SIGMOD Officers, Committees, and Awardees

Chair	Vice-Chair	Secretary/Treasurer
Donald Kossmann Systems Group ETH Zürich Cab F 73 8092 Zuerich SWITZERLAND +41 44 632 29 40 <donaldk AT inf.ethz.ch>	Anastasia Ailamaki School of Computer and Communication Sciences, EPFL EPFL/IC/IIF/DIAS Station 14, CH-1015 Lausanne SWITZERLAND +41 21 693 75 64 <natassa AT epfl.ch>	Magdalena Balazinska Computer Science & Engineering University of Washington Box 352350 Seattle, WA USA +1 206-616-1069 <magda AT cs.washington.edu>

SIGMOD Executive Committee:
Donald Kossmann (Chair), Anastasia Ailamaki (Vice-Chair), Magdalena Balazinska, K. Selçuk Candan, Yanlei Diao, Curtis Dyreson, Yannis Ioannidis, Christian Jensen, and Tova Milo.

Advisory Board:
Yannis Ioannidis (Chair), Rakesh Agrawal, Phil Bernstein, Stefano Ceri, Surajit Chaudhuri, AnHai Doan, Joe Hellerstein, Michael Franklin, Laura Haas, Stratos Idreos, Tim Kraska, Renee Miller, Chris Olsten, Beng-Chin Ooi, Tamer Özsu, Sunita Sarawagi, Timos Sellis, Gerhard Weikum, John Wilkes

SIGMOD Information Director:
Curtis Dyreson, Utah State University <curtis.dyreson AT usu.edu>

Associate Information Directors:
Manfred Jeusfeld, Georgia Koutrika, Wim Martens, Mirella Moro

SIGMOD Record Editor-in-Chief:
Yanlei Diao, University of Massachusetts Amherst <yanlei AT cs.umass.edu>

SIGMOD Record Associate Editors:
Pablo Barceló, Vanessa Braganholo, Marco Brambilla, Chee Yong Chan, Rada Chirkova, Anastasios Kementsietsidis, Olga Papaemmanouil, Aditya Parameswaran, Anish Das Sarma, Alkis Simitsis, Nesime Tatbul, Marianne Winslett, and Jun Yang.

SIGMOD Conference Coordinator:
K. Selçuk Candan, Arizona State University

PODS Executive Committee:
Tova Milo (Chair), Diego Calvanse, Wenfei Fan, Martin Grohe, Rick Hull, Maurizio Lenzerini

Sister Society Liaisons:
Raghu Ramakhrishnan (SIGKDD), Yannis Ioannidis (EDBT Endowment), Christian Jensen (IEEE TKDE).

Awards Committee:
Elisa Bertino (Chair), Surajit Chaudhuri, Maurizio Lenzerini, Kartin Kersten, Umesh Dayal

Jim Gray Doctoral Dissertation Award Committee:
Tova Milo (Co-Chair), Juliana Freire (Co-Chair), Ashraf Aboulnaga, Minos Garofalakis, Chris Jermaine, Renee Miller, Aditya Parameswaran, Andy Pavlo, Kian-Lee Tan.

[Last updated : June 30, 2015]

SIGMOD Officers, Committees, and Awardees (continued)

SIGMOD Edgar F. Codd Innovations Award

For innovative and highly significant contributions of enduring value to the development, understanding, or use of database systems and databases. Formerly known as the "SIGMOD Innovations Award", it now honors Dr. E. F. (Ted) Codd (1923 - 2003) who invented the relational data model and was responsible for the significant development of the database field as a scientific discipline. Recipients of the award are the following:

Michael Stonebraker (1992)	Jim Gray (1993)	Philip Bernstein (1994)
David DeWitt (1995)	C. Mohan (1996)	David Maier (1997)
Serge Abiteboul (1998)	Hector Garcia-Molina (1999)	Rakesh Agrawal (2000)
Rudolf Bayer (2001)	Patricia Selinger (2002)	Don Chamberlin (2003)
Ronald Fagin (2004)	Michael Carey (2005)	Jeffrey D. Ullman (2006)
Jennifer Widom (2007)	Moshe Y. Vardi (2008)	Masaru Kitsuregawa (2009)
Umeshwar Dayal (2010)	Surajit Chaudhuri (2011)	Bruce Lindsay (2012)
Stefano Ceri (2013)	Martin Kersten (2014)	Laura Haas (2015)

SIGMOD Contributions Award

For significant contributions to the field of database systems through research funding, education, and professional services. Recipients of the award are the following:

Maria Zemankova (1992)	Gio Wiederhold (1995)	Yahiko Kambayashi (1995)
Jeffrey Ullman (1996)	Avi Silberschatz (1997)	Won Kim (1998)
Raghu Ramakrishnan (1999)	Michael Carey (2000)	Laura Haas (2000)
Daniel Rosenkrantz (2001)	Richard Snodgrass (2002)	Michael Ley (2003)
Surajit Chaudhuri (2004)	Hongjun Lu (2005)	Tamer Özsu (2006)
Hans-Jörg Schek (2007)	Klaus R. Dittrich (2008)	Beng Chin Ooi (2009)
David Lomet (2010)	Gerhard Weikum (2011)	Marianne Winslett (2012)
H.V. Jagadish (2013)	Kyu-Young Whang (2014)	Curtis Dyreson (2015)

SIGMOD Jim Gray Doctoral Dissertation Award

SIGMOD has established the annual SIGMOD Jim Gray Doctoral Dissertation Award to *recognize excellent research by doctoral candidates in the database field.* Recipients of the award are the following:

- **2006** *Winner*: Gerome Miklau, University of Washington. *Honorable Mentions*: Marcelo Arenas and Yanlei Diao.
- **2007** *Winner*: Boon Thau Loo, University of California at Berkeley. *Honorable Mentions*: Xifeng Yan and Martin Theobald.
- **2008** *Winner*: Ariel Fuxman, University of Toronto. *Honorable Mentions*: Cong Yu and Nilesh Dalvi.
- **2009** *Winner*: Daniel Abadi, MIT. *Honorable Mentions*: Bee-Chung Chen and Ashwin Machanavajjhala.
- **2010** *Winner:* Christopher Ré, University of Washington. *Honorable Mentions*: Soumyadeb Mitra and Fabian Suchanek.
- **2011** *Winner*: Stratos Idreos, Centrum Wiskunde & Informatica. *Honorable Mentions*: Todd Green and Karl Schnaitterz.
- **2012** *Winner*: Ryan Johnson, Carnegie Mellon University. *Honorable Mention*: Bogdan Alexe.
- **2013** *Winner*: Sudipto Das, University of California, Santa Barbara. *Honorable Mention*: Herodotos Herodotou and Wenchao Zhou.
- **2014** *Winners*: Aditya Parameswaran, Stanford University, and Andy Pavlo, Brown University.
- **2015** *Winners*: Alexander Thomson, Yale University. *Honorable Mentions*: Marina Drosou, University of Ioannina and Karthik Ramachandra, IIT Bombay

A complete listing of all SIGMOD Awards is available at: **http://www.sigmod.org/awards/**

[Last updated : June 30, 2015]

Guest Editor's Notes

Welcome to this Special Issue of the ACM SIGMOD Record on Visionary Ideas in Data Management!

This special issue features a collection of articles describing visions of future systems, frameworks, algorithms, applications, and technology related to the management and use of data. With this special issue, we hope to promote the discussion and sharing of challenges and ideas that are not necessarily well-explored at the time of writing, but have potential for significantly expanding the possibilities and horizons of the field of databases and data management.

We received 33 submissions from around the world: Australia, Austria, Brazil, Canada, China, France, Germany, Greece, India, Ireland, Italy, Singapore, Switzerland, UK, and USA. The submissions were evaluated on their originality, significance, potential impact, and interest to the community, with less emphasis on the current level of maturity, technical depth, and evaluation. Of the 33 submissions, 9 were selected to appear in this special issue; 4 are currently under revision for possible inclusion in the Vision Track of future issues of the SIGMOD Record.

The selected papers cover a truly diverse landscape. Some of them respond to recent hardware and technology trends, by exploring new data management techniques and architectures for field-programmable gate arrays, large main memory, modern data centers, and assemblies of autonomous, heterogeneous components. Other papers tackle new types of users or novel applications, by envisioning a new architecture to unify querying over multiple data models, use of on-the-fly synopsis in the simulation sciences, a system for managing and mining sequence data, and support for exploratory computing by data enthusiasts. Last but not least, inspired by how the human brain works, a paper proposes "managed forgetting" of information to ensure that important content is kept safe and useful over time. We hope this collection of papers will give you a glimpse of what new and exciting directions that our community is pursuing.

I would like to thank a number of important people who helped make this special issue possible. My first thanks go to Yanlei Diao, the Editor-in-Chief of the SIGMOD Record, for first approaching me in 2014 with her vision of a permanent home for vision articles in our field, and for her generous help in putting together this issue. I would also like to thank the SIGMOD Execute Committee for supporting the idea of using the special issue to kick-start the effort. My heartfelt thanks go to my fellow Associate Editors—Marco Brambilla, Chee-Yong Chan, Tasos (Anastasios) Kementsietsidis, Olga Papaemmanouil, and Aditya Parameswaran—who graciously helped me with the editing responsibilities. Last but not least, I am deeply grateful to the authors and reviewers who contributed to this special issue—it would not have been possible without your enthusiasm and hard (and impressive) work under such a tight schedule.

On behalf of the SIGMOD Record Editorial board, I hope that you all enjoy reading this special issue of the SIGMOD Record!

Your submissions to the Record are welcome via the submission site:
http://sigmod.hosting.acm.org/record

Prior to submitting, please read the Editorial Policy on the SIGMOD Record's Web site:
http://www.sigmod.org/publications/sigmod-record/sigmod-record-editorial-policy

Jun Yang

June 30, 2015

Past SIGMOD Record Editors:

Ioana Manolescu (2009-2013)
Ling Liu (2000 – 2004)
Arie Segev (1989 – 1995)
Thomas J. Cook (1981 – 1983)
Daniel O'Connell (1971 – 1973)
Alexandros Labrinidis (2007 – 2009)
Michael Franklin (1996 – 2000)
Margaret H. Dunham (1986 – 1988)
Douglas S. Kerr (1976-1978)
Harrison R. Morse (1969)
Mario Nascimento (2005 – 2007)
Jennifer Widom (1995 – 1996)
Jon D. Clark (1984 – 1985)
Randall Rustin (1974-1975)

The FQP Vision: Flexible Query Processing on a Reconfigurable Computing Fabric

Mohammadreza Najafi[1], Mohammad Sadoghi[2], Hans-Arno Jacobsen[1]
[1]Technical University Munich
[2]IBM T.J. Watson Research Center

ABSTRACT

The Flexible Query Processor (FQP) constitutes a family of hardware-based data stream processors that support dynamic changes to queries and streams, as well as static changes to the processor-internal fabric in order to maximize performance for given workloads. FQP is prototyped on field-programmable gate arrays (FPGAs). To this end, FQP supports select, project and window-join queries over data streams. While processing incoming tuples, FQP can accept new queries, a key characteristic distinguishing FQP from related approaches employing FPGAs for stream processing. In this paper, we present our vision of FQP, focusing on few internal details to support the flexibility dimension, in particular, the segment-at-a-time mechanism to realize processing of tuples of variable sizes. While many of these features are readily available in software, their hardware-based realizations have been one of the main shortcomings of existing research efforts.

1. INTRODUCTION

There is rising interest in accelerating stream processing through FPGAs (*e.g.*, [3, 4, 7, 8].) Many of these approaches are based on "compiling" static queries and fixed stream schemas into hardware designs that are synthesized to configure FPGAs. It is not uncommon for this synthesis step to take on the order of minutes to hours (depending on the complexity of the design), which is the norm for FPGAs, but is too inflexible for modern-day stream processing needs, which require the application to be able to change the query and the schema on the fly, without having to wait an extended period of time for the synthesis computation to be completed.

Furthermore, existing approaches to accelerating stream processing through FPGAs [4, 7, 8] assume that for processing query and stream modifications the arriving stream is halted, the hardware design updates are synthesized into configuration information, and the new information is uploaded onto the FPGA before processing of the event stream can resume. While synthesis and stream processing may overlap, a significant amount of time and efforts are still required to halt, re-configure, and resume the operation, which may take up to several minutes in and onto itself. More importantly, this modus operandi requires logic for buffering, handling of dropped tuples, requests for re-transmissions, and additional data flow controlling tasks, which renders this style of processing difficult in practice. These concerns are often ignored in the approaches listed above, which assumed that processing stops entirely before a new query-stream processing cycle starts.

In this paper, we aim to fill the gap between software solutions which provide the greatest degree of flexibility in query modification needs and hardware solutions which offer massive performance gains by designing an FQP that accepts new queries in an online fashion without disrupting the processing of incoming event streams. While supporting query modifications at runtime is almost trivial for software-based techniques, they are highly uncommon for custom hardware-based approaches, such as FPGAs, and have so far not received much attention in the growing body of work on accelerating data processing with FPGAs.

FQP is comprised of a parameterizable number of "online programmable blocks" (referred to as OPBs) that are inter-connected into a customizable topology. Together with a number of auxiliary components for query and tuple buffering, routing, and dispatching, the OPBs form an instance of the FQP that operates entirely on the FPGA. The inter-connection topology for the OPBs can be chosen in the manner most advantageous for the queries to be processed. For example, if the query workload lends itself for parallelism, a parallel topology can be chosen, whereas for workloads with more data dependency, a pipelined topology can be chosen. The choice of topology is performed statically and an instance of FQP is synthesized that realizes this topology. The OPB is the processing core that implements the actual query operators. It enables online changes to queries based on a number of parameters, including variable tuple size, projection attributes, selection conditions, join conditions, and join-window size.

In the design of FQP, we dealt with a number of challenges: First, a static FPGA-based query processor must over-provision resources to handle the largest expected (intermediate) tuple size, which under-utilizes system

resources. Second, the change in tuple size between the join operation's inputs and output adds new challenges, especially when there is the need to use the join result as input for other operations. Third, determining a minimal processing core that can efficiently handle a variety of query operators, some of which are stateless, while others are stateful.

The contributions of this work are manifold: (1) We outline our vision for stream processing on hardware in the context of our proposed FQP architecture. (2) We develop FQP, that unlike the state-of-the-art, enables online changes of queries and stream schema without interrupting query processing over incoming streams and without the need to re-synthesize the design. (3) We unify and share the underlying storage buffer for both data and operator parameters of a query. (4) We support variable tuple sizes by proposing the segment-at-a-time processing model, namely, an abstraction that divides a tuple into smaller chunks that are streamed and processed as a consecutive set of segments. This strategy avoids the need for over-provisioning of hardware resources. (5) We design FQP as a family for instantiating stream processors that are based on the different inter-connection topologies most suitable for the expected workload.

2. RELATED WORK

Over the past few years several projects on accelerating stream processing with FPGAs have been undertaken, many among researchers in the broader data management community. This work effectively demonstrated that FPGAs are a viable option for accelerating certain data management tasks in general, and stream processing in particular (*e.g.*, [3, 4, 7, 8, 5].)

Lockwood *et al.* [3] present an FPGA library to accelerate the development of streaming applications in hardware. Similarly, Meuller *et al.* [4] present Glacier, a component library and compiler, that compiles continuous queries into logic circuits on an operator-level basis. These approaches (including [7, 8]) are characterized by the goal of representing queries in logic circuits to achieve the best possible performance. While performance is a major design goal for us as well, we additionally aim to offer the application flexibility of updating queries at runtime.

The prototype of a hardware stream processor were recently presented [5, 6]. Najafi *et al.* [5, 6] showed the viability of building a stand-alone stream processor with FPGAs. The work presented in this paper builds upon this prototype, elaborating on the full-blown *Flexible Query Processor* systems project. For example, here, details are provided about the *segment-at-a-time processing* mechanism to support the processing of stream tuples with variable tuple sizes. But, more importantly, in the current work, we offer our broader and long-term vision of the FQP project.

3. MOTIVATING FQP VISION

To present our vision of stream processing acceleration, we first need to better understand the design space that must be navigated by resorting to FPGAs in order to complement or to replace general purpose processors. We begin by describing the challenges faced by today's general purpose processors.

Large & complex control units — The design of general purpose processors is based on the execution of consecutive operations on data residing in the system's main memory. The design must guarantee a correct sequential order execution. In presence of such strict execution order, the processor includes complex logic to increase performance, *e.g.*, super pipelining; out-of-order execution; single instruction, multiple data; and hyper-threading. As a result, the performance gain comes at the cost of having to devote resources (*i.e.*, transistors) to large and complex control units, which could occupy up to 95% of chip area [1].

Memory wall & von Neumann bottleneck — The current computer architecture suffers from the limited bandwidth between CPU and memory. This bandwidth is small compared to the rate at which the CPU itself can process. This issue is often referred to as the memory wall, and is becoming a major scalability limitation as the gap between CPU and memory speed increases [2]. To mitigate this issue, processors have been equipped with large cache units. However, the effectiveness of these units depends on the memory access patterns of executing programs. Additionally, the von Neumann bottleneck also contributes to the memory wall by sharing the limited memory bandwidth between instructions and data.

Redundant memory accesses — The current computer architecture enforces that data arriving from an I/O device is first read/written to main memory before it is processed by the CPU, which is a cause for a great deal of memory bandwidth loss. Consider a simple data stream filtering operation that would not require the incoming data stream to be first written to main memory. If the data arrives from the network interface, then in theory the data could stream directly through the processor. However, today's computer architecture prevents this modus operandi. Essentially, any I/O to and from the computer system is channeled through memory, which is a potential bottleneck even for high-speed inter-connects such as infiniBand that introduces latency on the order of microseconds.[1]

These performance limiting factors in today's computer architecture have resulted in a growing interest in accelerating data management and data stream process-

[1]Emerging InfiniBand adaptors, such as Mellanox ConnectX-3 Pro (`http://www.mellanox.com/page/infiniband_cards_overview`), have the potential to enable FPGAs to process data in-line and avoid the data movement overhead between I/O and memory.

ing on FPGAs. By designing custom hardware accelerators tailored for streaming processing, we can afford to get by with simpler control logic and better usage of chip area, *i.e.*, we can achieve higher performance per transistor ratio. Furthermore, we can mitigate memory wall issue, by coupling processor and local memory and instantiating many of these coupled processors and memories as needed. The redundant memory access could be reduced by avoiding copying and reading memory whenever possible.

Additionally, the stream processing has a set of unique characteristics that enables us to further exploit underlying hardware. For instance, by its nature, there is a higher chance of repeated execution of the same set of queries over a given potentially unbounded stream with a known stream data schema. Such repetition creates an opportunity to customize data path of our processors for optimal computation (*e.g.*, avoid deep instruction execution pipeline). Essentially, we execute the "data over the instructions" not the other way around, namely, instructions are implemented in hardware as a custom logic. Another distinct feature of streaming application is the I/O nature, where tuples go from input to processing to output without the need for storage in off-chip memory for an extended period of time, *e.g.*, state-less operations such as selections and projections do not require any write to (external) memory; they can simply be processed online in a pure streaming fashion. Also given the custom hardware implementation of queries, there is a greater degree of predictability, essentially eliminating costly branch mispredictions, page misses, and fewer instructions fetching. All of these properties have motivated us to rethink the development of a revolutionary different architecture for a stream processor that avoids today's computer architecture challenges.

4. THE FLEXIBLE QUERY PROCESSOR

The Flexible Query Processor (FQP) is a customized hardware solution we designed for building stream processors that can be specifically tailored to a given set of queries executed over a data stream. Furthermore, new queries can be inserted at run-time without requiring expensive re-synthesis, as is commonplace today in related FPGA-based processing approaches.[2] Essentially, FQP represents a set of components that can be assembled in various ways to give rise to a whole family of stream processors. The basis of FQP are *Online Programmable-Blocks* (OPBs) (*i.e.*, the processing cores.) The OPB itself is a simple stream processing element that supports a number of basic query operators over stream tuples. An OPB can be dynamically programmed by inter-sparsing the input data stream with new or updated queries, represented by a simple instruction set. The structure of input data stream distribution, OPB arrangement, and query result collection components define the *connection topology* of FQP. As a result, aside from the flexibility of FQP to be reprogrammed with new queries, the topology can be tailored to a specific query set (application), to maximize processing performance. OPBs are designed such that they can be connected to each other serially (*i.e.*, a *pipeline* arrangement), in parallel (*i.e.*, a *parallel* arrangement), and in a mixed manner (*i.e.*, *hybrid* arrangement).

[2]This is a unique limitation of FPGA-based approaches not found in software-based approaches, in which dynamic query insertion and update is hardly an issue worth underlining.

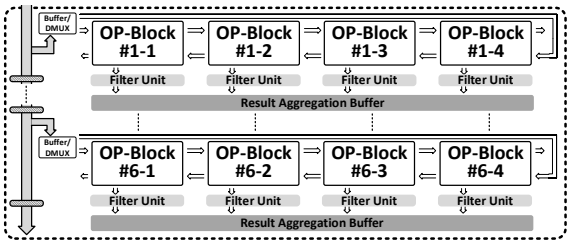


Figure 1: Partially parallel FQP topology.

4.1 FQP Overview

The processing performance of an instance of FQP is determined by its internal connection topology, the performance of each individual OPB, and the assignment of queries to OPBs. Figure 1 shows a small-scale partially parallel FQP topology comprised of 24 OPBs. Each one of the four consecutive OPBs per row are arranged in a pipelined fashion. All rows are arranged in parallel. Other topologies are possible, as determined appropriate by a pre-synthesis-time software-based configuration component that assembles an instance of an FQP, specifically tailored for the given or expected query workload. After synthesis, during query assignment, a query compiler determines a mapping of the input queries onto the given FQP instance. Queries can be inserted dynamically without requiring a resynthesis of the FQP instance, a major differentiation of our work from related approaches.

4.2 FQP Internal Architecture

Data stream distribution circuitry — In FQP instance shown above, we opted for a pipelined data stream distribution architecture, where each incoming tuple is inserted from the top and passes to the next pipeline stage (*cf.*, hashed blocks in the figure) in each clock cycle until it reaches the end of the path.

In each stage, the tuple is fed to the corresponding chain of OPBs. Depending on the configuration, more OPB-chains could be connected to a single stage. However, the number of attached chains is limited by a maximum fan-out. Attaching more chains to a single stage can result in a decrease of the FQP's clock frequency, leading to a performance degradation of the processor. The maximal fan-out is device (*e.g.*, FPGAs) dependent and has to be determined experimentally for a given FPGA. While feeding tuples through the pipeline of chains, some chains could be busy processing previous tuples. This imposes unwanted stalls in the distribution circuitry, leaving further chains idle. To address this is-

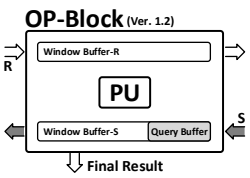


Figure 2: Stream processing element.

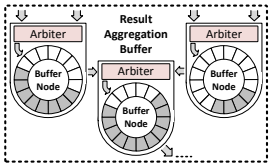


Figure 3: Result aggregation buffer (RAB).

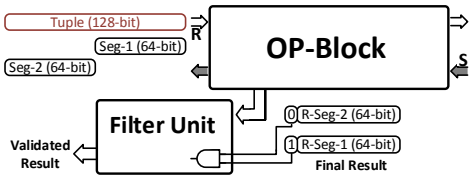


Figure 4: Segment-at-a-time at entry to OPB.

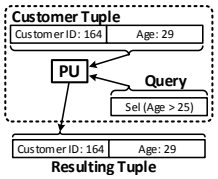


Figure 5: Customer segregation query.

sue, we use a *Buffer/DMUX* component which stores incoming tuples in its internal buffer and feeds them to the connected chain. This component also contains a demultiplexer which splits streams so they pass though the shared data stream distribution circuitry.

Online-programmable block (OPB) — FQP is comprised of a number of OPBs as basic stream processing elements that realize various query operators (*e.g.*, select, project, and join). Figure 2 shows a high-level diagram illustrating the ports of an OPB.

OPB itself is comprised of several components which work in parallel to maximize throughput. Here, we opt to briefly present the *Processing Unit* (PU) and the two window buffers as two important components of our design.

Designed to execute complex operators such as a window-join, the OPB includes two window buffers, with the maximum window sizes as pre-synthesis-time configurable parameters. *Window Buffer-R* is dedicated to input *Stream-R* (R) while *Window Buffer-L* is dedicated to input *Stream-S* (S) and to the reception of dynamically inserted queries, also referred to as *Query Buffer*.

The PU is the actual execution unit of OPB. Upon insertion of a new tuple, the PU fetches instructions from the *Query Buffer* and executes them against the tuples from one or both window buffers (depending on the query semantic). At the end of execution, the resulting tuples are emitted via the *Final Result* port or via the *Stream-R* output port for further processing by neighbouring OPBs (*i.e.*, for larger queries.)

Results at the *Final Result* port are gathered by the *Filter Unit* and after validation are fed to the *Result Aggregation Buffer* (RAB) for transmission to the output port of FQP. Validation includes tasks such as computing validity of an entire result tuple from its constituent parts, produced by the PU (*cf.*, segment-at-a-time mechanism discussed below.)

Result collection circuitry — After tuple processing, validated results are collected by the *Result Aggregation Buffer* (RAB), shown in Figure 3. The RAB is comprised of a structure of connected buffers (*i.e.*, *Buffer Nodes*) that are responsible for collecting results from two sources and guiding them from the OPBs to the output port of the FQP. In this collection step, a fairness granting mechanism makes sure that both sources are treated equally to avoid starvation. Appropriate tuning of *Buffer Node* parameters (*e.g.*, buffer size) and connectivity architecture is important as this affects overall FQP performance. In other words, poor assignment of parameters could result in bottlenecks in the transmission of resulting tuples, while the majority of buffers would be under-utilized.

4.3 Segment-at-a-time Processing

Not only queries change throughout the life of an application, but the streams themselves evolve as well. Their properties such as schema, tuple size, and input rate change continuously. These features are at odds with today's FPGA-based stream processing solutions, which have, for the most part, been tailored to process one specific tuple width before requiring re-synthesis if tuple size changes are permitted at all. This degree of flexibility poses a severe challenge for a hardware-based solution, as opposed to its software counter-part. Our design has been specifically built to afford this flexibility. The OPB-based design of FQP supports varying size tuples, thus, allowing for evolving data streams.

Generally speaking, hardware systems have fixed size input ports, internal communication buses, and output ports. FQP is no exception. However, flexibility in the face of varying size data streams stems from the way an OPB processes incoming tuples. The parametrized design of the OPB allows us to define its ports' width prior to design synthesis. By default, we configure FQP with a 64-bit port width. As a result, for any tuple larger than 64 bits, it is divided into 64-bit segments at the entry-point to FQP. The tuple segments arrive at the input port of an OPB as shown in Figure 4. Then, the OPB processes each segment, one at a time, and hands over the resulting segments to the *Filter Unit* through its *Final Result* port.

Figure 6 shows the segment-at-a-time processing mechanism in more details. Prior to processing a segmented tuple, queries also need to be updated to handle the segments. In our example, the query consists of two segments, of which the first segment corresponds to the first segment of the tuple, while the second segment of the query corresponds to the second segment of the tuple. Segmentation of queries is performed in software outside of FQP.

The PU fetches the first segment of the tuple from the *Window Buffer-R* as well as the first segment of the query. Then, the PU executes the segment of the query and produces a result segment with an additional flag which shows if the first segment of the tuple satisfies the (query) conditions in the first segment of the query. This process is repeated for the second segment of the tuple etc.

All resulting tuple segments are transmitted to and

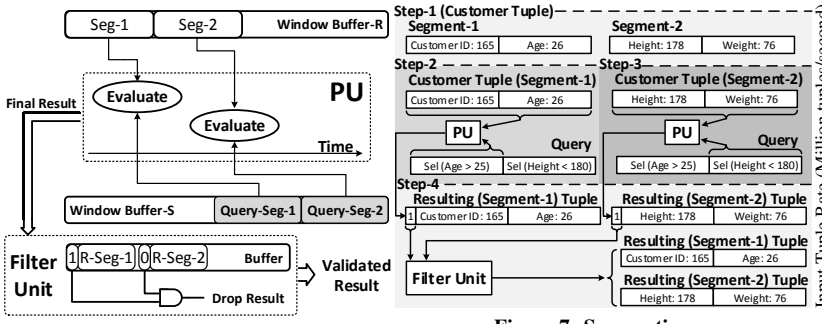


Figure 6: OPB **Segment-at-a-time.**

Figure 7: Segregation query.

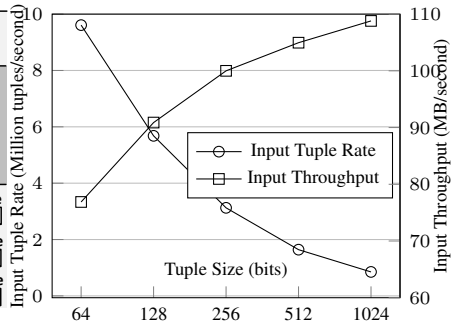


Figure 8: Effect of tuple size.

stored in internal buffers of the *Filter Unit* (FU), which evaluates the validity of the entire resulting tuple. For example, in a selection operator, one of the tuple segments may not pass the selection condition, while others do[3], which would render the entire tuple invalid. After receiving the final segment and positively validating the result, the FU hands the tuple (a segment at a time) over to the RAB to transfer it to the output port of FQP. Otherwise, the FU drops all result segments.

Segment-at-a-time for join operator – Query assignment is a task performed in software that maps the input queries onto the available blocks of the FQP configuration. This task determines the placement of operators, which is not known a priori (*i.e.*, we do not know where a join operator executes). Segment-at-a-time processing is necessary to support the further processing of tuples that result from a join operation as often, the join result is comprised of both input tuples (unless attributes are projected out).

Segment-at-a-time tuple size limit – The maximally accepted tuple size is determined by the size of *Window Buffer-(R&L)* in the OPB. From a conceptual point of view, the size of *Window Buffer-R* is not limited for stateless operators (*i.e.*, select and project), while this is not the case for stateful operators (*i.e.*, join). The actual limit depends on the resources available on the FPGA, which is highly device-specific and will only increase in future FPGAs. With today's technology, we have synthesized blocks with window sizes of up to 4K bytes.

5. VARIABLE TUPLE SIZE EXAMPLE

Here, we give an example to illustrate the segment--at-a-time mechanism realized by the OPBs. Assume a `Customer` stream with `Customer ID` and `Age` fields.

```
CREATE STREAM CS_SEL AS
SELECT *
  FROM Customer_Stream
 WHERE Age > 25
```

Furthermore, assume a query to segregate customers into two groups, those who are older and those who are younger than 25 years of age (*e.g.*, a retailer wanting to compute recommendations based on age.)

This query is programmed onto the OPB and executed over the customer tuples as shown in Figure 5. As the `Customer` stream evolves over time, new attributes, such as `Height` and `Weight`, are added (*e.g.*, for the retailer to better differentiate recommendations.)

```
CREATE STREAM CS_SEL AS
SELECT *
  FROM Customer_Stream
 WHERE Age > 25, Height < 180
```

Thus, the query is re-written as follows and through the segment-at-a-time mechanism, the OPB can execute the new query over the larger tuples without any changes as shown in Figure 7.

The processing of the updated (larger) tuple is done in four steps. In Step 1, after the query for the updated tuple schema was re-programmed onto its (target) OPB, the updated (enlarged) tuple is divided into two segments at the entry point of the FQP. In Step 2, that is, after the segments arrive at the target OPB, the *Processing Unit* fetches the first part of the query (`Age` $>$ 25) and executes it on the first segment of the tuple. In Step 3, the same process is repeated for the second part of the query (`Height` $<$ 180) and the second segment of the tuple. Each one of these steps produces a resulting tuple segment together with a validation flag. Finally, in Step 4, the resulting tuple segments are processed jointly using the *Filter Unit*. In case all segments have satisfied the query conditions, they are handed over to the RAB for transfer to the output port of the FQP. In this example, for illustration purposes, we have kept the data stream simple. In practice, segment-at-a-time is applicable to larger tuples with more attribute-value fields.

6. EXPERIMENTAL EVALUATION

We developed all FQP components in VHDL that are configured and synthesized on our Xilinx ML505 development board. In our experiments, the input was generated by a workload generator and passed through an Ethernet component and pipelined reception buffers to the FQP stream processor. The input streams consist of 64-bit long tuples (*i.e.*, 32-bit attribute and 32-bit value).

Raw processing power evaluation – We first present the raw processing power of various queries by focusing on the number of operators on a topology similar to Figure 1, where window size is 16 and clock frequency is 125MHz. For the selection and projection operators, OPB is capable of supporting |Window Buffer-L|/2 independent selection operators or |Window Buffer-L| independent projection operators. Each OPB is capable of realizing a single join operator. OPBs connected in

[3] *E.g.*, the higher order bits pass the condition, while the lower order ones do not (for two segments).

a chain (OP-Chain) can realize join operators with even larger window buffers. For example, utilizing two, three, or four OPBs increases the window size two, three, or four times, respectively. The processing performance of each OPB for the join operator tightly depends on its window buffers' sizes. For a window size of 16 tuples, the current version of OPB is capable of processing 1.44 million tuples per second. The raw processing power of the topology given in Figure 1 is summarized in Table 1.

Table 1: Tuple processing rate.

Operators	# Operators	Million Tuples/s
Selection	24×8	230.6
Projection	24×16	272.6
Join	24	34.5
Chained Join (4)	6	8.6

Each OPB is capable of processing at the rate of 9.61M, 11.36M, or 1.44M tuples per second for the selection, projection, and join operator, respectively, which translates to 230.6M, 272.6M, or 34.5M tuples per second for the topology in Figure 1. By chaining 4 OPBs we have 6 OP-Chains each with a window size of 4×16 and a total processing rate of 8.6M tuples per second.

Segment-at-a-time evaluation – To evaluate our segment-at-a-time feature of OPB, and to study its influences on the input rate, we utilized a data stream by varying the number of attribute-value pairs per tuple (1 to 16), in which the size of each attribute-value pair is 64 bytes. The clock frequency in this experiment was 125MHz. Figure 8 demonstrates the input tuple rate achieved as we feed larger tuples to an OPB. By feeding larger tuples, the sustainable input tuple rate decreases as expected, since the size of tuple and the number of attribute-value pairs doubles each time. However, interestingly for a double size tuple the processing time does not necessarily double as seen in this figure. This is due to the reduction in the amortized cost of tuple handling that is mostly for the first segment and decreases for the subsequent segments. These results are for the selection operator, but they are applicable for other operators including the projection and join operators.

7. CONCLUSIONS & OPEN PROBLEMS

Our broader vision is to identify key opportunities to exploit the strength of available hardware accelerators given the unique characteristics of stream processing. As a first step towards fulfilling this goal, we have developed FQP, a generic streaming architecture composed of a dynamically re-programmable stream processing elements, (*i.e.*, OPBs) that can be chained together to form a customizable processing topology (exhibiting a "Lego-like" connectable property). We argue that our proposed architecture may serve as a basic framework for both academic and industry research to explore and study the entire life-cycle of accelerating stream processing on hardware. Here we identify a list of important short- and long-term problems that can be tackled within our FQP framework.

• What is the complexity of query assignment to a set of custom hardware blocks (including but not limited to OPBs). Note, a poorly chosen query assignment may increase query execution time, leave some blocks unutilized, negatively affect energy use, and degrade the overall processing performance.

• How to formalize query assignment algorithmically (*e.g.,* develop a cost model), and what is the relationship between query optimization on hardware and classical query optimization in databases. Unlike the classical query optimization and plan generations, we are not just limited to join reordering and physical plan selections, but there is a whole new perspective on how to apply instruction-level and fine-level memory-access optimization (through custom hardware implementation, *e.g.*, different OPB implementations). For example, what is the most efficient method for wiring custom operators to minimize the routing distance? How to collect statistics during query execution and how to introduce dynamic re-wiring and movement of data given a fixed FQP topology?

• What is the best initial topology given a query workload as *a prior*? For example, one can construct a topology in order to reduce routing (*i.e.*, to reduce the wiring complexity) or to minimize chip area overhead (*i.e.*, to reduce the number of OPBs).

• Given the topology and the query assignment formalism, is it possible to generalize from single-query optimization to multi-query optimization, where we amortize executing cost across the shared processing of the n queries and explore inter- and intra-query optimization that are inspired by the capabilities of custom stream processors?

• Finally, how do we extend query execution on hardware to co-processor design by distributing and orchestration query execution over different hardware with unique features such as CPUs, FPGAs, and GPUs? An important design decision arises as to how these various devices communicate and whether or not they are placed on a single board, thus, having at least a shared external memory space, or placed on multi-boards and connected through interfaces such as PCIe.

8. REFERENCES

[1] Symmetric key cryptography on modern graphics hardware. Advanced Micro Devices, Inc., 2008.

[2] J. L. Hennessy and D. A. Patterson. *Computer Architecture, Fourth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., 2006.

[3] J. Lockwood, A. Gupte, N. Mehta, M. Blott, T. English, and K. Vissers. A low-latency library in FPGA hardware for high-frequency trading (HFT). In *HOTI*, 2012.

[4] R. Mueller, J. Teubner, and G. Alonso. Streams on wires: a query compiler for FPGAs. *VLDB*, 2009.

[5] M. Najafi, M. Sadoghi, and H.-A. Jacobsen. Flexible query processor on FPGAs. *VLDB*, 2013.

[6] M. Najafi, M. Sadoghi, and H.-A. Jacobsen. Configurable hardware-based streaming architecture using online programmable-blocks. In *ICDE*, 2015.

[7] M. Sadoghi, H.-A. Jacobsen, M. Labrecque, W. Shum, and H. Singh. Efficient event processing through reconfigurable hardware for algorithmic trading. In *VLDB*, 2010.

[8] M. Sadoghi, R. Javed, N. Tarafdar, R. Palaniappan, H. P. Singh, and H.-A. Jacobsen. Multi-query stream processing on FPGAs. In *ICDE*, 2012.

The BigDAWG Polystore System

Jennie Duggan
Northwestern

Aaron J. Elmore
Univ. of Chicago

Michael Stonebraker
MIT

Magda Balazinska
Univ. of Washington

Bill Howe
Univ. of Washington

Jeremy Kepner
MIT

Sam Madden
MIT

David Maier
Portland State Univ.

Tim Mattson
Intel

Stan Zdonik
Brown

ABSTRACT

This paper presents a new view of federated databases to address the growing need for managing information that spans multiple data models. This trend is fueled by the proliferation of storage engines and query languages based on the observation that "no one size fits all". To address this shift, we propose a *polystore* architecture; it is designed to unify querying over multiple data models. We consider the challenges and opportunities associated with polystores. Open questions in this space revolve around query optimization and the assignment of objects to storage engines. We introduce our approach to these topics and discuss our prototype in the context of the Intel Science and Technology Center for Big Data.

1. INTRODUCTION

In the past decade, the database community has seen an explosion of data models and data management systems, each targeted for a well-defined vertical market [3, 6]. These systems exemplify the adage that "no one size fits all" for data management solutions [21]. For example, relational column stores are poised to take over the data warehouse market; most of the major database vendors have embraced this technology. High-throughput OLTP workloads are largely moving to main memory SQL systems, with products available from multiple startups as well as Microsoft and SAP. Moreover, there has been a flood of NoSQL engines implementing a panoply of data models, and they typically operate on flexible storage formats such as JSON. The internet of things (IoT) calls for real-time stream processing and analytics that may be best served by either an OLTP engine or a stream processing engine. In addition, the jury is still out on the winning architecture for complex analytics and graph processing. Lastly, distributed file systems (a la HDFS) have become popular owing to their simple scalability and rich ecosystem of data processing tools.

Increasingly, we see applications that deploy multiple engines, resulting in a need to join data across systems. It is simply not reasonable for them to replicate all information on all platforms. This landscape calls for a new approach to federating data systems. At the Intel Science and Technology Center for Big Data, we have constructed a medical example of this new class of applications, based on the MIMIC II dataset [18]. These publicly available patient records cover 26,000 intensive care unit admissions at Boston's Beth Israel Deaconess Hospital. It includes waveform data (up to 125 Hz measurements from bedside devices), patient metadata (name, age, etc.), doctor's and nurse's notes (text), lab results, and prescriptions filled (semi-structured data). A production implementation would store all of the historical data augmented by real-time streams from current patients. Given the variety of data sources, this system must support an assortment of data types, standard SQL analytics (e.g., how many patients were given a particular drug), complex analytics (e.g., computing the FFT of a patient's waveform data and comparing it to "normal"), text search (e.g., finding patients who responded well to a particular treatment), and real-time monitoring (e.g., detecting abnormal heart rhythms).

Although it is conceivable to implement this entire application in a single storage system, the consequences of doing so would be dire. Not only would there be 1–2 orders of magnitude performance penalty on some of the workload, but the real-time requirements may not be achievable in a one-size-fits-all engine. In our reference implementation, which we call BigDAWG[1], we use SciDB [6] for archived time series data, Accumulo [1] for text, Postgres for patient metadata, and S-Store [8] for real-time waveform data. Obviously, there will be queries that span two or more storage engines. For example, to compare current waveforms to historical ones, one would query S-Store and SciDB. To find patient groups associated with particular kinds of prescriptions or doctor's notes, one would query Accumulo and Postgres. To run analytics on the waveforms from a particular cohort of patients, one would query Postgres and SciDB. It seems clear to us that a federation of multiple disparate systems will be a requirement in a myriad of future applications, especially those that are broad in scope like the MIMIC II example above.

We call such systems *polystores* to distinguish them from earlier federated databases that supported transparent access across multiple back ends with the same data model.

[1]So named after the ISTC Big Data Analytics Working Group

We begin by defining polystores and their architecture in Section 2. We then discuss our approach to query optimization within this framework in Section 3. After that, we turn to the monitoring system for query optimization and data placement. In Section 4, we discuss how our polystore model presents new challenges in assigning data to back ends. A discussion of prior work follows in Section 5.

2. POLYSTORE SYSTEMS

In this section, we present the semantic notion of polystores that we are exploring with our BigDAWG prototype. This approach is motivated by three goals. First, like previous federated databases, BigDAWG will support *location transparency* for the storage of objects. This transparency enables users to pose declarative queries that span several data management systems without becoming mired in the underlying data's present location or how to assign work to each storage engine. In this context, an *island of information* is a collection of storage engines accessed with a single query language. Section 2.1 explores this construct.

The second goal of our work is *semantic completeness*. A user will not lose any capabilities provided by his underlying storage engines by adding them to a polystore. Hence, BigDAWG will offer the union of the capabilities of its member databases. As we will see in Section 2.2, this calls for the system to support multiple islands.

Our third aim is to enable users to access objects in stored a single back end from multiple islands. For example, a user accessing MIMIC II waveform data may express real-time decision making in SQL with streaming semantics (e.g., "raise an alarm if the heart rate over this window exceeds some threshold"). On the other hand, he may express complex analytics using an array language with queries like, "compute the FFT over all heartrate waveforms, grouped by patient and day". For such a user, it is desirable to have the same data be queryable in both an array and a relational island.

Enterprises also have legacy systems that access newly federated datastores. Obviously, they want their existing applications to continue to run while simultaneously writing new ones in a more modern language. Again, we see a need for a single datastore to participate in many islands.

In summary, a polystore will support a many-to-many relationship between islands of information and data management systems over multiple, disparate data models and query languages. This framework is designed to minimize the burden associated with running many back ends while maximizing the benefits of the same.

2.1 Islands of Information

In BigDAWG we use islands of information as the application-facing abstraction with which a user interacts with one or more underlying data management engines. Specifically, we define an *island of information* as:

A data model that specifies logical, schema-level information, such as array dimensions or foreign key relationships.

A query language for that data model. It is with this language that users pose queries to an island.

A set of data management systems for executing queries written to an island. Here, each storage engine provides a *shim* for mapping the island language to its native one.

When an island receives a query, it first parses it to create an abstract syntax tree (AST) in the island's dialect. It also verifies that the query is syntactically correct. It then uses the query optimizer to slice the AST into subqueries, where subqueries are the unit with which the polystore assigns work to back ends. The island then invokes one or more shims to translate each subquery into the language of its target data store. The island next orchestrates the query execution over the engines and accumulates results. The island itself will do little computation other than concatenating query results from its data stores. This design permits us to take advantage of the optimizers in the back ends rather than managing data at the federator level.

Ideally, an island will have shims to as many databases as possible to maximize the opportunities for load balancing and query optimization. At the same time, individual users may be comfortable working in different query languages and data models. In general, we expect that users will desire islands for well-known data models, such as relational and streaming, and that these islands will overlap in practice. In BigDAWG, two systems developed by ISTC members will be used as initial islands, and we will build additional ones as we go along. The two initial systems are Myria [13], which uses relational-style semantics plus iteration, and D4M [15], which implements a novel data model based on associative arrays. In our prototype, both islands will mediate access to MIMIC II data stored in the same engines (Accumulo, SciDB, S-Store, and Postgres).

Neither island contains the complete functionality of any of the underlying engines. Differing support for data types, triggers, user-defined functions, and multiple notions of null are invariably system-specific. To satisfy our goal of semantic completeness, we require *degenerate* islands that have the complete functionality of each storage engine. Our initial prototype will have six islands: four degenerate ones (relational, array, streaming, and text) augmenting the two described above. In general BigDAWG will have as many islands as one wants, each providing location transparent access to one or more underlying storage engines.

2.2 Cross-Island Querying

An island within a polystore supports location transparency via a shim for each storage engine. For a storage engine to join an island, a developer writes a shim. If a single-island query accesses more than one storage engine, objects may have to be copied between local databases. Here, a *CAST* mechanism copies objects between back ends.

When a user command cannot be expressed in a single island's semantics, he must convey his query in multiple island languages, each of which is a subquery. To specify the island for which a subquery is intended, the user encloses

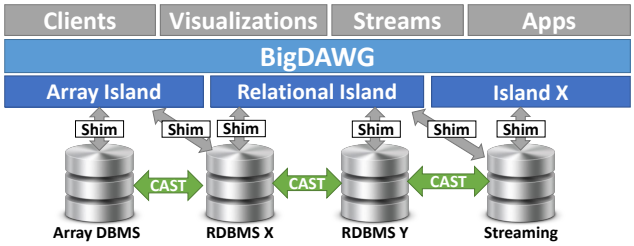


Figure 1: Polystore Architecture

his query in a *SCOPE* specification. A cross-island query will have multiple scopes to indicate the expected behavior of its subqueries. Likewise, a user may insert a CAST operation to denote when an object should be accessed with a given set of semantics.

For example, consider a cross-island operation, such as a join between an object in an array island and one in a table island. Obviously, we can CAST the array to the table island and do a relational join or we can do the converse and perform an array join. Since the each of these options produces a different output, a user must specify the semantics he desires using SCOPE and CAST commands. If he elects relational semantics, his query might be:

```
RELATIONAL(SELECT *
  FROM R, CAST(A, relation)
  WHERE R.v = A.v);
```

Here, the user specifies that the query will execute in a relational scope. From the user's perspective, the query will produce output as if A were translated into a table, shipped to a relational engine, and executed there. Because A is an array, the CAST converts it into a table when it is initially accessed. The user does not care whether the query is executed in the array store or relational one provided his prescribed island semantics are obeyed. Many queries may have implicit CASTS, and the polystore will insert these operations automatically as needed.

The full BigDAWG architecture is shown in Figure 1. This diagram shows multiple applications using a BigDAWG instance with three islands. Each island speaks to one or more engines through the use of shims. Objects can be CAST between engines, and only a subset of casts are shown for clarity. Here, a user issues a query to BigDAWG, and he specifies his commands using one or more island languages. Within an island, the polystore calls the shims needed to translate the query into the language(s) of the participating storage engines. When the query uses multiple islands, data may be shuffled among them using CAST operations.

As a result, the BigDAWG query language consists of the above SCOPE-CAST facility for a collection of islands over an overlapping set of storage engines. For simplicity, we leave the control of redundant copies of data objects for future work. In the rest of this paper we discuss our approach to query optimization and data placement.

3. QUERY OPTIMIZATION

In this section, we first introduce our approach to optimizing single-island queries. After that, we outline a mechanism for generating and managing the statistics needed for query optimization and data placement within a polystore. The section closes with a discussion of generalizing the optimizer to multi-island query planning.

3.1 Single Island Planning

Traditional query optimization is simply not capable of supporting cross-database queries. First, cost-based optimizers [19] require the planner to maintain a model for each operation to estimate its resource needs. This essentially obligates the optimizer to understand all of the operations in all storage engines as well as how the various shims work. Moreover, the primitives in each of the underlying storage engines may not map neatly to the operators in the island language. For example, a distributed array store may implement matrix multiplication with a purpose-built set of scatter-gather operations whereas a relational engine might categorize it as a group by aggregate. Reconciling these models would be non-trivial. Also, the optimizer would have to adapt whenever a new storage engine is added to the polystore. Lastly, conventional optimizers assume metadata about objects is available, such as their distribution of values. Local engines may or may not expose such information. As a result, we propose a black box approach in this section, whereby no information about the local optimizer is assumed.

If our query optimizer cannot be made robust using this approach, then we will selectively add more sophisticated knowledge of individual systems, recognizing that this may make adding new storage engines to a polystore more challenging. More detailed knowledge might include the sizes of operands, their data distribution, available access methods, and explanations of query plans.

We first consider simple queries, ones which have comparable performance on all of an island's storage engines. We anticipate that many select-project-join queries will be in this category and we will examine in how to identify such queries in Section 3.2.

Simple Queries For such queries we propose to minimize data movement among storage engines, so as to avoid costly data conversions and unnecessary network traffic. Rather, we should bring computation to the data whenever possible. Hence, we divide any simple query optimization into stages. In Stage 1 we perform all possible local computations that do not require any data movement. At the end of Stage 1, we are left with computations on collections of objects, where each one is located on a different storage engine. Since we have done all single-DBMS subqueries, we expect the number of remaining objects to be modest. In the next section we describe how to process this "remainder".

Complex Queries Now consider expensive operations, parts of the workload that have at least an order of magnitude

performance improvement relative to "one size fits all" architectures. Put differently, such operations are usually O(N^2) or O(N^3), and dwarf the cost of conventional data management commands, such as joins and aggregates. For example, the Genbase benchmark [22] demonstrated that complex analytics like linear regression enjoy a 10X speedup when run on an array store in comparison to a column store. For these queries, it will often make sense to move the data to a site where the high-performance implementation is available–which we learn empirically. The cost of the such moves will be more than amortized by the reduced execution time. To ensure a move is not too expensive, we will explore tactics for moving an object based on the engines involved, disk activity at both sites, and overall network activity.

In summary, we must pay careful attention to expensive operations. For polystores, we start by looking for select-project-join subqueries that are local to an object, and perform them in place as above. The "remainder" is expanded to include expensive functions, that we consider moving to a different site that offers high performance execution. The next section indicates how BigDAWG learns a preference list for assigning simple and complex queries to engines.

3.2 Workload Monitoring

To make efficient query optimization and data placement decisions, BigDAWG relies on black box performance profiling of its underlying systems. To rapidly acquire this information, the polystore query executor will have three modes of operation: training, optimized, and opportunistic. In training mode, BigDAWG has the liberty to run a subquery on all engines after inserting any casts that are needed. Optimized mode assigns the work of an incoming query to just one back end. Opportunistic mode measures the performance of the polystore engines by exploiting idle resources with an active learning approach.

Training For subqueries running in training mode, the federator records its elapsed time on each engine in an internal performance catalog. It will first run the leafs and branches from the AST at all possible locations in parallel to accumulate feedback. Since branches may involved multiple engines, there are many ways that a branch can be executed. Specifically, if there are N engines involved in a query, then there are $N!$ possibilities. In general we expect N to be small, and hence not a source of scalability issues.

Since training mode uses all engines, the optimizer will naturally have a comprehensive profile for each leaf and branch. Hence, it can develop a ranked *preference list* to determine the engine(s) best suited for each subquery. After that, new subqueries matching the characteristics of this subquery will need no additional experimentation.

Optimized If a query is run in optimized mode, then BigDAWG will execute no redundant computations for it. This mode is invoked in one of two cases: either the subquery has an existing preference list to guide its decision or there are not enough spare resources for expansive training. In the latter case, BigDAWG selects a database at random for query execution. Over time, the polystore builds up the same performance profiles offered more quickly by training mode.

Opportunistic BigDAWG maintains partial profiles on subqueries from optimized executions for further evaluation during periods of low system utilization. The system opportunistically executes the stored subqueries on new engines when their resources become available.

We expect a polystore to be in optimized mode most of the time or for it to start in training mode and gradually shift to optimized executions. In any case, over time BigDAWG assembles a database of subqueries and their duration on various engines. This monitoring framework is used to guide query optimization and data placement.

To accommodate complex operations, the experiments noted above will have a timeout. Hence, if an engine has not completed its trial within N multiples of the fastest running option it is halted. If objects are so large that data movement is problematic, we propose running a subquery on a statistical subset of the data to contain costs whenever possible. A subquery is then considered complex if there is significant variance in its runtime on different back ends.

3.3 Multi-Island Planning

We now consider queries that span multiple polystores. Here, the user is explicitly stating his semantics using CAST and SCOPE commands. Changing the order of these directives arbitrarily will generally result in ambiguous semantics in the query's execution. We will not support such indeterminateness. Instead, our main optimization opportunity is to identify circumstances where multiple islands have overlapping semantics. Consider an extension to the example query in Section 2.2:

```
RELATIONAL(SELECT count(*)
  FROM R, CAST(A, relation)
  WHERE R.v = A.v);
```

Here, the user issues the appropriate SCOPE-CAST query. This query, however, produces the same result, regardless of the join's island semantics. In other words, the join is sensitive to scope, but the subsequent aggregate masks these semantics. Hence, the query can be converted to the single-island query that is potentially amenable to traditional optimization. We will look for more instances where an island query may be simplified in this fashion.

We can also examine optimization opportunities between islands by analyzing their shims. For example, if two islands both have shims to the same back end, the optimizer may merge the output of their shims, deferring more of the query planning to a single storage engine's optimizer.

We will also investigate how to identify areas of intersection between multiple shims associated with the same storage engine. By identifying areas of the various island languages that produce the same query in the storage en-

gine's native dialect, we can rewrite the query in a single "intersection" island and further leverage the storage engine's query optimizer. More broadly, we will examine techniques for finding equivalences between the grammars of multiple islands. This will involve probing the space of queries in a manner similar as described for monitoring.

4. DATA PLACEMENT

Our polystore is an environment in which objects reside in some local execution engine. In a location transparent system it is possible to move objects without having to change any application logic. There are two reasons to move an object: load-balancing and optimization. In the former scenario, operations may take longer than necessary because one or more engines are overloaded. To shed load, objects must be relocated. In addition, for complex analytics the system would be more responsive if one or more objects were moved elsewhere because a faster implementation of the object's common workload is available. Since ultimately DBAs are in control of their data, our interface will have a mechanism for specifying that some objects are "sticky" and cannot be moved so that local control is supported. Non-sticky objects are available for relocation. For simplicity, we will initially consider a single client and later explore placement and load-balancing in the presence of concurrent queries. It is worth noting that if all access to underlying engines does not go through BigDAWG, then data placement decisions are likely to suboptimal.

We estimate the relative duration of each operation on each storage engine. From this, BigDAWG will predict the effect of placing object O_i on engine E_j. Its what-if analysis will estimate the effect of this move on both the performance of the source engine and any proposed destination. Hence, BigDAWG learns a matrix estimating the resource utilization of each object O_i on each storage engine E_j.

We initially assume that the various engines are not overloaded, and that early numbers represent an unloaded state. The polystore may identify overload situations by comparing the runtimes of queries to similar ones that were run in the past. For every engine E_j, we compute an average percentage deterioration of response time over all possible objects. When the percentage exceeds a threshold for a site E_j, we choose an object stored on that back end for relocation.

Our initial algorithm greedily selects the object O_i with maximum benefit to E_j and relocates it to the engine E_k with maximum headroom. Clearly this will not necessarily be the optimal choice, and we will explore other options.

So far, we have been assuming we are choosing the location of objects within a single island of information. As noted in the Section 2, there will typically be multiple islands with overlapping storage engines. This island abstraction imposes additional limitations on the operation of our polystore. Specifically, any object that is accessible in multiple islands may be moved to a different storage manager, but only to one with shims to all of its islands. Otherwise, some application programs may cease to work.

Each island maintains a set of metadata catalogs that contain the objects the island knows about along with the storage manager that houses that object. BigDAWG must be able to read all of these catalogs to implement its CAST-SCOPE mechanisms. As such, it is capable of identifying the collection of storage managers that may host a given object. This information is used to restrict data placement.

5. RELATED WORK

The federation of relational engines, or distributed DBMSs, was studied in depth during the 1980s and early 1990s. Here, Garlic [7], and TSIMMIS [9], and Mariposa [20] were early prototypes. The research community also explored distributed query processing, distributed transactions, and replica management. There was also research on federating databases with different schemas, and it focused on issues such as coalescing disparate schemas [4] and resolving semantics between applications [14]. More recently, Dataspaces were proposed as a mechanism for integrating data sources [12], however this approach has not seen large-scale adoption.

BigDAWG's approach is similar to the use of mediators in early federated database systems [7, 9, 23]. The use of shims to integrate individual databases is modeled on wrappers from these systems, and islands are similar to mediators in that they provide unified functionality over a set of disparate databases. However, mediators were often used to provide domain-specific functionality and did not span multiple data models. These systems focused on integrating independent sources, and did not consider the explicit controls, such as placement or replication, found in BigDAWG. We also will extend work on executing queries across several disjoint execution sites [5, 10]. This includes the ability to decompose queries into subqueries for partial execution [17], and adaptive query processing to handle uncertainty in database performance [2].

Historically, there has been limited interest in data federations inside the enterprise. Most enterprises have independent business units that are each in charge of their own data. To integrate disparate data sources, enterprises often use extract, transform and load (ETL) systems to construct a common schema, clean up any errors, transform attributes to common units, and remove duplicates. This *data curation* is an expensive process for constructing a unified database. In the past it was common practice to simply load curated information into a relational data warehouse for querying.

These data warehouses are incapable of dealing with real-time data feeds, as they are optimized for bulk loading. In addition, most are not particularly adept at text, semi-structured data, or operational updates. As such, we expect revitalized interest in federated databases as enterprises cope with these shortcomings.

Recently, many vendors have advocated "data lakes" wherein an organization dumps its data sources into a repository–often

based on the Hadoop ecosystem. However, Hadoop's lack of transactions, batch-oriented processing, and limited functionality means it is not often an ideal solution. As such, it is a plausible historical archive, but it would still require an additional polystore for the interactive and real-time components of the workload. Several recent research prototypes have explored multi-database systems that couple Hadoop with relational databases [11, 16].

6. SUMMARY

We believe that polystore systems will be a critical component as data management needs diversify. Here we presented our vision for a next-generation data federation, BigDAWG, that offers full functionality and location transparency through the use of explicit scopes and casts. With these primitives, we outlined tractable research agendas in query optimization and data placement.

7. REFERENCES

[1] Accumulo. https://accumulo.apache.org/.

[2] L. Amsaleg, A. Tomasic, M. J. Franklin, and T. Urhan. Scrambling query plans to cope with unexpected delays. In *Fourth International Conference on Parallel and Distributed Information Systems, 1996*, pages 208–219. IEEE, 1996.

[3] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS*, pages 1–16. ACM, 2002.

[4] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.

[5] L. Bouganim, F. Fabret, C. Mohan, and P. Valduriez. A dynamic query processing architecture for data integration systems. *IEEE Data Eng. Bull.*, 23(2):42–48, 2000.

[6] P. G. Brown. Overview of scidb: large scale array storage, processing and analysis. In *SIGMOD*, pages 963–968. ACM, 2010.

[7] M. J. Carey, L. M. Haas, P. M. Schwarz, M. Arya, W. F. Cody, R. Fagin, M. Flickner, A. W. Luniewski, W. Niblack, and D. Petkovic. Towards heterogeneous multimedia information systems: The Garlic approach. In *Data Engineering: Distributed Object Management*, pages 124–131. IEEE, 1995.

[8] U. Cetintemel, J. Du, T. Kraska, S. Madden, D. Maier, J. Meehan, A. Pavlo, M. Stonebraker, E. Sutherland, and N. Tatbul. S-Store: A Streaming NewSQL System for Big Velocity Applications. *PVLDB*, 7(13), 2014.

[9] S. Chawathe, H. G. Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS Project: Integration of Heterogeneous Information Sources. In *IPSJ*, 1994.

[10] A. Deshpande and J. M. Hellerstein. Decoupled query optimization for federated database systems. In *ICDE*, pages 716–727. IEEE, 2002.

[11] D. J. DeWitt, A. Halverson, R. Nehme, S. Shankar, J. Aguilar-Saborit, A. Avanes, M. Flasza, and J. Gramling. Split query processing in polybase. SIGMOD, pages 1255–1266, 2013.

[12] M. Franklin, A. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *Sigmod Record*, 34(4):27–33, 2005.

[13] D. Halperin, V. Teixeira de Almeida, L. L. Choo, S. Chu, P. Koutris, D. Moritz, J. Ortiz, V. Ruamviboonsuk, J. Wang, A. Whitaker, et al. Demonstration of the Myria big data management service. In *SIGMOD*. ACM, 2014.

[14] R. Hull. Managing semantic heterogeneity in databases: a theoretical prospective. In *PODS*, pages 51–61. ACM, 1997.

[15] J. Kepner, W. Arcand, W. Bergeron, N. Bliss, R. Bond, C. Byun, G. Condon, K. Gregson, M. Hubbell, and J. Kurz. Dynamic distributed dimensional data model (d4m) database and computation system. In *ICASSP*. IEEE, 2012.

[16] J. LeFevre, J. Sankaranarayanan, H. Hacigümüs, J. Tatemura, N. Polyzotis, and M. J. Carey. MISO: souping up big data query processing with a multistore system. In *SIGMOD*, pages 1591–1602, 2014.

[17] L. M. Mackinnon, D. H. Marwick, and M. H. Williams. A model for query decomposition and answer construction in heterogeneous distributed database systems. *Journal of Intelligent Information Systems*, 11(1):69–87, 1998.

[18] M. Saeed, M. Villarroel, A. T. Reisner, G. Clifford, L.-W. Lehman, G. Moody, T. Heldt, T. H. Kyaw, B. Moody, and R. G. Mark. Multiparameter Intelligent Monitoring in Intensive Care II (MIMIC-II): A public-access intensive care unit database. *Critical Care Medicine*, 39:952–960, 2011.

[19] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *SIGMOD*, pages 23–34. ACM, 1979.

[20] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: a wide-area distributed database system. In *The VLDB Journal*, volume 5, pages 48–63. Springer, 1996.

[21] M. Stonebraker and U. Cetintemel. "One Size Fits All": An Idea Whose time has come and gone. In *ICDE*, pages 2–11, 2005.

[22] R. Taft, M. Vartak, N. R. Satish, N. Sundaram, S. Madden, and M. Stonebraker. Genbase: A complex analytics genomics benchmark. In *SIGMOD*, pages 177–188. ACM, 2014.

[23] G. Wiederhold. Mediators in the architecture of future information systems. *Computer*, pages 38–49, 1992.

Database Challenges for Exploratory Computing

Marcello Buoncristiano[1], Giansalvatore Mecca[1], Elisa Quintarelli[2]
Manuel Roveri[2], Donatello Santoro[1], Letizia Tanca[2]

[1]Università della Basilicata – Potenza – Italy
[2]Politecnico di Milano – Milano – Italy

1. INTRODUCTION

Helping users to make sense of very big datasets is nowadays considered an important research topic. However, the tools that are available for data analysis purposes typically address professional *data scientists*, who, besides a deep knowledge of the domain of interest, master one or more of the following disciplines: mathematics, statistics, computer science, computer engineering, and programming.

On the contrary, in our vision it is vital to support also different kinds of users who, for various reasons, may want to analyze the data and obtain new insight from them. Examples of these *data enthusiasts* [4, 9] are journalists, investors, or politicians: non-technical users who can draw great advantage from exploring the data, achieving new and essential knowledge, instead of reading query results with tons of records.

The term *data exploration* generally refers to a data user being able to find her way through large amounts of data in order to gather the necessary information. A more technical definition comes from the field of statistics, introduced by Tukey [12]: with *exploratory data analysis* the researcher explores the data in many possible ways, including the use of graphical tools like boxplots or histograms, gaining knowledge from the way data are displayed.

Despite the emphasis on visualization, exploratory data analysis still assumes that the user understands at least the basics of statistics, while in this paper we propose a paradigm for *database exploration* which is in turn inspired by the *exploratory computing* vision [2]. We may describe exploratory computing as the step-by-step "conversation" of a user and a system that "help each other" to refine the data exploration process, ultimately gathering new knowledge that concretely fullfils the user needs. The process is seen as a conversation since the system provides active support: it not only answers user's requests, but also suggests one or more possible actions that may help the user to focus the exploratory session. This activity may entail the use of a wide range of different techniques, including the use of statistics and data analysis, query suggestion, advanced visualization tools, etc.

The closest analogy [2] is that of a human-to-human dialogue, in which two people talk, and continuously make reference to their lives, priorities, knowledge and beliefs, leveraging them in order to provide the best possible contribution to the dialogue. In essence, through the conversation they are exploring themselves as well as the information that is conveyed through their words. This exploration process therefore means investigation, exploration-seeking, comparison-making, and learning altogether. It is most appropriate for *big collections of semantically rich data*, which typically hide precious knowledge behind their complexity.

In this broad and innovative context, this paper intends to make a significant step further: it proposes a model to concretely perform this kind of exploration over a database. The model is general enough to encompass most data models and query languages that have been proposed for data management in the last few years. At the same time, it is precise enough to provide a first formalization of the problem and reason about the research challenges posed to database researchers by this new paradigm of interaction.

2. A MOTIVATING EXAMPLE

We illustrate the process of exploring a large and semantically rich dataset using as example the database of recordings of the fitness tracker AcmeBand, a wrist-worn smartband that continuously records user steps, and can be used to track sleep at night.

Our user is allowed access to the measurements of a large fragment of AcmeBand users; in this example, for the sake of simplicity we shall assume that the database is a relational one, and focus on conjunctive queries as the query language of reference. However, we want to emphasize that the techniques

discussed in this paragraph can be applied to any data model that is based on the primitives of object collection, attribute and attribute value, and object reference: as a consequence our approach may incorporate the majority of data models that have been proposed in the last few years to model rich data. In our simplified case, the database has the following structure:

(i) a AcmeUser(id, name, sex, age, cityId) table with user data;

(ii) a Location(id, cityName, state, region) table to record location data about users (here region may be east, west, north or south);

(iii) an Activity(id, type, date, start, length, userId) table to record step counts for user activities of the various kinds (like walks, runs, cycling etc.);

(iv) a Sleep(id, date, start, length, quality, userId) table to record user sleep and its quality (like deep sleep, restless etc.).

We notice that the database may be quite large, even if the number of users is limited and the timeframe for activities and sleep restricted. Our casual exploratory user intends to acquire some knowledge about fitness levels and sleep habits of this fragment of AcmeBand users.

We do not assume any a-priori knowledge of a database query language, nor the availability of precomputed summaries as the ones that are typically used in OLAP applications. On the contrary, the system we have in mind should be able to guide and support our users throughout their (casual) information-seeking tasks.

2.1 Starting the Conversation

We envision this process as a *conversation* between the exploratory user and the system, where s/he provides some initial hints about her/his interests, and the system suggests "potentially interesting perspectives" over the data that may help to refine the search.

From the technical viewpoint, the conversation is modeled as a lattice of nodes. Each node is a *view* over the database described intensionally as a conjunctive query, and therefore represents a subset of database objects (*tuples*, in our relational case). To formalize this notion, we assume we are given a relational database schema $\mathcal{R} = \{R_1, \ldots, R_k\}$, and an instance I of $\mathcal{R}$, defined as usual. We introduce the notion of a *tuple-set* $\mathcal{T}^Q$ as the result of any conjunctive query Q over I, i.e., $\mathcal{T}^Q = Q(I)$.

In our example, we assume that the user does not have clear goals, and therefore expects the system to suggest some promising starting points for the exploration. However, we will see that our model can also handle the alternative scenario in which the user starts the conversation by formulating some explicit, albeit vague queries.

Given our sample database, to start the conversation the system suggests some initial *relevant features*. We need to make this notion more precise, but for the time being consider that, in the relational context, we may assume that a feature is the set of values taken by one or more attributes in the tuple-sets of the view lattice, while relevance is defined starting from the statistical properties of these values. As concrete examples, consider that to trigger the exploration the system might suggest something along the following lines:

($\mathcal{S}_1$) "It might be interesting to explore the types of activities. In fact: *running* is the most frequent activity (over 50%), *cycling* the least frequent one (less than 20%)";

($\mathcal{S}_2$) "It might be interesting to explore the sex of users with running activities. In fact: more than 65% of the runners are male";

($\mathcal{S}_3$) "It might be interesting to explore differences in the distribution of the length of the running activities between male and female. In fact: male users generally have longer running activities".

2.2 A Model of Relevance

Before going on with our example, we want to discuss how relevant features are extracted. The notion of relevance is based on the frequency distribution of attributes in the view lattice. To start the conversation, the system populates the lattice with a set of initial views. These will typically correspond to the database tables themselves, and joins thereof according to foreign keys. Broadly speaking, each view node can be seen also as a *concept* of the conversation, described by a sentence in natural language. For example, the node corresponding to the Activity table represents an "activity" concept (a *root* concept, from where the system may proactively start the conversation), while the join of AcmeUser and Location represents the concept of "user location", and so on.

The system builds histograms for the attributes of these views, and looks for those that might have some interest for the user. We adopt a comparative notion of relevance which has an information-theoretic root, and look for attributes that show some significant deviation w.r.t. an expected distribution. From the practical viewpoint, for each attribute we identify a number of reference distributions that correspond to expected – or *uninteresting* – ones, and deem the attribute *relevant* when its distribution shows some clear difference wrt the

reference ones. To do this, we assume the availability of a statistical similarity test, denoted by $test(d, d')$, for value distributions d, d'. Examples of this test are statistical hypothesis tests (as the ones mentioned in Section 4).

For a root concept, the reference distribution may be the uniform one, although different choices are possible according to the semantics of the attribute. In our example, the type attribute of Activity is considered as potentially relevant (suggestion $\mathcal{S}_1$ above), since its distribution shows significant statistical difference, according to *test*, w.r.t. to the uniform distribution. Note that $\mathcal{S}_1$ provides, along with a description of the relevant feature, some values as an explanation of these differences.

For views that are lower in the node lattice the notion of relevance is slightly more sophisticated. To introduce this, we need to formalize the lattice structure of views and tuple-sets. In our simple relational example we do so by reasoning about the relationship among conjunctive queries: we say that the tuple-set $\mathcal{T}^Q$ returned by a query Q over $\mathcal{R}$ *derives* from the tuple-set $\mathcal{T}^{Q'}$ returned by a query Q' if Q is obtained from Q' by adding one or more selections, one or more projections, one or more joins. We denote this by $Q \preceq Q'$, and, in turn, $\mathcal{T}^Q \preceq \mathcal{T}^{Q'}$.

In our example, the view AcmeUser $\bowtie$ Location derives from views AcmeUser and Location. Similarly, view $\sigma_{type='running'}$(AcmeUser $\bowtie$ Activity) derives from AcmeUser and Activity, but also from view $\sigma_{type='running'}$(Activity).

If we add, as a convention, a *top view* Q_{top} such that $Q \preceq Q_{top}$ for any query Q, it is possible to see that the relationship "derives from" among views over a schema $\mathcal{R}$ induces a join-semilattice.

Notice that the lattice also encodes a taxonomy: whenever the user imposes a restriction over the current view (concept), this amounts to somehow identifying one of its "sub-concepts". The lattice does not need to be pre-computed, and will be typically constructed in a dynamic fashion.

Given a tuple-set $\mathcal{T}$, and an attribute A in $\mathcal{T}$, we compute the distribution of values for A in $\mathcal{T}$, i.e., its $\mathcal{T}$-histogram. To formalize this notion of relevance we notice that, whenever a query Q derives from Q', it is possible to keep track of the relationship among the attributes of $Q(I)$ and those of $Q'(I)$.[1] To uniquely identify attributes within views, we denote attribute A in table R by the name R.A. We say that attribute R.A in $Q(I)$ *matches* attribute R.A of any $Q'(I)$ that is an ancestor or descendant of $Q(I)$ in the query lattice.

We can now formalize the notion of *relevance*. We say that attribute R.A of node $Q(I)$ is *relevant* if its distribution d is statistically different from the distribution of any matching attribute from any ancestor node. Note that the case of the root is a special one: we propose to assume the distribution of an attribute in the root node as the "default one" for that attribute; however the system administrator can modify this setting based on his/her knowledge of the domain of interest.

2.3 How the Conversation Goes On

We have now all the tools in place to present an example of a complex conversation. Assume as a first step the user is presented with items $\mathcal{S}_1$-$\mathcal{S}_3$ above, and selects item $\mathcal{S}_1$:

($\mathcal{S}_1$) "It might be interesting to explore the types of activities. In fact: *running* is the most frequent activity (over 50%), *cycling* the least frequent one (less than 20%)".

This choice is interpreted by the system as an interaction with the lattice of views. The user has selected view Activity, and relevant feature type. The system shows a subset of values for the type attribute, the ones that justify its relevance. The user may pick one or more of these values: assume s/he selects value *running*. This is interpreted by the system as some interest in the set of tuples that correspond to running activities. As a consequence, a new view is generated and added to the lattice:

$$\sigma_{\mathsf{type}=running}(\mathsf{Activity})$$

The addition of a new node to the lattice triggers a new interaction, with the system trying to suggest new and relevant hints. To do this, it may add further nodes. Here, it finds that a relevant feature is related to the region of runners (users that perform running activities). Notice that this requires to compute a view:

$$\sigma_{\mathsf{type}=running}(\mathsf{Location} \bowtie \mathsf{AcmeUser} \bowtie \mathsf{Activity})$$

The system realizes that the distribution of the region attribute within this view is significantly different both from the uniform distribution, and from the ones of the matching attributes in ancestors. Assume, in fact, that users are evenly distributed across regions. On the contrary, runners are especially active in the west. Among the potentially relevant features, the system will suggest that:

($\mathcal{S}_{1.1}$) "It might be interesting to explore the region of users with *running* activities. In fact: while users are evenly distributed across regions, 65% of users with *running* activities are in the *west* and only 15% in the *south*".

[1] For the sake of simplicity, here we do not consider self-joins, i.e., joins of a table with itself. With a bit more work the definition can be extended to handle self-joins

The user will select the suggestion ($\mathcal{S}_{1.1}$), and then pick up value *south*. The process is triggered again, and the system realizes that:

($\mathcal{S}_{1.1.1}$) "It might be interesting to explore the sex of users with *running* activities in the *south* region. In fact: while sex values are usually evenly distributed among users, only 10% of users with *running* activities in the *south* are *women*".

It can be seen that the discovery of this new relevant feature requires to compute the following view and add the corresponding node to the lattice:

$$\sigma_{\mathsf{type}=running,\mathsf{region}=south,\mathsf{sex}=female}(\mathsf{Location} \bowtie \mathsf{AcmeUser} \bowtie \mathsf{Activity})$$

After s/he has selected the suggestion ($\mathcal{S}_{1.1.1}$), the user is effectively exploring the set of tuples corresponding to female runners in the south. S/he may decide to ask the system for further advices, or browse the actual database records, or use an externally computed distribution of values to look for relevant features. Assume, for example, s/he is interested in studying the quality of sleep. S/he downloads from a Web site an Excel sheet stating that over 60% of the users complain about the quality of their sleep. When these data are imported into the system, the system suggests that, unexpectedly, 85% of sleep periods of southern women that run are of good quality. Having learned new insight about the data, the user is satisfied.

From the technical viewpoint, the system has constructed a complex lattice of views, and the conversation consists of a walk of this lattice.

3. RESEARCH CHALLENGES

The exploration model we have presented poses quite a number of technical challenges to database researchers. In this section, we overview some of these and highlight the literature that tackles some aspects.

The most related research topic is *faceted search*, also called faceted navigation, which (often visually) supports the access to information items based on a taxonomy of *facets* [13] (roughly our *features*). In faceted search, items are initially classified according to a given taxonomy. The user can browse the set making use of the facets and of their values (e.g. the feature *Activity*, or the value *running*) and will inspect the set of items that satisfy the selection criteria. Faceted search has similar aims w.r.t. database exploration and would greatly benefit of the foundational model we propose, which can support all its complexity and scalability challenges.

Responsiveness To guarantee a satisfactory user experience we need a "reasonably fluent" conversation: users will not tolerate long computing times. This imposes strong constraints over the computation of the view lattice and related feature statistics. Let us first distinguish the case in which the database is static from the one in which it may receive updates. In the first case a portion of the view lattice and the related features may be precomputed. In the second case, this might not be acceptable, at least in case of high update frequencies.

Responsiveness is related to a research area that is traditional for databases: the DBMSs build and maintain *histograms* representing the distribution of attribute values to estimate the (possibly joint) selectivity of attribute values for use in query optimization [3]. Fast, incremental histogram computation is a good example of a technique that can be effectively employed for speeding up the assessment of relevance in a conversation step.

Summarization In both the above cases, fully pre-computing the lattice seems unreasonable, especially for truly big data. This imposes to study two different optimization directions.

On the one side, it suggests to work with data summaries, the size of which must be large enough to estimate statistical parameters and distributions, but manageable from the computation viewpoint. To solve this problem many summarization techniques can be explored. Some works, for instance [1], have proposed to extract knowledge by means of data mining techniques, and to use it to intensionally describe sets in an approximate way. To summarize the contents of large relations and permit efficient retrieval of approximate query results, authors in [11] have used histograms, presenting a histogram algebra for efficiently executing queries on the histograms instead of on the data, and estimating the quality of the approximate answers. The latter research approaches can be used to support query response during the conversation.

On the other side, we need effective pruning techniques for the view lattice: not all node-paths are interesting from the user viewpoint, and the ones that fail to satisfy this requirement should be discarded as soon as possible. This introduces interesting problems related to detecting the different relevance that a feature might have *depending on the users*: a feature may be relevant if it is *different from*, or maybe *close to*, the user's expectations; in their turn, the users' expectations may derive from their previous background, common knowledge, previous exploration of other portions of the database, etc. In Section 4 we discuss how to define notions of relevance that are more general than the

one proposed by the motivating example.

CPUs vs GPUs Another open problem is if traditional CPU-RAM-disk architectures are fast enough for this purpose, or we need to resort to different ones. GPUs seem promising, provided that the memory limitations are handled, and avoid bottlenecks related to data transfers. Fast implementations of statistical computations over the GPU are a promising research direction [8, 10].

Fast Statistical Operators The previous points bring us to another fundamental requirement: the availability of fast operators to compute statistical properties of the data.

Surprisingly, despite years of data-mining research, there are very few research proposals towards the goal of fast computing statistical parameters, and comparing them. Subgroup discovery [6, 5], endeavors to discover the maximal subgroups of a set of objects that are statistically "most interesting" with respect to a property of interest. Subgroup discovery has been proposed in many flavors, differing as for the type of encompassed target variable, the description language, the adopted quality measure and the search strategy. Again, we believe that some of its techniques might be profitably adopted to assess the relevance of features in each of our exploration steps.

User Involvement A fundamental aspect to bear in mind when designing a database exploration tool is the relevance of an answer for a specific user: Section 4 discusses some of the technical challenges posed by the need to compute relevance in a fast and effective way.

Given that, having to deal with the interpretation of the user expectations, relevance remains largely a subjective factor, the immediately-related research challenge is to predict users' interests and anticipations in order to issue the most relevant (possibly serendipitous) answer. Therefore, user studies are crucial to classify different notions of relevance and devise strategies to customize the system response to the user behavior. This requires to identify real scenarios, and involve user groups to the end of understanding how they access data and what they perceive as the data-exploration needs addressed by current systems.

This quest for an effective user interaction immediately raises the important issue of developing intuitive visualization techniques. An exploratory interface should support appealing, synthetic visualization of the query answers. It should also be able to highlight the relevant properties of current and past query answers, in order for the users to get the gist of the data and decide the next step.

4. A STEP FORWARD

To make the vision outlined in this paper more concrete, we now describe a number of techniques that may contribute to solve the technical problem of implementing a database-exploration system.

The preliminary, critical step is the development of a statistical algorithm to measure the difference between two tuple-sets $\mathcal{T}^{Q_1}$ and $\mathcal{T}^{Q_2}$ with a common target feature, in order to compute the relative relevance. We discuss the very general setting in which the two tuple-sets may have arbitrary origins, and not necessarily are the result of queries that are one the refinement of the other. This generality is needed to accommodate the various needs discussed in the previous sections, capturing a flexible notion of relevance that accounts for different kinds of user expectations.

The method relies on an ensemble of hypothesis tests operating on randomly-extracted subsets of the original tuple-sets. The main intuition is that hypothesis tests should be conducted incrementally, in order to increase scalability, while at the same time keeping the emergence of false positives under control by means of the standard Bonferroni correction [7].

The process relies on four steps that are conducted iteratively. We first briefly discuss the steps, and then give an example.

Step 1 – Sampling: We sample tuple sets $\mathcal{T}^{Q_1}$ and $\mathcal{T}^{Q_2}$ to extract subsets q_1 and q_2 of cardinality much lower than the one of $\mathcal{T}^{Q_1}$, $\mathcal{T}^{Q_2}$, i.e., $|q_i| \ll |\mathcal{T}^{Q_i}|$. This can be done using different sampling strategies: sequential, random or hybrid.

Step 2 – Comparison: Let X_1 and X_2 be the projections of q_1 and q_2 over a specific attribute (feature). The data in X_1 and X_2 can be either numerical or categorical. The comparison step aims at assessing the discrepancy between X_1 and X_2 through theoretically-grounded statistical *hypothesis tests*, of the form $test_i(X_1, X_2)$. Examples of these tests are [7]: (i) *the two-sided t-test*, assessing variations in the mean value of two Gaussian-distributed subsets; (ii) *the two-sided Wilcoxon rank sum test*, assessing variations in the median value of two distribution-free subsets; (iii) *the one/two-sample Chi-square test*, assessing the distribution of a subset with discrete values w.r.t. a reference distribution or assessing variations in proportions between two subsets with discrete values; and (iv) *the one/two-sample Kolmogorov-Smirnov test*, to assess whether a subset comes from a reference continuous probability density function or whether two subsets have been generated by two continuous dif-

ferent probability density functions.

Step 3 – Iteration: We repeat the extraction and comparison steps M times. At the j-th iteration, a new pair of subsets X_1 and X_2 are extracted and $test_i(X_1, X_2)$ is computed. If the test rejects the null hypothesis, we stop the incremental procedure since we have enough statistical confidence that there is a difference in the data distributions of $\mathcal{T}^{Q_1}$ and $\mathcal{T}^{Q_2}$. Otherwise, the procedure proceeds to the next iteration.

Step 4 – Query ranking: The procedure described above can be applied to different pairs of tuple sets. The difference between their empirical distributions is computed using the *Hellinger distance*. Based on this, we can rank the tuple sets to find out those exhibiting the largest differences.

This technique could be applied to the examples $(\mathcal{S}_1)$-$(\mathcal{S}_3)$ in Section 2.1, by considering the following hypothesis tests:

(i) the one-sample Chi-square test can be used to assess whether the types of activities follow a uniform discrete distribution. This would allow to reject the null hypothesis that all the types of activities are equally frequent (i.e., *running* is the most frequent activity).

(ii) the two-sample Chi-square test can be used to assess whether the sex of users with *running* activities is equally distributed. This would allow to reject the null hypothesis of equal probability between male and female (i.e., male is the most prominent sex among the runners).

(iii) the two-sample Kolmogorov-Smirnov test can assess whether a statistical difference in the distribution of the length of the *running* activities between male and female runners exists. This would allow to asses that the distribution of activity lengths for male runners is statistically different from that of female ones (i.e., male runners are generally characterized by longer running activities).

Notably, the above method might be viewed as a subgroup discovery technique with the following distinctive features: (i) it manages both categorical and numerical attributes; (ii) it represents subgroups as SQL queries; (iii) the classification of attributes into unusualness or interest comprises the use of statistical hypothesis tests and the Hellinger distance; (iv) the search of relevant attributes relies on the joint use of sampling and incremental mechanisms for statistical hypothesis tests.

Actually, this is only one of the many possibilities to assess relevance during the conversation. We believe that the exploratory paradigm we are proposing lends itself to a plethora of variants and brand-new ideas, providing as many interesting challenges to database researchers.

5. REFERENCES

[1] E. Baralis, P. Garza, E. Quintarelli, and L. Tanca. Answering XML queries by means of data summaries. *TODS*, 25(3), 2007.

[2] N. D. Blas, M. Mazuran, P. Paolini, E. Quintarelli, and L. Tanca. Exploratory computing: a challenge for visual interaction. In *AVI*, pages 361–362, 2014.

[3] P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. In *PVLDB*, pages 466–475, 1997.

[4] P. Hanrahan. Analytic database technologies for a new kind of user: the data enthusiast. In *SIGMOD*, pages 577–578, 2012.

[5] F. Herrera, C. J. Carmona, P. González, and M. J. del Jesús. An overview on subgroup discovery: foundations and applications. *Knowl. Inf. Syst.*, 29(3):495–525, 2011.

[6] W. Klösgen. Explora: A multipattern and multistrategy discovery assistant. In *Advances in Knowledge Discovery and Data Mining*, pages 249–271. 1996.

[7] E. L. Lehmann and J. P. Romano. *Testing statistical hypotheses*. Springer Science & Business Media, 2006.

[8] U. Milic, I. Gelado, N. Puzovic, A. Ramirez, and M. Tomasevic. Parallelizing general histogram application for cuda architectures. In *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII)*, pages 11–18, 2013.

[9] K. Morton, M. Balazinska, D. Grossman, and J. D. Mackinlay. Support the data enthusiast: Challenges for next-generation data-analysis systems. *PVLDB*, 7(6):453–456, 2014.

[10] C. Nugteren, G. van den Braak, H. Corporaal, and B. Mesman. High performance predictable histogramming on gpus: Exploring and evaluating algorithm trade-offs. In *Proc. of the Fourth Workshop on GPGPU*, pages 1:1–1:8. ACM, 2011.

[11] V. Poosala, V. Ganti, and Y. E. Ioannidis. Approximate query answering using histograms. *IEEE Data Eng. Bull.*, 22(4):5–14, 1999.

[12] J. W. Tukey. *Exploratory data analysis*. Addison-Wesley, Reading,MA, 1977.

[13] D. Tunkelang. *Faceted Search*. Synthesis Lectures on Information Concepts, Retrieval, and Services. Morgan & Claypool Publishers, 2009.

On-the-Fly Data Synopses: Efficient Data Exploration in the Simulation Sciences

Thomas Heinis[†], David A. Ham[‡]

[†]*Department of Computing* [‡]*Department of Mathematics*

Imperial College, London, UK

ABSTRACT

As a consequence of ever more powerful computing hardware and increasingly precise instruments, our capacity to produce scientific data by far outpaces our ability to efficiently store and analyse it. Few of today's tools to analyse scientific data are able to handle the deluge captured by instruments or generated by supercomputers.

In many scenarios, however, it suffices to analyse a small subset of the data in detail. What scientists analysing the data consequently need are efficient means to explore the full dataset using approximate query results and to identify the subsets of interest. Once found, interesting areas can still be scrutinised using a precise, but also more time-consuming analysis. Data synopses fit the bill as they provide fast (but approximate) query execution on massive amounts of data. Generating data synopses after the data is stored, however, requires us to analyse all the data again, and is thus inefficient.

What we propose is to generate the synopsis for simulation applications on-the-fly when the data is captured. Doing so typically means changing the simulation or data capturing code and is tedious and typically just a one-off solution that is not generally applicable. In contrast, our vision gives scientists a high-level language and the infrastructure needed to generate code that creates data synopses on-the-fly, as the simulation runs. In this paper we discuss the data management challenges associated with our approach.

1. INTRODUCTION

Over the last half century, computer simulations of natural phenomena have become an indispensable part of the scientific method [2, 17]. For the study of many natural phenomena, building a model and simulating it complements the understanding developed using traditional methods. In fields as diverse as cosmology, seismology and climate science, simulation is essential in understanding phenomena when a physical experiment is impossible.

The models used today are as big and detailed as the memory of the simulation infrastructure allows, and with every successive hardware generation, the models grow larger. As the computing infrastructure evolves, the amount of data generated increases. Even today, datasets may be as large as petabytes and are growing rapidly [10], making them difficult to store.

Storage capacity, however, is only one aspect of the problem. The data generated by simulations is so big that the analysis of the data is also a severe challenge. Complete analysis, however, is rarely required and what scientists need instead is explorative access to the data to find interesting subsets that need further detailed and time-consuming analysis. Data synopses providing approximate (with a tight bound for the maximum error) but fast query results are a promising first step to allow for the scalable exploration of simulation results.

Creating the synopses after the simulation output has been written to disk, however, means revisiting and analysing all the data; a time-consuming process (multiple times depending on the synopsis). We consequently propose to develop a broadly applicable approach through generating code for creating the synopsis on-the-fly, according to the needs of the scientists and optimised for the architecture used. This would leverage and complement recent work [11, 12] on the development of high-level languages to generate low-level, hardware-optimised simulation code. We envision a system in which, given the simulation code and model, the user describes the architecture and the questions they are interested in and the system generates the code to produce this exact synopsis as the simulation runs.

To build such a system, we need to address several challenges. First, we need to define a language the user can use to express their interest. Second, depending on the user's interests, a particular type of synopsis must be chosen. For example, if a user needs to ask for simple range or count queries, a synopsis based on histograms may be sufficient. Third, the generation of the synopsis needs to be fast and thus optimised for the hardware architecture used.

This work was supported by the Natural Environment Research Council [NERC grant reference number NE/K008951/1].

Finally, to make the best use of the space needed to store the synopsis, we need synopses which are more accurate in areas that scientists are interested in and less so elsewhere.

Explorative queries on scientific data [9] (potentially relying on model-based compression [14]) have already been proposed in the past. Our vision applies similar ideas of approximate query answers (albeit not with iterative, i.e., with improving precision [9, 1]) to the simulation sciences to automatically produce data synopses. Work on reasoning about approximate and uncertain data [6] ideally complements our work in that it can be used to reason about approximate query answers.

2. BACKGROUND

The vision we propose touches on three different areas of research.

2.1 Simulation Sciences

Over the last half century, computer simulation of physical phenomena has joined theory and experiment to form a third, and indispensable, pillar of scientific research [17]. For the vast range of physical phenomena which are continuous, the simulations centre on the calculation of discrete approximate solutions to partial differential equations. The spatial extent of the simulation is typically modelled by a finite vector of solution values, connected by a grid or mesh structure. The scientist specifies an initial simulation state, and the simulation state at the next discrete time step is calculated by applying numerical operations.

Scientists constantly push the precision, size and duration of simulations to the limits of their computational infrastructure. The size of the simulation is typically limited by the CPU power, speed of data transfer and, size of main memory in which the current state is stored.

The massive simulation result that has to be analysed considerably challenges the state of the art in data analysis as most current approaches do not scale well. One of the biggest challenges in simulation science today is indeed the size of the simulation result. While ever bigger disks enable simulation results of increasing size to be stored, the lack of scalable analysis tools limits the simulation duration in many instances [13]: why simulate for longer if the results cannot be analysed?

2.2 Generating Simulation Code

Numerical methods and simulation hardware have become so sophisticated in recent years that it takes scientists enormous levels of effort and very specialised computational skills to take advantage of them [12]. They frequently spend months or years reinventing implementations for numerical methods that are not optimal for the underlying highly parallel hardware. To make matters worse, the resulting code is often hundreds of thousands of lines, making it very difficult and time-consuming to maintain in the face of ever more sophisticated algorithms designed for ever more complex computer hardware.

Recent advances, however, have introduced higher-level languages to formulate the simulation problem on a mathematical level with only a few hundreds of lines of code [11, 12]. The mathematical specification is used to automatically generate highly optimised code for the hardware it is run on. Using a high-level language combined with code generation creates a critical separation of concerns: the scientist specifies the numerical algorithm but not its implementation. It is the role of the code generation system to produce a high performance parallel implementation highly optimised for the underlying hardware platform.

The Unified Form Language (UFL), which is employed by both the FEniCS [11] and Firedrake [12] automated simulation packages, enables scientists to work at this very high level, specifying mathematical operations over data stored in a wide range of representations on a computational mesh.

2.3 Data Synopses

Research in data synopses has in the past primarily been driven by applications like data streams and cardinality estimation for query processing [5]. In their very nature, however, they present a great opportunity to accelerate approximate query execution in the context of big data.

The basic idea of data synopses is that the full dataset is summarised, typically by using compression [5]. The synopsis acts as a surrogate for the data and is queried instead of the complete dataset. Through compression or summarisation, the synopsis is usually considerably smaller than the full dataset itself and consequently queries are executed faster. The execution time of a query depends primarily on the size of the synopsis. The size of the synopsis in turn depends on dataset characteristics, for example how compressible the data is, and is further controlled by the level of the compression.

Because of its substantial compression ratios, lossy compression is often used leading to imprecise representations of the data. Data synopses based on lossy compression can thus only answer queries approximately. Clearly, there is a trade-off between the size of the data synopsis (and thus query execution time) and the quality of the approximation: the smaller the synopsis is, the less accurate the approximation is and thus the bigger the error. Regardless of the quality of approximation used, the key to data synopses is to provide a user with tight

error bounds expressing how accurate the result is.

Data synopses have in the past primarily been used for data streams and cardinality estimation (for query planning) [5]. With data growing beyond what can be today handled efficiently, however, data synopses are rediscovered and are considered an interesting approach: instead of analysing the potential petabytes in big data applications in a time-consuming process, substantially smaller synopses can be queried almost instantly.

One major obstacle hindering adoption of data synopses is that the process to build them is time-consuming. Only once the experiment data is written to disk, are the synopses computed. All data, terabytes or more, has to be (potentially repeatedly) again read from the disk, loaded into memory, processed and written to disk. Having researchers wait until the synopsis is built before they can analyse the data considerably limits productivity [13].

Considerable effort in the past has primarily developed four types of synopses. First, random sampling [15] is very well suited for aggregate queries and takes samples at random either out of the results of the running simulation as they are streamed from the supercomputer to disk or after the dataset is stored on disk after the simulation. The former, sampling as the simulation runs, is very efficient, but requires all potential queries to be known at runtime already. For the latter, in case the dataset is stored on disk, samples can be taken online, at query time from the full dataset, or offline. For massive data, however, taking the samples online from the full dataset, as the query is asked, is unlikely to be feasible due to the high cost of random access to disk. Instead, sampling for big data needs to take samples offline. A second well-researched type of synopses are histograms [8] which play a central role for cardinality estimation in databases. Histograms summarise the data into bins, each with its own value range, e.g., the bins store count of values/tuples in its range. Doing so makes them particularly useful for range-count queries, but they are also used for general analysis queries [5]. Synopses based on wavelets summarise and approximate the data through wavelets [3]. Essentially, wavelet transformation is applied to relations or to time series, resulting in a collection of wavelet coefficients. The size of the synopsis depends on how many coefficients are stored, which in turn defines the accuracy with which queries can be answered. The size of the synopsis, however, alone does not define the query execution time: at runtime, query execution can choose to ignore coefficients, thereby reducing query execution time, but also precision. Synopses based on sketches are a relatively new development [4]. As opposed to sampling, all data is considered for sketches, but only a small summary is retained (e.g., for a sum query all values are added up and only the sum is stored). A different sketch has to be defined for each query and this approach thus requires considerable effort.

3. GENERATING SYNOPSES

At the core of our approach is the idea of using data synopses for the efficient exploration of data produced as the result of the simulation. Producing the synopses after the data has been written on disk means revisiting all the data again to compute the and is therefore costly.

3.1 Core Idea

Instead, we propose to instrument the model code to generate the synopsis in a unified mesh and vector data model during the simulation. Given a space budget and, potentially, priorities specified by the scientists, the system will select an appropriate synopsis representation and resolution. This will enable scientists to efficiently analyse, query and explore the simulation result approximately (with a tight maximum error bound) as soon as the data is output: possibly even while the simulation still continues. Scientists will specify queries in a high level mathematical manner and the required query code will be generated automatically. Scientists can use the synopsis with or without the full dataset (and writing all data to disk may be avoided). In some cases the synopsis alone already allows for a detailed enough analysis whereas in other instances, the synopsis serves as a surrogate for finding interesting phenomena that are then analysed in the complete dataset.

As we will discuss later, the choice of the type of synopses heavily depends on the type hypothesis to be validated through the simulation. As a consequence, we further propose that scientists formulate queries (based on the simulation model) they wish to answer using the simulation results.

The queries can additionally help to improve the usefulness of the synopsis. Given a limited space budget, the scientist may also be able to indicate areas or time ranges of interest *a priori*. We thus propose to generate synopses with variable precision, i.e., having a higher precision in areas of interest and lower precision elsewhere. Higher precision translates to smaller error bounds of the approximate answers but also requires more space. Clearly, it is not wise to only focus on this range as other queries could no longer be answered, so variable precision introduces an interesting trade-off in the face of a limited space budget. The queries serve as a crucial hint to generate a synopsis with more details in areas of the data where scientists want to query.

To optimise the generation of the synopsis, we

also require information about the underlying hardware. Information about the hardware is used as input to the code generation, so that computing and storing the synopsis as well as the result can be optimised for the underlying hardware. Doing so can optimise use of limited computation power and bandwidth to storage devices, therefore ultimately reducing interference with the running simulation.

Ultimately our goal is to generate the code to efficiently compute and store a synopsis of the simulation results on disk, along with code for the simulation itself. The synopsis written is of variable precision and the code generation is based on a space budget, the simulation model, the user queries used as hints, as well as information about the underlying hardware. The resulting code efficiently generates a data synopsis that will enable scientists to quickly and scalably analyse simulation results, without having to tinker with code or deal with the peculiarities of data synopses and hardware.

While similar ideas could be used for observation sciences as well, doing so is difficult in practice. Although pushing filters deep into the data acquisition pipeline is already standard practice so that events of little interest are filtered close to the detector, the instruments in the observational sciences are custom built, making code generation challenging.

3.2 Application Example

To better understand the brain and develop new drugs for brain-related diseases, the scientists in the Human Brain Project [13] build small-scale bio-realistic spatial models of a rat brain to simulate them in-silico. The spatial models they design are based on millions of three-dimensional cylinders, where several thousand cylinders together reconstruct the spatial shape of one neuron.

In their simulations on a supercomputer (BlueGene/Q) they study the propagation of electrical impulses through the models. At every time step of the simulation, the voltage (or other parameters) at each cylinder of every neuron (out of the billions in the model) is measured. The simulation output is essentially the state over the course of the simulation, i.e., one time series per cylinder of the model.

The resulting datasets are oftentimes of unprecedented size but do not need to be examined in detail in a first instance. Instead, to understand the simulation result, multiple queries of an explorative nature are asked. An example of a frequent post-simulation analysis used to understand the simulation data is to determine in what areas of the model the measured parameter, e.g., voltage, exceeds a particular threshold. To find such events, the neuroscientists execute queries with a combination of spatial, temporal and voltage predicates. Analyses based on the simulation model in general follow this template, i.e., restricting the temporal, spatial as well as voltage values to particular ranges.

Data synopses generated on-the-fly are ideal for this application scenario. Data synopses can be generated instead of the simulation data and allow for a substantially faster approximate analysis and exploration. Alternatively, in case the synopsis is generated along with the full data, it can be used to find interesting areas while the detailed analysis can be performed on the identified subsets in the full data.

Researchers in general typically carry out experiments with assumptions or expectations and consequently are predominantly interested in particular ranges of the values (areas, temperature ranges, etc.). This information can be based on previous analyses or on assumptions by the researchers but is crucial, as the synopsis can be more precise in areas of interest and more approximate in others. This is no different in the Human Brain Project, where researchers may be interested in finding particular events (shapes of the voltage curve) in a voltage range between 10 and 23. Clearly, a higher precision of the synopsis in this range is helpful for them.

Writing such synopses to permanent storage is challenging as the underlying simulation hardware typically features novel and cutting edge hardware components. The BlueGene/Q used in the Human Brain Project, for example, features SSD storage that can be used to buffer the simulation output. Doing so is essential when generating synopses as the simulation results can be buffered and summarised/consolidated (using the CPUs of the SSD racks) on-the-fly as the simulation is running, before writing the synopsis to the disk.

4. RESEARCH CHALLENGES

Generating code for on the fly synopses entails interesting data management challenges around synopses themselves (choice of adequate type as well as synopses with variable precision), the language to express user queries used as hints and finally optimising their computation used for the hardware. We discuss each of the challenges in the following.

4.1 Data Synopses

The research challenges around synopses centre around deciding what type of synopsis to use, as well as how to configure it. More challenging is the how to support variable precision in data synopses.

4.1.1 Data Synopsis Type and Configuration

One research question that needs to be addressed is the choice of data synopses. Most types of synopsis developed in the past primarily focus on answering aggregate queries. Histograms [8] are ideal for aggregates while sampling [15], wavelets [3] or sketches [4] can be used for more general queries.

It is therefore key to understand, based on the sample queries the user provides, what questions need to be answered and what type of synopsis is suitable for these queries. If the sample queries hint at aggregate queries, a synopsis based on histograms can be used. Also based on the queries is the decision on how to configure the synopsis. In the case of histograms the question is what statistics are stored per bin (e.g., for maximum predicates, each bucket must store the maximum value). If sketches are used, the example queries indicate what information the sketches need to capture.

In the context of the Human Brain Project (HBP), the simulation output primarily comprises of time series measuring the voltage in different areas of the brain. The queries asked for the most part are threshold queries, e.g., in what time series does the voltage exceed a given threshold. For this type of queries wavelets may be most suitable.

4.1.2 *Synopsis with Variable Precision*

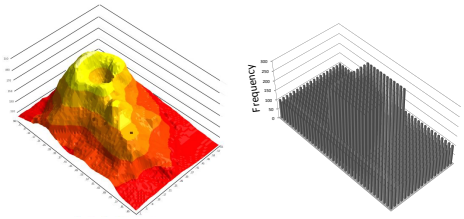
Key to the vision of generating synopses on-the-fly is to use variable precision or resolution. Using variable resolution allows us to reduce the space required by the synopsis, as areas which are not of much interest can be represented with less data.

The sample queries are used to infer what ranges (e.g., areas in a spatial model) the scientist is interested in exploring and analysing. The resolution is set higher for areas the scientist is interested in, and lower elsewhere. Improving the precision in areas of interest can be accomplished by storing more samples in the case of sampling, storing more coefficients in the case of wavelets or by using smaller bins in the case of histograms.

Figure 1 illustrates a synopsis with variable precision. The left hand side shows a spatial model (of a volcano with the temperature plotted) where the scientists are primarily interested in the center. As consequence, the center of the model is sampled with a higher frequency to produce a synopsis based on sampling as is shown on the right hand side (the y-axis shows sampling frequency while the other axes show geographic location).

In the sample application of the HBP, the neuroscientists primarily want to study voltage spikes and consequently they need higher resolution in ranges where spikes occur. Similarly, they are frequently interested in a particular spatial area. Only simulating the area of the model they are interested in is not feasible as the behaviour of the entire model must be simulated to obtain correct results. Specifying what areas to store with high precision is key to use space and bandwidth efficiently.

Using variable precision, however, introduces several challenges. First, given a space budget for the synopsis, the question is how much space should be used for ranges that the scientist is interested in, and how much should be used for the remaining areas. Using all space for the ranges of the areas of interest, will make analysis of other areas outright impossible and an adequate trade-off needs to be found. Second, query answers are straightforward to compute if the query only touches parts of the synopsis with the same precision. New approximate query answering approaches, however, need to be developed for the case where the precision is mixed. Third, crucial to this idea is that the increased precision is also quantifiable. That is, in areas with high precision, the error bounds are tighter compared to other areas. New methods have to be developed to quantify the error bounds in case ranges with varying precision are used.



(a) Spatial Model (b) Sampling Freq.

Figure 1: Sampling with higher frequency in areas of interest.

4.2 Model Definition & Query Language

The critical insight which makes the code generation approach feasible is that a wide range of simulation software employs variations of the same mesh and vector data model. Mathematical queries written in UFL can therefore be mapped to the data of models which are not themselves written in UFL. In addition to UFL, however, it will be necessary to define a suitable spatial query language. This will enable the scientist to specify the area or points over which the query should operate. This language is the layer which must be mapped to a database query to efficiently extract the relevant data.

4.3 Hardware Optimizations

Taking into account the available hardware can yield considerable improvements when computing and writing the simulation synopsis (as well as the simulation result) to permanent storage. Writing the synopsis efficiently in the face of limited disk bandwidth is crucial, as is exploiting special purpose hardware to offload computation of the simulation result so as not to interfere with the computation of the simulation result.

Supercomputers used for simulations frequently only have an aggregated disk bandwidth of a few tens of Gb/s for sequential I/O shared among sev-

eral thousand cores. A BlueGene/Q deployment, for example, has 16K cores sharing 40Gb/s of bandwidth, creating a serious bottleneck. Given the limited bandwidth, individual cores cannot simply write data to disk whenever they have new simulation results or new data synopses information. Instead, data has to be buffered at the cores and written to disk orchestrated, thereby wasting main memory that could be used to store bigger models.

Novel hardware solutions to alleviate this issue are at the ready. Active, flash-based storage [16] can be used to a) buffer bursts of simulation results, and b) use spare cycles of the active storage to compute the synopses. As a consequence, the simulation results are written on external flash drives where their synopses can be computed on-the-fly, thereby also offloading the computational overhead of computing the synopses.

Graphics Processing Units (GPUs) are becoming increasingly prevalent in the simulation sciences where they are used to offload parallel computations. The data transfer needed to move computation from the CPU to the GPU and simulation results back again, however, is a bottleneck because of the limited bandwidth of the PCI bus [7]. Data has to be accumulated so it can be transferred in a batch and incurs as little overhead as possible. Also in this scenario, the question is where to compute the synopsis and how to transfer it to the CPU where it can be stored permanently.

Crucially, the hardware has to be considered in order to optimise when and where to compute the synopses and when to write it to disk. Optimising use of the hardware, however, is beyond the abilities of a simulation scientist and is a challenge for which the code is ideally generated as well. The research challenge consequently is to determine how to describe the hardware infrastructure and how to generate code that uses efficient strategies to compute and store data synopses. The HBP, for example, has flash-based storage to buffer simulation results. Its efficient use, however, is beyond the abilities of simulation scientists and using code generation to develop code for producing synopses efficiently on the active storage is crucial.

5. CONCLUSIONS & FURTHER IDEAS

Simulations produce massive amounts of data that are difficult to analyse efficiently. What limits simulation size and length today is not only disk space, but our capacity to analyse the data within a reasonable time. Data synopses offer an interesting approach to the problem of the data deluge resulting from simulations. They are, however, challenging to compute and use, particularly for domain scientists with little training in software development.

With this vision, we propose an approach that avoids domain scientists having to develop highly specialised code to write data synopses. Instead, the code is automatically generated based on a space budget for the synopsis, the simulation model, example queries and a specification of the hardware used. The synopses enable scientists to scalably analyse massive data.

An interesting future avenue for this research is to generate optimised code for indexes as well. Clearly, each particular simulation will need different types of indexes but with sample analysis queries we can potentially infer what indexes are required.

A further interesting research question is if and how queries can be answered based on the static analysis or symbolic execution of the simulation code. Clearly, randomness is inherent in scientific simulations and the results will depend on the input parameters but assumptions about either can be made to potentially reason about query results without executing the simulation.

6. REFERENCES

[1] S. Agarwal, B. Mozafari, and et al. BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. EuroSys, 2013.

[2] J. L. Casti. *Would-be Worlds: How Simulation is Changing the Frontiers of Science*. Springer, 1996.

[3] K. Chakrabarti, M. N. Garofalakis, R. Rastogi, and K. Shim. Approximate Query Processing Using Wavelets. In *VLDB '06*.

[4] G. Cormode and M. Garofalakis. Sketching Streams Through the Net: Distributed Approximate Query Tracking. In *VLDB '05*.

[5] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches. *Found. Trends databases*, 4(1):1–294.

[6] B. Gonçalves and F. Porto. ϒ-DB: Managing Scientific Hypotheses as Uncertain Data. *VLDB*, 2014.

[7] C. Gregg and K. Hazelwood. Where is the Data? Why You Cannot Debate CPU vs. GPU Performance Without the Answer. ISPASS '11.

[8] Y. Ioannidis. The History of Histograms (Abridged). In *VLDB '05*.

[9] M. L. Kersten, S. Idreos, S. Manegold, and E. Liarou. The Researcher's Guide to the Data Deluge: Querying a Scientific Database in Just a Few Seconds. *VLDB*, 2011.

[10] B. Lawrence, V. Bennett, J. Churchill, M. Juckes, P. Kershaw, S. Pascoe, S. Pepler, M. Pritchard, and A. Stephens. Storing and manipulating environmental big data with JASMIN. In *Proceedings of the IEEE International Conference on Big Data,*, 2013.

[11] A. Logg, K.-A. Mardal, G. N. Wells, et al. *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012.

[12] G. R. Markall, D. A. Ham, and P. H. J. Kelly. Towards Generating Optimised Finite Element Solvers for GPUs from High-level Specifications. *Procedia Computer Science*, 1(1). ICCS 2010.

[13] H. Markram and et al. Introducing the Human Brain Project. volume 7, pages 39 – 42, 2011. Proceedings of the 2nd European Future Technologies Conference and Exhibition 2011.

[14] H. Mühleisen, M. L. Kersten, and S. Manegold. Capturing the Laws of (Data) Nature. *CIDR*, 2015.

[15] G. Piatetsky-Shapiro and C. Connell. Accurate Estimation of the Number of Tuples Satisfying a Condition. In *SIGMOD '84*.

[16] F. Schürmann and et al. Rebasing I/O for Scientific Computing: Leveraging Storage Class Memory in an IBM BlueGene/Q Supercomputer. In *Supercomputing*, Lecture Notes in Computer Science. 2014.

[17] H. Stephan. The World as a Process: Simulations in the Natural and Social Sciences, 2005.

Instant recovery for data center savings

Goetz Graefe
Hewlett Packard Laboratories
Palo Alto CA 94304
Goetz.Graefe@HP.com

ABSTRACT

Today's data centers routinely employ triple redundancy, i.e., each disk page of a database or of a key value store is stored three times (or even more, e.g., in database and file system backups). In contrast, write-ahead logging can reduce the cost of database operations and of data centers, assuming suitable techniques for logging, log archiving, backing up, and recovery. The present paper summarizes our work to-date on single-page repair, instant restart, and instant restore; it describes our techniques for self-repairing indexes, single-pass restore, and virtual backups; and outlines the opportunities for single-copy databases, for avoidance of ever taking a backup (full or incremental), yet for availability and reliability matching today's double- or triple-redundant data centers.

1. INTRODUCTION

Data centers are very expensive, from real estate to construction, compute and storage equipment, power and cooling, etc. Reducing data center costs for databases, key value stores, and file systems is our goal, principally by replacing double- or triple-redundancy storage through single copies plus write-ahead logging.

The purpose of double- and triple-redundancy is to enable and ensure continuous service, not only database service but also applications and web service. Any alternative design must provide comparable or better availability and reliability. This paper outlines a vision for single-copy databases with high availability and reliability yet lower costs.

The principal technique is efficient recovery based on traditional write-ahead logging plus a few techniques of moderate complexity. These include per-page chains of log records, partial sorting during log archiving, and indexes for database backups and for log archives. Note that the recovery log formats of some major products such as Oracle and SQL Server already include per-page chains of log records and that some products already compress and aggregate log records during archiving, and other products sort log records during restart and restore operations. The required indexes for backups and log archives are b-trees loaded from sorted streams.

In addition to requiring fewer nodes in a data center, the proposed techniques enable self-repairing indexes and virtual backups. Self-repairing indexes enable continuous comprehensive consistency checks and automatic recovery after failures and defects. Virtual backups produce media or files equal to traditional backups but do so without any involvement of the database server. Efficient virtual backups render traditional backups obsolete, i.e., future database require neither processing nor networking bandwidth for backup operations in addition to query and transaction processing. Efficient restore algorithms provide up-to-date replacement databases without the expense of incremental backups or of log replay.

2. FOUNDATIONS

Following [2], the discussions below assume a simple database system or key-value store implemented on a conventional computer system. The principal hardware assumptions include page-access persistent storage and a buffer pool in volatile memory. Moreover, there is no hardware or software replication or mirroring, except that storage of log records is very reliable. The required "stable storage" often relies on mirrored storage for the recovery log and, if one exists, for the log archive. The techniques in this paper do not address failures in the recovery log or in the log archive, instead assuming the fiction of "stable storage" like prior research and commercial work in database recovery. Similarly, the techniques in this paper do not address memory failures, i.e., correctness of volatile memory such as traditional DRAM.

The only form of redundant storage is write-ahead logging supporting transactions, commit log records, in-place updates of pages and records, "exactly once" log records including rollback (compensation) log records, and checkpoints. Transactions support all ACID properties: atomicity ("all or nothing"), consistency, isolation (concurrency control), and durability (persistence). Atomicity is guaranteed even in cases of transaction failures (rollback), system failures (e.g., operating system crash), media failures (e.g., head scratch), and single-page failures (e.g., local wear). Concurrency control may employ page-level locking as well as record-level locking, key-value locking, or key-range locking.

Figure 1 illustrates the main data structures participating in update processing, system restart after system failures, and restore operations after media failures. Transaction logic modifies images of database pages in the buffer pool and writes appropriate log records to the recovery log. Database backups provide long-term storage for database contents and the log archive provides long-term storage for log records. In case of a

system failure, restart ensures up-to-date buffer pool contents from the available database and the recovery log, along with other server state in the transaction manager and the lock manager. In case of a media failure, restore operations combine database backups and log archive into an up-to-date replacement database.

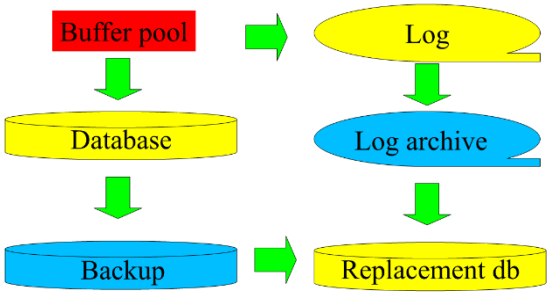


Figure 1. System model and update progagation.

2.1 ARIES

ARIES [6] recovery relies on write-ahead logging [1]. ARIES enables fine granularities of locking, e.g., row-level locking (ARIES/IM [7]) and key-value locking (ARIES/KVL [8]). Transaction rollback is logical, i.e., it performs the reversing update and writes a compensation log record.

Each log record describing a transaction update points to the most recent prior log record of the same transaction. Each rollback log record includes a field called "next undo lsn" (a log sequence number is the address of a log record within the recovery log), which guides rollback after an interruption, e.g., a system failure during transaction rollback. Each database page contains a PageLSN field that indicates the most recent log record reflected in the page image, enabling "exactly once" application of log records to database pages.

2.2 Log shipping

Traditional techniques for high availability require one or more secondary servers running on separate nodes in a cluster or in separate failure domains. Each secondary server holds a complete copy of the database and keeps it up-to-date at all times. Maintenance of such secondary database copies may rely on writing each dirty database page not only to the primary database copy but also to each secondary copy. Alternatively, the primary database server may send its log records to all secondary servers, which apply those log records to their local database copy in continuous log replay and "redo" recovery. Common names for these alternatives are database mirroring and log shipping.

When the primary database server or its network connection fails, a secondary server takes over. Typically, the new primary server rolls back all incomplete transactions.

3. ON-DEMAND PAGE RECOVERY

Traditional write-ahead logging and log-based recovery enable transaction-by-transaction "undo" – the first innovation towards instant recovery is page-by-page "redo," originally motivated by flash storage.

3.1 Single-page failures and repair

None of the traditional failure classes considered in transaction processing and database systems properly describe failures of individual pages on storage such as flash. Single-page failures are different from transaction failures, from media failures, and from system failures [3]. Among the three traditional failure classes, single-page failures are most similar to media failures. They differ from media failures, however, since only individual pages fail, not an entire device. Treating a few failed pages as a failure of the entire device seems very wasteful.

Single-page recovery uses a page image from a backup and the history of the page from the recovery log, specifically the "redo" portions of log records for the specific page. Efficient access to all relevant log records requires a pointer to the most recent log record and, within each log record, a pointer to the prior one.

The original proposal for single-page failures suggests a "page recovery index" for each database or each table space. With an index entry for each page in the database, the page recovery index points to the most recent log record for each database page not present in the buffer pool. Each time the buffer pool writes a dirty database page to storage, an entry in the page recovery index requires an update with a new LSN value.

For the per-page list of log records, each log record merely embeds the PageLSN value found before the log record's update. In other words, this value is easy to obtain during transaction processing and it can serve other purposes than single-page recovery. These include consistency checking during log-based replication and during log replay while restoring a failed storage device. The recovery logs of Oracle and Microsoft databases contain per-page chains of log records, but not for the purpose of single-page recovery.

Lookup in the page recovery index, in a backup file, and in the log record might require multiple I/O operations. While seemingly expensive, it is cheap and efficient when compared to data loss or device mirroring with doubled costs from initial hardware acquisition to each write operation.

3.2 Self-repairing indexes

Self-repairing indexes [5] combine efficient (yet comprehensive) detection of single-page failures with immediate single-page recovery. Comprehensive fault detection requires in-page checks as well as cross-page

checks. In a self-repairing b-tree index, each node includes low and high fence keys that define the node's permissible key range. Along the left and right edges of the b-tree, these fence keys have values $-\infty$ and $+\infty$, including in the root node. Otherwise, a node's fence keys equal two keys in the node's parent, i.e., typically branch keys. A node and its left-most child share the same low fence key value; a node and its right-most child share the high fence key value.

For both fault detection and repair, each parent-to-child pointer in a self-repairing b-tree carries an expected PageLSN value for the child page. For simplicity of maintenance, this requires that there be at all times only a single pointer to each page as in Foster b-trees [4].

3.3 Instant restart

Database system failures and the subsequent recovery disrupt many transactions and entire applications, usually for an extended duration. For those failures, new on-demand "instant" recovery techniques reduce application downtime from minutes or hours to seconds. These new recovery techniques work for all data stores that employ write-ahead logging, including databases, file systems, and key-value stores.

In traditional recovery from a system failure, e.g., a crash of the database server process, applications may resume and start new transactions after recovery has performed the "redo" actions of all pre-crash log records and then all "undo" (compensation) actions for failed transactions, i.e., those left incomplete at the time of the crash. Both "redo" and "undo" phases may require many random database reads and thus a relatively long time. The design and some implementations of ARIES support new transactions concurrent to the "undo" phase after lock acquisition during the "redo" phase.

For even better availability, instant restart permits new database transactions immediately after log analysis and thus before "redo" and "undo" work, imposes little load concurrent to new transactions, and prioritizes recovery of database contents to unblock new transactions. Lock re-acquisition for loser transactions must precede new transactions. Techniques new for log analysis include a backward log scan, transaction-by-transaction lock re-acquisition after log analysis, and database checkpoints immediately after log analysis and lock re-acquisition. After log analysis, new transactions guide on-demand recovery. An access to an "in doubt" database page triggers single-page repair and a lock conflict with a loser transaction triggers single-transaction rollback. Traditional "redo" and "undo" may run as background process. Techniques new for restart include page-by-page "redo" and transaction-by-transaction "undo," both single-threaded or in many parallel threads.

4. BACKUP AND RESTORE

An informal poll of database administrators identified backup operations as one of the most onerous aspects of owning a database. Thus, we set out to reduce or remove the need for taking database backups. The new techniques not only eliminate the need to take backups but also speed up restore operations, even permitting query and transaction processing against a replacement device practically instantly.

4.1 Single-pass restore

In this design [9], the log archive primarily serves log replay after media failures. Single-page recovery may use both the recovery log (for recent log records) and the log archive (for older log records). The discussion within this section focuses on the log archive and recovery from media failures.

The essence of restoring a database volume to an up-to-date state in a single pass is the order of database pages and of log records. For backup and replacement media, the ideal order proceeds by page identifier, from the lowest to the highest, i.e., a single sequential pass over backup and replacement media. The problem is that transactions write the recovery log in the order of time or of LSN values, not in the order of database pages.

A new technique enables single-pass continuous log archiving as well as immediate single-pass up-to-date restore operations: to sort log records during archiving, yet to divide the logic of external merge sort between log archiving and restore operation. The obvious way to divide this logic is to separate run generation and merging. In other words, log archiving not only suppresses some log records and compresses the remaining ones but also sorts log records into runs. Each run in the log archive captures a time interval in the original recovery log, i.e., a continuous range of LSN values. Within each run, log records are sorted by the page identifier within the database.

When compared to a fully sorted log archive, the crucial advantage of a partially sorted log archive is the efficiency in its creation. Creation of a partially sorted log archive is akin to run generation in an external merge sort. Thus, log archiving writes each log record to storage only once, the final log archive. Of course, due to in-memory sorting and run generation, archiving writes log records not immediately but after a short delay for in-memory processing.

When compared to a traditional, unsorted log archive, the crucial advantage of a partially sorted log archive is the efficiency in its use, i.e., during a restore operation. Replaying the log records in an unsorted log archive requires many random accesses in the replacement database. In contrast, a single merge step can merge many runs from a partially sorted log archive. The

merge logic may pipeline the log records into the restore logic without intermediate files. Thus, the merge logic applies the sorted log records directly to the stream of database pages flowing from the backup to the replacement device, not to the replacement device in a second step after an initial restore operation.

4.2 Virtual backups

A virtual backup operation produces an up-to-date backup without even accessing the database. In other words, it is not a backup at all in the sense that it does not read any information from the active database.

A virtual backup does not require any new techniques: take an old backup and replay the recovery log in order to produce not an up-to-date database image but an up-to-date backup image. In the past, however, this process would require completely sorting the recovery log covering days or weeks or it would require decompressing the old backup onto standard database storage followed by log replay with hours or days of random I/O operations.

With an older backup and a partially sorted log archive, a virtual backup can be simple and fast. The operation merely merges the older backup and the partitions of the log archive, very similarly to a single-pass restore operation. The difference, if any, is the precise format of the produced backup. For example, while a restore operation writes page images on page boundaries in the target space, a backup operation may compress page contents, suppress empty space within pages (often >25% in b-tree pages), etc.

4.3 Instant restore

Obviously, it is impossible to finish a restore operation much faster than single-pass restore does. Therefore, "instant restore" merely gives the illusion of a truly instant restore operation: it permits queries and updates practically immediately after a replacement device is available, i.e., formatted but empty. Nonetheless, in spite of concurrent transactions, the restore operation may complete in about the same time as an offline single-pass restore operation, i.e., much faster than a traditional restore operation with log replay using log records in their original order.

The principal technique enabling the appearance of instant restore is on-demand restore operations for the immediately needed pieces of the failed media. In other words, the failed media and its replacement are conceptually divided into segments (a page or a set of contiguous pages), and each time a transaction attempts to access one of those segments for the first time after the failure, that particular segment is recovered from the most recent backup and from the log archive.

In order to enable efficient on-demand single-segment media recovery, both the backups and the log archive require appropriate indexes. A full backup does not require an index if a database page identifier implies a byte offset in the backup file, i.e., if the backup logic fails to skip over unallocated pages, to eliminate free space within pages, and to forgo other opportunities for compression. In the indexes for the backups, each index entry maps a segment, e.g., a database page identifier, to a location in the backup file. A large device might be broken into 1 M segments and therefore the index requires up to 1 M entries per partition or run.

With those indexes, on-demand recovery restores one segment at a time, i.e., a pre-defined set of contiguous database pages, with the logic of single-pass restore within each segment. The indexes enable efficient access to the relevant database pages in the backup and log records in the partitions of the log archive.

Moreover, since data structures and algorithms are so similar, segment-at-a-time restore operations can run both lazily, i.e., on demand and guided by active transactions, and eagerly, i.e., sweeping through all segments in the manner of single-pass restore. Thus, a restore operation should complete in about the same time as with offline single-pass restore, i.e., much faster than traditional restore operations with log replay and many random I/O operations, and on-demand restore guided by active transactions should not extend media recovery time.

5. CLUSTERS AND FAILOVER

Instant failover assumes a cluster with multiple nodes and log shipping from a primary node to one or more secondary nodes. Using a buffer pool, the database, and a recovery log, a primary node executes queries and updates while each secondary node holds a backup and the log archive. Their principal communication is continuous log shipping from the primary to each secondary node. The principal design question is how quickly a secondary node can take over query and transaction processing after the primary node fails.

In the traditional approach to high availability and fast failover, the secondary node holds a complete copy of the database and always keeps it up-to-date by immediate "redo" of all log records received via log shipping. In the alternative design, instant failover, a secondary node does not require an up-to-date copy of the database. It merely requires empty space for the database, a full database backup (days, weeks, or months old), and a log archive covering the time since the full database backup. Both database backup and log archive must support fast access by page identifier, typically using indexes similar to those required for instant restore.

5.1 Instant failover

Failover requires recovery of server state and database contents. The server state includes transaction manager, lock manager, and buffer pool. Recovery of server state resembles system restart. Recovery of database contents resembles media restore operations. Thus, instant failover must combine log information and recovery actions from instant restart and instant restore.

The relevant server state for restart and failover includes the set of active transactions (except read-only transactions) and the set of locks held by active transactions (except read-only locks). Instant failover does not need in-doubt pages modified by recent and active transactions because all database pages require recovery. Instant failover recovers server state using techniques adapted from instant restart.

In instant restart, log analysis recovers the required server state such that transaction processing can resume immediately with concurrent "redo" recovery actions (in the form of single-page repair) and "undo" actions (in the form of single-transaction rollback). For instant failover, the new primary (formerly secondary) site may perform log analysis either continuously while receiving log records, i.e., before the failover, or as part of taking over primary responsibility for the database, i.e., during the failover. Mixed models are possible. For example, a secondary site may track the set of active transactions continuously during log shipping yet acquire locks only after a failover. With such a design, a secondary site avoids expensive lock management yet can acquire locks for active transactions quickly based on log records of active transactions, which are readily available using the per-transaction log chains.

Transaction processing after a failover assumes that the database on the new primary site is up-to-date and transaction-consistent. Instant failover starts with a full database backup and a log archive, with some of the log records still being sorted in order to form a partially sorted log archive indexed by a partitioned b-tree. Those log records may still linger in memory, i.e., the failover site has received but not added them yet to the persistent log archive. Instant failover recovers database contents using techniques adapted from instant restore.

The first task towards database recovery during instant failover sorts and indexes those log records. They may remain in memory and not immediately be added to the persistent log archive. The important aspect is that the restore logic can readily access all log records received from the failed former primary site. If additional sites require further log shipping, the new primary site sets up appropriate connections and log-shipping streams.

The second task provisions media for future database storage, recovery log, and log archiving.

The third task initiates single-pass restore in the background. It will run until all restore operations are complete. It will coordinate with on-demand incremental restore operations by avoiding concurrent recovery of the same database segment, by marking database segments it recovered, and by skipping over database segments already recovered by on-demand incremental restore operations.

The fourth task of instant failover resumes transaction processing. When a transaction invokes the buffer pool for a database page not yet recovered, the logic of instant restore recovers the appropriate segment on demand, exploiting appropriate indexes on the backup and on the partitions of the log archive. When a transaction requests a lock conflicting with one of the transactions active prior to the failover, that transaction must roll back. Transaction rollback may invoke restore operations for one or multiple segments by invoking the buffer pool for database pages not yet recovered.

The final task of instant failover rolls back pre-failover transactions not yet aborted on demand upon lock conflicts with new transactions. If desired, this task may start earlier, in which case it may well trigger restore operations for individual database segments. On the other hand, earlier execution of this task focuses early recovery efforts on the application's working set within the database. Of course, pre-failover transactions write rollback log records as during rollback without failover.

In summary, instant failover requires log shipping before the failure and techniques from instant restart and from instant restore after the failover. As in traditional log shipping, a transaction is durable even in the event of a complete node failure only after the secondary site has received all its log records including the commit log record. As in instant restart, log analysis recovers server state, in particular the transaction manager's set of active transactions and these transactions' non-read-only locks. As in instant restore, a partially sorted and indexed log archive permits log replay by merging backup and partitions of the log archive.

Log shipping for instant failover using these techniques requires much fewer resources on the secondary site than traditional log shipping. Instant failover merely requires a database backup, whereas traditional log shipping and failover requires an active database; and instant failover merely merges backup and log archive partitions in quasi-sequential I/O operations, whereas log replay in traditional log shipping and failover requires either many random I/O operations or an extremely large, dedicated buffer pool. Nonetheless, failover latency is quite similar in both techniques, gated by the delay in log shipping, and transaction processing performance after the failover suffers only slightly immediately after instant restart.

5.2 Failover pools

For the fastest possible failover, a secondary site may pre-start a database server process with no information yet in transaction manager, lock manager, and buffer pool. Such a server can take over for any database. A pre-started database server might serve other databases before and after it takes over for a failed database. Thus, instant failover may add information into the server state of an active server, which might require evicting some information from within this server, in particular database pages from the buffer pool.

In general, there might be multiple secondary sites. In case of a failure of the primary site, one of the secondary sites must take over. We ignore here the question on how to choose among multiple secondary sites and focus on techniques for instant failover on the chosen secondary site.

Conversely, a single site may partition its data such that failover spreads responsibility widely to multiple secondary sites. In this case, log shipping must split the log records to the correct secondary sites. The remainder discusses failing over an entire database but it might be only a partition within a database.

Finally, a site may serve as a secondary site for multiple databases or even multiple primary sites. This assumes, of course, that this site can access backups of all pertinent databases and receives a log-shipping stream from all relevant databases and sites. If site failures are rare and double failures are exceedingly rare, such a design promises to be viable.

Two possible deployment options suggest themselves. First, a large set of sites may serve as secondary sites for one another, in such a way that each site has multiple partitions and any failure spreads the load across multiple sites. Thus, even with a small number of failures, service continues with reasonable performance. Each partition requires multiple primary sites such that load balancing after is possible in cases of multiple site failures. This design also permits load balancing in the absence of failures, e.g., while one database or partition experiences high temporary workloads.

Second, a large set of primary sites serve the database workload with only a few secondary sites. All secondary sites have access to all backups and all log archives, e.g., in shared network-attached storage. If a primary site fails, any secondary site can take over and assume the role of the specific failed site. With this deployment design, a few extra servers may achieve high availability rather than 3× the number of servers as required in traditional designs for high availability.

6. CONCLUSIONS

In summary, two simple techniques – per-page chains of log records in the recovery log and partially sorting log records in the log archive – enable page-by-page "redo" and thus a wealth of new features and functions in database backup and recovery. Combining recovery of server state from instant restart with recovery of database contents from instant restore enables instant failover to a database node that holds a database backup and a log archive but no up-to-date database image.

A third simple technique – error detection by parity or cyclic redundancy check calculations in data pages and check values in inodes and indirection pages – transfers the capabilities of self-repairing indexes to file systems and their data pages, despite their lack of page headers as found in database pages. Thus, if a file system can be extended from "double-write" journaling to write-ahead logging with a durable log archive, all forms of instant recovery including instant failover and failover pools can apply to practically all storage. This can reduce redundancy in data centers from typically 3N to N+3, i.e., by almost a factor of 3.

REFERENCES

[1] Jim Gray: Notes on data base operating systems. Advanced course on operating systems. Springer LNCS #60, 1978: 393-481.

[2] Goetz Graefe, Wey Guy, Caetano Sauer: Instant recovery with write-ahead logging: page repair, system restart, media restore. Morgan & Claypool Synthesis Lectures on Data Management, 2014.

[3] Goetz Graefe, Harumi A. Kuno: Definition, detection, and recovery of single-page failures, a fourth class of database failures. PVLDB 5(7): 646-655 (2012).

[4] Goetz Graefe, Hideaki Kimura, Harumi A. Kuno: Foster b-trees. ACM TODS 37(3): 17 (2012).

[5] Goetz Graefe, Harumi A. Kuno, Bernhard Seeger: Self-diagnosing and self-healing indexes. DBTest 2012: 8.

[6] C. Mohan, Donald J. Haderle, Bruce G. Lindsay, Hamid Pirahesh, Peter M. Schwarz: ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. ACM TODS 17(1): 94-162 (1992).

[7] C. Mohan, Frank E. Levine: ARIES/IM: an efficient and high concurrency index management method using write-ahead logging. ACM SIGMOD 1992: 371-380.

[8] C. Mohan: ARIES/KVL: a key-value locking method for concurrency control of multi-action transactions operating on b-tree indexes. VLDB 1990: 392-405.

[9] Caetano Sauer, Goetz Graefe, Theo Härder: Single-pass restore after a media failure. BTW 2015: 217-236.

In-memory Databases – Challenges and Opportunities

From Software and Hardware Perspectives

Kian-Lee Tan, Qingchao Cai, Beng Chin Ooi*,
Weng-Fai Wong, Chang Yao, Hao Zhang

National University of Singapore
{tankl, caiqc, ooibc, wongwf, yaochang, zhangh}@comp.nus.edu.sg

ABSTRACT

The increase in the capacity of main memory coupled with the decrease in cost has fueled research in and development of in-memory databases. In recent years, the emergence of new hardware has further given rise to new challenges which have attracted a lot of attention from the research community. In particular, it is widely accepted that hardware solutions can provide promising alternatives for realizing the full potential of in-memory systems. Here, we argue that naive adoption of hardware solutions does not guarantee superior performance over software solutions, and identify problems in such hardware solutions that limit their performance. We also highlight the primary challenges faced by in-memory databases, and summarize their potential solutions, from both software and hardware perspectives.

1. INTRODUCTION

Despite the dominance of disk-based data processing systems for Big Data [17], in-memory computing is recently gaining traction rapidly. This is fueled by several contributing factors: the increased capacity of main memory, the low cost of DRAM, and more importantly, the orders of magnitude higher main memory bandwidth than the most advanced disk- or flash-based storage. While in-memory databases have been studied since the 80s, recent advances in hardware technology have re-generated interests in hosting the entirety of the database in memory in order to provide faster accesses and real-time analytics. A comprehensive survey on in-memory data management and processing can be found in [33].

However, in-memory databases not only embrace opportunities with the emergence of new technology, they also face challenges and problems that are non-trivial to be addressed. Simply replacing the storage layer of a traditional disk-based database with memory will not satisfy the real-time performance requirements because of the retention of the clumsy components from traditional database systems, such as the buffer manager, latching, locking and logging [8, 34]. Other sources of overhead from pointer-chasing, cache-un-friendly structures, transaction isolation and syscalls, further exacerbate the performance problems. In addition to the classical storage layer performance issues, in-memory data-bases are increasingly hitting the communication and concurrency bottlenecks [35, 29].

A significant amount of researches have been done to address these challenges, through the design of new algorithms/data structures on top of existing software stack, from the aspects of in-memory data placement [27], parallelism [7], efficient logging [21], concurrency control [29, 31], etc. Nevertheless, advances in hardware are fast changing the commodity processor scene. The availability of technologies such as NUMA architecture [20], SIMD instructions [30], RDMA networking, hardware transactional memory (HTM) [16], non-volatile memory (NVM), and on-chip GPUs [5], FPGAs and other hardware accelerators, can potentially provide better performance with low overhead [16, 11, 9, 19].

In this paper, we argue that these are by themselves not magic bullets. Each hardware solution has its own limitations and idiosyncrasies. Without new software optimization techniques, and extensive tuning, it will be hard to fully realize the potentials that the technology brings. We highlight some of these issues, and identify promising research directions that our community can contribute for both OLTP and OLAP systems.

The remainder of the paper is structured as follows. Section 2 discusses the challenges faced by in-memory databases, and Section 3 surveys existing approaches to address these challenges from both software and hardware perspectives. In Section 4, we point out the potential problems coming from hardware solutions. We propose some open research directions in Section 5, and conclude in Section 6.

*Corresponding author.

2. CHALLENGES FOR IN-MEMORY DATABASES

2.1 Parallelism

In general, there are three levels of parallelism, i.e., data-level parallelism (e.g., bit-parallel, SIMD), shared-memory scale-up parallelism (e.g., thread/ process) and shared-nothing scale-out parallelism (e.g., distributed computation). Ideally, we would like to achieve linear scalability as the computation resources increase. This, however, is non-trivial and requires considerable tuning and well-designed algorithms. The fact that all these three levels of parallelism have been deployed in a wide variety of combinations further compounds the problem.

Research challenge 1: How can OLTP and OLAP systems benefit from the wide variety of parallelism paradigms present in today's systems?

2.2 Concurrency Control

An efficient concurrency control protocol is necessary and important in order to ensure the atomicity and isolation properties, and not to offset the benefit derived from parallelism. With the increasing number of machines that can be deployed in a cluster and the increasing number of CPU cores in each machine, it is not uncommon that more threads/processes will run in parallel, which dramatically increase the complexity for concurrency control. Surprisingly, current concurrency control algorithms fail to scale beyond 1024 cores [32].

Research challenge 2: We need truly scalable concurrency control.

2.3 Communication

Network communication is incurred for a variety of critical operations: data replication for fault tolerance, information exchange for coordination, data transmission for data sharing or load balancing, and so on. The limited size of main memory of a single server, in contrast to the big volume of data, exacerbates the network communication requirement. However, the data access latency gap between main memory and network is huge, making communication efficiency important to the overall performance.

Research challenge 3: How can we bridge the communication gap that is growing both in magnitude and diversity?

2.4 Storage

Even though in-memory databases store all the data in the main memory, the data should also be persisted to non-volatile storage for durability and fault tolerance. In traditional disk-based databases, this is achieved by logging each data update to a disk-resident write-ahead log. Logging to disk, however, is prohibitively expensive in the context of in-memory databases due to the extremely slow disk access, in contrast to the fast memory access.

Research challenge 4: How can we scale in-memory databases to exploit expanding NVM capacities effectively?

3. SOFTWARE AND HARDWARE SOLUTIONS

We shall summarize the potential solutions to these challenges, from both software and hardware perspectives in this section.

3.1 Parallelism

On the software level, we can impose different computation models or techniques to realize parallelism of different granularities, namely, fine-grained, coarse-grained and embarrassing parallelism. Multi-threaded programs are a common case that utilizes the multi-core architecture to scale up [29], while distributed computing takes advantage of the resources from hundreds or thousands of servers to scale out the computing/storage capability.

NUMA architecture is proposed to solve the data starvation problem in modern CPUs, by eliminating the coordination among different processors when accessing its local memory, and thus fully exerting the accumulated power from multiple processors. SIMD provides an easier alternative to achieve data-level parallelism, for its capability to operate on multiple data objects in one instruction [30]. In addition, bit-parallel algorithms are proposed to fully unleash the intra-cycle parallelism of modern CPUs, by packing multiple data values into one register, which can be processed in parallel [7]. The introduction of on-chip hardware accelerators such as GPUs, FPGAs, and other heterogeneous cores presents another challenge, as their usage does not just require new code but in fact new algorithms.

3.2 Concurrency Control

Following the canonical categorization in [32], concurrency control protocol can either be lock-based or timestamp-based, from the software perspective. A significant amount of research is trying to improve its efficiency, such as Very Lightweight Locking (VLL) [25] for lock-based approaches, and some MVCC protocols [13, 23] for timestamp-based approaches. High overhead from 2PC for distributed transactions are alleviated by some 2PC-avoidance partition schemes [10] and speculative concurrency

strategies [24, 14].

The ideal case for concurrency control is that all transactions are executed in parallel without any concurrency control protocol overhead, which is usually very hard to achieve in practice. It has been shown that HTM-based timestamp-based concurrency control performs quite close to the ideal and hence simply cannot be ignored [16].

3.3 Communication

To improve the network performance, there are mainly two main approaches.

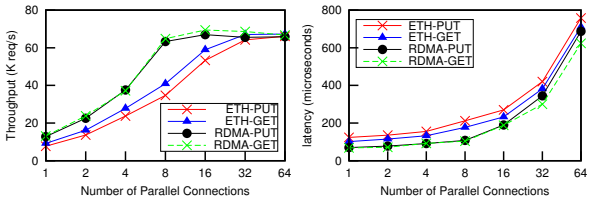
Minimizing communication. Data locality is key to minimizing communication overhead. With good data locality, there will be less frequent access to remote data. This can be achieved by a good partitioning algorithm [24], or an efficient query routing mechanism to push computation close to the data [14]. The data locality can only be considered from the software perspective, since the strategy is usually application-specific.

Improving the communication efficiency. From a software perspective, the network communication efficiency can be improved by object coalescing or batch communication [35, 2], kernel-bypass networking (e.g., netmap [26]), data-plane operating systems (e.g., IX [1]), and other techniques. On the other hand, RDMA boosts the networking performance from the hardware perspective, by enabling zero-copy and one-sided networking [22, 2]. In particular, RDMA enables user-level data transmission without involving the kernel or the CPU, thus achieving low latency and high throughput on the hardware level. It is free from the complexities and problems (e.g., lack of congestion control, retransmission) imposed by software solutions.

3.4 Storage

To alleviate durability overhead for in-memory databases, recent studies [21] proposed to use *command logging*, which logs only operations instead of the updated data, and combines with *group commit* to further reduce the number of loggings. However, there still exists a fundamental design trade-off between the high durability and logging overhead. If only a small number of transactions are committed each time, then the logging overhead may still substantially affect the transaction throughput due to its orders of magnitude higher latency than the execution time of transactions. On the other hand, accumulating a large number of transactions for *group commit* can lead to more data lost upon failure.

The emergence of NVM, including flash, PCM or STT-RAM, offers a new promising alternative for



(a) Throughput (b) Latency

Figure 1: Redis Performance with Ethernet and RDMA

data storage. These non-volatile technologies have been touted as DRAM replacements [6]. However, if we look deeper, things are not as rosy. The denser form of flash memory, i.e., NAND flash, works in units of pages. PCM and STT-RAM are bit addressable but PCM suffers from inherent endurance and write disturbance issues [15] while STT-RAM suffers from inherently non-zero write failure probabililty [18]. Furthermore, all three classes of NVM have asymmetric read and write latencies [18].

4. POTENTIAL PROBLEMS WITH HARDWARE SOLUTIONS

The challenges faced by in-memory databases can be addressed from both software and hardware perspectives. Software solutions are constrained by the underlying software stack, and increasingly hitting the performance bottleneck [3], while hardware solutions can boost the performance from the lowest underlying layers (i.e., transistors, circuits), which usually does not introduce much overhead.

However, naïve adoption of hardware solutions does not guarantee superior performance over software solutions. For example, simply using RDMA does not necessarily improve the performance greatly. In Figure 1, we show the performance difference for Redis [28], with its default Ethernet network (1 Gbps), and simple replacement of RDMA network (40 Gbps), as an illustration for the unexpected behavior by only relying on hardware. Specifically, we can see that RDMA does not improve its performance significantly in terms of both throughput and latency, even though RDMA throughput is 40× faster than Ethernet. With enough connections (i.e., more than 32), the performance is almost the same, losing the potential high performance advantage of RDMA networking. We summarize some issues that arise from hardware solutions below.

Mismatch with traditional software/OS stack. Newly-emerged hardware sometimes does not match with the traditional software/OS stack, which will cause unexpected behaviors or performance degradation. For example, although the byte

addressability and durable writes of NVM render it perfect for building an in-memory database, simply replacing DRAM or the disk with NVM to build an in-memory database fails to take advantage of these features. This is because of the mismatch between block/page granularity of OS paging and the byte granularity of NVM, and the fact that the system is oblivious to the limited lifetime of NVM. The disparity in the write patterns between two partitions as described in Section 3.4, and the workload with high write skewness, can result in some NVM cells being written much more times and consequently worn out much earlier than others. This poses a stringent requirement for the wear-levelling algorithm. In addition, traditional file system requires *read* or *write* syscalls to access the file, whose overhead can play a significant role in the total latency when accessing NVM.

Scalability. The scale of some new hardware cannot catch up with advances of other parts of the system, and some new hardware cannot easily scale up/out without significant performance degradation. For example, it is not uncommon that a multicore machine has more than 100 cores and hundreds or thousands gigabytes of memory, but HTM can only scale to a limited number of CPU cores [16], and Xeon Phi coprocessor can only support up to 16 GB memory. Doubts have also been raised on the scalability of some of the RDMA solutions [4]. RDMA memory consumption also poses a big challenge on large cluster due to the flexibility of communication model and user-dependent message passing implementation, losing the advantages that can be derived from mature network stack mechanism and memory management by OS.

Another scalability issue is that the current hardware support for virtual memory does not scale to terabytes (and beyond) of physical memory. The use of small page sizes and fine protection boundaries will require significant space overhead in the corresponding page tables. The naïve solution of enlarging page sizes is only a stop-gap measure as it will eventually lead to large page protection boundaries, fragmentation, and inefficient use of memory.

Generality/Compatibility. Hardware solutions are usually architecture specific, and not general enough to satisfy the different requirements from a variety of applications. For example, RDMA cannot easily communicate with the traditional Ethernet network directly. In addition, not all database transaction semantics can be expressed using HTM, since there are some other factors restricting its usage, e.g., limited HTM transaction size that is restricted by L1 cache, unexpected transaction abort due to cache pollution problems. Moreover, data alignment requirement for SIMD makes SIMD-based implementation architecture specific.

Extra overhead. In order to utilize some hardware, there are extra preparation work to do, which may offset the performance gain from the hardware. For example, in order to use SIMD instructions, we need to gather data into SIMD registers and scatter it to the correct destination locations, which is tricky and can be inefficient. RDMA only allows a pre-registered memory region to be the transmission source/destination, which also cannot be released/ modified until the transmission is completed.

Bottleneck shift. Even though the use of new hardware may tackle one bottleneck, the contribution to the system's overall performance may be restricted. In some cases, it only results in a bottleneck shift. For instance, the adoption of RDMA usually moves the bottleneck from network I/O to CPU, as with a fast networking, CPU is usually busy with data preparation and notification checking [11, 2]. And even though HTM can reduce the overhead caused by concurrency control to some extent, some components such as durability and communication still restrict the overall performance.

5. INTEGRATED SOFTWARE AND HARDWARE DESIGN

In this section, we discuss some open research directions in taking advantage of both software and hardware. We believe that hardware solutions, when combined with software solutions, would be able to fully exploit the potentials of in-memory databases.

Atomic primitives can be used for single object synchronization, and virtual snapshot by *fork*-ing facilitates a hardware-assisted isolation among processes [12]. HTM combined with other concurrency control mechanisms (e.g., timestamp-based) can be an alternative to the lock-based mechanism, but its special features (e.g., limited transaction size, unexpected aborts under certain conditions) should be taken into consideration. A mix of these protection mechanisms should enable a more efficient concurrency control model. Since the bottleneck for in-memory databases shifts from disk to memory. a good concurrency control protocol also needs to consider the underlying memory hierarchy, such as NUMA architecture and caches, whose performance highly depends on the data locality. The coordination overhead caused by 2PC protocol can be further alleviated, by eliminating distributed transactions via a dynamic data partition strategy, or designing a protocol based on distributed atomic operations provided by RDMA, for example. A client-

oriented transaction execution strategy, where the processing is performed at the client side simultaneously rather than in a centralized server, is also promising, which is made viable by the one-sided networking feature provided by RDMA.

In order to speed up the performance, various levels of parallelism should be exploited. Specifically, the data-level parallelism (e.g., bit-parallel, SIMD) can make extensive use of the "circuits" for parallelization. GPU, with a massively parallel architecture consisting of thousands of smaller cores, fits perfectly for embarrassingly parallel algorithms (e.g., filter, deep learning). Thus a software-coordinated CPU-GPU framework, which combines CPU's generality and GPU's specificity, can be utilized to distribute the tasks with different parallelism properties to different units in the warehouse or OLAP systems. The emergence of MIC co-processors (e.g., Intel Xeon Phi), provides a promising alternative for parallelizing computation, with many lower-frequency in-order cores and wider SIMD. Nevertheless, robust data structures that are parallelism-conscious, memory-economical, and access-efficient form the foundation for further parallelism exploration. For example, the skip-list, which allows fast point- and range-queries of an ordered sequence of elements with O($logn$) complexity, can potentially be an alternative to B-trees for in-memory databases, as it can be implemented latch-free easily and can be structured to be more parallelism-conscious (e.g., SIMD-friendly, NUMA-aware).

Distributed computing requires fast networking in order to achieve high scalability, where RDMA can play a big role. However, simply relying on RDMA networking is not guaranteed to improve the system performance, due to the restrictions of RDMA, bottleneck shift issues, etc. Combined with a good partition strategy (i.e., to achieve data locality), and efficient communication model (e.g., batch or coalescing transmission), the communication performance can be significantly enhanced. Besides, special features provided by RDMA should be taken into consideration, such as inline data, unsignalled, unreliable transport type, to fully exert its performance potential. And a heterogeneous software-coordinated communication model is also worth investigating, which can exploit the advantages from both the Ethernet and RDMA networks. Moreover, with the increasing throughput of RDMA network, the throughput of intra- and inter-server (i.e., memory bus among NUMA nodes and network in a cluster) is becoming similar. Hence, it is possible to develop a unified system framework that can be used in both a single server with multiple NUMA nodes and a cluster connected via high-speed networks.

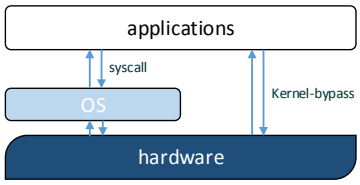


Figure 2: Interfaces with Hardware

For NVM-based in-memory databases, we believe a unified space management is required to effectively exploit its features (e.g., byte addressability and durable write). Although NVM is proposed to be placed side-by-side with DRAM on the memory bus, its distinct characteristics, such as limited endurance, write/read asymmetry, uncertainty of ordering and atomicity make it difficult to work effectively and efficiently. One way is to manage the NVM space as the main memory in a log-structured manner, such that the unnecessary reads/writes and the expensive syscalls, if used as a block device, will be eliminated, and the sequential write can be fully exploited. Due to the append-only feature of the log, the writes to NVM will be distributed uniformly among all cells, which in turn prolongs the lifetime of NVM. Besides, NVM enables more efficient fault tolerance strategies, if equipped with carefully designed algorithms to guarantee write atomicity and deterministic orderings.

As shown in Figure 2, the manipulation of hardware can be achieved either through syscalls or kernel-bypass methods. Some new hardware already provides direct kernel-bypass interfaces. But with kernel-bypass, the mature functionalities of the OS, such as memory management, concurrency control, buffer management are no longer available. It will also mean breaking the traditional boundaries of protection and separation of responsibilities. It is essential to retain key features of traditional OSes, even if direct access to the hardware is enabled. This can be achieved by moving the data path from the kernel space to the user space, resulting in a data-plane OS. Alternatively, new ABI boundaries will have to be drawn so that infrequent yet secure operations are handled over to the OS, while others are executed directly in the user-space.

6. CONCLUSIONS

In this paper, we summarized the challenges of in-memory databases, and their solutions from both the software and hardware perspectives. While hardware solutions are known for its efficiency with less overhead, as shown in this paper, they do not always outperform software solutions. In order to fully ex-

ploit the potentials of in-memory systems, we believe that a combined hardware-software solution is needed. There are a lot more that our community can bring to the table!

7. ACKNOWLEDGMENTS

This work was supported by the National Research Foundation, Prime Ministers Office, Singapore under Grant No. NRF-CRP8-2011-08.

8. REFERENCES

[1] A. Belay, G. Prekas, A. Klimovic, S. Grossman, C. Kozyrakis, and E. Bugnion. Ix: A protected dataplane operating system for high throughput and low latency. In *OSDI '14*, pages 49–65, 2014.

[2] Q. Cai, H. Zhang, G. Chen, B. C. Ooi, and K.-L. Tan. Memepic: Towards a database system architecture without system calls. Technical report, NUS, 2015.

[3] C. Cascaval, C. Blundell, M. Michael, H. W. Cain, P. Wu, S. Chiras, and S. Chatterjee. Software transactional memory: Why is it only a research toy? *Queue*, 6(5):40:46–40:58, Sept. 2008.

[4] Chelsio. Roce at a crossroads. Technical report, Chelsio Communications Inc., 2014.

[5] S. Damaraju, V. George, S. Jahagirdar, T. Khondker, R. Milstrey, S. Sarkar, S. Siers, I. Stolero, and A. Subbiah. A 22nm ia multi-cpu and gpu system-on-chip. In *ISSCC '12*, pages 56–57, 2012.

[6] J. DeBrabant, A. Joy, A. Pavlo, M. Stonebraker, S. Zdonik, and S. R. Dulloor. A prolegomenon on oltp database systems for non-volatile memory. In *ADMS '14*, pages 57–63, 2014.

[7] Z. Feng, E. Lo, B. Kao, and W. Xu. Byteslice: Pushing the envelop of main memory data processing with a new storage layout. In *SIGMOD '15*, 2015.

[8] S. Harizopoulos, D. J. Abadi, S. Madden, and M. Stonebraker. Oltp through the looking glass, and what we found there. In *SIGMOD '08*, pages 981–992, 2008.

[9] S. Jha, B. He, M. Lu, X. Cheng, and H. P. Huynh. Improving main memory hash joins on intel xeon phi processors: An experimental approach. In *PVLDB '15*, pages 642–653, 2015.

[10] E. P. C. Jones, D. J. Abadi, and S. Madden. Low overhead concurrency control for partitioned main memory databases. In *SIGMOD '10*, pages 603–614, 2010.

[11] A. Kalia, M. Kaminsky, and D. G. Andersen. Using rdma efficiently for key-value services. In *SIGCOMM '14*, pages 295–306, 2014.

[12] A. Kemper and T. Neumann. Hyper: A hybrid oltp&olap main memory database system based on virtual memory snapshots. In *ICDE '11*, pages 195–206, 2011.

[13] P.-A. Larson, S. Blanas, C. Diaconu, C. Freedman, J. M. Patel, and M. Zwilling. High-performance concurrency control mechanisms for main-memory databases. In *PVLDB '11*, pages 298–309, 2011.

[14] J. Lee, Y. S. Kwon, F. Farber, M. Muehle, C. Lee, C. Bensberg, J. Y. Lee, A. H. Lee, and W. Lehner. Sap hana distributed in-memory database system: Transaction, session, and metadata management. In *ICDE '13*, pages 1165–1173, 2013.

[15] S. Lee, M. Kim, G. Do, S. Kim, H. Lee, J. Sim, N. Park, S. Hong, Y. Jeon, K. Choi, et al. Programming disturbance and cell scaling in phase change memory: For up to 16nm based $4F^2$ cell. In *VLSIT '10*, pages 199–200, 2010.

[16] V. Leis, A. Kemper, and T. Neumann. Exploiting hardware transactional memory in main-memory databases. In *ICDE '14*, pages 580–591, 2014.

[17] F. Li, B. C. Ooi, M. T. Özsu, and S. Wu. Distributed data management using mapreduce. *ACM Computing Surveys*, 46(3):31:1–31:42, Jan. 2014.

[18] H. Li, X. Wang, Z.-L. Ong, W.-F. Wong, Y. Zhang, P. Wang, and Y. Chen. Performance, power, and reliability tradeoffs of STT-RAM cell subject to architecture-level requirement. *IEEE Transactions on Magnetics*, 47(10):2356–2359, Oct. 2011.

[19] D. Loghin, B. M. Tudor, H. Zhang, B. C. Ooi, and Y. M. Teo. A performance study of big data on small nodes. In *PVLDB '15*, 2015.

[20] L. M. Maas, T. Kissinger, D. Habich, and W. Lehner. Buzzard: A numa-aware in-memory indexing system. In *SIGMOD '13*, pages 1285–1286, 2013.

[21] N. Malviya, A. Weisberg, S. Madden, and M. Stonebraker. Rethinking main memory oltp recovery. In *ICDE '14*, pages 604–615, 2014.

[22] C. Mitchell, Y. Geng, and J. Li. Using one-sided rdma reads to build a fast, cpu-efficient key-value store. In *USENIX ATC '13*, pages 103–114, 2013.

[23] T. Neumann, T. Mühlbauer, and A. Kemper. Fast serializable multi-version concurrency control for main-memory database systems. In *SIGMOD '15*, 2015.

[24] A. Pavlo, C. Curino, and S. Zdonik. Skew-aware automatic database partitioning in shared-nothing, parallel oltp systems. In *SIGMOD '12*, pages 61–72, 2012.

[25] K. Ren, A. Thomson, and D. J. Abadi. Lightweight locking for main memory database systems. In *PVLDB '13*, pages 145–156, 2013.

[26] L. Rizzo. Netmap: A novel framework for fast packet i/o. In *USENIX ATC '12*, pages 101–112, 2012.

[27] S. M. Rumble, A. Kejriwal, and J. Ousterhout. Log-structured memory for dram-based storage. In *FAST '14*, pages 1–16, 2014.

[28] S. Sanfilippo and P. Noordhuis. Redis. http://redis.io, 2009.

[29] S. Tu, W. Zheng, E. Kohler, B. Liskov, and S. Madden. Speedy transactions in multicore in-memory databases. In *SOSP '13*, pages 18–32, 2013.

[30] T. Willhalm, N. Popovici, Y. Boshmaf, H. Plattner, A. Zeier, and J. Schaffner. Simd-scan: Ultra fast in-memory table scan using on-chip vector processing units. In *PVLDB '09*, pages 385–394, 2009.

[31] C. Yao, D. Agrawal, P. Chang, G. Chen, B. C. Ooi, W.-F. Wong, and M. Zhang. Dgcc: A new dependency graph based concurrency control protocol for multicore database systems. *ArXiv e-prints*, Mar. 2015.

[32] X. Yu, G. Bezerra, A. Pavlo, S. Devadas, and M. Stonebraker. Staring into the abyss: An evaluation of concurrency control with one thousand cores. In *PVLDB '15*, pages 209–220, 2014.

[33] H. Zhang, G. Chen, B. C. Ooi, K.-L. Tan, and M. Zhang. In-memory big data management and processing: A survey. *TKDE*, 27(7):1920–1947, July 2015.

[34] H. Zhang, G. Chen, W.-F. Wong, B. C. Ooi, S. Wu, and Y. Xia. Anti-caching-based elastic data management for big data. In *ICDE '15*, pages 592–603, 2014.

[35] H. Zhang, B. M. Tudor, G. Chen, and B. C. Ooi. Efficient in-memory data management: An analysis. In *PVLDB '14*, pages 833–836, 2014.

Forgetful Digital Memory: Towards Brain-Inspired Long-Term Data and Information Management

Claudia Niederee
L3S Research Center
University of Hannover
Hannover, Germany
niederee@L3S.de

Nattiya Kanhabua
L3S Research Center
University of Hannover
Hannover, Germany
kanhabua@L3S.de

Francesco Gallo
EURIX Research and Innovation
Turin, Italy
gallo@eurix.it

Robert H. Logie
Human Cognitive Neuroscience
University of Edinburgh
Edinburgh, UK
r.h.logie@ed.ac.uk

ABSTRACT

With the growing volumes of and reliance on digital content, there is a clear need for better information management approaches that keep *relevant* information accessible and usable in long-term encapsulated together with the evolving context information that is needed for its interpretation. Inspired by the role of forgetting in the human brain, in this paper, we envision a concept of *managed forgetting* for systematically dealing with information that progressively ceases in importance and also with redundant information - leading to a form of *Forgetful Digital Memory*. Managed forgetting is meant to complement rather than copy human remembering and forgetting. It can be regarded as functions of attention and significance dynamics relying on multi-faceted information assessment. Its goal is to introduce an alternative to the dominating "keep-it-all" strategy for digital content, which ensures that the important content is kept safe, useful, and understandable over time.

1. INTRODUCTION

In human memory, forgetting plays a crucial role for focusing on important things and neglecting irrelevant details. In digital memories, the idea of systematic forgetting has found little attention. At first glance, forgetting seems to contradict the purpose of digital memories. However, we are currently facing a tremendous growth in volumes of digital content in the public, organizational and personal context [4]. Thus, it becomes more important to focus on relevant and important content, while forgetting irrelevant details, redundancies and noise.

This also holds true for the survival of content; Let's consider the large quantities of photos that we typically bring back from a holiday trip as an example. Although safely stored (with a backup), they might become inaccessible over time due to digital format changes or failures of technology such as hard disk crashes. The automated selection of important content as it is supported by managed forgetting thus, does not only make an image collection more enjoyable. It also helps in deciding about where to invest in making the important content more future-proof, e.g., by relying on preservation management methods.

We envision the notion of *managed forgetting* that is aimed at improving the management of digital memories (organizational or personal collections). Managed forgetting is related to the idea of data rotting presented in *Big Data Space Fungus* [8]. However, we aim to learn from human forgetting and remembering processes. Thereby, it does not make sense to just simulate human memory, but to support human activities and human memory. This goal can be better achieved if the digital memory complements the human memory process.

Translating the high-level goal of "complementing" human memory into concrete methods and research questions is not trivial. On the one hand, there is a large number of complex processes and effects in human memory; thus implying different notions of complementing. On the other hand, we expect that to just do the opposite of what the human brain does (for example, just remember every-

thing digitally, when the human possibly wants to forget) would neither lead to a satisfying user experience nor fully exploit the potential of combining human and digital memory. For the above example of a holiday trip, it would be preferable, if the system could "forget" about the boarding pass after the flight in the same way as the human does. However, it might make sense to keep photos of all subevents of the trip for reminiscing, even if some of the subevents will be quickly forgotten by the human. It is noteworthy that the way a digital memory supports the human will - on the long run - also influence what the human keeps in memory.

An interdisciplinary model for flexible and gradual managed forgetting has to be developed that meets human expectations. As a core contribution, in this paper, we present such a brain-inspired conceptual model for a virtual digital memory that embraces human and digital memory and relies on managed forgetting. To this end, we present the Preserve-or-Forget (PoF) framework as a basis for the development of long-term information management systems based on our model.

2. RELATED WORK

In psychology, memory is the process in which information is encoded, stored, and retrieved. For complementing human memory, episodic and working memory are most relevant.

Episodic memory is the memory of autobiographical events (times, places, associated emotions, and other contextual knowledge). A characteristic of episodic memory is that details are lost very rapidly and that it is subject to interference [18]. Digital items such as photos and videos can play an important role in verifying (or falsifying [7]) the reconstructed memory of an event, thus having high potential for complementing human memory.

Human working memory [9] refers to the moment to moment activation and use of information in daily life. It activates a small amount of information just long enough to complete a task. This allows to focus on the current task while minimizing irrelevant information from the environment and from episodic and semantic memory.

There is general agreement among human memory researchers as to the basic principles of memory and forgetting, but there are several different conceptual and computational models of human memory and forgetting, e.g. [18]. On a global level, societies face difficult situations for societal memory, whether in the case of state archives detailing a dictatorial past, or sensational media reports that are subsequently shown to be false, and the unending digital memories created. This results in a growing understanding that *forgetting* also should be considered there, especially for information about individuals in the Web [10].

A search technology, such as Google, has shown effects on human memory [15]. Similarly, shared, retrieval-induced conversations in a social network can reshape the memories of people who are involved forming so-called collective memories [3]. In recent years, there have been several works addressing long-term information management, e.g., focus on the system design to support human memories [1], and personal information retrieval [6].

In the area of multifaceted information value assessment, several valuation methods have been proposed by employing a rich variety of criteria. Many approaches take observed usage in the past as the main indication for information value, i.e., probability of future use [11]. This type of information value is highly associated to short-term interests [17]. Existing works on time decay models can, for example, be found in the field of processing data streams [12] and time-aware information retrieval [13, 16].

3. FOUNDATIONS

In this section, we present a conceptual model of the forgetful approach to information management. Particularly, we identify five main characteristics for a forgetful digital memory: value-driven, forgetful, brain-inspired, evolution-aware, and integrative.

3.1 Value-driven: Acting upon Short-term and Long-term Information Value

One of our core ideas is to deviate from the dominating *keep-it-all* approach, which makes the implicit assumption that all information has the same value to be kept or preserved (invested into for long-term storage). In general the value of information is multifaceted and can be considered from different perspectives. An important distinction is between the short-term value for current activities and the long-term value of a resource.

The **short-term value** refers to the value of content for the current focus of activity, e.g., documents used for a task at hand are of high short-term value. Here, we will see a high dynamics in the information value (due to changing interests and tasks) and a high influence on interaction-based evidences on the information value. In terms of the human brain, this is roughly comparable to the working memory (see Section 3.3), although human working memory has a higher change frequency. Identifying the short-term value of a document is of high interest for creating immediate benefit in information management, e.g., by de-cluttering the desktop

or re-ranking search results by preferring more important content. Our anticipated methodology is to give high priority or visibility to resources with high short-term value. For this purpose, we coin the term *Memory Buoyancy*, which is inspired by the idea of resources decreasing in importance and sinking away from the user.

The **long-term value** refers to the value that a resource has on the long run and it can be used to decide about the investment of preserving the respective resource. Long-term value is expected to be more difficult to compute, since it includes estimating future use of resources. Furthermore, long-term value is driven by (partially) different factors than short-term value. It is expected that more objective aspects, such as diversity, coverage and quality will play a more important role. In this case, we have coined the term *Preservation Value* for the long-term value of a resource.

3.2 Forgetful: Focus on Important Things

We introduce the idea of a forgetful approach to long-term data and information management as an alternative to the dominating *keep-it-all* approach. Forgetting enables humans to focus on the relevant things and to efficiently make decisions in their current life situations. The forgetful approach opts for conscious decisions about what is important and thus should be kept (and preserved) replacing a form of random forgetting (or losing) information, e.g., disk crashes, or obsolescence of formats and technology. Since preservation comes at a cost, it is important to make such conscious decisions about (1) what to preserve, (2) how much to invest in the preservation, and (3) of which part of the information space. For this purpose, our forgetful approach is a good fit.

A forgetful approach is based on *Information Value* assessment, i.e. computing and predicting the value of information resources (see also Section 3.1). Effective information value assessment, especially for long-term information value, is a complex task involving a variety of parameters and heuristics. Based on such value, preservation decisions can be taken. On a high level, these decisions could include choice of preservation service as well as decisions about the level of redundancy and investment into preservation processes such as format transformations.

3.3 Brain-inspired: What to Learn from Human Forgetting and Remembering?

The idea of complementing human memory suggests to take an embracing perspective on human and digital memory, which considers them as a joint system. Thus, the interactions and mutual influence of the two parts are taken into account as well. Figure 1 (a) shows such a joint system perspective. The human memory (on the left) and the digital memory (on the right) together contribute to a form of virtual memory a human can rely on. The underlying model of the human memory is based on the work presented in [9].

Working Memory (Short-term). Together with the currently activated episodic and semantic memory, the *verbal short term memory* (things just heard) and the *visual short term memory* (things just seen) form the working memory, which frames the current situation. Knowledge is activated on demand from the semantic and episodic memory according to current needs via the so-called *executive functions*.

Perception is one driver for such activation. It is worth noting that perceived signals do not directly become part of the verbal or visual short term memory, which are constantly updated, but they are rather filtered and interpreted by things already in the memory for making sense of them.

Similarly, we also foresee a working memory within the digital memory This is composed of the digital resources currently relevant, e.g., used or relevant for current tasks or activities. The idea here is to keep those resources as close as possible to the user and easily accessible. In an automated digital working memory signals from resource usage, pattern of usage and change as well as relationships between resources will be used to update the digital working memory. The introduced managed forgetting function controls the transitions between the different parts of the digital memory.

Together, the working memory and the digital working memory form the virtual working memory. Clearly, there is an influence between both of them. In the ideal case, the digital working memory would show to the user just all the information that the user needs in the current situation, but does not have in her working memory. Note, that it is also possible that there is an influence of the way the digital memory works on the human (working) memory. For example, with the easy storage of phone numbers in mobile phones, humans no longer bother to remember phone numbers.

Mid-term and Long-term Memory. Managed forgetting functions are also used to identify content that is of long-term value (see Section 3.1) and should, therefore, be preserved. In Figure 1 (a), we distinguish between 1) information management for re-use, as it is, e.g., done on a desktop computer or a server, and 2) the system for archival

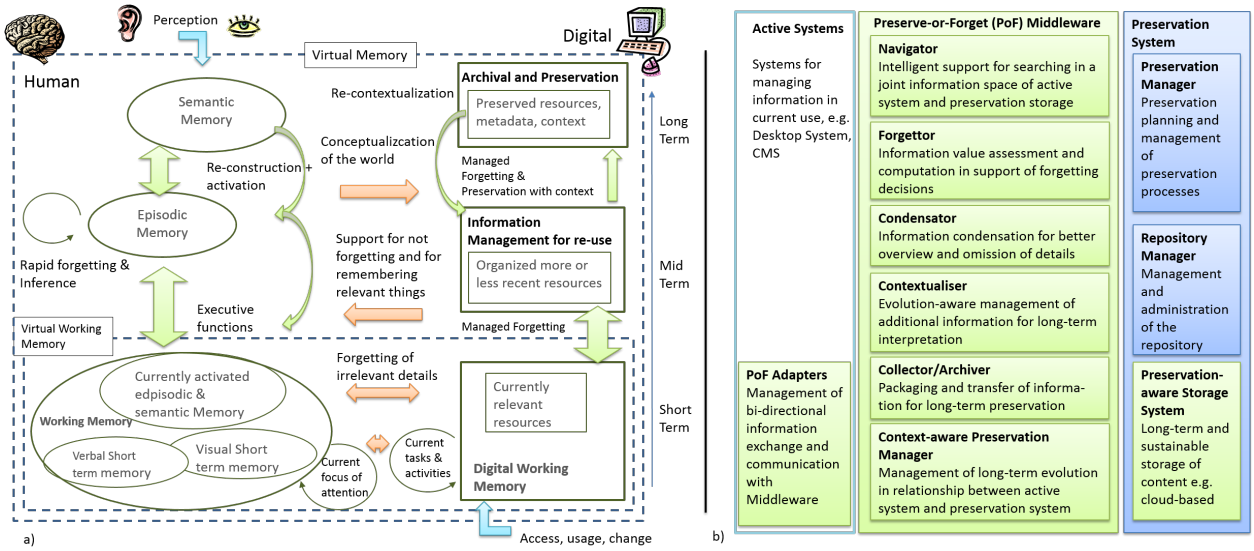


Figure 1: (a) Joint Perspective on Forgetful, Interacting Human and Digital Memory, and (b) Core Components of the Preserve-or-Forget (PoF) Framework.

and preservation. When content is transferred to *Archival and Preservation*, it makes sense to add context information to it (*contextualization*). This prepares the content for re-contextualization, which is required, when preserved content is brought back at a (much) later time. Re-contextualization is aimed to connect the re-activated content with the current environment, or at least, to make it understandable. The idea of re-contextualization as an active situation-dependent process is again inspired from human memory: when we as humans remember things this is also a re-construction process, which depends upon the current situation.

Episodic memory is mainly a detailed storage of events. It is typically subject to fast forgetting as well as blurring between the memory of similar events due to interference. Here, complementing human memory, e.g. via photos, can serve as a reminder of things that are forgotten, but that one might want to remember or refresh in a later point in time, e.g., reminiscing about past events.

Semantic memory is a more conceptual storage of memory, which stresses on patterns, abstractions and lessons learned. Here, the strongest interaction between human memory and digital memory is that the organization of digital resources does or should reflect the conceptualization of the world of the user, which is linked to the user's semantic memory. A more explicit modeling of the conceptualization of the world and a richer annotation of resource with this knowledge in *Information Management for Re-use* and in *Archival and Preservation*, such as, the Semantic Desktop approach proposed in [14], can ease navigation and search of the user and thus re-finding things in the digital memory.

3.4 Evolution-aware: The Long-term Perspective

Since we are targeting long-term integrated management systems, we are operating in a long-lived context, covering a time perspective up to several decades. Even things that are considered relatively stable in the current setting of an information system will change over time. For being prepared for sustainable operation it is important to be prepared for such changes. For this purpose, several types of evolution with different impacts have to be considered. This refers to the active system (which is the system used for active information management such as a desktop system or a CMS) as well as to the Preservation System, which is responsible for the long-term storage of the information.

Changes in conceptual model of the Active System can be due, for example, to changes in the organizational ontology underlying the content structuring as well as processes described in the content. This creates a semantic gap between the archived content (relying on the old implicit or explicit ontology) and the active content (structure by new ontology). This gap has to bridged, at latest when preserved content is brought back into the active environment, in order to enable correct interpretation of the re-activated content.

Furthermore, **evolution in the technology** of

the active system and of the Preservation System as well as **exchange of those systems** have to be considered if we look at time frames of several decades. In spite of such changes the content should stay accessible and usable. In the case of exchange of the Preservation System, for example, This implies the migration of content into a new Preservation System. In the ideal case this should have as little impact on the Active System as possible.

Finally, for **change in best-practices and technology** such as technology or formats, which become obsolete, it is mainly possible to rely on preservation system functionality, which focus on such kinds of issues.

3.5 Integrative: Bridging the Gap

In the long run, it has to be foreseen that the Active System used as well as the employed Preservation System will change. Therefore, the idea is a flexible integration, which is prepared for major changes in the overall environment.

A core part of integration is to enable the smooth transition of content to be preserved into the Preservation System and the senseful reactivation of content back into the Active System after a -possibly very long - period. An integrative solution should also embrace the idea of a *joint information space*, where the information in the Preservation System stays conceptually accessible, e.g., visible in search results, even if the content is only available in the Preservation System.

4. COMPUTATIONAL FRAMEWORK

In the European project ForgetIT, we have developed the prototypical Preserve-or-Forget (PoF) Framework to validate the core ideas of a forgetful Digital Memory. Figure 1 (b) summarizes the core parts of the Framework, which consists of three layers: Active Systems, PoF Middleware and Preservation System. An Active System corresponds to the information management system (IMS) to manage the information actively used. The PoF Middleware implements the core concepts of the conceptual model. The Preservation System (archive) is responsible for the long-term storage.

In the following, we focus mainly on the role of the Forgettor, which includes the core functionality for managed forgetting. Concerning the other components in Figure 1 (b), it is worth mentioning that they mainly implement different functionalities associated to forgetful long-term information management, including bi-directional information exchange between the Active System and the archive, information condensation (Condensator), contextualization (Contextualizer), synergetic preservation (Context-aware Preservation Manager), and advanced forgetful search (Navigator). Many components of the PoF architecture have been omitted on purpose here, such as those responsible for process scheduling, work-flow management, data management and communication, because they are more related to the implementation.

The Preservation System leverages cloud technologies, with an innovative approach based on Storlets, which are components capable of running preservation processes in the storage.

For supporting managed forgetting, the Active System has to collect and exchange data with the Forgettor component. This includes content metadata (e.g. size, authors), but also context and usage information (timestamps, usage activities such as usage frequency and recency of use, user ratings, recurrent usage pattern, social context data). The Forgettor interacts with the Active System based on different strategies, depending on the application case. It can, for example, be triggered periodically or - for organizations - at major project milestones.

Inside the Forgettor, the *Assessor* is responsible for evaluating resources with respect to their current importance, as well as their intrinsic preservation worthiness for the future reflected by the quantitative measures for information value assessment, **Memory Buoyancy** (MB) and **Preservation Value** (PV), respectively. MB is influenced by a variety of factors that can be roughly grouped in the following categories: usage information, type and provenance information, relationship with other resources, and temporal parameters such as age and lifetime specifications. The PV reflects the expected value of a resource for the future and will be used to decide if and when to move a resource to the Preservation System. Partly, PV is influenced by similar values as MB, but it serves a different purpose: a resource with a high MB value might already be moved to the archive (as a copy) because of its high PV value on the contrary, a resource with both low MB and PV values might be preserved only in its condensed version or it might be decided not to preserve it at all. To assess the MB and PV properly, it is often necessary to have knowledge of previously computed MB and PV values (e.g., statistics for the method development). Therefore the Assessor component has access to a historic value repository.

5. ESTIMATING PRESERVATION VALUE

Information value assessment resulting in MB and PV values is in the core of our approach. For MB, the model for MB assessment basically embraces

two aspects, namely, estimating the importance of digital objects and their retention in human brains by analyzing the observed activity logs (time-decay model) [12, 13], and associating them with their related objects, or with context information in the information space. In addition, we propagate the estimated MB scores along different connections to other digital objects (propagation model), or some of which might be unobserved by the activity logs.

Computation of PV is a much more demanding task, since it has to deal with usefulness expected in the future. Research is still required for understanding how the long-term value of information objects can be best estimated (using various kinds of evidences and features). However, first exploratory research in preservation value and its driving factors has already provided some insights to build further work upon. In experiments on expectation oriented photo selection for preservation with more than 18,000 photos [2], it has been shown that (automatically extracted) depicted concepts and the presence of persons are key features, while image quality is only of secondary importance. In addition, in contrast to other work in photo selection, it has been shown that the aspect of coverage (of all sub-events) is less dominating for the task of photo selection for preservation. In a second experiment [5], the retention preferences for Social Web content for the example of Facebook have been investigated. Here the level of social interaction (e.g., likes, comments), and the type of the content (e.g., images vs. status information) have a strong influence on retention decisions. The experiments also confirmed a general decay in the interest for older posts.

6. CONCLUSIONS

In this paper, we presented our vision of introducing *forgetting* into Digital Memory by following a brain-inspired approach. We proposed model and framework, which are promising steps for creating a new form of focused, long-term data and information management. However, a lot of research and experimentation are still required to ensure that such systems really complement human memory and also meet human expectations.

Acknowledgments This work was partially funded by the European Commission Seventh Framework Program (FP7 / 2013-2016) under grant agreement No.600826 for the ForgetIT project.

7. REFERENCES

[1] S. Bowen and D. Petrelli. Remembering today tomorrow: Exploring the human-centred design of digital mementos. *International Journal of Human-Computer Studies*, 69(5):324–337, May 2011.

[2] A. Ceroni, V. Solachidis, C. Niederée, O. Papadopoulou, N. Kanhabua, and V. Mezaris. To keep or not to keep: An expectation–oriented photo selection method for personal photo collections. In *Proceedings of International Conference on Multimedia Retrieval*, ICMR '15, 2015.

[3] A. Coman and W. Hirst. Cognition through a social network: the propagation of induced forgetting and practice effects. *J Exp Psychol Gen*, 141(2):321–36, 2012.

[4] E. Cutrell, S. T. Dumais, and J. Teevan. Searching to eliminate personal information management. *Commun. ACM*, 49(1):58–64, 2006.

[5] K. Djafari Naini, R. Kawase, N. Kanhabua, and C. Niederee. Characterizing high-impact features for content retention in social web applications. In *Proceedings of International Conference on World Wide Web*, WWW Companion '14, 2014.

[6] S. Dumais, E. Cutrell, J. Cadiz, G. Jancke, R. Sarin, and D. C. Robbins. Stuff i've seen: a system for personal information retrieval and re-use. In *Proceedings of ACM SIGIR Conference on Research and Development in Informaion Retrieval*, SIGIR '03, 2003.

[7] D. L. Greenberg. President bush's false [flashbulb] memory of 9/11/01. *Applied Cognitive Psychology*, 18(3):363–370, 2004.

[8] M. Kersten. Big Data Space Fungus. In *Proceedings of the Conference on Innovative Data Systems Research*, CIDR '15, 2015.

[9] R. H. Logie. The functional organization and capacity limits of working memory. *Current Directions in Psychological Science*, 20(4):240–245, 2011.

[10] V. Mayer-Schönberger. *Delete - The Virtue of Forgetting in the Digital Age*. Morgan Kaufmann Publishers, 2009.

[11] S. Mitra, M. Winslett, and W. W. Hsu. Query-based partitioning of documents and indexes for information lifecycle management. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, 2008.

[12] T. Palpanas, M. Vlachos, E. Keogh, D. Gunopulos, and W. Truppel. Online amnesic approximation of streaming time series. In *Proceedings of International Conference on Data Engineering*, ICDE '04, 2004.

[13] M.-H. Peetz and M. de Rijke. Cognitive temporal document priors. In *Proceedings of the 35th European conference on Advances in Information Retrieval*, ECIR'13, 2013.

[14] L. Sauermann, A. Dengel, L. V. Elst, A. Lauer, and M. S. Schwarz. Personalization in the epos project. In *Proceedings of the International Workshop on Semantic Web Personalization at the ESWC 2006 Conference*, pages 42–52, 2006.

[15] B. Sparrow, J. Liu, and D. M. Wegner. Google effects on memory: Cognitive consequences of having information at our fingertips. *Science*, 333:776–778, 2011.

[16] N. K. Tran, A. Ceroni, N. Kanhabua, and C. Niederée. Back to the past: Supporting interpretations of forgotten stories by time-aware re-contextualization. In *Proceedings of International Conference on Web Search and Data Mining*, WSDM '15, 2015.

[17] R. W. White, P. N. Bennett, and S. T. Dumais. Predicting short-term interests using activity-based search context. In *Proceedings of International Conference on Information and Knowledge Management*, CIKM '10, 2010.

[18] J. T. Wixted. The psychology and neuroscience of forgetting. *Annu. Rev. Psychol.*, 55:235–269, 2004.

Data Series Management: The Road to Big Sequence Analytics

Themis Palpanas
Paris Descartes University
themis@mi.parisdescartes.fr

ABSTRACT

Massive data series collections are becoming a reality for virtually every scientific and social domain, and have to be processed and analyzed, in order to extract useful knowledge. Current data series management solutions are ad hoc, requiring huge investments in time and effort, and duplication of effort across different teams. Systems like relational databases, Column Stores, and Array Databases are not a suitable solution either, since none of these systems offers native support for data series. Our vision is to design and develop a general-purpose Data Series Management System, able to cope with big data series, that is, very large and fast-changing collections of data series, which can be heterogeneous (i.e., originate from disparate domains and thus exhibit very different characteristics), and which can have uncertainty in their values (e.g., due to inherent errors in the measurements). Just like databases abstracted the relational data management problem and offered a black box solution that is now omnipresent, the proposed system will allow analysts that are not experts in data series management, as well as common users, to tap in the goldmine of the massive and ever-growing data series collections they (already) have.

1. INTRODUCTION

[**Motivation.**] Data series have gathered the attention of the data management community for almost two decades [7, 17, 25]. Data series are one of the most common types of data, and are present in virtually every scientific and social domain: they appear as audio sequences [13], shape and image data [29], financial [24], environmental monitoring [21] and scientific data [11], and they have many diverse applications, such as in health care, astronomy, biology, economics, and others.

Recent advances in sensing, networking, data processing and storage technologies have significantly eased the process of generating and collecting tremendous amounts of data series at extremely high rates and volumes. It is not unusual for applications to involve numbers of sequences in the order of hundreds of millions to billions [1, 3].

[**Data Series.**] A *data series*, or *data sequence*, is an ordered sequence of data points[1] (if the dimension that imposes the ordering of the sequence is time then we talk about time series). Formally, a data series $T = (p_1, ...p_n)$ is defined as a sequence of points $p_i = (v_i, t_i)$, where each point is associated with a value v_i and a time t_i in which this recording was made, and n is the size (or length) of the series.

A key observation is that analysts need to process and analyze a sequence (or subsequence) of values as a single object, rather than the individual points independently, which is what makes the management and analysis of data sequences a hard problem. Note that even though a sequence can be regarded as a point in n-dimensional space, traditional multi-dimensional approaches fail in this case, mainly due to the combination of the following two reasons: (a) the dimensionality is typically very high, i.e., in the order of several hundreds to several thousands, and (b) dimensions are strictly ordered (imposed by the sequence itself) and neighboring values are correlated. Therefore, specialized techniques need to be used, as we discuss in Section 2.

[**Data Series Management Systems.**] There are important reasons why data Series (or Sequence) Management Systems (SMSs) are on the cusp of becoming a focal point for research activity in data management. The solutions that are currently available require custom code and the development of ad hoc systems for various tasks, requiring huge investments in time and effort, and duplication of effort across different teams.

Consider for instance, that for several of their analysis tasks, neuroscientists are currently reducing each of their 3,000 point long sequences to a single number (the global average) in order to be

[1]For the rest of this paper, we are going to use the terms *data series* and *sequence* interchangeably.

able to analyze their huge datasets [1]. Similarly, the increasing number of adults that monitor their health (e.g., a heart rate monitor is producing 9GB of sequence data per month), cannot use the data they produce, which could otherwise enable them to monitor their lifestyles, and (with the assistance of their doctor) identify changes in their health, or the consequences of different dietary or medicinal choices. In astronomy, there are currently available more than 70TB of spectroscopic sequence data from 200 million sky objects, collected by the Sloan Digital Sky Survey [3], allowing scientists to study the universe. These data have to be processed and analyzed, in order to identify patterns, gain insights, detect abnormalities, and extract useful knowledge.

In all the above cases, existing approaches (e.g., based on DBMSs [6], Column Stores [26], or Array Databases [27]) do not provide a viable solution, since they have not been designed for managing and processing sequence data. Therefore, they do not offer a suitable declarative query language, storage model, auxiliary data structures (such as indexes), and optimization mechanism that can support a variety of sequence query workloads (see also Section 3.1) in an efficient manner.

We argue that a SMS is necessary in order to enable big sequence analytics, since it will offer the abstractions, tools, and automations needed for achieving this goal. Just like databases abstracted the relational data management problem and offered a black box solution that is now omnipresent, the proposed system will make it feasible for analysts that are not experts in data series management, as well as common users, to tap in the goldmine of the massive and ever-growing data series collections they (already) have.

[**Challenges.**] Several works have studied the problems of sequence management, processing and analysis, focusing on some specific problems in these areas. The results of such studies offer an excellent starting point for our project, but do not constitute a viable solution to our problem.

There is a rich literature on sequence summarization techniques, but the corresponding indexing techniques are rather rigid (i.e., can only answer fixed-length queries), do not deliver in terms of performance what is required for the applications we envision, and most importantly, do not allow for cost-based optimization. We also note the need for a completely new generation of sequence query answering techniques that can efficiently operate on massive data sequence collections without the extremely expensive preprocessing step that current indexes impose. When considering uncertain sequences, we observe that current representation techniques are simplistic (with corresponding side-effects in their effectiveness), and new models and relevant indexing methods are in need.

Designing the right abstractions and data models for a SMS is also challenging. Data sequences could eventually be accommodated in existing systems (such as relational databases, Column Stores, and Array Databases), albeit in an unnatural way, since none of these systems treats sequences as first class citizens. These systems do not offer native support for sequences (e.g., data model, index structures, etc.), thus, any solution built on top of them will suffer in terms of expressive power, usability, and performance.

[**Contributions.**] Our ambitious goal is to design and develop a general-purpose data Series Management System. The contributions we envision can be summarized as follows.

- The system will be able to cope with big data sequences, that is, massive collections of sequences, which can be heterogeneous (i.e., originate from disparate domains and thus exhibit very different characteristics), and which can have uncertainty in their values (e.g., due to inherent errors in the measurements).
- It will efficiently support a wide range of sequence queries and mining operations, by developing ground-breaking techniques for scalable sequence management and analysis, while exploiting the benefits of physical and logical independence.
- A key feature of our solution will be cost-based optimization, which will enable the system to automatically pick the right storage and execution strategies, leading to remarkable improvements in time performance and scalability that would otherwise be untenable.
- In addition, we propose to make advances by leveraging ideas from adaptive systems and distributed computing, two areas that have not been studied in the past in the particular context of data sequences.

[**Organization of this paper.**] We review the related work and existing techniques and approaches in Section 2. In Section 3, we describe the main components of the proposed system, and outline the open research directions. Finally, we conclude in Section 4.

2. STATE OF THE ART

The main objective of the proposed research is to build a general purpose SMS for big sequences. Such a system should include techniques to support

sequences with very different characteristics (originating from disparate domains), as well as uncertain sequences, answer sequence queries fast even for massive sequence collections, and rely on cost-based optimization for query execution.

[Data Series Summarizations.] In order to efficiently process and analyze large volumes of data sequences, we have to operate on summaries (or approximations) of these sequences. Several techniques have been proposed in the literature for the summarization of sequences (e.g., DFT, DCT, DWT, SVD, APCA, PAA, SAX), and it is interesting to note that averaged over many datasets, there is little difference between all the above approaches in terms of accuracy of representation [18] (even though it *is* the case that certain representations favor particular data types, e.g., DFT for star-light-curves, APCA for bursty data, etc.).

[Data Series Indexing.] These summarizations are the basis for the index structures that make possible the efficient execution of sequence queries over very large collections of data sequences. In general, data series indexes operate by pruning the search space based on the summarizations of the data series and corresponding lower bounds, and only use the raw data of the data series in order to filter out the false positives.

Agrawal et al. [4] presented the first data series index, based on the DFT summarization. Various indexes, specific to data series, have also been proposed in the literature [25, 28]. A general observation when using sequence indexes though, is that query answering can degenerate to perform worse that a serial scan. To make things worse, these situations cannot be predicted for the current generation of sequence indexes.

[Relevant Systems.] We note that current relational DBMSs [6], Column Stores [26], and Array Databases [27] could eventually be used to store and process sequences. Nevertheless, they cannot efficiently support complex data mining queries, (that is, queries that treat the entire sequence as a single object, such as sequence similarity queries, clustering, classification, etc.), which require fast distance computations among the sequences in the collection, since they do not natively support any mechanisms for pruning the search space. Consequently, these systems cannot offer optimization functionality for the execution of DM queries, which is a key requirement for a SMS.

3. DATA SERIES MANAGEMENT

In this section, we provide an overview of the necessary components of a SMS, and we discuss the envisioned functionality and open research problems for each one of them.

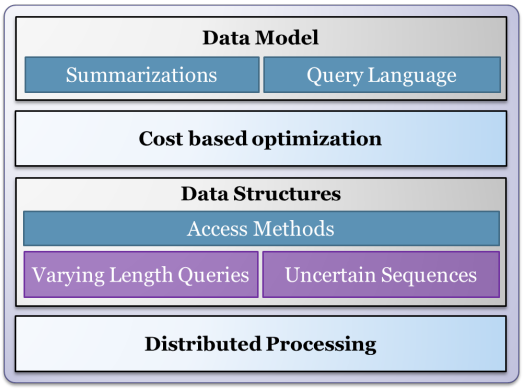


Figure 1: The architecture of a data series management system.

A key element of a SMS is the design of a cost-based optimizer for the execution of sequence queries, with a special focus on complex data mining queries. There is currently no optimizer available for sequence queries, even though it is a necessary component for efficient and scalable processing and analytics. As we discuss next, traditional approaches fail in our setting, and therefore, major breakthroughs are needed in this direction.

The optimizer should depend on and be closely related to the storage and indexing solutions for sequences, two research areas that should also be addressed. The design of the data model should accommodate various sequence summarization techniques, including novel techniques for uncertain sequences, and innovative access methods (i.e., storage and indexing) that will be able to adapt to the user needs (i.e., the query workload). Moreover, particular attention should be paid to optimizations specific to data sequence techniques relevant to modern hardware and distributed environments.

Last, but not least, there is a need for the development of a benchmark specifically designed for data sequences, which as we discuss below is a challenging task in itself. This benchmark will serve to evaluate the state of the art and identify promising ideas and concepts, as well as guide the efforts in the development of new techniques.

In Figure 1, we illustrate the general architecture of a SMS. We elaborate on the individual components of the system in the following sections. We discuss optimization last, since it touches on the rest of the components, and also include a discussion on the need for a data sequence benchmark.

3.1 Data Series Queries

There are various types of data sequence queries that analysts need to perform: (a) simple Selection-Projection-Transformation (SPT) queries, and (b) more complex Data-Mining (DM) queries. Simple SPT queries are those that select sequences and project points based on thresholds, point positions, or specific sequence properties (e.g., above, first 10 points, peaks), or queries that transform sequences using mathematical formulas (e.g., average). An example SPT query could be one that returns the first x points of all the sequences that have at least y points above a threshold. The majority of these queries could be handled (albeit not optimally) by current database management systems, which nevertheless, lack a domain specific query language that would support and facilitate such processing.

DM queries on the other hand are more complex by nature: the processing has to take into consideration the entire sequence, and treat as a single object, therefore being much more complex to process. Examples under this category are: queries by content (range and similarity queries, nearest neighbors), clustering, classification, outlier patterns, frequent sub-sequences, and others. These queries cannot be supported by current data management systems, since they require specialized data structures, algorithms and storage methods in order to be performed efficiently.

Note that the data series datasets and queries may refer to either static, or streaming data. In the case of streaming data series, we are interested in the sub-sequences defined by a sliding window. The same is also true for static data series of very large size (e.g., an electroencephalogram, or a genome sequence), which we divide into sub-sequences using a sliding (or shifting window). The length of these sub-sequences is chosen so that it can contain the patterns of interest.

3.2 Data Model

As we discussed earlier, neither the relational model nor the array model can adequately capture the characteristics of sequences. In the case of relational data, there are various options available for translating sequences into relations and each one of them has significant limitations. On the other hand, in Array Databases we lack the expressive power to define collections of sequences, and are restricted to defining large multi-dimensional matrices that encode both sequence and meta-data on an equal basis, which hinders efficiency.

An ideal sequence model should instead be able to effectively describe collections of sequences and allow us to do operations on them. It should allow us for example to select sequences based on meta-data or based on their values, project them as complete sequences or sub-sequences and join them in a variety of ways for computing calculations. At the same time such a model should intuitively allow for both intra-sequence and inter-sequence aggregations, and be compatible with different sequence summarization methods. Finally, the corresponding query language could be based on previous works [15, 22], suitably extended to deal with data series as single objects, as well as with DM queries.

3.3 Data Structures

A large collection of access methods has been proposed in the literature, able to evaluate different queries under various settings, including both indexes and scan-based methods. A recently proposed representation technique, SAX, has several desired properties, including good approximation quality, small memory footprint, and fast bitwise operations [25]. However, the indexing structure proposed for SAX in certain cases exhibits degenerate performance (i.e., worse than a serial scan), which is not desirable in practice. Initial work in this area is encouraging [7,8], with iSAX2+ demonstrating scalability to dataset sizes 2-3 orders of magnitude more than the current state of the art.

[Adaptive Indexes.] Nevertheless, new techniques are necessary in order to further improve scalability, and more importantly, to significantly boost time performance, in order to match the requirements of modern applications. Our experience with massive sequence collections [8] shows that indexing itself can become a bottleneck in a data sequence analysis task: for example, it takes more than 24 hours to build a state-of-the-art index over a dataset of 1 billion sequences in a modern server machine. It is imperative therefore, to develop techniques that interactively and adaptively build parts of the index, focusing on the data necessary to answer the queries, and making the data available for querying (almost) immediately. Preliminary results in this area are very promising, demonstrating a 7-fold improvement in the time to prepare an index on 1 billion data series and answer 100,000 *approximate* queries [31].

Other promising directions should also be explored, such as methods that rely on fast scans of the data [14, 19]. These directions can provide viable alternatives to the indexes discussed above, and in several situations can be the access method of choice. This is especially true given the data management trend on large-scale parallelization, the usage of compres-

sion, multi-cores, SIMD architectures and the exploitation of available GPUs [20].

We also propose to extend these techniques along two orthogonal dimensions: supporting queries of varying length, and uncertain sequences.

[Varying Length Queries.] Existing techniques only consider collections of data series with the same length, leading to indexes that can answer queries of a fixed (predefined) length. As a result, new access methods that also consider varying length queries have to be developed. Contrary to previous approaches [12], we argue that the information already captured by certain data sequence indexes can be exploited, and is possible to develop new varying-length query answering techniques on top of this.

[Uncertain Sequences.] In several cases, data sequences can be uncertain, that is, the raw data have an inherent uncertainty in their values (e.g., because of errors introduced by the measurement devices), and integrate the solutions to the proposed system. There exist promising studies on modeling and analyzing uncertain sequences [5, 23, 30], but more work is needed in order to improve the quality and time performance [9]. A promising direction in this respect is the modeling of uncertain sequences with possible world semantics based on full-joint distributions, which can retain the correlation information among neighboring points [10]. Nevertheless, there are still important scalability issues to be overcome in order for such techniques to be used with large sequence collections.

3.4 Distributed Processing

During the last years there has been a lot of research on MapReduce systems, where various methods have been proposed to support the indexing of large multidimensional data [16], where an index is distributed among several compute nodes. Nevertheless, up to this point work on sequential data query processing using MapReduce has mainly concentrated on efficiently performing parallel scans of the complete dataset, while all indexing-related studies only consider read-only operations. Even though various approaches have been proposed for speeding up iterative algorithms, none of the proposed models is a suitable match for the algorithms and techniques we need, where timely communications among workers play a crucial role in reducing the amount of total work done. Therefore, there is need for more work in this area, taking into consideration new paradigms as well [2].

3.5 Cost based optimization

As we discussed above, there can be multiple different execution strategies for answering the same query, including the various choices of serial scans, indexes, and processing methods (e.g., parallelization, GPU, etc.). The challenge in choosing the right execution strategy is to estimate the amount of data that such a query will need to access before executing it. For example, a fast parallel SIMD-enabled scan on compressed data might be a better option than the use of a non-optimized index when SIMD instructions are available, but not a better choice when such instructions are not available. All these characteristics have to be exploited by the cost-based optimization models, and considered in a way that is transparent to the user. This problem becomes even more challenging when complex queries involving several operators need to be executed (e.g., consider an analysis task that combines a series of SPT operators as a pre-processing step, and then applies a DM operator).

While in traditional relational databases there are simple and efficient ways in order to estimate query selectivity [6], this is not the case for sequence similarity queries that lie in the heart of most sequence mining algorithms. The challenges in this context arise from the combination of the very high dimensional and sequential nature (i.e., the inherent correlations among neighboring values) of these data.

Up to this point, no efficient methods have been proposed to solve this problem, and ground-breaking work needs to be done. We believe that a promising direction is to carefully study the hardness of a query: being able to control the effort needed to answer a query can be the right step stone for solving the inverse problem, that of estimating the effort it will take to answer a query, before executing it.

3.6 Data Series Benchmarking

Despite the rich literature on methods for indexing and answering similarity queries on data sequences, we note the absence of any related benchmarks. We argue for the need of fair benchmarks that can stress-test sequence processing techniques in a controlled way and to pre-defined levels of query hardness. Such benchmarks will be designed to capture differences in the quality of summarization methods, indexes and storage methods, when working in *combination*, which is what makes the design of such a benchmark a challenging task. Our ongoing work constitutes the first solution towards this directions: it hows that the amount of effort employed by data series indexes can be consistently captured across different indexing approaches, using implementation-invariant measures [32].

4. CONCLUSIONS

Even though data series are a very common data type, there is currently no system that can inherently accommodate, manage, and support complex analytics for this type of data. In this paper, we argue for the special nature of the sequences data type, and articulate the necessity for rigorous work on data series management systems. We propose a SMS that will employ a data model specialized to sequences. The system will be distributed by design, and consider the large volume of sequences, their heterogeneity (in terms of properties and characteristics), and possible uncertainty in their values. Finally, the system will support cost-based optimization, thus, leading to the desired scalability for big sequence analytics.

Acknowledgements

I would like to thank my collaborators, A. Camerra, M. Dallachiesa, J. Gehrke, S. Idreos, I. Ilyas, E. Keogh, M. Linardi, Y. Lou, K. Mirylenka, B. Nushi, and J. Shieh. Special thanks go to K. Zoumpatianos, who has been the driving force behind several of the ideas discussed in this paper.

References

[1] Adhd-200. `http://fcon_1000.projects.nitrc.org/indi/adhd200/`, 2011.

[2] Orleans: Distributed virtual actors for programmability and scalability. MSR-TR-2014-41, 2014.

[3] Sloan digital sky survey. `https://www.sdss3.org/dr10/data_access/volume.php`, 2015.

[4] R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient similarity search in sequence databases. In *FODO*, 1993.

[5] J. Aßfalg, H. Kriegel, P. Kröger, and M. Renz. Probabilistic similarity search for uncertain time series. In *SSDBM*, 2009.

[6] M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. Gray, P. P. Griffiths, W. F. K. III, R. A. Lorie, P. R. McJones, J. W. Mehl, G. R. Putzolu, I. L. Traiger, B. W. Wade, and V. Watson. System R: relational approach to database management. *TODS*, 1(2):97–137, 1976.

[7] A. Camerra, T. Palpanas, J. Shieh, and E. Keogh. iSAX 2.0: Indexing and mining one billion time series. In *ICDM*, 2010.

[8] A. Camerra, J. Shieh, T. Palpanas, T. Rakthanmanon, and E. J. Keogh. Beyond one billion time series: indexing and mining very large time series collections with isax2+. *KAIS*, 39(1):123–151, 2014.

[9] M. Dallachiesa, B. Nushi, K. Mirylenka, and T. Palpanas. Uncertain time-series similarity: Return to the basics. *PVLDB*, 5(11):1662–1673, 2012.

[10] M. Dallachiesa, T. Palpanas, and I. F. Ilyas. Top-k nearest neighbor search in uncertain data series. *PVLDB*, 8(1):13–24, 2014.

[11] P. Huijse, P. A. Estévez, P. Protopapas, J. C. Principe, and P. Zegers. Computational intelligence challenges and applications on large-scale astronomical time series databases. *IEEE Comp. Int. Mag.*, 9(3):27–39, 2014.

[12] S. Kadiyala and N. Shiri. A compact multi-resolution index for variable length queries in time series databases. *KAIS*, 15(2):131–147, 2008.

[13] K. Kashino, G. Smith, and H. Murase. Time-series active search for quick retrieval of audio and video. In *ICASSP*, 1999.

[14] S. Kashyap and P. Karras. Scalable knn search on vertically stored time series. In *KDD*, 2011.

[15] A. Lerner and D. Shasha. Aquery: Query language for ordered data, optimization techniques, and experiments. In *VLDB*, 2003.

[16] H. Liao, J. Han, and J. Fang. Multi-dimensional index on hadoop distributed file system. In *NAS*, 2010.

[17] J. Lin, R. Khade, and Y. Li. Rotation-invariant similarity in time series using bag-of-patterns representation. *J. Intell. Inf. Syst.*, 39(2), 2012.

[18] T. Palpanas, M. Vlachos, E. J. Keogh, and D. Gunopulos. Streaming time series summarization using user-defined amnesic functions. *TKDE*, 20(7), 2008.

[19] T. Rakthanmanon, B. J. L. Campana, A. Mueen, G. E. A. P. A. Batista, M. B. Westover, Q. Zhu, J. Zakaria, and E. J. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *KDD*, 2012.

[20] V. Raman, G. K. Attaluri, R. Barber, N. Chainani, D. Kalmuk, V. KulandaiSamy, J. Leenstra, S. Lightstone, S. Liu, G. M. Lohman, T. Malkemus, R. Müller, I. Pandis, B. Schiefer, D. Sharpe, R. Sidle, A. J. Storm, and L. Zhang. DB2 with BLU acceleration: So much more than just a column store. *PVLDB*, 6(11):1080–1091, 2013.

[21] U. Raza, A. Camerra, A. L. Murphy, T. Palpanas, and G. P. Picco. Practical data prediction for real-world wireless sensor networks. *IEEE Trans. Knowl. Data Eng.*, accepted for publication, 2015.

[22] R. Sadri, C. Zaniolo, A. M. Zarkesh, and J. Adibi. A sequential pattern query language for supporting instant data mining for e-services. In *VLDB*, 2001.

[23] S. R. Sarangi and K. Murthy. DUST: a generalized notion of similarity between uncertain time series. In *KDD*, 2010.

[24] D. Shasha. Tuning time series queries in finance: Case studies and recommendations. *IEEE Data Eng. Bull.*, 22(2):40–46, 1999.

[25] J. Shieh and E. Keogh. iSAX: Indexing and Mining Terabyte Sized Time Series. KDD, 2008.

[26] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. J. O'Neil, P. E. O'Neil, A. Rasin, N. Tran, and S. B. Zdonik. C-store: A column-oriented DBMS. In *VLDB*, 2005.

[27] M. Stonebraker, P. Brown, A. Poliakov, and S. Raman. The architecture of scidb. In *SSDBM*, 2011.

[28] Y. Wang, P. Wang, J. Pei, W. Wang, and S. Huang. A data-adaptive and dynamic segmentation index for whole matching on time series. *PVLDB*, 6(10):793–804, 2013.

[29] L. Ye and E. J. Keogh. Time series shapelets: a new primitive for data mining. In *KDD*, 2009.

[30] M. Yeh, K. Wu, P. S. Yu, and M. Chen. PROUD: a probabilistic approach to processing similarity queries over uncertain data streams. In *EDBT*, 2009.

[31] K. Zoumpatianos, S. Idreos, and T. Palpanas. Indexing for interactive exploration of big data series. In *SIGMOD*, 2014.

[32] K. Zoumpatianos, Y. Lou, T. Palpanas, and J. Gehrke. Query workloads for data series indexes. In *KDD*, 2015.

...like Commanding an Anthill: A Case for Micro-Distributed (Data) Management Systems

Holger Pirk
MIT CSAIL, Cambridge, USA
holger@csail.mit.edu

ABSTRACT

Computer system architecture has changed: an assembly of autonomous components has replaced the omnipotent CPU and its legion of dumb devices. Database Management System (DBMS) architecture, however, does not yet reflect this change: it is still dominated by a centralized kernel that limits the autonomy of the devices and, thus, their ability to exploit their increased "smartness". Distributed data management research can serve as an inspiration for an architecture that addresses this problem. However, the respective algorithms were never designed with CPU efficiency in mind implementing principles like dynamic programming and recursion.

More than two decades ago, the transition to memory resident databases spawned a plethora of research on CPU-efficient query processors. We predict that hardware heterogeneity will trigger a similar line of research on CPU-efficient distributed algorithms and architectures. In this paper, we examine benefits and challenges that come with such a *micro-distributed* database management system. We also discuss a number of approaches that we consider steps towards a micro-distributed system.

1. INTRODUCTION

Modern computer systems are not the centralized machines they used to be: they are an assembly of autonomous components. While connected through relatively simple interfaces, these systems incorporate quite intricate control logic. Modern Solid State Disk (SSD) firmwares, e.g., have thousands of lines of code and implement features such as garbage collection, buffer management and request queuing [6]. This stands in stark contrast to the assumptions that classic data management system design was based on: a *Central* Processing Unit (CPU) with autocratic powers that micro-manages the program execution flow. In such an architecture, subsystems and devices get specific instructions (move arm, read sector, perform arithmetic operation) from the CPU and any delay in execution causes stall cycles in the CPU which hurts performance. Increased execution autonomy could, however, be used by the devices to increase performance, energy efficiency or even lifespan. Aforementioned SSD firmwares, e.g., combine multiple Out-of-Order (OOO) writes which reduces overhead when writing to memory cells and improves wear-leveling. Similarly, massively parallel co-processors merge multiple sequential programs into a single (massively) parallel program.

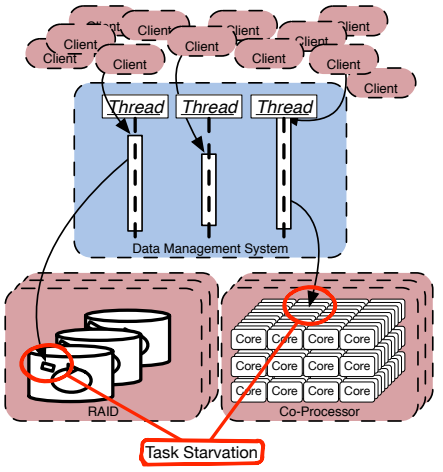


Figure 1: The DBMS: a Concurrency Bottleneck

However, all of these benefits hinge on a single point: the software has to provide the necessary degree of autonomy to each of the subsystems. While device autonomy has a number of dimensions [11], let us focus on execution autonomy for now:

a *device with execution autonomy* has the freedom to execute operations in any way or order it sees fit.

To ensure utilization there have to be enough operations to choose from at any given time. Unfortunately, the required OOO degree is highly device specific: SSD command queue length is currently limited to 32 by the SATA specification but in-

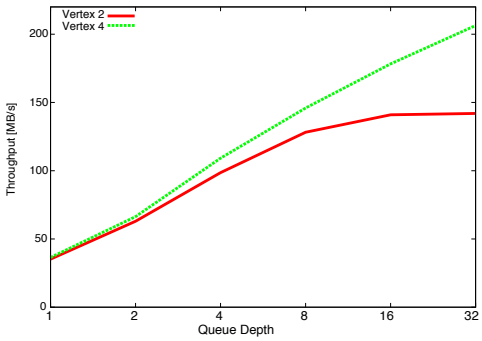


Figure 2: Impact of out-of-order degree on SSD performance (taken from [15])

creasing to many thousands in next-generation devices [4]; co-processor cards already support hundreds to thousands of hardware threads. To encompass all of these as well as future devices, it is common to go for maximum, i.e., massive (data) parallelism (e.g., [10]). Unfortunately this approach is not applicable to task parallelism: even the degree of parallelism of current parallel devices cannot be sustained by the centralized components of classic data management systems (see Figure 1) leading to *Task Starvation*. *Task Starvation*, i.e., a lack of independent work items, leads to insufficient optimization opportunities and, thus, suboptimal performance of hardware devices (we dicuss this point in more detail in Section 2.1).

DBMSs are not the only systems suffering from this problem. In fact, they have inherited it from the underlying component in the application stack: the operating system. The root of the problem lies in preemptive multitasking and the associated context thrashing: to micro-manage an autonomous device, a thread has to be frequently awoken, its context be loaded, the device status checked, the context persisted and the tread retired. This causes substantial overhead creating a scalability bottleneck.

This problem, however, is not entirely new to data management systems: distributed Database Management Systems (DBMSs) by definition deal with autonomous, remote subsystems. When working with such autonomous subsystems, the system cannot make assumptions about the order, timeliness or quality of the result [11]. We argue that, to withstand the challenges of the age of massively parallel hardware, centralized databases systems have to become equally robust against autonomous behavior as their distributed cousins: become *micro-distributed*.

Unfortunately, transplanting ideas from the domain of distributed databases into centralized systems is not entirely trivial. We will dedicate the next two sections to a discussion of benefits (Section 2) and challenges (Section 3) of such an approach. We will also provide a brief survey of the approaches that may be steps towards a micro-distributed data management system in Section 4. We will conclude with a brief outlook in Section 6

2. BENEFITS

The primary benefit of a micro-distributed DBMS is, naturally, the increase in performance and scalability that comes with increased execution autonomy. However, there are a number of secondary benefits that we want to cover in this section.

2.1 Performance

Naturally, the performance impact of micro-distribution depends on the degree to which the targeted device can exploit the OOO-degree. It is, therefore, hard to make substantiated predictions about the performance impact. However, there are studies that can help to gain an impression of the performance impact that high OOO-degrees can have on device performance.

Figure 2 shows the result of a simple micro-experiment: random read requests to a single SSD (comparing devices with different internal OOO-degrees). The experiment shows a steady logarithmic increase in transfer rate when scaling the degree of parallelism. While this experiment was conducted using a hand-coded driver on an FPGA, it demonstrates the potential of execution autonomy in the hardware. In current-generation SATA-SSDs the OOO-degree is limited to 32. Next generation non-volatile-ram devices have, by specification, an OOO-degree of 65,536. This illustrates the need for more OOO-parallelism in the DBMS.

2.2 Secondary Benefits

Besides the performance benefits through device autonomy, there are a number of secondary benefits inherited from classic distributed systems.

A Unified Architecture

Most industry grade DBMSs support distributed as well as centralized databases. However distributed data management is usually implemented in a separate module that is only activated if needed. The core, i.e., centralized, query processor usually follows the "autocratic" model we described earlier while the distributed module is implemented under the assumption that remote systems have substantial autonomy.

What we propose essentially merges the two subsystems. While there were good reasons to separate them (see Section 3), unifying them promises a significant reduction in system complexity.

Separation of Concerns

An increase in autonomy not only enables devices to execute multiple requests OOO, it also allows them to evaluate complex operations completely autonomous. This evidently holds for Graphics Processing Unitss (GPUs), which became programmable almost a decade ago. However, recent work turned even storage devices such as SSDs into *Smart Devices*, i.e., devices with builtin processing functionality. They can be used to efficiently perform operations such as prefiltering [1], aggregation [16] and even document ranking [12] without CPU-involvement. However, such functionality further aggravates the latency of involved storage devices.

Device Hot-Swapping & Virtualization

One of the appealing traits of distributed databases is the capability to add and remove subsystems at runtime. A micro-distributed system extends this capability to centralized DBMSs. This capability allows, e.g., the virtualization of co-processors - a service that cloud services like Amazon's EC2 do not yet provide. In a multi-tenant setup, e.g., the system could swap in an accelerator card to deal with load spikes at much lower (transfer) costs than scaling out to an additional node.

3. CHALLENGES

When considering the benefits, it seems surprising why centralized and distributed database kernels ever became separated. The reason lies in the effort of managing of autonomous subsystems. We dedicate this section to a discussion of the challenges that come with a (re)unification of centralized and distributed kernels.

3.1 Efficiency

As indicated previously, distributed data management research can provide a good starting point for work towards micro-distributed systems. Naturally, the amount of research focusing distributed data management is staggering ([11] provides the background for much of our discussion). Distributed data management addresses a number of specific challenges such as distributed concurrency control, data and query placement, transfer-conscious processing and schema diversity. The CPU-efficiency of the underlying algorithms, however, is not usually considered an important factor. This has two main reasons:

- the number of concurrent queries usually run in an order that does not pose much of a scalability problem (hundreds rather than tens of thousands) and
- computational resources are usually abundant because the query costs are dominated by data transfer over the network.

These do not hold for micro-distributed DBMSs. In fact, we believe scalable CPU-efficiency to be the main challenge for micro-distributed systems.

When in-memory data management solutions became feasible due to falling prices, they faced a similar situation: the obvious first step was to simply run existing software on memory-resident data - Data structures (tree-indices, slotted pages, ...) as well as system architectures (buffer managers, volcano-style query processors, ...) were left untouched. However, it quickly became apparent that the CPU overhead of these techniques became the dominating cost factor [7]. It took almost two decades of research to increase CPU efficiency to a point at which the memory is the dominating bottleneck again [3]. We believe similar challenges lie ahead in micro-distributed systems.

3.2 Finding Appropriate Work

While autonomous devices are to a large extend programmable, they still have limitations. GPUs, e.g., have thousands of processing cores but only few instruction schedulers. For that reason, the cores in one processing unit (between 32 and 128) have to share an instruction scheduler. This restricts the program to only a few (around 50) independent instruction streams. Similarly, smart SSDs can evaluate range predicates on integer values and calculate (simple) hashes but cannot provide functionality like floating point arithmetic or sorting. It will be a challenge for DBMS to break down complex queries into executable sub-tasks and, if need be, re-program the devices. Finding the right level of abstraction to "software-define" devices will be important.

3.3 Conveying High-Level Knowledge

With the limitations of the the devices also comes a lack of high-level understanding of the problem at hand. While the DBMS can keep track of high-level information such as operator dependencies or characteristics, it will be hard to convey these insights to the devices. On the data management side, this has the potential to spawn research on the creative use of existing functionality. On the hardware side, it can fuel work on novel interfaces and optimizations.

4. THE STATE OF THE ART

Before concluding, we want to briefly discuss promising steps towards a micro-distributed DBMS.

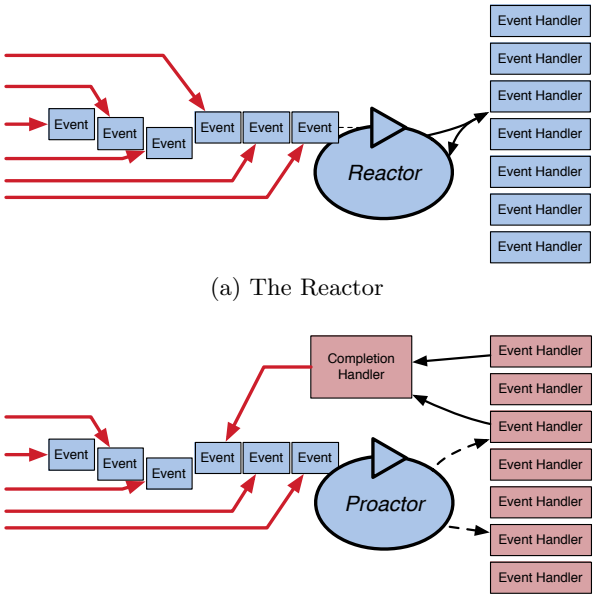
4.1 Data-Parallel Systems

Since the primary goal of micro-distribution is an increase in the OOO-degree in the underlying hardware, it is sensible to look at the most obvious source of parallelism: data. In fact, data parallelism is comparatively easy to manage since it is usually localized within a kernel operator. Data management on GPUs, e.g., has exploited data parallelism for almost a decade [9]. However, the data parallel approach falls short for operations that are not inherently data parallel such as index accesses or transaction processing. Consequently, data-parallel micro-distribution is currently limited to analytical workloads.

4.2 Multi-Kernel Approaches

We are by no means the first to recognize the convergence of modern computer architecture and distributed system design. Consequently, related work can be found in other fields of computer science research. In the field of operating systems, for example, the concept of the *Multi-Kernel OS* has recently been introduced [2]. The idea is to overcome multicore scalability issues of user applications by running an OS-Kernel instance per core and statically assigning execution threads to kernel instances. Any inter-thread communication is performed using asynchronous message passing. Existing message passing implementations, however, were designed for comparatively rare inter-machine or inter-process communication. The sheer amount of traffic that is generated by Multi-Kernel applications justifies a reimplementation of the message passing concept - aware of factors like Cache Line Sizes, Coherence Protocols and Non-Uniform Memory Access [2]. The existing work on Multi-Kernel OSes indicates comparable performance yet better scalability than existing operating systems. This is exactly the kind of benefits we strive for in the database domain.

A recent piece of work [14] has performed the first step towards such a micro-distributed data management system: a direct port of a classic distributed system. We believe this to be a step similar to the first direct ports of disk-resident data management kernels to memory-resident databases: a compelling case for the concept in a very restricted scenario (a replicated shared-nothing or perfectly partitionable database&workload). It seems unlikely that such a simple port will be competitive under less partitionable workloads. Just like a Multi-Kernel OS is fundamentally different from multiple Single-Kernel OS kernels, a micro-distributed DBMS is different from multiple federated database instances on a single node. We believe that, just like in operating systems, the key is lightweight message passing. Fortunately, there is a precedent for systems built around the idea of highly concurrent lightweight message passing as well: *Reactive Systems*.



(a) The Reactor

(b) The Proactor

Figure 3: The Design Patterns of Reactive Systems

4.3 Reactive Systems

Reactive systems implement the *Reactor Design Pattern* [5] (see Figure 3a). This Pattern was developed to address the high level of concurrency in the hardware components of communication systems which usually only support moderate parallelism in hardware. But also many highly concurrent networking software packages such as Twisted or Node.js implement the reactor pattern.

The principal idea is that requests are sequentialized by a single, lightweight *Reactor* thread and subsequently dispatched to *Event Handlers* which perform the actual work. Such reactive systems replace preemptive multithreading with a form of cooperative multithreading. This leads to much better scalability for task-parallel applications as long as all threads cooperate.

However, this points out at a crucial point: *Event Handlers* must only take a relatively short amount of time to process an *Event*. Since the *Reactor* is blocked while a *Handler* is active, the system be-

comes unresponsive if a *Handler* takes a long time to complete.

Therefore, many reactive systems actually implement a combination of the *Reactor Pattern* and the very closely related *Proactor Pattern* [13]. The main difference between the two is the concurrency model. In a *Proactor* implementation (see Figure 3b), the *Event Handlers* do not "borrow" the execution thread of the *Reactor* [13]. Instead, *Event Handlers* are invoked with a *Completion Handler* that is invoked by the *Event Handler* once it is done processing a particular *Event*. This makes the *Proactor* more suited for handling (relatively) long running operations because the *Reactor* thread is not blocked and can continue to dispatch *Events*. The *Proactor Pattern* effectively re-parallelizes the sequentialized events by dispatching them to concurrently running *Event Handlers*. This pattern efficiently multiplexes the concurrent events of the parallelization bottleneck (the task-sequential device) and reparallelizes them when appropriate (before processing them using the task-parallel device).

5. A REACTIVE DBMS

Earlier, we identified preemptive multithreading as a major scalability bottleneck when managing micro-distributed devices. Since the *Reactor Pattern* was designed for this very problem, it is sensible to consider a *Reactive Data Management System* as an appropriate response to this challenge. We, thus, want to use this section to outline the design of such a system.

The Reactor. Most reactive systems implement the *Reactor* in some kind of *Event Loop* that processes occurring *Events* in the order in which they are enqueued. Naturally, a reactive DBMS would also incorporate such an *Event Loop* (see Figure 4). However, since the *Event Loop* is merely responsible for dispatching work to the appropriate *Handlers*, this component is very lightweight. The actual component logic is encapsulated in the *Event Handlers*.

The Event Handlers. All of the *actual* functionality of a reactive DBMS is implemented in the *Event Handlers*. This includes classic components like Optimization, Recovery or Buffer Management. However, it also includes all data access operations which are usually part of high-level operators.

Reactive Query Processing.

A typical select & project query on a column oriented database, e.g., is likely to go through a high number of *Event Handlers*: imagine starting with

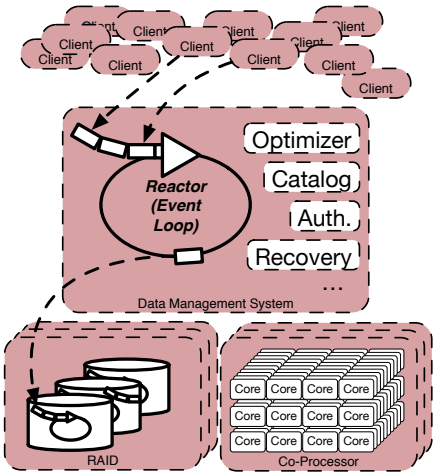


Figure 4: A Reactive Data Management System Architectures

a GPU-based handler performing massively parallel scan of the selection-column to retrieve the ID(s) of matching tuple(s); after that, every projected attribute could result in at least one call to a project-handler touching disk-resident columns. Even potential accesses to secondary index structures such as Primary- or Foreign-Key indices are processed using individual *Events* with every access to a node of tree-based index constituting an event. By taking the concept to these extremes, we can maximize the number of outstanding SSD requests and, thus, the resulting optimization opportunities.

This indicates one of the greatest challenges in the development of a *Reactive DBMS*: event processing overhead. While asynchronicity helps to increase the OOO-degree available to devices, the CPU overhead of the event processor can result in significant costs. A potential solution could be to have multiple *Event Loops* hardcoded for a specific class of operations such as tree lookups or lock managing. In doing so, we approach the design of an interesting artefact of related work: *StagedDB* and the underlying idea of *one operator - many queries* [8].

One Operator - Many Queries.

StagedDB is, to the best of our knowledge, the first system to part from the *one query - many operators* model, replacing it with a *one operator - many queries* approach. The idea is to have a single, static operator instance per "operator class" rather than an instance per query plan node. Such a design improves instruction cache locality and allows the system to exploit occuring work-sharing opportunities. However, each operator identifies sharing

opportunities itself and uses synchronous interfaces to the underlying hardware. This not only increases complexity and makes performance vulnerable to changes of hardware parameters, it also spends significant resources on identifying sharing opportunites which often counteracts the benefits.

In contrast to *StagedDB*, our reactive approach does not invest effort in identifying sharing opportunities. Instead, it generates many OOO-request and leaves the identification of sharing opportunities to the hardware itself. In doing so, a reactive DBMS not only removes the substantial overhead of work-sharing analysis from the CPU but pushes it to the component that is specifically optimized for such analysis: the hardware device controller. *StagedDB* may, however, serve as a blueprint to mitigate the CPU overhead of a reactive DBMS by clustering low-level operations into static groups.

6. CONCLUSION

We have argued that the increasing heterogeneity of computer systems makes them more akin to distributed systems than their centralized ancestors. We believe that, to efficiently manage such heterogeneous systems, centralized DBMSs have to adopt distributed data management techniques. However, the need for CPU-efficiency makes a direct port of these techniques unfeasible. Like memory-resident data management triggered an era of research on CPU-efficient query processing, we believe that heterogeneity will spawn an era of research on CPU-efficient distributed algorithms. We have provided indications that this era has already begun but also provided a number of challenges that still lie ahead.

7. REFERENCES

[1] Acharya, A., Uysal, M., and Saltz, J. Active disks: Programming model, algorithms and evaluation. In *ACM SIGOPS Operating Systems Review* (1998), vol. 32, ACM.

[2] Baumann, A., Barham, P., Dagand, P.-E., et al. The multikernel: a new os architecture for scalable multicore systems. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (2009), ACM.

[3] Boncz, P. A., Kersten, M. L., and Manegold, S. Breaking the memory wall in monetdb. *Communications of the ACM 51*, 12 (2008).

[4] Cobb, D., and Huffman, A. Nvm express and the pci express ssd revolution, 2012.

[5] Coplien, J., and Schmidt, D., Eds. *Pattern Languages of Program Design*. Addison-Wesley, 1995, ch. Reactor: An object behavioral pattern for concurrent event demultiplexing and event handler dispatching.

[6] Cornwell, M. Anatomy of a solid-state drive. *Commun. ACM 55*, 12 (2012).

[7] Garcia-Molina, H., and Salem, K. Main memory database systems: An overview. *Knowledge and Data Engineering, IEEE Transactions on 4*, 6 (1992).

[8] Harizopoulos, S., Ailamaki, A., et al. Stageddb: Designing database servers for modern hardware. *IEEE Data Engineering Bulletin 28*, 2 (2005), 11–16.

[9] He, B., Lu, M., Yang, K., Fang, R., Govindaraju, N. K., Luo, Q., and Sander, P. V. Relational query coprocessing on graphics processors. *ACM Transactions on Database Systems (TODS) 34*, 4 (2009).

[10] Ladner, R. E., and Fischer, M. J. Parallel prefix computation. *Journal of the ACM (JACM) 27*, 4 (1980).

[11] Özsu, M. T., and Valduriez, P. *Principles of distributed database systems*. Springer Science & Business Media, 2011.

[12] Putnam, A., Caulfield, A. M., Chung, E. S., Chiou, D., Constantinides, K., Demme, J., Esmaeilzadeh, H., Fowers, J., Gopal, G. P., Gray, J., et al. A reconfigurable fabric for accelerating large-scale datacenter services. In *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on* (2014), IEEE.

[13] Pyarali, I., Harrison, T., Schmidt, D. C., and Jordan, T. D. Proactor-an object behavioral pattern for demultiplexing and dispatching handlers for asynchronous events. *Proceedings of the 4th Annual Pattern Languages of Programming Conference* (1997).

[14] Salomie, T.-I., Subasu, I. E., Giceva, J., and Alonso, G. Database engines on multicores, why parallelize when you can distribute? In *Proceedings of the sixth conference on Computer systems* (2011), ACM.

[15] Sidler, D. Column storage for fpga-accelerated data analytics. Master's thesis, ETH Zürich, 2013.

[16] Woods, L., István, Z., and Alonso, G. Ibexan intelligent storage engine with support for advanced sql off-loading. *Proc. VLDB Endowment (PVLDB)* (2014).

The Best of Two Worlds: Present Your TODS Paper at SIGMOD

Christian S. Jensen
csj@cs.aau.dk

It just became even more attractive to publish your research results in *ACM Transactions on Database Systems*: The leadership of ACM SIGMOD and TODS have decided to offer the authors of certain TODS papers the opportunity to present their paper at the "next" SIGMOD conference. This agreement aims to make it more attractive to members of the SIGMOD community to publish in TODS, as well as to further enrich the technical program at the SIGMOD conferences.

Journal and conference publication differ in a number of respects. In the following, I review important differences, from the perspective of journal publication, and present a case for publication in TODS.

World One: Journal Publication. When a submission is received for consideration of publication in TODS, the submission is assigned to an Associate Editor who then is in charge of handling the submission and, in a sense, serves as the submissions ombudsman: The handling Associate Editor aims to do what is right for the submission and will take into account the authors responses to reviews. While the aim is to provide review results within 4 months, the journals review process can accommodate special circumstances as needed to get things right. For example, additional reviews can be obtained in a review round, and an additional round of reviewing can be introduced.

The traditional conference review process has a fixed schedule of deadlines and does not offer this flexibility. Some conferences have tried to achieve some of the flexibility by allowing one round of revision. Some conferences have also introduced procedures that may be viewed as a means of approximating the Associate Editor role as found at journals. They have introduced program committee vice-chairs and meta-reviewers, and they have introduced author feedback.

In my experience, these innovations to the conference review process are valuable but do not combine to yield the benefits of the journal review process. Specifically, what I call "hit-and-run" reviews still occur at times. These are superficial reviews that simply reject a paper without offering specifics. Key reasons why such reviews occur is that they are fast to do and that reviewers know that they can get away with them because there is no time for iteration. And I believe that the vice-chair and meta-reviewer roles are not always effective, a key reason also being tight deadlines. When serving in those roles, one often has to make accept/reject recommendations with the information already available.

As another difference between the journal and conference review processes, the Associate Editor who handles a submission recruits the reviewers from the global population of researchers, not from a fixed program committee that represents a fraction of the community. Admittedly, the program committees of top conferences consist of excellent researchers, and, even with the possibility of recruiting from the entire population, it is not always possible to recruit the top experts as reviewers of a particular journal submission.

The characteristics of the journal review process described here enable a more careful journal review process. The additional iterations that characterize the journal review process generally yield improvements to the results being published and their presentation.

Next, TODS papers are allowed to be 45 pages long in TODS format, and accompanying electronic-only appendices with no length restrictions are allowed. This arrangement affords authors exactly the space they need in order to present their results as best as possible.

Further, accepted TODS papers are available online at the TODS website when they are ready; they are not hidden until they appear in a journal volume (or until a conference proceeding is made available, in the conference setting).

World Two: Conference Publication. However, conference publication also carries benefits inherently not available with traditional journal publication. When publishing in conference proceedings, the authors get to present their paper at a conference, typically either in a talk, as a poster, or both. And conference publication often means that the authors get funding that allows them to attend the conference, which then also allows them to attend other talks and to network with colleagues. Con-

ference presentation also increases the visibility of the work.

TODS Paper Presentation at SIGMOD. The new agreement with ACM SIGMOD means that authors of TODS papers also get these benefits! The eligible papers include original research papers (i) that are not extensions of conference papers, (ii) that were accepted or published approximately within the year preceding the particular SIGMOD conference, and (iii) that were not already offered this opportunity at the previous conference. Focused surveys, critiques, corrections, and similar type papers are not eligible. These limitations mean that only entirely new, full papers are accommodated.

The presentation of a TODS paper at SIGMOD is in the form of a poster. As the data management community grows, more and more papers will be accepted at conferences such as SIGMOD. This means that there either has to be more parallel sessions with talks or that only some papers can be selected for presentation in the form of a talk. My guess is that poster presentation will gain in prominence in the years to come.

The agreement can be found in the "Policies" section of TODS's website, at http://tods.acm.org.

I hope that the opportunity to present TODS papers at SIGMOD will attract additional submissions to TODS.

Acknowledgments: Walid Aref and Divesh Srivastava provided helpful suggestions. However, they should not be held accountable for the views expressed here.