Mindtree
A Larsen & Toubro Group Company

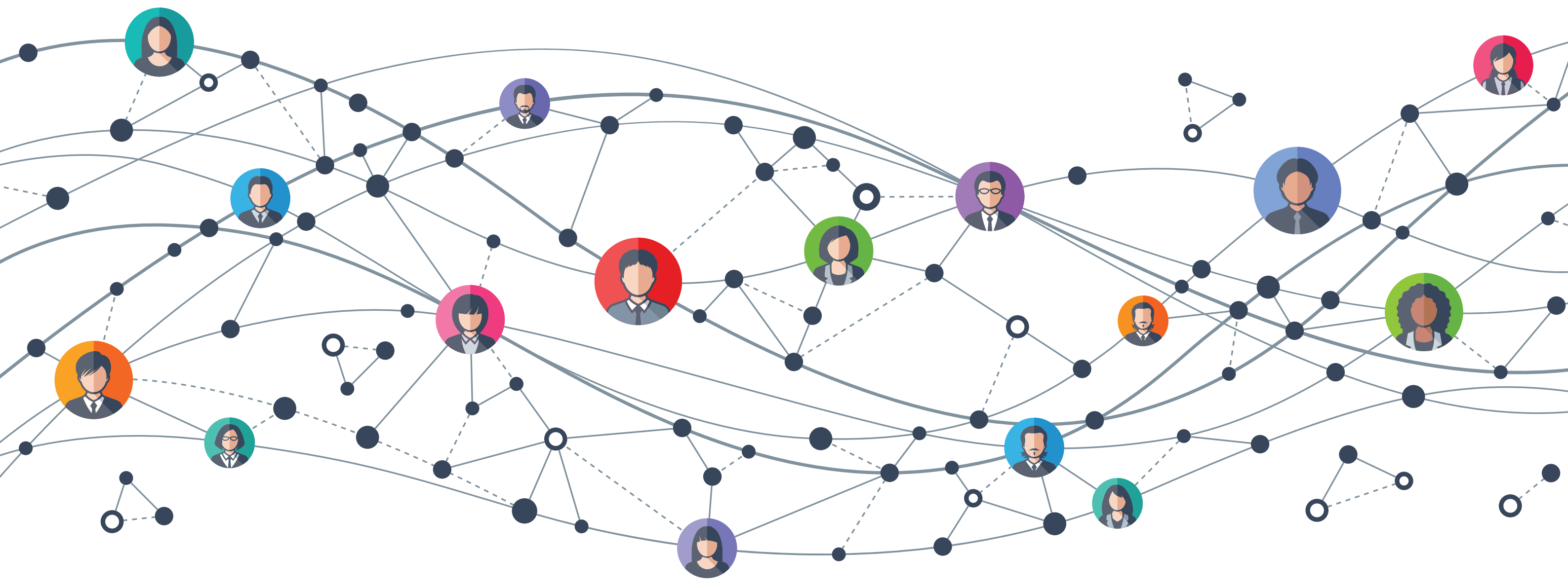
Six Steps to Modernize your Data Ecosystem to make Collaborative Intelligence Possible

Manoj Karanth
Sowjanyakumar Kothapalli

Welcome to possible

A Mindtree White paper | 2020

Table of Contents



Organizations across the world are striving towards being more data-driven in their decision-making. The right mix of human and machine intelligence is crucial for organizations to succeed in this journey. Machine intelligence needs to be supported with the right data infrastructure, and organizations have invested in setting up the same with the likes of data lakes, data warehouses etc.

At the same time, these investments have not quite provided the outcome that organizations had expected. A common set of challenges that organizations have faced are:

Business value of insights: Choosing the right use cases and KPIs which could generate valuable insights for the business has always been a challenge.

Time to insight: While Big Data improved the capability to process data faster, organizations have proceeded to crunch more data. However, the availability of the data in time remains a key aim.

Cost per insight: Once the data is gathered in the system, a big challenge in making it available for other teams to use is the cost. Big data environments guzzle a lot of computing power which increase the cost of the environment specially in the cloud.

This has led organizations to take a re-look at their data estates and look to address these challenges. Over the years, technologies in the Big Data landscape have continued to change with Spark emerging as the de-facto processing mechanism for data needs. These technologies alleviate the limitations in first-generation big data systems built with Apache Hadoop-based systems with distributions like Cloudera, Hortonworks etc.

In this paper, we highlight how one can approach this modernization path. We have identified Databricks on AWS as the target environment. New generation data platforms are unified i.e. we have the same stack for batch, streaming, machine learning. We have chosen Databricks since it is best performing Spark engine and is the leading player in bringing unified platforms to life

The modernization approach is composed of the following steps

Separation of Compute and Storage

Workload Type and Resource Usage

Data Processing and Data Store Optimization

Design Consumption Landscape

Assess Security and Governance

Data Migration and Movement

Separation of Compute and Storage

One of the founding principles in Hadoop was that for data processing to be scaled horizontally, compute had to be moved to where the storage resided. This would reduce the load on network I/O transfer and make the systems truly distributed. To process the data efficiently, these machines or nodes would have high memory and CPU requirements.

When we transported the same concept to cloud-based environments, the cost of running and scaling started becoming increasingly high. When one is running a data lake, most of the time one needs the storage and not the processing capacity. In Hadoop-based data environments, compute and storage are tied together with HDFS as the file system. E.g. In AWS, d2 is the most cost-efficient storage instance type

In the cloud, object storage is durable, reliable and cheap, while the network capabilities continue to increase. This led to a decoupling of compute and

storage, and most cloud-native data architectures are adopting this mode with object storage as the Data Lake. Modern data platforms like Databricks provide the elastic capability required to utilize the power of separating storage and compute.

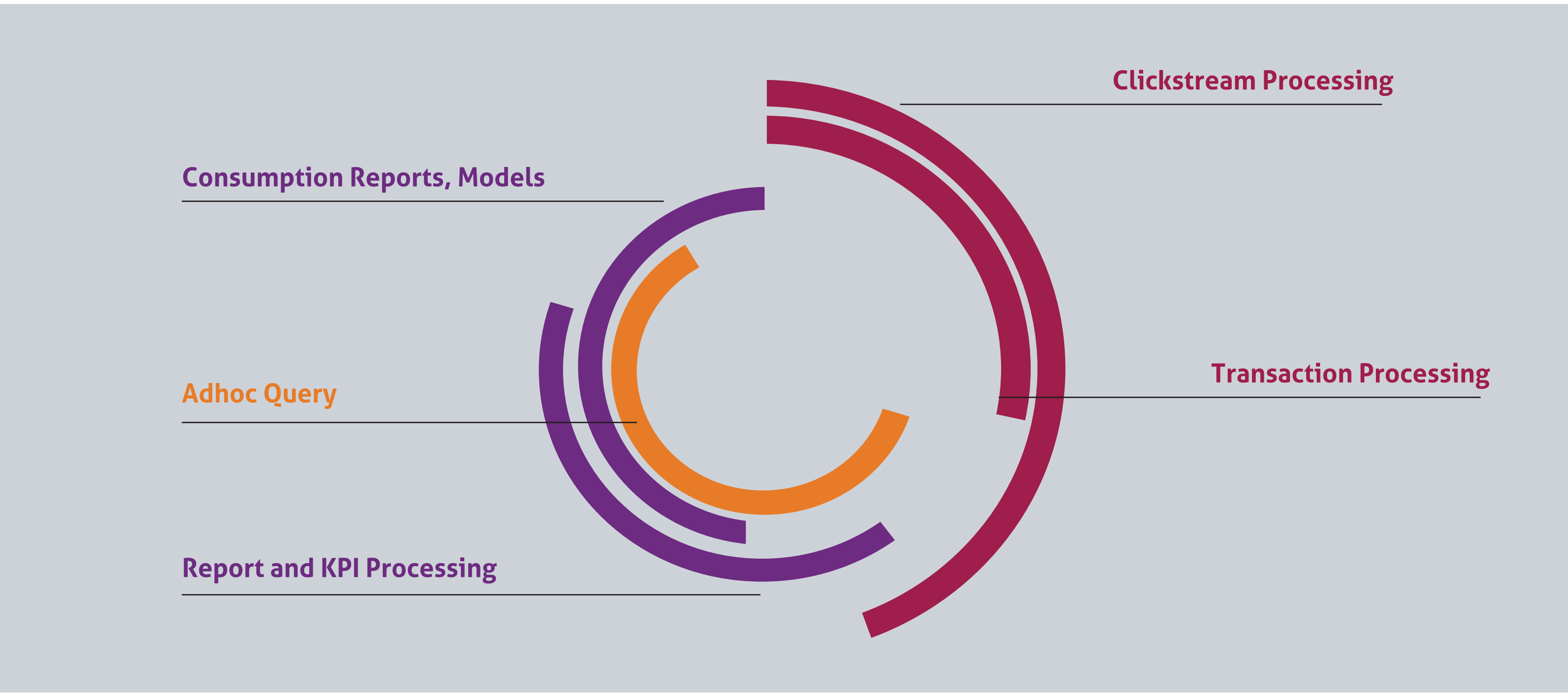
This has a significant impact on the cost as outlined in the example below taking AWS as an example to store 1PB of data using a 40-node cluster. The calculation was done using the logic that a M5a.8X large cluster will run for about 12 hours a day. This instance type cluster is slightly high, since in most practical cases, due to varied loads, much smaller instance type clusters with fewer nodes can be configured.

Hadoop	Databricks
48 TB per node * 40 d2.8x nodes = 5.52* 40 + 15*0.78 other	**Storage: S3 cost = 25K** **Processing: 40 node M5a.8x large** **Hosting (40 * 0.867x360 hours) = 12.5K** **DBU cost (40* 0.4*4.91*360) = 28.5K**
100K USD	**66K USD**

Illustrative Cost Comparison (50% utilization)

Assess Workload Type

In addition to separation of storage and compute, data processing time and cost can be optimized through an understanding of the workloads. This impacts both time to insight and cost per insight. In Hadoop-based environments, multiple workloads run on the same cluster to optimize the spend. Hence, it is important to assess the different workloads and their most efficient processing environments. Following is an example of different workloads



Illustrative Workload Distribution on a 24-Hour Time Scale

- Batch-based data ingestion workloads: These are usually ingested in fixed time intervals. Here too, the workloads are varied.
  - Input data like clickstream usually have a lot of JSON which requires memory bound processing.
  - Batch processing of typical structured data have more crunching which is usually more CPU bound.
- Continuous data ingestion in small data streams. Some examples are live transaction data which again could vary between memory and CPU bound processing. However, the required capacity is much lesser.
- The real reason behind the data ingestion is to derive insights. Therefore, we have a lot of processing for which we require data aggregation and summary calculations. This has a lot of memory processing.
- Along with this, we have workloads for report calculations, predictive algorithm data processing, ad hoc queries etc.

While one could optimize it at the time of production, over time, the usage patterns change. Data ingestion increases with addition of new data sources. This puts additional pressure on the platform. The increased data processing also requires more time for KPI calculations, leading to contention in resources, in turn also limiting the time and resource available for ad hoc queries. E.g. One Hive query could hog the entire cluster.

As a result, customers experience both under-utilized capacity and a capacity-crunch because of fluctuating demand. It is also not easy to scale up or down on demand due to the specific machine requirements (e.g. EC2 instance types with ephemeral storage as an additional cost). This means that new environments cannot be brought up quickly.

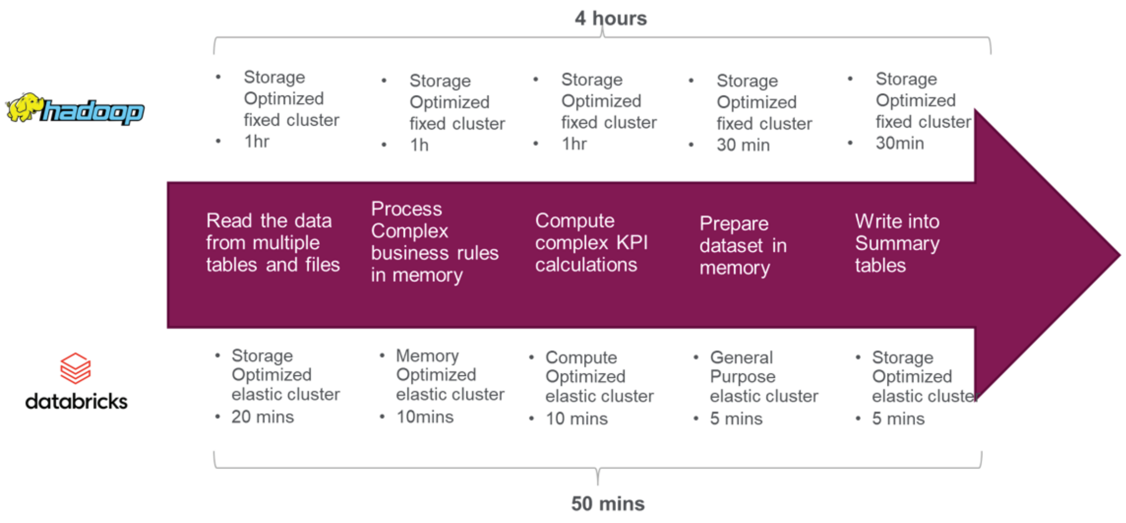
One of the promises of new data technology on the cloud is serverless and on-demand data infrastructure. Separating the workload types and processing times helps us plan for the same. If we look at Databricks, which is designed for running spark effectively in the cloud, we see a built-in cluster manager, which provides features like auto scaling and auto termination. Databricks also provides connectors with cloud storage and integrated notebook environment which helps with development immensely.

Given the power of auto scaling and auto termination, one could design the environment more optimally for the workload types as outlined in the example.

Workload Types and Cluster Configurations

Workload Type	Cluster Type	Cost Benefit
Memory Bound Data Ingestion Jobs	Run these workloads in their own cluster which is a combination of on-demand and spot instances. Use instances which are memory optimized. Shut down once the job is done	Spot instances help keep the cost down. No issues with noisy neighbours
CPU-intensive Data Ingestion Jobs	Similar to the earlier case, but on a cluster with compute optimized instances	Same as above
Ad hoc Queries	The workload requests are varied across different departments. Using a shared cluster designed for auto-scaling helps manage the varied demand	Having auto-scaling instead of a fixed cluster helps keep the cost down and maintain the balance between on-demand and spot instances
Reports, KPI Crunching, Statistical Models	Depending on the usage being time-bound, one can design a specific cluster with pre-built libraries for easier configuration	This provides more control based on the job. E.g. One can run a complete cluster based on spot instances for a data exploration workload

In addition to cost, using the right cluster sizes and types leads to a decrease in processing time. The following example highlights the same.



4 hours
hadoop
• Storage Optimized fixed cluster • 1hr
• Storage Optimized fixed cluster • 1h
• Storage Optimized fixed cluster • 1hr
• Storage Optimized fixed cluster • 30 min
• Storage Optimized fixed cluster • 30min
Read the data from multiple tables and files
Process Complex business rules in memory
Compute complex KPI calculations
Prepare dataset in memory
Write into Summary tables
databricks
• Storage Optimized elastic cluster • 20 mins
• Memory Optimized elastic cluster • 10mins
• Compute Optimized elastic cluster • 10 mins
• General Purpose elastic cluster • 5 mins
• Storage Optimized elastic cluster • 5 mins
50 mins

Performance Difference between Data Environments

Data Processing and Data Store optimization

While data processing was substantially improved through cluster and workload optimization, further improvements can be made by looking at the data stores and intermediate data processing. Typical Big Data tools have limitations with respect to source/sinks & type of data processing - whether batch or streaming. Hadoop does not integrate well with multiple cloud source/sinks e.g. Data Warehouse, NoSQL databases. This leads to multiple tools being used with an additional workflow (Oozie, Step Functions, Data Pipeline etc.) on top. This can cause non-optimized code, as data needs to be written to an intermediate storage multiple times. Also, there may be delays as developers with disparate skill sets need to collaborate.

This is best highlighted by the dual use of HBase and Hive as the data store formats. HBase is used primarily for updating dimensions and Hive tables for appending transactions. While HBase is write-optimized, it isn't as query-friendly as Hive, which is read-optimized. Therefore, most systems have both transaction and report stores. Typically, this leads to data in HBase getting converted to Hive through a complex intermediate staging layer. Additionally, hive tables stored on top of parquet files perform very badly if they need to read many small files. Hence, ingesting data from streaming applications needs an additional administrative task of merging small files. This increases the administrative complexity (e.g. merging small files), while increasing the data processing time.

Having a common data store and processing (data management system) can greatly alleviate this pain of multiple processing technologies and data stores. This starts with agreeing on a common open file format for data storage. Today, Parquet has emerged as the most common used format, since it is better optimized for fast query and data compression.

With Databricks, we have delta which is a unified data management system that fits this need. HBase and Hive external tables can be replaced with a unified table, Delta (Parquet-based), which is read-optimized. The ACID merge features ensure that the performance of read on HBase tables is comparable to Hive tables. Most importantly, we don't have to convert the HBase tables into Hive tables for downstream analysis.

The solution also enables RDBMS features such as ACID transactions, UPDATE/Merge and DELETE. Elegant OPTIMIZE/VACUUM is available for the consolidation/update of small part files. This makes it easier to clean up or correct bad data at a record level. This also means that we don't have to run additional administrative tasks like merging small files which translates to redirecting the available resources for this purpose. Additionally, data cleanup of Hive tables is easy.

From a performance perspective in our experience, we have seen significant improvements in read and write speeds across Hive and HBase.

- Improvement in Hive table reads by about 30-40% on Databricks delta post tuning with techniques like ZOrdering (co-locate related information in the same set of files) while reads on HBase tables saw 70-80% improvement.
- In Hive, 60 -70% improvement in inserts was observed while updates are almost same speed as HBase

An additional complexity that sometimes arises is when both batch and stream data sets need to be processed together. Often, this needs to be done using different tools which bring their own challenges. Spark and more specifically Databricks can provide a unified API which can handle both batch & streaming data sources.

Taken together, this makes the data processing pipeline more performant and much easier to develop and maintain. Along the way, there are a few nice touches that Databricks provides to further improve the processing speed and improve productivity. Some of these are

- Data caching to improve query and processing speeds
- Schema enforcement and Schema Evolution, which help manage data changes and evolutions more effectively.
- Time-travel: Databricks Delta automatically versions the Big Data that is stored in the data lake allowing one to access any version of that data. This allows for audit and roll back data in case of accidental bad writes or deletes to its original value. Multiple versions of data can be accessed using either the timestamp or version number. This is similar to temporal tables in Amazon RDS for SQL Server, and was missing from Big Data systems.

Design Consumption Landscape

The reason for the data infrastructure is to drive insights. Therefore, the consumption layer which includes the analytical and reporting store becomes extremely important. This feeds the reporting layer, data APIs, machine learning APIs among others. There are two primary modes of consumption we need to focus on. These are SQL engine performance and Machine Learning workspaces.

SQL

In traditional Hadoop-based architectures, Hive is largely the analytical layer with queries, followed by HBase in some scenarios. As the performance improvement need increases, we also need the presence of data-marts and data-warehouses.

It is in this context that we need to view the evolution of Spark SQL. While Spark was initially only a compute platform, today, it is a fully functioning SQL engine capable of interfacing with the JDBC engine. As stated by Spark, Spark SQL is designed to be compatible with the Hive Metastore, SerDes and UDFs.

We saw an increased usage of Cloud Datawarehouse like Snowflake, Azure Synapse, AWS Redshift among others. A common theme among these architectures is an increased focus on the outcome as against the design. i.e. The focus is on SQL query performance. e.g. AWS Redshift now has Aqua which focuses on query performance, so does serverless querying on Azure Synapse and Google Big Query. However, for the purpose of this paper, the key insight is that SQL continues to be strong and has emerged as the most important language for insight generation.

Databricks Delta, in addition to providing ACID compliant tables, also provides faster query execution with indexing, statistics, and auto-caching support. Like other organizations, query performance continues to be a key theme, with the recent introduction of dynamic file pruning, which increases query performance by 2x to 8x. This will be followed by a faster JDBC driver going by the Databricks public roadmap.

Therefore, since the choices have evolved, any modernization of the data environment will require the evaluation of these platforms, based on the consumption needs.

Machine Learning

Commercial distributions of Hadoop ship their own Machine Learning workbench, which allow for secure and collaborative data science workloads. However, these collaboration mechanisms are proprietary and not based on open standards.

The dominant standard today in MLflow: MLflow brings in the discipline of DevOps to the Machine Learning world. This helps us track the experiments, code and model repositories, and experimentation to deployment along with an integrated notebook environment. Databricks and AWS provide a couple of options to integrate MLflow in the Machine Learning workflow.

Databricks Machine Learning Runtime (MLR) provides scalable clusters that supports popular frameworks like Keras, Tensorflow, PyTorch, SparkML and Scikit-learn. MLR enables data scientists and ML practitioners to rapidly build models using its Auto-ML capabilities. Managed MLflow can help manage MDLC (Model Development Life Cycle) like experimentation, deployment, and model repository. MLR also supports MLeap and the portability of models across platforms and flexible deployments on docker containers, SageMaker and other cloud provider platforms.

Assess Security and Governance

This is an often-overlooked part of the assessment process. The security and operational fitness for Hadoop environments was designed without the cloud in mind. Cloud-based Data Lake offerings have evolved from HDFS-compatible cloud distributions to native cloud Data Lakes Should be built on proven object storage. This allows for data organization based on finer grained time scale partitions, and richer retention and control policies with seamless identity and role propagation for data zones.

Current data platforms like Databricks on Cloud therefore use the security and operational harness provided by the cloud providers. With respect to AWS, Databricks provides controls like IAM credential pass-through to integrate with the AWS ecosystem. Other AWS principles like VPC peering, PrivateLink and Policy enforcement only add to this.

Along with this, from a data governance stand-point, we need an integration with data catalogues. On the AWS platform, the natural integration is with AWS Glue. Databricks can leverage Glue as the meta-store, even across multiple workspaces. All the metadata can reside in one data catalog, easily accessible across their data lake which can be accessed from entire data lake. One advantage of keeping all the metadata in Glue is that it can be leveraged by other tools in the AWS stack, e.g. Athena & CloudWatch etc. Having a single meta-store across all AWS resources brings in significant operational efficiencies while designing enterprise ETL and reporting, as one doesn't have to sync multiple meta-stores, and can query AWS Glue using powerful built in APIs additionally.

Data Migration

Based on our experience, data migration from on-premise Hadoop-backed Data Lake to Databricks on Cloud needs to be planned and executed across multiple areas. The data estate includes HDFS Files, Hive and HBase tables etc.

Feature and control structure mapping, rationalization of data sets, choice of right migration strategy among one-time full refresh, incremental copy, parallel run and optional sync are key blocks in migration planning. A well-defined and battle-tested Audit-Balance-Control framework and associated task lists provide guidance for clean data migration execution. Following is a detail of the two main approaches.

- **One-time full refresh:**In this approach, parquet files for Hive tables can be moved as is into S3 (Object Storage). We can create external tables on this data and load them into Databricks delta. However, if you have dimensional type of data in HBase, you have to first convert it into Hive and then move the data into S3 for loading data into delta tables.

- **Incremental loads:** Incremental loads can be achieved by using the timestamp of the record creation date. Using this timestamp, we should get the data as of that day and write it into parquet files on S3. Subsequently, the steps outlined above will remain the same.

Preserving DDLs and Schema is a best practice as Databricks and Delta use Hive meta-store to persist table metadata. This not only makes the migration easier specially for Hive tables, but also helps in the migration of security policies associated with a particular column/table. The processing for data loading into Delta itself can be made faster using multiple clusters, thereby increasing the parallelism.

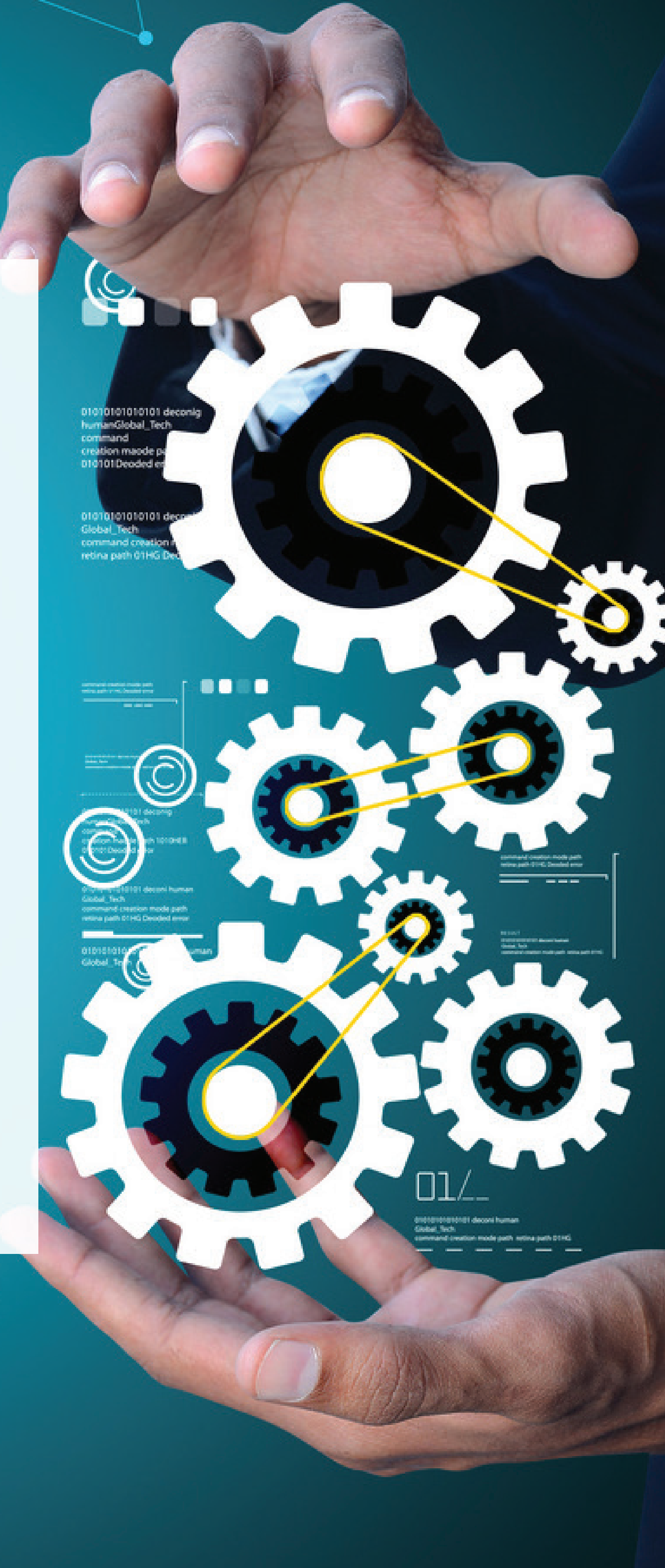
The data pipelines can be ported or rewritten depending on the tools used. Any RDBMS data ingestion pipelines created using Sqoop can be replaced easily by Spark jobs, as Spark can ingest from a JDBC source and offer similar scaling benefits. Any 'Legacy' MapReduce job should be rewritten to take numerous advantages offered by Spark. Cloud-managed orchestration tools are recommended for complex workflow management, while Databricks Jobs API can be used for simple workflows.

Hive workloads can be migrated to SparkSQL with minimal changes, thanks to SparkSQL's high affinity to Hive and support for UDFs like Hive. Serving Layer (BI Tool Landscape) can be provided very well by built-in connections as well as JDBC/ODBC connectors.

What does Mindtree bring to the table?

Our core philosophy is that data by itself does not inspire action. We need the right mix of human and machine intelligence for real world solutions and insights. Unless data infrastructure allows enterprise consumers to experiment and access the data, this goal cannot be achieved. We have worked with Global Top 1000 organizations in achieving their data modernization journey.

We bring these experiences to modernize your data environments with certainty. Our accelerators housed under 'Decision Moments,' backed by the partnership with the right technology partners accelerates this journey. Together, we can decrease the cost per insight, increase the value through the identification of right business use cases and improve the time to insight.



About Authors

Manoj Karanth

Global Head- Data science and Engineering, Mindtree

With an industry experience of 20+ years, Manoj Karanth heads the Cloud, Data Science and Engineering (Data Engineering, ML, AI) teams globally for Mindtree's Digital Business. He works with Global 2000 clients to accelerate their cloud-first journey and become insight-driven businesses using Mindtree's cloud-native technology platforms. Manoj has built the technology center of excellence with leading cloud and data analytics partners to deliver differentiated digital solutions. He has lived cloud and product engineering through his two decade career

Sowjanyakumar Kothapalli

Principal- Big Data Architect, Mindtree

Sowjanyakumar Kothapalli is a Program Architect in our digital practice and has 24 years of experience working with fortune 500 companies; of which 15 years were spent in building complex data platforms across various industries like infrastructure management, structured finance, credit risk management & retail

References

https://docs.databricks.com/administration-guide/capacity-planning/cmbp.html

https://databricks.com/blog/2017/05/31/top-5-reasons-for-choosing-s3-over-hdfs.html

https://www.datanami.com/2018/05/16/big-data-file-formats-demystified/

https://databricks.com/blog/2017/10/25/databricks-delta-a-unified-management-system-for-real-time-big-data.html

https://databricks.com/blog/2019/09/24/diving-into-delta-lake-schema-enforcement-evolution.html

https://www.knowledgelens.com/assets/migrating-to-data-lake-3_0.pdf

https://spark.apache.org/docs/latest/sql-migration-guide-hive-compatibility.html

https://databricks.com/blog/2020/04/30/faster-sql-queries-on-delta-lake-with-dynamic-file-pruning.html

https://docs.aws.amazon.com/whitepapers/latest/building-data-lakes/amazon-s3-data-lake-storage-platform.html

https://docs.azuredatabricks.net/_static/notebooks/delta/optimize-python.html

https://spark.apache.org/docs/latest/sql-migration-guide-hive-compatibility.html

https://databricks.com/blog/2020/03/16/security-that-unblocks.html