Huawei Open Data Analytics Platform

Gary Verhaegen*, Charles Bonneau*, Antonios Tsaltas*

*Avenue Albert Einstein, 2a, B-1348 Louvain-la-Neuve
gary.verhaegen,charles.bonneau,antonios.tsaltas@huawei.com,
http://www.huawei.com

Abstract. New frameworks such as Spark, Tez, Flink, or Hive offer new possibilities, but using more than one framework is generally not easy. We build an abstraction layer that allows users to define their big data operations in a declarative way. This abstraction layer is backed by a platform that optimizes the workflow by carefully profiling each operation and running them, transparently, on the most appropriate framework given the description of the job and the currently available resources.

1 Introduction

For a time, it looked like the world would settle on Apache Hadoop for all its big data needs, but the Hadoop project has been split, and while its data storage part, HDFS, has indeed become the lingua franca of big data processing frameworks, its processing engine, Hadoop MapReduce, has seen a lot of competition lately. New frameworks such as Apache Spark (see Zaharia et al. (2010)), Apache Tez (based on Verma et al. (2011)), Apache Flink (continuation of Warneke and Kao (2009)), or Apache Hive offer new possibilities, along with tradeoffs and challenges.

In this work, we build an abstraction layer that allows users to define their big data operations in a declarative way. This abstraction layer is backed by a platform that optimizes the workflow by carefuly profiling each operation and running them, transparently, on the most appropriate framework given the description of the job and the currently available resources.

In Section 2, we describe the high-level API exposed to users and the general principles behind it. In Section 3, we describe the architecture of the underlying platform. Section 4 presents our demonstration scenario, and, finally, Section 5 concludes.

2 High-Level API

We expose an API rather than a GUI for our platform. This has multiple advantages: we can focus on the platform itself, support multiple clients in the future, and allow other programs to interact with the platform. These other programs may be IDEs, including textual and graphical DSL frontends.

The API is based upon the REST principles, as described in Fielding (2000). The central resource type is a dataset, which presents the user with metadata about an actual dataset present

```
{"schema": [{"name": "id", "type": "string"}, {"name": "age", "type": "number"},
            {"name": "hair-color", "type": "string"},
            {"name": "height", "type": "number"}],
 "id": "initial-dataset",
 "_controls":
 {"op:remove-cols": {"href": "/some/other/url", "method": "POST",
                     "template": {"name": "remove-cols",
                                  "datasets": ["initial-dataset"]}},
                                  "@columns": {"type": "array",
                                               "items": {"type": "integer"}}}}
```

FIG. 1 – *An example of (a subset of) a dataset representation in JSON*

```
{"name": "remove-cols", "datasets": ["initial-dataset"], "columns": [0, 2]}
```

FIG. 2 – *An example request to remove two columns from the dataset in Figure 1*

on the server. The actual contents of the dataset is not, in the general case, available for download, as we assume that the datasets are too big for the client to handle. An important part of the metadata available about a given dataset is the list of operations that the client can request on that dataset. This list includes all of the information the client needs to submit a well-formed request for each operation. The server is therefore responsible for determining which operation can be applied on which dataset.

Whenever the client requests that an operation be performed on a dataset, the result is the creation of a new resource. For most operations, that new resource is itself a dataset, which can be used in other operations, building a graph of datasets. For some operations, the result is a model, which is a computational entity of which the client can subsequently ask questions and get responses. The new resource is immediately resolvable and can return some of its metadata even before the computation starts on the cluster. This allows the client to quickly chain operations together.

An example usage of the API will yield more insight. The examples are given in JSON (see Crockford (2006)), and are somewhat abridged to fit in the constraints of this paper. Full representations contain additional information. We start with an initial dataset, as represented on Figure 1. Without going into the details of the representation, it should be clear that the representation advertizes the available operations and how to request them.

The client can then issue a request as represented on Figure 2. The response from the server is a new dataset representation, similar in format to Figure 1. The cycle can continue until the client is satisfied with the result. The list of supported operations is constantly evolving; we plan to support operations ranging from the usual data cleaning and extraction found in typical ETL tools to advanced machine learning. We have a team of data scientists working on new, state-of-the-art algorithms for automatic feature extraction and deep learning that we plan to incorporate within the platform.

3 Underlying Platform

Within the underlying platform, requested operations are placed on a queue. An orchestrator accesses that queue and processes operations in the correct order given their interdependencies. For efficiency reasons, all jobs run on a single, shared cluster. Indeed, if at a given point in time Spark is not running at all, and Hive is consuming all of its available resources, in a typical setup it is not easy at all to redirect some of the Spark machines to the Hive cluster. With our unified approach, resources are dynamically allocated to each framework by a single orchestrator.

From a functional perspective, the orchestrator has two objectives: starting jobs in the correct order, given that there are dependencies between jobs, and ensuring that jobs can be piped together even when they run on different frameworks.

From a performance perspective, the orchestrator can do quite a lot more. For some operations, there can be multiple implementations with different resource usage patterns, and the orchestrator can choose the best one given the current circumstances: which nodes are available, where the required data is, how much data there is, and so on. The orchestrator can also analyze the graph of operations present on the queue and eliminate or reorder some of them, as long as it does not affect the results of the user-visible results (datasets for which the user can request full data, and models). This includes identifying jobs that have already been computed, and reusing their results.

As the ODAP platform will be installed in very different environments, and any given installation could evolve dramatically over time, it is very important that the platform can dynamically adapt to its running environment. To that end, all operations will be profiled and the above optimizations will be based on cost estimates derived from real, recent measurements on the actual environment. We strive for minimal manual tuning.

Finally, to make it easy to deploy our platform, it is built to run within portable, self-contained Docker containers. This makes it quick and easy to setup new nodes and add them to the cluster, and just as easy to remove the platform from a machine after installation.

4 Scenario and Implementation

The whole system is built atop the Java platform, the same as Hadoop (from which we reuse Yarn and HDFS) and the vast majority of the supported frameworks. We use ZooKeeper (see Hunt et al. (2010)) to record the distributed state of the application (job queue and list of existing datasets). The API server is completely stateless. The orchestrator is mostly stateless: its only internal state is the communication channels it has with the currently running jobs. In other words, a crash of the orchestrator can, at worst, necessitate that the jobs that were currently running have to run again. The jobs themselves are not taken out of the queue before they are fully completed, so even if the whole Yarn subsystem crashes, no job definition is lost. ZooKeeper itself is distributed and fault-tolerant.

For the demonstration, the use-case will be what we call a billboard optimization problem: given a set of cities, a set of users, a set of products, and a marketing budget, decide in which city each product should be advertized. The problem combines general data cleaning (extract the price paid by each user for each service), traditional data mining (recommendation to estimate how much each person would pay for a service he does not have yet), and optimization (find a good placement of products in cities).

5 Conclusion and Future Work

We have built, and presented, a platform to unify big data applications, particularly data mining and machine learning. The platform dynamically optimizes its workload, taking into account its available resources. It unifies different frameworks, making different types of operations available, on a single data lake, increasing resource utilization. Finally, this platform presents a high-level, declarative interface shielding the user from the differences between frameworks.

References

Crockford, D. (2006). The application/json media type for javascript object notation (json). RFC 4627, IETF.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Ph. D. thesis. AAI9980887.

Hunt, P., M. Konar, F. P. Junqueira, and B. Reed (2010). Zookeeper: Wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, Berkeley, CA, USA, pp. 11–11. USENIX Association.

Verma, A., L. Cherkasova, and R. H. Campbell (2011). Aria: Automatic resource inference and allocation for mapreduce environments. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, New York, NY, USA, pp. 235–244. ACM.

Warneke, D. and O. Kao (2009). Nephele: Efficient parallel data processing in the cloud. In *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*, MTAGS '09, New York, NY, USA, pp. 8:1–8:10. ACM.

Zaharia, M., M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica (2010). Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pp. 10–10. USENIX Association.

Résumé

Des outils tels que Spark, Tez, Flink ou Hive offrent de nouvelles possibilités, mais il est souvent difficile d'utiliser plusieurs de ces outils en même temps. Nous présentons une couche d'abstraction qui permet aux utilisateurs de définir, de manière déclarative, des opérations sur de grandes quantités de données. Cette couche d'abstraction cache une plateforme intelligente qui optimise les opérations demandées en choisissant les outils appropriés, en tenant compte des caractéristiques des opérations demandées et des ressources disponibles.

Résumé

L'entreposage de données et l'analyse en ligne se sont imposées comme des outils fondamentaux et incontournables de l'informatique décisionnelle. Elles sont aujourd'hui confrontées à de nouveaux défis scientifiques qui apparaissent avec la prolifération de nouveaux types de données, de nouvelles architectures et infrastructures, etc. La conférence francophone sur les entrepôts de données et l'analyse en ligne EDA, arrivée à sa onzième édition, vise à créer un contexte pour la rencontre et l'échange entre chercheurs, industriels et utilisateurs intéressés par les avancées dans les entrepôts de données et l'analyse en ligne. Le présent recueil constitue les actes du colloque EDA 2015, qui s'est déroulé à Bruxelles, Belgique, les 2 et 3 avril 2015, avec un programme comprenant douze présentations scientifiques, deux conférences invitées, et des démonstrations scientifiques.

Summary

The data warehousing and on-line analysis processing (OLAP) technologies have emerged as fundamental and indispensable tools for business intelligence. They are now confronted with new scientific challenges that appear with the proliferation of new data types, new architectures and infrastructures, etc. The French conference on data warehousing and on-line analysis EDA, which reached its eleventh edition, aims to create a context for the meeting and exchange between researchers, industry and users interested in advances in data warehouses and on-line analysis. This book constitutes the proceedings of the EDA 2015 symposium, which was held in Bruxelles, Belgium, on April 2–3, 2015, with a program consisting of twelve scientific presentations, two invited lectures and scientific demonstrations.

Rédacteurs invités

Esteban Zimányi is a professor and a director of the Department of Computer and Decision Engineering (CoDE). He started his studies at the Universidad Autónoma de Centro América, Costa Rica. He received a B.Sc. (1988) and a doctorate (1992) in computer science from the Faculty of Sciences at the ULB. His current research interests include data warehouses and business intelligence, semantic web and web services, geographic information systems and spatio-temporal databases. He has co-authored and co-edited 8 books and published more than 100 peer-reviewed papers on these topics. He is Editor-in-Chief of the Journal on Data Semantics (JoDS) published by Springer. He is the coordinator of the Erasmus Mundus Master and the Erasmus Mundus Doctorate "Information Technologies for Business Intelligence" (IT4BI). He has served as PC Chair for DOLAP 2009, General Chair for ER 2011, and on numerous program committees, including VLDB, EDBT, ACM GIS, and DaWaK. He is a member of the ER Steering Committee.

Toon Calders graduated in 1999 from the University of Antwerp with a diploma in Mathematics. He received his PhD in Computer Science from the same university in May 2003, in the database research group ADReM, and continued working in the ADReM group as a postdoc until 2006. From 2006 till 2012 he was an assistant professor in the Information Systems group at the Eindhoven Technical University. In 2012 he joined the CoDE department at the ULB as associate professor. His main research interests include data mining and machine learning. Toon Calders published over 60 conference and journal papers in this research area and received several scientific awards for his works, including the recent "10 Year most influential paper" award for papers published in ECMLPKDD 2002. Toon Calders regularly serves in the program committees of important data mining conferences, including ACM SIGKDD, IEEE ICDM, ECMLPKDD, SIAM DM, was conference chair of the BNAIC 2009 and EDM 2011 conferences, and is a member of the editorial board of the Springer Data Mining journal and Area Editor for the Information Systems journal.

Stijn Vansummeren is a professor at the CoDE Department of the ULB, where he teaches databases and web technologies. He received a Bachelor in Computer Science (1999) from the Faculty of Sciences at Hasselt University, Belgium; a Master in Computer Science (2001) from the Faculty of Sciences at the University of Maastricht, the Netherlands; and a PhD in Science from Hasselt University, Belgium. His research work covers all aspects of data management, both traditional on the Web. His recent work focuses on aspects of provenance in scientific databases; database programming languages; the foundations of query languages for XML and RDF; and query processing on the Web.