Dell In-Memory Appliance for Cloudera Enterprise

Spark Technology Overview and Streaming Workload Use Cases

Author: Armando Acosta
Hadoop Product Manager/Subject Matter Expert
Armando_Acosta@Dell.com/ Twitter- @Armando75

Overview

This paper is intended to provide an overview of the Dell In-Memory Appliance for Cloudera Enterprise with Spark support. This paper is intended for Hadoop users and customers looking for a better understanding of the Dell In-Memory Appliance, built on Cloudera Enterprise with Spark, as well as an outline of the use cases.

The paper will cover a high-level technology overview of Spark and streaming workload use cases. This paper is not a solution sizing document or deployment guide for the Dell In-Memory Appliance for Cloudera Enterprise.

Customers with interest in deploying a proof of concept (POC) can utilize the Dell Solution Centers. The Dell Solution Centers are a global network of connected labs that enable Dell customers to architect, validate and build solutions. Customers may contact their account team to have them submit a request to take advantage of these free services.

Executive Summary

Data is the new currency and competitive differentiator, data is being created and consumed at rates never before seen. This is not unique to a single industry; it will affect all vertical markets. Customers are struggling to ingest, store, analyze, and build insights from all this data. As more connected devices and machines with embedded sensors proliferate throughout the world, this will create even greater challenges for customers. This new type of streaming data is causing customer's needs to change based on new use cases and the need to analyze data in as fast and as efficient a manner as possible, and within a short window of time, and on a continuous basis.

Customers are evolving, they want to ingest and analyze data in a short window of time. Hadoop traditionally uses MapReduce as the processing framework; MapReduce is batch processing. Running batch jobs can take minutes or hours to complete. Batch processing was suitable for the types of data customers were analyzing, yet with the proliferation of connected devices, sensors and smarter, connected devices this has changed. Customers need to speed up their processing based on these new live streams of data that require a new modern manner of processing.

Customers need a processing engine to be more efficient and have the ability to process data in seconds. Spark is an alternative to the traditional batch MapReduce model and Spark can be used for streaming data processing and fast interactive queries that finish within seconds.

Additionally, data analysis is a complex process made up of many steps and various tools that allow users to handle different types of analysis. In order to accomplish the work they need to complete, businesses build a workflow comprised of many tools that slow down productivity because the users must provide the translation necessary to give context to each different tool. These extra steps dramatically affect productivity, as well as the speed to deliver the results a business needs in order to make immediate, proactive decisions.

Dell, together with Cloudera and Intel, want to help customers solve this problem with a turnkey, purpose built in-memory advanced analytics data platform. The Dell In-Memory Appliance for Cloudera Enterprise represents a unique collaboration of partners within the big data ecosystem. Together Dell, Cloudera and Intel deliver both the platform and the software to help enterprises capitalize on high-performance data analysis by leveraging the Cloudera Enterprise in-memory features (Spark) for interactive analytics and multiple types of workloads. Cloudera Enterprise also features Impala for fast query and Cloudera Search for interactive search. The result – one tool for both processing and analyzing streaming workloads – this simplifies the customer workflow. This one tool reduces the amount of time spent writing code to help translate workflow the between different tools.

Apache Spark

The Apache Spark project is a fast and general engine for large-scale data processing and interactive analysis. Spark was created out of the Berkley AMP Lab and is focused around speed, ease of use, and sophisticated analytics. The Berkley AMP Lab Team had three goals for Spark: 1. Speed: Build an Open Source processing engine with speed, 2. Easy: Make Spark easy to use - write applications faster, and

Integrate analytics giving user's one tool for data processing and analysis. 3. Generality: Combine SQL, streaming, and complex analytics. As a result, Spark allows users to do in-memory computing resulting in programs running up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk. Spark powers a stack of high-level tools including Spark SQL, MLlib for machine learning, GraphX, and Spark Streaming. You can combine these frameworks seamlessly in the same application.

Spark runs in Hadoop clusters through Hadoop YARN or Spark's standalone mode, and it can process data in HDFS, HBase, Cassandra, Hive, and any Hadoop InputFormat. It is designed to perform both general data processing (similar to MapReduce) and new workloads like streaming, interactive queries, and machine learning. Spark supports Scala, Java and Python programming languages.

Spark can process data efficiently using Spark Streaming. Many applications need the ability to process and analyze not only batch data, but also streams of new data in a short window of time. Running on top of Spark, Spark Streaming enables powerful interactive and analytical applications across both streaming and historical data, while inheriting Spark's ease of use and fault tolerance characteristics. It readily integrates with a wide variety of popular data sources, including HDFS, Flume, Kafka, and Twitter.

Spark enables a user to cache a data set, or an intermediate result, in the memory across cluster nodes, performing iterative computations faster than with Hadoop MapReduce and allowing for interactive analysis. Spark's application programming interfaces (APIs) also allow a user to express distributed computations—not only map and reduce but also filtering for subsets of data, joining multiple data sets together and more—all as operations over Resilient Distributed Datasets (RDDs), and all with less code to write. This abstraction enables simpler, more concise implementations and performance improvements, especially for complex sequences of operations. Other scripting languages and abstractions simplify MapReduce jobs (e.g., Pig and Cascading) but are built on the Hadoop MapReduce engine and thus lack the advantages of in-memory data sharing. Another advantage of Spark's APIs, in particular the Python API, is compatibility with existing libraries for scientific computing and visualization.

RDDs are built with fault tolerance; RDDs can automatically recover from failures. Individual RDDs track transformations used to build a lineage graph. The lineage graph is used to rerun in failed operations and to reconstruct any lost partitions. As a result there won't be a need to replicate data, saving time that would have been spent writing data over the network. Additionally, if a piece of data is lost from a partition because of node failure in the cluster, Spark will automatically start rebuilding by applying a filter to the corresponding block on that HDFS file.

Spark Technologies

Resilient Distributed Datasets (RDD)

RDD technology is a core piece of the Spark architecture.

Let's begin with background on how MapReduce works as this will help in the understanding of RDDs.

MapReduce is a computing model that divides a large computation into two steps: a 'map' step, in which data are partitioned and analyzed in parallel, and a 'reduce' step, where intermediate results are combined or summarized. During each step, data must be loaded from disk for each operation, which can slow iterative computations (including many machine-learning algorithms), and makes interactive, exploratory analysis difficult.

Spark extends and generalizes the MapReduce model, by introducing a concept called a resilient distributed data set (RDD) creating a memory caching layer. A user can cache a data set, or an intermediate result, in the memory across cluster nodes. RDDs are efficient for applications that reuse intermediate results across multiple computations. For example, data reuse is common in many iterative machine learning and graph algorithms, including PageRank, K-means clustering, and logistic regression. Another compelling use case is interactive data mining, where a user runs multiple adhoc queries on the same subset of the data.

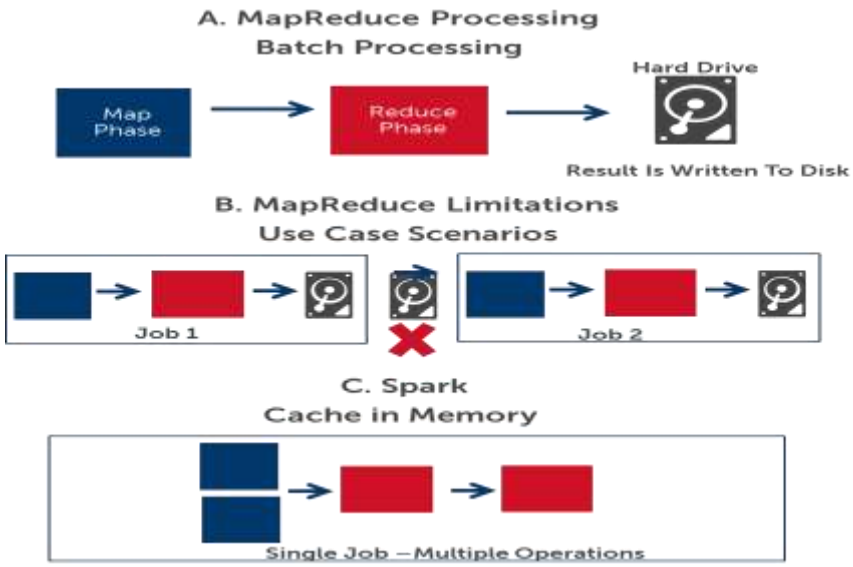
Figure 3 provides an example in how jobs are executed in MapReduce versus Spark.

A. MapReduce jobs-after reduce phase, the result is written to disk, if the result needs be re-used it must be pulled up from disk, very inefficient for certain use cases.

B. Pipelined MapReduce Jobs (Multiple MapReduce Jobs Pipelined Together)- each job is unaware of other jobs resulting in un-optimized jobs since it has to pull data from disk for every job in the pipeline. This causes unnecessary data replication misusing network resources.

C. Spark Jobs- multiple operations in a single job, data is cached in-memory via higher level operations provided by the Spark API.

Figure 3: MapReduce and Spark Processing



Spark API

Spark exposes RDDs through a language-integrated API, where each dataset is represented as an object and transformations are invoked using various methods on these objects. Spark lets users quickly write applications in Java, Scala, or Python. It comes with a built-in set of over 80 high-level operators that allow users to interactively query data within the Scala or Python shell. Users create RDDs by applying operations called transformations (these transformations include map, filter, and groupBy), to data in a stable storage system, such as the Hadoop Distributed File System (HDFS). The high level Spark API and built-in operations allow users to write code more efficiently and with fewer lines of code.

There are two types of operations: transformations, which define a new dataset based on previous ones, and actions, which kick off a job to execute on a cluster. Programmers start by defining one or more RDDs through transformations on data from HDFS. Users can then use these RDDs in actions, which are operations that return a value to the application or export data to HDFS. Spark computes RDDs lazily the first time they are used in an action, so that it can pipeline transformations.

Figure 4 provides an example of how Spark computes job stages. Boxes with solid outlines are RDDs. Partitions are shaded rectangles, in black if they are already in memory. To run an action on RDD G, we build stages at wide dependencies and pipeline narrow transformations inside each stage. In this case, stage one's output RDD is already in RAM, so we run stage two and then three.

Stage one: an RDD-A is created in-memory to process a groupBy transformation resulting in RDD-B which is persisted in memory.

Stage two: a new RDD-C is created in-memory to process a map transformation resulting in a new RDD-D then pipelining an additional filter transformation that results in a new RDD-E.

Stage three calls for the result of stage 2, RDD-E to be joined with the result in stage one RDD-B, then joins the results in RDD-F.

This leads to fast execution of the job because each stage automatically pipelines the next function to the next stage and avoids having to go to disk because of in-memory data locality.

Figure 4: Example of How Spark Computes Job Stages

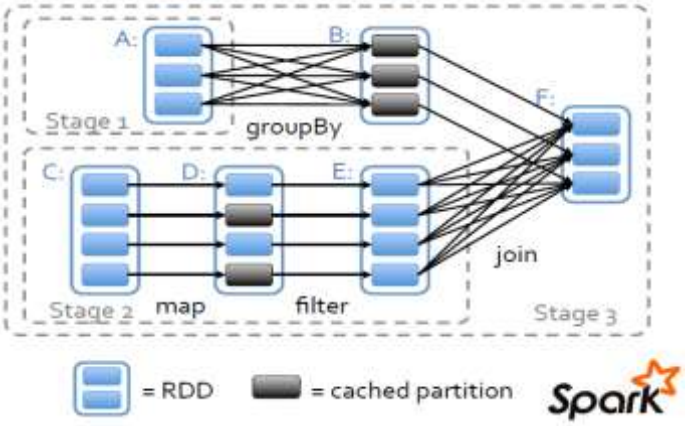


Table 2: Transformations and actions available on RDDs in Spark

Transformations	map- Return a new distributed dataset formed by passing each element of the source through a function func Filter- Return a new dataset formed by selecting those elements of the source on which func returns true. flatMap- Similar to map, but each input item can be mapped to 0 or more output items (so func should return a Seq rather than a single item) sample- Sample a fraction of the data, with or without replacement, using a given random number generator seed groupByKey- When called on a dataset of (K, V) pairs, returns a dataset of (K, Seq[V]) pairs reduceByKey- When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function union- Return a new dataset that contains the union of the elements in the source dataset and the argument join- When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key cogroup- When called on datasets of type (K, V) and (K, W), returns a dataset of (K, Seq[V], Seq[W]) tuples. sortByKey- When called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the oolean ascending argument
Actions	count- Return the number of elements in the dataset collect- Return all the elements of the dataset as an array at the driver program reduce- Aggregate the elements of the dataset using a function func (which takes two arguments and returns one) first- Return the first element of the dataset takeSample- Return an array with a random sample of num elements of the dataset, with or without replacement, using the given random number generator seed saveAsTextFile- Write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call toString on each element to convert it to a line of text in

	the file saveAsSequenceFile- Write the elements of the dataset as a Hadoop SequenceFile in a given path in the local filesystem, HDFS or any other Hadoop-supported file system countByKey- Only available on RDDs of type (K, V). Returns a `Map` of (K, Int) pairs with the count of each key foreach- Run a function func on each element of the dataset. This is usually done for side effects such as updating an accumulator variable (see below) or interacting with external storage systems

Spark Analytics

Spark provides a unique solution for customers because it provides not only an in-memory processing engine but, is also integrated with tools for different types of analysis.

Spark Streaming provides the ability to process and analyze not only batch data, but also streams of new data. It processes and analyzes that data interactively and on a continuous basis. Running on top of Spark, Spark Streaming enables powerful interactive and analytical applications across both streaming and historical data, while inheriting Spark's ease of use and fault tolerance characteristics. It readily integrates with a wide variety of popular data sources, including HDFS, Flume, Kafka, and Twitter.

Machine Learning: MLlib built on top of Spark, MLlib is a scalable machine learning library that delivers both high-quality algorithms and speed. The library is usable in Java, Scala, and Python as part of Spark applications, that way you can include it in the complete workflow.

Graph Processing: GraphX extends the distributed fault-tolerant collections, API and interactive console, of Spark with a new graph API which leverages recent advances in graph systems enabling users to easily and interactively build, transform, and reason graph structured data at scale.

The Spark interactive shell via Scala or Python, allows users the ability to quickly interact with data without having to write an application.

Dell In-Memory Appliance for Cloudera Enterprise Use Cases

The Dell In-Memory Appliance for Cloudera Enterprise provides a single platform for data processing and interactive analysis of streaming data. Spark makes business sense when customers have a need to analyze data interactively, and when the data set requires different types of analysis like interactive, iterative, graph, or streaming analysis. Typically the data sets are large and comprised of structured, semi-structured, and unstructured data. In addition the data streams into a customer environment on a continuous basis and needs to be analyzed in a small window of time in order to gain insights from the data.

Spark processing excels on workloads that require iterative computation or on complex pipelined jobs. The benefits of Spark are realized on iterative computation workloads because of in-memory processing combined with RDDs that allow intermediate results to be cached in-memory without having to go to disk. Machine Learning is an example of iterative computation; an algorithm is run iteratively across multiple computations over the same dataset. A prime use case is clustering, it takes items in a particular class and organizes them into naturally occurring groups, the items are bundled into the same group because of similarity, and this is beneficial for statistical analysis.

The benefits of Spark are recognized on complex pipelined jobs because of in-memory processing, combined with the high level Spark API, that write Spark jobs in multiple stages. Because each stage is aware of the next, the previous stage pipelines the next function while continuing in-memory thus leading to a faster execution of the job. Next, Spark is a fit for interactive data mining, in this workload multiple ad hoc queries are run over the same dataset.

The Dell In-Memory Appliance for Cloudera Enterprise also features Impala and Cloudera Search. Impala is a massively parallel processing (MPP) SQL query engine that runs natively in Apache Hadoop, enabling users to directly query data stored in HDFS and Apache HBase. This allows users to be able to quickly interact with the data, saving the user time and effort. This is very useful for analysis that requires fast query capabilities on large data sets. Cloudera Search allows users to do interactive search and navigated drill-down to help find relevant data across large, disparate data stores with mixed formats and structures. Users can discover the "shape of data" quickly and easily and expedite data modeling and result exploration

The next section will provide more specific use case examples with greater detail around the end-to-end solution. Each use case represents a hypothetical architecture with options for multiple ecosystem tools to build the solution – specific customer environments might require different technologies.

Use Cases

Streaming Log Aggregation and Analysis for Machine Logs allow customers to continually ingest log data from different data sources to process and analyze the data in a short window of time to provide proactive insights for faster decision making.

Customers can gain an understanding of all transactions in their environment by analyzing log data. Log data is valuable because it provides a definitive record of all user transactions, including customer behavior, sensor activity, machine behavior, security threats, or fraudulent activity.

In this example, infrastructure/operations management, users see how to proactively monitor IT to ensure uptime by rapidly pinpointing and resolving problems. The goal is to identify infrastructure relationships, establish baselines, and do predicative analysis by utilizing the Dell In-Memory Appliance for Cloudera Enterprise.

In this use case example, the customer is streaming in-machine logs from client, server, and network switches to process and analyze the logs to help predict system failure before it happens. The result, help limit exposure due to system downtime.

Data Sources:

Logs are created from client, server, and network switches. These logs are sent over the network to a data collection tool, examples- Kafka, Scribe, Fluent, Flume, Storm and Syslog.

Data Collection:

The data collection tool provides a messaging system that allows data to be collected and distributed into the in-memory spark appliance for data processing and analysis.

Steam Processing & Analytics:

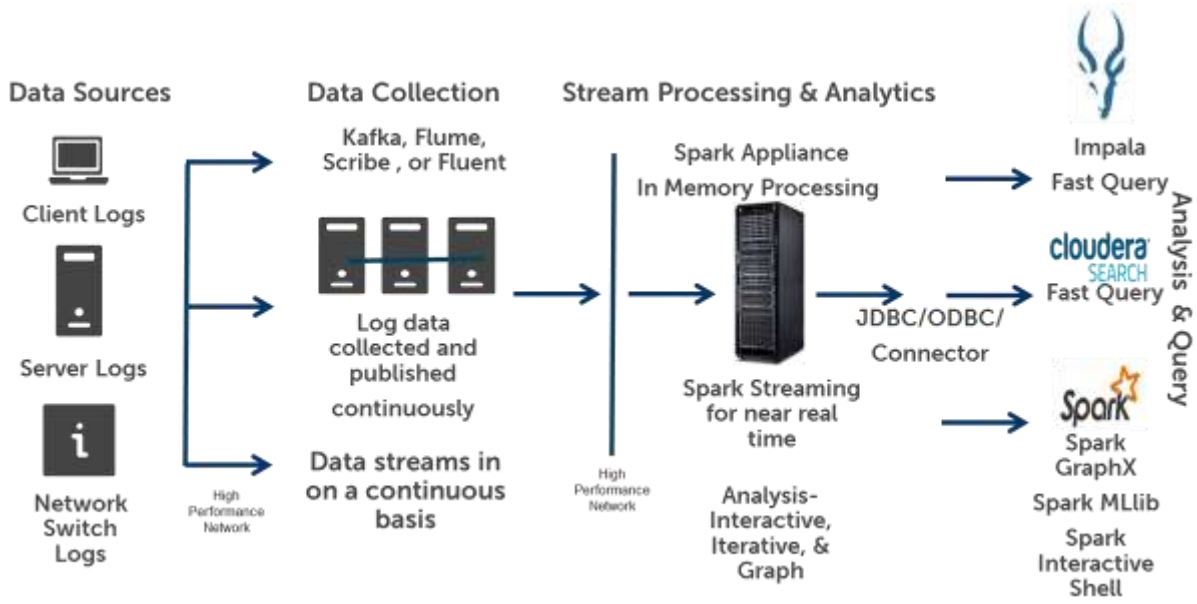
The Dell In-Memory Appliance for Cloudera Enterprise consumes the message data from the data collection tool. The in-memory appliance communicates with the data collection tool by either receiving or pulling data. The data is written to the Hadoop distributed file system for storage once the data is ready to be analyzed via spark streaming where it is put into memory for processing and analysis

During the analysis phase, data is transformed, joined, aggregated, and processed via algorithms based on a model created by data scientists and then written in scala or python by software developers.

Customers can use Impala for fast query, use Spark GraphX for graph processing analysis, use Spark MLlib to build machine learning algorithms, and use the Spark interactive shell for fast data access.

Figure 5 provides a high level architectural view of the solution as it breaks down by data source, data collection, streaming processing and analytics and it shows the different tools available for analysis & query.

Figure 5: Streaming Log Aggregation and Analysis of Machine Logs



Use Case:

Time series data on smart utility meters provides customers with points-in-time occurrences of an event as it happens. By analyzing event data, customers can identify trends that will allow users to better forecast future events. Time series data provides point-in-time event data as the events occur. Time series data is used to understand what is happening at a given point time and then is analyzed to look for trends to help predict future events. Smart utility meters with embedded sensors will send data to central points for data ingestion. The time series data is usually compromised of a time stamp, object ID, and event ID. The time series data can be collected by the second, minute, hour, or over days.

In this use case a utility company is collecting time series data from smart utility meters with embedded sensors. The utility company collects the time series data on an hourly basis. The utility company wants to collect time series data that includes a time stamp, event- electrical usage, and meter ID #. The utility company will analyze that data to compare utility usage on an hourly basis by street, block, neighborhood, city, district, and grid location. The utility company will look for trends and patterns to help them predict future events like spikes in power consumption and/or potential failures.

Data Sources:

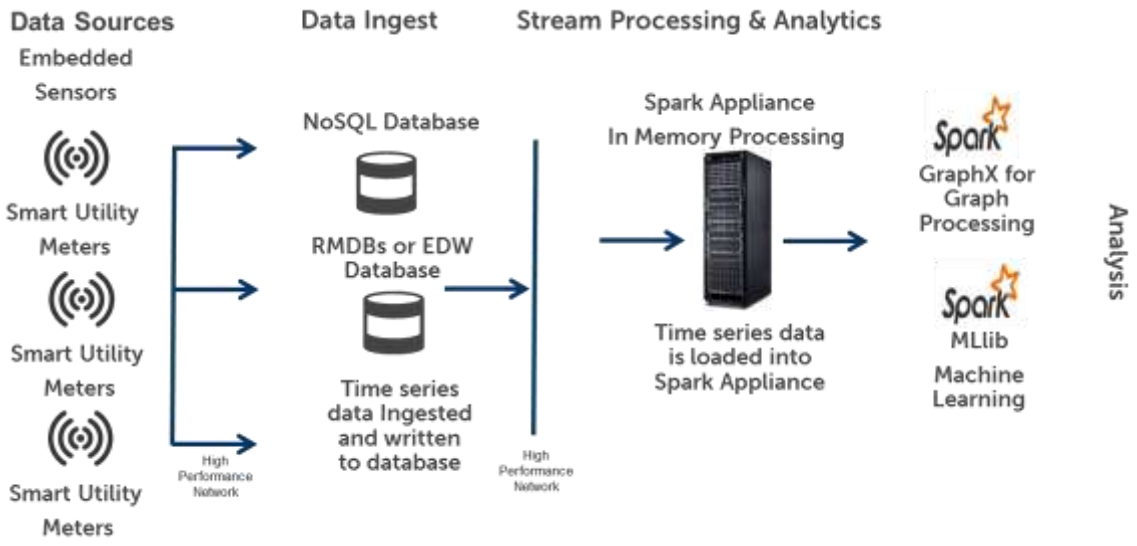
Data sources include smart meters attached to houses and business offices where embedded sensors send time series data to a central data center for processing and analysis.

Data Ingest: The time series data is sent over the network to a central data center where the data is typically written to a RMDBs/EDW or NoSQL DB (there are other options available, too), data can also be sent to a message system like Kafka or storm. From here the data is parsed and sent to the Spark appliance for processing and analysis. The data can be sent over to the appliance throughout the day in intervals of seconds, minutes, hours or days.

Stream Processing Analytics: The Dell In-Memory Appliance consumes the data for processing and analysis. The in-memory appliance communicates with the data ingest tool by either receiving or pulling data. The data is written to both a Hadoop distributed file system for storage and to memory to be analyzed via Spark streaming processing, to Spark Graphx for graph processing, or MLlib for machine learning analysis.

Figure 6 provides a high level architectural view of the solution broken down by data source, data ingest, streaming processing, and analytics, it then shows the different tools for analysis.

Figure 6: Time Series Data on Smart Utility Meters



Complex Extract, Transform, Load (ETL) jobs is the process by which raw data is migrated from data sources, transformed into a consumable and normalized format, then loaded into a target system for performing advanced analytics, analysis, and reporting.

In this use case, different data is sent to the Spark appliance where the data is transformed so that it can be processed and analyzed. The ETL jobs occur on a continuous basis as data changes in the different data sources. The data from different databases throughout the organization in different areas like – finance, procurement, operations, and sales – can be transformed or joined to help normalize the data for analysis. The goal is to join, group, and merge all of this data so that it can be analyzed for trends, relationships and patterns that can lead to valuable insights.

Data Sources:

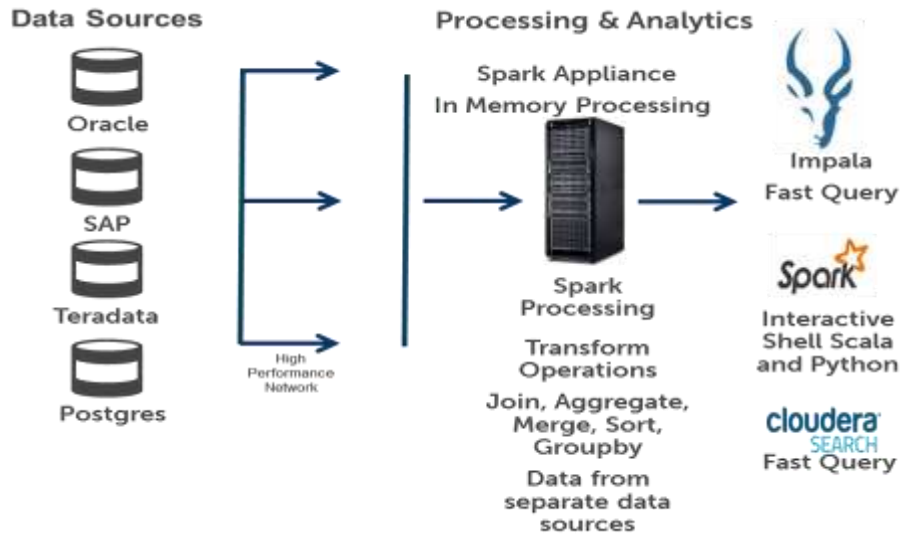
Data is stored in various databases across a customer environment and typically in different database technologies. These databases contain valuable data around customer transactions, financial reporting, product research, procurement data, customer data. The ETL process can take place in a dedicated RMDB or EDW, yet those systems are very expensive, they don't scale well, and they eat up precious resources needed for other database functions.

Processing & Analytics:

In this use case the Spark appliance would transform the data and then analyze/query the data with Impala and Spark Interactive Shell. The transform operations occur in memory. Spark API provide the capability to write complex ETL jobs with multiple stages and operations in a single job. This allows for faster and more efficient processing because the RDDs are persisted in-memory, after each operation/stage the results become a new cached partition in-memory allowing the operations to continue into stage two, three, and so on. This leads to fast execution of the job because each stage automatically pipelines the next function and to the next stage without having to go to disk.

Figure 7 provides a high level architectural view of the solution broken down by data source, data and processing, and analytics and includes the different tools available for analysis & query.

Figure 7: Time Series Data on Smart Utility Meters



Summary

To simplify the customer workflow, Dell, together with Cloudera and Intel, have collaborated to deliver this turnkey, purpose built in-memory advanced analytics data platform, the Dell In-Memory Appliance for Cloudera Enterprise. This one tool, reduces the amount of time spent writing code to help translate workflow between different tools and provides a solution that delivers quick answers to interactive queries for faster time to value.

For additional information contact us at: Hadoop@Dell.com.

SOURCES:

RESILIENT DISTRIBUTED DATASETS: A FAULT-TOLERANT ABSTRACTION FOR IN-MEMORY CLUSTER COMPUTING; MATEI ZAHARIA, MOSHARAF CHOWDHURY, TATHAGATA DAS, ANKUR DAVE, JUSTIN MA, MURPHY MCCAULEY, MICHAEL J. FRANKLIN, SCOTT SHENKER, ION STOICA UNIVERSITY OF CALIFORNIA, BERKELEY

FAST AND INTERACTIVE ANALYTICS OVER HADOOP DATA WITH SPARK; TATHAGATA DAS, ANKUR DAVE, JUSTIN MA, MURPHY MCCAULEY, MICHAEL J. FRANKLIN, SCOTT SHENKER, ION STOICA

DATABRICKS WEBSITE- ../AppData/Local/Microsoft/AppData/Local/Microsoft/Windows/Temporary Internet Files/Content.Outlook/AppData/Local/Microsoft/Windows/Temporary Internet Files/Content.Outlook/6C0JL4C2/www.databricks.com/spark

APACHE SPARK WEBSITE- www.spark.apache.org

CLOUDERA SPARK WEBCAST- SPARK, THE NEXT GENERATION OF MAPREDUCE