Report from Dagstuhl Seminar 14511

Programming Languages for Big Data (PlanBig)

Edited by

James Cheney[1], Torsten Grust[2], and Dimitrios Vytiniotis[3]

1 **University of Edinburgh, GB,** jcheney@inf.ed.ac.uk
2 **Universität Tübingen, DE,** torsten.grust@uni-tuebingen.de
3 **Microsoft Research UK – Cambridge, GB,** dimitris@microsoft.com

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 14511 "Programming Languages for Big Data (PlanBig)". The seminar was motivated by recent developments in programming languages, databases, machine learning, and cloud computing, and particularly by the opportunities offered by research drawing on more than one of these areas. Participants included researchers working in each of these areas and several who have previously been involved in research in the intersection of databases and programming languages. The seminar included talks, demos and free time for discussion or collaboration. This report collects the abstracts of talks and other activities, a summary of the group discussions at the seminar, and a list of outcomes.

Seminar December 15–19, 2014 – http://www.dagstuhl.de/14511
1998 ACM Subject Classification D.3.2 [Programming Languages]: Language Classifications – Applicative (functional) languages, H.2.3 [Database Management]: Languages – Query Languages, H.2.4 Systems - Distributed Databases, Query Processing; H.2.8 Database Applications – Data mining, Scientific databases
Keywords and phrases Programming languages, databases, data-centric computation, machine learning, cloud computing
Digital Object Identifier 10.4230/DagRep.4.12.48
Edited in cooperation with Alexander Ulrich

1 Executive Summary

James Cheney
Torsten Grust
Dimitrios Vytiniotis

Large-scale data-intensive computing, commonly referred to as "Big Data", has been influenced by and can further benefit from programming languages ideas. The MapReduce programming model is an example of ideas from functional programming that has directly influenced the way distributed big data applications are written. As the volume of data has grown to require distributed processing potentially on heterogeneous hardware, there is need for effective programming models, compilation techniques or static analyses, and specialized language runtimes. The motivation for this seminar has been to bring together researchers working on foundational and applied research in programming languages but also data-intensive computing and databases, in order to identify research problems and opportunities for improving data-intensive computing.

To this extent, on the database side, the seminar included participants who work on databases, query languages and relational calculi, query compilation, execution engines, distributed processing systems and networks, and foundations of databases. On the programming languages side, the seminar included participants who work on language design, integrated query languages and meta-programming, compilation, as well as semantics. There was a mix of applied and foundational talks, and the participants included people from universities as well as industrial labs and incubation projects.

The work that has been presented can be grouped in the following broad categories:

- Programming models and domain-specific programming abstractions (Cheney, Alexandrov, Vitek, Ulrich). How can data processing and query languages be integrated in general purpose languages, in type-safe ways and in ways that enable traditional optimizations and compilation techniques from database research? How can functional programming ideas such as monads and comprehensions improve the programmability of big data systems? What are some language design issues for data-intensive computations for statistics?
- Incremental data-intensive computation (Acar, Koch, Green). Programming language support and query compilation techniques for efficient incremental computation for data set or query updates. Efficient view maintainance.
- Interactive and live programming (Green, Vaz Salles, Stevenson, Binnig, Suciu). What are some challenges and techniques for interactive applications. How to improve the live programming experience of data scientists? Ways to offer data management and analytics as cloud services.
- Query compilation (Neumann, Henglein, Rompf, Ulrich). Compilation of data processing languages to finite state automata and efficient execution. Programming languages techniques, such as staging, for enabling implementors to concisely write novel compilation schemes.
- Data programming languages and semantics (Wisnesky, Vansummeren). Functorial semantics for data programming languages, but also foundations for languages for information extraction.
- Foundations of (parallel) query processing (Suciu, Neven, Hidders). Communication complexity results, program equivalence problems in relational calculi.
- Big data in/for science (Teubner, Stoyanovich, Ré). Challenges that arise in particle physics due to the volume of generated data. Howe we can use data to speed up new material discovery and engineering? How to use big data systems for scientific extraction and integration from many different data sources?
- Other topics: architecture and runtimes (Ahmad), coordination (Foster), language runtimes (Vytiniotis), weak consistency (Gotsman).

The seminar schedule involved three days of scheduled talks, followed by two days of free-form discussions, demos, and working groups. This report collects the abstracts of talks and demos, summaries of the group discussion sessions, and a list of outcomes resulting from the seminar.

2 Table of Contents

3 Overview of Talks

3.1 Self-Adjusting Computation for Dynamic and Large Data Sets

Umut A. Acar (Carnegie Mellon University – Pittsburgh, US)

Developing efficient and reliable software is a difficult task. Rapidly growing and dynamically changing data sets further increase complexity by making it more challenging to achieve efficiency and performance. We present practical and powerful abstractions for taming software complexity in this domain. Together with the algorithmic models and programming languages that embody them, these abstractions enable designing and developing efficient and reliable software by using high-level reasoning principles and programming techniques. As evidence for their effectiveness, we consider a broad range of benchmarks including sophisticated algorithms in geometry, machine-learning, and large data sets. On the theoretical side, we show asymptotically significant improvements in efficiency and present solutions to several open problems using the proposed techniques. On the practical side, we present programming languages, compilers, and related software systems that deliver significant improvements in performance, usually with little effort from the programmer. This talk is based on research done jointly with collaborators including A. Ahmed, G. Blelloch, M. Blume, Y. Chen, J. Dunfield, M. Fluet, M. Hammer, R. Harper, B. Hudson, R. Ley-Wild, O. Sumer, K. Tangwongsan, D. Turkoglu.

3.2 Deconstructing Big Data Stacks

Yanif Ahmad (Johns Hopkins University, US)

Modern big data applications deployed in datacenter environments are complex layered software stacks that provide functionality ranging from the networking and storage hardware, to the high-level analytics logic required by the application. Today's data systems play a central role in facilitating distributed data movement, query scheduling and fault tolerance for large-scale data processing. In this talk, we survey and deconstruct the design decisions made in the modern data systems architectures commonly found in a Big Data stack. This includes the storage services provided for input data as well as large intermediate results, support for both mid-query and inter-query fault tolerance, and the architectural impact of providing low-latency results, ideally without a long tail. The systems considered include HDFS, Hadoop, Spark, Impala, Storm and briefly NoSQL and NewSQL DBMS.

3.3 Data Analytics with Flink

Alexander Alexandrov (TU Berlin, DE)

Joint work of Katsifodimos, Asterios; Alexandrov, Alexander
URL http://flink.apache.org/

In this demo session we give an overview of Apache Flink – an open-source system for scalable data analysis. We present Flink's functional programming model and discuss some unique system features: (1) the approach of managing a JVM-based heap through aggressive object serialization on byte buffers, (2) the cost-based dataflow optimizer, and (3) the support for native incremental iterations and their resemblance with semi-naive Datalog evaluation.

3.4 Interactive & Visual Data Exploration

Carsten Binnig (DHBW – Mannheim, DE)

Joint work of Sam Zhao; Binnig, Carsten; Tim Kraska; Ugur Cetintemel; Stan Zdonik

Data-centric applications in which data scientists of varying skill levels explore large data sets are becoming more and more relevant to make sense of the data, identify interesting patterns, and bring aspects of interest into focus for further analysis. Enabling these applications with ease of use and at "human speeds" is key to democratizing data science and maximizing human productivity. As a first step towards visual interactive data exploration, we implemented a visual index for computing histograms based on a B^+-tree. The major differences to the traditional B^+-tree are: (1) We annotate the index nodes with count values as discussed before. (2) We offer optimized index traversal strategies for all requested bins of a histogram. (3) We use typical bin definitions of a histogram as separators for the upper levels instead of using the normal balancing rules.

3.5 From LINQ to QDSLs

James Cheney (University of Edinburgh, GB)

Joint work of Cheney, James; Lindley, Sam; Wadler, Philip
Main reference J. Cheney, S. Lindley, P. Wadler, "A practical theory of language-integrated query," in Proc. of the 18th ACM SIGPLAN Int'l Conf. on Functional programming (ICFP'13), pp. 403–416, ACM, 2013.
URL http://dx.doi.org/10.1145/2544174.2500586

Language-integrated query techniques ease database programming by placing queries and ordinary program code on the same level, so that the language implementation can check and coordinate communication between the host language and database. Such techniques are based on foundations developed in the 90s including comprehension syntax, normalization results for nested relational calculus, and more recent work on generalizing normalization to a higher-order setting and embedding query languages in host languages using quotation (a technique we identify as Quotation-based Domain Specific Languages, or QDSLs). In this talk I give an overview of this prior work exemplifying interaction between database and programming language research, and illustrate its impact on LINQ for F#.

3.6 Demo: Normalization and Query Composition in LINQ

James Cheney (University of Edinburgh, GB)

Joint work of Cheney, James; Lindley, Sam; Wadler, Philip
Main reference J. Cheney, S. Lindley, P. Wadler, "A practical theory of language-integrated query," in Proc. of the 18th ACM SIGPLAN Int'l Conf. on Functional programming (ICFP'13), pp. 403–416, ACM, 2013.
URL http://dx.doi.org/10.1145/2500365.2500586

In this demo I explained the underlying ideas of LINQ in F#, and application of recent work with Lindley and Wadler on normalization of query expressions. LINQ already performs some transformations to query expressions at run time using quotation and reflection capabilities of F#, but it has some gaps in support for queries that involve higher-order functions. Our work overcomes this limitation by providing a guarantee that all query expressions of a certain class normalize to a form that can be turned into SQL – even if the query expression makes use of lambda-abstraction and application. This has subtle implications, and allows writing efficient queries using lambda-abstraction that are not executed efficiently by the built-in F# LINQ library, and constructing queries at run time by recursion over in-memory data (illustrated by showing how XPath queries and their mapping to SQL can be defined in F# LINQ)

3.7 The Homeostatis Protocol: Avoiding Transaction Coordination Through Program Analysis

Nate Foster (Cornell University – Ithaca, US)

Joint work of Roy, Sudip; Bender, Gabriel; Kot, Lucja; Ding, Bailu; Foster, Nate; Gehrke, Johannes; Koch, Christoph

Many datastores rely on distribution and replication to achieve improved performance and fault-tolerance. But correctness of many applications depends on strong consistency properties – something that can impose substantial overheads, since it requires coordinating the behavior of multiple nodes. This work developed a new approach to achieving strong consistency in distributed systems while minimizing communication between nodes. The key insight was to allow the state of the system to be inconsistent during execution, as long as this inconsistency is bounded and does not affect transaction correctness. In contrast to previous work, our approach used program analysis to extract semantic information about permissible levels of inconsistency and is fully automated. We also developed a novel "homeostasis protocol" to allow sites to operate independently, without communicating, as long as any inconsistency is governed by appropriate treaties between the nodes. We designed mechanisms for optimizing treaties based on workload characteristics to minimize communication, built a prototype implementation, and conducted experiments to demonstrate the benefits of our approach on transactional benchmarks.

To appear in SIGMOD 2015.

3.8 Weak Consistency in Cloud Storage

Alexey Gotsman (IMDEA Software Institute, ES)

Joint work of Bernardi, Giovanni; Cerone, Andrea; Burckhardt, Sebastian; Yang, Hongseok; Zawirski, Marek

Modern geo-replicated databases underlying large-scale Internet services guarantee immediate availability and tolerate network partitions at the expense of providing only weak forms of consistency, commonly dubbed *eventual consistency*. At the moment there is a lot of confusion about the semantics of eventual consistency, as different systems implement it with different sets of features and in subtly different forms, stated either informally or using disparate and low-level formalisms.

We address this problem by proposing a framework for formal and declarative specification of the semantics of eventually consistent systems using axioms. Our framework is fully customisable: by varying the set of axioms, we can rigorously define the semantics of systems that combine any subset of typical guarantees or features, including conflict resolution policies, session guarantees, causality guarantees, multiple consistency levels and transactions. We prove that our specifications are validated by an example abstract implementation, based on algorithms used in real-world systems. These results demonstrate that our framework provides system architects with a tool for exploring the design space, and lays the foundation for formal reasoning about eventually consistent systems.

3.9 Live Programming for Big Data

Todd J. Green (LogicBlox – Atlanta, US)

Joint work of Green, Todd J.; Olteanu, Dan; Washburn, Geoffrey

We observe that the emerging category of self-service enterprise applications motivates support for "live programming" in the database, where the user's iterative exploration of the input data triggers changes to installed application code and its output in real time. This talk discusses the technical challenges in supporting live programming in the database and presents the solution implemented in version 4.1 of the LogicBlox commercial database system. The workhorse architectural component is a novel "meta-engine" that incrementally maintains metadata representing application code, guides compilation of input application code into its internal representation in the database kernel, and orchestrates maintenance of materialized views based on those changes. Our approach mirrors LogicBlox's declarative programming model and describes the maintenance of application code using declarative meta-rules; the meta-engine is essentially a "bootstrap" version of the database engine proper. Beyond live programming, the meta-engine turns out effective for a range of static analysis and optimization tasks, which we discuss. Outside of the database systems context, we speculate that our design may even provide a novel means of building incremental compilers for general-purpose programming languages.

3.10 Towards Regular Expression Processing at 1 Gbps/core

Fritz Henglein (University of Copenhagen, DK)

Joint work of Bjørn Grathwohl; Henglein, Fritz; Ulrik Rasmussen
Main reference N. B. B. Gratewohl, F. Henglein, U. T. Rasmussen, "Optimally Streaming Greedy Regular Expression Parsing," in Proc. of the 11th Int'l Colloquium on Theoretical Aspects of Computing (ICTAC'14), LNCS, Vol. 8687, pp. 224–240, Springer, 2014.
URL http://dx.doi.org/10.1007/978-3-319-10882-7_14
URL http://www.diku.dk/kmc

We describe how type theory, prefix codes, nondeterministic automata, streaming and determinization to register automata yield a worst-case linear-time regular expression parser for fixed regular expressions. Early tests indicate that it operates at a sustained 100+ Mbps rate on complex regular expressions and large data sets; this seems to be significantly faster than existing tools, which operate at 2 to 20 Mbps (commodity PC). We sketch how we believe an expressive regular expression processor executing at 1 Gbps per 64-bit core can be designed and implemented, without employing machine-specific or hardware oriented tricks.

3.11 MapReduce Optimisation in the Nested Relational Calculus

Jan Hidders (TU Delft, NL)

Joint work of Grabowski, Marek; Hidders, Jan; Sroka, Jacek; Vansummeren, Stijn
Main reference M. Grabowski, J.n Hidders, J. Sroka ,"Representing mapreduce optimisations in the nested relational calculus," in Proc. of the 29th British Nat'l Conf. on Big Data (BNCOD'13), LNCS, Vol. 7968, pp. 175–188, Springer, 2013.
URL http://dx.doi.org/10.1007/978-3-642-39467-6_17

We introduced sNRC, a variant of the Nested Relational Calculus over bags which allows heterogeneous bags and has two special operations: basic value equality and a duplicate elimination operator that selects only basic values. In this language we can readily represent a MapReduce operator, and so reasoning about equivalence of expressions in the language becomes equivalent to reasoning over MapReduce workflows over nested data. It is discussed how it might be possible to axiomatise equivalence of expressions with relatively simple equations. We also show some conjectures about the decidability of this problem for the presented fragment, and how this relates to existing results and open problems.

3.12 Incremental Computation: The Database Approach

Christoph Koch (EPFL – Lausanne, CH)

Main reference C. Koch, Y. Ahmad, O. Kennedy, M. Nikolic, An. Nötzli, D. Lupei, A. Shaikhha, "DBToaster: higher-order delta processing for dynamic, frequently fresh views," The VLDB Journal, 23(2):253–278, 2014.
URL http://dx.doi.org/10.1007/s00778-013-0348-4

In this talk, I presented the database approach to incremental computation – incremental view maintenance by compile-time query transformation. I first presented the classical approach

to incremental view maintenance using delta queries and then presented the DBToaster approach – recursive or higher-order incremental view maintenance. I also gave a demo of the DBToaster system, available at www.dbtoaster.org. Finally, I presented our recent work on higher-order incremental view maintenance for nested relational queries and the simply-typed lambda calculus, available as a preprint as [1].

References

1 Daniel Lupei, Christoph Koch, and Val Tannen. Incremental View Maintenance for Nested Relational Algebra. http://arxiv.org/abs/1412.4320, 2014.

3.13 Compiling SQL Queries into Executable Code

Thomas Neumann (TU München, DE)

Joint work of Neumann, Thomas; Leis, Viktor
Main reference T. Neumann, V. Leis, "Compiling Database Queries into Machine Code," IEEE Data Engineering Bulletin, 37(1):3–11, 2014.
URL http://sites.computer.org/debull/A14mar/p3.pdf

On modern servers the working set of database management systems becomes more and more main memory resident. Slow disk accesses are largely avoided, and thus the in-memory processing speed of databases becomes an important factor. One very attractive approach for fast query processing is just-in-time compilation of incoming queries. By producing machine code at runtime we avoid the overhead of traditional interpretation systems, and by carefully organizing the code around register usage we minimize memory traffic and get excellent performance. In this talk we show how queries can be brought into a form suitable for efficient translation, and how the underlying code generation can be orchestrated. By carefully abstracting away the necessary plumbing infrastructure we can build a query compiler that is both maintainable and efficient. The effectiveness of the approach is demonstrated by the HyPer system that uses query compilation as its execution strategy and achieves excellent performance.

3.14 Parallel-Correctness and Transferability for Conjunctive Queries

Frank Neven (Hasselt University – Diepenbeek, BE)

Joint work of Ameloot, Tom; Geck, Gaetano; Ketsman, Bas; Neven, Frank; Schwentick, Thomas
Main reference T. J. Ameloot, G. Geck, B. Ketsman, F. Neven, T. Schwentick, "Parallel-Correctness and Transferability for Conjunctive Queries," arXiv:1412.4030v2 [cs.DB], 2015.
URL http://arxiv.org/abs/1412.4030v2

A dominant cost for query evaluation in modern massively distributed systems is the number of communication rounds. For this reason, there is a growing interest in single-round multiway join algorithms where data is first reshuffled over many servers and then evaluated in a parallel but communication-free way. The reshuffling itself is specified as a distribution policy. We introduce a correctness condition, called parallel-correctness, for the evaluation of queries with respect to a distribution policy. We study the complexity of parallel-correctness for conjunctive queries as well as transferability of parallel-correctness between queries. We also investigate the complexity of transferability for certain families of distribution policies, including, for instance, the Hypercube distribution.

3.15 DeepDive: A Data System for Macroscopic Science

Christopher Ré (Stanford University, US)

Main reference C.r Ré, A. Abbas Sadeghian, Z. Shan, J. Shin, F. Wang, S. Wu, C. Zhang, "Feature Engineering for Knowledge Base Construction," IEEE Data Engineering Bulletin, 37(3):26–40, 2014; pre-print available as arXiv:1407.6439v3 [cs.DB].
URL http://sites.computer.org/debull/A14sept/p26.pdf
URL http://arxiv.org/abs/1407.6439v3
URL http://DeepDive.stanford.edu

Many pressing questions in science are macroscopic in that these questions require that a scientist integrate information from many data sources. Often, these data sources are documents that contain natural language text, tables, and figures. Such documents contain valuable information, but they are difficult for machines to understand unambiguously. This talk describes *DeepDive*, a statistical extraction and integration system to extract information from such documents. For tasks in paleobiology, *DeepDive*-based systems are surpassing human volunteers in data quantity, recall, and precision. This talk describes recent applications of *DeepDive* and *DeepDive*'s technical core. One of those core technical issues is efficient statistical inference. In particular, we describe our recent *Hogwild!* and *DimmWitted* engines that explore a fundamental tension between statistical efficiency (steps until convergence) and hardware efficiency (efficiency of each of those steps). In addition, we offer thoughts about how domain specific languages can help.

DeepDive is open source and available from http://DeepDive.stanford.edu.

3.16 An Efficient SQL to C Compiler in 500 lines of Scala

Tiark Rompf (Purdue University, US)

For hard-core systems level programming, low-level C code is still the industry standard. The drawbacks are well known: buggy systems, security vulnerabilities, poor programmer productivity, etc. Generative programming is an alternative: writing expressive high-level programs that generate fast low-level code at runtime. While many languages come with basic code generation facilities, generative programming has remained somewhat of a black art. Recent developments, however, promise to make generative programming much more accessible. This talk will provide a step-by-step introduction to the open-source LMS (Lightweight Modular Staging) framework, which brings runtime code generation and compilation to Scala programs. We will build a SQL query engine that outperforms commercial and open source database systems and consists of just about 500 lines of high-level Scala code. Along the way, we will discuss concepts such as mixed-stage data structures that contain both static and dynamic parts (e.g. static schema and dynamic values for data records) and staged interpreters which can be mechanically turned into compilers (e.g. for SQL queries or regular expressions).

3.17 F#3.0 – Strongly-Typed Language Support for Internet-Scale Information Sources

Andrew Stevenson (Queen's University – Kingston, CA)

Joint work of Syme, Don; Battocchi, Keith; Takeda, Kenji; Malayeri, Donna; Fisher, Jomo; Hu, Jack; Liu, Tao; McNamara, Brian; Quirk, Daniel; Taveggia, Matteo; Chae, Wonseok; Matsveyeu, Uladzimir; Petricek, Tomas

Main reference D. Syme, K. Battocchi, K. Takeda, D. Malayeri, J. Fisher, J. Hu, T. Liu, B. McNamara, D. Quirk, M. Taveggia, W. Chae, U. Matsveyeu, T. Petricek, "F#3.0 – Strongly-Typed Language Support for Internet-Scale Information Sources," Technical Report, MSR-TR-2012-101, 2012.

URL http://research.microsoft.com/apps/pubs/?id=173076

A growing trend in both the theory and practice of programming is the interaction between programming and rich information spaces. From databases to web services to the semantic web to cloud-based data, the need to integrate programming with heterogeneous, connected, richly structured, streaming and evolving information sources is ever-increasing. Most modern applications incorporate one or more external information sources as integral components. Providing strongly typed access to these sources is a key consideration for strongly-typed programming languages, to insure low impedance mismatch in information access. At this scale, information integration strategies based on library design and code generation are manual, clumsy, and do not handle the internet-scale information sources now encountered in enterprise, web and cloud environments. In this report we describe the design and implementation of the type provider mechanism in F# 3.0 and its applications to typed programming with web ontologies, web-services, systems management information, database mappings, data markets, content management systems, economic data and hosted scripting. Type soundness becomes relative to the soundness of the type providers and the schema change in information sources, but the role of types in information-rich programming tasks is massively expanded, especially through tooling that benefits from rich types in explorative programming.

3.18 (Big) Data Challenges in Materials Science and Engineering

Julia Stoyanovich (Drexel University – Philadelphia, US)

Materials Science and Engineering (MSE) is focused on the process of engineering matter into new and useful forms. It is a vast field that seeks to understand the properties of materials, to create materials appropriate for particular tasks, and to predict material behavior. Like many other disciplines, MSE is looking for ways to leverage data-driven approaches to make the process of scientific discovery and engineering more efficient. In this talk I present two interesting MSE use cases, outline ongoing efforts towards making MSE a data-intensive domain, and discuss ingredients of an MSE cyberinfrastructure.

3.19 Big Data Management with the Myria Cloud Service

Dan Suciu (University of Washington – Seattle, US)

Joint work of Halperin, Daniel; de Almeida, Victor Teixeira; Choo, Lee Lee; Chu, Shumo; Koutris, Paraschos; Moritz, Dominik; Ortiz, Jennifer; Ruamviboonsuk, Vaspol; Wang, Jingjing; Whitaker, Andrew; Xu, Shengliang; Balazinska, Magdalena; Howe, Bill; Suciu, Dan
Main reference D. Halperin, V. Teixeira de Almeida, L. L. Choo, S. Chu, P. Koutris, D. Moritz, J. Ortiz, V. Ruamviboonsuk, J. Wang, A. Whitaker, S. Xu, M. Balazinska, B. Howe, D. Suciu, "Demonstration of the Myria big data management service," in Proc. of the 2014 ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD'14), pp. 881–884, ACM, 2014.
URL http://dx.doi.org/10.1145/2588555.2594530
URL http://myria.cs.washington.edu/

Myria is a novel cloud service for big data management and analytics designed to improve productivity. Myria's goal is for users to simply upload their data and for the system to help them be self-sufficient data science experts on their data – self-serve analytics. From a web browser, Myria users can upload data, author efficient queries to process and explore the data, and debug correctness and performance issues. Myria queries are executed on a scalable, parallel cluster that uses both state-of-the-art and novel methods for distributed query processing.

3.20 Communication Cost in Parallel Query Processing

Dan Suciu (University of Washington – Seattle, US)

Joint work of Beame, Paul; Koutris, Paraschos; Suciu, Dan
Main reference P. Beame, P. Koutris, D. Suciu, "Skew in parallel query processing," in Proc. of the 33rd ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems (PODS'14), pp. 212–223, ACM, 2014.
URL http://dx.doi.org/10.1145/2594538.2594558

We study the problem of computing a conjunctive query q in parallel using p servers on a large database. We consider algorithms with one round of communication, and study the complexity of the communication. We prove matching upper and lower bounds based on the fractional edge packing of the query.

3.21 Big Data Problems in Particle Physics

Jens Teubner (TU Dortmund, DE)

Joint work of Teubner, Jens; Spaan, Bernhard

The Large Hadron Collider at CERN is often cited as a source of extremely large data volumes, or "Big Data". The talk gives a brief intuition of the type of experiments that are being ran at CERN (specifically the LHCb sub-project) and I will show what types of data are being produced and how they are being accessed by physical analyses. I will sketch my vision on how database-oriented techniques could be used to allow for more efficient data analysis and – as a consequence – to improve the insights that can be gained from the experimental data.

3.22 Query Compilation Based on the Flattening Transformation

Alexander Ulrich (Universität Tübingen, DE)

Main reference A. Ulrich, T. Grust, "The Flatter, the Better – Query Compilation Based on the Flattening Transformation," to appear in Proc. of the 34th ACM SIGMOD Int'l Conf. on the Management of Data (SIGMOD'15).

We tackle the problem of supporting an expressive, fully compositional list-based query language that allows nested results efficiently on off-the-shelf relational query engines. Query formulation is centered around comprehensions and a rich set of order-aware combinators including grouping, aggregation and sorting. This query language provides a basis for the construction of language-integrated query systems that seamlessly embed querying capabilities into functional programming languages. In this talk, we sketch the internals of a query compiler centered around the *flattening transformation*, a program transformation originally conceived to support nested data parallelism on vector processors. Adapted to query compilation, the flattening-based approach shreds nested queries into a small, statically determined number of efficient relational queries. In contrast to previous work, flattening-based query compilation (a) consists of a composition of simple steps that build on previous work and are easy to reason about (b) supports ordered collections and operations like aggregation, grouping and sorting and (c) produces efficient code.

In addition, we demonstrate Database-Supported Haskell (DSH), an implementation of flattening-based query shredding. DSH is an embedded query DSL that allows to formulate complex queries in idiomatic Haskell style. DSH queries are constructed from (higher-order) combinators and comprehensions, support abstraction over sub-queries and are subject to the same static typing discipline as other parts of a Haskell program. DSH compiles such queries with nested results into a bundle of efficient flat queries for off-the-shelf relational query engines.

3.23 Spanners: A Formal Framework for Information Extraction

Stijn Vansummeren (University of Brussels, BE)

Joint work of Fagin, Ron; Kimelfeld, Benny; Reiss, Frederick; Vansummeren, Stijn

An intrinsic part of information extraction is the creation and manipulation of relations extracted from text. In this talk, we present a foundational framework where the central construct is what we call a spanner. A spanner maps an input string into relations over the spans (intervals specified by bounding indices) of the string. The focus of this presentation is on the representation of spanners. Conceptually, there are two kinds of such representations. Spanners defined in a primitive representation extract relations directly from the input string; those defined in an algebra apply algebraic operations to the primitively represented spanners. This framework is driven by SystemT, an IBM commercial product for text analysis, where the primitive representation is that of regular expressions with capture variables. We define additional types of primitive spanner representations by means of two kinds of automata that assign spans to variables. We prove that the first kind has the same expressive power as regular expressions with capture variables; the second kind expresses precisely the algebra of the regular spanners – the closure of the first kind under standard relational operators.

The core spanners extend the regular ones by string-equality selection (an extension used in SystemT). We give some fundamental results on the expressiveness of regular and core spanners.

3.24 Challenges in Interactive Applications

Marcos Vaz Salles (University of Copenhagen, DK)

Joint work of Vaz Salles, Marcos; Kefaloukos, Pimin Konstantin; Zachariasen, Martin
Main reference P. K. Kefaloukos, M. A. Vaz Salles, M. Zachariasen, "Declarative cartography: In-database map generalization of geospatial datasets," in Proc. of the 2014 IEEE 30th Int'l Conf. on Data Engineering (ICDE'14), pp. 1024–1035, IEEE, 2014.
URL http://dx.doi.org/10.1109/ICDE.2014.6816720

Interactive applications, such as data visualizations and maps, computer games and simulations, or in-memory transactional and analytics systems, are becoming ever more pervasive and important to our society. In this talk, we describe lessons learned and challenges emerging from our research with these applications. First, we explore the challenge of declarative precomputation of complex data transformations in these applications, discussing an example of selecting data for zoomable maps [1]. Second, we discuss the challenge of performance visibility in programming models for online computations, suggesting a way to revisit the transaction model for this goal [2].

References

1 Pimin Konstantin Kefaloukos, Marcos Vaz Salles, and Martin Zachariasen. *Declarative Cartography: In-Database Map Generalization of Geospatial Datasets*. Proc. ICDE 2014, Chicago, Illinois, USA, 2014.
2 Vivek Shah. *Transactional Partitioning: A New Abstraction for Main-Memory Databases*. VLDB PhD Workshop, Hangzhou, China, 2014. Best paper runner-up.

3.25 The R Project and Language

Jan Vitek (Northeastern University – Boston, US)

URL http://www.r-project.org

Jan introduced the seminar attendees to the R project for statistical computing and the associated R scripting language. Through a series of live examples, from simple and obvious to quirky and outright surprising, Jan demonstrated relevant bits of the R language semantics. The discussion with the audience had a particular focus on R's family of collection data types (vectors, matrices, arrays, lists, factors, and data frames). Issues of R's interpreted execution model and the possibility of compiling R code were brought up later in the seminar.

Jan maintains his collection *AllR* of R-related implementation projects on GitHub: https://github.com/allr/.

3.26 Broom: Sweeping Out Garbage Collection from Big Data systems

Dimitrios Vytiniotis (Microsoft Research UK – Cambridge, GB)

Many popular systems for processing "big data" are implemented in high-level programming languages with automatic memory management via garbage collection (GC). However, high object churn and large heap sizes put severe strain on the garbage collector. As a result, applications underperform significantly: GC increases the runtime of typical data processing tasks by up to 40%. We propose to use region-based memory management instead of GC in distributed data processing systems. In these systems, many objects have clearly defined lifetimes. It is natural to allocate these objects in fate-sharing regions, obviating the need to scan a large heap. Regions can be memory-safe and could be inferred automatically. Our initial results show that region-based memory management reduces emulated Naiad vertex runtime by 34% for typical data analytics jobs.

3.27 The Functorial Data Model

Ryan Wisnesky (MIT – Cambridge, US)

Joint work of Wisnesky, Ryan; Spivak, David
Main reference D. I. Spivak, R. Wisnesky, "Relational Foundations for Functorial Data Migration," arXiv:1212.5303v5 [cs.DB], 2014.
URL http://arxiv.org/abs/1212.5303v5

We study the data transformation capabilities associated with schemas that are presented by directed multi-graphs and path equations. Unlike most approaches which treat graph-based schemas as abbreviations for relational schemas, we treat graph-based schemas as categories. A schema S is a finitely-presented category, and the collection of all S-instances forms a category, S-inst. A functor F between schemas S and T, which can be generated from a visual mapping between graphs, induces three adjoint data migration functors, $\Sigma_F : S\text{-inst} \to T\text{-inst}$, $\Pi_F : S\text{-inst} \to T\text{-inst}$, and $\Delta_F : T\text{-inst} \to S\text{-inst}$. We present an algebraic query language FQL based on these functors, prove that FQL is closed under composition, prove that FQL can be implemented with the select-project-product-union relational algebra (SPCU) extended with a key-generation operation, and prove that SPCU can be implemented with FQL.

4 Working Groups

The participants expressed a clear preference to avoid splitting into smaller groups to have discussions; instead, on Thursday and Friday there were plenary discussions in the main seminar room.

"Standard" intermediate language for data-centric programming

There are now lots of "programming languages for big data", exhibiting signs of convergent evolution, with similar primitives (usually starting with some variation on map and reduce operations). Nevertheless, most such languages seem not to be well-informed by principles of programming language design, or at least, these appear to be afterthoughts. One discussion session considered the question whether these efforts are now stable enough that there is a case for a community-led "standard" – drawing inspiration from the lazy functional programming community, which consolidated its effort behind a single language design (Haskell) after a number of exploratory language designs gained momentum in the 1980s and early 1990s.

There was an extensive discussion of what this would mean, with different participants taking different views of what a "standard" would mean and what its benefits would be. One question raised was the community that such a standard would serve – would it serve PL researchers (as a "standard calculus" for language-based work on big data / data-centric computation)? Would it serve system developers (as an API) or users (as a standard surface language)? Another concern raised was that industry tends to view academic work as irrelevant due to limited scale – would this limit the value of a standard language model?

One participant mentioned recent experience with eventual consistency: after an initial burst of enthusiasm, industry seems to be reverting to stronger consistency models and tested higher-level abstractions such as transactions. Thus, it may be premature to try to consolidate effort on language designs/calculi for dealing with big data, as work in this area may still be at an experimental stage and may be at risk of abandonment if its value is not realized soon.

At a more concrete level, participants discussed what kind of standard would be of value for their research. The lambda-calculus was cited as a (widely successful) example of a "standard" formalism that programming languages researchers use as a starting point for understanding and formalizing language design ideas, abstracting away somewhat from the full complexity of implemented languages. By analogy, a calculus that plays a similar role for cloud computing, MapReduce systems, or multicore CPU or GPU code could be valuable (it should be noted that there are already some such proposals). It might be a good idea to take experience from the OpenFlow standard in software-defined networking into account; OpenFlow was established by an industry consortium but has enabled programming languages and systems researchers to work to a common interface. Likewise, formalisms such as the relational calculus/algebra (and formal standards such as SQL) have played a similar role in the database community for decades.

An interesting issue for a proposed "standard model" is that of cost modeling: a calculus or language that attempts to abstract away from the implementation details risks abstracting away the computational costs as well, so there is a tension between abstraction/portability and performance transparency/scalability. A standard model that is *operationally transparent* would be valuable for parallel or distributed computing (but there was no clear consensus on what this would mean). It would be desirable for such a model to give an explicit account of physical properties or distances between components in the system to avoid cost-opacity. Cellular automata models were mentioned as an example of how to do this but it was argued that they are too low-level. The Delite system was also mentioned as an example providing a set of high-level operators that can be mapped to different execution architectures; it is higher-level than real hardware or systems and needs to be mapped to abstract machines that model the underlying hardware well. A standard formalism might need to handle multiple layers of abstraction (by analogy with relational query optimization with its logical, physical and run-time layers). Something that is "good enough" for typical uses and portable might

be the best tradeoff (analogously to C which is not perfect but represents a workable tradeoff between abstraction and performance).

In addition, there was a short side-discussion about the desirability of benchmarking and diversity clusters for the evaluation of "big data" systems (and language techniques for them). This would aid performance tuning and portability. The Stabilizer system from the University of Massachusetts was mentioned as an example of this. The general topic of reproducibility for computer science/systems research was also mentioned (and it was pointed out that this is currently receiving attention from several quarters).

Community-building

Another topic that was discussed was the need for, and options for, building a community to improve communication among and interaction between communities relevant to the topics of the seminar. There seemed to be consensus that it would be beneficial to encourage community-building in this area. Some participants expressed concern that existing workshops seem to be diminishing in popularity and value, while it is at least possible (sometimes with greater effort) to publish work with (for example) a significant DB component in PL venues or vice-versa. Others expressed the opinion that workshops are no longer as worthwhile and a lighter-weight approach such as Dagstuhl-like events every 2 years or so is preferable. This approach, however, has the disadvantage that it limits participation to those whom the organizers can identify well in advance of the event, so may limit diversity and community growth.

One concrete option that was discussed was the possibility of organizing a new conference (rather than workshop) on "data-centric computing" to encourage work and cross-fertilization between PL and systems/databases/machine learning. The pros and cons of this strategy were discussed. On the one hand, it was recognized that this would require buy-in from "big names" / thought leaders (beyond the participants in the Dagstuhl seminar). Another potential challenge was the need to encourage significant industry participation, which could impose constraints on logistics or venues. On the other hand, participants cited recent experience with new workshops on hot topics such as USENIX HotCloud and HotSDN workshops, the ACM Symposium on Cloud Computing, which has grown rapidly to an independent event since its inception in 2010.

Overall, it was recognized that a new venue might be feasible but a strong scientific case (going beyond identifying the shortcomings of existing venues) needs to be made, in terms of increased benefit to participants and better science. One participant (Umut Acar) volunteered to coordinate subsequent discussion of the idea of a new "data-centric computation" conference. Establishing such a new conference may be difficult and so experience with DBPL 2015 may help build the case for this.

DBPL

The final morning of the seminar saw a discussion of the future of DBPL, the International Symposium on Database Programming Languages, which has been running biennially since 1987. Recent occurrences of DBPL in 2011 and 2013 had seen a decrease in submissions and participation compared to previous years. Members of both events PC chair teams were present and as of the week of the seminar its status in 2015 was unclear. There was some

feeling that DBPL may have run its course, but also that it would be a shame for the series to end when events such as this Dagstuhl seminar showcase such a range of relevant activity. It was felt that this question was largely orthogonal to the question of developing a new conference venue (though a strong showing for DBPL in 2015 might contribute to a case for the "data-centric computation" conference idea).

DBPL had been co-located with VLDB (a major database conference, which seminar participants from the DB community would typically attend) until 2013, and since 2009 took place as a one-day workshop. In 2015, VLDB takes place the same week as ICFP, a major PL conference (and one which a number of seminar participants would normally attend). This clash highlighted a problem with DBPL's recent role as a "VLDB workshop": even in years when there is no clash with other events, participants from outside the DB community may find it difficult to justify the time/expense of attending another conference (or of just attending one day of an event they would otherwise not attend).

A number of alternatives were discussed, including the possibility of co-locating DBPL with ICFP in 2015, holding it as a stand-alone event (close in time/space to VLDB or ICFP but not formally affiliated with either), or seeking another co-location option . The possibility of co-locating with SPLASH 2015 (an umbrella PL conference including OOPSLA and several other events) was also raised, but did not seem to generate much enthusiasm at the seminar. An alternative proposal was considered, which attracted considerable support: to try to hold DBPL at *both* venues, with a video link connecting speakers and audience members at VLDB (in Hawaii) and ICFP (in Vancouver). Although this arrangement was recognized to have disadvantages (e.g. the inability to talk to speakers or other participants informally outside the conference room), participants felt that it offered the most promising route if it could be done. Of approximately 20 participants present in the discussion, a clear majority indicated willingness to either help organize or participate in/submit to DBPL if it were held in 2015.

5 Outcomes

- Umut Acar agreed to coordinate a discussion of the possibility of starting a "data-centric computation" conference.
- James Cheney started a "data-centric programming languages" mailing list, invited Dagstuhl participants to join and subsequently advertised it on relevant mailing lists such as TYPES and DBworld. The list currently has over 120 members.
- Fritz Henglein and Torsten Grust agreed to investigate the possibility of DBPL taking place "virtually" at two locations, with VLDB in Hawaii and ICFP in Vancouver connected by a video link. This turned out to be infeasible due to the high up-front cost of the link.
- Based on a straw poll conducted with Dagstuhl participants it was decided to approach the SPLASH 2015 organizers to see if DBPL could be co-located there. The SPLASH organizers were willing to approve this without going through the formal workshop application process. The two co-chairs are James Cheney and Thomas Neumann and 6 of the 10 PC members were participants in the Dagstuhl seminar.

Participants

- Umut A. Acar
Carnegie Mellon University – Pittsburgh, US
- Yanif Ahmad
Johns Hopkins University – Baltimore, US
- Alexander Alexandrov
TU Berlin, DE
- Carsten Binnig
DHBW – Mannheim, DE
- Giuseppe Castagna
University Paris-Diderot, FR
- James Cheney
University of Edinburgh, GB
- Laurent Daynès
Oracle Corporation, FR
- Nate Foster
Cornell University – Ithaca, US
- Pierre Geneves
INRIA – Grenoble, FR
- Alexey Gotsman
IMDEA Software – Madrid, ES
- Todd J. Green
LogicBlox – Atlanta, US
- Torsten Grust
Universität Tübingen, DE
- Fritz Henglein
University of Copenhagen, DK
- Jan Hidders
TU Delft, NL
- Christoph Koch
EPFL – Lausanne, CH
- Tim Kraska
Brown University, US
- Sam Lindley
University of Edinburgh, GB
- Todd Mytkowicz
Microsoft Corp. – Redmond, US
- Thomas Neumann
TU München, DE
- Frank Neven
Hasselt Univ. – Diepenbeek, BE
- Ryan R. Newton
Indiana University – Bloomington, US
- Kim Nguyen
University Paris-Sud – Gif sur Yvette, FR
- Klaus Ostermann
Universität Tübingen, DE
- Christopher Ré
Stanford University, US
- Tiark Rompf
Purdue University, US
- Andrew Stevenson
Queen's Univ. – Kingston, CA
- Julia Stoyanovich
Drexel Univ. – Philadelphia, US
- Dan Suciu
University of Washington – Seattle, US
- Jens Teubner
TU Dortmund, DE
- Alexander Ulrich
Universität Tübingen, DE
- Jan Van den Bussche
Hasselt Univ. – Diepenbeek, BE
- Stijn Vansummeren
Université Libre de Bruxelles, BE
- Marcos Vaz Salles
University of Copenhagen, DK
- Jan Vitek
Northeastern University – Boston, US
- Dimitrios Vytiniotis
Microsoft Research UK – Cambridge, GB
- Ryan Wisnesky
MIT – Cambridge, US