Better Performance for Big Data

One of the Largest Banks in Italy Speeds Processing for Big Data Analytics with Intel® Distribution for Apache Hadoop Software

EXECUTIVE SUMMARY

A large Italian bank needed a more cost-effective way to manage the vast amounts of data it must organize and report on to comply with government regulations. It worked with Intel to pilot a solution based on Intel® Distribution for Apache Hadoop software. The new solution helped the bank meet its data challenges and enjoy faster performance for data acquisition, extraction, and analysis. It also delivered more flexibility to let the bank adapt to the nature of the data generated. The bank is now investigating whether other applications and datasets could benefit from an infrastructure based on Apache Hadoop software.

The new solution helped the bank meet its data challenges and enjoy faster performance for data acquisition, extraction, and analysis. It also delivered more flexibility to let the bank adapt to the nature of the data generated.

Authors

Bhasker Allene, Intel Corporation
bhasker.allene@intel.com

Marco Righini, Intel Corporation
marco.righini@intel.com

The Data Challenge

Financial institutions, especially in the European Union (EU), must comply with numerous regulations. In most countries, they need to report to their respective central banks the overall health of both the financial institution itself and the entities it supervises.

In Italy, banks require vast amounts of data from several sources, using it for control activities and to help identify new areas of administrative inspection as well as to understand their market segment. A bank uses the information to make risk assessments for the entities it supervises and to identify early signs of anomalies to prevent potential crises.

Basic information and business analysis of the data also allows the bank to:

- **Exercise** its powers of authorization and veto regarding the structure and operation of intermediaries in the cases provided by law
- **Study** the evolution of the banking and financial system and its potential to update the regulatory framework
- **Comply** with requests for information from institutional representatives of the supervisory board (i.e., judiciary, Parliament, ministries, and other organizations and national and foreign supervisory authorities)

The data in the statistical reports allows the bank to carry out systematic analysis on the technical standing, risk profiles, assets, and income of banks and financial groups and to verify compliance with both individual and consolidated prudential rules.

Today, most banks analyze their data using either mainframes or very costly solutions. One bank in Italy wanted to prove that it could get the same results with less expensive and more scalable and flexible solutions such as those based on Apache Hadoop, an open-source framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models.

One of the biggest banks in the country had started years ago to migrate workloads from the mainframe to a distributed environment. Cost control, scalability, flexibility, and choice were the primary drivers behind this strategy, which implies calculated risks around the company's core business.

In 2001, the bank purchased an application stack that was able to get input data from the mainframe, extract it, and perform all mandatory monthly and quarterly reports to the central bank.

The project was only partially successful. For the last four years, the bank has been considering how to completely redesign its

solution and move from its current application to an Oracle*-based, closed solution. If successful, this would improve on all the negative aspects of the previous solution, which included:

- **Long** processing time
- **Inability** of the existing database to handle data tables with more than 1,024 columns
- **Need** for a specific programming language to interface with the bank's proprietary solution
- **Limited** capacity and data quality and inability to create a bigger data warehouse containing all the required data
- **The increasing costs** associated with a three-year postponement of the migration to the new solution, caused by lack of tools, the syntax of semi-structured data, and the bank's speed of change

Intel proposed a pilot solution based on Intel Distribution for Apache Hadoop software, using its flexibility to address all these challenges and deliver:

- **Faster performance** in the load (data acquisition) phase
- **The same or better performance** in the extraction and analysis phase
- **More flexibility** to adapt to the nature of the data generated by the bank (a perfect fit with the Hadoop usage model)
- **Containment** of the Oracle footprint within the bank

Besides these critical considerations, the bank realized that Hadoop was able to do many of the most frequent conversions and that the Intel Distribution for Apache Hadoop software is a great tool for downsizing batch processing.

The return on investment was another key consideration. The bank could pay for the new hardware, software, and support subscription, plus four weeks of professional services, through savings on storage changes. The bank realized that moving from storage area network (SAN) storage to Hadoop-based distributed storage would, on its own, pay back most of the initial investment.

Historically, financial institutions have run their core IT banking applications on mainframes. Data is stored on a Mainframe in Extended Binary Coded Decimal Interchange Code (EBCDIC). To comply with mandatory regulations, the bank needs to run monthly jobs on the mainframe, export some specific data, and transform it from EBCDIC to ASCII.

Those files then need to be converted and loaded monthly into a data warehouse, which keeps 24 months worth (approximately 74 terabytes) of data online.

It is hard to process the data with traditional approaches due to:

- **The variety of data.** The data changes structure frequently. Every month, columns can vary in both quantity and order.
- **Semi-structured data.** The type of data the bank uses is semi-structured and very close to XML.
- **Processing speed.** To meet legal requirements, reports based on the data need to be delivered on time. It was challenging for the bank to transform the data and put it into a structured database. A single 130-gigabyte file, for example, was taking close to seven hours to process.

This Italian bank was considering a new solution called SISBA* 3, which would offload MIPS from the mainframe onto the X86 architecture using Oracle CUBE*. However, the bank disagreed with Oracle on licensing and decided to try a new approach using the Hadoop framework.

The bank's goals for a successful pilot with Intel Distribution for Apache Hadoop software were to:

- **Reduce** software licensing costs
- **Scale** more and different types of data
- **Convert** and process data more quickly
- **Gain** flexibility in data types, table column formats, and data transformation on column changes
- **Reduce** overall infrastructure costs
- **Maintain** skill sets and query tools in which the bank had already invested
- **Have** an open solution without vendor lock-in

Technical Solution

What is Hadoop?

Apache Hadoop is an open-source software framework, licensed under the Apache v2 license, that supports data-intensive distributed applications. It enables the running of applications on large clusters of commodity hardware. Hadoop was derived from Google's MapReduce* and Google File System* (GFS*) papers.

The Hadoop framework transparently provides both reliability and data motion to applications. Hadoop implements MapReduce, a computational paradigm that divides the application into many small fragments, each of which may be executed, or re-executed, on any node in the cluster. MapReduce also provides a distributed file system that stores data on the compute nodes, providing very high aggregate bandwidth across the cluster.

Both MapReduce and the distributed file system are designed so that node failures are automatically handled by the framework. This enables applications to work with thousands of computation-independent computers and petabytes of data.

The entire Apache Hadoop platform is now commonly considered to consist of the Hadoop kernel, MapReduce, and Hadoop Distributed File System* (HDFS*), as well as a number of

related projects including Apache Hive*, Apache HBase*, and others.

Written in the Java* programming language, Hadoop is an Apache top-level project being built and used by a global community of contributors. Hadoop and its related projects (e.g., Hive, HBase, and Zookeeper*) have many contributors from across the ecosystem. Though Java code is most common, any programming language can be used with streaming to implement the map and reduce parts of the system.

A Good Fit

Hadoop immediately appeared to be a good fit for this Italian bank, both from the architectural perspective and because of the nature of data to be processed.

Architecturally, Hadoop provides the intrinsic scalability and high availability a bank requires without need for huge investments. The scheduling management tools, the control of the MapReduce tasks with the self-healing mechanisms like speculative scheduling, and Oozie*, a powerful workflow engine, all made this solution a perfect fit for this bank.

From a data perspective, the advantage of the Hadoop solution was not the quantity of the data itself (today, approximately 75 terabytes of online data), but the nature of this data.

Figure 1 shows some records opened with a tail command in EBCIDIC format this before conversion. Figure 2 shows how each single record is formatted and self-described so that these semantics are used for conversion.

The data in Figure 2 is based on fixed columns at the beginning and a field describing the length of that specific record.

The three files considered in the pilot follow the same pattern line, fixed part of records with fixed length columns, variable part length, and variable part with key value pairs somewhat similar to JSON.

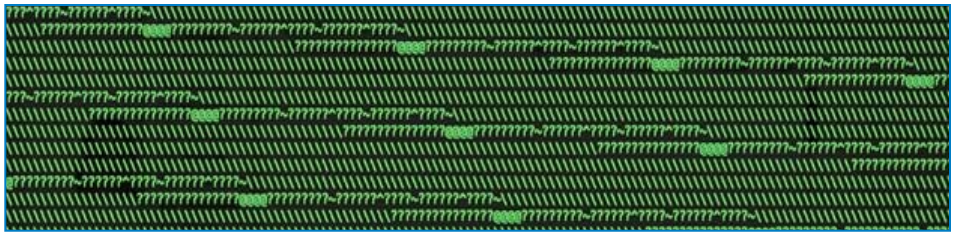

Figure 1. EBCDIC File Opened with Tail Command

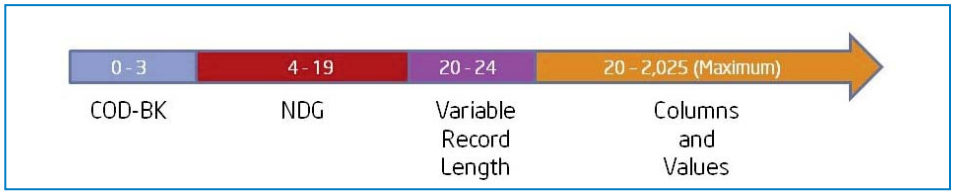

Figure 2. Record Split within Figure 1

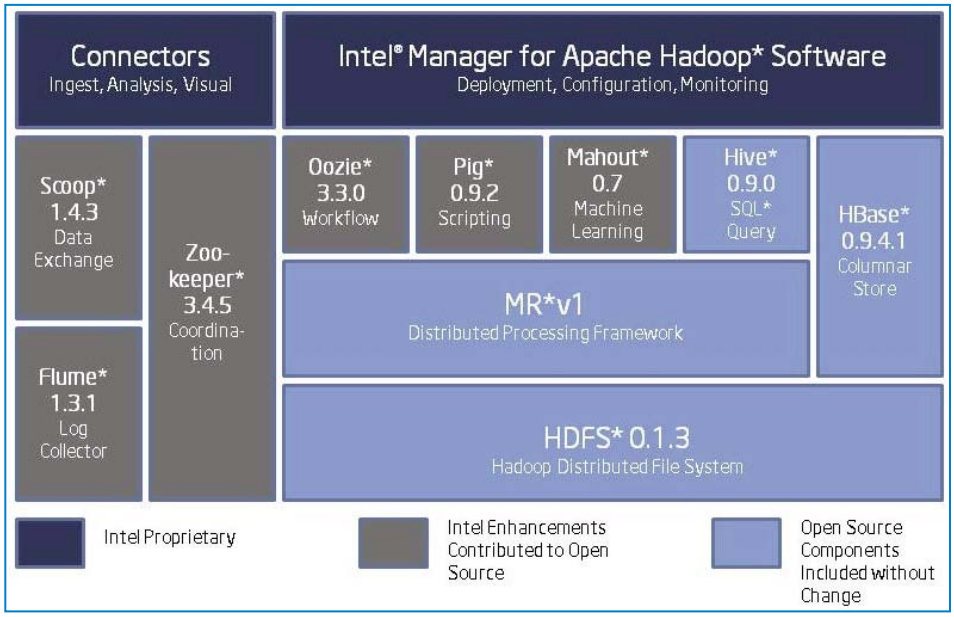

Figure 3. Intel Distribution for Apache Hadoop Software Framework

The number of columns can change without notice and, therefore, the semantics need to adapt easily. Each new added column would be integrated seamlessly without the need to reprocess older data. The table structure is altered accordingly.

The number of columns was reaching the limit of the Microsoft SQL Server* database version used for the actual application.

Pilot Steps

To move quickly, the team divided the pilot into two major steps:

- Demo proof of concept (PoC)
- Pilot

Demo PoC

This step aimed to:

- **Understand** whether the project was viable and whether Hadoop could meet the base requirements for performance and application compatibility.
- **Verify** whether, for the bank's basic queries, the Hadoop-based solution would deliver the same results in an acceptable amount of time.

Since the schema is complex, understanding all of the components was not a simple exercise.

The bank provided anonymized data that was brought onto a remote lab running on an Intel internal cloud infrastructure. The dataset consisted of:

- **One table** with 10.6 million records
- **Records** with between two and 20 columns

Data were not necessarily present in all records. Columns were optional and could appear in any order.

Figure 3 shows the Intel Distribution for Hadoop framework with its components used during the PoC. Not all components have been used.

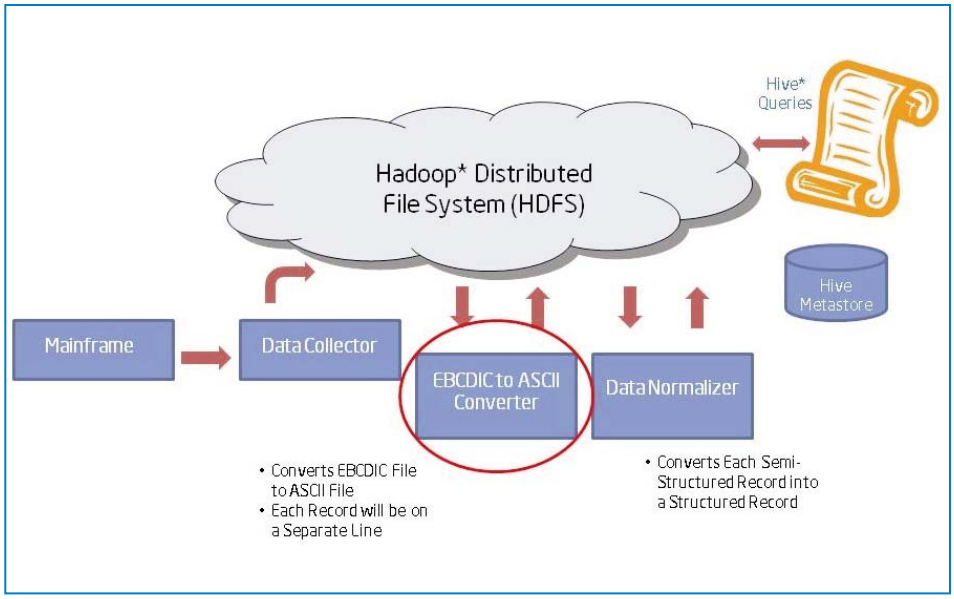

Figure 4. High-Level Data Flow

- **Intel Manager** facilitates cluster administration and monitoring of jobs
- **Sqoop* and Flume*** are connectors to external data
- **Zookeeper** provides distributed configuration service
- **Oozie** is a workflow scheduler
- **Pig*** scripting language interface to Hadoop
- **Mahout*** is a free implementation of distributed scalable machine learning algorithms
- **Hive** enables SQL queries on Hadoop
- **MapReduce** is the distributed processing framework
- **HDFS** is the Hadoop Distributed Filesystem
- **HBase** is a non-relational, distributed database

During this phase, a first activity:

- **Converted** the data set from EBCDIC to ASCII. This was developed with JAVA from Intel Consulting
- **Optimized** the data for Hadoop
- **Ran** simple queries in an acceptable way

From PoC to Pilot

The pilot required real, non-anonymized data with a very large dataset. Intel delivered to the bank's data center a rack of five Intel® Xeon® processor-based servers.

Hardware included:

- **Five** Intel Xeon processor E5-2690-based servers
- **Twenty** JBOD SAS* drives for each server
- **Two** Intel® Solid-State Drives 3700 series, each with 64 gigabytes of RAM and one 10-gigabyte Intel® Ethernet Controller X520

Software included:

- Centos* 6.3 x64 operating system
- Intel Distribution for Apache Hadoop software version 2.3
- Intel® Cache Acceleration Software (Intel® CAS)

The production dataset included three input files:

- **File name O0A011:** 18.6 million records with 109 columns
- **File name O0A021:** 25.2 million records and 464 columns
- **File name O0A025R:** 21.7 million records and 822 columns

Each record had between three and 822 columns. Not all data was present in all the records. Columns were optional and appeared in any order.

The pilot required treating the bank's production data with the same procedures as the PoC. The difference was that the pilot used more complex queries and compared the outputs and execution times to the bank's production environment.

For the pilot, the team connected the Intel Distribution for Hadoop software cluster to the mainframe to download the data and start converting the data from EDCIDIC into ASCII, and then optimized the ASCII file into a better format for HIVE.

During the first phase, the team did not use any workflow mechanism. In the pilot, they started to use Oozie to schedule and manage the workflows.

Figure 5 shows the steps used in Oozie. Figure 6 shows the Oozie workflow.

After first EBCDIC conversion, the data file looks like Figure 7.

Why Optimize the Data Already Converted from EBCDIC to ASCII?

One of the steps was to optimize the data for HIVE. This step is, in theory not required, but data is complicated for analysis. Each record is amalgamated with metadata.

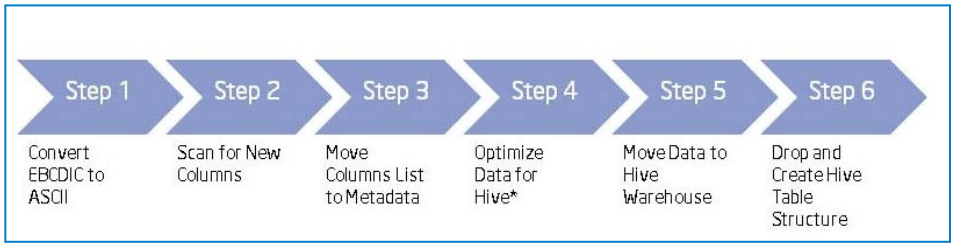

Figure 5. All Phases Executed into the Oozie Workflow

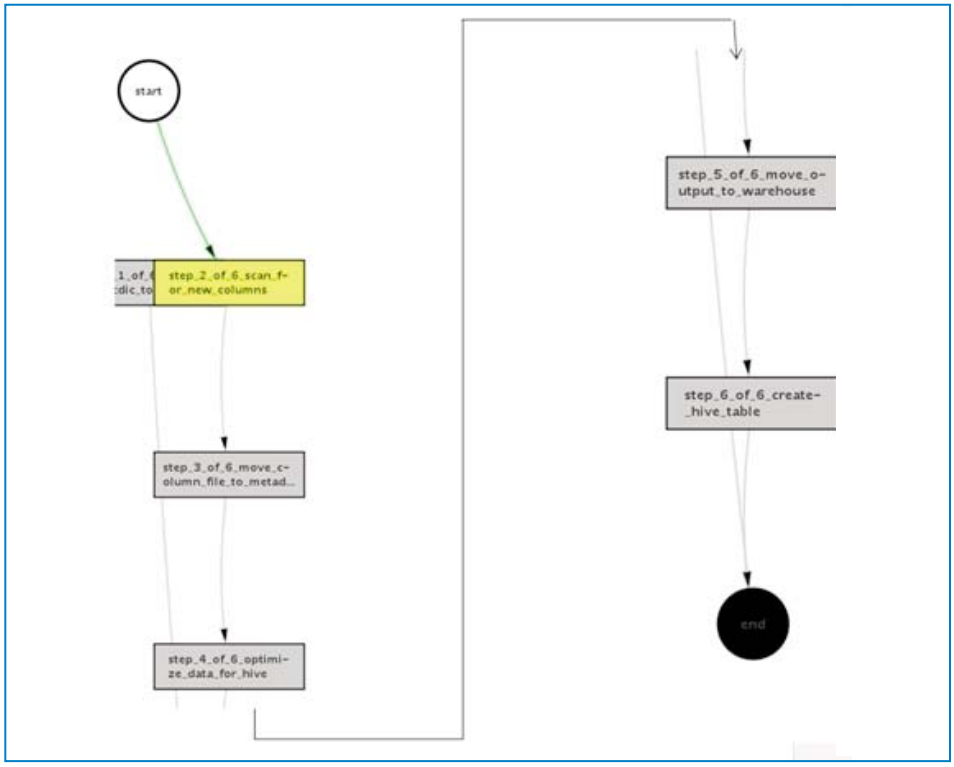

Figure 6. Oozie Workflow

For example, to fetch all records where the N011 value is 1:

- Split each record with ";" and further split each split with "="
- Scan through all splits to find out a split which got N011 in the first part and 1 in the second part

By optimizing the data, we:

- Scan the file for new columns
- Add the columns
- Use "CTRL-A" to delimit each column

Figure 8 shows the output usable file after optimizations.

This results in several benefits:

- **Data** is stored in a normalized way, similar to a table in RDBMS without the constraints of RDBMS
- **Hive queries** are very similar to RDBMS queries
- **The learning curve** is minimized, with fewer computations and less disk space
- **Data** is easily consumable for data analysis tools

Once the record is normalized, the records looks like this:

- **Record 1:** N047=627901;N051=121004;N071=290800;N002=221213;
- **Record 2:** N071=200600;N002=221213;N056=654123;
- **Record 3:** N047=627901;N002=221213;

Table 1 shows a populated table with the data coming from the imported and converted records.

Figure 7. Data After EBCDIC Conversion

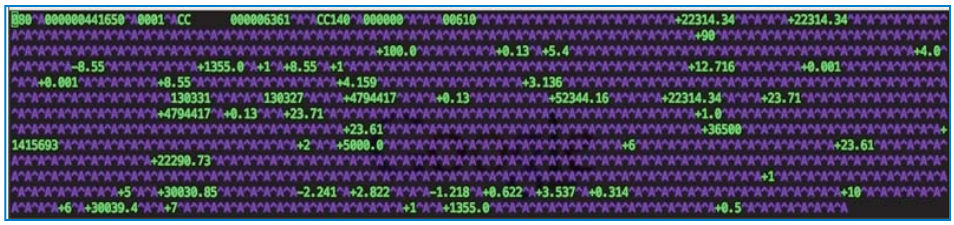

Figure 8. Optimized Records within Hadoop

Table 1. Records Extracted with HIVE Query

N047	N051	N002	N056	N071
627901	121004	221213		290800
		221213	654123	200600
627901		221213		

Table 2. Time to Import and Optimize Data

Filename	Approx. Size (GB)	Millions of Records	No. Columns	Copy Time from Mainframe (Seconds)	Conversion Time (Seconds)	Optimization Time (Seconds)	Total Time (Minutes)
00A011	10	18	109	300	300	75	11
00A021	15	25	464	480	480	118	8
00A025R	10	21	822	300	300	62	11

Table 2 shows how the time to import convert and optimize the data dropped from approximately six hours to between 11 and 18 minutes.

Results

The goal of the pilot was to see if Hadoop could, in the short-term, replace the RDBMS with which this bank runs important communications to the central bank and, long-term, replace other data warehouses.

The bank gave the team some queries to be run on HIVE on the data previously loaded. The bank's IT team has broad SQL* knowledge and, therefore, wanted to use SQL statements as much as possible.

The team ran three queries:

Query 1

- **RDBMS:** select distinct tp_ndg as N010 from scontrpf50m0130331
- **Hive:** SELECT DISTINCT N010 FROM 00A011 LIMIT10;

Query 2

- **RDBMS:** SELECT DISTINCT B. COD_UO, b. Tp_conto, a. Tp_ndg as N010 FROM A JOIN SCONTRPF50M0130331 CUBOM0100M0130331 B ON A. NDG = B. NDG WHERE POSIZ_SOFF_INCAGLT = '1 'and a. TP_NDG in ('DIN', 'IOC', 'SPF') and b. dt_accs_rapprt> = '2013-03-01 'and b. dt_accs_rapprt <= '2013-03-15' ORDER BY B. COD_UO, b. Tp_conto, a. TP_NDG
- **HIVE:** SELECT DISTINCT B.DOOR, B.TP_INCOME, A.N010 FROM 00A011 A JOIN CUBOM0100M0130331 B ON A.NDG = B.NDG WHERE N011 ='1' AND A.N010 in ('DIN', 'IOC', 'SPF') AND B.R021 > '130100' AND B.R021 < '130316'; ORDER BY DOOR, TP_INCOME, N010;

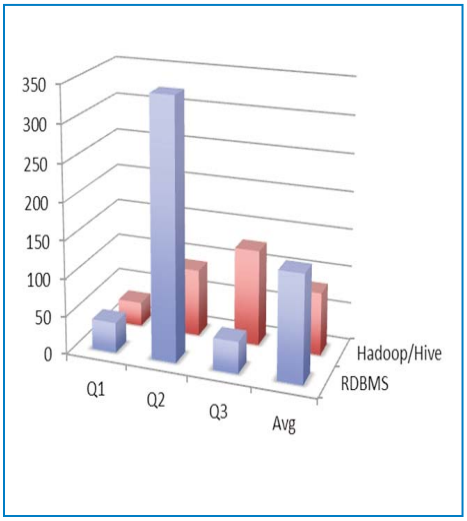

Figure 9. Execution Tme for Three Typical Queries

Query 3

- **RDBMS:** select b. cod_uo, b. forma_tec, TP_NDG as N010, substring (a. sae_rae, 1, 3) as N003, a. u_segmgest_2004 as N088, a. u_modserv_gest as N089, sum (b. qc_rata_scd) as D505 from scontrpf50m0130331 to join cubom0100m0130331 b on a. ndg = b. ndg where a. TP_NDG in ('DIN', 'IOC', 'SPF') and b. forma_tec in ('MW500', 'MW100', 'MW200') group by b. cod_uo, b. forma_tec, a. TP_NDG, substring (a. sae_rae, 1, 3), a. u_segmgest_2004, a. u_modserv_gest order by cod_uo, forma_tec, TP_NDG, substring (a. sae_rae, 1, 3), a. u_segmgest_2004, a. u_modserv_gest

Table 3 and Figure 9 show the time to execute and give results back for each of the three queries comparing execution time and results between a traditional RDBMS and Intel Distribution for Apache Hadoop software HIVE query.

After one and one-half months, the team was able to prove that Hadoop is a great fit for the

Table 3. Execution Tme for Three Typical Queries

Query	RDBMS	Hadoop/ Hive
1	40	32
2	343	89
3	42	127
Average	**141.66**	**82.66**

types of workloads the bank runs, able to match the performance of its existing system and perform even faster in extracting the data. At the same time, the new solution provides much more flexibility, reduces load times, and dramatically reduces the bank's overall costs.

The flexibility and the velocity with which the pilot was able to solve some of the bank's long-standing issues proved that Hadoop is the right solution for these types of workloads. The bank is now investigating whether other applications and datasets could benefit from a Hadoop infrastructure.

Flexible and Cost-Effective Platform

Consider whether the same regulations apply to your institution and your central bank. This step could begin an evolution toward a flexible architecture complementary to your existing RDBMS solution. Although architected for big data, Hadoop allows data analysis and predictive modeling on a more flexible and cost-effective platform.

For more information, contact Marco Righini (**marco.righini@intel.com**) for technical questions and Paolo Ossola (**paolo.ossola@intel.com**) for business-related questions.

Appendix: Glossary

Hive

Hive is a data warehouse system for Hadoop that facilitates easy data summarization, ad-hoc queries, and the analysis of large datasets stored in Hadoop-compatible file systems. Hive provides a mechanism to project structure onto this data and query the data using a SQL-like language called HiveQL*. At the same time this language also allows traditional map/reduce programmers to plug in their custom mappers and reducers when it is inconvenient or inefficient to express this logic in HiveQL.

Oozie

Oozie is a workflow scheduler system to manage Apache Hadoop jobs.

Oozie Workflow jobs are directed acyclical graphs (DAGs) of actions.

Oozie Coordinator jobs are recurrent Oozie Workflow jobs triggered by time (frequency) and data availability.Oozie is integrated with the rest of the Hadoop stack supporting several types of Hadoop jobs out of the box (e.g., Java* map-reduce, streaming map-reduce, Pig*, Hive, Sqoop*, and Distcp*) as well as system-specific jobs (e.g., Java programs and shell scripts).

Oozie is a scalable, reliable, and extensible system.

HBase

HBase is a column-oriented database management system that runs on top of Hadoop Distributed File System (HDFS). It is well suited for sparse data sets, which are common in many big data use cases. Unlike relational database systems, HBase does not support a structured query language like SQL*. In fact, HBase isn't a relational data store at all. HBase applications are written in Java, much like a typical MapReduce application.