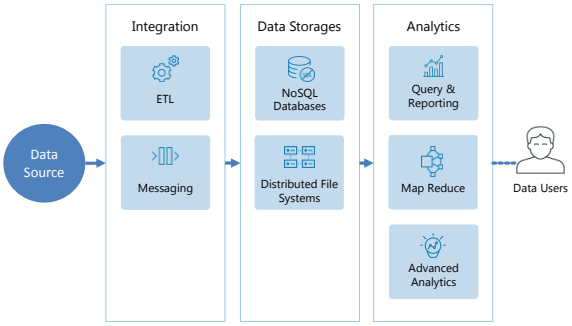
Pure Non-relational

Reference Architecture for Data Analytics

Description: This reference architecture does not rely on relational model principles. Often it is built on NoSQL storage, Hadoop, Search Engines and is highly effective for processing semi and unstructured data.



Consequences:

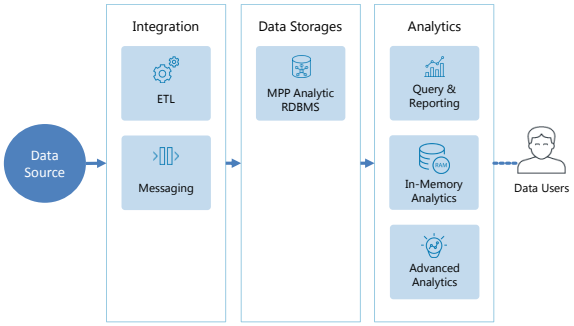
★★ **Ad-hoc analysis** - ad-hoc real-time query support is more difficult than in relational architecture
★★½ **Real-time analysis** – real-time with one-at-a-time processing
★★★ **Unstructured data processing** – supports easy storing and processing of semi and unstructured data
★★★ **Scalability** – can scale keeping petabytes
★★★ **Cost economy** – cost minimized due to open-source technologies

Sample implementations: Data Discovery, Data Lake, Operational Intelligence, Business Reporting

Extended Relational

Reference Architecture for Data Analytics

Description: Although this reference architecture is completely based on relational model principles and SQL-based DBMS, it intensively uses MPP and In-Memory techniques to improve scalability and extensibility.



Consequences:

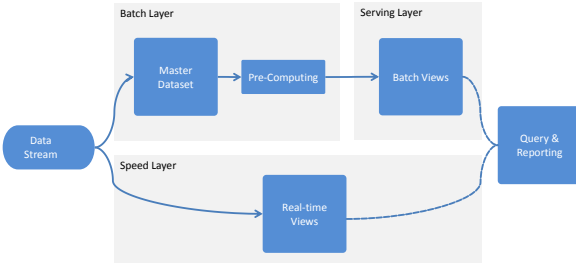
★★★ **Ad-hoc analysis** – supports complex ad-hoc real-time read queries
★★ **Real-time analysis** – near-real time with micro-batching technique
★★ **Unstructured data processing** – supports ingesting and querying semi-structured data such as JSON/XML
★★ **Scalability** – can run terabytes with MPP and clustering capabilities
★ **Cost economy** – MPP RDBMS license cost is quite expensive

Sample implementations: Business Reporting, Enterprise Data Warehousing, Data Discovery

Lambda Architecture (Hybrid)

Reference Architecture for Data Analytics

Description: This reference architecture enables real-time operational and historical analytics in the same solution. While the batch layer is based on non-relational techniques (usually Hadoop), the speed layer is based on streaming techniques to support strict real-time analytics requirements.



Consequences:

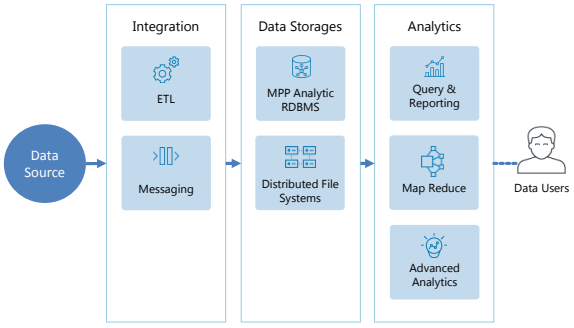
★★ ½ **Ad-hoc analysis** - ad-hoc real-time query support is more difficult than in relational architecture
★★★ **Real-time analysis** – streaming approach with low data latency
★★★ **Unstructured data processing** – supports processing of semi and unstructured data
★★★ **Scalability** – can scale keeping petabytes
★★★ **Cost economy** – cost minimized due to open-source technologies

Sample implementations: Real-time Intelligence, Data Discovery, Business Reporting

Data Refinery (Hybrid)

Reference Architecture for Data Analytics

Description: This reference architecture is a mix of relational and non-relational techniques. Non-relational part acts as an ETL to refine semi and unstructured data and load it cleansed into relational data warehouse for further analysis.



Consequences:

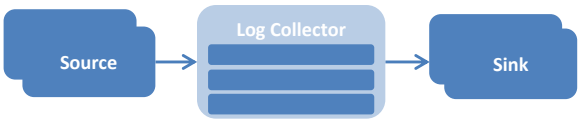
★★★ **Ad-hoc analysis** – supports complex ad-hoc real-time read queries
★ **Real-time analysis** – data latency is high due to batch processing
★★★ **Unstructured data processing** – supports processing of semi and unstructured data
★★ **Scalability** – can run terabytes with MPP and clustering capabilities
★ **Cost economy** – MPP RDBMS license cost is quite expensive

Sample implementations: Data Discovery, Business Reporting, Enterprise Data Warehousing

Data Collector

Family/Integration/Messaging

Description: This pattern aims to collect, aggregate and transfer log data for later use. Usually Data Collector implementations offer out of the box plug-ins for integrating with popular event sources and destinations.



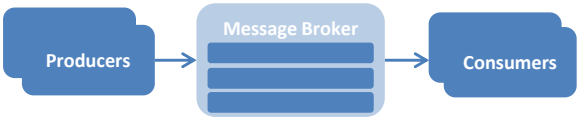
Consequences:
★★ **Performance –** can handle large amounts of data in real time
★★★ **Compatibility –** can be plugged with popular event sources and destinations
★★ **Flexibility –** imposes limitations on usage scenarios compared to the Message Broker pattern

Sample implementations: Apache Flume, Logstash, Fluentd, Scribe

Distributed Message Broker

Family/Integration/Messaging

Description: This pattern is a descendant of a more traditional Message Broker, but offers high scalability by distributing messages across multiple nodes. Most implementations offer Pub/Sub and Peer-to-Peer modes.



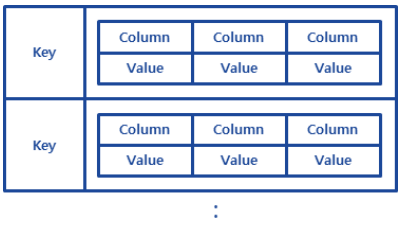
Consequences:
★★★ **Performance –** can handle high throughput with low latency
★ **Compatibility –** in most cases requires writing of custom code to be plugged with event producers and consumers
★★★ **Flexibility –** can be used for multiple purposes – routing, transformation, aggregation, pub-sub, etc.

Sample implementations: RabbitMQ, Apache Kafka, Apache ActiveMQ, Amazon SQS

Column-Family

Family/Data Storage/NoSQL Database

Description: Extends Key-Value databases by storing not strictly defined collections of one or more key-value pairs that match a record. Can be presented as two dimensional arrays whereby each key has one or more key-value pairs attached to it.



Consequences:

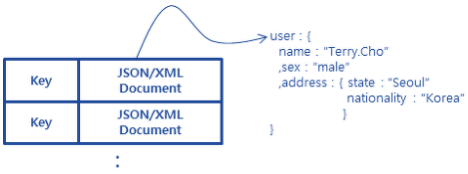
★★★ **Performance** – is extremely fast due to absence of schema definition, relational, transactional or referential integrity functionality
★★★ **Scalability** – can be linearly scaled by splitting data across servers using hash value calculated based on a row key
★★★ **Availability** – high availability is provided by clustering and distributed file system (e.g. HDFS)
★ **Ad-hoc Analysis** – supports secondary indexing, but no aggregate functions

Sample implementations: Cassandra, HBase

Document-Oriented

Family/Data Storage/NoSQL Database

Description: Works similarly to Column-Family database, but allows much deeper nesting and complex structures to be stored (e.g. a document, within a document, within a document). Documents overcome constraints of one or two level of key-value nesting of Column-Family databases. Complex and arbitrary structure can form a document, which can be stored as a record.



Consequences:

★★ **Performance** – performance varies significantly from one implementation to the next, but overall not as fast as Key-Value databases
★★★ **Scalability** – over 100 organizations run clusters with 100+ nodes. Some clusters exceed 1,000 nodes
★★★ **Availability** – high availability is provided by clustering and replication
★½ **Ad-hoc analysis** – somewhat better than other NoSQL families, but still not as good as relational databases or interactive query engines

Sample implementations: MongoDB, CouchDB

Distributed File System

Family/Data Storage

Description: The modern distributed file systems are highly fault-tolerant and designed to run on low-cost hardware. Open source implementations such as HDFS (Hadoop Distributed File System) and CFS (Cassandra File System) provide high throughput access to application data and are suitable for applications that process large data sets.

Consequences:

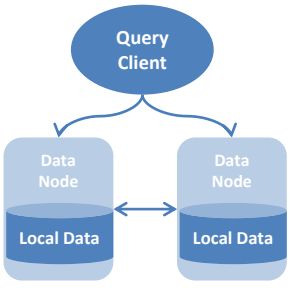
★★ **Performance** – designed for fast sequential read/write access, really good for batch processing (Map Reduce). For random read/write recommended using NoSQL databases (e.g. HBase on top of HDFS).
★★★ **Scalability** – massively and linearly scalable, number of nodes theoretically is unlimited, existing production clusters with up to 10,000 nodes
★★★ **Availability** – default replication of data to 3 nodes, rack and datacenter-awareness, no single-points of failure

Sample implementations: Hadoop Distributed File System (HDFS), Cassandra File System (CFS)

Interactive Query Engine

Family/Analytics/Search & Query

Description: Distributed Query Processor is aimed for running batch as well as interactive analytic queries against data sources of large size.



Consequences:

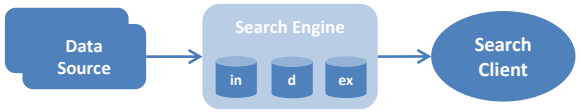
★★ **Performance –** can query large amounts of data in human-time (2-30 seconds), however still not as fast as relational data warehouse
★★½ **Reliability** – some implementations provide long-running query support and mid-query fault recovery
★★★ **Ad-hoc analysis** – best in class, similar to relational MPP data warehouse engines

Sample implementations: Impala, Apache Hive, Spark SQL

Distributed Search Engine

Family/Analytics/Search & Query

Description: Scalable indexing solution with full-text, interactive search. Most implementations provide API that allows executing complex queries on semi-structured data such as logs, web pages and document files.



Consequences:

★★ **Ad-hoc analysis** – query languages often include faceted and geospatial search, stat functions and simple joins to query, analyze and visualize data

★★★ **Scalability** – can be linearly scaled by providing distributed indexing

★★★ **Availability** – high availability is provided by clustering and replication

Sample implementations: Elasticsearch, Apache Solr, Splunk Indexer