an Adaptable Scalable Analytics Platform

Compute and Data Engine Modeling

Deliverable no.: 3.1

28/02/2015

ASAP is funded by the European Commission DG-INFSO
Seventh Framework Programmed, Contract no.: 619706

Deliverable Title	Compute and Data Engine Modeling
Filename	D3.1.docx
Author(s)	K. Doka, N. Papailiou, C. Mantas, D.Tsoumakos
Date	28-02-1015

Start of the project: 01/03/2014
Duration: 3 years
Project coordinator organization: FORTH

Deliverable title: Compute and Data Engine Modeling
Deliverable no.: 3.1

Due date of deliverable: 28/02/2015
Actual submission date: 28/02/2015

Dissemination Level

X	PU	Public
	PP	Restricted to other programme participants (including the Commission Services)
	RE	Restricted to a group specified by the consortium (including the Commission Services)
	CO	Confidential, only for members of the consortium (including the Commission Services)

Deliverable status version control

Version	Date	Author
0.1	16/01/2015	K. Doka
0.2	10/02/2015	K. Doka, N. Papailiou, C. Mantas
1.0	15/02/2015	K. Doka, D. Tsoumakos
2.0	27/02/2015	K. Doka

Abstract

This deliverable presents the design, architecture and methods in the current state of the Intelligent Resource Scheduling (IReS) platform, which constitutes a core component of ASAP and is responsible to i) model operator performance according to different engines and their resources and ii) adaptively decide on which operator version to run based on the optimization policy and the available engines. In this first period, attention has been given towards defining the methodology and tools in order to model the performance and costs of the different compute and data engines.

Keywords

Workflow optimization, modeling, profiling, benchmarking, compute engine, data store

Table of Contents

List of Figures

List of Tables

List of Abbreviations

FS	File System
IaaS	Infrastructure as a Service
IReS	Intelligent Resource Scheduler
ML	Machine Learning
M/R	MapReduce
NLP	Natural Language Processing
RDBMS	Relational Database Management System
SPJ	Select-Project-Join
VM	Virtual Machine

1 Introduction

1.1 IReS Overview

Big data analytics have become a necessity in the majority of industries [1] , taking the lead in risk assessment, business process effectiveness, market analysis, etc. [98] [2] . Enabling engineers, analytics experts and scientists alike to tap the potential of vast amounts of business-critical data has grown increasingly important. Such data analysis demands a high degree of parallelism, in both storage and computation: Business datacenters host huge volumes of data, stored over large numbers of nodes with multiple storage devices and process them using thousands or millions of cores.

The demand for near-real-time, data-driven analytics has given rise to diverse execution engines and data stores that target specific data and computation types (e.g., [11] [12] [13] [21] [69] [96] , etc.). With the advent of virtualized computing, these platforms are offered as a service by many IaaS providers, enabling a very wide deployment range. For some of those engines there exist approaches in the literature that manage to optimize their performance (e.g., [40] [41]) by automatically tuning a number of configuration parameters. Yet, these schemes work on a single engine (mainly the Hadoop ecosystem), merely considering specific data formats and query/analytics task types.

However, modern workflows have become increasingly long and complex [66] . Specifically, workflows may include multiple data types (e.g., relational, key-value, graph, etc.) generated from different resources. They are also executed under varying constraints and policies (e.g., optimize for performance or cost, require different fault-tolerance degrees, etc.). Finally, workflow operators can be greatly diverse, from simple Select-Project-Join (SPJ) and data movement to complex NLP-, graph- or custom business-related operations. There currently exists no single platform that can optimize for this complexity [32] .

Sensing this trend, cloud software companies now offer software distributions in pre-cooked VM images or as a service. These distributions incorporate different processing frameworks, data stores and libraries to alleviate the burden of multiple installations and configurations (e.g., [28] [46] [47] [82]). Yet, such multi-engine environments lack a meta-scheduler that could automatically match tasks to the right engine(s) according to multiple criteria, deploy and run them without manual intervention. A recent attempt along this line [5] [4] focuses more on lower-level database operators, emphasizing on their automatic translation from/to specific engines via an XML-based language. Yet, this is a proprietary tool with limited applicability and extension possibilities for the community.

To address multi-engine optimization, the ASAP project employs the **Intelligent Multi-Engine Resource Scheduler (IReS)**, an integrated, open source platform for managing, executing and monitoring complex analytics workflows. Its goal is to provide adaptive, cost-based and customizable resource management of the diverse execution and storage engines available. Moreover, since the area of high performance analytics advances daily, ASAP's goal is to present a repeatable process that will allow easy inclusion of different technologies, if so desired. IReS includes a modeling framework that constantly evaluates the cost, quality and performance of data and computational resources in order to decide on the most advantageous store, indexing and execution pattern available.

To this direction, our system will be able to handle existing open-source execution models (e.g., Map-Reduce, Bulk Synchronous Parallel) as well as state-of-the-art centralized and distributed storage engines (RDBMSs, NoSQL, distributed file-systems, etc.) in order to have a broad applicability and increased performance gains. IReS plans to optimize workflows consisting of tasks that range from simple group-by, aggregation or complex joins between different data sources to machine-learning tasks and queries on graph data in combination with relational data. In the current implementation, the system bases its operation on the following elements:

- A profiling and modeling engine that learns operator output per different engine configuration. Outputs are collected via budget-constraint executed benchmarks. The learned models are stored and utilized for the planning phase of the workflow.
- A JSON-based metadata language that describes operators in abstract and instantiated forms, enabling search and matching of operators that perform a similar task in the planning phase.
- A decision-making and enforcing process that chooses among different equivalent workflow execution plans (i.e., on different engines, resulting in equivalent output) based on cost or performance and schedules the execution.

The resulting optimization is enhanced by any optimization effort within a single engine. IReS is a fully open-source platform that targets both low (e.g., join, sort, etc.) as well as high level (e.g., machine learning, graph processing) operators, treating them as black boxes. The generic profiling/modeling method it relies upon allows for easy addition of new operators and engines.

1.2 Purpose of the Document

The purpose of deliverable 3.1 is to describe the design, architecture and methods in the current state of the IReS platform. The basic abilities of the IReS platform will be to i) model operator performance according to different engines and their resources and ii) adaptively decide on which operator version to run based on the optimization policy and the available engines. In this first period, attention has been given towards defining the methodology and tools in order to model the performance and costs of the different compute and data engines.

1.3 Document Structure

D3.1 is structured as follows:

- Chapter 2 contains an overview of the state of the art tools for big data analysis. The most commonly used tools are presented and their basic characteristics are explained.
- Chapter 3 describes the architecture of the IReS platform, explaining the role of each component and presents the main system workflows, namely the modeling and the planning and execution workflows.

- Chapter 5 presents in detail the methodology that will be followed to model the various runtimes and data stores in terms of performance, cost or any user-defined metric. Moreover, the chapter contains an experimental evaluation of the proposed methodology.
- Chapter 5 concludes the deliverable.

2 Big Data Analytics Technologies and Platforms

This chapter reviews existing technologies and platforms that deal with Big Data analytics. This review aims at providing a thorough analysis on the characteristics of the different runtimes and data stores so that an informed selection on the ones to be supported by ASAP can be made. The list is by no means exhaustive, but rather representative. Prominent examples for each of the pertinent system categories are included. It contains the engines and data stores that are either already being used by WIND and IMR in their current analytics workflows, or could possibly be considered for usage in the ASAP use cases, namely Web Analytics and Telecommunication Analytics, as described in deliverables D8.2 and D9.2 respectively.

2.1 Compute Models and Engines

What all distributed execution engines have in common is the ability to manipulate data loaded from and stored to a distributed file system. In particular, many of the popular systems use the Hadoop File System [44] as their basis (see Section 2.1.1). The basic assumption behind all these engines is that both storage and computation are delegated to a large number of clustered nodes in a manner that ensures progress and fault tolerance. Data is usually read from or written to the distributed file system but the memory of the nodes in the cluster or local storage can be used for storing intermediate results during the course of the computation. To this end, the most prevalent programming models are **Map-Reduce** and **Bulk Synchronous Parallel model.**

Map-Reduce [34] is a programming model for processing and generating large data sets on a cluster. It uses a specific parallel and distributed algorithmic paradigm based on those two functions. It was originally developed by Google for the purpose of efficiently indexing the Web graph and computing PageRank. It is loosely based on the on the map and reduce functions used in functional programming but applies that paradigm to distributed computation. The user describes the computation as a series of Map and Reduce operations over key-value data. The map function generates intermediate key-value results that the reduce function merges based on the intermediate key into the output of the operation. The framework manages the parallelization and distribution of the execution, the data transfers and communications in a fault tolerant manner.

The M-R model has been proved largely successful for big data manipulations and has a number of implementations. The prevalent implementation of Map-Reduce is the one used in the Hadoop Framework.

The Bulk Synchronous Parallel (BSP) model [100] was originally conceived as a bridge model between programming and hardware models for parallel computation, sitting between software and hardware. The Model actors were defined as follows:

1. A number of components, each performing processing and/or memory functions; (i.e. processors)

2. A router that delivers messages point to point between pairs of components; (i.e. network)

3. Facilities for synchronizing all, or a subset of, the components at regular intervals of L time units where L is the periodicity parameter. (i.e. barrier)

The distributed computation in BSB is a sequence of supersteps. In each one of those supersteps the processors are allocated with tasks consisting of a mix of:

a) local computations

b) message transmissions (to other processors)

c) message arrivals (from others)

After L expires a global check is performed, in order to determine whether the all processors have completed the superstep. If that is so the machine will proceed to the next superstep.

While BSP was proposed as a model for parallel processing, it is a good fit for distributed systems too. As such it was recently adopted by Google in the design of Pregel [67] . Its main advantage over the Map-Reduce model is that BSP is superior in handling graph-based and iterative computations which are common in ML algorithms.

The following subsections present the most popular implementations of the aforementioned models. Table 1 summarizes the main characteristics of the reviewed compute engines.

2.1.1 Hadoop Framework

Apache Hadoop is, mainly, a Map-Reduce based framework for big data manipulation. It comprises of a collection of open-source tools written in Java. Hadoop consists of 4 base modules and an ecosystem of tools built on top of those. The basic modules of Hadoop are:

- Hadoop File System [44] : a distributed, user-space file system written in Java.
- Hadoop YARN [43] Resource Manager: YARN (Yet Another Resource Negotiator) is a cluster manager technology that tries to allow for multiple heterogeneous data processing engines to handle data and resources in a single platform. It provides consistency, security and data governance tools for the applications and libraries running on the Hadoop framework.
- MapReduce: A Map-Reduce execution engine, based on YARN. Most of the tools of the Hadoop ecosystem are built on top of MapReduce and either try to simplify it or extend its basic functionality.

Apart from these core modules most users use a number of open source tools by the Apache Project that use that basic framework. These include but are not limited to: Pig, Hive, HBase, Mahout and Spark.

2.1.2 Spark

Hadoop MapReduce is the de facto standard when it comes to big data analysis. However, two of its main design attributes make it unfit for interactive processing of simple queries. The first one has to do with the fact that the user has to implement the queries in low level Java following the rather restrictive map-combine-reduce paradigm. And while this can be worked around with libraries such as Hive, which offer an abstraction for M/R, one cannot change the fact that MapReduce was designed for batch processing. This means that its main focus is processing datasets of unstructured data that are much larger than the main memory of the cluster. Because of that, MapReduce uses the disk-based file system to store intermediate results and is thus bound to the slower performance characteristics of secondary storage. It follows that interactive queries are not a good fit for it. Another example of analysis that is not a fast to implement in MapReduce is iterative processing. Many machine learning and graph-based algorithms fall in that category. Since data needs to touch disk twice, after each map and reduce phase, running many iterations of a function over the data means incurring the cost of as many disk spills.

However, none of those limitations is theoretically bound to datasets that can fit in the RAM available to the cluster. This is the motivation behind the development of Spark [21] . Spark is a much better fit for Hadoop for in-memory datasets. In those kinds of processing it offers execution times that are orders of magnitude faster [39] [88] It also natively supports streaming data and SQL analysis. It is worth mentioning that Spark is available alongside MapReduce in most commercial Hadoop distributions. Spark is most commonly run over Hadoop but can also run standalone or, over Mesos. Data can be read from HDFS, the local file system, HBase, Cassandra and S3. The user can implement the data processing in Scala, Java, Python or SQL. It is also possible to issue interactive queries via the Scala or Python shells using a library of readily available operators. Despite being relatively new, a large number of inter operable data analytics libraries are built on top Spark and it increasingly preferred for big-data Machine Learning implementations (See Chapter. 2.3.2)

2.1.3 Stratosphere/Flink

Stratosphere [24] was originally a research project with the ambitious goal of developing the next-generation Big Data Analytics Platform and addressing the shortcomings of Map-Reduce implementations. Since then it became an Apache "incubator" Project called Flink [9] As a concept, Flink is similar to Spark in the sense that it is optimized for in-memory scalable data analytics and can provide realtime results as well as true streaming operation. Flink provides APIs in Scala and Java with Python and SQL counterparts under development. It also supports operations specific to Graph Processing with Spargel [92] API, implementing the BSP programming model. Flink takes a more declarative approach to describing operations and aims to optimize the query execution in a manner similar to traditional DBMSs. It thus supports join algorithms that are automatically optimized as well as delta iterations processing. Flink also aims to allow the definition, optimization and execution of DAGs of operators.

2.1.4 HAMA

Apache Hama [12] (short for "Hadoop Matrix") is a pure BSP processing engine following a design principle similar to Google Pregel [67] . It is thus most effective with streaming data processing and graph, network and matrix algorithms requiring iterative computations. For such computations, HAMA promises a drastically improved performance compared to Hadoop and Mahout [76] .

2.1.5 Stream and Realtime Data Processing

Stream processing frameworks serve the goal of handling data that arrive in real time as a stream of events. Common examples of such processing are aggregations, continuous queries, and pattern detection. This type of processing is relatively new and has been made a necessity by the APIs of Twitter and Facebook as well as large sensor networks which can provide a high Velocity (2.2) stream of events. There are a number of dedicated systems as well a number of Distributed Stream processing libraries that are built on top of existing computational engines like Hadoop and Spark.

2.1.5.1 Apache Samza

Apache Samza [104] is a distributed Stream processing framework built on top of Apache Hadoop YARN and Apache Kafka [103]. Samza (as well as Kafka) is developed primarily by LinkedIn in order to provide elastic and fault-tolerant processing of its realtime feeds.

2.1.5.2 Apache Spark Streaming:

Spark's Streaming [77] is a library lets you use the same Spark code for batch processing and stream data. It uses the same Java and Scala API as batch jobs, provides fault tolerance and is tightly integrated with the rest of the Spark libraries.

2.1.5.3 Druid

Originally developed by Metamarket but later open-sourced, Druid [105] is also aimed at the storage and processing of realtime data. It assumes an append-only input and can provide efficient, lock-free, low latency ingestion and processing. It uses bitmap indexes for ad-hoc multi-dimensional filtering. Data can be kept in-memory or on disk in a column-oriented storage approach.

Runtime Engines	Type of Processing	Good for	Popularity	Potential use in ASAP
Hadoop	Batch Disk based	Batch processing of large datasets	Facebook, Yahoo!, Amazon, Google, HP, eBay, etc.	Web, Telecom
Spark	Batch, In-Memory	Iterative jobs (ML, graph processing), Interactive analytics	Yahoo!, Intel, Many startups	Web, Telecom

Stratosphere/Flink	Graph, BSP model	Graph, iterative and streaming analytics	Still in Research	Web, Telecom
Hama	BSP model	Iterative computations (graph, ML)	Korean telecom, some universities...	Web, Telecom
Samza/Spark Streaming/ Druid	Streaming	Interactive processing of append-only streams	LinkedIn, Metamarket, Netflix, Yahoo!	Telecom

Table 1 Summary of runtime characteristics and potential use in ASAP

2.2 DataStores

The basic layer of every data analytics system is the data storage system. Conventionally any organization would use a centralized Relational OLTP and OLAP system to store structured data on a local filesystem.

With the rise of "big data" the storage needs web-scale datasets became beyond the scope of any centralized system. The challenges of handling data grew in at least 3 aspects [65] :

- **Volume:** The sheer size of the datasets in the cloud era makes it impossible to store and process effectively in centralized systems
- **Variety:** Social applications, SaaS offerings and sensor data, logs are one of the few things that cannot fit in the relational model.
- **Velocity:** The rate at which data is produced is an ever-growing challenge.

Gradually, the need for a distributed file system became prevalent and the requirement for ACID assurances was relaxed in order to adapt to the new demands.
A new range of distributed database systems was developed that abandoned full SQL support and compromised hard consistency goals for the sake of scalability to a large number of machines, high availability and partition tolerance. The most commonly used systems in this category are open-source.
We can roughly classify most of the available systems under the following categories (and combinations thereof).

Distributed File Systems: Such block storage systems are usually ran on user-level and sacrifice POSIX compatibility in favor of the ability to provide high combined throughput, fault tolerance and ability to store large files despite residing on commodity hardware.

Relational Databases: Traditional DBMSs are still widely used and have evolved to support a shared-nothing infrastructure and provide replication, sharding and scalability while still providing ACID compliance and support for SQL.

NoSQL Databases: Such distributed systems usually utilize a different data model than that used in Relational Databases, do not fully support the full SQL standards and operations and provide eventual consistency [101] . This category covers a large range of storage system models like: key-value, document, key-map, graph, column, time-series.

Proprietary Parallel Databases: Following the progress of CPU and network architectures, many proprietary DBMS vendors chose to try and offer scalability for their products by parallelizing their storage and execution to clustered resources. Parallel DBMS's offer improved processing and I/O performance by using multiple CPUs and disks.

Column-Oriented databases: These types of systems forgo the common row-based storage for the stored data in favor of columnar storage. While this approach is not new, it was reintroduced recently with C-Store [95] (which later took shape as a product in the form of Vertica DBMS – see 0). Storing data in columns can be significantly faster in scenarios where aggregation operations and range queries are dominant. Another important value of column stores is that, due to its nature, it can allow a significant decrease in stored data when compression is used in the storage layer. As a result, column stores are a good fit for data warehousing and log storage/processing. A prominent example of columnar storage is MonetDB [69] . A column-based approach is also used in new NoSQL systems like Dremel [68] and its open-source clone, Apache Drill [8] , as well as Impala [59] . In those systems it was argued [109] to provide a significant increase in performance and decrease query latency.

The following subsections review the most prevalent data stores chosen from the aforementioned categories. Table 2 summarizes their characteristics and their possible use within ASAP.

2.2.1 HDFS

HDFS (short for "the HaDoop File System") is an open-source distributed file system developed as the basic layer of the Hadoop Framework. It is developed in Java, runs in userspace and shares some design principles with GFS [38] .

The basic assumptions it is based on are the use of commodity hardware, scalability to a high number of nodes (each offering local computation and storage), redundancy for fault tolerance and the use of files that are typically large (GBs to TBs), immutable and sequentially read and written. HDFS is a part of the "core" Hadoop framework and is used by nearly all Apache projects (HBase, Hive, Mahout, Pig, Spark, Tez, ZooKeeper).

Data in HDFS can be accessed from the native Java API or via any other language using the Thrift Protocol [22] . Clients for HDFS exist in C++, Perl, Python, Ruby, Erlang, Haskell, C# and PHP. There is also a command line interface and it can be browsed through the HDFS-UI web app over HTTP.

2.2.2 Cassandra

Apache Cassandra [7] is an Open-Source distributed NoSQL database written in Java that was originally built by Facebook by using Google's BigTable [27] data model and Amazon's Dynamo [35] architecture. Its purpose is handling large amounts of structured data spread out across a high number of commodity servers with no single point of failure. It is possible for a single cluster to spawn multiple data-centers across geographic areas while offering high performance and availability as well as failure resilience.

Cassandra uses its own query language, CQL [30] , which is very similar to SQL but offers a subset of its functionalities. Moreover Cassandra is not ACID compliant but does support durable transactions and a tunable level of consistency. There is an API available through Thrift [22] and a native one in JAVA.

Its main advantages are excellent scalability to hundreds of cluster nodes, high write throughput and read efficiency.

Cassandra is the industry leader of NoSQL Databases and is used by large organizations such as Adobe, Comcast, eBay, Rackspace, Netflix, Twitter, and Cisco.

2.2.3 Hbase

Apache HBase is another non-relational distributed database developed in Java, which follows BigTable's design [27] . HBase is a member of the Hadoop ecosystem and thus uses HDFS for storage in the same way BigTable uses GFS. HBase harvests the fault tolerance characteristics of HDFS.

The best use case for HBase is storing sparse Data in large quantities with versioning. HBase offers high availability and random, real-time access over very large (e.g. web graph) tables. One of the main advantages of HBase is its tight integration with Hadoop MapReduce and another one is its near-lineal scalability to a large number of nodes. HBase uses its own query language that is not as rich as SQL or CQL and there are Thrift, REST and native java APIs availalbe.

HBase has a wide industry adoption and is the second used NoSQL database. Companies that are using it include Adobe, Facebook, Twitter, and Yahoo!.

2.2.4 ElasticSearch

ElasticSearch [52] is a standalone, open-source search server written in Java. Its core free-text search functionality is provided by Lucene [49] , but Elastic search wraps this functionality in a simpler, API and provides sharding, clustering and replication of the Lucene indices. ES promises real-time text queries and analytics, multi-tenancy, and high availability. The client API is REST-ful with a JSON format but there are also clients for Perl, PHP, Python and Ruby while there is also compatibility with Hadoop. ElasticSearch is a part of ElasticSearch ELK Stack [51] for creating a search server. ELK also includes tools for encryption, managing time-based data, visualization of the results and monitoring of the cluster.

2.2.5 MySQL

MySQL [71] , the most popular server-side relational DBMS, is developed in C and is open source. MySQL provides extensive SQL support, ACID compliance, and transactions. With MySQL Cluster [54] , it also offers the ability for automatic sharding and distributed operation with no single point of failure and good scalability characteristics. Support is extensive with APIs available in virtually any programming language.

2.2.6 PostgreSQL

PosgreSQL [77] is an open source object-relational DBMS developed in C. It provides near full support for SQL, durable transaction and is ACID compliant. PostgreSQL has the ability of failure resilience through master-slave replication and can also serve reads from slaves. There are APIs available in C, C++, Java, Python, Perl, Tcl and ECPG. PostgreSQL has a strong market share as far as databases are concerned and is often preferred over MySQL due to some of the more advanced features it offers. MonetDB

2.2.7 MonetDB

MonetDB [69] is a column-oriented DBMS. It was created during the late 90's as a part of a research project and introduced many novel optimizations (e.g. for CPU caches). Its current, open-source form was introduced with version 4 in 2004. MonetDB provides extensive SQL support and has a JDBC client for Java, as well as clients for PHP, Python, Perl, Ruby and R. MonetDB does not have a significant commercial user base but is used in a number of academic projects.

2.2.8 Proprietary analytics PDMS

Vertica (HP): An implementation based on a grid-based, column-oriented storage approach was Vertica's database which is now a part of the HP Vertica "Dragline" [57] platform. This vertical DBMS approach, was spawned from the C-Store [95] research system, is mostly used for data warehousing and runs on grids of Linux commodity servers. It is also available as a hosted DBMS provisioned by and running on the Amazon Elastic Compute Cloud. It also integrates with Hadoop. Apart from Vertica's analytics database, HP's platform includes Autonomy's unstructured data analytics engine, IDOL 10.5 [48] . The platform combines structured and unstructured data into a unified workflow.

Greenplum: Pivotal Greenplum Database is another shared-nothing, MPP implementation of an OLAP system that is a part of Pivotal's cloud services portfolio with a focus on business intelligence and analytical processing. It adopts a hybrid row/column approach and promises lineal parallelization and also offers native Map-Reduce in its parallel engine, support for HDFS as well as support for PL/Java, optimized C, and Java functions.

Data Store	Type of Data	Good for	Popularity	Potential use in ASAP
HDFS	DFS	sequential reads and writes of large files in a batch manner	universal	Web, Telecom

HBase	NoSQL, column-family oriented	Write throughput Strong consistency	Facebook, Adobe, Yahoo!, HP, etc.	Web, Telecom
Cassandra	NoSQL, column-family oriented	Write throughput, multi-datacenter configuration, error resilience	Facebook, Yahoo!, Adobe, AoL, BestBuy, ,Ebay, FedEx, GitHub +many more	Web, Telecom
ElasticSearch	Indexed data	Search engine capabilities (secondary indices)	Wikimedia, Mozilla, Foursquare, etc.	Web
MySQL	RDMS	Centralized database storage with transactions	Industry leader for relational storage	Telecom
PostgreSQL	ORDMS	Replicated database storage with transactions	Skype, IMDB, LAMP, Apple, SourceForce	Telecom
MonetDB	RDBS	Append-only workload, aggregation operations	Research	Telecom
Vertica/Greenplum	P-RDMS	Interactive SQL queries, OLAP	commercial	Web

Table 2 Summary of data store characteristics and potential use in ASAP

2.3 Libraries and Big Data operations

In the last decade, with the rise of Cloud Computing and social networking, managing large, web-scales datasets became ever more important for creating novel products and applications. Big data analysis does not provide an excellent basis for commercial products, but is also crucial for developing business intelligence. With the prevalence of the Hadoop ecosystem a number of software solutions have emerged for covering that need. Broadly speaking we could split the various solutions in two groups. On the one hand there are the SQL-like query processing tools and on the other hand the Machine Learning libraries. A summary of the various tools can be found in Table 3 Summary of machine learning and query processing libraries and applications

2.3.1 Query processing

There has been a lot of industry and research effort dedicated on executing SQL or similar queries over Big-Data Analytics engines. Pig, Hive, Sawzall, Jaql and Tenzing fall in the category of offering SQL processing on top of Hadoop. Those systems implement

SQL or SQL-like functionalities over the Map-Reduce model and are widely used by large enterprises in the industry. Spark SQL is a similar effort for Spark. These Query processing systems serve the purpose of allowing a user to express queries in a language that resembles SQL which is the most prevalent query language in the industry.

2.3.1.1 Hive

Hive [14] is a framework allowing the execution SQL-like queries on data stored in HDFS. It is is an interface layer for Hadoop converting the user-provided query in "HiveQL" language into Map-Reduce jobs. Hive is often used as a user-friendly tool for data warehousing applications. Hive also uses a centrally located repository for storing metadata which is called "metastore". Hive was originally developed by Facebook but is now an Apache open-source project widely used in the industry from companies like Netflix.

2.3.1.2 Pig

Pig is system similar to Hive in the sense that it is an interface layer between a high level query language and Hadoop M-R jobs. The language used by Pig is called "Pig Latin" and is extendable by User Defined Functions which can be written in Java, Python, JavaScript, Ruby or Groovy. In contrast with Hive, Pig allows for more data transformations and provides more control over the executed workflow. It is not uncommon for Pig and Hive to be used alongside despite the fact that their functionality is largely similar. Pig was originally developed by Yahoo! but is now an Apache open-source project. It is also widely used in the industry.

2.3.1.3 SparkSQL [93]

Spark SQL was developed as a unified query language for the Spark framework. It incorporates ETL (Extract-Transform-and-Load), SQL-like and Hive relational queries and Map-Reduce jobs in a single language. It also allows native Scala queries as well as access to externally stored data from existing RDDs [81] , Parquet [18] and JSON files. Spark has integrated APIs in Python, Scala and Java. It is also tightly integrated with Hive since it reuses the Hive frontend and metastore, providing full compatibility with existing Hive data, queries, and User Defined Functions.

2.3.2 Machine Learning

As the Hadoop ecosystem grows more mature, it is increasingly used for advanced analytics. In particular, a good fit for its capabilities seems to be running Machine Learning operations on large text datasets. There exists ample research on this field and various projects have evolved to open-source implementation that are widely used. While Mahout, running on top of MapReduce is the most prominent one with an extensive set of available operations, MLlib over the faster Spark framework quickly catches up and seems likely to replace it. Newer tools, based on the BSP paradigm, also show a lot of promise for managing graph data.

2.3.2.1 Mahout

Apache Mahout [16] is an open source machine learning library, written in Java and built on top of Hadoop MapReduce. It offers distributed implementations for common algorithms, primarily for clustering, classification and batch based collaborative filtering. Mahout can be run in a single node but all the operations will follow the Map-Reduce paradigm. Mahout is in active development but is already established in the Hadoop ecosystem.

2.3.2.2 MLlib

MLlib [20] is library for Spark that offers implementations for common ML algorithms, tests and data generators. The available algorithms fall in the categories of binary classification, regression, clustering and collaborative filtering. Similarly to Mahout/MapReduce, MLlib uses the underlying engine of Spark just extending its available operators. MLlib is tightly integrated with Spark and is in active development. Like Spark itself MLlib is gaining traction over Mahout and MapReduce since it can provide more interactive operations.

2.3.2.3 GraphX

Spark's GraphX [10] is a distributed analytics system for graph data. Following a similar motivation as Pregel [67] , GraphX expresses graph computations in the Spark framework. Its goal is to "unify data-parallel and graph-parallel analytics". As most implementations for Spark, GraphX focuses on in-memory computations and is efficient for iterative computations. The GraphX library implementation is a topic of active research [104] .

2.3.2.4 WEKA

The Waikato Environment for Knowledge Analysis is an open source suite of ML algorithms implemented in Java [103] . WEKA can be used standalone or as API in Java code. It has a very extended collection of tools for arithmetic and text processing. Namely, it offers algorithms in the areas of pre-processing, classification, regression, clustering, association rules, and visualization. WEKA is the default tool used by both academia and industry for centralized ML processing.

2.3.2.5 OpenNLP

Apache OpenNLP [17] is a toolkit for natural text ML processing. It features implementations for tokenization, sentence segmentation, part-of-speech tagging, named entity extraction, chunking, parsing, and co-reference resolution. It is often used as a building block for larger and more complex NLP projects

Runtime libraries	Type of Processing	Good for	Popularity	Potential use in ASAP
MLlib	distributed	Machine Learning library over Spark	Companies that use Spark	Web, Telecom
Mahout	distributed	Machine Learning lib over Hadoop/Spark	AOL, Foursquare, LinkedIn	Web, Telecom
WEKA	centralized	Machine Learning software written in Java	Very popular	Web, Telecom
Hive	distributed	Query processing and data warehousing on top of Hadoop Mapreduce	Netflix	Web, Telecom
Spark SQL	distributed	SQL over Spark	Companies that use Spark	Web, Telecom
Pig	distributed	Query processing over Hadoop MapReduce	Twitter, LinkedIn, AOL	Web, Telecom

Table 3 Summary of machine learning and query processing libraries and applications

2.4 Workflow Management platforms

The field of optimizing the execution of workflows over multiple execution engines is a relatively new field of research. There has already been a promising work in the form of HFMS but most available tools for workflow management are focused on a small set of scientific tasks executed locally.

HFMS [89] builds on top of previous work on multi-engine execution optimization [90] . On those platforms optimization occurs on two levels. Namely for single engine and multi-engine execution. The actual engines used are the Hadoop MapReduce distributed execution engine and a centralized PostgreSQL database. We believe that ASAP will leverage knowledge from those approaches and generalize it in a more all-around framework for multi-engine workflow optimization.

Pegasus [50] is a workflow management System that allows users to easily express multi-step computational tasks. The workflow it accepts as input is in the form of a DAG, where the tasks are represented as nodes and task dependencies as edges. This DAG is expressed in an XML file. Pegasus offers APIs for Java, Python and Perl, offers support for MySQL, PostgreSQL, Oracle and Microsoft databases and can run on Amazon EC2 infrastructure.

Taverna [56] is another management system for scientific workflows. It can that help user specify the group of tasks that constitute a scientific pipeline and orchestrate their execution by using several underlying tools. However Taverna's compatibility is limited to database available via JDBC [53] , some proprietary tools and R programming language operations and does not support any distributed execution engines.

3 The IReS Architecture

IReS focuses on highly efficient and user-customizable execution of analytics tasks (or workflows). This is made possible through the transparent modeling, monitoring and scheduling that involves different execution engines and storage technologies. Our system is able to execute all types of analytics workflows by adaptively choosing to execute each sub-part of the workflow to a (possibly different) deployed engine. The IReS platform assigns sub-tasks to the most advantageous technology(-ies) available and ensures resource and dataflow scheduling in order to enhance performance: If a single engine is used, enhancement will be achieved through optimized resource allocation and elasticity modeling (e.g., execute on more VMs, or on smaller cluster with larger main memory, etc.); if multiple ones are required, enhancements will relate both to single-engine optimization and to workflow management that decides what is the best execution workflow and data-flow (e.g., execute sub-task 1 first, intermediate results should be stored on a NoSQL engine and then sub tasks 2 and 3 run in parallel, that write the final results to HDFS files).

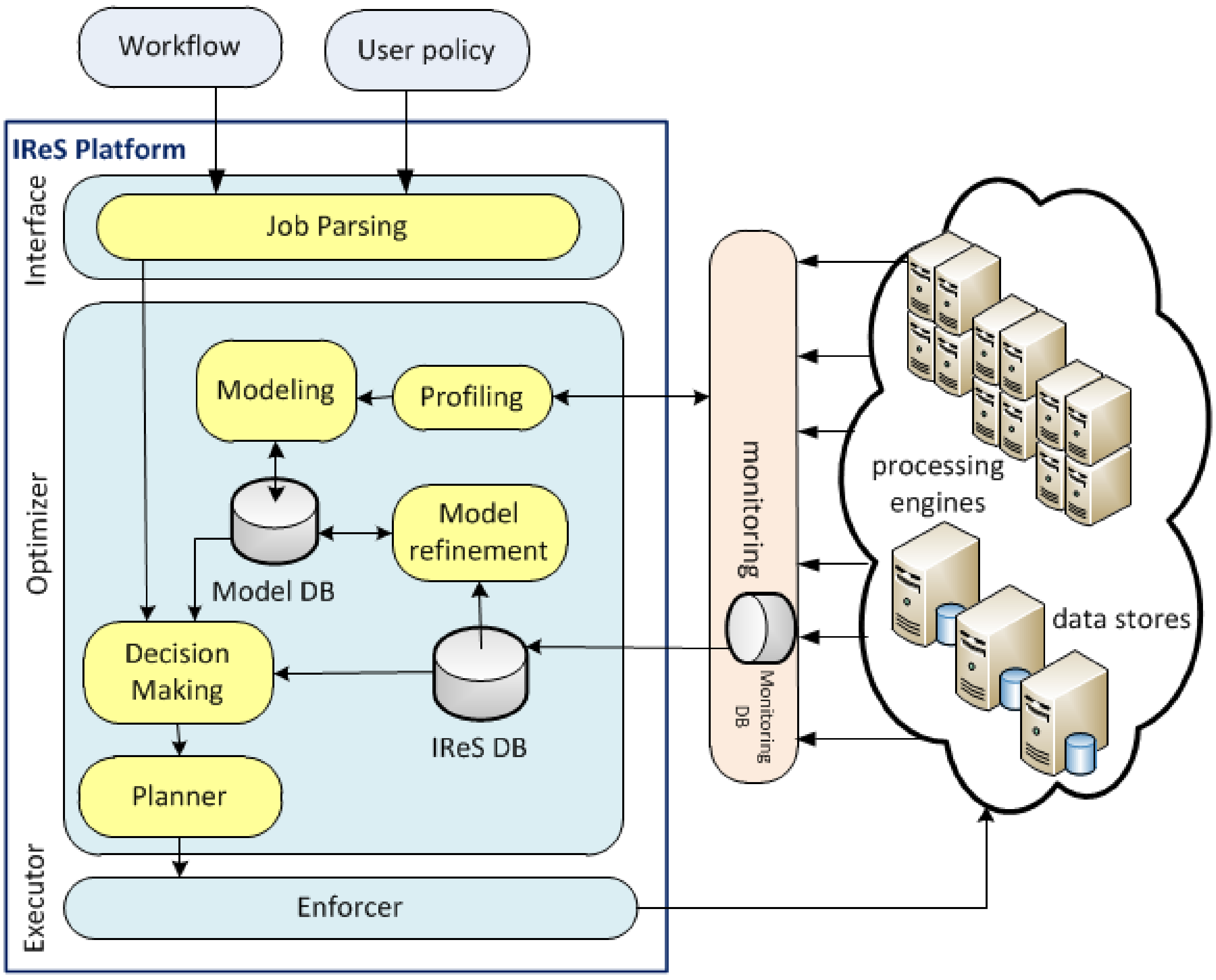


Figure 1 Architecture of the IReS platform

The central notion behind the IReS platform is to create detailed models of the costs and performance characteristics of various analytics operations over multiple execution engines. These models will then be used to intelligently match the user optimization policy with the available execution engines.

The architecture of the IReS platform is depicted in Figure 1. IReS comprises of three layers, the **interface**, the **optimizer** and the **executor** layer. In the following we describe in more detail the role, functionality and internals of these layers and the most important modules of the platform.

3.1 The Interface Layer

The **interface** layer is the upper layer of the IReS platform, responsible for communicating with the workflow description language (defined in WP5) and the application UI in order to receive the input that is necessary for its operations. It comprises of the *job parsing module*, which extracts execution artifacts such as operators, data, their dependencies and accompanying metadata from the user-defined wokflow. Moreover, it validates the user-defined policy. All this information must be robustly identified, structured in a dependency graph and stored. The job parsing module is more thoroughly described in the following.

3.1.1 Job parsing module

This module takes as input the user-defined workflow, formulated in a dependency graph format and expressed in the workflow description language designed and implemented in T5.1. This language, which allows for various levels of abstraction, is described in detail in D5.1. Moreover, the module takes as input the user optimization parameters, which could translate to performance, cost, availability, etc. All this information is gathered and concisely described using a metadata framework that will facilitate the process of identifying the optimal workflow execution plan.

The main challenge of defining such a workflow description metadata framework is the fact that it requires to be abstract at the user level. The user should be able to describe the data and operators that compound her workflow in a way as abstract as she desires. The IReS planner and workflow scheduler need to remove that abstraction, find all the alternative ways of materializing the workflow and select the most beneficial, according to the user-defined policy.

Our proposed metadata framework describes **data** and **operators**. Data and operators can be either **abstract** or **materialized**. Abstract are the operators and datasets that are described partially or at a high level by the user when composing her workflow whereas materialized are the actual operator implementations and existing datasets, either provided by the user or residing in a repository.

Both data and operators need to be accompanied by a set of metadata, i.e., properties that describe them and can be used to match

(a) abstract operators to materialized ones and
(b) data to operators.

Such properties include input data types and parameters of operators, location of data objects or operator invocation scripts, data schemata, implementation details, engines etc. The metadata defined for each object have a generic tree format (JSON). To avoid restricting the user and allow for extensibility, the first levels of the metadata tree are predefined but users can add their ad-hoc subtrees to define their custom data or operators. Moreover, some fields (mostly the ones related to the operator and data requirements, located under the constraints field) are *compulsory* while the rest (i.e., known cost models, statistics etc.) are *optional*. Materialized data and operators need to have all their compulsory fields filled in with information. Abstract data and operators do not adhere to this rule. Apart from having empty fields, they can also support regular expressions (e.g., the * symbol under a field means that the abstract object matches materialized ones with any value of that field).

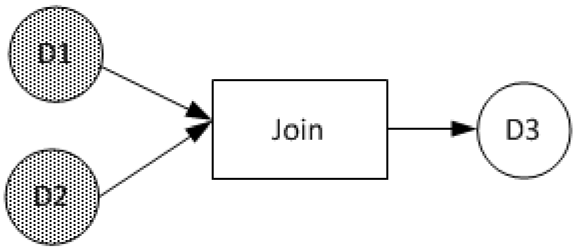


Figure 2 Workflow example: Simple join operation between two datasets

To describe the proposed language we will use the following example: The user has 2 materialized datasets, one stored in HBase (D1) and one stored in MySQL (D2) and wants to perform a join operation on them. Figure 2 depicts the abstract workflow given by the user. Circles represent data objects while rectangles represent operators. Shaded shapes designate materialized objects.

The metadata descriptions of D1 (designated as hbase_dataset) and D2 (sql_dataset) are depicted in Figure 3 and Figure 4 respectively. Since the datasets are materialized, all compulsory fields are populated. These fields include information about the data itself, such as the attributes of the dataset and their types, as well as engine specific information (which attribute is the HBase key, where is the dataset located, etc.). Under the optional optimization field, we place additional information that assists in the optimization of the workflow, in our case the dataset size and the number of records (unique_keys).

```
{"hbase_dataset": {
  "constrains":{
      "data_info": {
        "attributes": [
          {"customer_name":      {"type": "ByteWritable"}},
          {"customer_purchases": {"type": "List<ByteWritable>"}}]},
      "engine": {
        "DB.NoSQL.HBase": {
          "key:":      "customer_id",
          "value":     "customer_purchases",
          "location": "83.212.118.9"}}},
  "optimization":{
    "size":"1TB",
    "unique_keys":"13000000"
  }}}
```

Figure 3 Metadata description of dataset D1

```
{"sql_dataset": {
  "constrains":{
      "data_info": {
        "attributes": [
          {"customer_id":    {"type": "Varchar(15)"}},
          {"customer_name": {"type": "Varchar(15)"}}]},
      "engine": {
        "DB.Relational.MySQL": {
          "key:":      "customer_id",
          "location": "83.212.118.9" }}},
  "optimization":{
    "size":"1GB",
    "unique_keys":"1000000"
  }}}
```

Figure 4 Metadata description of dataset D2

Let us consider the join operator on a single attribute of the above example. In its abstract form, the joinOp operator (see Figure 5) needs only define the minimum compulsory fields under the constraints key, namely the two input parameters, the condition under which they are joined, an output parameter and the description of the operation to performed.

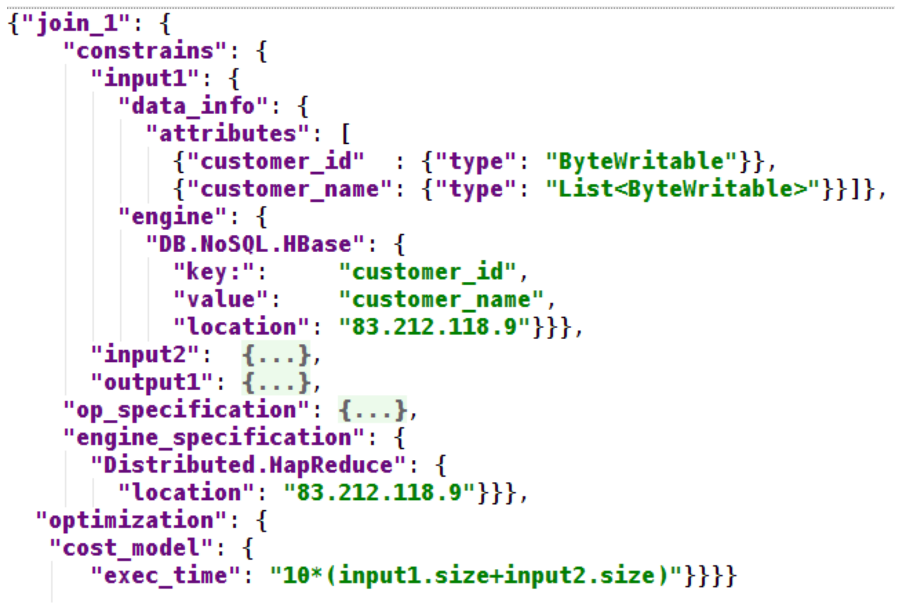
```
{"joinOp": {
  "constrains": {
    "input1": {
      "data_info": {
        "attributes": ["attr1", "attr2"]}},
    "input2": {
      "data_info": {
        "attributes": ["attr1", "attr2"]}},
    "output1": {
      "data_info": {
        "attributes": ["attr1", "attr2"]}},
    "op_specification": {
      "algorithm": {
        "join": {
          "join_condition":
            "input1.attr1=input2.attr1"}}}}}}
```

Figure 5 Metadata description of the abstract join operator

Each of the input parameters and the output are abstract data_info objects with two attributes: "attr1" represents the field of the join predicate while "attr2" represents the second available field in each data_info object. The op_specification field of this operator specifies its operation, a single join algorithm, and defines the join condition (in this case an inner join).

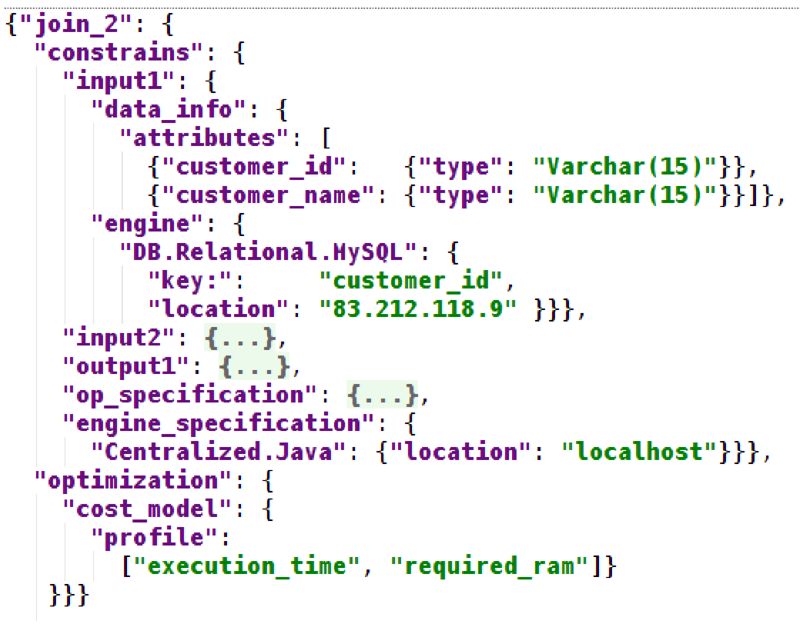
In short, the abstract join operator defines a format that any join operator implementing the specific functionality needs to follow.

The materialized operators include, on top of that, all information required in order to perform the operation on an execution engine. In join_1 (see Figure 6), the operator executes the join over Hadoop; it thus includes Hadoop-specific information about the input, output and the engine. The inputs and output in this case have specific attribute types and an engine specification (under engine) containing the location of the data and information about their structure. The operator itself also has an engine specification (engine_specification) indicating its execution location. The example in Figure 7 describes join_2, which joins an HBase and a relational table and outputs the result to HDFS. It runs as a local java process.



```
{"join_1": {
    "constrains": {
      "input1": {
        "data_info": {
          "attributes": [
            {"customer_id"   : {"type": "ByteWritable"}},
            {"customer_name": {"type": "List<ByteWritable>"}}]},
        "engine": {
          "DB.NoSQL.HBase": {
            "key:":      "customer_id",
            "value":     "customer_name",
            "location": "83.212.118.9"}}},
      "input2":  {...},
      "output1": {...},
    "op_specification": {...},
    "engine_specification": {
      "Distributed.HapReduce": {
        "location": "83.212.118.9"}}},
  "optimization": {
   "cost_model": {
      "exec_time": "10*(input1.size+input2.size)"}}}}
```

Figure 6 Metadata descriptions of the first materialized join operator



```
{"join_2": {
  "constrains": {
    "input1": {
      "data_info": {
        "attributes": [
          {"customer_id":    {"type": "Varchar(15)"}},
          {"customer_name": {"type": "Varchar(15)"}}]},
      "engine": {
        "DB.Relational.HySQL": {
          "key:":      "customer_id",
          "location": "83.212.118.9" }}},
    "input2": {...},
    "output1": {...},
    "op_specification": {...},
    "engine_specification": {
      "Centralized.Java": {"location": "localhost"}}},
  "optimization": {
    "cost_model": {
      "profile":
        ["execution_time", "required_ram"]}
   }}}
```

Figure 7 Metadata descriptions of the second materialized join operator

To discover the actual implementations that comply with the description of an abstract operator provided by the user, we employ a tree matching algorithm to make sure that all metadata constraints are met, i.e., compulsory fields are consistent. This is performed by the decision making module, described subsequently (Section 3.2.3). In our example, both join_1 and join_2 match join and are thus considered when constructing the optimized execution plan.

Apart from the compulsory fields, which are necessary for the matching of abstract to materialized operators, the metadata descriptions of the materialized joins both contain the optional optimization field, which holds additional information that assists in the optimization of the workflow. In the case of join_1, a cost function is provided by the developer of the operator while for join_2 the platform is instructed to create one by profiling over specific metrics (execution time and required ram).

3.2 The Optimizer Layer

The **optimizer** layer is the layer that performs all the necessary actions to optimize the execution of an analytics workflow with respect to the policy provided by the user. The core component of the optimizer is the *decision making module*, which determines in real-time the optimal execution plan. This entails deciding where each subtask is to be run, under what amount of resources provisioned, the plan for moving data to/from their current locations and between runtimes if more than one is chosen and defining the output destinations. Such a decision must rely on the characteristics of the analytics task in hand and the models of all possible engines. These models are produced by the *modeling module* and stored in the model database. The initial model of an engine results from profiling and benchmarking operations over them in an offline manner, through the *profiling module*. This module directly interacts with the pool of physical resources and the monitoring layer in-between. While the workflow is being executed, the initial models are refined in an online manner by the *model refinement module*, using monitoring information of the actual run. Such monitoring information is kept in the IReS DB and is utilized by the decision making module as well, to enable real-time, dynamic adjustments of the execution plan based on current knowledge.

In the following, each module is described in greater detail.

3.2.1 Modeling Module

This module is responsible for constructing models on a per operator-engine combination basis. The relevant literature review [83] [23] [109] has revealed that models already exist for a very limited number of operators and engines and some of them entail knowledge of the code to be executed. Contrarily, we treat materialized operators as "black boxes", assuming no prior knowledge of their internals, and model them using profiling in an offline mode, as well as machine learning over actual runs. The detailed modeling methodology used in ASAP is thoroughly described in Section 4.

3.2.2 Profiling Module

The profiling module functions in an operator-agnostic way, having no prior knowledge other than the profiler input parameters. These parameters fall into three categories:

- Data specific parameters: These parameters describe the data to be used for the operator profiling, e.g., the type of data and its size.
- Operator specific parameters: These parameters relate to the algorithm of the operator, e.g., the number of output clusters in k-means.
- Resource specific parameters: These parameters define the resources to be tweaked during profiling, e.g., \#VMs, storage size, main memory, etc.

The output of each run is the profiled operator's performance (e.g., completion time) with each combination of the input parameter values for specific user-defined optimization metrics, such as cost in $ or I/O, latency, throughput, etc. Both the input parameters as well as the output metrics are given by the user/developer.

The aim of the profiling module is to create a surrogate estimation model [105] , including neural networks, SVM, interpolation and curve fitting techniques, for each operator running over a specific engine. To that end, we need to sample the operator function by running automated experiments for various values of each of the input parameters and measure the outputs. To create the most accurate surrogate within a budget of experiments, adaptive sampling techniques are adopted to select the combinations of values to be used as input of each run. The detailed profiling methodology is presented in the following section (Section 4).

3.2.3 Decision Making Module

This module is charged with the intelligent exploration of all the available execution plans and the discovery of the optimal execution plan according to the user defined optimization objectives. Initially, it transforms the abstract workflow representation into a materialized workflow graph that contains all the alternative paths of materialized operators that match the abstract workflow. To do so, for each abstract operator it searches the library of available materialized operators to find all matches. Our decision module uses an efficient tree matching algorithm to avoid unnecessary comparisons and follow the hierarchical structure of the tree-based metadata constrains, as described in Section 3.1.1. When all operator matches are discovered, the decision making module consults the input and output specifications of the materialized operators and adds the required move/transform operators. Those operators are needed in order to connect operators of different engines and input/output configurations and generate the final materialized workflow graph.

Figure 8 depicts the complete of all alternative execution plans for the simple join example of Figure 2 in section 3.1.1. To be able to match the join_1 operator, which joins two datasets over Hadoop and thus requires both its inputs in HBase, with the dataset D2 that resides in MySQL, a move operator must be added (depicted in dash line). As its metadata description in Figure 9 reveals, the move operator moves a dataset from MySQL to HBase. This operator is placed between D2 and join_1 to produce a dataset (D'2) that complies with the input constraints of join_1.

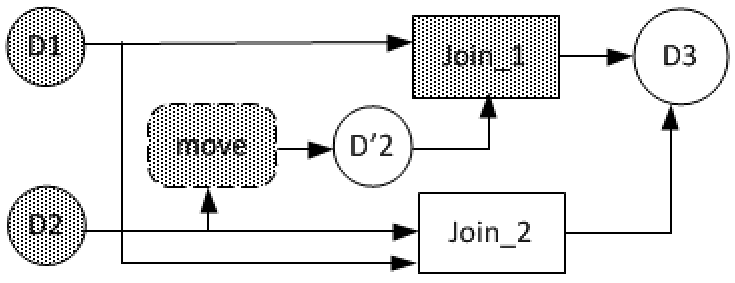


Figure 8 Complete graph of execution plans

```
{"move": {
  "constrains": {
    "input1": {
      "data_info": {
        "attributes": [
          {"customer_id":   {"type": "Varchar(15)"}},
          {"customer_name": {"type": "Varchar(15)"}}]},
      "engine": {
        "DB.Relational.MySQL": {
          "key:":      "customer_id",
          "location": "83.212.118.9" }}},
    "output1": {
      "data_info": {
        "attributes": [
          {"customer_name":      {"type": "ByteWritable"}},
          {"customer_purchases": {"type": "List<ByteWritable>"}}]},
      "engine": {
          "DB.NoSQL.HBase": {
            "key:":      "customer_id",
            "value":    "customer_purchases",
            "location": "83.212.118.9"}}
    },
    "op_specification": {
      "algorithm": {"move": {}}},
    "engine_specification": {
      "Centralized.Java": {"location": "localhost"}}
  },
  "optimization": {
    "cost_model": {
      "profile":
        ["execution_time", "required_ram"]}
 }}}
```

Figure 9 Metadata description of operator move

To find the optimal execution plan, our decision module uses a dynamic programming planner that explores the materialized workflow graph in order to find the plan that best matches the user optimization policy. To estimate operator performance metrics, our planner consults the *Model DB* that holds surrogate estimator models for each one of the materialized operators. In our current implementation, our planner can be configured to optimize one metric or a function of multiple performance metrics that the user is interested in. We are currently investigating methods for optimizing multiple dimensions of performance metrics, like finding Pareto frontier execution plans.

For our running example, let's assume a user optimization policy which includes minimizing execution time while guaranteeing fault tolerance. The performance and fault tolerance estimation that derives from the IReS surrogate models designate the execution plan that better fits these criteria, marked in green in Figure 10.

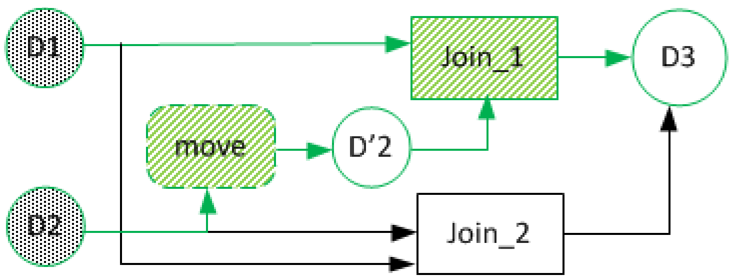


Figure 10 The selected execution plan

In the course of the workflow execution, the real-time monitoring information is fed back to the decision making module in order to take into account current running conditions and adapt accordingly. Moreover, our planner considers more than a single final plan to ensure that alternatives will exist in case of failures or other unpredictable circumstances without having to run the whole decision making process from scratch. These alternatives include the top-k (instead of the best) plans according to the user's optimization preferences or a sample of the multi-dimensional space covering different environments.

3.2.4 Model Refinement Module

Compute and data engine models are initially created during the offline modeling process, as described in section 3.2.1. These models are refined during the online modeling process, that is, during the actual runs of a workflow.

3.3 The Executor Layer

The **executor** layer is the layer that enforces the optimal plan over the physical infrastructure. It includes methods and tools that translate high level "start runtime with using x resources", "move data from site Y to Z" type of commands to a workflow of primitives as understood by the specific runtimes and storage engines. Moreover, it is responsible for ensuring fault tolerance and robustness through real-time monitoring. It's core module is the *enforcer module*, described in the next section.

3.3.1 Enforcer Module

The enforcer module undertakes the execution of the ensuing plan. First, the enforcer needs to validate the plan by checking the availability of resources and data, the load of the engines etc. After ensuring that everything is correct, it enforces the plan actions by translating the plan steps to standard, low-level API calls. Such actions might entail code and/or data shipment if necessary. In case of on-the-fly faults and failures an alternative plan will substitute the current.

3.4 Workflows

In this section, we present the functionality of our IReS platform by describing the major workflows involved.

3.4.1 Profiling Workflow

In order for our decision module to be able to estimate the performance of an operator, we need to obtain knowledge about the behavior of operators over different engines, resource configurations, input parameters, dataset sizes etc. This knowledge can be generated both from on-line learning, during the execution of different operators, as well as from offline profiling that automatically executes and monitors the operators using different resource and dataset configurations. The offline profiling process can help our decision module have a steeper learning curve and avoid planning errors for operators with unknown performance.

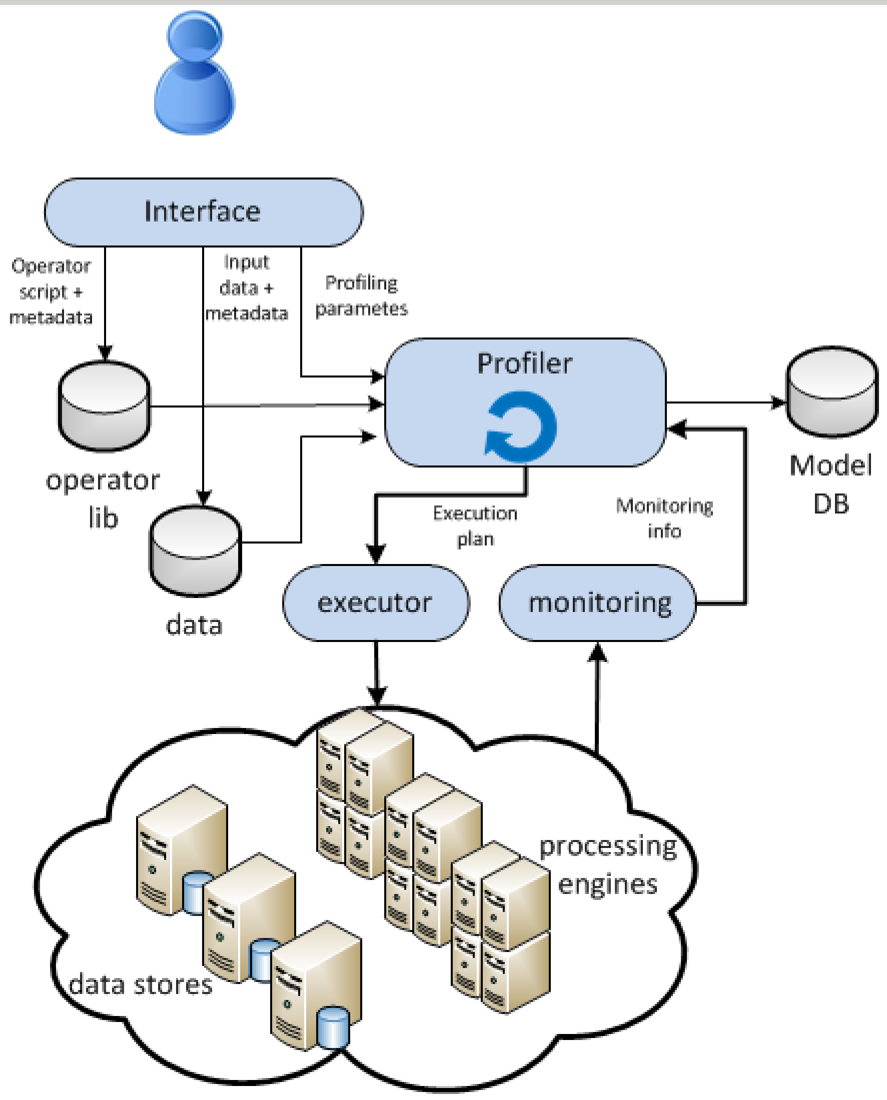


Figure 11 Profiling workflow

Whenever a new materialized operator gets inserted in the IReS platform, the profiling workflow takes place, depicted in Figure 11. During profiling, a number of different operator configurations are selected, executed and monitored in order to identify the relationship between a specific configuration and the operator's performance metrics. All the user provided performance monitoring metrics are measured and evaluated in order for IReS to generate a knowledge base, Model DB, that can be used to facilitate the decision making process. The main challenge for the Profiling module is to intelligently choose the set of configurations to be profiled.

Each operator execution has a respective cost both in time and money. Therefore, the Profiler attempts to tackle the problem of generating the most accurate profile within a user specified budget of experiments.

More details and UML use case diagrams can be found in deliverable D1.2.

3.4.2 Planning and Execution Workflow

This is the main workflow of the IReS platform, depicted in Figure 12. As mentioned in Section 3.1.1, the user provides an abstract description of the workflow she wants to execute. The first task of the IReS platform is to match the abstract operators present in the user provided abstract workflow with the materialized operators imported in the platform's operator library. The result of the operator matching process is the materialized graph of the workflow that contains all the possible alternative execution plans that match with the abstract workflow plan.

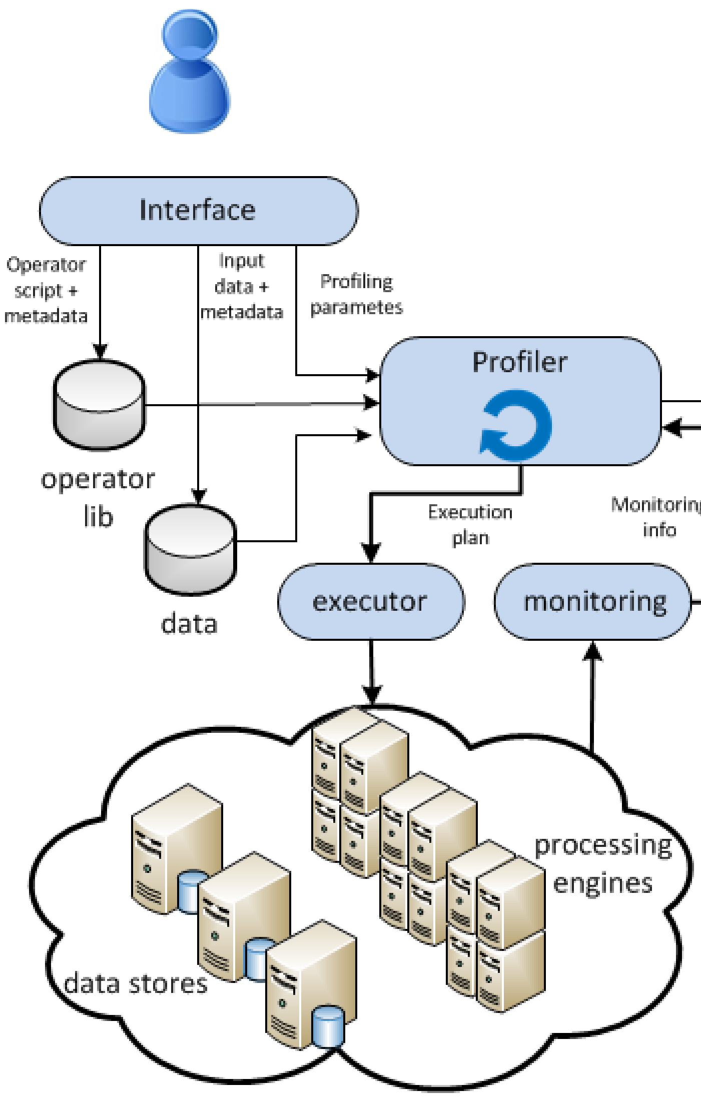


Figure 12 Planning and execution workflow

When a new workflow execution is triggered the user can provide the optimization policy that she wants to enforce on its execution. This policy can consist of one or a function of multiple operator performance metrics like cost, execution time, etc. Then, the Decision Making module explores the materialized graph of the workflow in order to find the plan that best matches to the user defined policy. When the optimal plan is located, its execution is enforced by the Enforcer module. As mentioned in Section 3.2.3, apart from the optimal plan, IReS locates the top-k best plans in order to be able to fall back to the execution of another plan. This can happen during the validation of the plan, when the Enforcer detects that the actual plan execution deviates largely from its expected execution. Lastly, IReS manages the elasticity of the underlying infrastructure by monitoring the utilization of the engine resources. Based on this monitoring information it can take decisions for allocating and de-allocating computing resources in order to improve the general execution of workflows and operators.

More details and UML use case diagrams can be found in deliverable D1.2.

3.5 IReS Interaction with other ASAP modules

This section describes how the IReS platform interfaces with the rest of the ASAP system components, as defined in work packages 2-5.

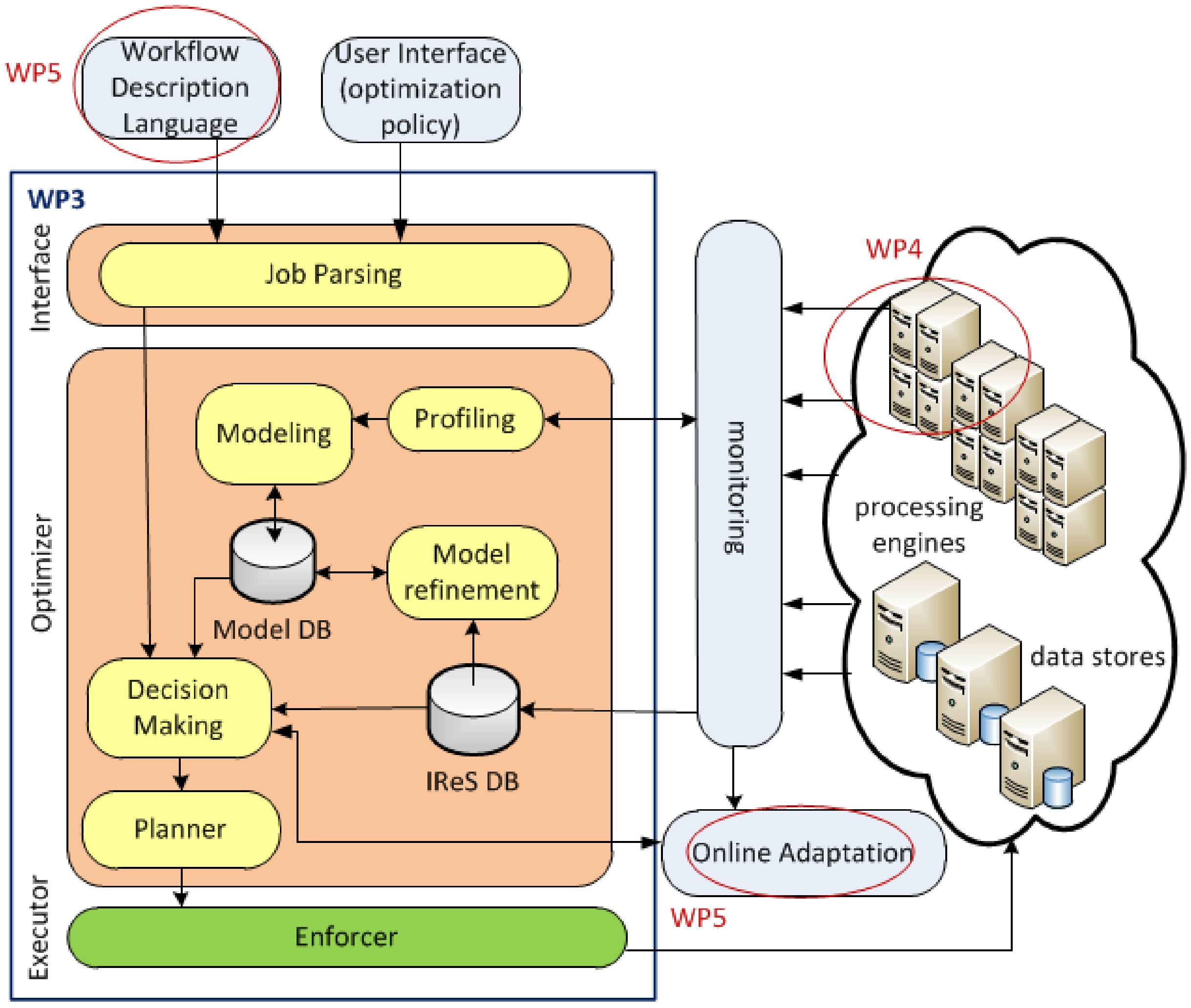


Figure 13 Interaction of IReS with other ASAP modules

Figure 13 depicts the position of the IReS platform in the ASAP system and the points of interaction with the rest of the research work packages. There are 5 external modules that interact with the IReS platform:

- The **Workflow Description Language**, which describes the user-defined workflow

- The **User Interface**, where the user defines her optimization policy.
- The **Monitoring module**, which monitors the execution of the workflow over the system infrastructure, namely the processing engines and data stores.
- The **Online Adaptation module**, which allows the user to change the parameters of a long running workflow without restarting the whole workflow computation.
- The **infrastructure** itself, where the various parts of the workflow are deployed and executed.

Thus, these are the 4 actors of the IReS platform.

The following Table (Table 4) describes the ways in which IReS interacts with the above described external actors. The exact APIs are still under definition.

Functionality	**Actors**	**Description**
Add operator	Workflow Description Language, User Interface	The ASAP user can add operators along with their description using either the Workflow Description language or the ASAP User interface.
Add dataset	Workflow Description Language, User Interface	The ASAP user can add datasets along with their description using either the Workflow Description language or the ASAP User interface.
Add Abstract Workflow	Workflow Description Language	An abstract workflow is provided using the ASAP workflow description language. This abstract workflow contains abstract operators that match with several of the materialized operators already described and available in the IReS platform.
Add optimization policy	User Interface	The user provides the optimization parameters of her workflow through the User Interface.
Materialize/ Optimize workflow	Workflow Description Language, User Interface, Online Adaptation module	After the description of an abstract workflow the ASAP user can trigger its materialization and optimization phase using both the Workflow Description language and the ASAP User interface. The output of this procedure is a materialized workflow that contains all the possible execution paths that match with the abstract workflow. The IReS platform also handles the optimization of the workflow according to user specified policies. The Online Adaptation module can also trigger this method in order to retrieve information about the execution of multiple workflows using several policies.
Execute	Workflow	After the materialization and optimization phases,

workflow	Description Language, User Interface, Monitoring module, Infrastructure,	the workflow is ready to be executed. The execution can be triggered by the ASAP user through both the Workflow Description language and the ASAP User interface. The execution of the required operators is scheduled and monitored by the IReS platform.
Profile operator	Workflow Description Language, User Interface	The ASAP user can trigger the profiling of a described operator. The profiling loop executes the specified operator using different input specifications and monitors its user defined output parameters. During the profiling loop, the input/output data gathered are used to train surrogate estimator models for the operator. The data are also used to guide the adaptive sampling techniques proposed by our profiling system, in order to achieve the best estimation accuracy within a user specified budget of profiling experiments.
Monitor workflow	Monitoring module, Infrastructure, Online Adaptation module	Throughout the execution of operators and workflows, the IReS platform uses the monitoring module to gather the user defined monitoring metrics. The metrics gathered are persistently stored and used to further train the surrogate estimator models of the operators. The metrics are also used by the Online Adaptation module to trigger changes on the workflow execution plan.

Table 4 High level external interface of the IReS platform

4 Compute and Data Engine Modeling

In this section, we present our approach on modeling and estimating the execution characteristics of specific engines and operators. In order for ASAP and the IReS platform to be able to decide on the engine and operator implementation that best fits a user defined workflow execution policy, we need a way to estimate performance metrics for different operators and engines. More specifically, this task undertakes the process of creating realistic, updatable and multi-dimensional models of the performance and cost of the different runtime and store technologies in order to be used by the scheduling algorithm for optimal workflow execution. The generated models should be able to estimate the effect of multiple dimensions on the performance and cost metrics of an operator. Such dimensions could be the type and complexity of the task/data to be run/stored, the amount of resources/storage available, the data/load skew, the architecture, the elastic properties, etc. The performance and cost metrics modeled should be user defined and extendable in order to allow the users to define the optimization policies that best suit their needs. Initial operator and engine models can be generated by running automated benchmarking experiments for different configurations. The models should also be updated using performance measurements retrieved from the actual execution of operators on the IReS platform.

4.1 State of the Art

In this section, we present related work on techniques used for benchmarking, engine and runtime modeling as well as automated application profiling.

4.1.1 Benchmarks

HiBench [58] is a Hadoop benchmark consisting of a collection of common Hadoop applications. It takes a hybrid approach, using partly the micro-benchmarks, included in the Hadoop package, and partly selected, common, real-world applications. Namely, it includes the indexing workload from the Nutch open-source search engine [61] , the PageRank, Bayesian Classification and K-means Clustering from the Mahout library [16] and the Join and Aggregation queries of the Hive performance benchmarks. As inputs, HiBench uses a dump of Wikipedia for PageRank and Classification, randomly generated data from statistic distributions for Clustering, and the inputs and queries defined in [3] for Hive Joins and Aggregation.

MalStone [25] , runs a custom analytics on automatically generated data. The authors argue that web-scale analytics datasets are often proprietary and thus not available to the general public. To tackle this challenge, they developed MalGen, synthetic data generator, simulating multiple web sites' log files and web-user behavior. The benchmark contains multiple implementations of classifiers and algorithms that try to automatically detect web-user patterns and strange behaviors.

In MRBench [62] the authors try to create a custom version of the industry standard decision support systems benchmark suite, TPC-H [29] , specifically for MapReduce workloads. Their approach is to convert each of the 22 TPC-H queries into MR jobs, each one consisting of several Map and Reduce steps. Apart from this, they also offer an implementation of MRBench in Java for the Hadoop framework.

In an approach similar to MRBench, the work presented in [70] also tries to run the TPC-H suite on top of the Hadoop framework, with the difference that the authors re-write the queries using the "Pig latin" [74] high level query language and use the Pig system to translate them into a MapReduce workflow.

The approach in [108] is similar to the one in MRBench and [70] but this time the TPC-H queries are executed using the Hive SQL implementation.

The work in [72] tests a modified subset of the TPC-H queries in the MongoDB NoSQL database (6 of 22 queries). The results are presented in comparison with PostgreSQL and show that MongoDB induces performance overheads when dealing with complex analytic calculations.

4.1.2 Modeling

The work presented in [45] is an effort to describe the performance of a MapReduce job execution using mathematical models. The authors state that the map and reduce phases can also be further divided to more primitive sub-tasks. They leverage this fact in order to analytically model dataflow and cost information at a fine granularity, using a set of parameters based on the framework's configuration, the properties of the input data and some cost factors. The paper defines mathematical formulas describing the performance of each MR sub-task but lacks any experimental data arguing about the accuracy of any of the parts of the model.

4.1.3 Profiling

Predicting the performance of applications running over virtualized resources is vividly researched in the literature. In [85] , Kundu et al. proposed an iterative model training technique for Neural Networks with which the authors managed to predict the minimum possible Virtual Machine (concerning its resources) which would fulfill their objectives with respect to the SLAs. An extension of this work [86] also utilized Support Vector Machines for the same objective. Their work achieved highly accurate predictions, however the authors did not address the problem of sampling the input domain space, as we do in this work.

Furthermore, Iqbal et al. in [102] propose a method which, at first, identifies a workload pattern and secondarily builds a model capable to predict the application's capacity (the number of requests it can serve without violating given constraints). This work focuses on web applications and the prediction happens with regression models.

Similarly, Do et al. in [6] presented a profiling technique which utilizes the Canonical Correlation Analysis, able to identify the relationship between the allocated resources and the application performance. This work targets to predict the performance of a newly allocated Virtual Machine when it is deployed in a specific host running other Virtual Machines. Other works focus on predicting specific application metrics based on I/O workload and access patterns such as [79] and [84] .

4.2 The IReS modeling approach

In this section, we present the techniques used by the IReS platform in order to estimate the performance and cost characteristics of different operators and engines. This module lies behind project ASAP's ability to provide optimized, adaptive and highly extensible analytics execution. The main idea is to provide predictions for each operator's performance by actually running the operator in representative configuration combinations. Using these measurements we can train surrogate estimator models that can be used to approximate its performance for non-tested configurations. To do so, in a generic and extendable way we propose a black box operator profiling framework.

4.2.1 Black Box profiling

In order to provide a generic operator-profiling framework, we follow the black box profiling approach. According to this, we model each operator as a black box that has user defined inputs and outputs. The input space of an operator can be also described as its design space and contains all the parameters that affect its performance and need to be varied in order to profile it. For example, the input space of an operator can contain parameters like:

- platform runtime resources (e.g. number of VMs, number of CPUs, available RAM, etc.)
- Data attributes (e.g. dataset size in GBs, type, distribution, etc.)
- Operator-specific parameters (e.g., the number of clusters in k-means, number of iterations, accuracy, etc.)

As mentioned before, the input space of an operator is user defined, giving the users the capability of defining the parameters that affect the operator's performance. The user should also give the type of each parameter in order for our profiling system to be able to vary it and test different configuration automatically. For example, a parameter such as the number of VMs is a discrete integer value that can have a minimum and maximum value in order to prune the possible combinations. Concerning data input parameters, like the dataset size or type, the user can provide a set of sample datasets or a dataset generator that can be used in order to test various configurations.

The output space of an operator can also be described as its optimization space and contains all performance/cost metrics that need to be approximated for the various input configurations. For example, the output space of an operator can contain the following metrics:

- Execution time
- Cost
- Accuracy of the result
- Throughput, latency
- Min, max and average CPU, memory consumption, etc

Our profiling framework is generic and allows the user to define the output parameters that she wants to optimize for different operators. A new optimization parameter can be defined simply by giving a monitoring probe that can measure it when running the operator.

4.2.2 Profiling challenges

The operator profiling is a process that allows the automated execution of operators and monitors their behavior over representative input space configurations. The collected information can form the basic knowledge used to train *surrogate* estimator models that can approximate the operator's behavior (the function that relates the input/configuration space parameters with the output/optimization metrics).

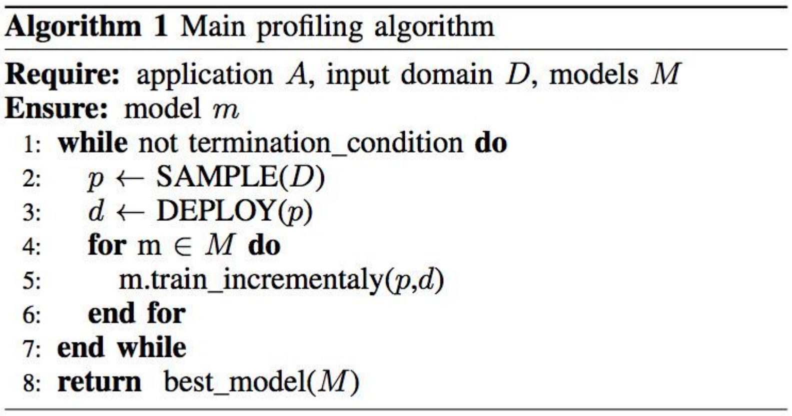
The main challenge for the Profiler is to intelligently choose the set of profiled configurations. For example, if we have an operator with 3 integer input parameters that range from 1 to 10, there exist 10^3 different deployment configurations. Furthermore, each execution of an operator has a respective temporal and monetary cost in order to be sufficiently profiled. A brute force profiler would need to execute and monitor all those configurations. In such case, the execution time of the profiler could be exponential to the number of inputs, something which is not acceptable. ASAP's Profiler should be able to intelligently narrow down the field of profiling scenarios. Therefore, the Profiler attempts to tackle the problem of generating the most accurate operator profile within a user specified profiling budget of experiments.

The nature of operator profiling is clearly multi-objective, often requiring tradeoffs between diverse and conflicting objectives. While the input parameters, design space, of an application include the number of VMs, their RAM, their disk capacity etc., an application user can be interested in various objectives such as cost, throughput, latency etc. Therefore, the operator can be modeled as a function that maps the design space (number of VMs, RAM, data size, operator parameters, etc.) to the user defined objective space (cost, execution time, etc.). This function represents the operator's profile. Our Profiler will use targeted operator runs, according to a specified financial and time budget, to provide a global surrogate approximation model of the operator's profile function that maps its design space to its optimization space.

Many engineering and science problems require expensive experiments or time consuming simulations to generate sample points of the mapping between the input and the output parameters of a system. In such cases, researchers have focused on building accurate surrogate approximation models that, when properly constructed, can mimic the behavior of the system while being computationally cheap to evaluate. Examples of surrogate models include: Kriging models [91] , Splines [33] Artificial Neural Networks [37] , Support Vector Machines [60] etc. The challenge here is how to generate a surrogate model that is as accurate as possible over the domain of interest and at the same time minimize the cost of the performed experiments. Since the system's response behavior is not known upfront and the sample data points are too costly to obtain, the main approach followed is the iterative adaptive sampling of the design space. Each data point obtained is used to update the surrogate approximation model as well as the sampling function. In each iteration, the sampling function selects the next sampling point according to an estimation of its benefit to the surrogate approximation accuracy. This technique is called importance or adaptive sampling and is also known as sequential design.

4.2.3 Profiling approach

In Algorithm 1, we provide the general methodology used to create a profile for a given operator. The algorithm expects a valid operator/application description A followed by an input domain D, representing the possible setups the operator can be executed with and a list of surrogate models. The profiling process occurs iteratively: while the termination condition is not fulfilled, the domain space is sampled, a new point p is picked and the operator is executed according to p. The deployment produces an optimization vector d, containing the measured outputs, which is then used to train in an incremental manner all the available surrogate models. The output of the profiling process is the surrogate model which achieves the highest accuracy, according to a user specified metric.



```
Algorithm 1 Main profiling algorithm
Require: application $A$, input domain $D$, models $M$
Ensure: model $m$
while not termination_condition do
    $p \leftarrow$ SAMPLE($D$)
    $d \leftarrow$ DEPLOY($p$)
    for m $\in M$ do
        m.train_incrementaly($p$,$d$)
    end for
end while
return best_model($M$)
```

Figure 14 Main profiling algorithm

The termination condition can vary. It can be a threshold of sampled points that, if reached, the condition is true and the algorithm terminates. In other cases, it can be related to the achieved accuracy: if the trained model achieves to predict the objective function with error lower than a user defined threshold, the termination condition is reached. As we will present in the following section, the nature of the termination condition is directly entwined with the nature of the sampling algorithm.

4.2.4 Adaptive sampling

The sampling procedure occurs at the beginning of each profiling loop. The sampler receives as input the domain space D of the operator, which is composed of all the acceptable deployment points. Each point returned by the sampler is used for execution. The operator's output metrics are measured and then an approximation model is trained using the acquired information.

There are many methodologies for sampling a multidimensional space. We can categorize the methods we support in the following categories:

1. Static sampling, where the sampler needs no other information than the domain space characteristics (dimensions and acceptable values) to pick the next sample
2. Adaptive sampling, where the sampler exploits the knowledge obtained by the deployment of previously picked samples.

The static approach does not take into consideration the operator's performance. Typical examples of static sampling are the Random sampler, that returns random points and the Uniform sampler which constructs a multidimensional grid in the input space D, and returns points belonging to the grid. We opt for an adaptive sampling approach which exploits the knowledge obtained from each deployment/sample, enabling the sampler to retrieve more samples in regions of the domain space D where the performance appears to have fluctuations or the models have the maximum estimation errors. Equivalently, an adaptive sampler favors areas of D where the operator performance has the most deviations in order to use them to provide more accurate approximation models.

4.2.5 Approximation models

When a new sample is picked by the sampler and executed, the performance metrics are stored and given as input to an approximation model. The training set of the model consists of the chosen samples along with their output values. After the training process is finished, the model will be able to approximate the objective function for the entire space D. There exist many methodologies for approximating an unknown function. We can categorize them in two major categories: regression based techniques and classification techniques. Algorithms on the former category create an analytical form of the objective function. The classification techniques, on the other hand, do not target to create an analytical function but to classify the points of the domains space in classes. These objects are treated in a similar manner, indicating that the same properties stand for objects in the same class.

In our approach, we utilize the approximation models offered by WEKA [103] , an open source data mining software which implements a variety of machine learning algorithms. Specifically, the supported approximation techniques are the following:

- Gaussian Process, that approximates the objective function using Gaussian distributions
- Multilayer Perceptron, that represents a typical neural network with many hidden layers and neurons
- Linear Regression (Least Median Squares), that implements the methodology introduced at [75]
- Bagging, that executes classification as described in [63]
- Random SubSpace, that constructs a decision tree using the approach presented at [97]
- Regression by Discretization, that enforces regression over a discretized domain of the input space
- RBF Network, which trains a Radial Basis Function Network, as presented at [31]

The accuracy of each one of the aforementioned models is highly affected from the configuration of the model and the nature of the objective function. For example, a linear hyper- plane will be approximated faster using a linear regression method. On the contrary a complex surface which has spikes and valleys is more likely to be approximated more accurately using a non-linear approach. All the available models are trained in parallel by the system, and the model which achieves the best accuracy is eventually chosen.

4.3 Experimental evaluation

To evaluate the performance of our Profiler, we have selected a set of distributed analytics operators that are deployed over large scale virtualized resources. The first benchmark application is TeraSort [73] , a well-known benchmark that sorts a set of key values. We test it with datasets of 10M up to 50M key-values (1GB to 5GB of data respectively) and run the TeraSort in Hadoop clusters with different number of nodes and different number of cores per node. The second operator is a BSP-based implementation of PageRank [64] , a well-known graph algorithm implemented over the Apache Hama framework. We utilize 50K to 100K node graphs, each of which has at most 50 outgoing edges and execute PageRank over different cluster sizes as above. Finally, the third operator is a BSP implementation of the Single Source Shortest Path (SSSP) algorithm [87] implemented for the Apache Hama framework. For SSSP, we create synthetic graphs consisting of 50k up to 500k vertices and at most 50 edges per node.

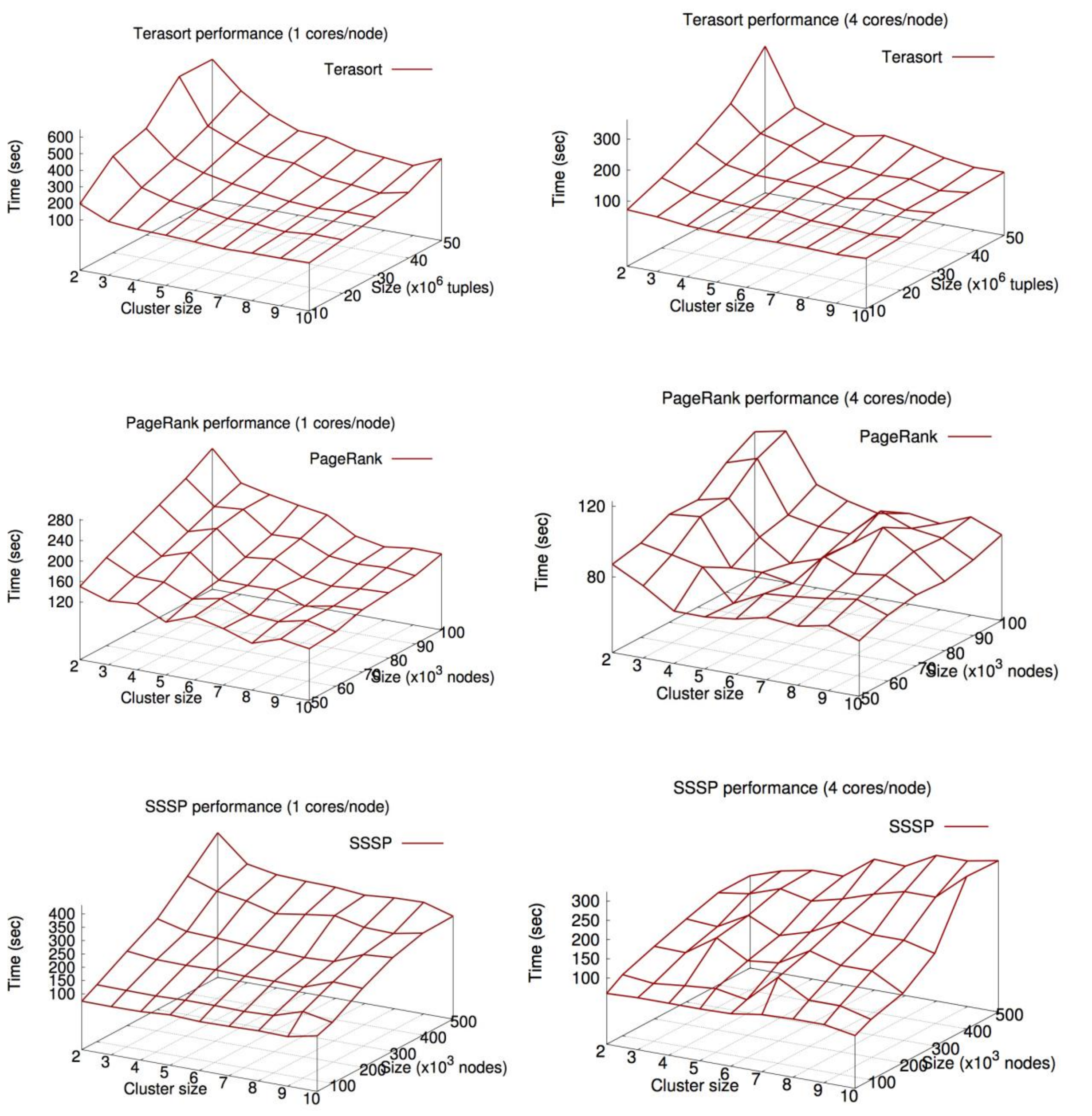


Figure 15 Profile functions for different operators

The running times for various deployment configurations of all the three benchmark applications is given in Figure 15. We provide the execution time of the Terasort benchmark with regard to the size of the cluster and the dataset size (measured in millions of key-values). It is obvious that the execution time is inversely proportional to the cluster size and proportional to the dataset size. Furthermore, for large clusters we notice that the execution time decreases less rapidly, because the communication overheads affect more the overall execution time.

The execution time for both PageRank and SSSP are also shown in Figure 15. PageRank has a similar behavior to the Terasort case. SSSP, on the other hand, presents a slightly different behavior in terms of scalability. Specifically, when more nodes are added to the Hama cluster, the execution time remains unaffected for smaller dataset sizes (e.g., 50k nodes). For larger datasets it decreases, but less rapidly than in the other cases. This is due to the larger number of supersteps executed by SSSP. Specifically, for our datasets, each SSSP job requires about 25–30 Hama supersteps while PageRank requires only a third of them. As a consequence, SSSP needs more sequential steps thus more time for synchronization between the BSP workers. Thus, due to this cost, the addition of more workers does not greatly benefit SSSP.

One of the greatest factors that affect the performance of our Profiler is the sampling rate. This is defined as the ratio between the number of the chosen points and the total number of acceptable deployments. Lower sampling rates lead to fewer chosen points, offering the classifiers less knowledge for the objective function (the performance of the application). We use the coefficient of determination R^2 [80] to quantify the accuracy of the profiling methods. R^2 declares the degree in which a classifier fits the original data. It is calculated as follows:

$$R^2 = 1 - \frac{\sum_i (y_i - f_i)^2}{\sum_i (y_i - \overline{y})^2}$$

Where: y_i are the real performance values, f_i are the predicted values and $\overline{y}$ is the mean of the observed data. The closer R^2 gets to 1, the better the performed approximation. In our experiments, R^2 was calculated taking into consideration all the available points (sampled or not), thus R^2 indicates the accuracy of our models in the entire domain space. Simultaneously, we also utilize the Mean Absolute Error metric which is defined as:

$$MAE = \frac{1}{n} \sum_i |y_i - f_i|$$

Where: i, f_i and y_i have the same notation as before. Both metrics have been evaluated for the entire input space, including both the points picked during the sampling phase and the rest of the points in order to capture the resemblance between the approximated and the objective function in the entire domain space. In the general case this will not be possible, since the performance will only be given in the sampled points; In those cases the metrics will be evaluated using only the deployed points.

For this experiment, we applied the sampling methodologies presented in the previous section and trained all the available approximation models with the chosen points along with the respective performance values for different sampling rates. In Figure 16we provide the accuracy level of the best model for each sampling rate for all three applications using the coefficient of determination. The best model is defined as the model that presents the highest coefficient of determination. We also provide the Mean Absolute Error for each application.

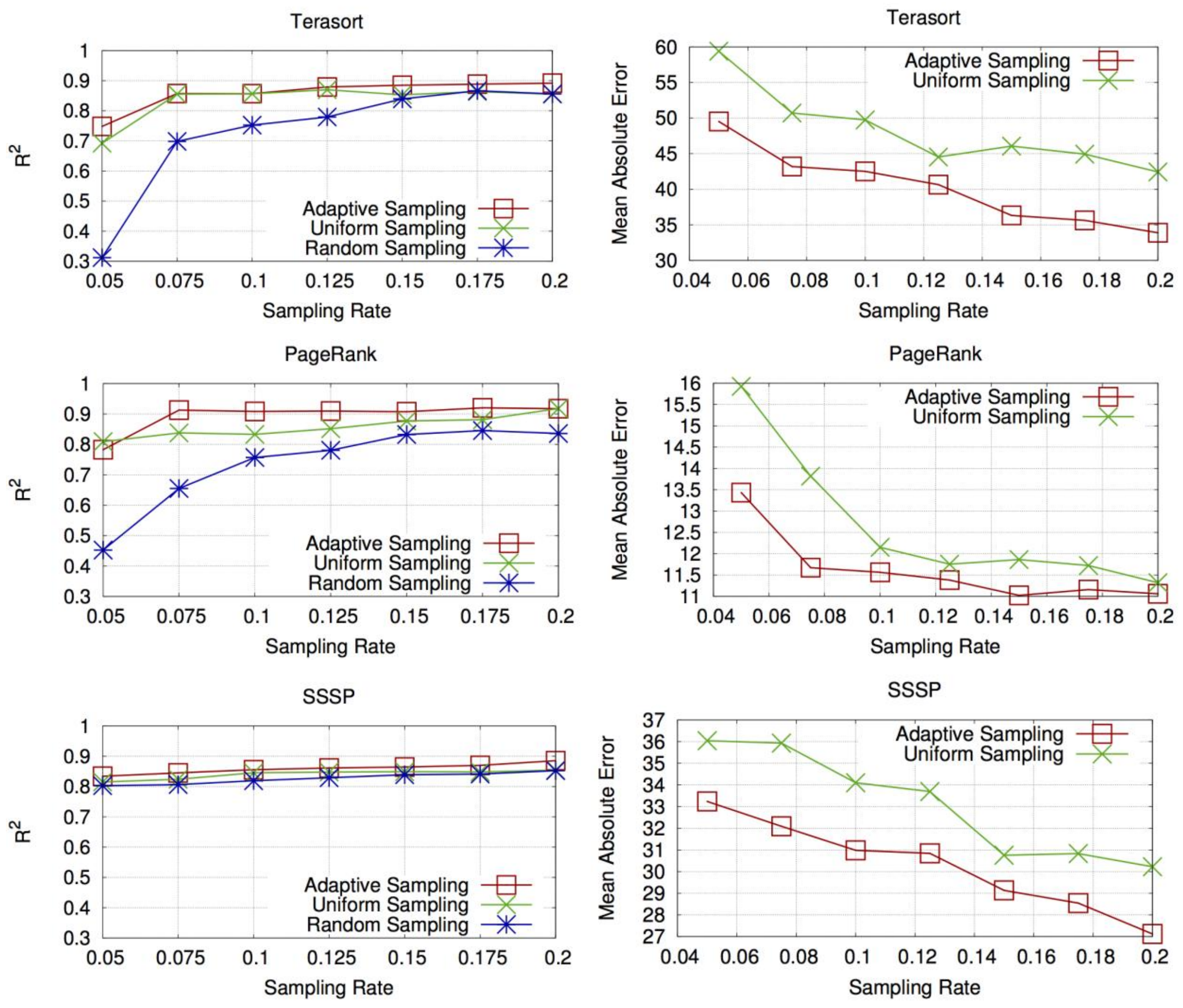


Figure 16 Estimation accuracy for different sampling techniques

In our results, we notice that the most accurate models present slightly different behavior for each one of the three applications. In all of them, it is obvious that an increase in the Sampling Rate leads to higher accuracy. This result is expected, since higher sampling rate means that more points are picked, thus the model will obtain more knowledge for the objective function. However, in many cases this might not be the case: The sampler may pick more points but if they are not representative ones, they may mislead the model and eventually, this may cause lower accuracy. For example, this is the case for Terasort, when increasing the sampling rate from 0.125 to 0.15 for the Uniform sampler. More points are chosen, but very few of them are picked in the regions where the execution time is high, thus the model cannot make more accurate predictions. This behavior is avoided using the Adaptive Sampler, since it constantly checks the ranges with the steepest differences and favors them, leading the models to identify those regions faster.

However, when the Sampling Rate is relatively low, this behavior could also mislead the models, as seen in the Terasort case. Specifically, because of the first phase of the Adaptive algorithm, where the border points are returned, the models may eventually achieve worse accuracy than the one achieved with the Uniform sampler which, again, picks more points in the intermediate region and enables the models to make more accurate predictions. In any case, we observe that the difference in accuracy is acceptable while, eventually, for bigger sampling rates the Adaptive sampler outperforms the Uniform sampler. In the PageRank case, we notice that the Adaptive Sampler outperforms the rest of the samplers for most rates, whereas in the SSSP case, the surface which is to be approximated resembles a linear hyperplane. This enables the regression techniques to accurately approximate it with a relatively small number of points. This also explains the smaller increase observed in accuracy and the reason for which the models achieve high accuracy with the lowest sampling rates (something not observed in the previous applications).

The Random sampler is not included in the MAE figures, since it had the worst performance (between 100 and 80 for Terasort, 30 to 25 for Pagerank and 60 to 50 for SSSP). It is obvious once again that increasing the sampling rate leads to lower MAE, thus higher accuracy. It is also obvious that the rate with which MAE is decreasing tightly coupled with the form and the nature of the objective function. Specifically, in the Terasort and SSSP cases MAE decreases almost linearly with sampling rate; On the other hand PageRank appears to stabilize MAE decrease very soon. The reason for this is that the performance of PageRank appears to have more oscillations than the other applications. This makes it harder for the models to capture this behavior and, eventually, increasing the sampling rate does not benefit the model. This is a very interesting phenomenon: when the performance of the application oscillates (because of its nature, the virtualization overhead, etc.), the threshold for which increasing the sampling rate creates more accurate models is decreasing.

In conclusion, the provided models in cooperation with the sampling methods enable the system to create an accurate profile of the application even when the sampling rate is less than 10% of the points of the domain space. At the same time, the profiling process is quite fast: Even when the Sampling Rate is 20%, the total time spent in training the models does not take more than 1.5 seconds. The input space of our experiments consists of 135 discrete points for the Terasort case and 162 points for the SSSP and Pagerank cases. Thus sampling with 20% of these spaces leads to 27 and 32 points respectively. Thus the training time of our models is less than 1.5 seconds when there exist 32 points for training. This is the time needed to successfully train all the available models. The time needed to deploy and run an operator dominates the profiling process and its typical duration is in the order of minutes. Finally, in more complex cases where the Input Space consists of thousands or tens of thousands of points a Sampling Rate of 20% is prohibitive. In such complex cases, our adaptive sampling approach can approximate the objective function with the best accuracy, within a user defined budget of experiments.

5 Conclusion

This deliverable describes the architecture of the IReS platform, the core component of the ASAP project that is responsible for managing, executing and monitoring complex analytics workflows. Its goal is to provide adaptive, cost-based and customizable resource management of the diverse execution and storage engines available. Moreover, since the area of high performance analytics advances daily, ASAP's goal is to present a repeatable process that will allow easy inclusion of different technologies, if so desired.

IReS incorporates a modeling framework that constantly evaluates the cost, quality and performance of data and computational resources in order to decide on the most advantageous store, indexing and execution pattern available. The methodology of the runtime and data store modeling process has been thoroughly described and an initial evaluation has been presented in order to showcase its efficacy.

References

[1] 84% Of Enterprises See Big Data Analytics Changing Their Industries' Competitive Landscapes In The Next Year . Forbes Magazine, 2014.

[2] A. Pariyani, U. G. Oktem, and D. L. Grubbe. Process risk assessment uses big data, 06-03-2013. http://bit.ly/1vDlTVk.

[3] A. Pavlo, A. Rasin, S. Madden, M. Stonebraker, D. DeWitt, E. Paulson, L. Shrinivas, and D. J. Abadi. "A Comparison of Approaches to Large-Scale Data Analysis", SIGMOD, June, 2009

[4] A. Simitsis, K. Wilkinson, and P. Jovanovic. xPAD: A Platform for Analytic Data Flows. In ACM SIGMOD 2013.

[5] A. Simitsis, K. Wilkinson, U. Dayal, and M. Hsu. HFMS: Managing the Lifecycle and Complexity of Hybrid Analytic Data Flows. In ICDE. IEEE, 2013.

[6] A. V. Do, J. Chen, C. Wang, Y. C. Lee, A. Y. Zomaya, and B. B. Zhou. Profiling applications for virtual machine placement in clouds. In Cloud Computing (CLOUD), 2011 IEEE International Conference on, pages 660–667. IEEE, 2011.

[7] Apache Cassandra, http://cassandra.apache.org/

[8] Apache Drill, http://drill.apache.org/

[9] Apache Flink, http://flink.apache.org/

[10] Apache GraphX, https://spark.apache.org/graphx/

[11] Apache Hadoop. http://hadoop.apache.org/

[12] Apache Hama, https://hama.apache.org/

[13] Apache HBase. http://hbase.apache.org/

[14] Apache Hive, https://hive.apache.org/

[15] Apache Kafka, http://kafka.apache.org/

[16] Apache Mahout, http://mahout.apache.org/

[17] Apache OpenNLP, https://opennlp.apache.org/

[18] Apache Parquet, http://parquet.incubator.apache.org/documentation/latest/

[19] Apache Samza, http://samza.apache.org/

[20] Apache Spark MLlib,https://spark.apache.org/mllib/

[21] Apache Spark, https://spark.apache.org/

[22] Apache Thrift, https://thrift.apache.org/

[23] B. Sharma, T. Wood, and C. R. Das. Hybridmr: A hierarchical mapreduce scheduler for hybrid data centers. In ICDCS. IEEE, 2013.

[24] Battré, Dominic, et al. "Nephele/PACTs: a programming model and execution framework for web-scale analytical processing." Proceedings of the 1st ACM symposium on Cloud computing. ACM, 2010.

[25] Bennett, Collin, et al. "Malstone: towards a benchmark for analytics on large data clouds." Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2010.

[26] Brewer, Eric. "Pushing the cap: Strategies for consistency and availability." Computer 45.2 (2012): 23-29.

[27] Chang, Fay, et al. "Bigtable: A distributed storage system for structured data." ACM Transactions on Computer Systems (TOCS) 26.2 (2008): 4.

[28] Cloudera Distribution CDH 5.2.0. http://www.cloudera.com/content/cloudera/en/downloads/cdh/cdh-5-2-0.html.

[29] Council, Transaction Processing Performance. "TPC-H benchmark specification." Published at http://www. tcp. org/hspec. html (2008).

[30] CQL, https://cassandra.apache.org/doc/cql/CQL.html
[31] D. S. Broomhead and D. Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. Technical report, DTIC Document, 1988.
[32] D. Tsoumakos and C. Mantas. The Case for Multi-Engine Data Analytics. In Euro-Par 2013: Parallel Processing Workshops. Springer, 2014.
[33] De Boor, Carl, et al. A practical guide to splines. Vol. 27. New York: Springer-Verlag, 1978.
[34] Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113.
[35] DeCandia, Giuseppe, et al. "Dynamo: amazon's highly available key-value store." ACM SIGOPS Operating Systems Review. Vol. 41. No. 6. ACM, 2007.
[36] elasticsearch. http://www.elasticsearch.org/overview/elasticsearch/.
[37] Funahashi, Ken-Ichi. "On the approximate realization of continuous mappings by neural networks." Neural networks 2.3 (1989): 183-192.
[38] Ghemawat, S., Gobioff, H., & Leung, S. T. (2003, October). The Google file system. In ACM SIGOPS operating systems review (Vol. 37, No. 5, pp. 29-43). ACM.
[39] Gu, Lei, and Huan Li. "Memory or Time: Performance Evaluation for Iterative Operation on Hadoop and Spark." High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on. IEEE, 2013.
[40] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu. Starfish: A Self-tuning System for Big Data Analytics. In CIDR, 2011.
[41] H. Lim, H. Herodotou, and S. Babu. Stubby: A Transformation-based Optimizer for Mapreduce Workflows. VLDB, 2012.
[42] Hadoop Distributed File System. http://hadoop.apache.org/docs/r1.2.1/hdfs design.html.
[43] Hadoop YARN, http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html
[44] HDFS, http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
[45] Herodotou, Herodotos, and Shivnath Babu. "Profiling, what-if analysis, and cost-based optimization of MapReduce programs." Proceedings of the VLDB Endowment 4.11 (2011): 1111-1122.
[46] heroku add-ons. https://addons.heroku.com/.
[47] Hortonworks Sandbox 2.1. http://hortonworks.com/products/hortonworks-sandbox/.
[48] http://h10120.www1.hp.com/expertone/datacard/Exam/HP0-A105
[49] http://lucene.apache.org/core/
[50] http://pegasus.isi.edu/
[51] http://www.elasticsearch.org/overview/
[52] http://www.elasticsearch.org/overview/elasticsearch
[53] http://www.oracle.com/technetwork/java/javase/jdbc/
[54] http://www.oracle.com/us/products/mysql/mysqlcluster/overview/index.html
[55] http://www.pivotal.io/big-data/pivotal-greenplum-database
[56] http://www.taverna.org.uk
[57] http://www.vertica.com/hp-vertica-products/dragline/
[58] Huang, Shengsheng, et al. "The HiBench benchmark suite: Characterization of the MapReduce-based data analysis." Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on. IEEE, 2010.

[59] Impala, http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html
[60] Joachims, Thorsten. "Making large scale SVM learning practical." (1999).
[61] Khare, Rohit, et al. "Nutch: A flexible and scalable open-source web search engine." Oregon State University 1 (2004): 32-32.
[62] Kim, Kiyoung, et al. "Mrbench: A benchmark for mapreduce framework." Parallel and Distributed Systems, 2008. ICPADS'08. 14th IEEE International Conference on. IEEE, 2008.
[63] L. Breiman. Bagging predictors. Machine Learning, 24(2):123–140, 1996.
[64] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. 1999.
[65] Laney, Douglas. "3D data management: Controlling data volume, velocity and variety." META Group Research Note 6 (2001).
[66] M. Ferguson. Architecting a big data platform for analytics. A Whitepaper Prepared for IBM, 2012.
[67] Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., & Czajkowski, G. (2010, June). Pregel: a system for large-scale graph processing. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data (pp. 135-146). ACM.
[68] Melnik, Sergey, et al. "Dremel: interactive analysis of web-scale datasets." Proceedings of the VLDB Endowment 3.1-2 (2010): 330-339.
[69] Monetdb. https://www.monetdb.org/.
[70] Moussa, Rim. "TPC-H benchmarking of Pig Latin on a Hadoop cluster." Communications and Information Technology (ICCIT), 2012 International Conference on. IEEE, 2012.
[71] MySQL, www.mysql.com
[72] N. Rutishauser, "TPC-H applied to MongoDB: How a NoSQL database performs," 25 February 2012, supervised by: Prof. Dr. Michael BÖhlen, Amr Noureldin.
[73] O. OMalley. Terabyte sort on apache hadoop. Yahoo, available online at: http://sortbenchmark. org/Yahoo-Hadoop.pdf,(May), pages 1–3, 2008.
[74] Olston, Christopher, et al. "Pig latin: a not-so-foreign language for data processing." Proceedings of the 2008 ACM SIGMOD international conference on Management of data. ACM, 2008.
[75] P. J. Rousseeuw and A. M. Leroy. Robust regression and outlier detection. 1987.
[76] PageRank Benchmark test 0.6.4 vs 0.6.3 http://wiki.apache.org/hama/Benchmarks#PageRank_Benchmark_test_0.6.4_vs_0.6.3
[77] PostgreSQL, http://www.postgresql.org/
[78] Postgresql. http://www.postgresql.org/.
[79] Q. Noorshams, D. Bruhn, S. Kounev, and R. Reussner. Predictive performance modeling of virtualized storage systems using optimized statistical regression techniques. In Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering, pages 283–294. ACM, 2013.
[80] R. G. D. Steel and J. H. Torrie. Principles and procedures of statistics: with special reference to the biological sciences. 1960.
[81] RDD, http://spark.apache.org/docs/1.2.0/quick-start.html
[82] Running Databases on AWS. http://aws.amazon.com/running_databases/.
[83] S. Babu. Towards automatic optimization of mapreduce programs. In ACM symposium on Cloud computing, 2010.

[84] S. Kraft, G. Casale, D. Krishnamurthy, D. Greer, and P. Kilpatrick. Io performance prediction in consolidated virtualized environments. In ACM SIGSOFT Software Engineering Notes, volume 36, pages 295– 306. ACM, 2011.
[85] S. Kundu, R. Rangaswami, A. Gulati, M. Zhao, and K. Dutta. Modeling virtualized applications using machine learning techniques. ACM SIGPLAN Notices, 47(7):3–14, 2012.
[86] S. Kundu, R. Rangaswami, K. Dutta, and M. Zhao. Application performance modeling in a virtualized environment. In High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on, pages 1–10. IEEE, 2010.
[87] S. Pettie. Single-source shortest paths. Encyclopedia of Algorithms, pages 847–849, 2008.
[88] Shahrivari, Saeed. "Beyond Batch Processing: Towards Real-Time and Streaming Big Data." Computers 3.4 (2014): 117-129.
[89] Simitsis, Alkis, et al. "HFMS: Managing the lifecycle and complexity of hybrid analytic data flows." Data Engineering (ICDE), 2013 IEEE 29th International Conference on. IEEE, 2013.
[90] Simitsis, Alkis, et al. "Optimizing analytic data flows for multiple execution engines." Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. ACM, 2012.
[91] Simpson, Timothy W., et al. "Kriging models for global approximation in simulation-based multidisciplinary design optimization." AIAA journal 39.12 (2001): 2233-2241.
[92] Spargel, http://flink.apache.org/docs/0.6-incubating/spargel_guide.html
[93] Spark SQL, http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/
[94] SPARKSTREAMING, https://spark.apache.org/streaming/
[95] Stonebraker, Mike, et al. "C-store: a column-oriented DBMS." Proceedings of the 31st international conference on Very large data bases. VLDB Endowment, 2005.
[96] Stratosphere Project. http://stratosphere.eu/.
[97] T. K. Ho. The random subspace method for constructing decision forests. IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(8):832–844, 1998.
[98] The Power of Combining Big Data Analytics with Business Process Workflow. CGI Whitepaper, 2013.
[99] TPC-H benchmark. http://www.tcp.org/hspec.html.
[100] Valiant, Leslie G. "A bridging model for parallel computation." Communications of the ACM 33.8 (1990): 103-111.
[101] Vogels, Werner. "Eventually consistent." *Communications of the ACM* 52.1 (2009): 40-44.
[102] W. Iqbal, M. N. Dailey, and D. Carrera. Black-box approach to capacity identification for multi-tier applications hosted on virtualized platforms. In Cloud and Service Computing (CSC), 2011 International Conference on, pages 111–117. IEEE, 2011.
[103] WEKA, http://weka.wikispaces.com
[104] Xin, Reynold S., et al. "GraphX: Unifying data-parallel and graph-parallel analytics." arXiv preprint arXiv:1402.2394 (2014).
[105] Y. Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. Swarm and Evolutionary Computation, 2011.
[106] YAGO2s: A High-Quality Knowledge Base. http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/.

[107] Yang, Fangjin, et al. "Druid: a real-time analytical data store." Proceedings of the 2014 ACM SIGMOD international conference on Management of data. ACM, 2014.
[108] Yuntao Jia. "Running the TPC-H Benchmark on Hive", https://issues.apache.org/jira/secure/attachment/12416257/TPC-H_on_Hive_2009-08-11.pdf. 2009
[109] Z. Zhang, L. Cherkasova, A. Verma, and B. T. Loo. Automated profiling and resource management of pig programs for meeting service level objectives. In Conference on Autonomic computing. ACM, 2012.