The Financial Effect of Migration to NSPM Ecosystem

Reduce Time to Market and Reduce Cloud Service Bills in Data Engineering and AI Application Development

Ali HASSAN
Director, NSPM Academy
nspm-academy.fr
Paris, France
ali.hassan@nspm-academy.fr

I. Abstract

Financial aspects are deciding criteria for all business projects including bigdata and AI business. Expensive cloud bills and expensive software development costs are a challenge. A one-page SQL script on Databricks or Snowflake can result millions of dollars of annual cloud service bill when applied frequently to petabyte datalakes. In this paper we present the financial effect of migrating to NSPM ecosystem on these bills. Cloud service bills will be reduced 61.9% on average, software development costs will be reduced 82%[1], and migration to NSPM ecosystem from legacy datalake is 94% automatic using online tools.

To make our arguments clear, reproducible, and portable, we use TPC-H widely accepted datalake standard [3]. Also, we build on our previously published paper [7] that handle performance aspects. We do not want to decide your architecture and technology choices, we only invite you to think of NSPM ecosystem as a possible choice. Let's go!

II. Introduction

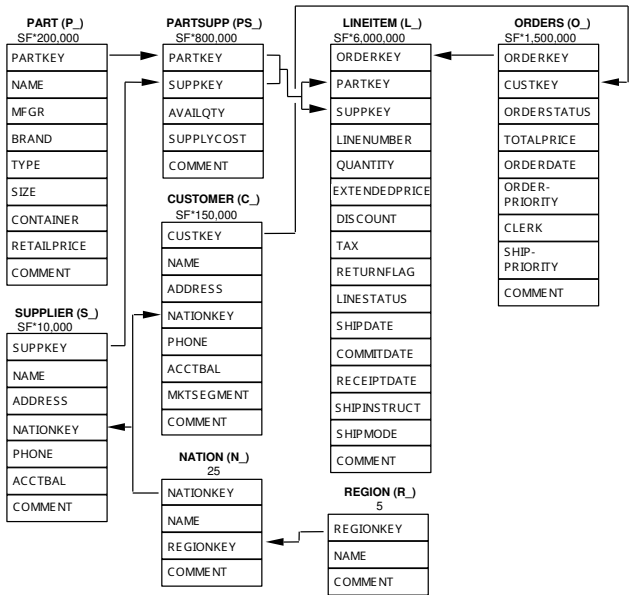
In figure 1 we have TPC-H DBMS SQL Schema [3] and in listing 1 we have associated DBMS SQL constraints. This is necessary and sufficient to define our DBMS datalake for all functional uses.

```
alter table Nation
    add CONSTRAINT fpk_nation_n_regionkey
    FOREIGN KEY (n_regionKey)
    REFERENCES Region (r_regionKey);
alter table Supplier
    add CONSTRAINT fpk_supplier_s_nationkey
    FOREIGN KEY (s_nationKey)
    REFERENCES Nation (n_nationKay);
alter table Customer
    add CONSTRAINT fpk_customer_c_nationkey
    FOREIGN KEY (c_nationKey)
    REFERENCES Nation (n_nationKay);
```

Ali Hassan is a Ph.D. holder from IMT Atlantique in distributed computing and expert in parallel computing including bigdata.

[1]We think of that as shorter time to market, since development time and development costs are interchangeable.

Figure 2: The TPC-H Schema



Legend:

- The parentheses following each table name contain the prefix of the column names for that table;
- The arrows point in the direction of the one-to-many relationships between tables;
- The number/formula below each table name represents the cardinality (number of rows) of the table. Some are factored by SF, the Scale Factor, to obtain the chosen database size. The cardinality for the LINEITEM table is approximate (see Clause 4.2.5).

Fig. 1. The TPC-H Schema: Database Entities, Relationships, and Characteristics. From [3].

```
alter table PartSupp
    add CONSTRAINT fpk_partsupp_ps_partkey
    FOREIGN KEY (ps_partKey)
    REFERENCES Part (p_partKey);
alter table PartSupp
    add CONSTRAINT fpk_partsupp_ps_suppkey
    FOREIGN KEY (ps_suppKey)
    REFERENCES Supplier (s_suppKey);
alter table LineItem
    add CONSTRAINT fpk_lineitem_l_orderkey
    FOREIGN KEY (l_orderKey)
    REFERENCES Orders (o_orderKey);
alter table LineItem
```

```
    add CONSTRAINT fpk_lineitem_l_partkey
    FOREIGN KEY (l_partKey)
    REFERENCES Part (p_partKey);
alter table LineItem
    add CONSTRAINT fpk_lineitem_l_suppkey
    FOREIGN KEY (l_suppKey)
    REFERENCES Supplier (s_suppKey);
alter table Orders
    add CONSTRAINT fpk_orders_o_custkey
    FOREIGN KEY (o_custKey)
    REFERENCES Customer (c_custKey);
```

Listing 1. The TPC-H SQL Table Constraints. From [3].

Now assuming that our datalake consist of these 8 tables, and assuming that the total size of the datalake is multi-hundreds of terabytes, how do we manage this datalake for transformations and analytics and research queries? It does not need special expertise to know that for any query on such datalake we need a join, a multi-terabyte join! This means: multi-terabyte shuffle data and less than 15% CPU saturation [7] and [4]. If you accept these conditions, you have a feasibility solution, however, you can have a performance solution described is [7]. We will not re-describe NSPM solution because it is already described in the mentioned reference along with its advantages, rather we will describe M2C tool, an automation tool in the ecosystem of NSPM model in the following two scenarios:

- migrate DBMS datalake to NSPM datalake
- develop scala spark transformations in all steps in bigdata ETL applications.

III. Migrate DBMS datalake to NSPM datalake

A widely available definition of datalake is the following: "A data lake is a system or repository of data stored in its natural/raw format." In big data community, a greate debate has started to understand the meaning of 'natural' and 'raw' in the previous definition, in the context of big data. The context of big data rais several challenges:

- size: data is measured by terabytes and petabytes [2]
- memory: data can not be stored on a single machine, neither ram nor disk, which forces distributed memory model. In other words: all single machine data structure and algorithms are out of scope of big data. This is not simple.

Is the format presented in the specification of TPC-H standard, which is presented in figure 1 and listing 1 natural? The answer is: no it is not natural, rather, it is traditional in the pre-bigdata era: DBMS format. We will call it DBMS datalake. In France, along with the majority of European counties, this is widely used bigdata format for data-lakes, and I dare to say, it is the only used format. It is borrowed from pre-bigdata era, and all computer science graduates can use it directly: SQL query is famous. Unfortunately, DBMS SQL format is not compatible with datalake for big data for many reasons described in [7] and [6], while NSPM model and data format is [7] and [6].

[2]10 petabytes – estimated approximate size of the Library of Congress's collection, including non-book materials, as of 2005. Size of the Internet Archive topped 10 PB in October 2013.

Assuming you have DBMS SQL datatable, as example TPC-H DBMS datalake described in this paper, M2C can help you migrate this datalake to NSPM format, as follows:

- Step 1: use M2C online service to automatically generate necessary schema and datalake migration spark code/-batch.
- Step 2: invoke datalake migration spark-batch

At the success of step 2, datalake migration is accomplished.

A. Datalake migration step 1

```
case class Nation(n_nationKay: Identifier,
                  n_name: String,
                  n_regionKey: Identifier,
                  n_Comment: String)

case class Region(r_regionKey: Identifier,
                  r_name: String,
                  r_comment: String)

case class Supplier(s_suppKey: Identifier,
                    s_name: String,
                    ... ,
                    s_comment: String)


...
case class LineItem(l_orderKey: Identifier,
                    l_partKey: Identifier,
                    ... ,
                    l_comment: String)
```

Listing 2. The TPC-H Schema in scala / spark format

M2C is an online tool that accepts files as input, and generates files at output:
Input:
SchemaDBMS.scala
TPC_H_Constraints.scala

Output:
LineItem_EncodeDecode.scala
LineItemWide.scala
LoadTablesDBMS.scala
LoadTablesWide.scala
NspmSchema.scala
TransformTables.scala

B. Datalake migration step 2

Step 2: invoke datalake migration spark batch generates in step 1 as in listing 3. That's all! More over, this is open source, ie you will receive scala spark code that you can understand and modify, nothing is black box in this process!

C. Datalake migration costs

In table II we provide precise statistics of number of lines of code necessary for this migration. We can see that we have 52 lines to be written manually by data engineers, and 756 lines of code that are generated automatically by M2C tool. That translates to 94% automatic code generation and 6% manual code development. And the 6% manual code is simply the legacy datalake DBMS schema and DBMS constraints. All the NSPM types, case classes, serialization, deserialization,

File Name	Description
NspmSchema.scala	This file contains the core of NSPM model, the corresponding NSPM schema of dbms schema, for example Lineitem table will have a corresponding Lineitem dataset in NSPM datalake with a schema that is automatically generated using M2C ecosystem.
LoadTablesWide.scala	This file contains All necessary code to load NSPM datasets from NSPM datalake. Do not forget that NSPM LIneitem for example, is serialized in LineitemWide parquet format. This serialization/deserialization requires encode/decode which is automatically generated and has negligible time and memory overhead.
LoadTablesDBMS.scala	This file contains all scala spark code necessary to load tables in DBMS TPC-H datalake. COde in this file contains all necessary type conversion for each column to reach dbms strong typed datasets.
LineItem_EncodeDecode.scala	This file contains trait LineItem_EncodeDecode. Declaration of develop all serialization/deserialization to HDFS, encode/decode, and construction mechanisms for this table.
LineItemWide.scala	This file contains case class LineItemWide, this is the format that is used internally to save NSPM LineItem dataset on disk using parquet file format. Implementation of all serialization/deserialization to HDFS, encode/decode, and construction mechanisms of this table.
TransformTables.scala	This file contains the datalake migration batch. The code in this file loads necessary dbms tables from TPC-H dbms datalake, applies necessary table joines that correspond to DBMS constraints provided in input file, the result is NSPM datasets, which is encoded and serialized to disk HDFS in parquet format. All NSPM serialzed datasets on HDFS form NSPM datalake.

TABLE I
DESCRIPTIONS OF M2C GENERATED OUTPUT FILES IN DATALAKE MIGRATION PHASE. THIS IS M2C FOR LINEITEM TABLE.

encode, decode, are automatically generated by the NSPM ecosystem. This is shown in figure 2.

```
implicit val session: SparkSession =
                        SparkIO.createSparkSession()
TransformTables.transformLineItem()
```

Listing 3. The code required to invoke datalake migration from DBMS to NSPM. This is Step 2 in datalake migration, which is the final step.

Category	Lines of Code	File Name
Input (Manual)	31	Tables.scala
Input (Manual)	21	SchemaDBMS.scala
Subtotal	**52**	—
Output (Auto)	39	LineItem_EncodeDecode.scala
Output (Auto)	253	LineItemWide.scala
Output (Auto)	157	LoadTablesDBMS.scala
Output (Auto)	114	LoadTablesWide.scala
Output (Auto)	92	NspmSchema.scala
Output (Auto)	95	TransformTables.scala
Output (Auto)	6	Types.scala
Total Output	**756**	**M2C Generated Total**

TABLE II
DATALAKE MIGRATION: M2C PROCESS: INPUT VS. OUTPUT FILE STATISTICS.

It is more important to mention that This phase is applied once! This is non-recurring batch for any project. You migrate legacy

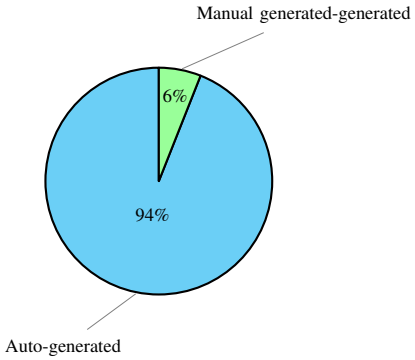


Fig. 2. Automatically generated code vs manually developed code necessary for TPC-H DBMS datalake migration to TPC-H NSPM datalake.

DBMS SQL based lake, such as databricks data lake [1] or snowflake datalake [2], you migrate it to NSPM datalake once, and after, your NSPM datalake will grow infinitely with no go back to DBMS architecture. Please see figure 3. I hope this is clear enough. The automation and cost-reduction mentioned in subsection III-C and figure 2 remove the barrier of datalake migration by automating schemas and scala spark migration code generation, this code that will be used only once. NSPM schemas that are generated in this phase will be used forever, though.

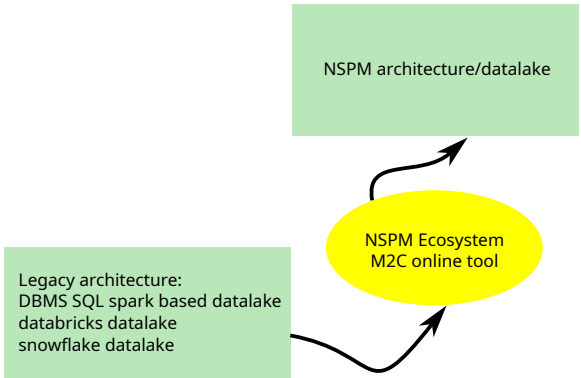


Fig. 3. Legacy datalake architecture including snowflake and databricks datalakes can be migrated to the modern NSPM architecture/datalake, this is also discussed in sub-section V-B. Moreover, the NSPM ecosystem automate this migration.

IV. DEVELOP SCALA SPARK TRANSFORMATIONS

Now that we have our NSPM datalake ready, we will start building transformations. As example, we will use Q100 query/transformation, benchmarked in [7]. We will go into details from the definition of Q100, code development: manual vs automatic, and costs.

A. Development of Q100 transformation

Starting with LineItem table from TPC-H standard, in its NSPM schema format `LineItem` in our new NSPM datalake populated in section III, create a new table with schema name `LineItem_SchemaNSPM_Q100`.

The new table `LineItem_SchemaNSPM_Q100` will contain all columns of `LineItem`, in addition we add three new columns as follows [7]:

1) column `l_local` it will contain isLocal, which will be true if customer nation name is identical to supplier nation name.
2) column `text` will concat all comment columns and put result in text column.
3) column `hash` will apply a text processing on text column calculated in step 2 In this case it is simple find string function, which results. export_true or export_false[3].

Finally, save transformation result to the efficient compressed parquet format. Steps required to develop this transformation, ie Q100, in NSPM ecosystem are three steps:

- build `LineItem_SchemaNSPM_Q100` case class structure.
- develop scala spark to map `LineItem_SchemaNSPM_Q100` data members to `LineItem` data members, directly of some columns, and calculate the 3 new columns.
- develop all serialization/deserialization to HDFS, encode/decode, and construction mechanisms.
- maximize the use of NSPM ecosystem to auto-generate code when applicable.

```
// Manually generated code

case class LineItem_SchemaNSPM_Q100(
         l_orderKey: Orders,
         l_partKey: Part,
         l_suppKey: Supplier,
         l_lineNumber: Long,
         l_quantity: Float,
         ....
         l_shipMode: String,
         l_comment: String,
         l_local: Boolean,
         text: String,
         hash: String
                     )

object LineItem_SchemaNSPM_Q100 {

  def apply(param: LineItem,
            isLocal: Boolean,
            text: String,
            hash: String):
  LineItem_SchemaNSPM_Q100 = {
    new LineItem_SchemaNSPM_Q100(
      param.l_orderKey,
      ....
      param.l_comment,
      isLocal,
      text,
      hash)
  } // closes method apply

} // closes object LineItem_SchemaNSPM_Q100
```

Listing 4. LineItem_SchemaNSPM_Q100 case class structure and construction. Section IV.

Now we apply M2C tool on input files to generate output files as follows:
Input:
LineItem_SchemaNSPM_Q100.scala

[3]In case we try to find the string *export*.

NspmSchema.scala

Output:
LineItem_SchemaNSPM_Q100_EncodeDecode.scala
LineItem_SchemaNSPM_Q100Wide.scala

File Name	Description
LineItem_SchemaNSPM_Q100_EncodeDecode.scala	This file contains trait LineItem_SchemaNSPM_Q100_EncodeDecode. Declaration of develop all serialization/deserialization to HDFS, encode/decode, and construction mechanisms for this table.
LineItem_SchemaNSPM_Q100Wide.scala	This file contains case class LineItem_SchemaNSPM_Q100Wide, this is the format that is used internally to save NSPM LineItem_SchemaNSPM_Q100 dataset on disk using parquet file format. Implementation of all serialization/deserialization to HDFS, encode/decode, and construction mechanisms of this table.

TABLE III
DESCRIPTIONS OF M2C GENERATED OUTPUT FILES IN Q100 TRANSFORMATION BUILDING.

Results of M2C step are presented in table III. And we are ready to compile and run the transformation Q100 implementation presented in listing 5. In this code we see that data engineer focus on the functional logic during development, while he/she depends on NSPM ecosystem for type safety and automatic code generation for all serialization/deserialization and encode/decode as in innocent lines 1 and 2. More over, this code imports the automatically generated code in section III.

```
// Code is manually generated
package fr.hp.nmp24.manual.generated.code.benchmark.queries
package query100

import fr.hp.nmp24.auto.generated.code.{IOWide, LineItem}
import fr.hp.nmp24.manual.generated.code.benchmark.queries
                                                .PerformanceShared
import org.apache.spark.sql.{Dataset, SparkSession}

object Invoker {
  /**
   * This is the NSPM version of query Q100
   */
  def wide_lineItem_Q100()(
    implicit session: SparkSession): Unit = {
    import session.implicits._

    // the following line is innocent line 1
    val lineitemNSPM: Dataset[LineItem] =
      IOWide.wide_loadLineItem("""/path/lineitem.parquet""")

    val lineItem_SchemaNSPM_Q100_DS:
      Dataset[LineItem_SchemaNSPM_Q100] =
      lineitemNSPM.map { item: LineItem =>
        val isLocal: Boolean = item
          .l_orderKey
          .o_custKey
          .c_nationKey
          .n_name == item
          .l_suppKey
          .s_nationKey
          .n_name
        val text: String = item.l_comment +
          item.l_orderKey.o_comment +
          item.l_orderKey.o_custKey.c_comment +
          item.l_orderKey.o_custKey.c_nationKey.n_Comment +
item.l_orderKey.o_custKey.c_nationKey.n_regionKey.r_comment +
          item.l_partKey.p_comment +
```

```
          item.l_suppKey.s_comment +
          item.l_suppKey.s_nationKey.n_Comment +
          item.l_suppKey.s_nationKey.n_regionKey.r_comment
        val hash: String = PerformanceShared.hashString(text)
        LineItem_SchemaNSPM_Q100(item, isLocal, text, hash)
      }
    // the following line is innocent line 2
    IOWide.wide_save_Q100(lineItem_SchemaNSPM_Q100_DS)
  }
}
```

Listing 5. Q100 transformation code. In this code we see that data engineer focus on the functional logic during development, while he/she depends on NSPM ecosystem for type safety and automatic code generation for all serialization/deserialization and encode/decode as in innocent lines 1 and 2. More over, this code imports the automatically generated code in section III.

B. Q100 development costs

In table IV we provide precise statistics of number of lines of code necessary for this migration. We can see that we have 152 lines to be written manually by data engineers, and 694 lines of code that are generated automatically by M2C tool. That translates to 82% automatic code generation and 18% manual code development. And the 18% manual code is simply the case class schema of Q100 new table and the functional logic to build it. This is valuable point: manual development is devoted to functional business logic, which reduces time to market severally. All the NSPM types, case classes, serialization, deserialization, encode, decode, are automatically generated by the NSPM ecosystem. This is shown in figure 4.

Category	Lines of Code	File Name
Input (Manual)	55	LineItem_SchemaNSPM_Q100.scala
Input (Manual)	43	Q100_transformation.scala
Input (Manual)	44	PerformanceShared.scala
Input (Manual)	10	OutputResults.scala
Subtotal	**152**	—
Output (Auto)	39	LineItem_SchemaNSPM_Q100_EncodeDecode.scala
Output (Auto)	271	LineItem_SchemaNSPM_Q100Wide.scala
Output (Auto)	39	LineItem_EncodeDecode.scala
Output (Auto)	253	LineItemWide.scala
Output (Auto)	92	NspmSchema.scala
Total Output	**694**	**M2C Generated Total**

TABLE IV
Q100 TRANSFORMATION DEVELOPMENT: M2C PROCESS: INPUT VS. OUTPUT FILE STATISTICS. PERCENTAGE OF AUTOMATICALLY GENERATED CODE IS SHOWN IN 4.

V. COST REDUCTION

In the context of bigdata applications, NSPM provide many advantages including:

- reduce time to market
- reduce cloud service bill

This is an outstanding cost reduction. We will discuss in details.

A. Reduce time to market

Figures 2 and 4 speak for themselves. When you only need to develop 18% of the code manually and 82% will be automatically generated, that means you have more time to concentrate on improving logic, testing, bug fixing. All of that translates directly to less time to market. Migration from legacy data-lakes to NSPM datalake happen once, with only 6% manual code development. If this is not efficient, then I do not know what efficient is!

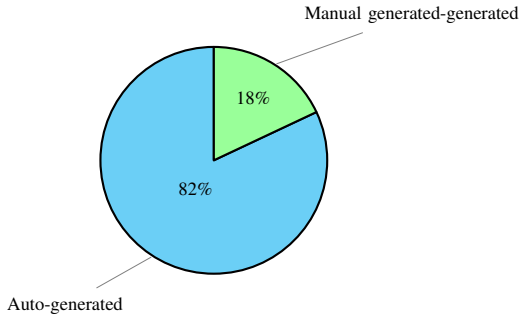


Fig. 4. Automatically generated code vs manually developed code necessary for Q100 transformation development.

B. Reduce cloud service bill

In cloud services, time means money, ie time of execution means more expensive bills. Using results from [7], specifically tables II and IV we get table V.

Transformation	DBMS-SQL-Spark - Time elapsed in minutes	NSPM-Spark - Time elapsed in minutes
Q100	78	15
Q105	21	12

TABLE V
EXECUTION TIMES FOR Q100 AND Q105 WITHOUT NSPM IN THE SECOND COLUMN, AND WITH NSPM IN THE THIRD COLUMN. IN BOTH CASES WE USE SCALA SPARK APPLICATION. THIS BENCHMARK IS EXECUTED ON OUR USED CLUSTER DESCRIBED IN SECTION IV PAPER [7]. NUMBER OF EXECUTORS USED FOR ALL CASES IS 12. FROM [7]. PLEASE NOTE THAT SECOND COLUMN REPRESENTS LEGACY ARCHITECTURE PRESENTED IN FIGURE 3, IE INCLUDING SNOWFLAKE AND DATABRICKS.

Cloud billing is calculated as:
$Cost = NumberofMachines \times Time \times HourlyRate$. For Q100 **Time Reduction: 80.77%**, ie cloud srvice bill reduction is **80.77%**.
For Q105 **Time Reduction: 42.86%**, ie cloud srvice bill reduction is **42.86%**.
Please see figure 5 and table VI. This reduction applies to all cloud service providers: Amazon, Azure, GCP, and OVH. Amazon, Azure, GCP, and OVH billing is based on resource consumption, which will increase in a linear or above-linear correlation with batch runtime and frequency.
The kind reader might notice that this huge time difference is directly related to the intelligent NSPM-architecture which succeeded to reduce shuffle to the minimum if not to zero. Not for Q100 or Q105, but for all practical applications, and this is discussed in [7] and in articles published on [5]. Snowflake [2], Databricks [1], Spark SQL library, and similar cloud services severally use data shuffle in an uncontrollable way, and they are equal in this context, and they are all based on spark SQL library. Please see figure 3. All optimization such as broadcast

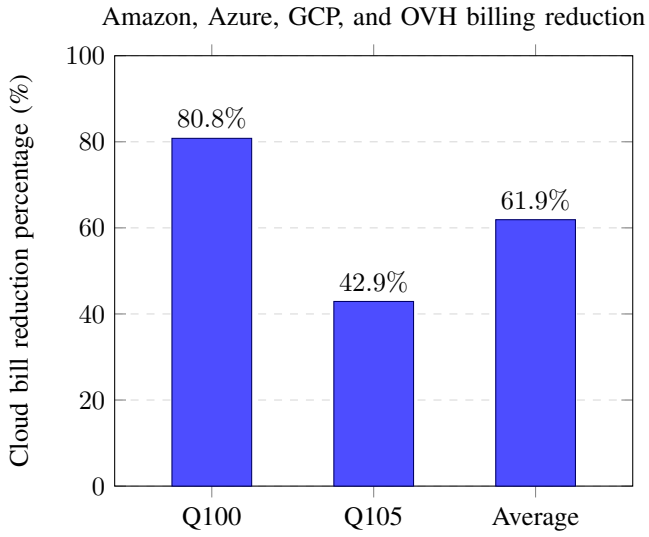


Fig. 5. Amazon, Azure, GCP, and OVH billing reduction when using NSPM ecosystem. Data is in table VI.

Transformation	DBMS-SQL-Spark - Time elapsed (min)	NSPM-Spark - Time elapsed (min)	Bill Reduction (%)
Q100	78	15	**80.8%**
Q105	21	12	**42.9%**
Statistical Average	—	—	**61.9%**

TABLE VI
EXECUTION TIMES AND REDUCTION RATES FOR Q100 AS BEST CASE AND Q105 AS WORST CASE AND AVERAGE CASE. THIS IS EXTENSION OF TABLE V. PLEASE NOTE THAT SECOND COLUMN REPRESENTS LEGACY ARCHITECTURE PRESENTED IN FIGURE 3, IE INCLUDING SNOWFLAKE AND DATABRICKS.

and filter before join are special case optimization. NSPM is the only theory-based optimization published for bigdata transformation until now.

C. Cost reduction example in dollars

To contextualize the potential impact of the NSPM Ecosystem, we examine three industry leaders whose business models rely heavily on traditional Big Data, data warehousing, and large-scale Spark/ETL workloads.

1. Lyft (Ride-Sharing & ETL)

Lyft represents a cloud-native architecture running massive Spark-based ETL and machine learning pipelines to calculate dynamic pricing and driver matching in real-time.

- **Provider:** Amazon Web Services (AWS).
- **Financial Commitment:** In its 2019 IPO filings, Lyft committed to an aggregate spend of **$300 million** on AWS between 2019 and 2021.
- **Annualized Impact:** This reflects a minimum baseline of **$100 million per year** dedicated to the data processing power required to sustain their transport network.

2. Airbnb (Data Warehousing & Analytics)

Airbnb utilizes a massive data lake and Big Data applications to power search rankings, fraud detection, and global analytics.

- **Provider:** Amazon Web Services (AWS).
- **Cloud Commitment:** As of 2019/2020, Airbnb's cloud commitments reached approximately **$1.2 billion** over a five-year term.
- **Workload Focus:** Significant portions of this budget—averaging **$240 million per year**—support internal Spark/Hadoop clusters and massive S3-based data warehousing.

3. Capital One (Financial Big Data)

A pioneer in financial digital transformation, Capital One closed all physical data centers to run intensive Big Data applications for risk modeling and fraud prevention on the cloud.

- **Provider:** Amazon Web Services (AWS).
- **Estimated Expenditure:** Industry analysts estimate Capital One's annual AWS spend in the range of **$250 million to $300 million**.
- **Transformation Profile:** Their workloads are heavy on SQL-like Spark jobs for regulatory reporting and risk assessment, mirroring the Q100 and Q105 transformation profiles.

The following figure 6 summarizes the financial effect of migration to NSPM ecosystem on the cloud service bill for the three companies Lyft (Ride-Sharing & ETL), Airbnb (Data Warehousing & Analytics), and Capital One (Financial Big Data).

VI. CONCLUSION

Reducing the cloud service bill you company annually pays from 240 Millions USD to 91.44 Millions USD is not marginal. Moreover, reducing software development costs by 82% is not only a financial gain but also a boost to your

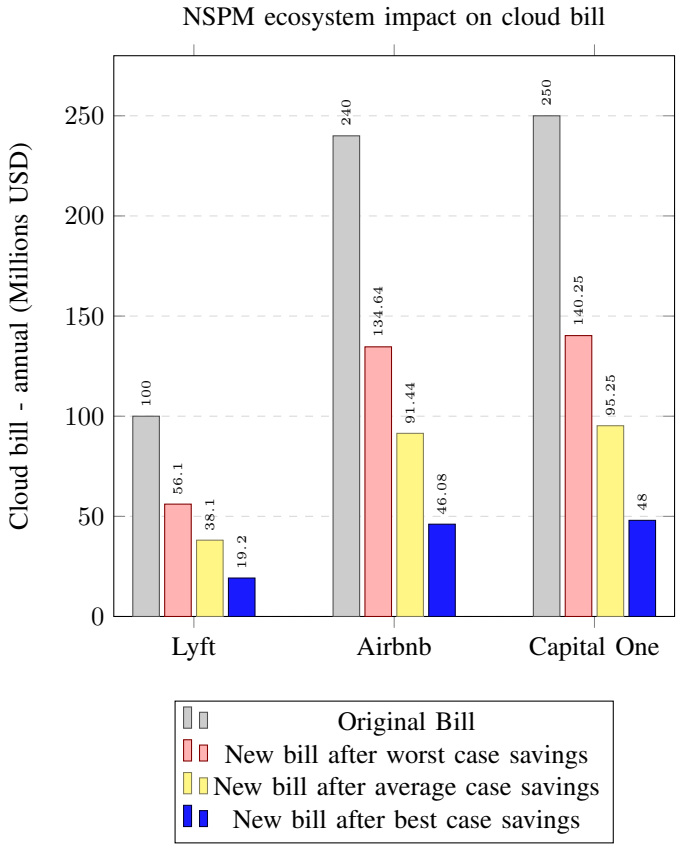


Fig. 6. Expected cloud bill after migration to NSPM ecosystem for three companies: Lyft (Ride-Sharing & ETL), Airbnb (Data Warehousing & Analytics), and Capital One (Financial Big Data). The original bill is extracted from published records. Then cost reduction is used from table VI to calculate new cloud bill value for best, average, and worst case.

time-to-market planing. Startups are equally interested in these financial aspects, same as medium sized companies and large companies. We do not want to decide your architecture and technology choices, we only invite you to think of NSPM ecosystem as a possible choice.

REFERENCES

[1] Databricks. https://en.wikipedia.org/wiki/Databricks. From Wikipedia, the free encyclopedia.

[2] Snowflake inc. https://en.wikipedia.org/wiki/Snowflake_Inc. From Wikipedia, the free encyclopedia.

[3] Tpc benchmarktm h standard specification revision 3.0.1. http://www.tpc.org. Transaction Processing Performance Council (TPC) Presidio of San Francisco © 1993 - 2022 Transaction Processing Performance Council.

[4] Ali Hassan. Datalake m2c. https://nspm-academy.fr/index.php/datalake-m2c/. Director, NSPM Academy - Paris, France © 2025 October.

[5] Ali Hassan. Nspm ecosystem articles. https://nspm-academy.fr/index.php/articles/. Director, NSPM Academy - Paris, France © 2025 May.

[6] Ali Hassan. Spark performance. https://amzn.eu/d/6ZUj4PZ. Amazon Book Publication - ISBN-13: 979-8285051671 - 2025.

[7] Ali Hassan. Tpc-h benchmark using novel spark performance model - nspm. nspm-academy.fr. Director, NSPM Academy - Paris, France © 2025 October.