MRQLProposal

Abstract

MRQL is a query processing and optimization system for large-scale, distributed data analysis, built on top of Apache Hadoop and Hama.

Proposal

MRQL (pronounced *miracle*) is a query processing and optimization system for large-scale, distributed data analysis. MRQL (the MapReduce Query Language) is an SQL-like query language for large-scale data analysis on a cluster of computers. The MRQL query processing system can evaluate MRQL queries in two modes: in MapReduce mode on top of Apache Hadoop or in Bulk Synchronous Parallel (BSP) mode on top of Apache Hama. The MRQL query language is powerful enough to express most common data analysis tasks over many forms of raw *in-situ* data, such as XML and JSON documents, binary files, and CSV documents. MRQL is more powerful than other current high-level MapReduce languages, such as Hive and PigLatin, since it can operate on more complex data and supports more powerful query constructs, thus eliminating the need for using explicit MapReduce code. With MRQL, users will be able to express complex data analysis tasks, such as PageRank, k-means clustering, matrix factorization, etc, using SQL-like queries exclusively, while the MRQL query processing system will be able to compile these queries to efficient Java code.

Background

The initial code was developed at the University of Texas of Arlington (UTA) by a research team, led by Leonidas Fegaras. The software was first released in May 2011. The original goal of this project was to build a query processing system that translates SQL-like data analysis queries to efficient workflows of MapReduce jobs. A design goal was to use HDFS as the physical storage layer, without any indexing, data partitioning, or data normalization, and to use Hadoop (without extensions) as the run-time engine. The motivation behind this work was to build a platform to test new ideas on query processing and optimization techniques applicable to the MapReduce framework.

A year ago, MRQL was extended to run on Hama. The motivation for this extension was that Hadoop MapReduce jobs were required to read their input and write their output on HDFS. This simplifies reliability and fault tolerance but it imposes a high overhead to complex MapReduce workflows and graph algorithms, such as PageRank, which require repetitive jobs. In addition, Hadoop does not preserve data in memory across consecutive MapReduce jobs. This restriction requires to read data at every step, even when the data is constant. BSP, on the other hand, does not suffer from this restriction, and, under certain circumstances, allows complex repetitive algorithms to run entirely in the collective memory of a cluster. Thus, the goal was to be able to run the same MRQL queries in both modes, MapReduce and BSP, without modifying the queries: If there are enough resources available, and low latency and speed are more important than resilience, queries may run in BSP mode; otherwise, the same queries may run in MapReduce mode. BSP evaluation was found to be a good choice when fault tolerance is not critical, data (both input and intermediate) can fit in the cluster memory, and data processing requires complex/repetitive steps.

The research results of this ongoing work have already been published in conferences (WebDB'11, EDBT'12, and DataCloud'12) and the authors have already received positive feedback from researchers in academia and industry who were attending these conferences.

Rationale

- MRQL will be the first general-purpose, SQL-like query language for data analysis based on BSP.
- Currently, many programmers prefer to code their MapReduce applications in a higher-level query language, rather than an algorithmic language. For instance, Pig is used for 60% of Yahoo! MapReduce jobs, while Hive is used for 90% of Facebook MapReduce jobs. This, we believe, will also be the trend for BSP applications, because, even though, in principle, the BSP model is very simple to understand, it is hard to develop, optimize, and maintain non-trivial BSP applications coded in a general-purpose programming language. Currently, there is no widely acceptable declarative BSP query language, although there are a few special-purpose BSP systems for graph analysis, such as Google Pregel and Apache Giraph, for machine learning, such as BSML, and for scientific data analysis.
- MRQL can capture many complex data analysis algorithms in declarative form.
- Existing MapReduce query languages, such as HiveQL and PigLatin, provide a limited syntax for operating on data collections, in the form of relational joins and group-bys. Because of these limitations, these languages enable users to plug-in custom MapReduce scripts into their queries for those jobs that cannot be declaratively coded in their query language. This nullies the benets of using a declarative query language and may result to suboptimal, error-prone, and hard-to-maintain code. More importantly, these languages are inappropriate for complex scientic applications and graph analysis, because they do not directly support iteration or recursion in declarative form and are not able to handle complex, nested scientic data, which are often semi-structured. Furthermore, current MapReduce query processors apply traditional query optimization techniques that may be suboptimal in a MapReduce or BSP environment.
- The MRQL design is modular, with pluggable distributed processing back-ends, query languages, and data formats.
- MRQL aims to be both powerful and adaptable. Although Hadoop is currently the most popular framework for large-scale data analysis, there are a few alternatives that are currently shaping form, including frameworks based on BSP (eg, Giraph, Pregel, Hama), MPI (eg, OpenMPI), etc. MRQL was designed in such a way so that it will be easy to support other distributed processing frameworks in the future. As an evidence of this claim, the MRQL processor required only 2K extra lines of Java code to support BSP evaluation.

Initial Goals

Some current goals include:

- apply MRQL to graph analysis problems, such as k-means clustering and PageRank
- apply MRQL to large-scale scientific analysis (develop general optimization techniques that can apply to matrix multiplication, matrix factorization, etc)
- process additional data formats, such as Avro, and column-based stores, such as HBase

- map MRQL to additional distributed processing frameworks, such as Spark and OpenMPI
- extend the front-end to process more query languages, such as standard SQL, SPARQL, XQuery, and PigLatin

Current Status

The current MRQL release (version 0.8.10) is a beta release. It is built on top of Hadoop and Hama (no extensions are needed). It currently works on Hadoop up to 1.0.4 (but not on Yarn yet) and Hama 0.5.0. It has only been tested on a small cluster of 20 nodes (80 cores).

Meritocracy

The initial MRQL code base was developed by Leonidas Fegaras in May 2011, and was continuously improved throughout the years. We will reach out other potential contributors through open forums. We plan to do everything possible to encourage an environment that supports a meritocracy, where contributors will extend their privileges based on their contribution. MRQL's modular design will facilitate the strategic extensions to various modules, such as adding a standard-SQL interface, introducing new optimization techniques, etc.

Community

The interest in open-source query processing systems for analyzing large datasets has been steadily increased in the last few years. Related Apache projects have already attracted a very large community from both academia and industry. We expect that MRQL will also establish an active community. Several researchers from both academia and industry who are interested in using our code have already contacted us.

Core Developers

The initial core developer was Leonidas Fegaras, who wrote the majority of the code. He is an associate professor at UTA, with interests in cloud computing, databases, web technologies, and functional programming. He has an extensive knowledge and working experience in building complex query processing systems for databases, and compilers for functional and algorithmic programming languages.

Alignment

MRQL is built on top of two Apache projects: Hadoop and Hama. We have plans to incorporate other products from the Hadoop ecosystem, such as Avro and HBase. MRQL can serve as a testbed for fine-tuning and evaluating the performance of the Apache Hama system. Finally, the MRQL query language and processor can be used by Apache Drill as a pluggable query language.

Known Risks

Orphaned Products

The initial committer is from academia, which may be a risk, since research in academia is publication-driven, rather than product-driven. It happens very often in academic research, when a project becomes outdated and doesn't produce publishable results, to be abandoned in favor of new cutting-edge projects. We do not believe that this will be the case for MRQL for the years to come, because it can be adapted to support new query languages, new optimization techniques, and new distributed back-ends, thus sustaining enough research interest. Another risk is that, when graduate students who write code graduate, they may leave their work undocumented and unfinished. We will strive to gain enough momentum to recruit additional committers from industry in order to eliminate these risks.

Inexperience with Open Source

The initial developer has been involved with various projects whose source code has been released under open source license, but he has no prior experience on contributing to open-source projects. With the guidance from other more experienced committers and participants, we expect that the meritocracy rules will have a positive influence on this project.

Homogeneous Developers

The initial committer comes from academia. However, given the interest we have seen in the project, we expect the diversity to improve in the near future.

Reliance on Salaried Developers

Currently, the MRQL code was developed on the committer's volunteer time. In the future, UTA graduate students who will do some of the coding may be supported by UTA and funding agencies, such as NSF.

Relationships with Other Apache Products

MRQL has some overlapping functionality with Hive and Tajo, which are Data Warehouse systems for Hadoop, and with Drill, which is an interactive data analysis system that can process nested data. MRQL has a more powerful data model, in which any form of nested data, such as XML and JSON, can be defined as a user-defined datatype. More importantly, complex data analysis tasks, such as PageRank, k-means clustering, and matrix multiplication and factorization, can be expressed as short SQL-like queries, while the MRQL system is able to evaluate these queries efficiently. Furthermore, the MRQL system can run these queries in BSP mode, in addition to MapReduce mode, thus achieving low latency and speed, which are also Drill's goals. Nevertheless, we will welcome and encourage any help from these projects and we will be eager to make contributions to these projects too.

An Excessive Fascination with the Apache Brand

The Apache brand is likely to help us find contributors and reach out to the open-source community. Nevertheless, since MRQL depends on Apache projects (Hadoop and Hama), it makes sense to have our software available as part of this ecosystem.

Documentation

Information about MRQL can be found at MRQL: an Optimization Framework for Map-Reduce Queries

Initial Source

The initial MRQL code has been released as part of a research project developed at the University of Texas at Arlington under the Apache 2.0 license for the past two years. The source code is currently hosted on GitHub at:
https://github.com/fegaras/mrql. MRQL's release artifact would consist of a single tarball of packaging and test code.

External Dependencies

The MRQL source code is already licensed under the Apache License, Version 2.0. MRQL uses JLine which is distributed under the BSD license.

Cryptography

Not applicable.

Required Resources

Mailing Lists

- mrql-private
- mrql-dev
- mrql-user

Subversion Directory

- Git is the preferred source control system: git://git.apache.org/mrql

Issue Tracking

- A JIRA issue tracker, MRQL

Wiki

- Moinmoin wiki, http://wiki.apache.org/mrql

Initial Committers

- Leonidas Fegaras <fegaras AT cse DOT uta DOT edu>
- Upa Gupta <upa.gupta AT mavs DOT uta DOT edu>
- Edward J. Yoon <edwardyoon AT apache DOT org>
- Maqsood Alam <maqsoodalam AT hotmail DOT com>
- John Hope <john.hope AT oracle DOT com>
- Mark Wall <mark.wall AT oracle DOT com>
- Kuassi Mensah <kuassi.mensah AT oracle DOT com>
- Ambreesh Khanna <ambreesh.khanna AT oracle DOT com>

- Karthik Kambatla <kasha AT cloudera DOT com>

Affiliations

- Leonidas Fegaras (University of Texas at Arlington)
- Upa Gupta (University of Texas at Arlington)
- Edward J. Yoon (Oracle corp)
- Maqsood Alam (Oracle corp)
- John Hope (Oracle corp)
- Mark Wall (Oracle corp)
- Kuassi Mensah (Oracle corp)
- Ambreesh Khanna (Oracle corp)
- Karthik Kambatla (Cloudera)

Sponsors

Champion

- Anthony Elder <antelder AT apache DOT org>

Nominated Mentors

- Alan Cabrera <adc AT apache DOT org>
- Anthony Elder <antelder AT apache DOT org>
- Alex Karasulu <akarasulu AT apache DOT org>
- Mohammad Nour <mnour AT apache DOT org>

Sponsoring Entity

Incubator PMC