Practical Hadoop Migration

How to Integrate Your RDBMS with the Hadoop Ecosystem and Re-Architect Relational Applications to NoSQL

Bhushan Lakhe

Foreword by Milind Bhandarkar

Apress®

Practical Hadoop Migration

How to Integrate Your RDBMS with the Hadoop Ecosystem and Re-Architect Relational Applications to NoSQL

Bhushan Lakhe

Apress®

Practical Hadoop Migration: How to Integrate Your RDBMS with the Hadoop Ecosystem and Re-Architect Relational Applications to NoSQL

Bhushan Lakhe
Darien, Illinois
USA

ISBN-13 (pbk): 978-1-4842-1288-2 ISBN-13 (electronic): 978-1-4842-1287-5
DOI 10.1007/978-1-4842-1287-5

Library of Congress Control Number: 2016948866

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springer.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text is available to readers at www.apress.com. For detailed information about how to locate your book's source code, go to www.apress.com/source-code/.

Printed on acid-free paper

To my mother....

Contents at a Glance

Contents

Foreword

We are in the midst of one of the biggest transformations of Information Technology (IT). Rapidly evolving business requirements have demanded agility in all aspects of IT. As more and more paper-based business processes are getting digital, rapid application development, staging, and deployment have become the norm. In addition, the data exhaust from these digital applications has become enormous and needs to be analyzed in real time. Growing volumes of historical data is considered valuable for improving business efficiency and identifying future trends and disruptions. Ubiquitous end-user connectivity, cost-efficient software and hardware sensors, and democratization of content production have led to the deluge of data generated in enterprises. As a result, the traditional data infrastructure has to be revamped. Of course, this cannot be done overnight. To prepare your IT to meet the new requirements of the business, one has to carefully plan re-architecting the data infrastructure so that existing business processes remain available during this transition.

Hadoop and NoSQL platforms have emerged in the last decade to address the business requirements of large web-scale companies. Capabilities of these platforms are evolving rapidly, and, as a result, have created a lot of hype in the industry. However, none of these platforms is a panacea for all the needs of a modern business. One needs to carefully consider various business use cases and determine which platform is most suitable for each specific use case. Introducing immature platforms for use cases that are not suited for them is the leading cause of failure of data infrastructure projects. Data architects of today need to understand a variety of data platforms, their design goals, their current and future data protection capabilities, access methods, and performance sweet spots, and how they compare in features against traditional data platforms. As a result, traditional database administrators and business analysts are overwhelmed by the sheer number of new technologies and the rapidly changing data landscape.

This book is written with those readers in mind. It cuts through the hype and gives a practical way to transition to the modern data architectures. Although it may feel like new technologies are emerging every day, the key to evaluating these technologies is to align your current and future business use cases and requirements to the design-center of these new technologies. This book helps readers understand various aspects of the modern data platforms and helps navigate the emerging data architecture. I am confident that it will help you avoid the complexity of implementing modern data architecture and allow seamless transition for your business.

—Milind Bhandarkar, PhD
Founder and CEO, Ampool, Inc.

Milind Bhandarkar was the founding member of the team at Yahoo! that took Apache Hadoop from 20-node prototype to datacenter-scale production system, and has been contributing and working with Hadoop since version 0.1.0. He started the Yahoo! Grid solutions team focused on training, consulting, and supporting hundreds of new migrants to Hadoop. Parallel programming languages and paradigms has been his area of focus for over 20 years. He has worked at the Center for Development of Advanced Computing (C-DAC), National Center for Supercomputing Applications (NCSA), Center for Simulation of Advanced Rockets, Siebel Systems, Pathscale Inc. (acquired by QLogic), Yahoo!, and Linkedin. Until 2013, Milind was chief architect at Greenplum Labs, a division of EMC. Most recently, he was chief scientist at Pivotal Software. Milind holds his PhD degree in computer science from the University of Illinois at Urbana-Champaign.

About the Author

Bhushan Lakhe is a Big Data professional, technology evangelist, author, and avid blogger who resides in the windy city of Chicago. After graduating in 1988 from one of India's leading universities (Birla Institute of Technology and Science, Pilani), he started his career with India's biggest software house, Tata Consultancy Services. Thereafter, he joined ICL, a British computer company, and worked with prestigious British clients. Moving to Chicago in 1995, he worked as a consultant with Fortune 50 companies like Leo Burnett, Blue Cross, Motorola, JPMorgan Chase, and British Petroleum, often in a critical and pioneering role.

After a seven-year stint executing successful Big Data (as well as data warehouse) projects for IBM's clients (and receiving the company's prestigious Gerstner Award in 2012), Mr. Lakhe spent two years helping Unisys Corporation's clients with Big Data implementations, and thereafter two years as senior vice president (information and data architecture) at Ipsos (the world's third-largest market research corporation), helping design global data architecture and Big Data strategy.

Currently, Mr. Lakhe heads the Big Data practice for HCL America, a $7 billion global consulting company with offices in 31 countries. At HCL, Mr. Lakhe is involved in architecting Big Data solutions for Fortune 500 corporations. Mr. Lakhe is active in the Chicago Hadoop community and is co-organizer for a Meetup group (www.meetup.com/ambariCloud-Big-Data-Meetup/) where he regularly talks about new Hadoop technologies and tools. You can find Mr. Lakhe on LinkedIn at www.linkedin.com/in/bhushanlakhe.

About the Technical Reviewer

Robert L. Geiger is currently Chief Architect and acting VP of engineering at Ampool Inc., an early stage startup in the Big Data and analytics infrastructure space. Before joining Ampool, he worked as an architect and developer in the solutions/SaaS space at a B2B deep learning based startup, and prior to that as an architect and team lead at Pivotal Inc., working in the areas of security and analytics as a service for the Hadoop ecosystem. Prior to Pivotal, Robert served as a developer and VP, engineering at a small distributed database startup, TransLattice. Robert spent several years in the security space working on and leading teams in at Symantec on distributed intrusion detection systems. His career started with Motorola Labs in Illinois where he worked on distributed IP over wireless systems, crypto/security, and e-commerce after graduating from University of Illinois Champaign-Urbana.

Acknowledgments

This is my second book for Apress (the first being *Practical Hadoop Security*) continuing the *Practical Hadoop* series, and I want to thank Apress for giving me the opportunity to write it. I would like to thank the Hadoop community and the user forums that bring innovation to this technology and keep the world interested! I have learned a lot from the selfless people in the Hadoop community who believe in being Good Samaritans.

On a personal note, I want to thank my friend Satya Kondapalli for making a forum of Hadoop enthusiasts available through our Meetup group Ambaricloud. I also want to thank our sponsors Hortonworks for supporting us. Finally, I would like to thank my friend Milind Bhandarkar (of Ampool) for taking time from his busy schedule to write a foreword and a whole section about his new Butterfly architecture.

I am grateful to my editors, Rita Fernando, Robert Hutchinson, and Matthew Moodie at Apress for their help in getting this book toegther. Rita has been there throughout to answer any questions that I have, to improve my drafts, and to keep me on schedule. Robert Hutchinson's help with the book structure has been immensely valuable. And I am also very thankful to Robert Geiger for taking time to review my second book technically. Bob always had great suggestions for improving a topic, recommending additional details, and of course resolving technical shortcomings.

Finally, the writing of this book wouldn't have been possible without the constant support from my family (my wife, Swati, and my kids, Anish and Riya) for the second time in the last three years, and I'm looking forward to spending lots more time with all of them.

Introduction

I have spent more than 20 years consulting for large corporations, and when I started, it was just relational databases. Eventually, the volumes of accumulated historical data grew, and it was not possible to manage and analyze this data with good performance. So, corporations started thinking about separating the parts (of data) useful for analaysis (or generating insights) from the descriptive data. They soon realized that a fundamental change was needed in the relational design, and a new paradigm called data warehousing was born. Thanks to the work done by Bill Inmon and Ralph Kimball, the world started thinking (and designing) in terms of Star schemas and dimensions and facts. ETL (extract, transform, load) processes were designed to load the data warehouses.

The next step was making sure that large volumes of data could be retrieved with good performance. Specialized software was developed, and RDBMS solutions (Oracle, Sysbase, SQL Server) added processing for data warehouses. For the next level of performance, it was clear that data needed to be preprocessed, and data cubes were designed. Since magnetic disk drives were slow, SSDs (solid state devices) were designed, and software that cached (or held data in RAM) data for speed of processing and retrieval became popular. So, with all these advanced measures for performance, why is Hadoop or NoSQL needed? For two reasons.

First, it is important to note that all this while, the data being processed either was relational data (for RDBMS) or had started as relational data (for data warehouses). This was structured data, and the type of analysis (and insights) possible was very specific (to the application that generated the data). The rigid structure of a warehouse put severe limits on the insights or data explorations that were possible, since you start with a design and fit data into it. Also, due to the very high volumes, warehouses couldn't perform per expectations, and a newer technology was needed to effectively manage this data.

Second, in recent years, new types of data were introduced: unstructured or semi-structured data. Social media became very popular and were a new avenue for corporations to communicate directly with people once they realized the power behind it. Corporations wanted to know what people thought about their products, services, employees, and of course the corporations themselves. Also, with e-commerce forming a large part of all the businesses, corporations wanted to make sure they were preferred over their competitors—and if that was not the case, they wanted to know why. Finally, there was a need to analyze some other types of unstructured data, like sensor data from electrical and electronic devices, or data from mobile devices sensors, that was also very high volume. All this data was usually hundreds of gigabytes per day.

Conventional warehouse technology was incapable of processing or managing this data. So, a new technology had to be designed to process it, and with good performance (since total volumes were in terabytes). In some cases, as the unstructured data (or insights from it) needed to be combined with structured data, the new technology needed to support interfacing with data warehouses or RDBMS.

Hadoop offers all these capabilities and in addition allows a schema-on-read (meaning you can define metadata while performing analysis) that offers a lot of flexiblity for performing exploratory analysis or generating new insights from your data.

This gets us to the final question: how do you migrate or integrate your existing RDBMS-based applications with Hadoop and analyze structured as well as unstructured data in tandem? Well, you have to read rest of the book to know that!

Who This Book Is For

This book is an excellent resource for IT management planning to migrate or integrate their existing RDBMS environment with Big Data technologies or Big Data architects who are designing a migration/integration process. This book is also for Hadoop developers who want to implement migration/integration process or students who'd like to learn about designing Hadoop applications that can successfully process relational data along with unstructured data. This book assumes a basic understanding of Hadoop, Kerberos, relational databases, Hive, Spark, and an intermediate level understanding of Linux.

Downloading the Code

The source code for this book is available in ZIP file format in the Downloads section of the Apress Web site (`www.apress.com/9781484212882`).

Contacting the Author

You can reach Bhushan Lakhe at `blakhe@aol.com` or `bclakhe@gmail.com`.

CHAPTER 1

RDBMS Meets Hadoop: Integrating, Re-Architecting, and Transitioning

Recently, I was at the Strata + Hadoop World Conference, chatting with a senior executive of a major food corporation who used a relational solution for storing all its data. I asked him casually if they were thinking about using a Big Data solution, and his response was: "We already did and it's too slow!" I was amazed and checked the facts again. This corporation had even availed of the consulting services of a major Hadoop vendor and yet was still not able to harness the power of Big Data.

I thought about the issue and possible reasons why this might have occurred. To start with, a Hadoop vendor can tune his Hadoop installation but can't guarantee that generic tuning will be valid for specific type of data. Second, the food corporation's database administrators and architects probably had no idea how to transform their relational data for use with Hadoop. This is not an isolated occurrence, and most of the corporations who want to make the transition to using of relational data with Hadoop are in a similar situation. The result is a Hadoop cluster that's slow and inefficient and performs nowhere close to the expectations that Big Data hype has generated.

Third, all NoSQL databases are not created equal. NoSQL databases vary greatly in their handling of data as well as in the models they use internally to manage data. They only work well with certain kind of data. So, it's very important to know the type of your data and select a NoSQL solution that matches it.

Finally, success in applying NoSQL solutions to relational data depends on identifying your objective in using Hadoop/NoSQL and on accommodating your data volumes. Hadoop is not a cure-all that can magically speed up all your data processing—it can only be used for specific type of processing (which I discuss further in this chapter). And Hadoop works best for larger volumes of data and is not efficient for lower data volumes due to the various overheads associated.

Electronic supplementary material The online version of this chapter (doi:10.1007/978-1-4842-1287-5_1) contains supplementary material, which is available to authorized users.

So, having defined the problem, let's think about a solution. You are probably familiar with the myriad design methodologies and frameworks that are available for use with relational data, but do you know of similar resources for Hadoop? Probably not. There is a good reason for that—none exists yet. Lambda is being developed as a design methodology (Chapter 12), but it is not mature yet and not very easy to implement.

So, what's the alternative? Do you need to rely on the expertise of your data architects to design this transition, or are there generic steps you can follow? How do you ensure an efficient and functionally reliable transition? I answer these questions in this book and demonstrate how you can successfully transition your relational data to Hadoop.

First, it is important to understand how Hadoop and NoSQL differ from the relational design. I briefly discuss that in this chapter and also discuss the benefits as well as challenges associated with using Hadoop and NoSQL.

It is also important to decide whether your data (and what you want to do with it) is suited for use with Hadoop. Therefore, factors such as type of data, data volume, and your business needs are important to consider. There are some more factors that you need to consider, and the latter part of this chapter discusses them at length. Typically, the four "V"s (volume, velocity, variety, and veracity) separate NoSQL data from relational data, but that rule of thumb may not always hold true.

So, let me start the discussion with conceptual differences between relational technology and Hadoop. That's the next section.

Conceptual Differences Between Relational and HDFS NoSQL Databases

Database design has had a few facelifts since E.F. Codd presented his paper on relational design in 1970.[1] Leading relational database systems today (such as Oracle or Microsoft SQL Server) may not be following Codd's vision completely; but definitely use the underlying concepts without much of modification. There is a central database server that holds the data and provides access to users (as defined by Database Administrator) after authentication. There are database objects such as views (for managing granular permissions) or triggers (to manipulate data as per data 'relations') or indexes for performance (while reading or modifying data).

The main feature, however, is that relations can be defined for your data. Let me explain using a quick example. Think of an insurance company selling various (life, disability, home) policies to individual customers. A good identifier to use (for identifying a customer uniquely) is customers' social security number. Since a customer may buy multiple policies from the insurance company and those details may be stored in separate database tables, there should be a way to relate all that data to the customer it belongs to.

Relational technology implements that easily by making the social security number as a primary key or primary identifier for the `customer` table and a foreign key or referential identifier (an identifier to identify the parent or originator of the information) for all the related tables, such as `life_policies` or `home_policies`. Figure 1-1 summarizes a sample implementation.

[1]www.seas.upenn.edu/~zives/03f/cis550/codd.pdf "A Relational Model of Data for Large Shared Data Banks"

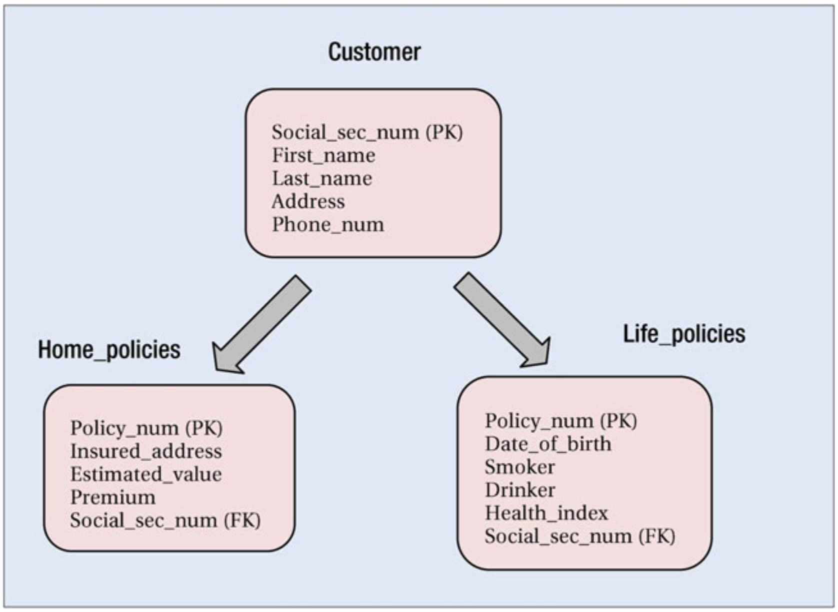
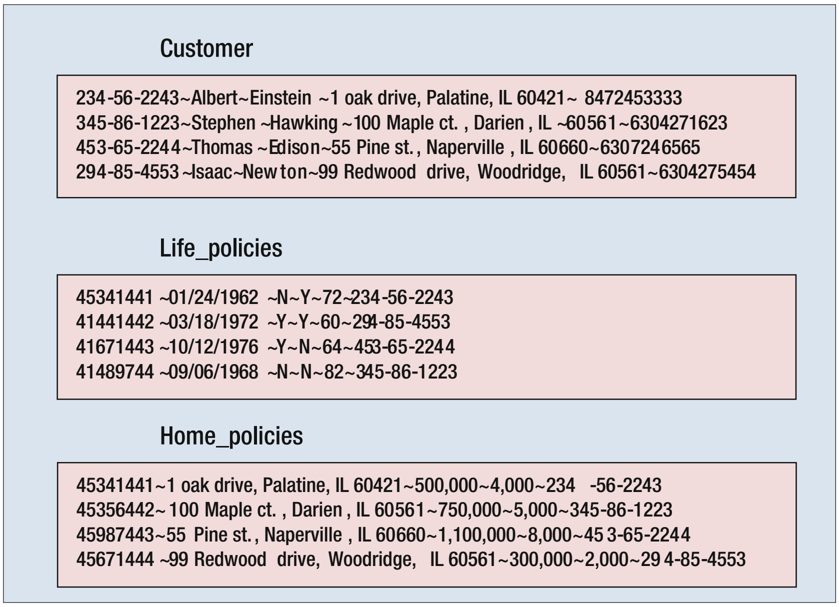


Figure 1-1. *Relational storage of data (logical)*

As you can see in Figure 1-1, the policy data is related to customers. This relation is established using the social security number. So, all the policy records for a customer can be retrieved using their social security number. Any modifications to the customer identifier (social security number) are propagated to maintain data integrity.

Next, let me discuss Hadoop and NoSQL databases that use HDFS for storage. HBase is a popular NoSQL database and therefore can be used as an example. Since HDFS is a distributed file system, data will be spread across all the data nodes in contrast to a central server. Kerberos is used for authentication, but HBase has very limited capability for granular authorization as opposed to relational databases. HBase offers indexing capabilities, but they are very limited and are no match for the advanced indexing techniques offered by RDBMS (relational database management systems). However, the main difference is absence of relations. Unlike RDBMSs, HBase data is not related. Data for HBase tables is simply held in HDFS files.

As you can see in Figure 1-2, the policy data is not related automatically with a customer. Any relating that's necessary will have to be done programmatically. For example, if you need to list all the policies that customer "Isaac Newton" holds, you will need to know the tables that hold policies for customers (here, Hbase tables `Life_policies` and `Home_policies`). Then you will need to know a common identifier to use (social security number) to match the rows that belong to this customer. Any changes to the identifier can't be propagated automatically and will need to be implemented manually.



Customer

```
234-56-2243~Albert~Einstein ~1 oak drive, Palatine, IL 60421~ 8472453333
345-86-1223~Stephen ~Hawking ~100 Maple ct. , Darien , IL ~60561~6304271623
453-65-2244~Thomas ~Edison~55 Pine st., Naperville , IL 60660~6307246565
294-85-4553 ~Isaac~Newton~99 Redwood drive, Woodridge, IL 60561~6304275454
```

Life_policies

```
45341441 ~01/24/1962 ~N~Y~72~234-56-2243
41441442 ~03/18/1972 ~Y~Y~60~294-85-4553
41671443 ~10/12/1976 ~Y~N~64~453-65-2244
41489744 ~09/06/1968 ~N~N~82~345-86-1223
```

Home_policies

```
45341441~1 oak drive, Palatine, IL 60421~500,000~4,000~234 -56-2243
45356442~ 100 Maple ct. , Darien , IL 60561~750,000~5,000~345-86-1223
45987443~55 Pine st., Naperville , IL 60660~1,100,000~8,000~453-65-2244
45671444 ~99 Redwood drive, Woodridge, IL 60561~300,000~2,000~294-85-4553
```

Figure 1-2. *NoSQL storage of data*

So, for example, if an error in social security number is discovered, then all the files containing that information will need to be updated separately (programmatically). Unlike RDBMS, HDFS or HBase doesn't offer any utilities to do that for you. The reason is that HBase (or any other HDFS-based NoSQL databases) doesn't offer any referential integrity—simply due to their purpose. HBase is not meant for interactive queries over a small dataset; it is best suited for a large batch processing environment (similar to data warehousing environments) involving immutable data. Till recently, updates for HBase involved loading the changed row in a staging table and doing a left outer join with the main data table to overwrite the row (making sure the staging and main data table had the same key).

With the new version of HBase, updates, deletes, and inserts are now supported, but for small datasets these operations will be very slow (compared to RDBMS) because they're executed as Hadoop MapReduce jobs that have high latency and incur substantial overheads in job submission and scheduling.

Starting with a large block size used by HDFS (default 64 MB) and distributed architecture that spreads data over a large number of DataNodes (helping parallel reads using MapReduce or Yarn), HBase (and other HDFS based NoSQL databases) are meant to perform efficiently for large datasets. Any transformations that need to be applied involve reading the whole table and not a single row. Distributed processing on DataNodes using MapReduce (or Yarn on recent versions) provides the speed and efficiency for such reads. Again, due to the distributed architecture, it is much more efficient to write the transformed data to a new "file" (or staging table for HBase). For the same reason, Hadoop and NoSQL databases are better equipped to store (and process) large image or video files, large blocks of natural language text, or semi-structured as well as sensor data.

Compare this with a small page size for RDBMS (for example, Microsoft SQL Server uses a page size of 8 KB) and absence of an efficient mechanism to distribute the read (or update) operations and you will realize why NoSQL databases will always win in any scenarios that involve data warehouses and large datasets. The strength of RDBMS, though, is where there are small datasets with complex relationships and extensive analysis is required on parts of it. Also, where referential integrity is important to be implemented over a dataset, NoSQL databases are no match for RDBMS.

To summarize, RDBMS is more suited for a large number of data manipulations for smaller datasets where ACID (Atomicity, Consistency, Isolation, Durability) compliance is necessary; whereas NoSQL databases are more suited for a smaller number of data manipulations to large datasets that can work with the "eventual consistency" model. Table 1-1 provides a handy comparison between the two technologies (relational and NoSQL).

Table 1-1. *Comparative Features of RDBMS vs. NoSQL*

	Feature	HDFS-based NoSQL	RDBMS
1	Large datasets	Efficient and fast	Not efficient
2	Small datasets	Not efficient	Efficient and fast
3	Searches	Not efficient	Efficient and fast
4	Large read operations	Efficient and fast	Not efficient
5	Updates	Not efficient	Efficient and fast
6	Data relations	Not supported	Supported
7	Authentication/Authorization	Kerberos	Built-in
8	Data storage	Distributed over DataNodes	Central Database server
9	ACID compliant	No	Yes
10	Concurrent updates to dataset	Not supported	Supported
11	Fault tolerance	Built-in	Not built-in
12	Scalability	Easily scalable	Not easily scalable

Figure 1-3 shows the physical data storage configurations (for the preceding example) including a Hadoop cluster (Hive/NoSQL) and RDBMS (Microsoft SQL Server).