Diplomarbeit

Exposé

Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät II
Institut für Informatik

eingereicht von: Max Schultze

Betreuer: Prof. Dr. Ulf Leser

1.1 Abstract

During the last years the amount of unstructured or semi-structured data generated through social networks, business applications and scientific research has grown rapidly. Typical data base management systems were no longer or only insufficiently able to process all that data in a timely fashion, which lead to the development of new Big Data management systems. Google's MapReduce programming paradigm [DG04] was a recent suggestion on how to break down computations in smaller tasks and run them in parallel on big clusters. Since then multiple systems have been developed to extend the initial MapReduce approach creating environments for the application of complex analysis pipelines on very large data sets.

AsterixDB [AST] and Apache Flink [AFL] are two systems that were developed around the same time in different research projects. AsterixDB runs on the Hyracks data-parallel platform [BCG+11] and has its own query language AQL. Apache Flink was first developed as the Stratosphere project [STR] and later on included into the Apache software foundation. Flink runs on Hadoop's YARN [YRN] and is embraced by a Java API and a Scala API.

The goal of this diploma thesis is to develop a module that translates AQL queries into executable Scala code for Apache Flink and to compare the two systems with each other running certain queries in their respective environments, yet on the same hardware.

1.2 AsterixDB

This section gives a very brief overview about the AsterixDB system as well as its query language AQL. AsterixDB is a Big Data management system developed by researchers at UC Irvine, UC Riverside and UC San Diego. It was initiated as the NSF-sponsored Asterix project in 2009 and was designed to develop new technologies for ingesting, storing, managing, indexing, querying and analyzing vast quantities of semi-structured information [AST]. The first system beta was released in June 2013 and a first full description of the system was published at the Very Large Data Bases Conference in september 2014 [AAA+14].

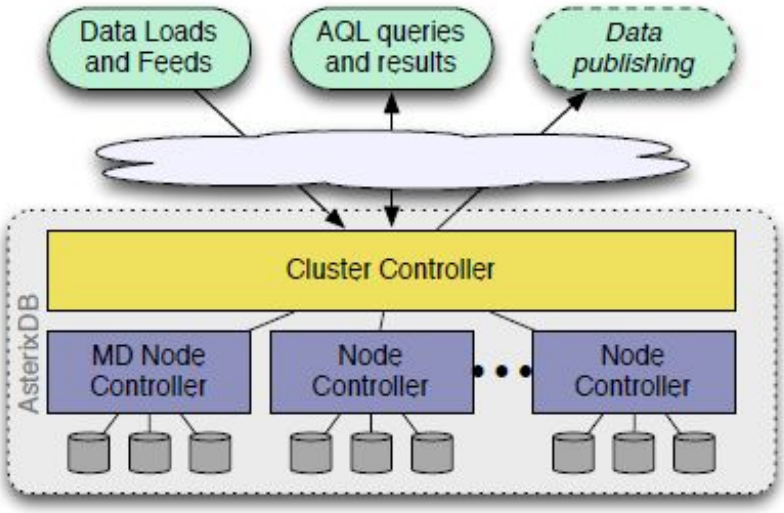

Figure 1.1: AsterixDB System Overview [AAA+14]

Figure 1.1 shows an overview of AsterixDB and its logical architecture. Data is inserted via loading, continuous feeds and/or insertion queries. It is accessed via queries through AsterixDB's own query language AQL. The logical entry point for user queries is the cluster controller. Herein lies the Asterix client interface, which is a HTTP-based API that receives the queries and returns the results either synchronously or asynchronously. The cluster controller also contains the AQL compiler, that translates AQL queries into job descriptions, as well as the job executor that distributes those job descriptions to the node controllers. The data flow execution engine is Hyracks [BCG+11]. Once AQL queries are compiled into DAGs of operators and connectors they are submitted as a Hyracks job. Hyracks than manages the computational resources of the underlying cluster as well as the data access and storage.

Asterix Query Language - AQL

AQL is AsterixDB's own query language to access and return data. In the contrary to other Big Data systems like Flink [AFL] and Hive [AHV] that use SQL like query languages, AQL is based on XQuery to have an easier access to non-flat data with a lack of a priori schemas as well as to have a good approach for nested queries. AQL queries are based on the FLWOR structure that was taken from XQuery. FLWOR stands for `for-let-where-order by-return` which are the five most common clauses of the AQL query syntax. `for` in AQL is like a `FROM` clause in SQL whereas `return` is like the `SELECT` clause.

Query 1

```
1: for $ds in dataset d return $ds;
```

Query 1 shows the simplest query that uses the `for` and `return` clauses to return all instances of a target AsterixDB dataset *d*.

Query 2

```
1: for $user in dataset Users
2: where $user.latest-login >= datetime('2015-01-01T00:00:00')
3: order by $user.language asc
4: return $user;
```

The `where` clause and the `order by` clause are similar to the ones used in SQL. Query 2 shows a simple filter condition to return all users from a dataset *User* that have been active since the beginning of 2015 sorted by their languages.

Query 3 from [AS2]

```
1: for $user in dataset Users
2: let $messages :=
3:  for $message in dataset Messages
4:  where $message.author-id = $user.id
5:  return $message.message
6: return
7:  {
8:  "uname":  $user.name,
9:  "messages":  $messages
10:  };
```

The `let` clause is similar to the `WITH` clause used in SQL. It is an easy way to construct nested queries. Query 3 shows an example where for each user in a dataset *Users* all the messages from a dataset *Messages* that he is assigned as an author for are returned together with his user name.

1.3 Apache Flink

Flink is a data processing system incubated in the Apache Software Foundation. It was originally developed as the DFG-funded Stratosphere project at TU Berlin, HU Berlin and HPI Potsdam to build a fast, reliable, expressive, easy to use and scalable alternative to Hadoop's MapReduce component. A first Stratosphere beta was released in 2012 and the Apache Flink system was most recently approved as a top-level Apache project.

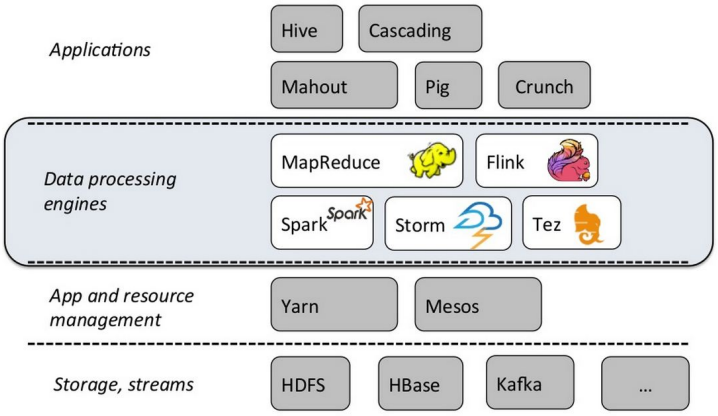

Figure 1.2: Flink in the Open Source Landscape [AFL]

Figure 1.2 shows where to locate Flink in the open source landscape. It is an alternative to Hadoop's MapReduce data processing engine that runs on top of Hadoop's YARN [YRN]. YARN manages the resources and nodes of the underlying Hadoop Distributed File System (HDFS). Flink supports a Java API and a Scala API. Previous versions of Stratosphere also had its own query language Meteor [HRL+12].

Scala

Scala is a functional, object-oriented, extendable and strongly typed programming language. It fits in seamlessly with Java and .NET-platforms as it runs on a JVM or .NET VM such that already existing frameworks and libraries can be used. A short list of programming constructs will be shown to give an overview of what can be done with Scala. Following that there is an example within the Apache Flink context.

```
val city = "Pforzheim"
var plz = 01234
```

There are two types of variables in Scala which are both shown above. `var` defines a variable that can be changed whereas `val` defines a fixed constant. As said before Scala is strongly typed. The type can be given explicitly or be implied from the initial value.

```
class MyClassTest {
  val theName = "Hans";
}
```

Classes in Scala are syntactically similar to Java classes and defined as shown above.

```
object MyApplication {
       def main (args: Array[String]) {
         println("Hello World")
         // do something...
       }
}
```

In Scala there are no static variables or methods. Instead `object` classifies a class with only a single instance, a singleton. E.g. an application can be realized as an object as shown above.

```
def plus (a: Int, b: Int, c: Int) = a + b + c
```

Functions in Scala are defined with the keyword `def`. It is not necessary to write a return statement as the value of the last expression is automatically returned. A `id:type` syntax is used for the definition as shown above.

Figure 1.3 shows the Flink wordcount example written in Scala which was taken from the Apache Flink Scala API documentation [AFL]. A given text is split into its single words that are each counted separate as one. Afterwards it is grouped by those words and summed up over each group. The example shows how functions can be concatenated in Scala and the result of their computations are stored in a single variable.

1.4 Comparison

The main goal of this thesis is to compare AsterixDB and Flink to each other. This section gives a brief overview about the main differences between the two systems.

As seen in Figure 1.4 the two systems are build on a similar architecture stack as they both have a data processing engine, a resource management layer and a storage layer. While Flink runs Hadoop's Yarn and uses the underlying HDFS, AsterixDB has its own resource management platform that already includes the data storage. The data processing engines are the main components Flink and AsterixDB. On top of that AsterixDB has its own query language AQL. In the contraire Flink only has a build in Java API and a Scala API that can be used to write and run queries. The red dashed arrow symbolizes the translation module that shall be written for this thesis to connect AQL to the Scala API of Flink.

Scala is a functional and object-oriented programming language. The Scala API of Flink also supports transformational functions like Map, Reduce and several joins for processed data sets. AQL does not support external User Defined Functions (UDFs), but only has a fixed set of commands. Because of that the aim is to fully translate AQL into Scala.

1.5 Thesis

The goal of this diploma thesis is to develop a module that translates AQL queries into executable Scala code for Apache Flink and to compare the two systems with each other running certain

```scala
object WordCountJob {
  def main(args: Array[String]) {

    // set up the execution environment
    val env = ExecutionEnvironment.getExecutionEnvironment

    // get input data
    val text = env.fromElements("To be, or not to be,--that is the question:--",
      "Whether 'tis nobler in the mind to suffer", "The slings and arrows of outrageous fortune",
      "Or to take arms against a sea of troubles,")

    val counts = text.flatMap { _.toLowerCase.split("\\W+") }
      .map { (_, 1) }
      .groupBy(0)
      .sum(1)

    // emit result
    counts.print()

    // execute program
    env.execute("WordCount Example")
  }
}
```

Figure 1.3: Scala Wordcount example in Flink [AFL]

queries in their respective environments, yet on the same hardware.

Procedure

To achieve the goal several steps have to be taken. First of all the two systems need to be setup in a local environment for development and testing. After that the existing interfaces need to be analysed to figure out the detailed specification for the translation module. This is partially done already and will be finished soon. To write the module it is necessary to parse AQL queries and analyse them semantically to automatically generate Scala code for the Flink system. Once this is done first queries will be run locally on both systems to get a first comparison. The last step will be to setup both systems separate on a big cluster environment and run representative queries on big underlying data to get a comparative look at their performances.

Evaluation

To evaluate the work of this thesis and to compare the two systems to each other several queries will be run. First of all the functionality of the two systems will be tested with complex queries that reach over the possibilities AQL has to offer. Second in order will be scalability, i.e. running those queries on different sized data sets as well as on cluster environments of different size. These tests should be comparable between the two systems.

On top of that some more queries will be run that favor one of the systems. First of all some queries with huge selectivity will be run. That should favor AsterixDB, as those will make use of AsterixDB's Indexes whereas Flink does not have any of those. To get use cases that reach into the specialties of each system a set of TPC-H queries will be run to represent classic relational processing, as well as some queries on XML data. The XML request should favor AsterixDB as it is specialized on handling AQL queries which are based on XQuery.

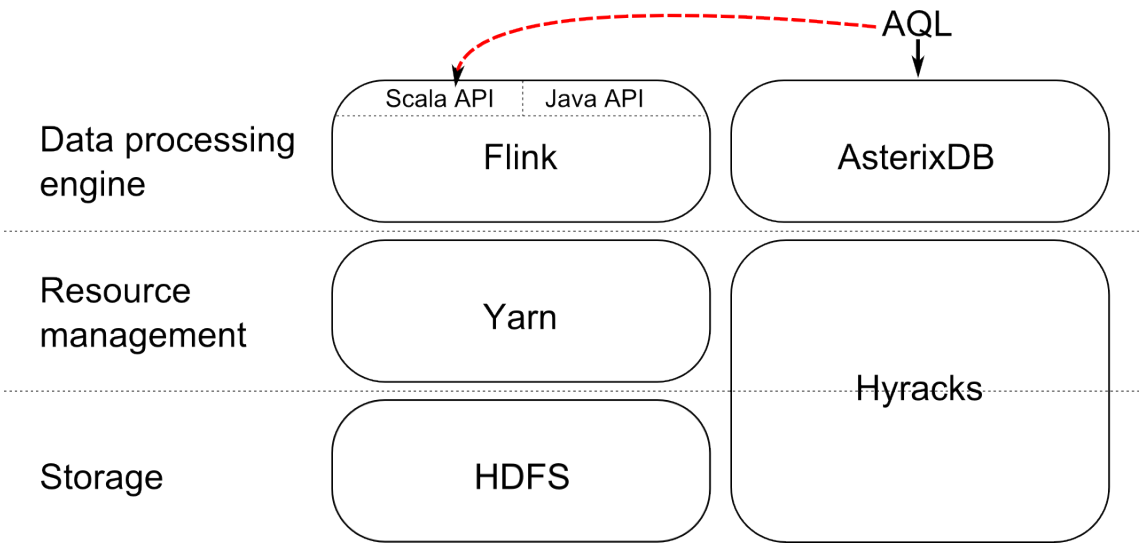

Figure 1.4: Comparison of Flink and AsterixDB

In addition some data inserts will be run on AsterixDB, that will not be translatable to Flink. Updates will not be tested as they are not the typical use case for those two systems.

Bibliography

[AAA+14] S. Alsubaiee, Y. Altowim, H. Altwaijry, A. Behm, V. R. Borkar, Y. Bu, M. J. Carey, I. Cetindil, M. Cheelangi, K. Faraaz, E. Gabrielova, R. Grover, Z. Heilbron, Y.-S. Kim, C. Li, G. Li, J. M. Ok, N. Onose, P. Pirzadeh, V. J. Tsotras, R. Vernica, J. Wen, and T. Westmann. Asterixdb: A scalable, open source BDMS. In *VLDB*, 2014.

[AFL] `http://flink.incubator.apache.org/`.

[AHV] `http://hive.apache.org/`.

[AS2] `https://asterixdb.ics.uci.edu/documentation/aql/manual.html`.

[AST] `https://asterixdb.ics.uci.edu/`.

[BCG+11] V. Borkar, M. Carey, R. Grover, N. Onose, and R. Vernica. Hyracks: A Flexible and Extensible Foundation for Data-intensive Computing. In *ICDE*, page 1151 1162, 2011.

[DG04] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, pages 137-150, 2004.

[HRL+12] Arvid Heise, Astrid Rheinländer, Marcus Leich, Ulf Leser, and Felix Naumann. Meteor/sopremo: An extensible query language and operator model. In *Workshop on End-to-end Management of Big Data, Istanbul, Turkey*, 2012.

[STR] `http://www.stratosphere.eu/`.

[YRN] `http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html`.