Use SQL-like languages for the MapReduce framework

Make Hadoop more accessible with high-level declarative interfaces

Sherif Sakr (ssakr@cse.unsw.edu.au)
Research Scientist
National ICT Australia

17 April 2012

Select the most suitable MapReduce implementation for large scale data analysis jobs based on your skills, preferences, and requirements. MapReduce is a simple and powerful programming model that enables the easy development of scalable parallel applications to process vast amounts of data on large clusters of commodity machines. It isolates the application from the details of running a distributed program. But many programmers are unfamiliar with the MapReduce programming style and prefer to use a SQL-like language to perform their tasks. In this article, read an overview of high-level languages and systems designed to tackle these problems and add declarative interfaces on top of the MapReduce framework.

Introduction

Over the last two decades, the steady increase in computational power has produced an overwhelming flow of data, which in turn has led to a paradigm shift in computing architecture and large scale data processing mechanisms. For example, powerful telescopes in astronomy, particle accelerators in physics, and genome sequencers in biology put massive volumes of data into the hands of scientists. Facebook collects 15 TeraBytes of data each day into a PetaByte-scale data warehouse. The demand for large-scale data mining and data analysis applications is growing in both industry (for example, web data analysis, click-stream analysis, and network-monitoring log analysis) and the sciences (for example, analysis of data produced by massive-scale simulations, sensor deployments, and high-throughput lab equipment). Although parallel database systems serve some of these data analysis applications, they are expensive, difficult to administer, and lack fault-tolerance for long-running queries.

MapReduce is a framework introduced by Google for programming commodity computer clusters to perform large-scale data processing in a single pass. The framework is designed in a way that a MapReduce cluster can scale to thousands of nodes in a fault-tolerant manner. But the MapReduce programming model has its own limitations. Its one-input and two-stage data flow is extremely rigid, in addition to the fact that it is very low-level. For example, you must write custom code for even the most common operations. Hence, many programmers feel uncomfortable with the MapReduce framework and prefer to use SQL as a high-level declarative language. Several projects (Apache Pig, Apache Hive, and HadoopDB) have been developed to ease the task of programmers and provide high-level declarative interfaces on top of the MapReduce framework.

First look at the MapReduce framework and then at the capabilities of different systems that provide high-level interfaces to the MapReduce framework.

The MapReduce framework

A main advantage of the MapReduce framework approach is that it isolates the application from the details of running a distributed program, such as issues of data distribution, scheduling, and fault tolerance. In this model, the computation takes a set of input key/value pairs and produces a set of output key/value pairs. The user of the MapReduce framework expresses the computation using two functions: `Map` and `Reduce`. The `Map` function takes an input pair and produces a set of intermediate key/value pairs. The MapReduce framework groups together all intermediate values associated with the same intermediate key `I` (shuffling) and passes them to the `Reduce` function. The `reduce` function receives an intermediate key `I` with its set of values and merges them together. Typically, only zero or one output value is produced per `reduce` invocation. The main advantage of this model is that it allows large computations to be easily parallelized and re-executed to be used as the primary mechanism for fault tolerance.

The Apache Hadoop project is open source Java™ software that supports data-intensive distributed applications by realizing the implementation of the MapReduce framework. It was originally developed by Yahoo! as a clone of the Google MapReduce infrastructure but it subsequently became open source. Hadoop takes care of running your code across a cluster of machines. In general, when a dataset outgrows the storage capacity of a single physical machine, it becomes necessary to partition it across a number of separate machines. Filesystems that manage the storage across a network of machines are called distributed filesystems. Hadoop comes with a distributed filesystem called HDFS (Hadoop Distributed Filesystem). In particular, HDFS is a distributed filesystem that stores files across all of the nodes in a Hadoop cluster. It breaks the files into large blocks and distributes them across different machines, plus makes multiple copies of each block so that if any one machine fails, no data is lost.

The MapReduce program in Listing 1 is expressed in pseudo-code for counting the number of occurrences of each word in a sequence of text lines. In Listing 1, the `map` function emits each word plus an associated mark of occurrences while the `reduce` function sums together all marks emitted for a particular word.

Listing 1. MapReduce program

```
     map(String key, String value):
     //key: line number, value: line text
     for each word w in value:
          EmitIntermediate(w, ?1?);

     reduce(String key, Iterator values):
          //key: a word, values: a list of counts
     int wordCount = 0;
     for each v in values:
          wordCount += ParseInt(v);
     Emit(AsString(wordCount));
```

Now assume the input sequence of text lines in Listing 2.

Listing 2. Input sequence

```
1, This is Code Example
2, Example Color is Red
3, Car Color is Green
```

Listing 3 shows the output of the `map` function of this input.

Listing 3. Output of `map` function

```
('This', 1), ('is', 1). ('Code', 1), ('Example', 1)
('Example', 1), ('Color', 1), ('is', 1), ('Red', 1)
('Car', 1), ('Color', 1), ('is', 1), ('Green', 1)
```

Listing 4 shows the output of the `reduce` function (the result).

Listing 4. Output of the `reduce` function

```
('Car', 1), ('Code', 1), ('Color', 2), ('Example', 2), ('Green', 1), ('Red', 1)
 , ('This', 1), ('is', 3)
```

For programmers, a key feature of the MapReduce framework is that there are only two high-level declarative primitives (`map` and `reduce`) that can be written in any programming language of choice without worrying about the details of their parallel execution. On the other hand, the MapReduce programming model has its own limitations:

1. Its one-input and two-stage data flow is extremely rigid. To perform tasks having a different data flow (for example, joins or n stages), you have to devise inelegant workarounds.
2. Custom code is written for even the most common operations (for example, projection and filtering), which leads to code that is usually difficult to reuse and maintain.
3. The opaque nature of the `map` and `reduce` functions impedes the ability of the system to perform optimizations.

Moreover, many programmers are unfamiliar with the MapReduce framework and prefer to use SQL (because they are more proficient in it) as a high-level declarative language to express their task while leaving all of the execution optimization details to the backend engine. In addition, it is also true that high-level language abstractions enable the underlying system to better perform automatic optimizations.

Let's examine the languages and systems designed to tackle these problems and add the SQL flavor on top of the MapReduce framework.

Pig

The Apache Pig project is designed as an engine for executing data flows in parallel on Hadoop. It uses a language , called Pig Latin to express these data flows. With Pig Latin, you can describe how data from one or more inputs should read, process, and then store to one or more outputs in parallel. The language takes a middle position between expressing tasks using high-level declarative querying model in the spirit of SQL and low-level/procedural programming using

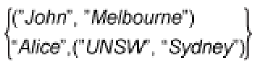
MapReduce. The Pig Latin data flows can be simple linear flows, but can also be complex workflows that include points where multiple inputs are joined and where data is split into multiple streams to be processed by different operators.

A Pig Latin program consists of a series of operations, or transformations, that are applied to the input data to produce the output. Taken as a whole, the operations describe a data flow that the Pig execution environment translates into an executable representation and is then executed. Under the covers, Pig turns the transformations into a series of MapReduce jobs.

With Pig, the data structures are much richer, typically being multi-valued and nested; and the set of transformations you can apply to the data are much more powerful. In particular, the Pig Latin data model consists of the following four types:

1. *Atom* is a simple atomic value such as a string or a number, for example, "John".
2. *Tuple* is a sequence of fields, each of which can be any of the data types, for example, ("John", "Melbourne").
3. *Bag* is a collection of tuples with possible duplicates. The schema of the constituent tuples is flexible where not all tuples in a bag need to have the same number and type of fields. The Bag in Figure 1 lists two tuples: ("John","Melbourne") and "Alice",("UNSW" "Sydney").

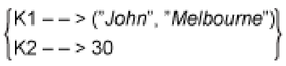
   Figure 1. A Bag

   $$\left\{\begin{array}{l}(\text{"John"}, \text{"Melbourne"}) \\ \text{"Alice"}, (\text{"UNSW"}, \text{"Sydney"})\end{array}\right\}$$

4. *Map* is a collection of data items, where each item has an associated key through which it can be looked up. As with bags, the schema of the constituent data items is flexible, however, the keys are required to be data atoms> The Map in Figure 2) list data items: K1-->("John","Melbourne") and K2-->30.

   Figure 2. A Map

   $$\left\{\begin{array}{l}K1 --> (\text{"John"}, \text{"Melbourne"}) \\ K2 --> 30\end{array}\right\}$$

Pig Latin includes operators for many of the traditional data operations (`join`, `sort`, `filter`, `group by`, `union`, and so on), as well as the ability for users to develop their own functions to read, process, and write data.

MapReduce provides the group by operation directly (of which the shuffle plus reduce phases essentially are), and it provides the order by operation indirectly through the way it implements the grouping. Filter and projection can be implemented trivially in the map phase. But other operators, particularly join, are not provided and must instead be written by the user. Pig provides some complex, nontrivial implementations of these standard data operations. For example, because the number of records per key in a dataset is rarely evenly distributed, the data sent to the reducers is often skewed. That is, one reducer will get 10 or more times the data than other reducers. Pig has join and order by operators that will handle this case and (in some cases) rebalance the reducers. Table 1 describes the main operators of the Pig Latin language.

In MapReduce, the data processing inside the map and reduce phases is opaque to the system. This means that MapReduce has no opportunity to optimize or check user code. Pig, on the other hand, can analyse a Pig Latin script and understand the data flow that the user is describing. MapReduce does not have a type system. This is intentional, and it gives users the flexibility to use their own data types and serialization frameworks. But the downside is that this further limits the system's ability to check user code for errors both before and during runtime. All of these points mean that Pig Latin is much lower cost to write and maintain than Java code for MapReduce.

Table 1. The main operators of the Pig Latin language

Operator	Description
LOAD	Loads data from the filesystem or other storage into a relation
DUMP	Prints a relation to the system console FILTER
DISTINCT	Removes duplicate rows from a relation
FOREACH ... GENERATE	Adds or removes fields from a relation
JOIN	Joins two or more relations
ORDER	Sorts a relation by one or more fields
LIMIT	Limits the size of a relation to a maximum number of tuples
STORE	Saves a relation to the filesystem or other storage
FILTER	Removes unwanted rows from a relation
GROUP	Groups the data in a single relation
CROSS	Creates the cross product of two or more relations
UNION	Combines two or more relations into one relation
SPLIT	Splits a relation into two or more relations

Listing 5 shows a simple Pig Latin program to find all employees with a high salary.

Listing 5. Find all employees with a high salary

```
employees = LOAD 'employee.txt' AS (id, name, salary);
highSalary = FILTER employees BY salary > 100000;
sortedList = ORDER highSalary BY salary DESC;
STORE sortedList INTO ' highSalary _Emp';
DUMP sortedList;
```

In this example, first load the input file into a bag called employees. Then, create a new bag called highSalary, which contains just those records where the salary field is greater than 100000. The sortedList bag orders the filtered records based on the salary value in a descending order. Finally, write the contents of the sortedList bag to the HDFS and print the bag contents on the screen.

Listing 6 shows how to describe`join` operations easily using Pig Latin.

Listing 6. `join` operations can be easily described using Pig Latin

```
employees = LOAD 'employee.txt' AS (id, name, salary, dept);
departments = LOAD 'dept.txt' AS (deptID, deptName);
joinList = JOIN employees BY dept, departments BY deptID;
STORE joinList INTO ' joinFile';
```

Traditionally, ad-hoc queries are done in languages such as SQL that make it easy to quickly form a question for the data to answer. For research on raw data, some users prefer Pig Latin. Because Pig can operate in situations where the schema is unknown, incomplete, or inconsistent, and because it can easily manage nested data, researchers who want to work on data before it has been cleaned and loaded into the warehouse often prefer Pig. Researchers who work with large data sets often use scripting languages such as Perl or Python to do their processing. Users with these backgrounds often prefer the dataflow paradigm of Pig over the declarative query paradigm of SQL.

Hive

The Apache Hive project is an open-source data warehousing solution built by the Facebook Data Infrastructure Team on top of the Hadoop environment. The main goal of this project is to bring the familiar relational database concepts (for example, tables, columns, partitions) and a subset of SQL to the unstructured world of Hadoop while still maintaining the extensibility and flexibility that Hadoop enjoyed. Thus, it supports all the major primitive types (for example, integers, floats, strings) as well as complex types (for example, maps, lists, structs). Hive supports queries expressed in an SQL-like declarative language, HiveQL (Hive Query Language), and therefore can be easily understood by anyone who is familiar with SQL. These queries automatically compile into MapReduce jobs that are executed using Hadoop. In addition, HiveQL enables users to plug in custom MapReduce scripts into queries.

HiveQL supports Data Definition Language (DDL) statements, which you can use to create, drop, and alter tables in a database. It allows users to load data from external sources and insert query results into Hive tables through the load and insert Data Manipulation Language (DML) statements respectively. However, HiveQL currently does not support the update and deletion of rows in existing tables (in particular, `INSERT INTO`, `UPDATE`, and `DELETE` statements), which allows the use of very simple mechanisms to deal with concurrent read and write operations without implementing complex locking protocols. The metastore component is the Hive's system catalog, which stores metadata about the underlying table. This metadata is specified during table creation and reused every time the table is referenced in HiveQL. The metastore distinguishes Hive as a traditional warehousing solution when compared with similar data processing systems that are built on top of MapReduce-like architectures like Pig Latin.

Listing 7 shows examples of HiveQL statements describing the operations to create a table, load a data, and query the table contents.

Listing 7. HiveQL statements describing the operations for creating a table, loading a data, and querying the table contents

```
CREATE TABLE employee (empID INT, name STRING, salary BIGINT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE;
LOAD DATA INPATH "employee_data" INTO TABLE employee;
SELECT * FROM employee WHERE salary > 100000 SORT BY salary DESC;
```

Hive also supports manipulation of data through user-created functions (see Listing 8).

Listing 8. Hive supports manipulations data through user-created functions

```
INSERT OVERWRITE TABLE employee
SELECT
TRANSFORM (empID, name, salary, address, department)
USING 'python employee_mapper.py'
AS (empID, name, salary, city)
FROM employee_data;
```

In general, Hive is a great interface for anyone from the relational database world, though the details of the underlying implementation aren't completely hidden. You do still have to worry about some differences in things like the most optimal way to specify joins for best performance and some missing language features. Hive does offer the ability to plug in custom code for situations that don't fit into SQL, as well as a lot of tools to handle input and output. Hive suffers from some limitations such as it lacks support for `UPDATE` or `DELETE` statements, `INSERT`ing single rows, and date or time datatypes, since they are treated as strings.

HadoopDB

The HadoopDB project, commercialized by the company Hadapt, is a hybrid system that tries to combine the scalability advantages of MapReduce with the performance and efficiency advantages of parallel databases. The basic idea behind HadoopDB is to connect multiple single node database systems (PostgreSQL) using Hadoop as the task coordinator and network communication layer. Queries are expressed in SQL but their executions are parallelized across nodes using the MapReduce framework such that single query work is pushed, as much as possible, into the corresponding node database.

In general, parallel database systems have been commercially available for nearly two decades and there are now about a dozen of different implementations in the marketplace (for example, Teradata, Aster Data, Greenplum). The main aim of these systems is to improve performance through the parallelization of various operations such as loading data, building indices, and evaluating queries. In general, some key reasons make MapReduce a more preferable approach over a parallel RDBMS in some scenarios:

- Formatting and loading a huge amount of data into a parallel RDBMS in a timely manner is a challenging and time-consuming task.
- The input data records might not always follow the same schema. Developers often want the flexibility to add and drop attributes and the interpretation of an input data record can also change over time.

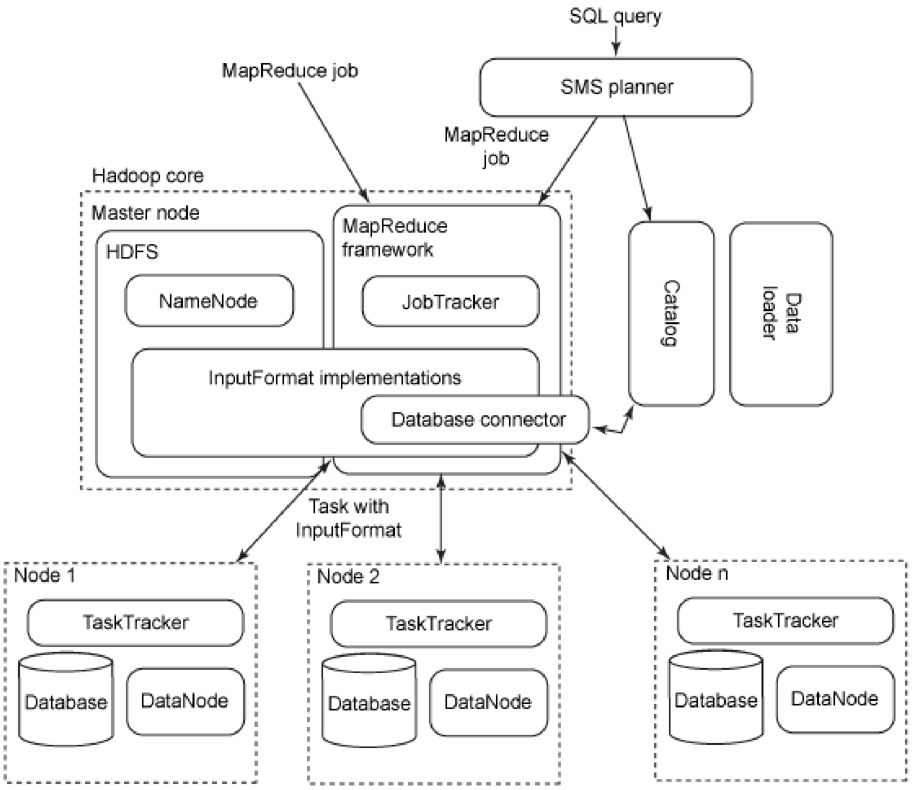
- Large scale data processing can be very time consuming and it is therefore important to keep the analysis job going even in the event of failures. While most parallel RDBMSs have fault tolerance support, a query usually has to be restarted from scratch even if just one node in the cluster fails. In contrast, MapReduce deals more gracefully with failures and can redo only the part of the computation that was lost because of a failure.

The comparison between the MapReduce framework and parallel database systems is a long-standing debate. A large scale comparison between the Hadoop implementation of MapReduce framework and parallel SQL database management systems in terms of performance and development complexity has been conducted. The results of this comparison show that parallel database systems displayed a significant performance advantage over MapReduce in executing a variety of data intensive analysis tasks. On the other hand, the Hadoop implementation was easier and more straightforward to set up and use in comparison to the parallel database systems. MapReduce has also shown to have superior performance in minimizing the amount of work that is lost when a hardware failure occurs. In addition, MapReduce (with its open source implementations) represents a very cheap solution in comparison to the very financially expensive parallel DBMS solutions.

Originally, the main applications of the MapReduce framework focused on the analysis of very large non-structured datasets such as: web indexing, text analytics, and graph data mining. Recently, as MapReduce is steadily developing into the de facto data analysis standard, it is repeatedly employed to query structured data. For a long time, relational database has dominated the deployments of data warehousing systems and the performance of analytical jobs on structured data. Interest is increasing in combining MapReduce and traditional database systems to maintain the benefits of both worlds. In particular, HadoopDB tries to achieve fault tolerance and the ability to operate in heterogeneous environments by inheriting the scheduling and job tracking implementation from Hadoop. It tries to achieve the performance benefits of parallel databases by doing most of the query processing inside the database engine.

Figure 3 illustrates the architecture of HadoopDB which consists of two layers: 1) A data storage layer or the HDFS, and 2) A data processing layer or the MapReduce framework.

Figure 3. The architecture of HadoopDB



In this architecture, HDFS is a block-structured file system managed by a central Name Node. Individual files are broken into blocks of a fixed size and distributed across multiple Data Nodes in the cluster. The Name Node maintains metadata about the size and location of blocks as well as their replicas. The MapReduce framework follows a simple master-slave architecture. The master is a single Job Tracker and the slaves or worker nodes are Task Trackers. The Job Tracker handles the runtime scheduling of MapReduce jobs and maintains information on each Task Tracker's load and available resources. The Database Connector is the interface between independent database systems residing on nodes in the cluster and Task Trackers. The Connector connects to the database, executes the SQL query and returns results as key-value pairs. The Catalog component maintains metadata about the databases, their location, replica locations, and data partitioning properties. The Data Loader component is responsible for globally repartitioning data on a given partition key upon loading and breaking apart single node data into multiple smaller partitions or chunks. The SMS planner extends the HiveQL translator and transforms SQL into MapReduce jobs that connect to tables stored as files in HDFS.

Jaql

Jaql is a query language which is designed for JavaScript Object Notation (JSON), a data format that is popular because of its simplicity and modeling flexibility. JSON is a simple, yet flexible way to represent data that ranges from flat data to semi-structured XML data. Jaql is primarily used to analyse large-scale semi-structured data. It is a functional, declarative query language that rewrites high-level queries, when appropriate, into a low-level query consisting of Map-Reduce jobs that are evaluated using the Apache Hadoop project. Core features include user extensibility and parallelism. Jaql consists of a scripting language and compiler, as well as a runtime component. It is able to process that either has no schema or has only a partial schema. However, Jaql can also exploit rigid schema information when it is available, for both type checking and improved performance. The following snippet in Listing 9 shows a sample JSON document.

Listing 9. A sample JSON document

```
[
  { id: 10,
    name: "Joe Smith",
    email: "JSmith@email.com",
    zip: 85150
  },
  { id: 20,
    name: "Alice Jones",
    email: "AJones@email.com",
    zip: 82116
  }
]
```

Jaql uses a very simple data model, a JDM (Jaql Data Model) value that is either an atom, an array, or a record. Most common atomic types are supported, including strings, numbers, nulls, and dates. Arrays and records are compound types that can be arbitrarily nested. In more detail, an array is an ordered collection of values and can be used to model data structures such as vectors, lists, sets, or bags. A record is an unordered collection of name-value pairs and can model structs, dictionaries, and maps. Despite its simplicity, JDM is very flexible. It allows Jaql to operate with a variety of different data representations for both input and output, including delimited text files, JSON files, binary files, Hardtop's Sequence Files, relational databases, key-value stores, or XML documents. Functions are first-class values in Jaql. They can be assigned to a variable and are high-order in that they can be passed as parameters or used as a return value. Functions are the key ingredient for reusability as any Jaql expression can be encapsulated in a function, and a function can be parameterized in powerful ways. Listing 10 is an example of a Jaql script that consists of a sequence of operators.

Listing 10. Jaql script that consists of a sequence of operators

```
import myrecord;
countFields = fn(records) (
records
-> transform myrecord::names($)
-> expand
-> group by fName = $ as occurrences
into { name: fName, num: count(occurrences) }
);
read(hdfs("docs.dat"))
-> countFields()
-> write(hdfs("fields.dat"));
```

The read operator loads raw data, in this case from the Hadoop Distributed File System (HDFS), and converts it into Jaql values. These values are processed by the countFields subflow, which extracts field names and computes their frequencies. Finally, the write operator stores the result back into HDFS. Table 2 describes the core expressions of the Jaql scripting language.

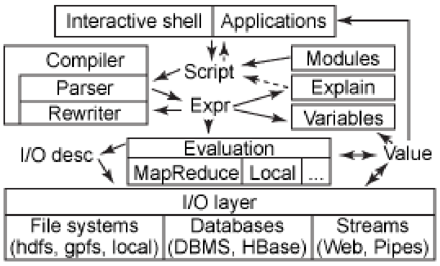
Table 2. Core expressions of the Jaql scripting language

Expression	Description

transform	The transform expression applies a function (or projection) to every element of an array to produce a new array.
expand	The expand expression is most often used to unnest its input array.
group by	Similar to GROUP BY in SQL, the group by expression in Jaql partitions its input on a grouping expression and applies an aggregation expression to each group.
Filter	The filter expression retains input values for which a specific predicate evaluates to true.
Join	The join expression supports equijoin of 2 or more inputs. All of the options for inner and outer joins are also supported.
Union	The union expression is a Jaql function that merges multiple input arrays into a single output array.
Control-flow	The two most commonly used control-flow expressions in Jaql are if-then-else and block expressions. The if-then-else expression is similar to conditional expressions found in most scripting and programming languages. A block establishes a local scope where zero or more local variables can be declared and the last statement provides the return value of the block.

At a high-level, the Jaql architecture depicted in Figure 4 is similar to most database systems.

Figure 4. The Jaql system architecture



Scripts are passed into the system from the interpreter or an application, compiled by the parser and rewrite engine, and either further explained or evaluated over data from the I/O layer. The storage layer is similar to a federated database. It provides an API to access data from different systems including local or distributed file systems (such as Hadoop's HDFS), database systems (such as DB2, Netezza, HBase), or from streamed sources like the web. Unlike federated databases, however, most of the accessed data is stored within the same cluster and the I/O API describes data partitioning, which enables parallelism with data affinity during evaluation. Jaql derives much of this flexibility from I/O API of Hadoop. It reads and writes many common file formats (such as delimited files, JSON text, and Hadoop Sequence files). Custom adapters are easily written to map a data set to or from the Jaql data model. The input can even simply be values that are constructed in the script itself. The Jaql interpreter evaluates the script locally on the computer that compiled the script, but spawns interpreters on remote nodes using MapReduce. The Jaql compiler automatically detects parallelization opportunities in a Jaql script and translates it into a set of MapReduce jobs.

Conclusion

MapReduce has emerged as a popular way to harness the power of large clusters of computers. Currently, MapReduce serves as a platform for a considerable amount of massive data analysis.

It allows programmers to think in a data-centric fashion where they can focus on applying transformations to sets of data records while the details of distributed execution and fault tolerance are transparently managed by the MapReduce framework. In practice, many programmers prefer to use high-level declarative (or SQL-like) languages to implement their parallelized large-scale data analysis jobs while leaving all of the execution optimization details to the backend engine.
In this article,you saw a state-of-the-art overview of the high-level declarative interfaces for the MapReduce framework. Minimize your programming burden with high-level language abstractions that enable the underlying systems to perform automatic optimizations at the execution level and improve the performance of the user tasks.

Resources

Learn

- Visit the Apache Hadoop Project website to learn about this software library that is a framework that allows the distributed processing of large data sets across clusters of computers using a simple programming model.
- Visit the Apache Hadoop Distributed File System website. Hadoop Distributed File System (HDFS) is he primary storage system used by Hadoop applications.
- Visit the Apache Pig Project website to learn about a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs.
- Visit the Apache Hive Project website. Hive is a data warehouse system for Hadoop that facilitates easy data summarization, ad-hoc queries, and the analysis of large datasets stored in Hadoop compatible file systems.
- Visit the Cloudera Apache Hadoop Distribution website to learn about this open source software package designed for sophisticated analysis and transformation of both structured and unstructured complex data.
- Visit the HadoopDB Project website to explore this architectural hybrid of MapReduce and DBMS technologies for analytical workloads.
- Visit the Hadapt - The Adaptive Analytical Platform website and check out the first big data platform to combine the benefits of Apache Hadoop and relational DBMS technology into a single system for apps that rely on multi-structured data analytics. Hadapt was designed for the cloud, and is optimized for virtualized environments.
- Visit the Aster Data Systems website and learn to tap into your data.
- Visit the Jaql website. The Jacl query language, designed for JavaScript Object Notation (JSON), is mostly used to analyze large-scale semi-structured data.
- Read the Hadoop MapReduce tutorial at Apache.org.
- Using MapReduce and load balancing on the cloud (Kirpal A. Venkatesh, Kishorekumar Neelamegam, and R. Revathy, developerWorks, July 2010): Learn to implement the Hadoop MapReduce framework in a cloud environment and to use virtual load balancing to improve the performance of both a single- and multiple-node system.
- Big Data Glossary (Pete Warden, O'Reilly Media, ISBN: 1449314597, September 2011: Explore the large number of new data tools available in a guide that describes 60 of the most recent innovations.
- Hadoop: The Definitive Guide (Tom White, O'Reilly Media, ISBN: 1449389732, September 2010: Discover how Apache Hadoop can unleash the power of your data.
- Programming Pig ((Alan Gates, O'Reilly Media, ISBN: 1449302645, September 2011): Read this learning tool and reference for Apache Pig, the open source engine for executing parallel data flows on Hadoop.
- HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads (Azza Abouzeid et al., Proceedings of the VLDB Endowment, 2(1), August 2009): This paper explores the feasibility of building a hybrid system that takes the best features from both technologies.

- Jaql: A Scripting Language for Large Scale Semi-Structured Data Analysis (Kevin S. Beyer et al., Proceedings of the VLDB Endowment, 4(9), September 2011.
- MapReduce: Simplified Data Processing on Large Clusters, Jeffrey Dean and Sanjay Ghemawat, OSDI, 2004
- SQL/MapReduce: A practical approach to self-describing, polymorphic, and parallelizable user-defined functions (Eric Friedman, Peter Pawlowski, and John Cieslewicz, Proceedings of the VLDB Endowment, 2(2), 2009): Read this paper that describes the motivation for this new approach to UDFs as well as the implementation within AsterData Systems' nCluster database.
- Building a HighLevel Dataflow System on top of MapReduce: The Pig Experience, Alan Gates et al., Proceedings of the VLDB Endowment, 2(2), August 2009): This paper describes the challenges faced in developing Pig, and reports performance comparisons between Pig execution and raw Map-Reduce execution.
- Pig latin: a not-so-foreign language for data processing (Christopher Olston et al., SIGMOD Conference, 2008)
- A comparison of approaches to large-scale data analysis(Andrew Pavlo et al., SIGMOD Conference, 2009)
- MapReduce and parallel DBMSs: friends or foes? (Michael Stonebraker et al., Commun. ACM 53(1), 2010)
- Hive - A Warehousing Solution Over a Map-Reduce Framework (Ashish Thusoo et al., Proceedings of the VLDB Endowment, 2(2), 2009)
- Data warehousing and analytics infrastructure at Facebook (Ashish Thusoo, Zheng Shao, Suresh Anthony, Dhruba Borthakur, Namit Jain, Joydeep Sen Sarma, Raghotham Murthy, and Hao Liu, SIGMOD Conference, 2010): This paper presents how Scribe, Hadoop, and Hive have come together and enabled Facebook to implement a data warehouse that stores more than 15PB of data (2.5PB after compression) and loads more than 60TB of new data (10TB after compression) every day.
- A Survey of Large Scale Data Management Approaches in Cloud Environments (Sherif Sakr, Anna Liu, Daniel M Batista, and Mohammad Alomari, Journal of IEEE Communications Surveys and Tutorials, 13(3), 2011): In this paper, find a comprehensive survey of numerous approaches and mechanisms of deploying data-intensive applications in the cloud which are gaining a lot of momentum in both research and industrial communities.
- IBM InfoSphere BigInsights Basic Edition offers a highly scalable and powerful analytics platform that can handle incredibly high data throughput rates that can range to millions of events or messages per second.
- In the Open Source area on developerWorks, find extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM products.
- On the developerWorks Java technology zone, find articles, tutorials, forums, and more about every aspect of Java programming.
- Stay current with developerWorks technical events and webcasts focused on a variety of IBM products and IT industry topics.
- Attend a free developerWorks Live! briefing to get up-to-speed quickly on IBM products and tools as well as IT industry trends.

- Listen to developerWorks podcasts for interesting interviews and discussions for software developers.
- Follow developerWorks on Twitter.
- Watch developerWorks demos that range from product installation and setup for beginners to advanced functionality for experienced developers.

Get products and technologies

- IBM InfoSphere BigInsights Basic Edition -- IBM's Hadoop distribution -- is an integrated, tested and pre-configured, no-charge download for anyone who wants to experiment with and learn about Hadoop.
- Get Hadoop 0.20.1 from Apache.org.
- Get Pig from Apache.org.
- Get Hadoop MapReduce.
- Get Hadoop HDFS.
- Access IBM trial software (available for download or on DVD) and innovate in your next open source development project using software especially for developers.

Discuss

- Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis. Help build the Real world open source group in the developerWorks community.

About the author

Sherif Sakr

Dr. Sherif Sakr is a Research Scientist in the Software Systems Group at National ICT Australia (NICTA), Sydney, Australia. He is also a Conjoint Lecturer in The School of Computer Science and Engineering (CSE) at University of New South Wales (UNSW). He received his PhD degree in Computer Science from Konstanz University, Germany in 2007. He received his BSc and MSc degree in Computer Science from the Information Systems department at the Faculty of Computers and Information in Cairo University, Egypt, in 2000 and 2003 respectively. In 2011, Dr. Sakr held a visiting research scientist position in the eXtreme Computing Group (XCG) at Microsoft Research, Redmond, USA. Dr. Sakr is a Cloudera certified developer for Apache Hadoop.

© Copyright IBM Corporation 2012
(www.ibm.com/legal/copytrade.shtml)
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)