



technical white paper

Aster *n*Cluster In-Database MapReduce: Deriving Deep Insights from Large Datasets

Ajeet Singh, Director of Product Management

Contents

Introducing In-Database MapReduce	3
What is Google MapReduce?	3
A Simple Illustration of Google MapReduce	3
How Does Aster In-Database MapReduce Work?	4
How Do Developers Use In-Database MapReduce?	4
An Illustrative Example of In-Database MapReduce	6
Aster nCluster SQL/MR Syntax	7
Benefits of In-Database MapReduce	8
The Power of In-Database MapReduce: Simplifying ELT	8
The Power of In-Database MapReduce: Richer Analytics	8
The Power of Aster nCluster: Reliable Infrastructure	8
Comparison with Traditional Relational Databases	9
Who Can Use In-Database MapReduce?	10
Conclusion	10

Introducing In-Database MapReduce

A lack of expressive techniques to transform and analyze massive data volumes in organizations has led to costly workarounds in data warehousing. One of the first to face the challenge of analyzing petabyte-scale data, Google pioneered a software framework called MapReduce for high speed processing of large amounts of unstructured data. The framework was intended to run on clusters of commodity nodes. Successful with unstructured data, Google progressed to using MapReduce for analyzing and transforming structured as well as unstructured data.

In an industry first, Aster now brings the power of MapReduce to SQL RDBMS databases. Aster In-Database MapReduce combines the power of the MapReduce programming paradigm with the strengths of relational databases to allow users to perform fast analysis and transformations of structured data.

In-Database MapReduce enables developers to write rich analytic and transformation functions using MapReduce and invoke them easily through a standard SQL interface. In this way, developers can harness the parallel processing power of MapReduce while managing their data in Aster *n*Cluster, a highly scalable relational database for data warehousing.

What is Google MapReduce?

MapReduce is a software framework produced by Google to support parallel computations over large (multi-petabyte) datasets on clusters of commodity nodes.

Google MapReduce combines two classes of functions: *map* and *reduce*. These functions are defined with respect to data structured in (key, value) pairs. *Map* takes a pair of data with a type and returns a list of key-value pairs: $map(k1, v1) \rightarrow list(k2, v2)$.

The *map* function is applied in parallel to every item in the input dataset. This produces a list of $(k2, v2)$ pairs for each call. The framework collects all pairs with the same key from all lists and groups them together, thus creating one group for each key $k2$.

The *reduce* function is then applied in parallel to each group, which in turn produces a collection of values: $reduce(k2, Union_List(k2, v2)) \rightarrow List(v3)$.

The returns of all *reduce* functions are collected as the desired result set.

A Simple Illustration of Google MapReduce

A simple example of MapReduce is a program that counts the appearances of each unique word in a set of unstructured documents. In this program, the *map* function will emit $\langle word, countInDocument \rangle$ pairs for each word that occurs in each document. The *reduce* function for each word will receive partial counts from each document, sum up the partial counts and emit the final result.

```
map(String name, String document):  
    // Write code to open document  
    // Write code to scan document  
    for each word w in document:
```

```
EmitIntermediate(w, 1);

reduce(String word, Iterator intermediateCounts):
    // Write code to accept <word, intermediateCountList>
    // Write code to initialize final count, result
    for each v in intermediateCountList:
        result += ParseInt(v);
    Emit(word, result);
```

How Does Aster In-Database MapReduce Work?

Aster In-Database MapReduce operates on structured data stored within the *n*Cluster database. Functions written using Aster In-Database MapReduce are called SQL/MapReduce™ (SQL/MR™) functions. SQL/MR functions are embedded within standard SQL to enable simple data access and manipulation tasks via ordinary SQL, while performing higher value, logical manipulation of data structures using MapReduce.

Aster In-Database MapReduce reduces the time and effort required to implement analytic and transformation tasks that operate on large structured datasets. Specifically, Aster In-Database MapReduce provided the following benefits:

- **Ease of Development**
 - o Out-of-the-box data management capabilities of *n*Cluster relational database
 - o Standard SQL interface for rapid application development and invoking MapReduce functions
 - o Standard JDBC/ODBC support for seamless integration with ETL, BI, analytic tools and other systems
 - o Support for all standard SQL operations like INSERT, UPDATE, DELETE, CREATE TABLE AS, DROP TABLE
- **High Performance**
 - o Network-optimized data placement, data transport and query planning for performance and scalability
 - o Cost-based query optimization leveraging indexes, partition pruning, in-memory joins, sorts and aggregates
 - o Multi-tier architecture provides independent scaling of loads, query execution and concurrency management according to workload characteristics
- **Ease of Management**
 - o Transparent recovery and load-balancing of data in the event of transient or permanent node failures
 - o Backup and recovery capabilities for data protection
 - o Schema support for creating integrity constraints
 - o Resource management of queries for handling mixed workloads
 - o Security and access control through authentication, roles and privileges

How Do Developers Use In-Database MapReduce?

In-Database MapReduce queries are simple to write. They rely on SQL queries to manipulate underlying data and provide input. SQL/MR functions can procedurally manipulate input data and provide outputs that can be consumed by ordinary SQL queries

or be written into database tables. The Aster nCluster database understands the input and output data characteristics to automatically optimize SQL processing and provide fault tolerance, load balancing and workload management.

SQL/MR functions enable developers to push down data processing from their application tier into Aster nCluster. The processing push-down provides performance benefits from massively parallel execution as well as vastly reduced data transfer between database and application tiers.

SQL/MR functions can be implemented and used as follows:

1. Application programmers write their own *map* and *reduce* functions using programming languages such as Java, Python, Perl, C++, C, etc.
2. SQL/MR functions are uploaded into Aster nCluster at the Queen node,
3. SQL/MR functions are invoked from within ordinary SQL. A SQL statement can invoke multiple SQL/MR functions in an ad-hoc fashion, which creates seamless data flow between nCluster tables and SQL/MR functions,
4. Aster nCluster executes SQL and SQL/MR functions across the nodes of its massively parallel database
5. The results of the SQL/MR query execution are returned as a standard SQL result set to the application tier for further handling according to the application business logic.

The following graphic illustrates the flow of queries and the seamless execution of SQL and SQL/MR functions inside the nCluster database:

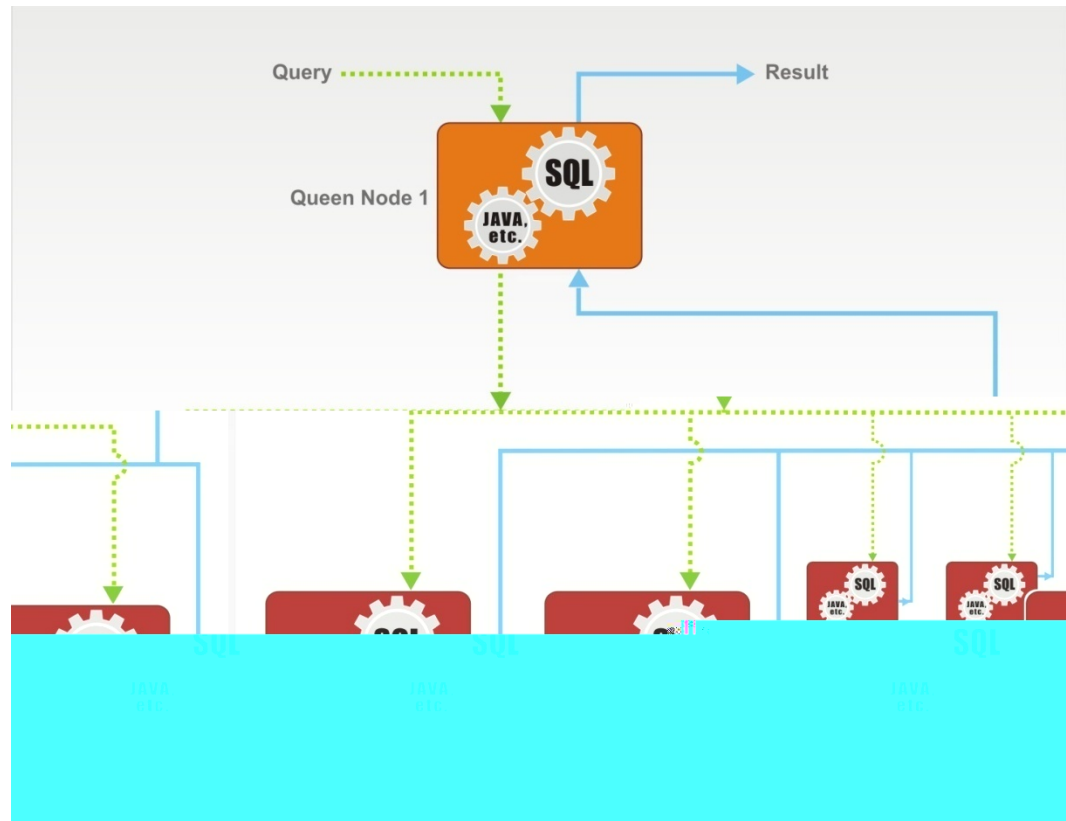


Figure 1: Seamless execution of SQL and SQL/MR functions in *n*Cluster

An Illustrative Example of In-Database MapReduce

The example program above of Google MapReduce counted the number of appearances of each unique word in a set of documents. In the case of structured data stored in an RDBMS, we don't need SQL/MR functions for simple queries such as word counts. Assuming that we had two columns called *wordId* and *docId* within a database called *WordOccurrences*, we would use the following ordinary SQL query to find the answer to the question of how many times a particular word occurred in the database:

```
select wordId, count(*)
from WordOccurrences
group by wordId ;
```

In other words, in RDBMS's there are many cases where simple queries or tasks do not require special *map* and *reduce* functions. Instead, a developer can leverage ordinary SQL directly for fast performance and correct answers via simple code.

However, suppose again that instead of a column called *wordId*, we had a column called *word* that stored each word as a string. And suppose we want to count, not just exact words, but all words with a common stem. Before counting, we want to stem all our words into tokens (i.e., counting “occurs”, “occurrence”, “occurred”, etc. as variants of “occur”). In this case, we would use the following SQL/MR function that relies on a *map* function for its tokenization and on SQL functions (i.e., GROUP BY) for its final reduce:

```
select token, sum(occurrences) as globalOccurrence
from map ( ON
  select word, count(*) as occurrences
  from WordOccurrences
  group by word )
group by token;
```

Notice that the *map* function here is more powerful than a User Defined Function (UDF) – the usual method for creating customized database functions. The *map* function can tokenize the words, maintain its own in-memory hash table of tokens and update the counters for the token “occur” even as it encounters “occurs”, “occurrence”, “occurred”, etc. and finally emit the list of tokens and their counters. This results in a single-pass algorithm over data that shrinks datasets during computation, resulting in much faster performance. A normal scalar UDF would emit as many tokens as the number of rows in WordOccurrences for which the GROUP BY command would be required to operate on a temporary table first written to, and then read from disk. In general, *map* functions are fully parallelizable, unlike most UDFs.

Aster *n*Cluster SQL/MR Syntax

The following example illustrates how a C++ based SQL/MR function might be implemented and invoked from within SQL:

Map function:

```
class Map {
  public void operateOnInput(RowIterator input, RowEmitter output)
  {
    //body ...
  }
}
```

Reduce Function:

```
class Reduce {
  public void operateOnInput(RowIterator input, RowEmitter output)
  {
    //body ...
  }
}
```

SQL Invocation:

```
SELECT key, value
FROM reduce (
  ON (
    SELECT key, value
    FROM map (
      ON source
    )
  )
  PARTITION BY key
);
```

Benefits of In-Database MapReduce

The Power of In-Database MapReduce: Simplifying ELT

A significant amount of the work involved in loading data into data warehouses is consumed by transformations after new data has been inserted into staging tables. With SQL/MR functions, multiple summaries and transformations can be computed in one pass over data.

As an example, take the **Sessionization** task in Internet data warehouses. Given a set of clickstream logs, we need to assign Session Identifiers to each record such that two visits by a user that are within (say) 30 minutes of each other are assigned the same Session Identifier; and a new Session Identifier is assigned to the first visit that is separated from the previous visit by more than 30 minutes.

A SQL/MR function for **Sessionization** is a simple SELECT query that invokes a ten line MapReduce function in Java or Python. Even better, it runs extremely fast, finishing the query in a single pass over the data without requiring multiple sorts of the large clickstream table!

As another example, take the **Tokenization** task in Customer or Product Inventory data warehouses. Given a set of URLs/Search keywords/People Names/Company Names, we need to stem the URLs, email addresses, keywords, names, correct mis-spellings and expand acronyms. This is an onerous task using SQL but very easy to program using the rich text manipulation functions present in Perl and Python that can be leveraged by a SQL/MR function!

The Power of In-Database MapReduce: Richer Analytics

SQL queries can handle generating simple reports that compute Key Performance Indicators (KPIs). A KPI is more often than not a simple running counter or sum that can be computed incrementally from (summary) tables. SQL queries are not designed for rich analytics – for example in computing statistical measures or patterns. A SQL/MR function can be used to compute patterns in one pass over data, using the power of parallelization to analyze large volumes of data in a short period.

As an example, take time series analysis in data warehouses. Many data warehouses need time series analysis; after all they are a repository of historical time-varying events. For example, we may be interested in finding all customers who have purchased product X (say, iPod) followed by Y (iPod accessory) and then recently Z (Phone) so that we can offer them product A (say, iPhone accessories) in a promotion. What makes this task complex in ordinary SQL is that the customer may have purchased products in the middle of the pattern that have nothing to do with X, Y, Z and A. But this task is almost trivial to write with SQL/MR functions leveraging regular expression matching.

The Power of Aster nCluster: Reliable Infrastructure

Aster nCluster provides In-Database MapReduce within a reliable infrastructure platform. The Aster nCluster database takes care of data distribution, transactional consistency,

query optimization, job scheduling, query prioritization, failure recovery and incremental scaling.

Each SQL/MR function executes in a Process Container that prohibits one query from consuming more resources than dictated by the Administrator. This means that an application developer's code runs within the warehouse without exposing the warehouse to the risk of poorly-written code that leaks memory and makes the warehouse unavailable to critical KPI reports.

Each SQL/MR function can be assigned a priority that forces it to play well with other critical warehousing tasks like loading, reporting, backups and exports. And each query that calls SQL/MR functions can be launched or stopped via the same database management console that monitors all SQL queries. SQL/MR functions can be used and re-used by multiple users via the same access-control primitives that protect data within the data warehouse.

Since database infrastructure tasks are taken care of, we can open up the data warehouse to become a public utility, a place of freedom for analysis, accessible to more people with ideas, providing them with an ability to extract value inside the warehouse under the careful eye of the data services group.

Comparison with Traditional Relational Databases

A lack of expressive techniques to transform and analyze massive data volumes has led to costly workarounds in data warehousing. A real-life data warehouse requires writing and debugging rigidly typed User Defined Functions (because relational databases require strong meta-data to explain a UDF's black-box code to database internal primitives), slow Stored Procedures (because traditional databases do not parallelize stored procedures well) or multi-pages of slow SQL queries (because several intermediate temporary tables are required to store partial computations).

The limitations of ordinary SQL have caused application programmers to move data out of the data warehouse and manipulate it in application servers using their own code using languages such as Java/Python/C++, etc.

With Aster In-Database MapReduce, there is now a new paradigm for analyzing and manipulating data. In particular, it provides key benefits over traditional relational databases:

- The push-down of computation into the database enabled by In-Database MapReduce significantly reduces data transport between ETL(extract/transform/load), analytics and database tiers, providing faster performance and ability to analyze larger datasets
- Massively parallel execution of SQL/MR functions on all nodes of the database provides high performance
- SQL/MR can be written in a wide choice of programming languages such as Java, Perl, Python, C++, R and more

- Polymorphic functions and dynamic typing allow reuse of SQL/MR functions with varied schema
- Better system availability through fault isolation of SQL/MR functions
- Management of memory resources consumed by SQL/MR functions

Who Can Use In-Database MapReduce?

SQL/MR functions can be leveraged by three classes of users:

1. Application developers: To create high-performance applications that analyze large datasets by writing SQL/MR functions
2. ELT developers: To write powerful (yet short) SQL/MR programs to transform datasets after loading
3. Analytic (Data Mining) developers: To write in-database functions that perform rich analysis of data without requiring slow exports

Conclusion

Aster In-Database MapReduce is a breakthrough innovation in analytic database capability. By bringing together the expressive flexibility and performance of the MapReduce framework with the rich functionality and familiarity of ordinary SQL, developers and analysts can help their businesses do more with data. For more information on Aster In-Database MapReduce, please visit the Aster Data Systems web-site at www.asterdata.com, or call us at 650.232.4400 / 1.888.Aster.Data.

About Aster Data Systems

Aster Data Systems is a proven innovator in massively parallel processing (MPP) databases for data warehousing and analytics – bringing deep insights on massive data analyzed on clusters of commodity hardware. Co-founded by three colleagues in the Stanford Computer Science Ph.D. program, the Aster *n*Cluster database provides patent-pending innovations in scalability, manageability, availability, and analytics. Aster is headquartered in Redwood City, California and is backed by Sequoia Capital, Cambrian Ventures, First-Round Capital and Stanford University Computer Science Professors David Cheriton and Rajeev Motwani.