A platform for eXtreme Analytics

A. Balmin
K. Beyer
V. Ercegovac
J. McPherson
F. Özcan
H. Pirahesh
E. Shekita
Y. Sismanis
S. Tata
Y. Tian

With the rapid increase in the volume of data that enterprises are producing, enterprises are adopting large-scale data processing platforms such as Hadoop® to store, manage, and run deep analytics to gain actionable insights from their "big data." At IBM Research - Almaden, we have been helping enterprise customers build solutions exploiting data-intensive analytics. Our deep experience with actual users has led to an extensive understanding of the platform requirements needed to support these solutions, and our goal is to provide a powerful analytics platform, which we call eXtreme Analytics Platform (XAP), that can be used to create solutions for customer problems that have not been economically feasible to solve until now. XAP provides Jaql [i.e., JavaScript® Object Notation (JSON) query language, a scripting language to specify data flows, tools, and techniques to optimize the runtime execution of these flows], an improved task scheduler, connectors to data warehouses, and libraries for advanced analytics. Many of these technologies have been transferred to the IBM InfoSphere BigInsights™ product. In this paper, we describe the overall design principles and technology of XAP.

Introduction

Many enterprises are exploring and starting to adopt large-scale data processing platforms such as Hadoop** for solving their data-intensive analytic problems. Our group at IBM Research - Almaden has been working with others in IBM and our customers to build such solutions. Through this experience, we have repeatedly observed certain use-case patterns and have focused our research on platform extensions that allow us to better develop solutions for these patterns. In this paper, we describe some of our customer experiences and design principles and then focus on our research on platform extensions.

Early large-scale data processing systems include the Google MapReduce paradigm [1] and its open-source implementation Hadoop [2], as well as Microsoft Dryad [3]. Web-centric enterprises, which were both developers and early adopters of scale-out architectures, quickly recognized the value of higher-level languages within this environment, evidenced by Google Sawzall [4], Microsoft DryadLINQ [5], the work of Facebook on Apache Hive [6], and the work of Yahoo! on Apache Pig [7]. Although these early systems focused on the internal needs of web companies, we focused on the adaption of these platforms for all enterprise customers.

There are many activities in industry and academia related to Hadoop. Some focus on specific solutions [8], and others focus on specific kinds of analytics such as text information extraction or machine learning [9–11]. There has been particular focus on the use of Hadoop for structured-data warehouse workloads [6, 12, 13]. Our observation is that customers want to use these data-intensive analytic platforms for a variety of use-case patterns, and indeed, the power of the platform is in its support for a great variety of data, the flexible integration of this data, and rich and varied analysis resulting from different use-case patterns and the interplay between these use cases. Our work has focused on the underlying platform to support this variety of workloads. This work is complementary to these other efforts, and furthermore, one of our design principles is that the platform must support the integration of a variety of data and analytics.

We describe a few areas of our work including the Jaql [JavaScript** Object Notation (JSON) query language] project, providing a high-level language that is automatically compiled to, and executed as, sequences of MapReduce jobs. We also describe the invention of runtime improvements

Digital Object Identifier: 10.1147/JRD.2013.2242693

inspired, in part, by our research experience with scheduling, relational database, and other enterprise system technologies. In addition, we discuss the integration of these new platforms with other systems found in the enterprise and the exploration of scalable analytics tailored for Hadoop. With all of our work, we have upheld two fundamental tenets: 1) diverse data types must be supported, ranging from unstructured to highly structured data; and 2) we must not compromise the scalability, fault-tolerance, or flexibility of Hadoop. The eXtreme Analytics Platform (XAP) is the early name given to a project at IBM Research - Almaden, which started in 2008 as a focused effort to extend the capabilities of Hadoop [2]. We collaborate with many research, product, and services groups in IBM. We use the term "XAP" as a convenient way to refer to the work described here. Many of the technologies discussed have been transferred to the IBM InfoSphere BigInsights* [14] product.

Our work with customers, both internal and external, can be summarized by the following three use-case patterns and customer examples that exemplify the value of many aspects of our project focus.

Pattern 1: Complex analytics for structured data

An early customer with whom we worked is an established corporation in the credit-card business, with leadership analytics capabilities for predicting credit-card fraud, and strong data-warehouse and traditional business intelligence capabilities for analyzing merchants, account holders, merchant and consumer banks, and many other critical aspects of the customer's business. Our work with the customer centered on the use of Hadoop to extend the customer's fraud analytic capabilities even further, by providing a scalable and cost-effective platform for storing many years of credit-card transaction data and supporting development of new prediction techniques as new fraud patterns emerge. This motivated an early focus on performance for highly structured data, in addition to our already strong support for unstructured data. Analytics are highly varied, using custom algorithms and many different third-party modules. This further highlighted the importance of the support of Jaql for external modules, and our exploration of the use of R, a statistical analysis system, in this massively scalable environment [15]. Valuable data exists in the customer's data warehouses regarding accounts, merchants, and other important entities that cannot be derived from just the raw transaction data, and thus, there is a need for strong integration between the customer's different enterprise systems.

Pattern 2: Ad hoc exploration

A more recent customer is a leading energy company for harnessing wind-generated power. Simulating wind patterns throughout the world is both critical for the customer's business and produces petabytes of data. We used XAP to develop applications that let the customer's scientists explore the simulation output using ad hoc queries. This application had to be fast, so we applied many ideas from relational databases, such as column stores [16] and co-partitioning [17], to make better use of resources and thereby reduce response times. In addition, we developed novel techniques for MapReduce, such as the FLEX scheduler [18] and Adaptive MapReduce [19], that reduce response times as well as improve manageability.

Pattern 3: Applications for extract-load-transform and analysis of unstructured data

Unstructured data, ranging from large document collections to social media and application logs, has been a rich source of use cases for XAP. Often, the data undergoes an *extract-load-transform* (ELT) process from which more regular, (semi-)structured data is extracted. For this purpose, we developed Jaql modules that harness SystemT [9] so that we can apply sophisticated information extraction rules and libraries in parallel. After the ELT stage, the data is often aggregated and exported to systems such as data warehouses or search indexes. In addition, complex analytics on the features derived from the raw data is often performed for topic modeling, frequent pattern analysis, or next-action recommendation. Examples of applications that adhere to this pattern include Cognos* Consumer Insights [20], intranet search with the semantic search engine ES2 (Enterprise Search 2) [21], and deep analysis of public data with Midas [8].

Architecture overview

Figure 1 shows the general architecture of XAP. XAP operates on a cluster of machines that are deployed and administered using cluster management tools and installed with a distributed file system such as HDFS (Hadoop Distributed File System) [22] or GPFS* (General Parallel File System*) [23]. In this paper, we focus on the new components of XAP and enhancements to MapReduce, which are shown in the numbered boxes in Figure 1 (1–5). The XAP stack improves performance and manageability and makes it easier to develop new applications as well as connect to the existing enterprise ecosystem, which include various tools such as those involving ELT tools, the data warehouse, and metadata tools. We first describe Jaql (labeled "1" in Figure 1), a high-level scripting language that is used for creating both ELT and analytic workflows on Hadoop. The data model of Jaql is based on JSON, and it provides flexible schema handling. Next, we describe runtime improvements including the FLEX scheduler, shown in Figure 1 (labeled "2"), which optimizes allocation of resources to Hadoop jobs, and the enhancements (Figure 1, labeled "3") that substantially improve performance and manageability of MapReduce-based applications. Here, we

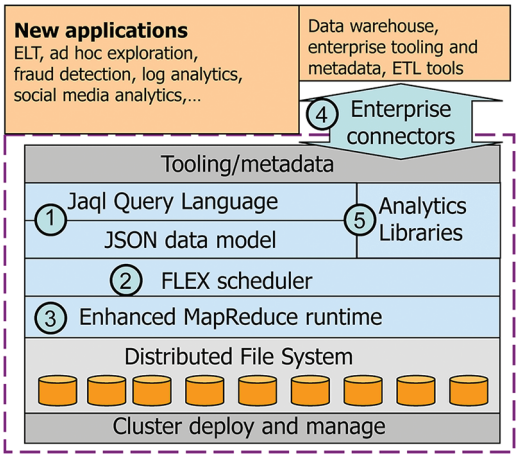

Figure 1

General architecture for the eXtreme Analytics Platform. (JSON: JavaScript Object Notation; ELT: extract-load-transform.)

use the term "fair scheduling" to denote a method of assigning resources to jobs such that jobs receive, on average, an equal share of resources through time. For the enterprise connectors shown in Figure 1 (labeled "4"), we focus on data-warehouse connectors to transfer data between Hadoop and the warehouse, as well as initiating Jaql jobs from within the warehouse. Finally, we describe the libraries and tools for complex analytics (Figure 1, labeled "5").

Jaql

Jaql is a data-processing system that provides a scripting language, compiler, and runtime system [24]. It was designed to process "large" collections (terabytes to petabytes) of "small" objects (bytes to megabytes) by harnessing the Hadoop implementation of MapReduce [1] for scalable data processing. Just as SQL (Structured Query Language) and relational database systems provided a proven abstraction for data management, high-level declarative languages such as Hive [6], Pig [7], and Jaql have exhibited similar advantages when compared with developing data flows directly with MapReduce. However, flexibility is still needed, so Jaql was designed to allow users to directly exploit MapReduce when needed, while blending such low-level specifications with declarative data flows. We refer to such blending as *physical transparency* and discuss it in more detail below. We first introduce the data model, then the scripting language and the runtime, and provide several examples where Jaql has been successfully used.

Enterprise data comes in many forms, ranging from highly heterogeneous data, such as web pages and log files, to homogeneous data such as weather simulation output and financial transactions. Consequently, the Jaql Data Model (JDM) was chosen to permit flexibility, so that such heterogeneity can be modeled while allowing structure to be specified when such information is available. The JDM is based on JSON [25], allowing collections of self-describing, nested, and heterogeneous data to be processed. In addition, the JDM has extended atomic types such as binary and date types. JDM schemas are used to specify homogeneity within a collection and are treated as a constraint in which the structure may be partially specified. This is useful for queries for which only parts of the data are known or needed, and the remainder can be treated as "payload" or "pass-through" data. The example in **Figure 2** shows the JSON representation of a collection of intranet pages as they are processed by a Jaql data flow. Box (a) in Figure 2 shows an *array* (or collection) of *records*, each record containing a set of *field* names and associated values. All records contain a Uniform Resource Identifier (URI) and in some cases content, metadata, or both. The subsequent boxes (b, c, and d) show how the data in box (a) is transformed by the subsequent Jaql operators.

The scripting language of Jaql was designed to easily specify data flows whose operators consume and produce JDM values. The overall design was influenced by functional and query languages. As a result, functions and lazy evaluation (e.g., expressions are evaluated when values are needed) are a central feature in Jaql, as is the core collection of "bulk" operators that were inspired by relational operators (e.g., filter, projection, grouping, and join). Because objects may be complex, Jaql was designed so that any expression could be applied at any level of nesting, thus emphasizing composability. Since Jaql was designed to be a "glue" language that integrates libraries and other data processing scripts, users can add their own operators, package scripts, libraries, and resources to modules. The data flow in Figure 2 illustrates familiar operators, such as *reading* and *writing*, *transforming* each element, and *grouping*, as well as the *expand* operator for nested data that performs an unnesting. This data flow extracts all record fields, unnests all field names into a single array, and counts how many times they occur.

The Jaql system is composed of 1) an interpreter that *parses* Jaql scripts into an internal representation called an ExprTree (expression tree), 2) a heuristic *rewriter* that transforms the ExprTree into a form that exploits parallelism and is more efficient, and 3) an *evaluator*. If possible, the rewriter produces a final plan that is a directed acyclic graph (DAG) of MapReduce jobs. Jaql evaluates the final plan by launching zero or more MapReduce jobs. For the data flow in Figure 2, Jaql requires only one MapReduce job for the data flow; the map step handles the read, transform, expand, and grouping key extraction, while the reduce step groups and writes the result.

Interestingly, properties of any ExprTree can always be used as an evaluation plan and evaluated, and it can always

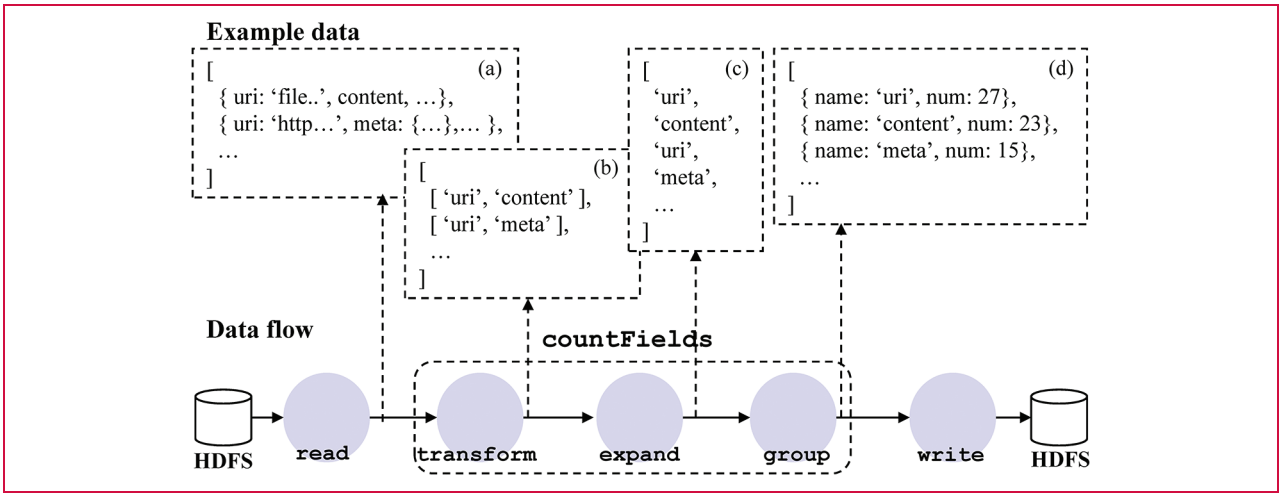

Figure 2

Dataflow that counts field occurrences in a collection. (HDFS: Hadoop Distributed File System.)

be decompiled into a valid Jaql script. If further performance tuning is needed, a user can manually optimize it by directly modifying the plan using a text editor. This can be contrasted with any RDBMS (relational database management system) for which the physical plan cannot be modified. When an RDBMS optimizer produces the wrong plans, *hints* can suggest which physical plan to use. However, hints do not cover all cases, which is why Jaql gives users full control when needed.

Such blending of declarative scripts with low-level plans is what we refer to as *physical transparency*. This feature illustrates many of the distinguishing features of Jaql. For example, MapReduce, which is a core Jaql operator, requires higher-order functions (functions as parameters). In addition, physical transparency facilitates bottom-up development—given a problem for which existing functionality is unsuitable, the developer first develops specialized access methods and operators, which can all be incorporated in a larger Jaql script. If warranted, these low-level operators can be packaged into functions and modules. Finally, if such functionality is more broadly applicable, the functionality can be integrated into Jaql by modifying the Jaql rewriter appropriately and, if need be, adding additional Jaql query language syntax. The rewriter and syntax are currently not extensible by anyone except the Jaql development team, but this is an interesting future direction to explore.

Jaql has been used in all of the patterns discussed in the introduction of this paper. For the ELT pattern, applications provide all of the scaffolding for metadata and simply use Jaql as a light-weight scripting engine. Because the data and transformation logic are often complex, errors are likely to arise for some small fraction of the data. We extended Jaql with sophisticated error-handling capability so that tolerance to such errors can be declaratively specified, leaving the appropriate adjustment of the evaluation plan to the rewriter [26]. As data becomes more structured, users often want to interact with the data using SQL. For this purpose, a SQL front-end to Jaql has been developed that compiles Jaql SQL to Jaql, thus using Jaql as a lightweight runtime.

Hadoop runtime improvements

FLEX scheduler

Originally, Hadoop employed first-in, first-out (FIFO) scheduling, but such simple schemes can result in suboptimal response time. The *Hadoop Fair Scheduler* (HFS) is a slot-based MapReduce scheme designed to ensure a degree of fairness among the jobs, by guaranteeing each job at least some minimum number of allocated slots. However, HFS does not attempt to actually optimize any specific scheduling metric. Most of our customer use cases involve concurrent execution of MapReduce jobs, and these Hadoop schedulers do not provide adequate ability to optimize concurrent job execution. Therefore, XAP includes a novel FLEX scheduler that is able to optimize a variety of metrics associated with individual jobs as well as entire Jaql scripts.

The goal of our FLEX algorithm is to optimize any of a variety of standard scheduling theory metrics while ensuring the same minimum job slot guarantees as in HFS, and ensuring maximum job-slot guarantees as well. The metrics can be chosen from a menu that includes response time, total schedule execution time (makespan), any of several metrics that reward or penalize job completion times compared with possible deadlines and service-level agreements (SLAs). The FLEX allocation scheduler can be

regarded as either a standalone plug-in replacement to the allocation scheme in HFS or as an add-on module that works synergistically with it. More details about FLEX have been published previously [18].

In order to optimize the schedule, we need to estimate the size of the jobs being scheduled. For the jobs that have already made some progress, we can extrapolate their total size from the already completed tasks. For the jobs not yet launched, we make predictions on the basis of prior invocations of the same job. To this end, we developed a technique that predicts the runtime performance for a fixed set of queries running over varying input datasets. Our prediction technique splits each query into several segments where the performance of each segment is estimated using machine-learning models. These per-segment estimates are used by a global analytical model to predict the overall query runtime. Our approach uses minimal statistics about the input datasets (e.g., tuple size and number of tuples), which are complemented with historical information about prior query executions (e.g., execution time). More details on the predictions module have been published previously [27].

Adaptive MapReduce

XAP includes new adaptive runtime techniques for MapReduce that improve performance and simplify job tuning. We implement these techniques by violating a key assumption of MapReduce that mappers run in isolation. Instead, our situation-aware mappers (SAMs) communicate through a distributed metadata store to obtain an aggregate view of the job state and to make globally coordinated optimization decisions. While implementing SAMs, we had to be careful to satisfy key MapReduce assumptions about scalability and fault tolerance, and not introduce noticeable performance overhead. For instance, SAMs can be executed in any order and re-executed at any time. We also avoid synchronization barriers and pay special attention to recovery from task failures in the critical path. We utilize SAMs to develop the set of techniques in the following paragraphs that make MapReduce more dynamic.

Adaptive mappers (AMs) dynamically change the granularity of checkpoints to trade off system performance, load balancing, and fault tolerance. In MapReduce, input data partition (split) size is a very important tuning parameter because having too few splits results in poor load balancing and decreased performance under faults, whereas with too many splits, the overhead of starting and checkpointing the tasks may dominate the running time of the job. In contrast, AMs make a decision after every split to either checkpoint or use another split and "stitch" it to one or more already processed ones. As a result, we obtain both reduced task startup overhead and dynamic load balancing.

Adaptive combiners (ACs) improve local aggregation by maintaining a cache of partial aggregates for the frequent keys. This technique was motivated by large group-by and aggregation queries that are often seen in our customers' workflows. ACs dramatically decrease the amount of data that needs to be sorted, shuffled, and merged for these queries, often improving their overall performance by as much as a factor of 3.

We often need an approximate histogram of map output keys. For instance, such a histogram is used in choosing cache size and replacement policy for ACs. Therefore, we implemented *adaptive sampling* (AS), which collects a sample of map output keys and aggregates them into a global histogram. During its initial sampling phase, every AM writes a subset of the output keys to a separate sample file and continuously updates the metadata store with whatever information is needed to determine whether a sufficient sample has been accumulated. The first mapper that detects that the stopping condition has been satisfied becomes the leader, collects all the sample files, and aggregates them into one histogram. AS utilizes AM to use the input splits in random order; thus, the histogram is equivalent to what a coarse block-level sampling would produce.

Another important use of the histogram produced by AS is *adaptive partitioning* (AP), which allows us to balance the partitioning of map outputs among the reducers. In particular, AP can produce ranges of map output keys with a roughly equal number of records. Such equal-sized range partitioning is required for the efficient parallel sort of map outputs. In the future, we plan to use AS outputs in other runtime optimization decisions, for individual Hadoop jobs (e.g., the join method) and Jaql queries (e.g., join order).

The flexible programming environment of Hadoop allowed us to implement SAMs and use them in adaptive techniques without any changes to Hadoop itself. Instead, the adaptive techniques are packaged as a library that can be used by Hadoop programmers through a simple API (application programming interface). To make the adaptive techniques completely "transparent" to the user, we integrated them into the Jaql query processor. (By "transparent," we mean that adaptive techniques are applied unbeknownst to the user, and the user does not know that adaptive techniques are used.) Our distributed metadata store uses Apache ZooKeeper, a scalable, fault-tolerant, transactional distributed coordination service. For more information about Adaptive MapReduce in XAP, please refer to [19].

Collocation

Hadoop has become an attractive platform for large-scale data analytics. However, there are still a number of limitations in Hadoop that affect performance. One of the limitations is its lack of ability to *collocate* related data on the same set of nodes. Collocation can be used to improve the efficiency of many operations, including indexing, grouping, aggregation, columnar storage, joins, and sessionization. To achieve collocation, we introduce

CoHadoop [17], a lightweight extension of Hadoop that allows applications to control where data is stored at the file-system level. CoHadoop extends HDFS with a new file-level property called a *locator* and modifies an HDFS data placement policy so that files with the same locator are placed on the same set of data nodes with *best effort*, including replicas. At the same time, CoHadoop also retains the benefits of Hadoop, including load balancing and fault tolerance. Through a detailed performance study of join and sessionization queries in the context of log processing, we observed that CoHadoop outperforms both plain Hadoop and previous work. In particular, compared with Hadoop, CoHadoop achieves a more than 13 times speedup for a sessionization query on 1.1 TB of log data. In addition, CoHadoop is more than 2 times faster for joins than an existing collocation approach, Hadoop++ [12], while also reducing the data loading time by more than half.

Column stores

Hadoop provides 1) the ability to use complex data types such as arrays, maps, and nested records, 2) the ability to write arbitrary map and reduce functions in a programming language instead of using a declarative query language, and 3) the choice of a widely accepted programming language, Java**, to express map and reduce functions. These features present a new challenge in designing a storage engine that traditional parallel DBMSs (database management systems) have not had to solve. For XAP, we developed several column-oriented storage and processing techniques specifically designed to support these features.

We identified performance challenges specific to complex data types in Hadoop, and we describe a novel skip list column format that enables lazy record construction, mostly inspired by the techniques used in column-oriented DBMSs [28]. A skip-list layout allows a reader to skip ahead a fixed number of variable-length rows in a file without having to interpret the intervening bytes. We also examined techniques that allow lazy decompression in Hadoop. This method can result in speedups of up to 1.5 times over an eager record construction strategy. It is important to emphasize that our column-oriented techniques make use of extensibility features that are already in Hadoop, so no modifications to the core of Hadoop are required. Moreover, these techniques do not require the use of a declarative query language and are designed to work even with hand-coded MapReduce jobs. In aggregate, our techniques can improve the performance of the map phase of a Hadoop job by as much as two orders of magnitude, and the overall job by more than one order of magnitude. Details of the design and implementation, as well as performance studies, are available at [16].

Learning from our experience with collocation, we ensured that the column-store format interacts with the replication policy of HDFS to co-locate column data. We demonstrated through experiments that Hadoop can make use of this storage format without incurring a large penalty for reconstructing records from the constituent columns.

Star-joins

XAP includes a subsystem called Clydesdale that shows dramatic performance improvements for structured data processing on an unmodified instance of Hadoop. Such a design also allows Clydesdale to inherit the fault-tolerance, elasticity, and scalability properties of MapReduce. Using the star schema benchmark [29], we showed that Clydesdale is 5 to 83 times faster than Apache Hive [6], an open-source SQL-based system.

The design of Clydesdale draws on several existing techniques from parallel DBMSs such as columnar storage, tailored n-way join plans, and block iteration (operating on a block of data at a time instead of a single row). However, adapting these techniques for the MapReduce environment is not straightforward, in particular when trying to preserve all the properties that make the platform attractive. The challenges arise from two important differences between a parallel DBMS and MapReduce: 1) the presence of a distributed file system and 2) the constraints of the task-scheduling infrastructure. Clydesdale uses the column-store format described in the previous section. The task-scheduling infrastructure in Hadoop was designed to provide locality-aware scheduling [1] at the granularity of map and reduce tasks. Map and reduce tasks are expected to last a short time (few minutes), and a typical job may consist of several map and reduce tasks. In contrast, DBMS runtimes have traditionally used iterator-based operators [30] that are scheduled to run for the entire duration of the query on a given server. Clydesdale uses carefully designed map tasks so that data structures used in query processing can be shared across multiple threads and even multiple tasks consecutively executed on any node. This allows Clydesdale to amortize the per-task overheads that can have an adverse impact on the performance of MapReduce jobs.

Clydesdale also exploits current hardware trends, such as servers with large memories and multiple cores and, as suggested, uses an appropriately tailored n-way join algorithm instead of repeated use of a generic two-way join. Clydesdale is aimed at workloads for which the data fits a star schema. Details of the design and implementation of Clydesdale were published previously [31].

Integration with data warehouses

With many of our customer engagements, we have observed that they use Hadoop-based solutions in conjunction with their data warehouses. In one particular scenario, one financial customer desires to run some targeted analysis on the customer's large transaction logs. The customer would like to offload the computation to a Hadoop cluster, but it

also needs to run their classical business intelligence tools, with which it is accustomed, on the results. A target account list is identified through some queries in the customer's warehouse. It would like to pass this list of account IDs to a Jaql script that will run the analysis. After Jaql writes its results into HDFS, the customer would like to generate reports using its business intelligence reporting tools and to accomplish this on a regular basis. For this purpose, the customer needed a solution driven by a database layer.

Another scenario that we often observe is the case in which customers desire to run a log analysis by enriching their log data with some reference data that is stored in the database. In some cases, the reference data changes slowly and can be replicated in HDFS, whereas in others the customer wants to keep the data in the database to handle updates.

To address the needs of our customers and provide some interoperability between Hadoop and the database, we developed connectors between XAP and DB2*. These connectors have been transferred to the IBM InfoSphere BigInsights platform. Next, we describe these features in detail.

Reading DB2 data in Jaql

Hadoop relies on an abstract InputFormat [2] interface to read data in parallel. While defining a MapReduce job, the InputFormat defines splits, which are abstract representations of data partitions. Each split is assigned to a map task. The task obtains an iterator over its split (data partition) from the InputFormat and passes it to the map function.

Hadoop provides *DBInputFormat* for reading data from relational databases, which relies on offset and limit to divide the query result among the mappers. In particular, every mapper uses a different start point (limit) to read *offset* number of tuples from the result set. This method creates a burden for the database manager, because the DBMS may have to run each independent query submitted by the mappers and return a different subset. This input format is not optimized for exploiting the database parallelism.

As a result, we created two different InputFormat interfaces to access DB2 data in parallel [32] with the goal of minimizing the burden on the DBMS and maximizing the utility of database parallelism. The *DB2DPFInputFormat* is specialized to load a table from DB2 with the Database Partitioning Feature (DPF)—the parallel, shared-nothing version of DB2. The InputFormat is configured with information to connect to the database as well as a table to read. Optionally, a predicate to filter rows and a project list to pick columns may also be provided. We use special features of DB2 DPF; namely, we create a split per DPF partition, where each split has information on its partition number.

This InputFormat can only read and consume partitioned tables (temporary or permanent). We also support non-DPF servers using the more general *DB2InputFormat*. This format moves the result of an arbitrary SQL query from DB2 (or any JDBC [Java Database Connectivity]-compliant database) into Hadoop. In addition to the query and connection information, the *DB2InputFormat* takes a sorted list of values, which explicitly define the ranges of values each mapper will process. A split is created for each consecutive pair of elements in this augmented list.

JaqlSubmit

JaqlSubmit is used to submit a Jaql query from DB2 to a Jaql server, which is an HTTP (Hypertext Transfer Protocol) gateway that is able to run Jaql jobs on Hadoop [32]. The *JaqlSubmit* user-defined function uses three input parameters: a Jaql script to run, an XML (Extensible Markup Language) document encoding the parameters to pass into the Jaql script, and the URL (Uniform Resource Locator) of the Jaql Server. *JaqlSubmit* itself does not have information on where the input data is coming from or where the output data is written to. It returns an XML document that contains the list of files containing the results of the Jaql script.

Ingesting HDFS data into DB2

HDFSRead is a DB2 user-defined table function that is used to read results generated by a Hadoop job back into the database. *HDFSRead* requires an XML handle as parameter, which can either be generated by a user or is returned by *JaqlSubmit*. This XML handle contains the list of files to read from HDFS and the output schema information of the data stored in HDFS.

HDFSRead is designed to read many files in parallel, as we expect to ingest into DB2 the output of a MapReduce job, where every reducer creates a separate file (or part) under the same directory.

Exporting DB2 data into Hadoop

The *HDFSWrite* function uses the HDFS client libraries to write database data directly into HDFS. It provides an alternative way to the JDBC-based InputFormats to copy DB2 data into Hadoop. Recall that the input format generates a separate SQL query for each mapper. On one hand, generating many mappers will result in a large number of SQL queries and a burden on the database, and on the other hand, generating too few mappers will limit the degree of parallelism on Hadoop. The *HDFSWrite* function circumvents these issues because each database logical partition directly outputs its data into a separate HDFS file under the same directory. The *HDFSWrite* function runs in parallel on all database partitions without any data communication between the partitions.

Support for analytics

Many of today's enterprises collect data at the most detailed level possible, thereby creating data repositories ranging from terabytes to petabytes in size. The knowledge buried in

these enormous datasets is invaluable for understanding and boosting business performance. The ability to apply sophisticated statistical analysis methods to this data can provide a significant competitive edge in the marketplace. For example, Internet companies such as Amazon or Netflix provide personalized recommendations of products to their customers, incorporating information about individual preferences. These recommendations increase customer satisfaction and thus play an important role in building, maintaining, and expanding a loyal customer base. Likewise, applications such as internet search and ranking, fraud detection, risk assessment, microtargeting, and ad placement gain significantly from fine-grained analytics at the level of individual entities. To this end, we have developed new techniques for advanced statistical analysis for huge amounts of data. There are many different analytic techniques, and there is a significant focus in IBM Research on scalable analytics, including [10]. Our work is complementary to this other work.

The workflow for a data analyst comprises multiple activities. Typically, the analyst first explores the data of interest, usually via visualization, sampling, and aggregation of the data into summary statistics. On the basis of this exploratory analysis, a model is built. The output of the model is itself explored—often through visualization and also through more formal validation procedures—to determine model adequacy. Multiple iterations of model building and evaluation may be needed before the analyst is satisfied. The final model is then used to improve business practices or support decision making. Feedback from model users can lead to further iterations of the model-development cycle. During this process, the traditional data analyst's indispensable toolkit is a statistical software package such as R, SPSS, SAS, or MATLAB**. Each of these packages provides a comprehensive environment for statistical computation, including a concise statistical language, well-tested libraries of statistical algorithms for data exploration and modeling, and visualization facilities.

However, most statistical software packages, including R, are designed to target the moderately sized datasets commonly found in other areas of statistical practice (e.g., opinion polls). These systems typically operate on a single server and entirely in main memory; they simply fail when the data becomes too large. Unfortunately, this means that data analysts are unable to work with these packages on massive datasets. Practitioners try to avoid this shortcoming either by exploiting vertical scalability—that is, using the most powerful machine available—or by working on only subsets or samples of the data. Both approaches have severe limitations: vertical scalability is inherently limited and expensive, and sampling methods may lose important features of individuals and of the tail of the data distribution. Our work in XAP focuses on overcoming these limitations. We describe XAP's two approaches for handling complex analytics in massive-scale data.

Ricardo

The first approach for handling the analysis is the novel Ricardo [15] analytics system, which rests on a decomposition of data-analysis algorithms into parts executed by the R statistical analysis system and parts handled by the Hadoop scalable data management system. Ricardo is named after David Ricardo, a famous economist of the early nineteenth century who studied conditions under which mutual trade is advantageous. Ricardo facilitates "trading" between R and Hadoop, with R sending aggregation-processing queries to Hadoop (written in the high-level Jaql query language), and Hadoop sending aggregated data to R for advanced statistical processing or visualization. Each trading partner performs the tasks that it does best. The decomposition attempts to minimize the transfer of data across system boundaries. Ricardo allows analysts to work on huge datasets from within a popular, well-supported, and powerful analysis environment. Because this approach avoids the need to re-implement either statistical or data management functionality, it provides a quick path to scalable analytics. Ricardo can be applied to a wide range of analyses of massive data, including time-series analysis, regression with outlier detection, principal component analysis, generalized linear models, clustering, and so on. Indeed, this work was strongly motivated by the example provided in Pattern 1 to explore approaches to integrate the functionality of R and Hadoop.

Ricardo is inspired by the work in [11], which shows that many deep analytical problems can be decomposed into a "small-data part" and a "large-data part." In Ricardo, the small-data part is executed in R, and the large-data part is executed in Hadoop/Jaql. For example, for principal component analysis, Hadoop/Jaql first constructs the empirical covariance matrix, and, next, R performs an eigenvector decomposition of the covariance matrix. A key requirement for the success of this combined approach is that the amount of data that must be communicated between both systems be sufficiently small. Fortunately, this requirement holds for almost all of the deep analytics mentioned above.

Ricardo facilitates some key tasks in an analyst's typical workflow: data exploration, model building, and model evaluation, all with respect to a very large dataset. For illustrative purposes in [15], we use the dataset provided for the Netflix movie-recommendation competition. Although the competition itself was based on a subset of just 100 million movie ratings, our experiments on a Hadoop cluster in the Amazon Elastic Compute Cloud (Amazon EC2) indicate that Ricardo can scale the functionality of R to handle the billions of ratings found in practice—more than a terabyte of data in our case.

Matrix factorizations

The second approach is specialized for the popular family of matrix-factorization-based analyses and rests on stochastic gradient descent (SGD), an iterative stochastic optimization algorithm. Low-rank matrix factorization has received much attention in recent years, since it is fundamental to a variety of mining tasks that are increasingly being applied to massive datasets. Specifically, low-rank matrix factorizations are effective tools for analyzing "dyadic data" in order to discover and quantify the interactions between two given entities. Successful applications include topic detection and keyword search (where the corresponding entities are documents and terms), news personalization (users and stories), and recommendation systems (users and items). In large applications, these problems can involve matrices with millions of rows (e.g., distinct customers), millions of columns (e.g., distinct items), and billions of entries (e.g., transactions between customers and items). At such massive scales, distributed algorithms for matrix factorization are essential to achieving reasonable performance. Low-rank matrix factorization approximations are employed because, in practice, exact factorization is generally neither possible nor desired. Low-rank approximation algorithms attempt to minimize a "loss function" that measures the discrepancy between the original input matrix and product of the factors returned by the algorithm.

We started with SGD, an iterative optimization algorithm, because it has been shown, in a sequential setting, to be very effective for matrix factorization. Although the generic SGD algorithm is not embarrassingly parallel and, hence, cannot directly scale to very large data, we can exploit the special structure of the factorization problem to obtain a version of SGD that is fully distributed and scales to extremely large matrices. We first developed a novel stratified SGD variant (SSGD) that operates repeatedly on a single piece ("stratum") of the data at a time (unlike SGD, which operates on all the data at a time). SSGD applies to general loss-minimization problems in which the loss function can be expressed as a weighted sum of the local losses in each stratum ("stratum losses"). A special version of the SSGD algorithm is created to obtain a new matrix-factorization algorithm, called DSGD [33], that can be fully distributed and run on massive datasets using Hadoop or other parallel processing frameworks. The specialized DSGD algorithm, though narrower in scope than Ricardo, improves upon the performance of Ricardo for the factorization problem.

SSGD [33] is applicable to general loss-minimization problems in which the overall loss can be expressed as a weighted sum of stratum losses. At each iteration, the algorithm takes a downhill step with respect to one of the stratum losses, i.e., approximately in the direction of the negative gradient of the stratum loss. Although each such direction is "wrong" with respect to minimization of the overall loss, we prove that under appropriate regularity conditions, SSGD will converge to a good solution for the overall loss if the sequence of strata is chosen carefully. The proof rests on stochastic approximation theory and regenerative process theory.

As mention, we then created a special version of SSGD to obtain a novel distributed matrix-factorization algorithm, called DSGD [33]. Specifically, we express the input matrix as a union of strata, with a particular block-diagonal shape. For each stratum, the stratum loss is defined as the loss computed over only the data points in the stratum (and appropriately scaled). The DSGD algorithm repeatedly selects a stratum according to the general SSGD procedure and processes the stratum in a distributed fashion. Importantly, both matrix and factors are fully distributed, so that DSGD has low memory requirements and scales to matrices with millions of rows, millions of columns, and billions of nonzero elements. When DSGD is implemented in MapReduce and compared with state-of-the-art distributed algorithms for matrix factorization, our experiments demonstrate that DSGD converges orders of magnitude faster, and has almost linear scale-out behavior.

Unlike many prior algorithms, DSGD is a generic algorithm in that it can be used for a variety of different loss functions. Of particular interest is the class of factorizations that minimize a "nonzero loss" in which a zero represents missing data and hence should be ignored when computing loss. A typical motivation for factorization in this setting is to estimate the missing values, e.g., the rating that a customer would likely give to a previously unseen movie.

Conclusion and future plans

In this paper, we described XAP, a powerful large-scale data processing system that is built upon Hadoop. At the core, Jaql provides the means to create both analytical and ELT flows on XAP. It is highly extensible through modules and provides features to deal with big in situ data with heterogeneous schemas. Having observed the need for efficient structured data processing on Hadoop and by building on our experience with relational databases, we provide various extensions and optimizations, including column store, star joins, and collocation. To address the need for integration with other systems in the enterprise, XAP provides warehouse connectors as well as bridges to analytical tools such as R. XAP also comes with analytical libraries to enable easy development of analytical workflows on Hadoop. Many of these technologies have been transferred to the IBM InfoSphere BigInsights product. In the future, we plan to extend our analytical libraries, integrate with more tools in the enterprise, and further improve the efficiency of jobs running on Hadoop.

*Trademark, service mark, or registered trademark of International Business Machines Corporation in the United States, other countries, or both.

**Trademark, service mark, or registered trademark of Apache Software Foundation, Sun Microsystems, MathWorks, Inc., Linus Torvalds, The Open Group, or Microsoft Corporation in the United States, other countries, or both.

References

1. J. Dean and S. Ghemawat, "MapReduce, simplified data processing on large clusters," in *Proc. USENIX Symp. OSDI*, 2004, pp. 137–150.
2. The Apache Software Foundation. Hadoop. [Online]. Available: http://hadoop.apache.org
3. M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *Proc. EuroSys*, Lisbon, Portugal, 2007, pp. 59–72.
4. R. Pike, S. Dorward, R. Griesemer, and S. Quinlan, "Interpreting the data: Parallel analysis with Sawzall," *Sci. Program. J.—Dynamic Grids Worldwide Comput.*, vol. 13, no. 4, pp. 277–298, 2005.
5. Y. Yu, M. Isard, D. Fetterly, M. Budiu, Ú. Erlingsson, P. Gunda, and J. Currey, "DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language," in *Proc. OSDI*, 2008, pp. 1–14.
6. A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive—A warehousing solution over a map-reduce framework," *Proc. PVLDB*, vol. 2, no. 2, pp. 1626–1629, 2009.
7. C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig latin: A not-so-foreign language for data processing," in *Proc. ACM SIGMOD*, 2008, pp. 1099–1110.
8. S. Balakrishnan, V. Chu, M. A. Hernandez, H. Ho, R. Krishnamurthy, S. Liu, J. H. Pieper, J. S. Pierce, L. Popa, C. M. Robson, L. Shi, I. Stanoi, E. L. Ting, S. Vaithyanathan, and H. Yang, "Midas: Integrating public financial data," in *Proc. ACM SIGMOD*, Indianapolis, IN, Jun. 6–11, 2010, pp. 1187–1190.
9. R. Krishnamurthy, Y. Li, S. Raghavan, F. Reiss, S. Vaithyanathan, and H. Zhu, "SystemT: A system for declarative information extraction," *ACM SIGMOD Rec.*, vol. 37, no. 4, pp. 7–13, Dec. 2008.
10. A. Ghoting, R. Krishnamurthy, E. Pednault, B. Reinwald, V. Sindhwani, S. Tatikonda, Y. Tian, and S. Vaithyanathan, "SystemML: Declarative machine learning on MapReduce," in *Proc. ICDE*, 2011, pp. 231–242.
11. C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. R. Bradski, A. Y. Ng, and K. Olukotun, "Map-reduce for machine learning on multicore," in *Proc. Neural Inf. Process. Syst.*, 2006, pp. 281–288.
12. J. Dittrich, J. Quiané-Ruiz, A. Jindal, Y. Kargin, V. Setty, and J. Schad, "Hadoop++: Making a yellow elephant run like a Cheetah (without it even noticing)," *Proc. PVLDB*, vol. 3, no. 1/2, pp. 515–529, Sep. 2010.
13. A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and A. Rasin, "HadoopDB: An architectural hybrid of MapReduce and DBMS technologies for analytical workloads," *Proc. PVLDB*, vol. 2, no. 1, pp. 922–933, Aug. 2009.
14. IBM InfoSphere BigInsights. [Online]. Available: http://www.ibm.com/software/data/infosphere/biginsights/
15. S. Das, Y. Sismanis, K. S. Beyer, R. Gemulla, P. J. Haas, and J. McPherson, "Ricardo: Integrating R and Hadoop," in *Proc. ACM SIGMOD*, 2010, pp. 987–998.
16. A. Floratou, J. M. Patel, E. J. Shekita, and S. Tata, "Column-oriented storage techniques for MapReduce," *Proc. PVLDB*, vol. 4, no. 7, pp. 419–429, Apr. 2011.
17. M. Y. Eltabakh, Y. Tian, F. Özcan, R. Gemulla, A. Krettek, and J. McPherson, "CoHadoop: Flexible data placement and its exploitation in Hadoop," *Proc. PVLDB*, vol. 4, no. 9, pp. 575–585, Jun. 2011.
18. J. Wolf, D. Rajan, K. Hildrum, R. Khandekar, V. Kumar, S. Parekh, K. Wu, and A. Balmin, "FLEX: A slot allocation scheduling optimizer for MapReduce workloads," in *Proc. Middleware*, 2010, vol. 6452, pp. 1–20, Lecture Notes in Computer Science.
19. R. Vernica, A. Balmin, K. S. Beyer, and V. Ercegovac, "Adaptive MapReduce using situation-aware mappers," in *Proc. Int. Conf. EDBT*, Berlin, Germany, Mar. 26–30, 2012, pp. 420–431.
20. IBM Corporation, Cognos Consumer Insight. [Online]. Available: http://www-01.ibm.com/software/analytics/cognos/analytic-applications/consumer-insight/
21. K. S. Beyer, V. Ercegovac, R. Krishnamurthy, S. Raghavan, J. Rao, F. Reiss, E. J. Shekita, D. E. Simmen, S. Tata, S. Vaithyanathan, and H. Zhu, "Towards a scalable enterprise content analytics platform," *IEEE Data Eng. Bull.*, vol. 32, no. 1, pp. 28–35, Mar. 2009.
22. HDFS (Hadoop Distributed File System). [Online]. Available: http://hadoop.apache.org/hdfs
23. GPFS. [Online]. Available: http://www.ibm.com/systems/software/gpfs/
24. K. S. Beyer, V. Ercegovac, R. Gemulla, A. Balmin, M. Eltabakh, C.-C. Kanne, F. Özcan, and E. Shekita, "Jaql: A scripting language for large scale semistructured data analysis," *Proc. PVLDB*, vol. 4, no. 12, pp. 1272–1283, 2011.
25. JSON. [Online]. Available: http://www.json.org
26. C. Kanne and V. Ercegovac, "Declarative error management for robust data-intensive applications," in *Proc ACM SIGMOD*, Scottsdale, AZ, May 20–24, 2012, pp. 205–216.
27. A. Popescu, V. Ercegovac, A. Balmin, M. Branco, and A. Ailamaki, "Same queries, different data: Can we predict runtime performance?" in *Proc. Int. Workshop Self Manag. Database Syst.*, Washington, DC, 2012.
28. D. Abadi, S. Myers, D. J. DeWitt, and S. Madden, "Materialization strategies in a column-oriented DBMS," in *Proc. ICDE*, 2007, pp. 466–475.
29. P. E. O'Neil, E. O'Neil, and X. Chen, *The Star Schema Benchmark (SSB)*. [Online]. Available: http://www.cs.umb.edu/~poneil/StarSchemaB.PDF
30. G. Graefe, "Volcano—An extensible and parallel query evaluation system," *IEEE Trans. Knowl. Data Eng.*, vol. 6, no. 1, pp. 120–135, Feb. 1994.
31. T. Kaldewey, E. J. Shekita, and S. Tata, "Clydesdale: Structured data processing on MapReduce," in *Proc. Int. Conf. EDBT*, 2012, pp. 15–25.
32. F. Özcan, D. Hoa, K. S. Beyer, A. Balmin, C. J. Liu, and Y. Li, "Emerging trends in the enterprise data analytics: Connecting Hadoop and DB2 warehouse," in *Proc. ACM SIGMOD Conf.*, 2011, pp. 1161–1164.
33. R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis, "Large-scale Matrix: Factorization with distributed stochastic gradient descent," in *Proc ACM KDD*, 2011, pp. 69–77.

Received July 22, 2012; accepted for publication August 21, 2012

Andrey Balmin *IBM Research Division, Almaden Research Center, San Jose, CA 95120 USA (abalmin@us.ibm.com).*

Dr. Balmin joined IBM Research - Almaden as a Research Staff Member after receiving his Ph.D. degree in computer science from the University California, San Diego. His areas of expertise include dynamic scheduling and optimization in Big Data platforms, as well as search and query processing of semi-structured and graph-structured data.

Kevin Beyer *Platfora, Inc., San Mateo, CA 94401 USA (kevin@platfora.com).*

Dr. Beyer is Principal Architect at Platfora, Inc. He earned a Ph.D. degree from the University of Wisconsin–Madison. Dr. Beyer initiated the Jaql project while he was a Research Staff Member at the IBM Almaden Research Center.

Vuk Ercegovac *IBM Research Division, Almaden Research Center, San Jose, CA 95120 USA (vercego@us.ibm.com).*
Dr. Ercegovac is a Research Staff Member Manager in the Computer Science department at IBM Research - Almaden. He received a B.S. degree in electrical engineering and computer science from the University of California, Berkeley, in 1997, and M.S. and Ph.D. degrees in computer science from the University of Wisconsin–Madison in 2000 and 2006, respectively. At the IBM Almaden Research Center, he has worked on search indexing and core Jaql infrastructure as well as its use in applications. He is an author or coauthor of 4 patents and 17 technical papers. Dr. Ercegovac is a member of the Association for Computing Machinery (ACM).

John McPherson *IBM Research Division, Almaden Research Center, San Jose, CA 95120 USA (jmcphers@us.ibm.com).*
Dr. McPherson is an IBM Distinguished Engineer and Manager of Data Intensive Analytics at IBM Research - Almaden. In the past, he held technical leadership and senior management positions in exploratory database systems research, DB2 development, business intelligence development, and enterprise search development. He received a joint Ph.D. degree from the University of Wisconsin–Madison in electrical and computer engineering as well as computer sciences.

Fatma Özcan *IBM Research Division, Almaden Research Center, San Jose, CA 95120 USA (fozcan@us.ibm.com).* Dr. Özcan is a Research Staff Member at IBM Research - Almaden. Her current research focuses on platforms and infrastructure for large-scale data analysis, Hadoop and database integration, and query optimization for semi-structured data. She received her Ph.D. degree in computer science from University of Maryland, College Park, in 2001, and her B.Sc. and M.Sc. degrees in computer engineering from Middle East Technical University, Ankara, in 1994 and 1996, respectively. Dr. Özcan is the coauthor of the book *Heterogeneous Agent Systems*, and coauthor of more than 35 articles and conference papers. She has chaired program committees for various conferences and served for the National Science Foundation (NSF). She is a member of the Association for Computing Machinery (ACM).

Hamid Pirahesh *IBM Research Division, Almaden Research Center, San Jose, CA 95120 USA (pirahesh@us.ibm.com).*
Dr. Pirahesh is an IBM Fellow, Association for Computing Machinery (ACM) Fellow, and a senior manager, responsible for the exploratory database department at IBM Research - Almaden in San Jose, California. He also has direct responsibilities for various aspects of IBM information management products. He has served as an associate editor of *ACM Computing Surveys* and has served on the program committee of major computer conferences. Dr. Pirahesh is also a member of the IBM Academy. His current focus is Big Data analytics on highly scalable servers. Dr. Pirahesh was a principal member of the original team that designed the query processing architecture of the IBM DB2 LUW (Linux**, Unix**, Windows**) relational DBMS and delivered the product to the marketplace. He has made major contributions to query language industry standards. His research areas include cloud computing, OLAP (online analytical processing) and aggregate data management, query optimization, data warehousing, Big Data systems, and the management of semi-structured and unstructured data, including NoSQL databases. He also serves as a consultant to various IBM divisions, including the Software division and IBM Global Services.

Eugene Shekita *Google, Inc., Mountain View, CA 94043 USA (shekita@google.com)*. Dr. Shekita has authored more than 45 research papers and 40 patents in the field of databases and Big Data. He is currently a Senior Staff Engineer at Google. Prior to Google, he was a researcher and manager at the IBM Almaden Research Center.

Yannis Sismanis *IBM Research Division, Almaden Research Center, San Jose, CA 95120 USA (syannis@us.ibm.com).* Dr. Sismanis received his Ph.D. degree in computer science from the University of Maryland, College Park. He is a Research Scientist at IBM Research - Almaden. His research focuses on massively parallel processing for big data, real-time business intelligence, structured and unstructured analysis, and information retrieval. He has published more than 25 papers in peer-reviewed journals, conferences, and workshops and holds numerous patents. Since 2004, he has served on more than 30 program committees of international conferences and workshops.

Sandeep Tata *IBM Research Division, Almaden Research Center, San Jose, CA 95120 USA (stata@us.ibm.com).* Dr. Tata received his Ph.D. degree from the University of Michigan in 2007 in computer science and engineering where his dissertation focused on declarative querying for large sequence datasets. He joined the Exploratory Data Management Systems Group at IBM Research - Almaden in 2007. His research interests are at the intersection of scale-out systems and data management.

Yuanyuan Tian *IBM Research Division, Almaden Research Center, San Jose, CA 95120 USA (ytian@us.ibm.com).* Dr. Tian is a Research Staff Member at IBM Research - Almaden. She received her Ph.D. degree in computer science and engineering from the University of Michigan in 2008. Her current research interests include large-scale systems for machine learning and graph analysis. Dr. Tian is a member of the Association for Computing Machinery (ACM) and the Institute of Electrical and Electronics Engineers (IEEE). She is the recipient of a Distinguished Achievement Award from the University of Michigan in 2008 for her research and academic accomplishments.