Mosaics in Big Data

Stratosphere, Apache Flink, and Beyond

Volker Markl
Database Systems and Information Management (DIMA) Group at the Technische Universität Berlin (TUB),

The Intelligent Analytics for Massive Data Department at the German Research Center for Artificial Intelligence (DFKI)
Berlin, Germany
volker.markl@tu-berlin.de

ABSTRACT

The global database research community has greatly impacted the functionality and performance of data storage and processing systems along the dimensions that define "big data", i.e., volume, velocity, variety, and veracity. Locally, over the past five years, we have also been working on varying fronts. Among our contributions are: (1) establishing a vision for a database-inspired big data analytics system, which unifies the best of database and distributed systems technologies, and augments it with concepts drawn from compilers (e.g., iterations) and data stream processing, as well as (2) forming a community of researchers and institutions to create the Stratosphere platform to realize our vision. One major result from these activities was Apache Flink, an open-source big data analytics platform and its thriving global community of developers and production users. Although much progress has been made, when looking at the overall big data stack, a major challenge for database research community still remains. That is, how to maintain the ease-of-use despite the increasing heterogeneity and complexity of data analytics, involving specialized engines for various aspects of an end-to-end data analytics pipeline, including, among others, graph-based, linear algebra-based, and relational-based algorithms, and the underlying, increasingly heterogeneous hardware and computing infrastructure. At TU Berlin, DFKI, and the Berlin Big Data Center (BBDC), we aim to advance research in this field via the Mosaics project. Our goal is to remedy some of the heterogeneity challenges that hamper developer productivity and limit the use of data science technologies to just the privileged few, who are coveted experts.

CCS CONCEPTS

• **Information Systems → Data Management Systems → Database Management System Engines → Database query processing** → Query optimization; Query operators • **Information Systems → Data Management Systems → Database Management System Engines** → Stream management; Online analytical processing engines; DBMS engine architectures

KEYWORDS

Apache Flink, big data, data science, declarative languages, federation, heterogeneous data management

DEBS '18, June 25 - 29, 2018, Hamilton, New Zealand.
© 2018 Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-5782-1/18/06…$15.00
https://doi.org/10.1145/3210284.3214344

1 INTRODUCTION

http://REPLACE_WITH_YOUR_FULL_DOI_URL % Use the data from your email from ACM rights managementOne can conduct research in data management along three major dimensions, namely, adding new functionality (e.g., designing a new page rank or Internet search algorithm), improving performance (e.g., developing novel join methods or devising ingenious methods for in-memory processing), and furthering ease-of-use (e.g., increasing developer productivity and reducing costs). The field of relational data management has been a boon for increasing developer productivity and safeguarding the developer from the complexity of the system (e.g., by automating transaction management or automating optimization using declarative languages that exploit well-defined, closed models for query formulation). This in turn has resulted in widespread adoption of data management in specialized applications, such as data warehousing and visual analytics, among others. The advent of even more advanced analytical methods has put relational data management to a test. Over the past decade, data management has steadily grown in complexity. Scientific institutions and enterprises, among others, are actively collecting vast volumes of highly diverse data, often with high ingestion rates. They are also building novel analytics that draw on theory and best practices in relational database management, graph analysis, machine learning, signal processing, statistical science, and mathematical programming. This heterogeneity of analytics has spurred the development of a diverse ecosystem of specialized data analytics engines, each tailored to a specific paradigm and use case. Examples of such engines include:

relational database systems (e.g., Postgres, MySQL, MonetDB), tools for numerical analysis (e.g., MATLAB, R, NumPy), emerging distributed data processing engines (e.g., Hadoop, Spark, Flink), distributed key-value stores (e.g., HBase, Cassandra), as well as specialized graph processing systems (e.g., Neo4J, Giraph, GraphLab [1]). Each of these engines has specific advantages and disadvantages. However, picking the right one – or the right combination – for a given problem can be daunting. In addition, we are observing an increase in the diversification of the hardware landscape, promising to improve data processing performance. First, processors are increasingly moving towards heterogeneous configurations that combine diverse architectures [2] (e.g., CPUs, GPUs, vector processors, FPGAs). Second, the availability of fast, high-capacity flash storage. Third, the emergence of non-volatile memory technology disrupting the traditional memory hierarchy [3]. And fourth, the continued evolution of network interconnects. Furthermore, the growing number of available hardware virtualization and infrastructure-as-a-service solutions implies that specially-tailored hardware configurations will now be readily available to basically anyone at the click of a button. This increase in variety will make it far more difficult to identify the hardware configuration that exploits hardware properties optimally for a target problem. The growing heterogeneity at the *problem*, *system*, and *hardware* level is making efficient data analysis increasingly formidable. Specifying and tuning data analysis programs (DAPs) require analysts to manually find the optimal combination of programming models, runtime engines, and hardware configurations from a vast number of possible alternatives.

Today's data analysts must be a "jack-of-all-trades." That is, be proficient in a multitude of different systems and languages, comprehend data and processing models, grasp the intricacies of tuning parameters and their corresponding performance impact, and be able to map a given analysis task to the ideal combination of systems with the most effective hardware configuration. This rare combination of skills (identified by both Accenture [4] and McKinsey [5] market researchers) is one of the key reasons behind the severe shortage of capable data scientists. Reducing the complexity of the data analysis process, the entry barrier, and the cost of analyzing large amounts of data at scale is one of the most important goals in data management research today. The only reasonable way to reduce this complexity is to automate the manual design and implementation choices data scientists regularly face in this heterogeneous environment (e.g., identifying the algorithms, system components, and tuning parameters).

Today, end-to-end optimization of complex data analysis, machine learning or AI pipelines consisting of data preparation (e.g., ELT/ETL), model building, and model application is limited to certain subdomains (e.g., optimizing relational queries in Hadoop/Hive [6] or Spark [7], MapReduce models (e.g., Manimal [8], Stratosphere/Flink [29, 30]), or higher order functions (e.g., PACT [31], Musketeer [35]). However, a holistic declarative specification of data analytics programs in a single language and automatic optimization, distribution, parallelization, and hardware adaptation of the entire data analysis pipeline based on a principled algebraic model is still missing. As we have pointed out in a vision paper [9], this deficiency effectively means that systems and platforms for big data management are in a state that database systems were prior to the 1980s: complex systems that are solely usable by experts who understand systems programming and tuning. Relational database systems have become the standard workhorse of data management in a multibillion dollar market with a tool ecosystem (e.g., business intelligence applications, data cubes, visual analytics tools) because Ted Codd invented relational algebra [10], as a principled underlying mathematical foundation resulting in declarative languages [11,12] with optimizing transformation rules and query optimizers [13] drawn from the fields of compilers, statistics, and artificial intelligence. The success of this approach ensured that database programmers and tool builders were relieved from the duty of having to worry about the efficient execution of database operations.

2 STRATOSPHERE & APACHE FLINK IN THE BIG DATA SPACE

Many specialized systems have been developed in order to analyze high-volume, high-velocity, high variety data efficiently and effectively, to take advantage of specialized algorithm implementations for problem domains such as linear algebra, natural language processsing, or graph analytics, often in conjunction with modern storage and processing architectures, such as NUMA or heterogeneous CPUs. This diversity in systems and hardware has greatly improved functionality and performance for many data analytics applications. Our contribution to this set of systems has been Apache Flink, which has come out of the Stratosphere research project. Many concepts of Stratosphere, such as the stream processing paradigm, the support for iterations, the query optimizer, among others, have been carried over to Flink [18, 30, 36, 37], while others, such as optimistic fault tolerance [40] have remained experimental features.

2.1 Stratosphere and the Origins of Flink [39]

The origins of Apache Flink can be traced back to 2008, when the author initially founded the Database Systems and Information Management (DIMA) Group at the Technische Universität (TU) Berlin. Soon after his arrival, he laid out the vision for a massively parallel batch data processing system based on post-relational user-defined functions, combining database and distributed systems concepts, with the goal of enabling modern data analysis and machine learning for big data.

The author's PhD students Stephan Ewen and Fabian Hüske built the very first prototype, teaming up with Daniel Warneke, a PhD student in Odej Kao's Complex and Distributed IT Systems (CIT) Group at TU Berlin. Soon after, this core team sought to collaborate with additional systems researchers in the greater Berlin area, in order to extend, harden, and validate their initial prototype.

In 2009, the author, jointly with researchers from TU Berlin, Humboldt University (HU) of Berlin and the Hasso Plattner Institute (HPI) in Potsdam, co-wrote a DFG (German Research Foundation) research unit proposal entitled "Stratosphere – Information Management on the Cloud [18]," which was funded in 2010. This initial DFG grant (spanning 2010-2012) extended the original vision to develop a novel, database-inspired approach to analyze, aggregate, and query very large collections of either textual or (semi-) structured data on a virtualized, massively parallel cluster architecture.

The follow-on DFG proposal entitled, "Stratosphere II: Advanced Analytics for Big Data" was also jointly co-written by researchers at TU Berlin, HU Berlin, and HPI and was funded in 2012. This second DFG grant (spanning 2012-2015) shifted the focus towards the processing of complex data analysis programs with low-latency. These early initiatives coupled with grants from the EU FP7 and Horizon 2020 Programmes, EIT Digital, German Federal Ministries (BMBF and BMWi), and industrial grants from IBM, HP, and Deutsche Telekom, among others provided the financial resources necessary to lay the initial foundation for what was later to become Apache Flink.

Certainly, funding plays a critical role, however, success could only be achieved with the support of numerous collaborators, including members at SICS (The Swedish Institute of Computer Science), SZTAKI (The Hungarian Academy of Sciences), DFKI (The German Research Centre for Artificial Intelligence), among many others who believed in our vision, contributed, and provided support over the years. The contributions from numerous students and researchers at TU Berlin paved the way for what is today Apache Flink, in particular Ufuk Celebi, Stephan Ewen, Fabian Hüske, Aljoscha Krettek, Robert Metzger, Till Rohrmann, and Kostas Tzoumas. A Stratosphere fork that became an Apache Incubator Project in March 2014 and then went on to become an Apache Top-Level Project in December 2014.

Since 2014, the Apache Flink Community has experienced exponential growth. As of January, 2018 there are more than 350 contributors (as reflected in github.com/apache/flink), more than 42 Meetups worldwide (meetup.com/topics/apache-flink/), and over 20000 participants in Apache Flink-related Meetup worldwide (see Figure 1). An important milestone for Flink was the adoption by major companies, such as Researchgate, Google, Zalando, AliBaba, Uber, Netflix [34].

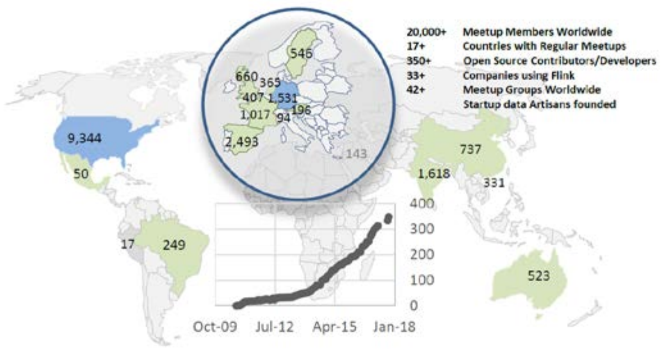
20,000+ Meetup Members Worldwide
17+ Countries with Regular Meetups
350+ Open Source Contributors/Developers
33+ Companies using Flink
42+ Meetup Groups Worldwide
Startup data Artisans founded
546
660
365
407
1,531
1,017
196
94
2,493
143
9,344
50
737
1,618
331
17
249
523
400
300
200
100
0
Oct-09
Jul-12
Apr-15
Jan-18

Figure 1: The Flink Community

In late 2014, Kostas Tzoumas and Stephan Ewen, along with the author and several further creators of the Apache Flink project from the Database Systems and Information Management group at TU Berlin founded data Artisans, a company focused on making Flink the next-generation open source platform for programming data-intensive applications. In summer 2014, data Artisans started with a seed financing round and raised a Series A round in April 2016. Since the company was founded, many team members from data Artisans have become active contributors to Apache Flink.

Collectively, these efforts showcase the path from a research idea to an open source software system that is in use across many companies, software projects, universities, and research institutions worldwide. Apache Flink is today one of the most active open source projects in the Apache Software Foundation with users in academia and industry, as well as contributors and communities all around the world.

2.2 Features and The Current State of Apache Flink [39]

The official Apache Flink project [18], describes Flink as follows: "Apache Flink is an open source platform for distributed stream and batch data processing. Flink's core is a streaming dataflow engine that provides data distribution, communication, and fault tolerance for distributed computations over data streams." In particular, Flink provides "results that are accurate, even in the case of out-of-order or late-arriving data; is stateful and fault-tolerant and can seamlessly recover from failures while maintaining exactly-once application state; and performs at large scale, running on thousands of nodes with very good throughput and latency characteristics."

The platform offers software developers varying application-programming interfaces (APIs), to create new applications to be executed on the Flink engine." Examples of these APIs include: an API for unbounded (data) streams (DataStream API), a complex event-processing library (CEP), a machine-learning library, and a graph-processing library (Gelly), among others. Some of Flink's distinguishing technological capabilities include "event time handling, state, and fault tolerance, low latency processing, and high throughput" [37, 18].

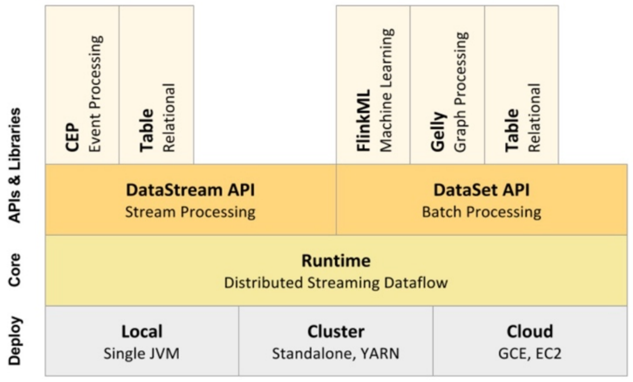

Figure 2. An overview of Apache Flink components [18]

According to an Apache Flink Wikipedia article [22], "Apache Flink is a community-driven open source framework for distributed big data analytics. … [that] aims to bridge the gap between MapReduce-like systems and shared-nothing parallel database systems. … Flink's pipelined runtime system enables the execution of bulk/batch and stream processing programs. Furthermore, Flink's runtime supports the execution of iterative algorithms natively. Flink programs […] are automatically compiled and optimized into dataflow programs that are executed in a cluster or cloud environment."

In particular, Flink offers the following features as discussed on the Flink official website [16]:

- "Flink allows for *stateful computations*. 'Stateful' means that applications can maintain an aggregation or summary of data that has been processed over time, and Flink's checkpointing mechanism ensures exactly-once semantics for an application's state in the event of a failure.

- Flink supports *stream processing and windowing with event time semantics*. Event time makes it easy to compute accurate results over streams where events arrive out of order and where events may arrive delayed.

- Flink supports *flexible windowing based on time, count, or sessions in addition to data-driven windows*. Windows can be customized with flexible triggering conditions to support sophisticated streaming patterns. Flink’s windowing makes it possible to model the reality of the environment in which data is created.

- Flink’s *fault tolerance is lightweight* and allows the system to maintain high throughput rates and provide exactly-once consistency guarantees at the same time. Flink recovers from failures with zero data loss while the tradeoff between reliability and latency is negligible.
- Flink’s *savepoints provide a state versioning mechanism*, making it possible to update applications or reprocess historic data with no lost state and minimal downtime.
- Flink is designed to *run on large-scale clusters with many thousands of nodes*, and in addition to a standalone cluster mode, Flink provides support for YARN and Mesos.
- *Batch programs are automatically optimized* to exploit situations where expensive operations (like shuffles and sorts) can be avoided, and when intermediate data should be cached.
- *One Runtime for Streaming and Batch Processing*. Flink uses one common runtime for data streaming applications and batch processing applications. Batch processing applications run efficiently as special cases of stream processing applications.
- *Continuous Streaming Model with Backpressure*. Data streaming applications are executed with continuous (long lived) operators. Flink's streaming runtime has natural flow control: Slow data sinks backpressure faster sources.
- *Fault-tolerance via Lightweight Distributed Snapshots*. Flink's fault tolerance mechanism is based on Chandy-Lamport distributed snapshots. The mechanism is lightweight, allowing the system to maintain high throughput rates and provide strong consistency guarantees at the same time.
- • *High Performance & Low Latency*. Flink's data streaming runtime achieves high throughput rates and low latency with little configuration. The charts below show the performance of a distributed item counting task, requiring streaming data shuffles.
- *Memory Management*. Flink implements its own memory management inside the JVM. Applications scale to data sizes beyond main memory and experience less garbage collection overhead.
- *Iterations and Delta Iterations*. Flink has dedicated support for iterative computations (as in machine learning and graph analysis). Delta iterations can exploit computational dependencies for faster convergence.
- *Streaming Data Applications*. The DataStream API supports functional transformations on data streams, with user-defined state, and flexible windows.
- *Batch Processing Applications*. Flink's DataSet API lets you write type-safe and maintainable code in Java or Scala. It supports a wide range of data types beyond key/value pairs, and a wealth of operators.
- *Library Ecosystem*. Flink's stack offers libraries with high-level APIs for different use cases: Complex Event Processing, Machine Learning, and Graph Analytics.
- *Broad Integration*. Flink is integrated with many other projects in the open-source data processing ecosystem. Flink runs on YARN, works with HDFS, streams data from Kafka, can execute Hadoop program code, and connects to various other data storage systems.”

3 MOSAICS

Today, one of the grand challenges in data management research is to reduce both the entry barrier and cost of analysing large amounts of data-at-scale by simplifying the data analysis process. In order to accomplish this, we will need to automate the design and implementation decisions that data scientists routinely make in heterogeneous environments based on varied theories, systems, and hardware-based solutions. Enabling such automation is the holy grail of data science. We can meet this challenge if we combine several data processing technologies established by the scientific and systems community. However, the major road-block for automation is that there is no principled model for scalable data science systems akin to relational algebra in database systems.

The principal goal of our research is to provide a declarative, algebraic, and optimizable representation for the entire data analysis process. At the data management group of TU Berlin, we aim to achieve this by integrating the disparate hardware and software components present in today’s data analysis architectures into a **unified mosaic of systems and/or hardware devices**, orchestrated by a **unifying mosaic of theories** that describe data analytics based on graphs, matrices, or relations. By soundly reasoning about these mosaics, we strive to: (i) significantly simplify the specification of data analysis problems, (ii) enable automatic optimization, distribution, parallelization, and hardware adaptation across the hodgepodge of heterogeneous computing systems and storage architectures, and (iii) improve scalability in both infrastructure, and people, by reducing system and human latency, respectively.

In order to achieve such a federated mosaic on the theory, systems, and hardware level, we propose to use algebraic theories across graph algorithms, linear algebra, and nested collections, jointly with monads as a promising basis for a unifying theory. This will enable us to overcome the heterogeneity in the model space and identify equivalences and morphisms across theories. It also serves as theoretical foundation to describe and reason about optimizing transformations on the model, engine, and hardware device space in a principled way. Our overall objective can be decomposed into six sub-goals of our research, as follows:

(1) Unifying Modelling Across Theories. One desiderata is a unifying model for heterogeneous data processing at scale that can serve as a foundation for systematic reasoning and aligned research with focus on program optimization. Here one could build past works [19, 20] and base a unifying model on ideas from the fields of algebraic specification and model theory. In particular, one could develop algebras equipped with equational theories for the three most commonly used data representations, namely, collections, graphs, and matrices. Rather than aiming for full characterization of these types in terms of initial semantics, however, one probably should strive to impose type structure from a computational perspective, such that a broad class of programs and program optimizations can be defined using structural recursion (i.e., the concept of folds [25]). To that end, data management researchers can build upon their experience and expertise in database and dataflow engine design. In contrast to previous works [19, 20, 23], one would need to tackle all three data types jointly and investigate their relationships. The proposed formal framework should allow for defining and applying optimizations across theories (e.g., fusion

for pipelined execution or unnesting transformations for improved data parallelism).

(2) Cross Theory Optimization. Current research in data analysis heavily relies on the equivalence among fragments of the programming abstractions available for collections, graphs, and linear algebra analytics (e.g., graph computations represented as linear algebra programs). This equivalence has not been studied in a principled way, which makes the problem of designing rule-based program optimizations across theories extremely hard. Moreover, data type modelling and theory development have often been motivated by the primary goal of characterizing complexity. In contrast to that, one could strive to identify a class of interesting optimizations, which can be seen in different specialized systems on the one side, but are obviously related on the other. These insights should be the driving force behind the modelling part, such that the outcome can explain these optimizations in a single theoretical framework (e.g., based on structural recursion, monads, and functors). Once this framework is in place, equivalences between models could be explored in a principled way. In conjunction with the unified set of optimizations, this would aid the development of a joint search space and rule-based search strategies for holistic optimization and compilation of heterogeneous data analysis pipelines.

(3) Optimizing Across Engines. In a mosaic of federated engines, each engine may provide multiple implementations of a logical operator, each with different execution strategies, data input assumptions, and costs. Therefore, one must translate data analysis programs (DAPs) into optimized execution plans, which possibly span multiple engines. To this end, one needs to define a physical execution model, which encodes the processing and storage capabilities of each engine. This execution model should be designed to be generic enough to allow for physical optimizations, which take into account (i) each engine's processing and storage capabilities, (ii) execution and data exchange strategies, as well as (iii) physical properties of the data that the engines assume as inputs and produce as outputs. The optimal execution strategy for DAPs is usually determined and scheduled for execution once, at compile time (i.e., before the execution starts). However, the conditions, which led to the selection of a certain strategy are not static. In particular, the cost of operations may change due to erroneous predictions or assumptions (e.g., poor estimates of intermediate result sizes or wrong assumptions about the underlying data distributions) or dynamic changes in the execution environment (e.g., less memory available during execution time than assumed during optimization due to further programs running concurrently). In order to avoid suboptimal strategies, we will have to introduce a feedback loop into the mosaic's optimizer, which monitors the execution and reacts by adapting the execution strategy whenever the original assumptions about the data, execution environment, or concurrent workload do not hold anymore and a better strategy exists.

(4) Predicting and Learning Program Runtimes. In order to effectively explore the vast plan search space across theories, engines, and hardware and determine when to apply which optimizing transformation, our field will require novel cost estimation models to automatically predict the runtime of arbitrary segments of an execution graph. These models must be able to capture aspects of the underlying engine (e.g., "How fast does engine X run operation Y on data with property Z?"), as well as aspects of the user-code (e.g., "How much time does function A spend on a data item, given it has properties B and C?"), and aspects of the hardware (e.g., "How would increasing clock frequency by X change the expected runtime?").

(5) Optimizing Across Hardware. Fully exploiting the hardware diversity for data processing, while reducing the complexity for the developer will require data management researchers to build an optimizer with capabilities for hardware-dependent decision-making. In particular, this will require research along two primary directions: (i) automatic selection of the right devices to be involved in processing, i.e., choosing the most economic configuration among the available ones with regard to a given metric and subject to a given budget, and (ii) (dynamically) adjusting the execution plan to match the specifics of the underlying hardware (e.g., switching to single-node algorithms that exploit shared memory or employing GPU-accelerated methods). One could start with prior work on hardware-specific optimization (e.g., [24, 25]) and related work in the relational world, and advance it to the richer model of DAPs across different theories for heterogeneous hardware devices.

(6) Generating Hardware-Targeted Code. Modern data processing frameworks primarily target distributed environments, putting scalability ahead of the efficient exploitation of all available resources [26]. This has two undesirable consequences. First, for problems that fit onto a single machine, these frameworks add unnecessary overhead and slow computations down [27]. Second, existing technology cannot easily exploit emerging, highly heterogeneous hardware architectures, like servers with GPUs and FPGAs, resulting in valuable resources remaining unused [28]. One could tackle these issues by investigating the option to bypass existing engines and directly generate hardware-targeted code for selected operations. To achieve this, one needs to develop novel code-generation methods that are able to: (a) target arbitrary hardware architectures by following a "hardware-oblivious" design approach [31, 38] and (b) automatically learn hardware-specific implementation strategies by observing the behavior of generated code and adjusting the code generation based on these observation.

Our first result in the context of Mosaics of Theories and Systems is Emma [32, 33], where we achieved implicit parallelism through deep language embedding, by leveraging comprehensions and translating Scala programs to a big data analytics platform without developers having to worry about the API of that platform (e.g., Flink or Spark). Other results include LARA, a vision for combining relational and linear algebra and executing optimizing transformations on these [41], and physical operators such as Blockjoin [42] that bridge the relational and matrix representations. In the context of the Mosaics of Hardware, we are working on Hawk, a hardware-adaptive query compiler for heterogeneous hardware.

ACKNOWLEDGMENTS

I would like to thank the members of the Stratosphere project, the Berlin Big Data Center, and my national and international collaborators. In particular, I would like to thank my former and current students and postdocs at DIMA/TU Berlin and DFKI. Without them it would not have been possible to realize the visions into concrete software system artifacts. Last, but not least, I would like to thank all of the contributors and users in the Apache Flink community, without them the Stratosphere project and the correlated research of the Berlin Big Data Center would not have achieved the worldwide impact that we have experienced over the last few years.

This work was supported by grants provided by the German Research Foundation (DFG), via FOR 1306 Stratosphere – and – the German Federal Ministry for Education and Research (BMBF), via ref. 01IS14013A, which enabled us establish the Berlin Big Data Center (BBDC).

REFERENCES

[1] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson and C. Guestrin, "PowerGraph: Distributed Graph-Parallel Computation.," in *USENIX*, Berkeley, 2012.

[2] S. Borkar and A. Chien, "The future of microprocessors," *Communications of the ACM,* vol. 54, no. 5, pp. 67-77, 2011.

[3] A. M. Caulfield, J. Coburn, T. Mollov, A. De, A. Akel, J. He, A. Jagatheesan, R. K. Gupta, A. Snavely and S. Swanson, "Understanding the Impact of Emerging Non-Volatile Memories on High-Performance, IO-Intensive Computing," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis,* Washington DC, 2010.

[4] J. G. Harris, N. Shetterley, A. Alter and e. al, "The Team Solution to the Data Scientist Shortage.," Accenture IfHP, 2013.

[5] J. Manyika and e. al, "Big data: The next frontier for innovation, competition and productivity.," McKinsey GI, 2011.

[6] A. Thusoo, S. S. J., N. Jain and e. al, "Hive - a petabyte scale data warehouse using Hadoop.," in *ICDE* , 2010.

[7] M. Armbrust, R. S. Xin, C. Lian and e. al, "Spark SQL: Relational Data Processing in Spark," in *SIGMOD* , 2015.

[8] E. Jahani, M. J. Cafarella and C. Ré, "Automatic Optimization for MapReduce Programs.," in *PVLDB 4*, 2011.

[9] V. Markl, "On Declarative Analysis and Data Independence in the Big Data Era.," in *PVLDB*, 2014.

[10] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks.," *CACM,* vol. 13, no. 6, pp. 377-387, 1970.

[11] D. D. Chamberlin and R. F. Boyce, "SEQUEL: A Structured English Query Language.," in *SIGMOD*, 1974.

[12] D. D. Chamberlin, A. M. Gilbert and R. A. Yost, "A History of System R and SQL.," in *VLDB*, 1981.

[13] P. G. Selinger and e. al, "Access Path Selection in a Relational Database Management System.," in *SIGMOD* , 1979.

[14] *Why Apache Beam?,* http://data-artisans.com/why-apache-beam/.

[15] "Stratosphere," [Online]. Available: http://stratosphere.eu.

[16] "Apache Flink," [Online]. Available: http://flink.apache.org.

[17] "Unified Stream & Batch Processing with Apache Flink," [Online]. Available: youtu.be/8Uh3ycG3Wew.

[18] "Apache Flink Article," [Online]. Available: https://en.wikipedia.org/wiki/Apache_Flink.

[19] P. P. Buneman and e. al, "Programming with Complex Objects and Collection Types.," *Theor. Comp. Sci.* , vol. 149, no. 1, pp. 3-48 , 1995.

[20] T. Grust, " Comprehending queries.," Universität Konstanz, 1999.

[21] G. Hutton, "A tutorial on the universality and expressiveness of fold.," *J. of Funct. Programming,* vol. 9, no. 4, 1999.

[22] P. P. Buneman and e. al, "Programming with Complex Objects and Collection Types.," *Theor. Comp. Sci.,* vol. 149, no. 1, pp. 3-48 , 1995.

[23] S. Bird, P. Buneman and W. C. Tan, "Towards a query language for annotation graphs.," CoRR cs.CL/0007023, 2000.

[24] M. Heimel, V. Markl and e. al, "Hardware-Oblivious Parallelism for In-Memory Column-Stores.," in *PVLDB 6(9)*, 2013.

[25] M. Heimel, M. Kiefer and V. Markl, "GPU-Accelerated KDE Models for MD Selectivity Estimation," in *SIGMOD* , 2015.

[26] A. Crotty, A. Galakatos, K. Dursun and e. al, ""Big" Data, Big Analytics, Small Clusters," in *CIDR* , 2015.

[27] F. McSherry, M. Isard and D. G. Murray, " "Scalability! But at what COST?"," in *HotOS XV*, 2015.

[28] D. Broneske, S. Breß, M. Heimel and G. Saake, "Toward Hardware-Sensitive Database Operations," in *EDBT* , 2014.

[29] W. Han, W. Kwak, J. Lee, G. M. Lohman and V. Markl, "Parallelizing Query Optimization," in *PVLDB 1(1)*, 2008 .

[30] A. Alexandrov, R. Bergmann, S. Ewen, J. C. Freytag, F. Hueske, A. Heise, O. Kao, M. Leich, U. Leser, V. Markl and e. al, "The Stratosphere platform for big data analytics," *VLDB J.* , vol. 23, no. 6, 2014.

[31] D. Battré, S. Ewen, F. Hueske, O. Kao, V. Markl and D. Warneke, "Nephele/PACTs: A programming model and execution framework for web-scale analytical processing," in *SoCC* , 2010.

[32] A. Alexandrov, A. Katsifodimos, G. Krastev and V. Markl, "Implicit Parallelism through Deep Language Embedding," in *SIGMOD Record*, 2016.

[33] A. Alexandrov, A. Kunft, A. Katsifodimos, F. Schüler, L. Thamsen, O. Kao, T. Herb and V. Markl, "Implicit Parallelism through Deep Language Embedding," in *SIGMOD*, 2015.

[34] Apache Flink, "Powered by Flink," [Online]. Available: https://cwiki.apache.org/confluence/display/FLINK/Powered+by+Flink. [Accessed 2017].

[35] I. Gog, M. Schwarzkopf and e. al, "Hand: Musketeer: all for one, one for all in data processing systems.," in *EuroSys*, 2015.

[36] S. Ewen, K. Tzoumas, M. Kaufmann and V. Markl, "Spinning Fast Iterative Data Flows.," *PVLDB,* vol. 5, no. 11, pp. 1268-1279, 2012.

[37] P. Carbone, A. Katsifodimos, A. Ewen, V. Markl and e. al:, "Apache Flink™: Stream and Batch Processing in a Single Engine.," *IEEE Data Eng. Bull.,* vol. 38, no. 4, pp. 28-38, 2015.

[38] S. Breß, H. Funke and J. Teubner, "Robust Query Processing in Co-Processor-accelerated Databases.," in *SIGMOD*, 2016.

[39] J. Soto and V. Markl, "A Historical Account of Apache Flink," [Online]. Available: http://www.dima.tu-berlin.de/fileadmin/fg131/ Informationsmaterial/Apache_Flink_Origins_for_Public_Release.pdf.. [Accessed 2017].

[40] S. Schelter, S. Ewen, K. Tzoumas and V. Markl, "All Roads lead to Rome: Optimistic Recovery for distributed iterative data processing.," in *CIKM*, 2013.

[41] A. Kunft, A. Alexandrov, A. Katsifodimos, . Markl: Bridging the gap: towards optimization across linear and relational algebra. BeyondMR@SIGMOD 2016: 1

[42] A. Kunft, A. Katsifodimos, S. Schelter, et al: BlockJoin: Efficient Matrix Partitioning Through Joins. PVLDB 10(13): 2061-2072 (2017)