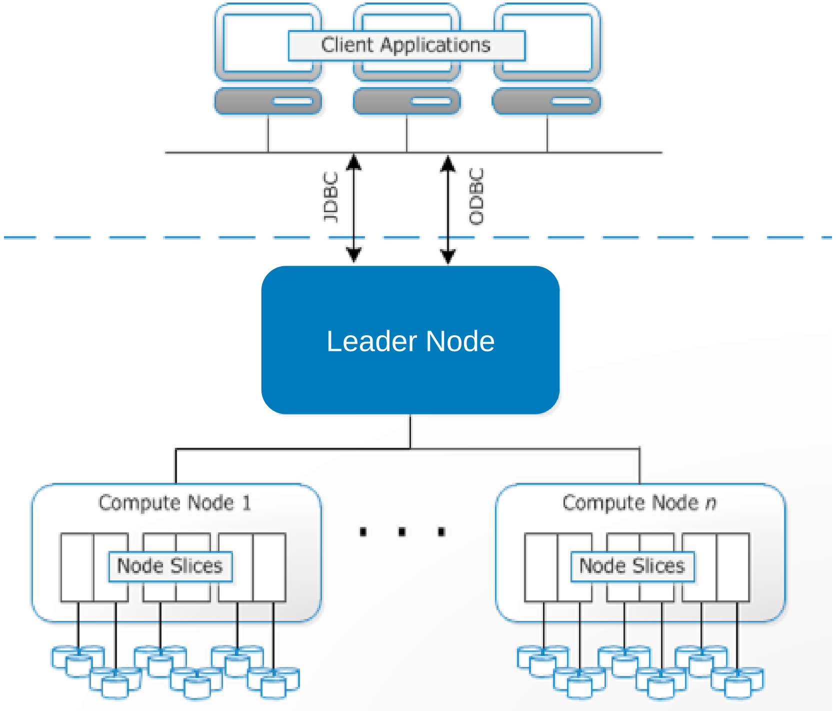
A deeper look at compute node architecture

Each node contains multiple slices

- DS2 – 2 slices on XL, 16 on 8 XL
- DC1 – 2 slices on L, 32 on 8 XL

Each slice is allocated CPU and table data

Each slice processes a piece of the workload in parallel



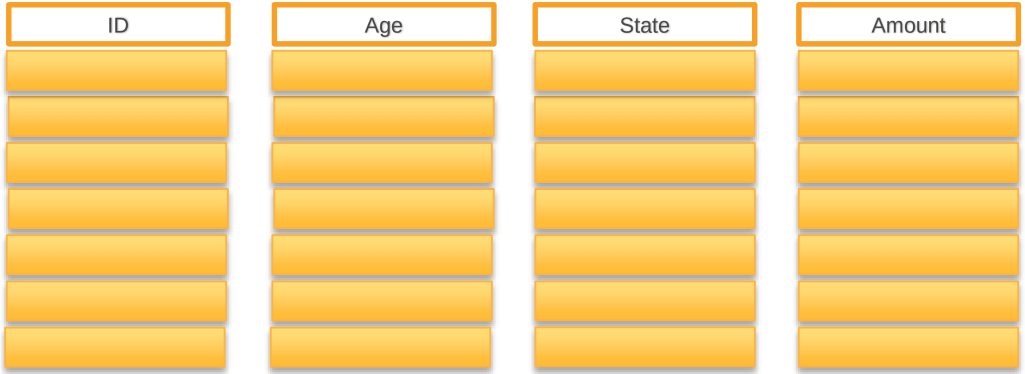
Amazon Redshift dramatically reduces I/O

Column storage

Data compression

Zone maps

ID	Age	State	Amount
123	20	CA	500
345	25	WA	250
678	40	FL	125
957	37	WA	375



- Calculating SUM(Amount) with row storage:
  - Need to read everything
  - Unnecessary I/O

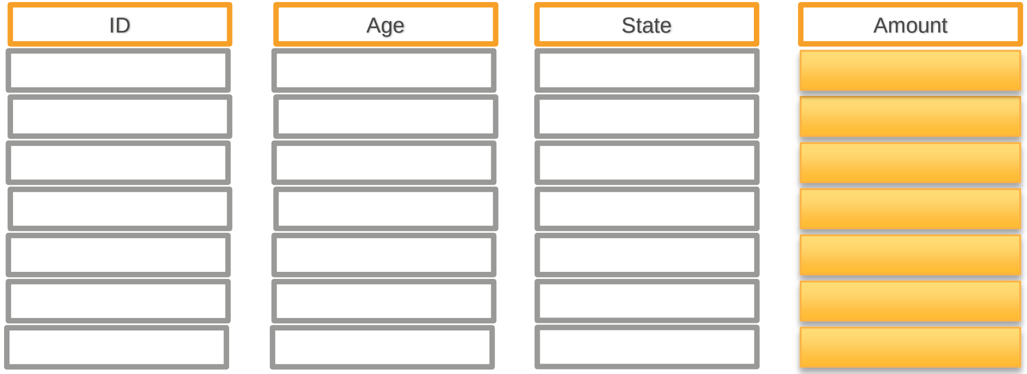
Amazon Redshift dramatically reduces I/O

Column storage

Data compression

Zone maps

ID	Age	State	Amount
123	20	CA	500
345	25	WA	250
678	40	FL	125
957	37	WA	375



- Calculating SUM(Amount) with column storage:
  - Only scan the necessary blocks

Amazon Redshift dramatically reduces I/O

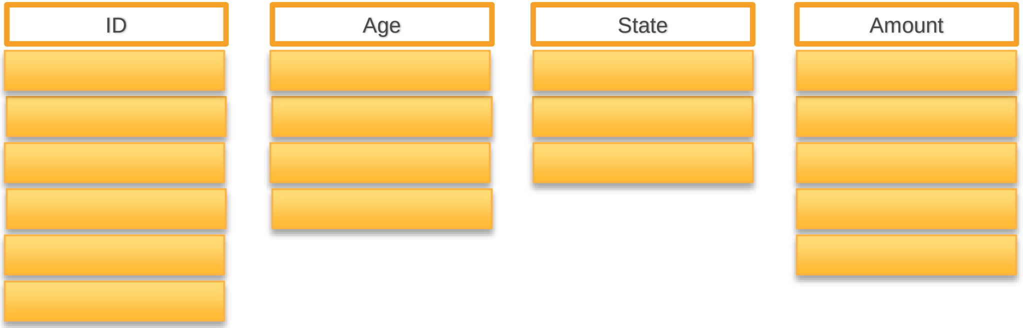
Column storage

Data compression

Zone maps

```
analyze compression orders;

  Table |    Column     | Encoding
--------+--------------+----------
 orders | id            | mostly32
 orders | age           | mostly32
 orders | state         | lzo
 orders | amount        | mostly32
```



- Columnar compression
  - Effective due to like data
  - Reduces storage requirements
  - Reduces I/O

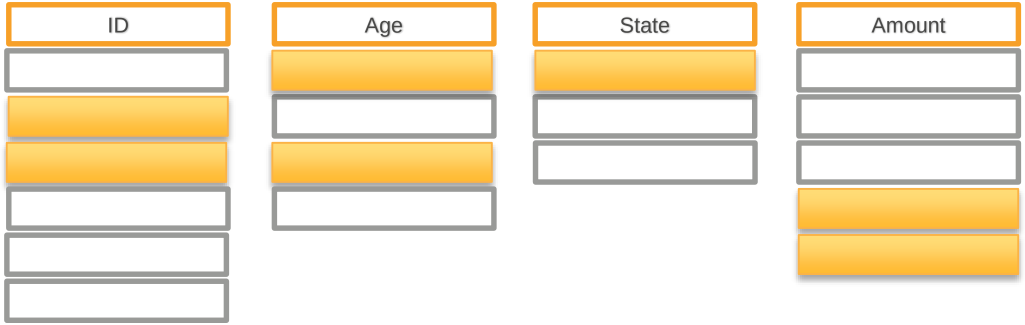
Amazon Redshift dramatically reduces I/O

Column storage

Data compression

Zone maps

- In-memory block metadata
- Contains per-block MIN and MAX value
- Effectively prunes blocks that don’t contain data for a given query
- Minimizes unnecessary I/O



Migration considerations

Forklift = BAD

On Amazon Redshift

- Optimal throughput at 8-12 concurrency, 50 possible
- Up-front table design is critical, not able to add indexes
  - DISTKEY and SORTKEY significantly influence performance
  - PRIMARY KEY, FOREIGN KEY, UNIQUE constraints will help
    Caution: These are not enforced
  - Compression is also for speed

On Amazon Redshift

- Optimized for large writes:
  - Blocks are immutable
  - Small write (~1-10 rows) has cost similar to a larger write (~100 K rows)
- Partition temporal data with time series tables and UNION ALL views

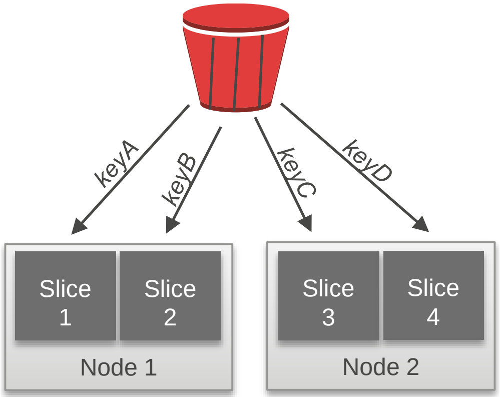
Optimization: schema design

Goals of distribution

- Distribute data evenly for parallel processing
- Minimize data movement
  - Co-located joins
  - Localized aggregations

Distribution key

Same key to same location



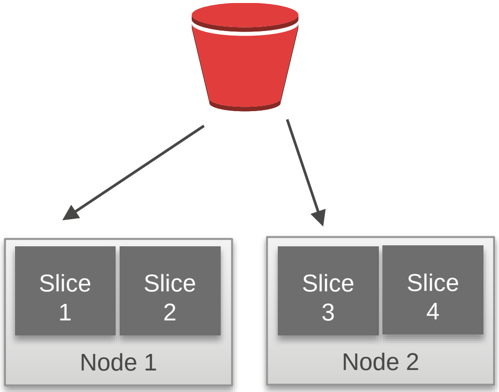
keyA
keyB
keyC
keyD

Slice 1 | Slice 2
Node 1

Slice 3 | Slice 4
Node 2

All

Full table data on first slice of every node

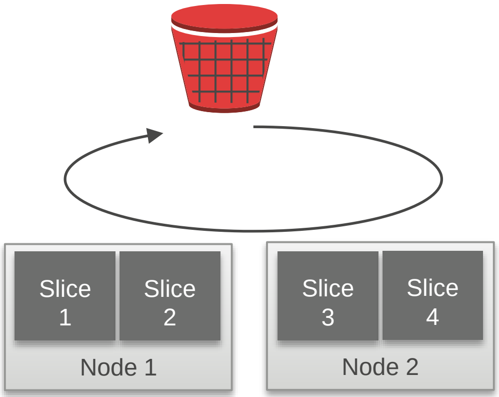


Slice 1 | Slice 2
Node 1

Slice 3 | Slice 4
Node 2

Even

Round-robin distribution



Slice 1 | Slice 2
Node 1

Slice 3 | Slice 4
Node 2

Choosing a distribution style

Key

- Large FACT tables
- Rapidly changing tables used in joins
- Localize columns used within aggregations

Even

- Tables not frequently joined or aggregated
- Large tables without acceptable candidate keys

All

- Have slowly changing data
- Reasonable size (i.e., few millions but not 100s of millions of rows)
- No common distribution key for frequent joins
- Typical use case: joined dimension table without a common distribution key

Goals of sorting

- Physically sort data within blocks and throughout a table
- Enable rrscans to prune blocks by leveraging zone maps
- Optimal SORTKEY is dependent on:
    - Query patterns
    - Data profile
    - Business requirements

Choosing a SORTKEY

- Primarily as a query predicate (date, identifier, …)
  - Optionally, choose a column frequently used for aggregates
  - Optionally, choose same as distribution key column for most efficient joins (merge join)

COMPOUND

- Most common
- Well-defined filter criteria
- Time-series data

INTERLEAVED

- Edge cases
- Large tables (>billion rows)
- No common filter criteria
- Non time-series data

Compressing data

- COPY automatically analyzes and compresses data when loading into empty tables
- ANALYZE COMPRESSION checks existing tables and proposes optimal compression algorithms for each column
- Changing column encoding requires a table rebuild

Automatic compression is a good thing (mostly)

“*The definition of insanity is doing the same thing over and over again, but expecting different results*”.

- Albert Einstein

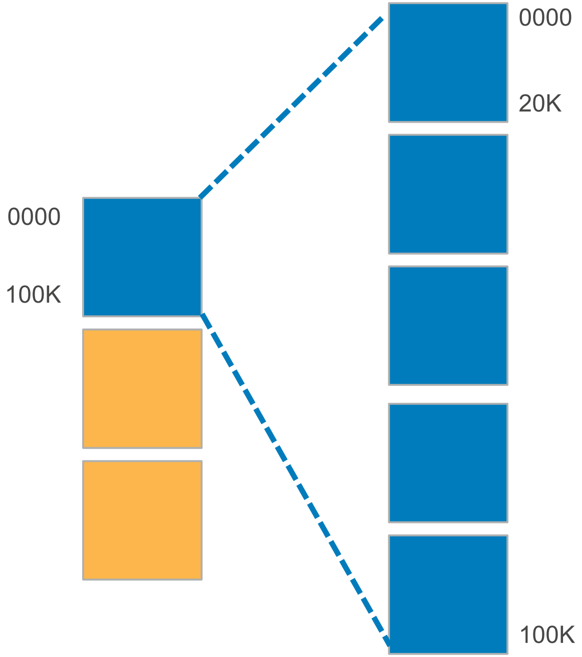
If you have a regular ETL process and you use temp tables or staging tables, turn off automatic compression:

- Use ANALYZE COMPRESSION to determine the right encodings
- Bake those encodings into your DDL
- Use CREATE TABLE … LIKE

Automatic compression is a good thing (mostly)

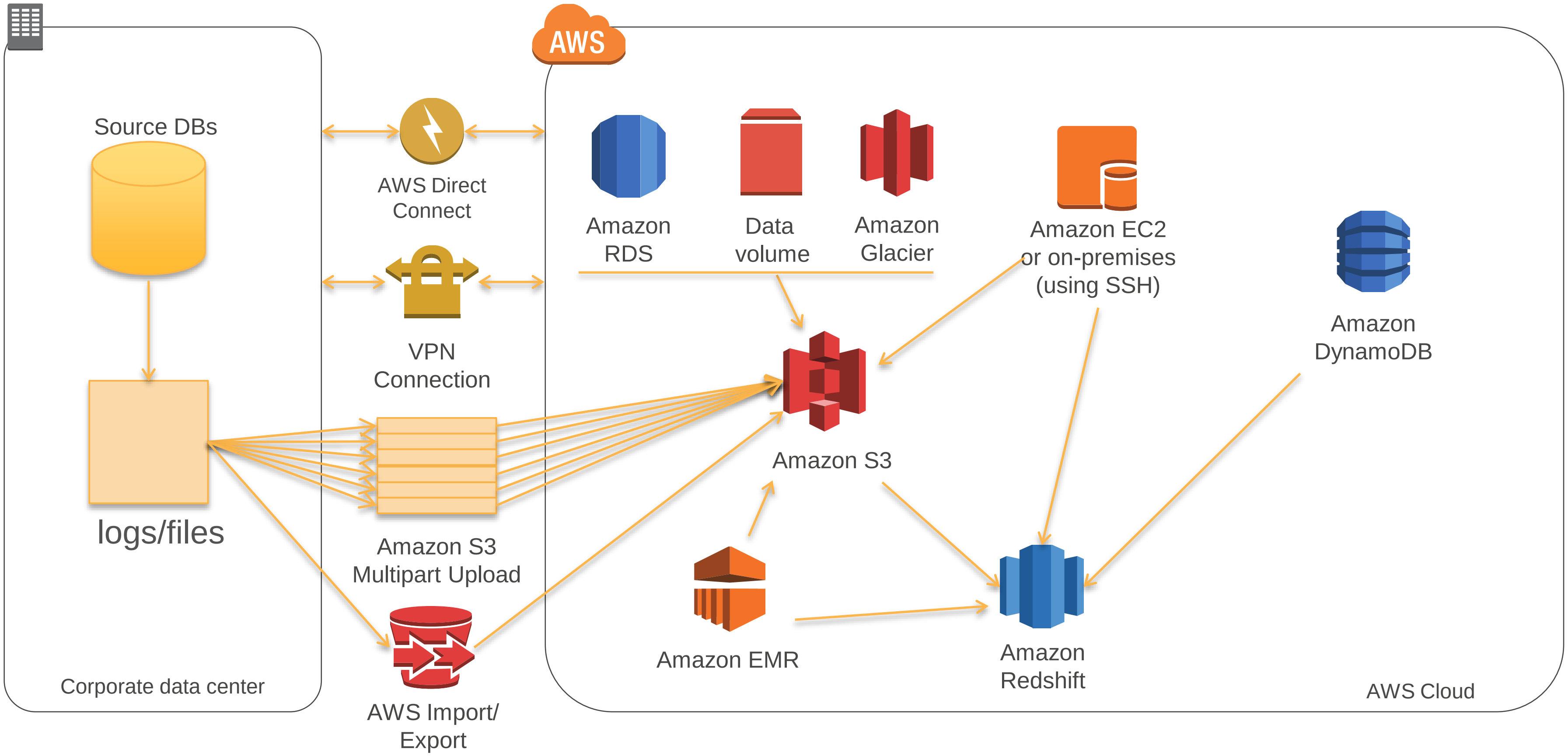
- From the zone maps we know:
    - Which blocks contain the range
    - Which row offsets to scan
- Highly compressed SORTKEYs:
    - Many rows per block
    - Large row offset

Skip compression on just the leading column of the compound SORTKEY



Optimization: ingest performance

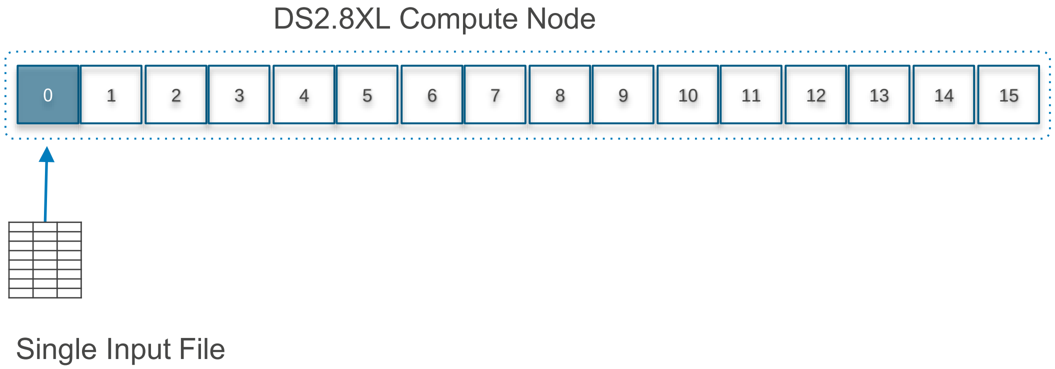
Amazon Redshift loading data overview



Parallelism is a function of load files

Each slice's query processors can load one file at a time:

- Streaming decompression
- Parse
- Distribute
- Write



A single input file means
only one slice is ingesting data

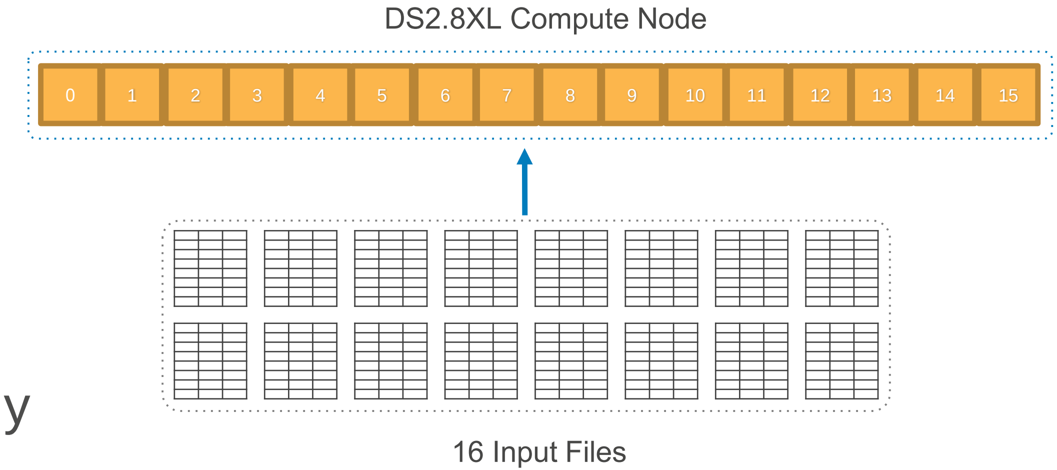
Realizing only partial cluster usage as 6.25% of slices are active

Maximize throughput with multiple files

Use at least as many input files as there are slices in the cluster

With 16 input files, all slices are working so you maximize throughput

COPY continues to scale linearly as you add nodes



Optimization: performance tuning

Optimizing a database for querying

- Periodically check your table status
- Vacuum and analyze regularly
  - SVV_TABLE_INFO
  - Missing statistics
  - Table skew
  - Uncompressed columns
  - Unsorted data
- Check your cluster status
  - WLM queuing
  - Commit queuing
  - Database locks

Missing statistics

- Amazon Redshift query optimizer relies on up-to-date statistics
- Statistics are necessary only for data that you are accessing
- Updated stats important on:
  - SORTKEY
  - DISTKEY
  - Columns in query predicates

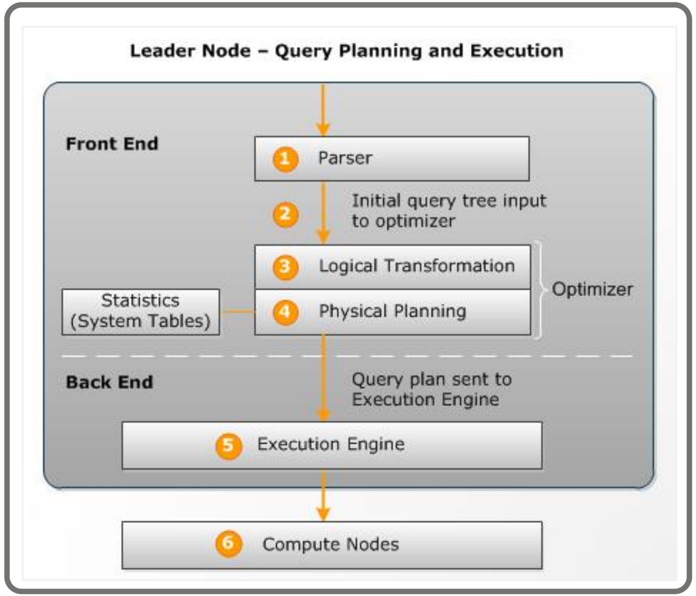

Leader Node – Query Planning and Execution
Front End
1 Parser
2 Initial query tree input to optimizer
3 Logical Transformation
Statistics (System Tables)
4 Physical Planning
Optimizer
Back End
Query plan sent to Execution Engine
5 Execution Engine
6 Compute Nodes

Table maintenance and status

Table skew

- Unbalanced workload
- Query completes as fast as the slowest slice completes
- Can cause skew inflight:
    - Temp data fills a single node, resulting in query failure

Unsorted table

- Sortkey is just a guide, but data actually needs to be sorted
- VACUUM or DEEP COPY to sort
- Scans against unsorted tables continue to benefit from zone maps:
    - Load sequential blocks

Cluster status: commits and WLM

WLM queue

Identify short/long-running queries and prioritize them

Define multiple queues to route queries appropriately

Default concurrency of 5

Leverage wlm_apex_hourly to tune WLM based on peak concurrency requirements

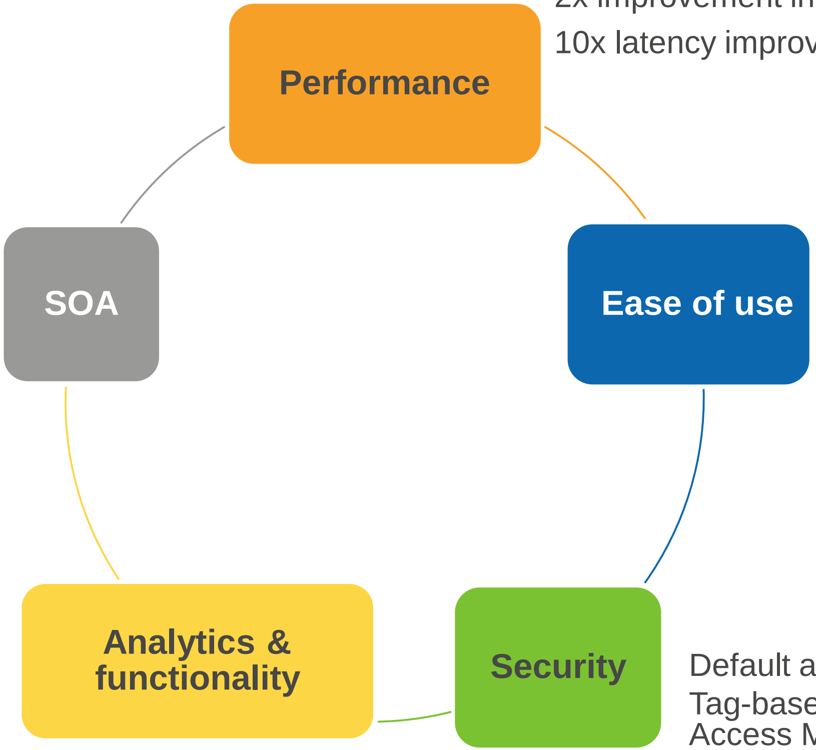
Commit queue

How long is your commit queue?

- Identify needless transactions
- Group dependent statements within a single transaction
- Offload operational workloads
- STL_COMMIT_STATS

What’s new?

Recent launches



Performance

Dynamic WLM parameters

Queue hopping for timed-out queries

Merge rows from staging to prod. table

2x improvement in query throughput

10x latency improvement for UNION ALL queries

Ease of use

BZIP2 format for ingestion

Table-level restore

10x improvement in vacuum perf

Security

Default access privileges

Tag-based AWS Identity and Access Management (IAM) access

IAM roles for COPY/UNLOAD

Analytics & functionality

SAS connector enhancements
Implicit conversion of SAS queries to Redshift

SOA

AWS Database Migration Service support from OLTP sources

Enhanced data ingestion from Amazon Kinesis Firehose

Improved data schema conversion to Amazon Machine Learning

VACUUM and throughput improvements

- 2x query throughput improvement
  - Improved memory management
  - Optimizations for query plans against UNION ALL views
  - Performance improvement for queries that stress the network
- 10x VACUUM improvement
- Improved backup performance

New features–ingestion

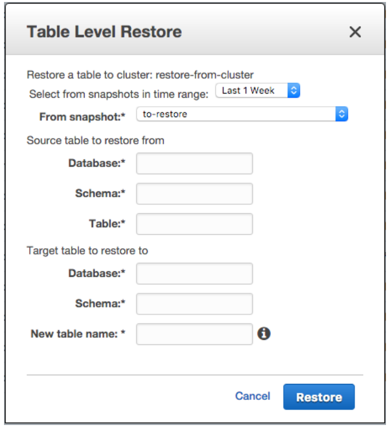
Backup option on CREATE TABLE

- For use on staging tables for enhanced load performance
- Table will not be present on restore

Alter Table Append

- Appends rows to a target table by moving data from an existing source table
- Data in the source table is moved to matching columns in the target table
- You cannot run ALTER TABLE APPEND within a transaction block (BEGIN ... END)

New feature–Table-level restore



Restore a table to cluster: restore-from-cluster

Select from snapshots in time range: Last 1 Week

From snapshot:* to-restore

Source table to restore from

Database:*

Schema:*

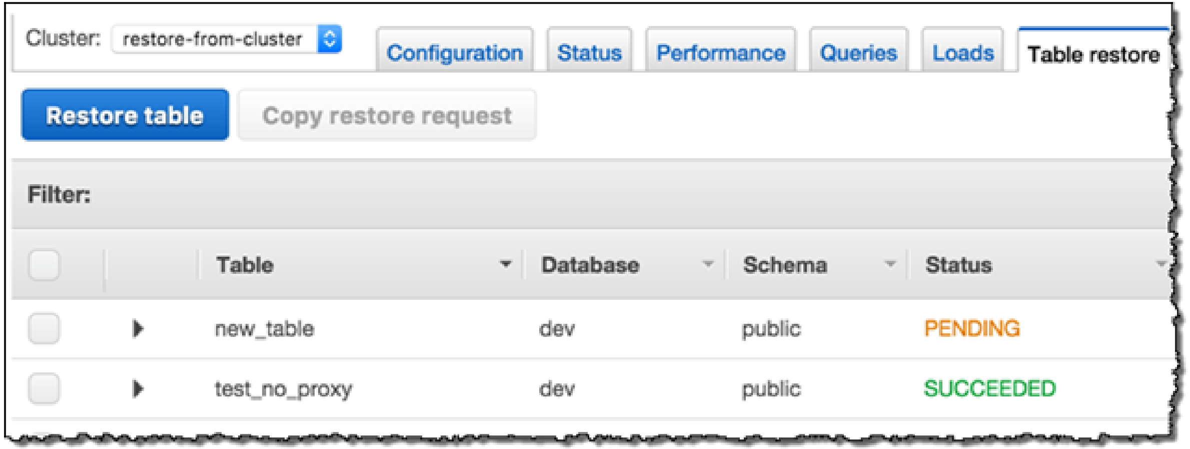
Table:*

Target table to restore to

Database:*

Schema:*

New table name: *



Cluster: restore-from-cluster

□		Table	Database	Schema	Status
□	▶	new_table	dev	public	PENDING
□	▶	test_no_proxy	dev	public	SUCCEEDED

```
aws redshift restore-table-from-cluster-snapshot --cluster-identifier mycluster-example
                                                 --new-table-name my-new-table
                                                 --snapshot-identifier my-snapshot-id
                                                 --source-database-name sample-database
                                                 --source-table-name my-source-table
```

Open source tools

https://github.com/awslabs/amazon-redshift-utils
https://github.com/awslabs/amazon-redshift-monitoring
https://github.com/awslabs/amazon-redshift-udfs

Admin scripts

Collection of utilities for running diagnostics on your cluster

Admin views

Collection of utilities for managing your cluster, generating schema DDL, etc.

ColumnEncodingUtility

Gives you the ability to apply optimal column encoding to an established schema with data already loaded

Remember to complete your evaluations!

AWS SUMMIT

Thank you!

amazon web services