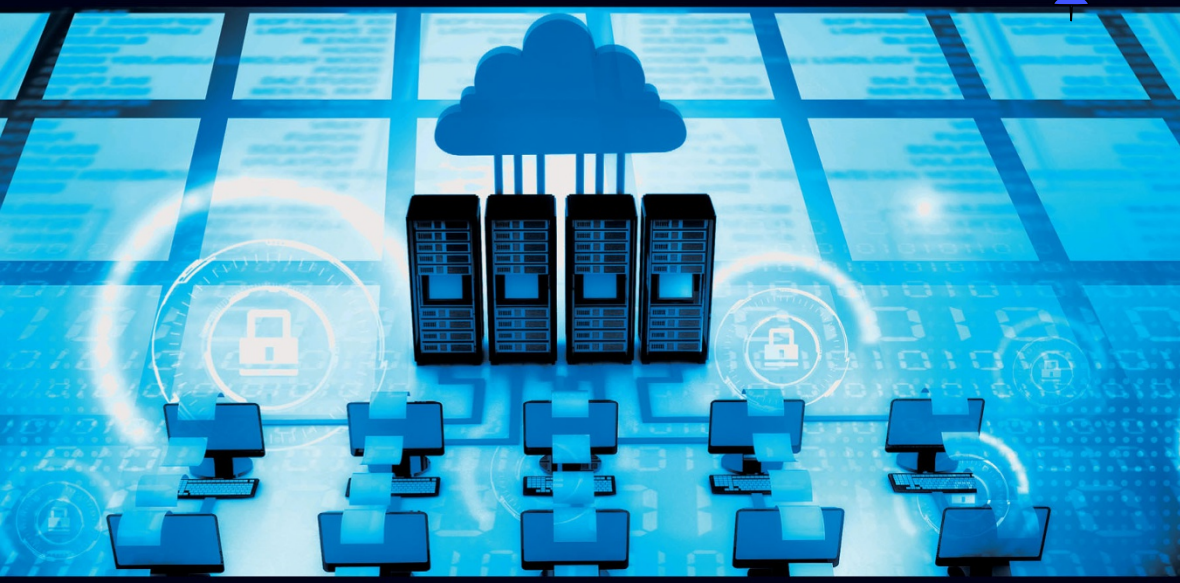
COMPUTER ENGINEERING SERIES

DATABASES AND BIG DATA SET



Volume 1

NoSQL Data Models

Trends and Challenges

Edited by
Olivier Pivert

ISTE

WILEY

NoSQL Data Models

Databases and Big Data Set

coordinated by
Dominique Laurent and Anne Laurent

Volume 1

NoSQL Data Models

Trends and Challenges

Edited by

Olivier Pivert

ISTE

WILEY

First published 2018 in Great Britain and the United States by ISTE Ltd and John Wiley & Sons, Inc.

ISTE Ltd
27-37 St George's Road
London SW19 4EU
UK

www.iste.co.uk

John Wiley & Sons, Inc.
111 River Street
Hoboken, NJ 07030
USA

www.wiley.com

Library of Congress Control Number: 2018941384

British Library Cataloguing-in-Publication Data
A CIP record for this book is available from the British Library
ISBN 978-1-78630-364-6

Contents

Foreword

This volume is part of a series entitled *Database and Big Data*, or DB & BD for short, whose content is motivated by the radical and rapid evolution (not to say *revolution*) of database systems during the last decade.

Indeed, since the 1970s, inspired by the relational database model, many research topics have emerged in the database community, such as, just to cite a few, Deductive Databases, Object-Oriented Databases, Semi-Structured Databases, Resource Description Framework (RDF), Open Data, Linked Data, Data Warehouses, Data Mining, and more recently, Cloud Computing, NoSQL and Big Data. Currently, the last three issues are becoming increasingly important and attract the most research efforts in the domain of databases.

Consequently, considering that Big Data environments are now to be handled in most current applications, the goal of this series is to address some of the latest issues in such environments. By doing so, while reporting on specific recent research results, we aim to provide readers with evidence that database technology is significantly changing, so as to face important challenges encountered in the majority of these applications.

More precisely, although relational databases are still commonly used in traditional applications, it is clear that most current Big Data applications cannot be handled by Relational DataBase Management Systems (RDBMSs), mainly because of the following reasons:

– there is a strong need to consider heterogeneous, structured, semi-structured or even unstructured data, for which no common schema exists.

RBMSs are not flexible enough to handle such variety of data, because these database systems were designed for handling tabular data;

– efficiency when facing Big Data in a distributed and replicated environment is now a key issue that RDBMSs fail to achieve, in particular when it comes to combining large tables.

New database systems have been proposed during the past few years, which are known under the generic term *NoSQL Databases*. These systems aim to solve the previous two points, and all claim to achieve their goal.

However, these systems need to be investigated further, because some important issues remain open (semantics of data, constraint satisfaction, transaction processing, privacy preservation, optimization, etc.). The volumes of this series aim to address some of these challenging issues and to present some of the most recent research results in this field.

Considering that the numerous currently available proposals are based on various concepts and data models (column-based, text-based, graph or hyper graph-based), this volume addresses issues related to trends and challenges related to NoSQL data models.

Anne LAURENT
Dominique LAURENT

Preface

As is well known, a major event in the field of data management was the introduction of the relational model by Codd in the early 1970s, which laid the foundations for a genuine theory of databases. After a somewhat slow start, due to the important Research and Development effort necessary to define efficient systems, relational database management systems reigned supreme for several decades.

However, around the end of the 20th Century, several phenomena modified the data management landscape. First, new types of applications in several domains were introduced to handle data for which the relational model appeared inadequate or inefficient. Typical examples are *semi-structured data* on the one hand, and *graphs* on the other (social networks, bibliographic databases, cartographic databases, genomic data, etc.) for which specific models and systems had to be designed. Second, a major event was the rise of the Semantic Web whose aim is, according to the W3C, to "provide a common framework that allows data to be shared and reused across application, enterprise and community boundaries". The Semantic Web uses models and languages specifically designed for linked data, which facilitate automated reasoning on such data. Besides, the amount of useful data in some application domains has become so huge that it cannot be stored or processed by traditional database solutions. This latter phenomenon is commonly referred to as *Big Data*. In terms of database technology, as a response to these new needs, we have seen the appearance of what have come to be called *NoSQL databases*.

The term NoSQL was coined by Carlo Strozzi in 1998, who designed a relational database system without SQL implementation and named it Strozzi NoSQL. However, this system is distinct from the circa-2009 general concept of NoSQL databases, which are typically non-relational. Many data models have been proposed: key-value stores, document stores (key-value stores that restrict values to semi-structured formats such as JSON), wide column stores, RDF, graph databases, XML, etc[1].

While the management of large volumes of data has always been subject to many research efforts, recent results in both the distributed systems and database communities have led to an important renewal of interest in this topic. Large scale distributed file systems such as Google File System[2] and parallel processing paradigm/environments such as MapReduce[3] have been the foundation of a new ecosystem with data management contributions in major database conferences and journals. Different (often open-source) systems have been released, such as Pig[4], Hive[5] or, more recently, Spark[6] and Flink[7], making it easier to use data center resources to manage Big Data. However, many research challenges remain, related, for instance, to system efficiency, and query language expressiveness and flexibility.

This book presents a sample of recent works by French research teams active in this domain. As the reader will see, it covers various aspects of NoSQL research, from semantic data management to graph databases, as well

1 GESSERT F., WOLFRAM W. FRIEDRICH S., "NoSQL database systems: a survey and decision guidance", *Computer Science - R&D*, vol. 32, nos 3–4, pp. 353–365, 2017.

2 GHEMAWAT S., GOBIOFF H., LEUNG S.-T., *Proceedings of the Symposium on Operating Systems Principles (SOSP)*, Bolton Landing, USA, pp. 29–43, 2003.

3 DEAN J., GHEMAWAT S., "MapReduce: simplified data processing on large clusters", *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

4 OLSTON C., REED B., SRIVASTAVA U. *et al.*, "Pig latin: a not-so-foreign language for data processing", *Proceedings of the SIGMOD International Conference on Management of Data*, Vancouver, Canada, pp. 1099–1110, 2008.

5 THUSOO A., SARMA J.S., JAIN N. *et al.*, "Hive – a petabyte scale data warehouse using Hadoop", *Proceedings of the International Conference on Data Engineering (ICDE)*, Long Beach, USA, pp. 996–1005, 2010.

6 ZAHARIA M., CHOWDHURY M., DAS T. *et al.*, "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing", *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, San Jose, USA, pp. 15–28, 2012.

7 CARBONE P., KATSIFODIMOS A., EWEN S. *et al.*, "Apache Flink: Stream and Batch Processing in a Single Engine", *IEEE Data Engineering Bulletin*, vol. 4, pp. 28–38, 2015.

as Big Data management in cloud environments, dealing with data models, query languages and implementation issues. The book is organized as follows:

Chapter 1, by Kim Nguyễn, from LRI and the University of Paris-Sud, presents an overview of NoSQL languages and systems. The author highlights some of the technical aspects of NoSQL systems (in particular, distributed computation with MapReduce) before discussing current research trends: join implementations on top of MapReduce, models for NoSQL languages and systems, and the perspective that consists of defining a formal model of NoSQL databases and queries.

Chapter 2, entitled "Distributed SPARQL Query Processing: A Case Study with Apache SPARK", by Bernd Amann, Olivier Curé and Hubert Naacke, from the LIP6 laboratory in Paris, is devoted to the issue of evaluating SPARQL queries over large RDF datasets. The authors present a solution that consists of using the MapReduce framework to process SPARQL graph patterns and show how the general purpose cluster computing platform Apache Spark can be used to this end. They emphasize the importance of the physical data layer for query evaluation efficiency and show that hybrid query plans combining partitioned and broadcast joins improve query performances in almost all cases.

Chapter 3, authored by Manel Achichi, Mohamed Ben Ellefi, Zohra Bellahsene and Konstantin Todorov, from the LIRMM laboratory in Montpellier, is entitled "Doing Web Data: From Dataset Recommendation to Data Linking". It deals with the production of web data and focuses on the data linking stage, seen as an operation which generates a set of links between two different datasets. The authors first study the prior task which consists of discovering relevant datasets leading to the identification of similar resources to support the data linking issue. They provide an overview of recommendation approaches for candidate datasets, then present and classify the different techniques that are applied by the currently available data linking tools. The main challenge faced by all of these techniques is to overcome different heterogeneity problems that may occur between the considered datasets, such as differences in descriptions at different levels (value, ontological or logical) in order to compare the resources efficiently, and the authors show that further research efforts are still needed to better cope with these heterogeneity issues.

Chapter 4, entitled "Big Data Integration in Cloud Environments: Requirements, Solutions and Challenges", by Rami Sellami and Bruno Defude, from CETIC Charleroi and Telecom SudParis respectively, presents and discusses the requirements of Big Data integration in cloud environments. In such a context, applications may need to interact with several heterogeneous data stores, depending on the types of data they have to manage (traditional data, documents, graph data from social networks, simple key-value data, etc.). A first constraint is that, to make these interactions possible, programmers have to be familiar with different APIs. A second difficulty is that the execution of complex queries over heterogeneous data models cannot currently be achieved in a declarative way and therefore requires extra implementation efforts. Moreover, cloud discovery as well as application deployment and execution are generally performed manually by programmers. The authors analyze and discuss the current state-of-the-art regarding four requirements (automatic data stores selection and discovery, unique access for all data stores, transparent access for all data stores, global query processing and optimization), provide a global synthesis according to three groups of criteria, and highlight important challenges that remain to be tackled.

Chapter 5 is authored by Vijay Ingalalli, Dino Ienco and Pascal Poncelet, from the LIRMM laboratory in Montpellier, and is entitled "Querying RDF Data: A Multigraph-based Approach". In this chapter, the authors cope with two challenges faced by the RDF data management community: first, automatically generated queries cannot be bounded in their structural complexity and size; second, the queries generated by retrieval systems (or any other application) need to be efficiently answered in a reasonable amount of time. In order to address these challenges, the authors advocate an approach to RDF query processing that involves two steps: an offline step where the RDF database is transformed into a multigraph and indexed, and an online step where the SPARQL query is transformed into a multigraph too, which makes query processing boil down to a subgraph homomorphism problem. An RDF query engine based on this strategy is presented, named AMbER, which exploits structural properties of the multigraph query as well as the indices previously built on the multigraph structure.

Chapter 6 is entitled "Fuzzy Preference Queries to NoSQL Graph Databases" and is authored by Arnaud Castelltort, Anne Laurent, Olivier Pivert, Olfa Slama and Virginie Thion; the first two authors being affiliated to

the LIRMM laboratory in Montpellier, and the last three authors to the IRISA laboratory in Lannion. This chapter deals with flexible querying of graph databases that may involve gradual relationships. The authors first introduce an extension of attributed graphs where edges may represent a fuzzy concept (such as *friend* in the case of a social network, or *co-author* in the case of a bibliographic database). Then, they describe an extension of the query language Cypher that makes it possible to express fuzzy requirements, both on attribute values and on structural aspects of the graph (such as the length or the strength of a path). Finally, they deal with implementation issues and outline a query processing strategy based on the derivation of a regular Cypher query from the fuzzy query to be evaluated, through an add-on built on top of a classical graph database management system.

Finally, Chapter 7, by Cédric du Mouza and Nicolas Travers, from CNAM Paris, is entitled "Relevant Filtering in a Distributed Content-based Publish/Subscribe System", and deals with textual data management. More precisely, it considers a crucial challenge faced by Publish/Subscribe systems, which is to efficiently filter feeds' information in real time. Publish/Subscribe systems make it possible to subscribe to flows of items coming from diverse sources and notify the users according to their interests, but the existing systems hardly address the issue of *document relevance*. However, numerous sources may provide similar information, or a new piece of information may be "hidden" in a large flow. The authors introduce a real-time filtering process based on relevance that notably integrates the notions of *novelty* and *diversity*, and they show how this filtering process can be efficiently implemented in a NoSQL environment.

Olivier PIVERT
May 2018

1

NoSQL Languages and Systems

1.1. Introduction

1.1.1. *The rise of NoSQL systems and languages*

Managing, querying and making sense of data have become major aspects of our society. In the past 40 years, advances in technology have allowed computer systems to store vast amounts of data. For the better part of this period, relational database management systems (RDBMS) have reigned supreme, almost unchallenged, in their role of sole keepers of our data sets. RDBMS owe their success to several key factors. First, they stand on very solid theoretical foundations, namely the relational algebra introduced by Edgar F. Codd [COD 70], which gave a clear framework to express, in rigorous terms, the limit of systems, their soundness and even their efficiency. Second, RDBMS used one of the most natural representations to model data: tables. Indeed, tables of various sorts have been used since antiquity to represent scales, account ledgers, and so on. Third, a domain-specific language, SQL, was introduced almost immediately to relieve the database *user* from the burden of low-level programming. Its syntax was designed to be close to natural language, already highlighting an important aspect of data manipulation: people who can best make sense of data are not necessarily computer experts, and vice versa. Finally, in sharp contrast to the high level of data presentation and programming interface, RDBMS have always thrived to

Chapter written by Kim NGUYỄN.

NoSQL Data Models: Trends and Challenges, First Edition. Edited by Olivier Pivert.

offer the best possible performances for a given piece of hardware, while at the same time ensuring consistency of the stored data *at all times*.

At the turn of the year 2000 with the advances in high speed and mobile networks, and the increase in storage and computing capacity, the amount of data produced by humans became massive, and new usages were discovered that were impractical previously. This increase in both data volumes and computing power gave rise to two distinct but related concepts: "Cloud Computing" and "Big Data". Broadly speaking, the Cloud Computing paradigm consists of having data *processing* performed remotely in data centers (which collectively form the so-called cloud) and having end-user devices serve as terminals for information display and input. Data is accessed on demand and continuously updated. The umbrella term "Big Data" characterizes data sets with the so-called three "V"s [LAN 01]: Volume, Variety and Velocity. More precisely, "Big Data" data sets must be large (at least several terabytes), heterogeneous (containing both structured and unstructured textual data, as well as media files), and produced and processed at high speed. The concepts of both Cloud Computing and Big Data intermingle. The sheer size of the data sets requires some form of distribution (at least at the architecture if not at the logical level), preventing it from being stored close to the end-user. Having data stored remotely in a distributed fashion means the only realistic way of extracting information from it is to execute computation close to the data (i.e. remotely) to only retrieve the fraction that is relevant to the end-user. Finally, the ubiquity of literally billions of connected end-points that continuously capture various inputs feed the ever growing data sets.

In this setting, RDBMS, which were the be-all and end-all of data management, could not cope with these new usages. In particular, the so-called ACID (Atomicity, Consistency, Isolation and Durability) properties enjoyed by RDBMS transactions since their inception (IBM Information Management System already supported ACID transactions in 1973) proved too great a burden in the context of massively distributed and frequently updated data sets, and therefore more and more data started to be stored outside of RDBMS, in massively distributed systems. In order to scale, these systems traded the ACID properties for performance. A milestone in this area was the MapReduce paradigm introduced by Google engineers in 2004 [DEA 04]. This programming model consists of decomposing a high-level data operation into two phases, namely the *map* phase where the data is transformed locally on each node of the distributed system where it resides,

and the *reduce* phase where the outputs of the map phase are exchanged and migrated between nodes according to a partition key – all groups with the same key being migrated to the same (set of) nodes – and where an aggregation of the group is performed. Interestingly, such low-level operations where known both from the functional programming language community (usually under the name *map* and *fold*) and from the database community where the map phase can be used to implement *selection* and *projection*, and the *reduce* phase roughly corresponds to *aggregation*, *grouping* and *ordering*.

At the same time as the Big Data systems became prevalent, the so-called CAP theorem was conjectured [BRE 00] and proved [GIL 02]. In a nutshell, this formal result states that no distributed data store can ensure, at the same time, optimal Consistency, Availability and Partition tolerance. In the context of distributed data stores, *consistency* is the guarantee that a read operation will return the result of the most recent *global* write to the system (or an error). *Availability* is the property that every request receives a response that is not an error (however, the answer can be outdated). Finally, *partition tolerance* is the ability for the system to remain responsive when part of its components are isolated (due to network failures, for instance). In the context of the CAP theorem, the ACID properties enjoyed by RDBMS consist of favoring consistency over availability. With the rise of Big Data and associated applications, new systems emerged that favored availability over consistency. Such systems follow the BASE principles (Basically Available, Soft state and Eventual consistency). The basic tenets of the approach is that operations on the systems (queries as well as updates) must be as fast as possible and therefore no global synchronization between nodes of the system should occur at the time of operation. This, in turn, implies that after an operation, the system may be in an inconsistent state (where several nodes have different views of the global data set). The system is only required to *eventually* correct this inconsistency (the resolution method is part of the system design and varies from system to system). The wide design space in that central aspect of implementation gave rise to a large number of systems, each having its own programming interface. Such systems are often referred to with the umbrella term *Not only SQL* (NoSQL). While, generally speaking, NoSQL can also characterize XML databases and Graph databases, these define their own field of research. We therefore focus our study on various kinds of lower-level data stores.

1.1.2. *Overview of NoSQL concepts*

Before discussing the current trends in research on NoSQL languages and systems, it is important to highlight some of the technical concepts of such systems. In fact, it is their departure from well understood relational traits that fostered new research and development in this area. The aspects which we focus on are mainly centered around computational paradigms and data models.

1.1.2.1. *Distributed computations with MapReduce*

As explained previously, the *MapReduce* paradigm consists of decomposing a generic, high-level computation into a sequence of lower-level, distributed, *map* and *reduce* operations. Assuming some data elements are distributed over several nodes, the *map* operation is applied to each element individually, locally on the node where the element resides. When applied to such an element e, the *map* function may decide to either discard it (by not returning any result) or transform it into a new element e', to which is associated a *grouping key*, k, thus returning the pair (k, e'). More generally, given some input, the *map* operation may output any number of key-value pairs. At the end of the map phase, output pairs are exchanged between nodes so that pairs with the same key are grouped on the same node of the distributed cluster. This phase is commonly referred to as the *shuffle phase*. Finally, the *reduce* function is called once for each distinct key value, and takes as input a pair $(k, [e'_1, \ldots, e'_n])$ of a key and all the outputs of the map function that were associated with that key. The *reduce* function can then either discard its input or perform an operation on the set of values to compute a partial result of the transformation (e.g. by aggregating the elements in its input). The result of the *reduce* function is a pair (k', r) of an output key k' and a result r. The results are then returned to the user, sorted according to the k' key. The user may choose to feed such a result to a new pair of *map/reduce* functions to perform further computations. The whole MapReduce process is shown in Figure 1.1.

This basic processing can be optimized if the operation computed by the reduce phase is associative and commutative. Indeed, in such a case, it is possible to start the *reduce* operations on subsets of values present locally on nodes after the *map* phase, before running the shuffle phase. Such an operation is usually called a *combine* operation. In some cases, it can drastically improve performance since it reduces the amount of data moved

around during the *shuffle phase*. This optimization works particularly well in practice since the *reduce* operations are often aggregates which enjoy the commutativity and associativity properties (e.g. sum and average).

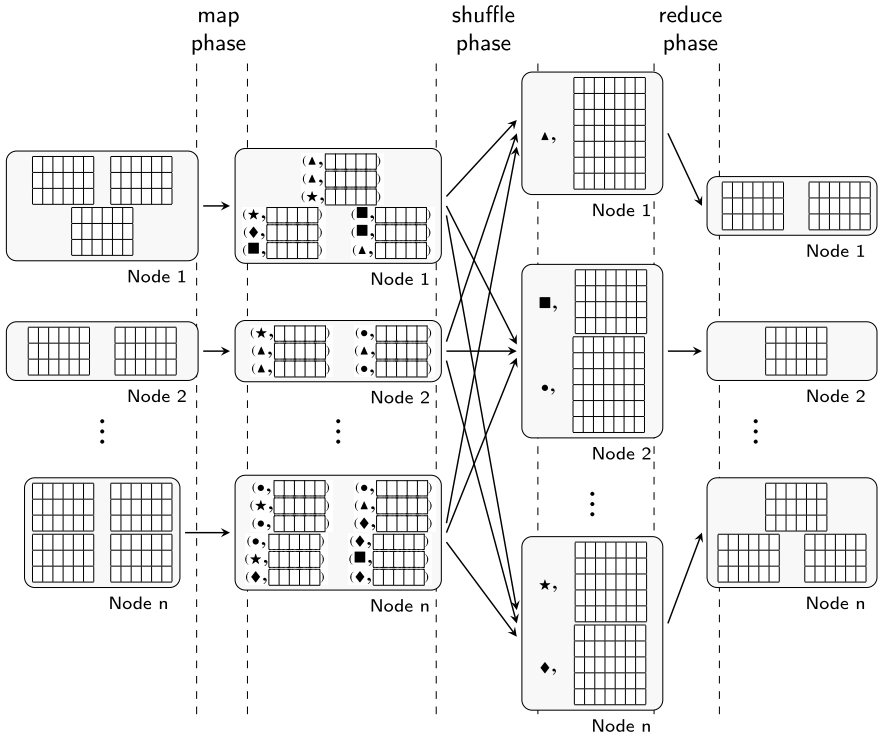


Figure 1.1. *MapReduce*

The most commonly used MapReduce implementation is certainly the Apache Hadoop framework [WHI 15]. This framework provides a Java API to the programmer, allowing us to express map and reduce transformations as Java methods. The framework heavily relies on the Hadoop Distributed File System (HDFS) as an abstraction for data exchange. The map and reduce transformations just read their input and write their output to the file system, which handle the lower-level aspects of distributing chunks of the files to the components of the clusters, and handle failures of nodes and replication of data.

1.1.2.2. *NoSQL databases*

A common trait of the most popular NoSQL databases in use is their nature as key-value stores. A key-value store is a database where collections are inherently *dictionaries* in which each entry is associated with a key that is unique to the collection. While it seems similar to a relational database where tables may have primary keys, key-value stores differ in a fundamental way from relational tables in that they do not rely on – nor enforce – a fixed *schema* for a given collection. Another striking aspect of all these databases is the relatively small set of operations that is supported natively. Updates are usually performed one element at a time (using the key to denote the element to be added, modified or deleted). Data processing operations generally consist of filtering, aggregation and grouping and exposing a MapReduce-like interface. Interestingly, most NoSQL databases do not support join operations, rather they rely on data *denormalization* (or materialized joins) to achieve similar results, at the cost of more storage usage and more maintenance effort. Finally, some databases expose a high-level, user-friendly query language (sometimes using an SQL compatible syntax) where queries are translated into combinations of lower-level operations.

1.1.3. *Current trends of French research in NoSQL languages*

NoSQL database research covers several domains of computer science, from system programming, networking and distributed algorithms, to databases and programming languages. We focus our study on the *language* aspect of NoSQL systems, and highlight two main trends in French research that pertains to NoSQL languages and systems.

The first trend aims to add support for well-known, relational operations to NoSQL databases. In particular, we survey the extensive body of work that has been done to add support for join operations between collections stored in NoSQL databases. We first describe how join operations are implemented in NoSQL systems (and in particular how joins can be decomposed into sequences of MapReduce operations).

The second trend of research is aimed at unifying NoSQL systems, in particular their query languages. Indeed, current applications routinely interact with several data stores (both relational and NoSQL), using high-level programming languages (PHP, Java, Ruby or JavaScript for Web applications,

2007

BENHAMOU Frédéric, JUSSIEN Narendra, O'SULLIVAN Barry
Trends in Constraint Programming

JUSSIEN Narendra
A TO Z OF SUDOKU

2006

BABAU Jean-Philippe *et al.*
From MDD Concepts to Experiments and Illustrations – DRES 2006

HABRIAS Henri, FRAPPIER Marc
Software Specification Methods

MURAT Cecile, PASCHOS Vangelis Th
Probabilistic Combinatorial Optimization on Graphs

PANETTO Hervé, BOUDJLIDA Nacer
Interoperability for Enterprise Software and Applications 2006 / IFAC-IFIP I-ESA'2006

2005

GÉRARD Sébastien *et al.*
Model Driven Engineering for Distributed Real Time Embedded Systems

PANETTO Hervé
Interoperability of Enterprise Software and Applications 2005