Apache Ignite™

Distributed In-Memory SQL Queries

Denis Magda
GridGain Product Manager
Apache Ignite PMC

http://ignite.apache.org

#apacheignite

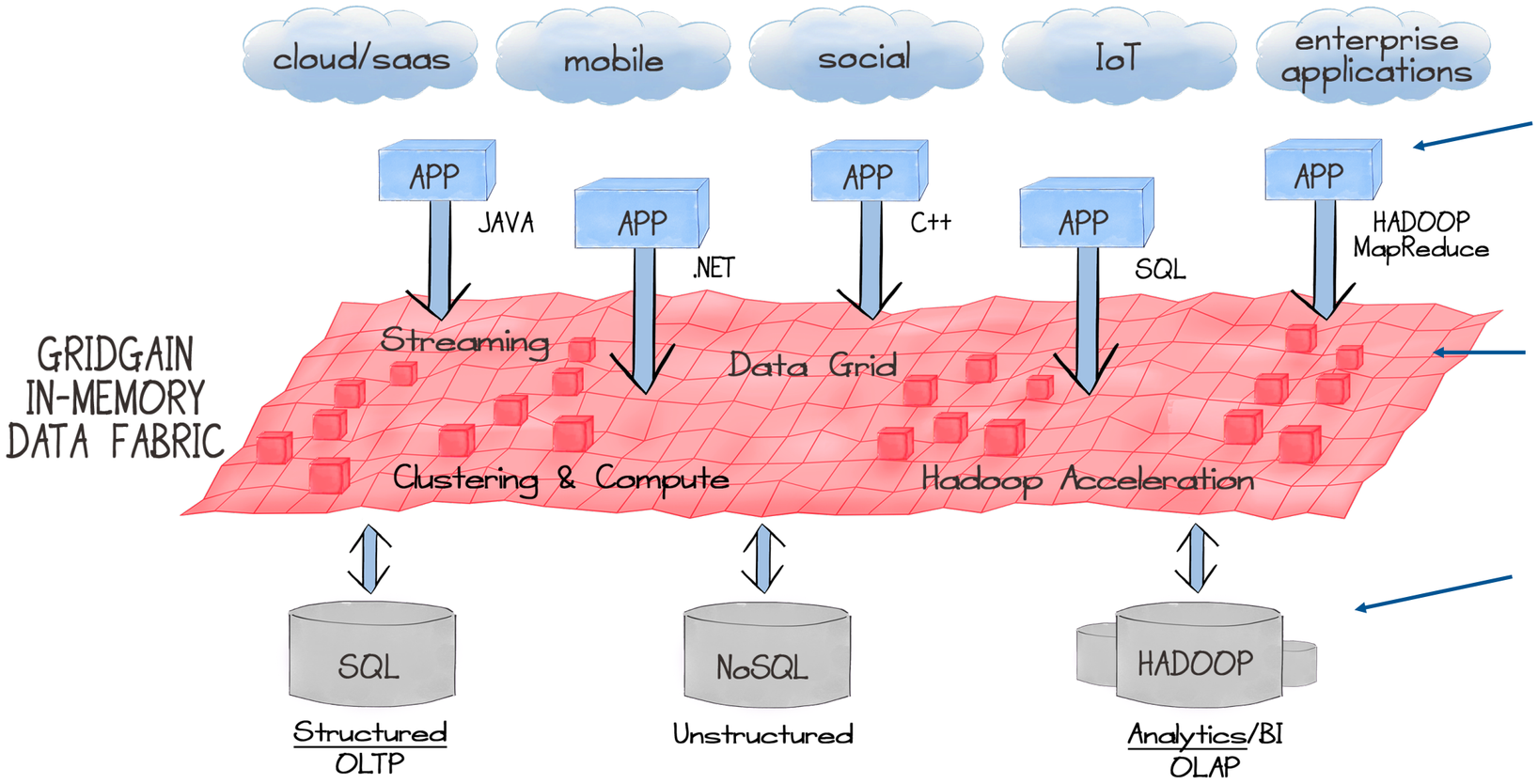
GridGain

Agenda

- Apache Ignite Overview
- Apache Ignite SQL Engine
  - Internals
  - Query Execution Flow
  - SQL API
  - Tips and Tricks
- Apache Ignite SQL Engine Of Tomorrow
- Demo

Apache Ignite Overview

Apache Ignite™ In-Memory Data Fabric: Strategic Approach to IMC



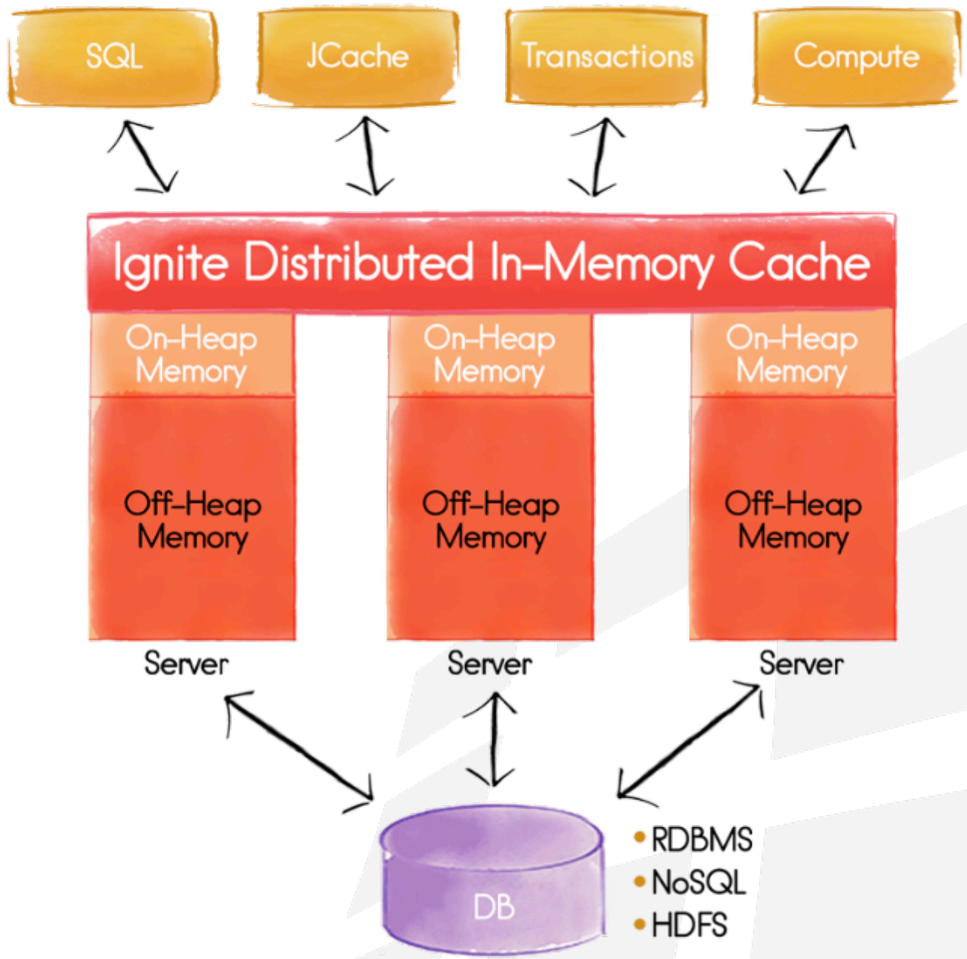
- Supports Applications of various types and languages

- Open Source – Apache 2.0
- Simple Java APIs
- 1 JAR Dependency
- High Performance & Scale
- Automatic Fault Tolerance
- Management/Monitoring
- Runs on Commodity Hardware

- Supports existing & new data sources
- No need to rip & replace

In-Memory Data Grid

- Distributed Key-Value Data Store
- Data Reliability
- High-Availability
  - Active replicas, automatic failover
- Data Consistency
  - ACID distributed transactions
- Distributed SQL
  - Advanced indexing



Apache Ignite SQL Engine

Apache Ignite SQL Engine

- Full ANSI-99 SQL Support
  - Aggregations, group by, sorting
  - Cross-cache joins, unions, etc.
- Distributed
  - Always consistent
  - Fault tolerant
- Advanced Indexing Support

Apache Ignite SQL Engine: H2 Database

- H2 Database
  - Fast in-memory and disk based DB
  - Written in Java and open sourced
- How Does it Relate to Ignite?
  - Started as a part of Ignite process
  - SQL parser and optimizer
  - Query execution and planning
  - Flexible indexing module
- Why not to use as is?
  - Not designed for key-value storages
  - Not a distributed engine by nature

Apache Ignite SQL Engine: Ignite + H2

- Ignite's Part of The Story
  - True cluster-wide distributed SQL
  - Distributed mapper and reducer
  - Paginated result sets
  - Fault tolerance and consistency
  - Sophisticated indexes implementations
- Data and indexes always stored on Ignite side!
  - No duplication

Apache Ignite SQL Engine: Indexing

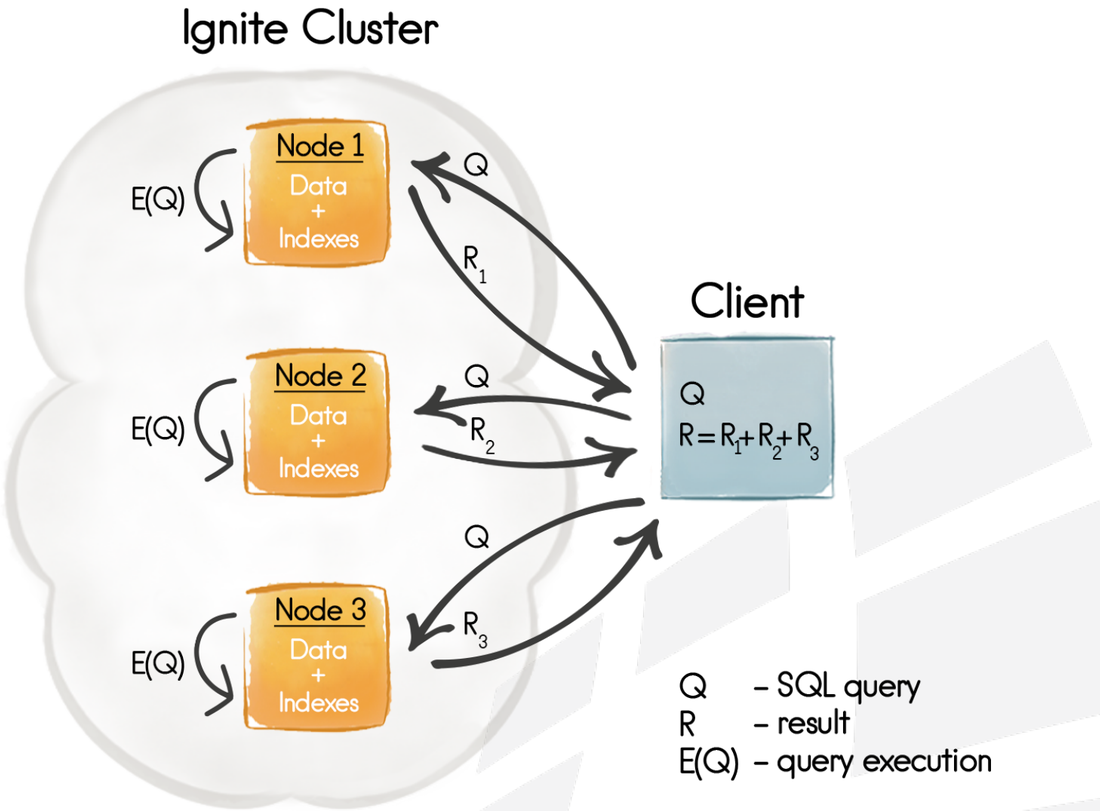
- Single Field and Group Indexes
  - Annotate in code
  - Predefine in the configuration
- On-Heap Indexes
  - AVL tree with fast cloning
  - Concurrent skip list (default)
- Off-Heap Indexes
  - AVL tree with fast cloning

```
public class Person implements Serializable {
  /** Will be indexed in ascending order. */
  @QuerySqlField(index = true)
  private long id;

  /** Will be visible in SQL, but not indexed. */
  @QuerySqlField
  private String name;

  /** Will be indexed in descending order. */
  @QuerySqlField(index = true, descending = true)
  private int age;
}
```

Query Execution Flow

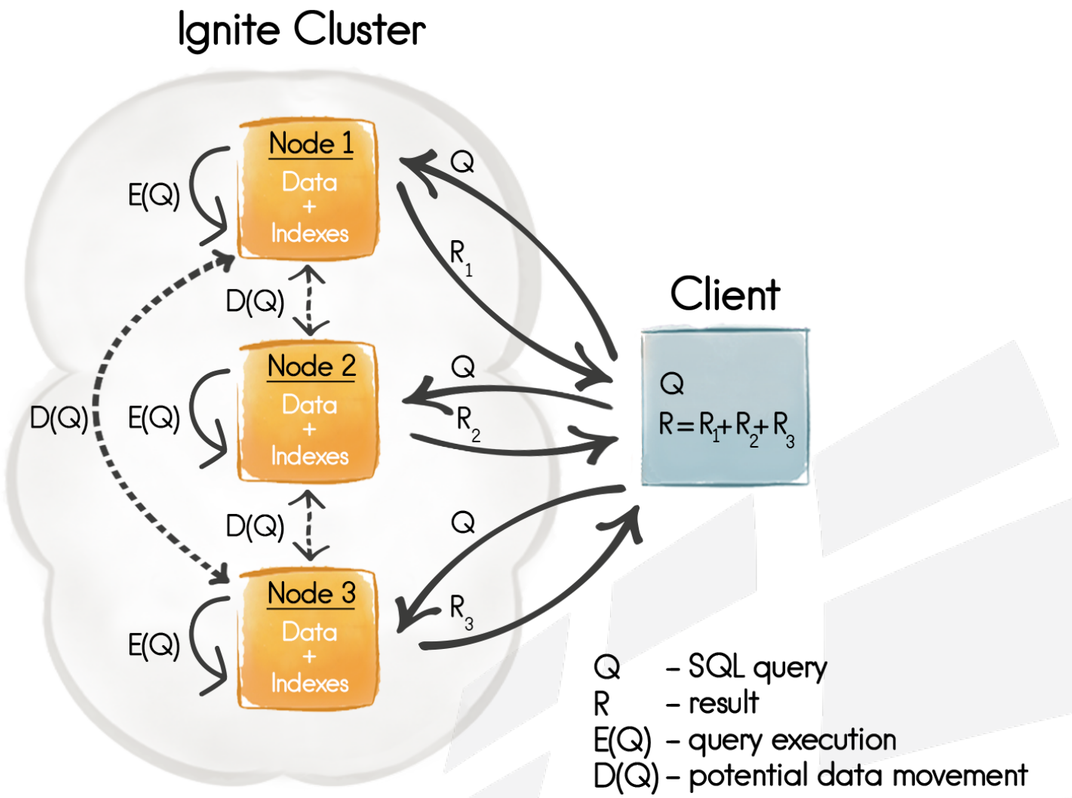
Apache Ignite SQL Engine: Collocated SQL Queries

- Collocated Mode
  - Any kind of JOINs (ANSI-99)
  - Data has to be collocated in advance*
- Recommended mode
  - No data movement between nodes
  - Enabled by default



Apache Ignite SQL Engine: Non-Collocated SQL Queries

- Non-Collocated Mode
  - Since Apache Ignite 1.7.0
  - No need to collocate data
  - Potential data movement between nodes
- Use case
  - No feasible to collocate data for particular SQL queries
  - Use collocated* mode as much as possible
- Disabled by default



Apache Ignite SQL Engine: Local Queries

- Local Query
  - Executed by H2 right away
  - Executed over local data set
  - Special query configuration parameter
- Use cases
  - Query over a single **replicated** cache
  - Query over partitioned or collocated data with IgniteCompute's affinity run

SQL API

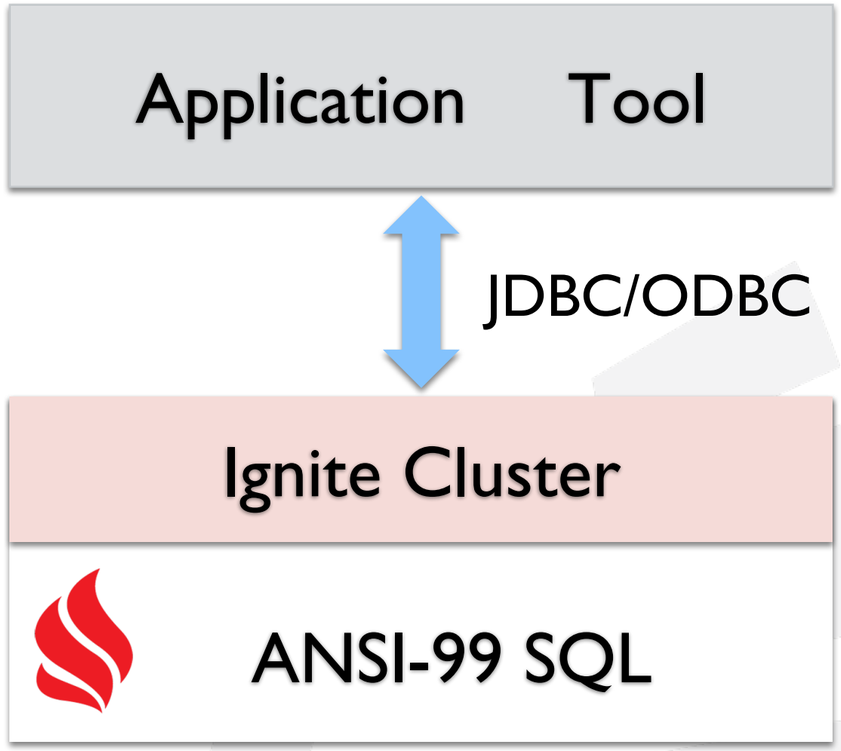
Apache Ignite SQL Engine: SQL API

- Unified Multi-Language API
  - Java & Scala
  - .NET
  - C++
- Two Query Types
  - SqlQuery
  - SqlFieldsQuery
  - Ad Hoc by nature
- Query Entity
  - Defines queryable type
  - Fields & indexes listing

```
IgniteCache<Long, Person> cache = ignite.cache("personCache");

// Execute query to get names of all employees.
SqlFieldsQuery sql = new SqlFieldsQuery(
  "select concat(firstName, ' ', lastName) from Person");

// Iterate over the result set.
try (QueryCursor<List<?>> cursor = cache.query(sql) {
  for (List<?> row : cursor)
    System.out.println("personName=" + row.get(0));
}
```

Apache Ignite SQL Engine: JDBC & ODBC

- Drivers for Standardized API
  - JDBC
  - ODBC
- Simple Start
  - Set cache configuration
  - Start a cluster and preload data
  - Connect from a driver side and act!
- JDBC Driver
  - Spawns and uses Ignite client node
- ODBC Driver
  - Connects to ODBC processor started on one of the cluster nodes



Tips and Tricks

Apache Ignite SQL Engine: Optimizations

- General Optimizations
  - Use group indexing
  - Don’t index everything
  - Adjust page size
- Off-Heap Optimizations
  - Tune on-heap row cache size
- Collocation Optimizations
  - Tend to collocate data
  - Use query’s *isCollocated* flag
  - Use compute affinity run along with local SQL queries

Apache Ignite SQL Engine: Debugging

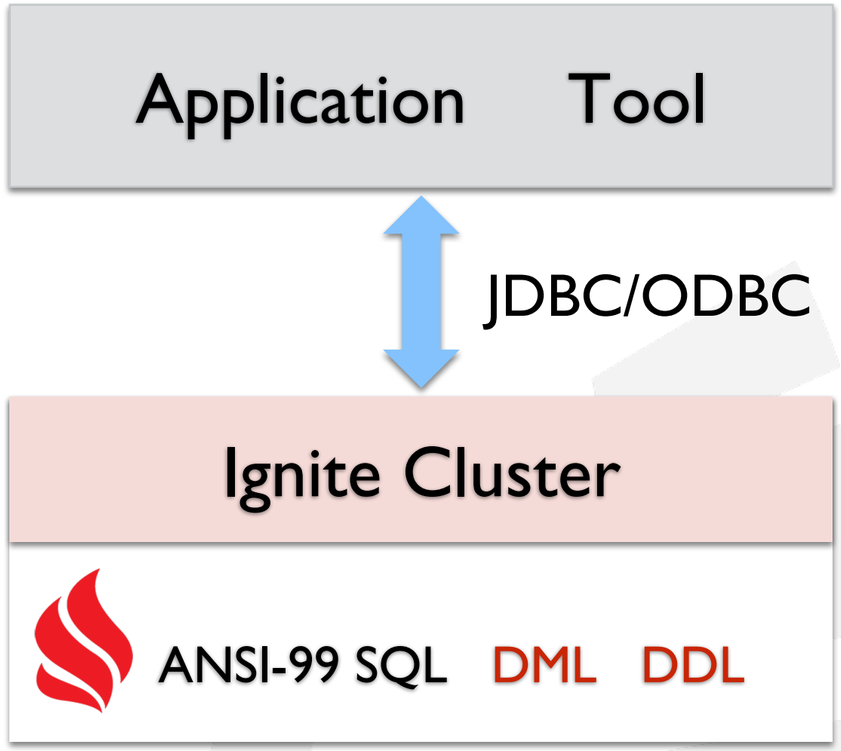
- EXPLAIN Statement
  - Indexes usage validation
- GridGain WebConsole
  - Query Execution & Debugging
  - Query Monitoring
- H2 Debug Console
  - Low level debugging

Apache Ignite SQL Engine of Tomorrow

GridGain

Apache Ignite SQL Engine: Future

- Connection Point
  - JDBC/ODBC
- Data Modifications
  - DML (INSERT, UPDATE, DELETE)
- Structure Modifications
  - DDL
  - Caches and indexes
- Querying
  - ANSI-99 SQL
- Don’t need to rewrite application from scratch!



Demo

ANY QUESTIONS?

Thank you for joining us. Follow the conversation.

http://ignite.apache.org

 #apacheignite