CS5351 Software Engineering 2023/24 Semester A Software Architecture Exercise

Dr W.K. Chan
Department of Computer Science
Email: `wkchan@cityu.edu.hk`
Website: `http://www.cs.cityu.edu.hk/~wkchan`

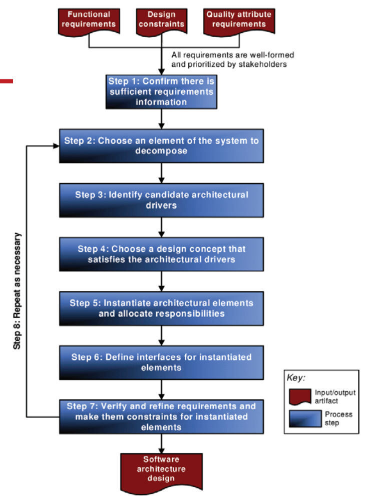
Exercise Goal

We will practice addressing quality attribute (QA) issues at the software architectural level.

In brief, ADD runs iteratively:

1. Choose an element to decompose
2. Identify QAs to be met by the element
3. Choose and evaluate a design concept to address the QAs of the element
4. Apply the design concept to the element possibly by decomposing the elements into sub-elements

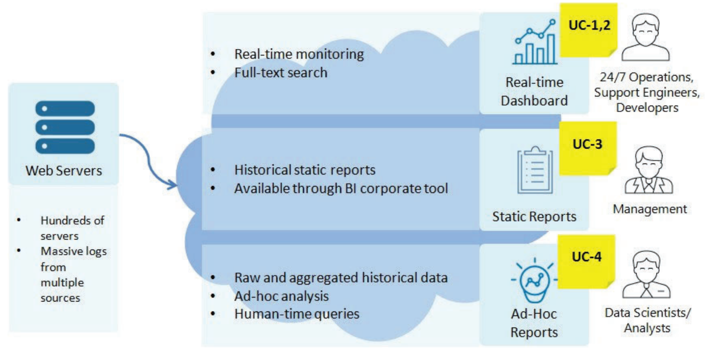
Attribute-Driven Design (ADD)


Functional requirements
Design constraints
Quality attribute requirements
All requirements are well-formed and prioritized by stakeholders
Step 1: Confirm there is sufficient requirements information
Step 2: Choose an element of the system to decompose
Step 3: Identify candidate architectural drivers
Step 4: Choose a design concept that satisfies the architectural drivers
Step 5: Instantiate architectural elements and allocate responsibilities
Step 6: Define interfaces for instantiated elements
Step 7: Verify and refine requirements and make them constraints for instantiated elements
Step 8: Repeat as necessary
Software architecture design
Key:
Input/output artifact
Process step

Context of the Case Study

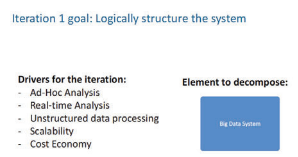
- You are a Software Architect who is going to design the software architecture for an important **Big Data system**.
- The company is an Internet company that provides popular contents to *millions of web users*.
- The system should collect and analyze a massive amount of semi-structured data coming from *hundreds of servers*.
- The stakeholders of the system include Operations Team, Data Scientists, and Management.

Background: Use Cases


Web Servers
Hundreds of servers
Massive logs from multiple sources
Real-time monitoring
Full-text search
Real-time Dashboard
UC-1,2
24/7 Operations, Support Engineers, Developers
Historical static reports
Available through BI corporate tool
Static Reports
UC-3
Management
Raw and aggregated historical data
Ad-hoc analysis
Human-time queries
Ad-Hoc Reports
UC-4
Data Scientists/ Analysts

ADD Iteration 1

- Aim: Decompose the system into an overall system architecture
- **Find architecture candidates**
- **Identify Quality Attributes (QAs)** as Driver
  - Many servers and many users => scalability
  - Nature of big data system => unstructured data processing
  - Nature of use cases: ad-hoc analysis, real-time analysis
  - Constraint: use open-source tools for cost-saving
- **Evaluate** the choices against the QAs


Iteration 1 goal: Logically structure the system

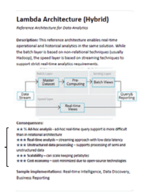
Drivers for the iteration:
- Ad-Hoc Analysis
- Real-time Analysis
- Unstructured data processing
- Scalability
- Cost Economy

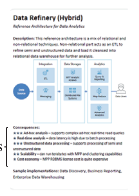
Element to decompose:

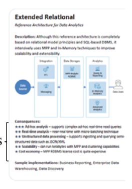
Big Data System

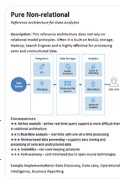
Chosen Candidate for Iteration 1

- *Lambda Architecture* is the best option (highest star points + balance)
  - If unbalanced, some stakeholders may not be happy

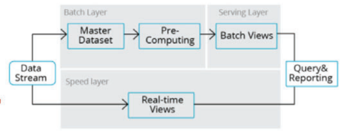

14.5 stars


10 stars


10 stars


13.5 stars

More Iterations on ADD



We now want to design each element in the adopted architecture.

Let's work on the following four elements in iterations 2 to 5. In general, we will have more iterations in practice.

- Iteration 1: Decompose the system into an overall system architecture
- Iteration 2: Design the *data stream*
- Iteration 3: Design the *Batch Layer*
- Iteration 4: Design the *Serving Layer*
- Iteration 5: Design the *Speed Layer*

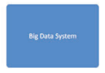
We need to look at the QAs for each iteration (see next slide).

Goals of the Five Iterations

Iteration 1 goal: Logically structure the system

Drivers for the iteration:
- Ad-Hoc Analysis
- Real-time Analysis
- Unstructured data processing
- Scalability
- Cost Economy

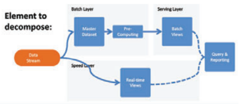
Element to decompose:



Iteration 2: Design Data Stream Element

Drivers for the iteration:
- Performance (for Family and Technology)
- Compatibility (for Family)
- Reliability (for Technology)

Element to decompose:

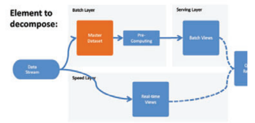


To Do: Select 1 Family card and 1 Technology card

Iteration 3: Design Batch Layer

Drivers for the iteration:
- Scalability
- Availability

Element to decompose:

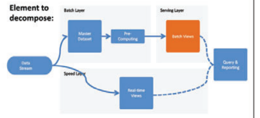


To Do: Select 1 Family card

Iteration 4: Design Serving Layer

Drivers for the iteration:
- Ad-hoc Analysis (for Family)
- Performance (for Family and Technology)

Element to decompose:

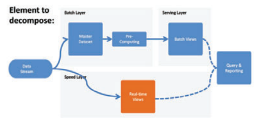


To Do: Select 1 Family and 1 Technology card

Iteration 5: Design Speed Layer

Drivers for the iteration:
- Ad-hoc Analysis (for the family)
- Real-time Analysis (for the technology)

Element to decompose:



To Do: Select 1 Family and 1 Technology card

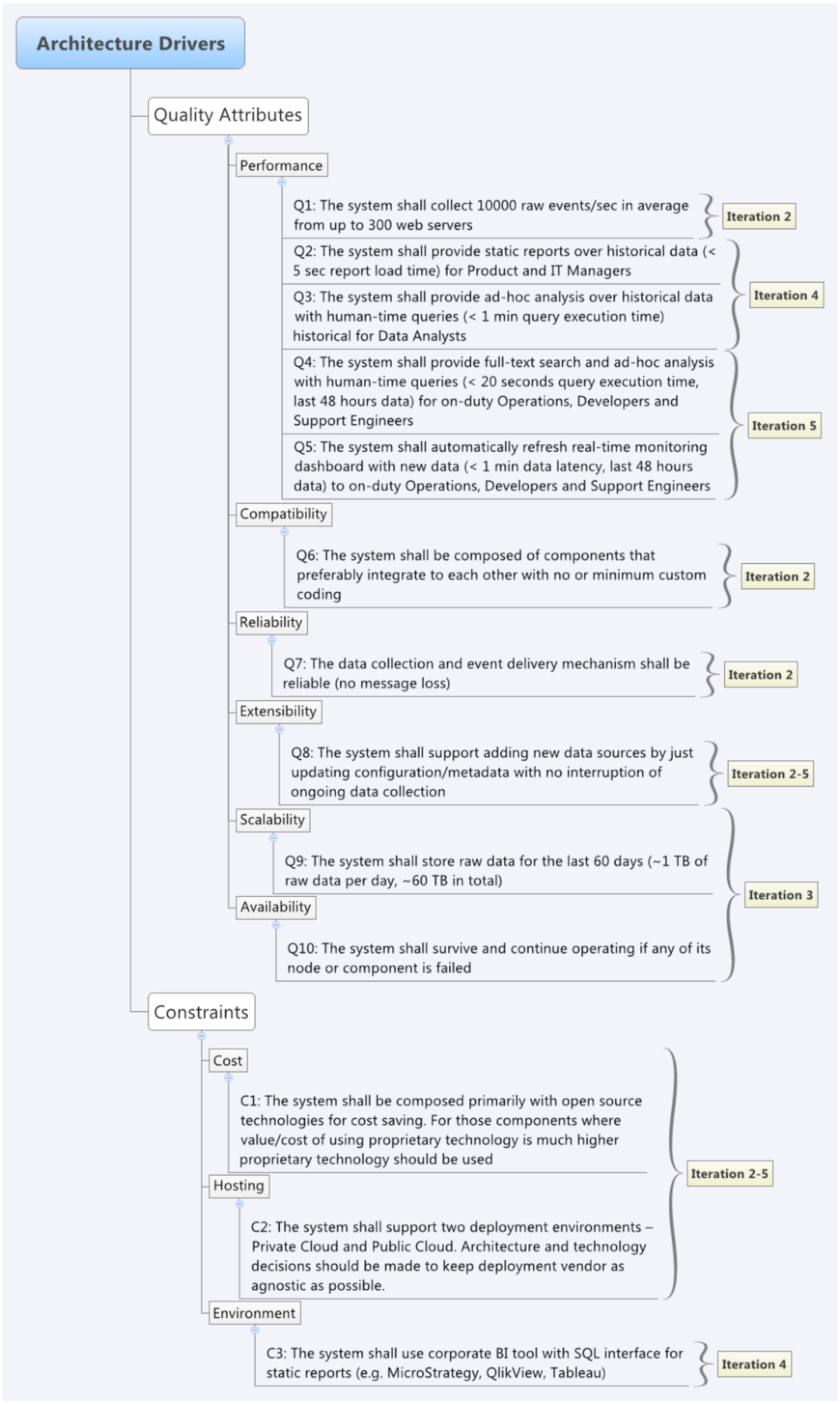
Background: Use Cases

UC1: Monitor online services	On-duty Operations Team can monitor the actual state of services and IT infrastructure (such as web server load, user activity, and errors) through a real-time operational dashboard to react on potential issues as soon as possible
UC2: Troubleshoot online service issues	The Operations Team (including Support Engineers and Developers) can do troubleshooting and root cause analysis on the latest collected logs by searching the log patterns and filtering messages
UC3: Provide management reports	Managers, including Product and IT Managers, can see historical information through predefined (static) reports in a corporate Business Intelligent tool, such as system load in time, features usage, SLA violations and quality of releases
UC4: Provide ad hoc data analytics	Data Scientists and Analysts can do ad hoc data analysis through SQL-like queries to find out specific data patterns and correlations to improve infrastructure capacity planning and customer satisfaction

Score Card for the Case Study

	Iteration					
	1	2	3	4	5	Total Score
(a) Design Decision (Names of selected design concept(s))	Lambda Architecture					
(b) Driver Selection Points (from Cards)	14.5					
(c) Analysis bonus points (from review to be released after all 5 iterations)	2					
(d) Iteration total = (a) + (b) + (c)	16.5					

Fill other cells when you perform Iterations 2-5

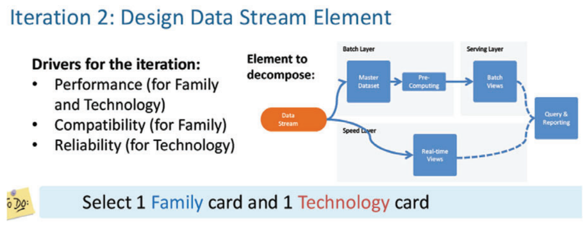


Architecture Drivers

- Quality Attributes
  - Performance
    - Q1: The system shall collect 10000 raw events/sec in average from up to 300 web servers — **Iteration 2**
    - Q2: The system shall provide static reports over historical data (< 5 sec report load time) for Product and IT Managers
    - Q3: The system shall provide ad-hoc analysis over historical data with human-time queries (< 1 min query execution time) historical for Data Analysts — **Iteration 4** (Q2–Q3)
    - Q4: The system shall provide full-text search and ad-hoc analysis with human-time queries (< 20 seconds query execution time, last 48 hours data) for on-duty Operations, Developers and Support Engineers
    - Q5: The system shall automatically refresh real-time monitoring dashboard with new data (< 1 min data latency, last 48 hours data) to on-duty Operations, Developers and Support Engineers — **Iteration 5** (Q4–Q5)
  - Compatibility
    - Q6: The system shall be composed of components that preferably integrate to each other with no or minimum custom coding — **Iteration 2**
  - Reliability
    - Q7: The data collection and event delivery mechanism shall be reliable (no message loss) — **Iteration 2**
  - Extensibility
    - Q8: The system shall support adding new data sources by just updating configuration/metadata with no interruption of ongoing data collection — **Iteration 2-5**
  - Scalability
    - Q9: The system shall store raw data for the last 60 days (~1 TB of raw data per day, ~60 TB in total)
  - Availability
    - Q10: The system shall survive and continue operating if any of its node or component is failed — **Iteration 3** (Q9–Q10)
- Constraints
  - Cost
    - C1: The system shall be composed primarily with open source technologies for cost saving. For those components where value/cost of using proprietary technology is much higher proprietary technology should be used
  - Hosting
    - C2: The system shall support two deployment environments – Private Cloud and Public Cloud. Architecture and technology decisions should be made to keep deployment vendor as agnostic as possible. — **Iteration 2-5** (C1–C2)
  - Environment
    - C3: The system shall use corporate BI tool with SQL interface for static reports (e.g. MicroStrategy, QlikView, Tableau) — **Iteration 4**

ADD Iteration 2

- We choose a **family of technology** to address Performance and Compatibility concerns and then **a specific technology under the selected family** to address Performance and Reliability issues for **Data Stream**
  - Note that only the QAs we care about should be counted in evaluating the suitability.



Iteration 2: Design Data Stream Element

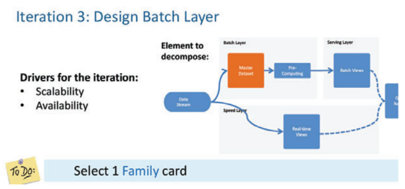
Drivers for the iteration:
- Performance (for Family and Technology)
- Compatibility (for Family)
- Reliability (for Technology)

Element to decompose:

Select 1 Family card and 1 Technology card

ADD Iteration 3

- Goal: Address the Scalability and Availability concerns in **Batch Layer**
- Like Iteration 2, we research the Family candidates, evaluate their suitability, and select the best one.



Iteration 3: Design Batch Layer

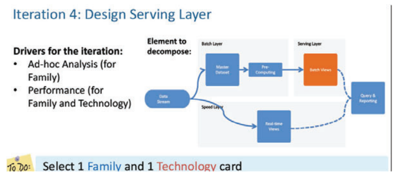
Drivers for the iteration:
- Scalability
- Availability

Element to decompose:

Select 1 Family card

ADD Iteration 4

- Goal: We address the need for Ad-hoc analysis and Performance concerns in this iteration in **Serving Layer**



Iteration 4: Design Serving Layer

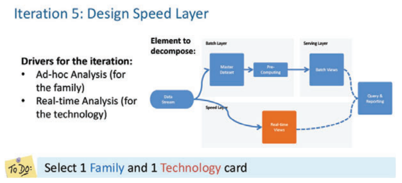
Drivers for the iteration:
- Ad-hoc Analysis (for Family)
- Performance (for Family and Technology)

Element to decompose:

Select 1 Family and 1 Technology card

ADD Iteration 5

- Goal: Address the ad hoc analysis and real-time analysis in the **Speed Layer** at the Family and Technology Levels, respectively.



Iteration 5: Design Speed Layer

Drivers for the iteration:
- Ad-hoc Analysis (for the family)
- Real-time Analysis (for the technology)

Element to decompose:

Select 1 Family and 1 Technology card

Apache Flume

Technology/Integration/Messaging/Data Collector

Description: A distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data. It has a simple and flexible architecture based on streaming data flows. It uses a simple extensible data model that allows for online analytic application. Flume is an Apache top-level project, Apache 2.0 license, written in Java.

Consequences:
★★ **Performance –** can handle high throughput with low latency, depends on channel configuration (memory, file etc.), number of sinks, threads etc. Supports horizontal scaling for performance (throughput) improvement.
★★ **Reliability** – uses a transactional approach to guarantee the reliable delivery of events (via a file channel). Usage of a memory channel improves performance, but can lead to message loss.
★★★ **Cost economy** - released as open source under the terms of the Apache License

Logstash

Technology/Integration/Messaging/Data Collector

Description: Logstash is a tool for managing events and logs. You can use it to collect logs, parse them, and store them in a centralized storage for later use (e.g. for searching). Logstash is a JRuby based application which requires the JVM to run. It is a part of the Elasticsearch family. The license is Apache 2.0.

Consequences:
★★ **Performance –** can handle high throughput with low latency (via leveraging multiple threads for filter workers, input/output workers etc., adding more instances)
★★ **Reliability** – depends on broker implementation, reported issues with losing messages
★★★ **Cost economy** - released as open source under the terms of the Apache License

Fluentd

Technology/Integration/Messaging/Data Collector

Description: Fluentd is an open source data collector (Apache 2.0 License), which unifies the data collection and consumption for a better use and understanding of data. Uses JSON as an internal format to unify all the collect, filter, buffer, and output mechanisms across multiple sources and destinations. Fluentd is written in a combination of C and Ruby.

Consequences:
★★ **Performance –** requires very little system resources. The vanilla instance runs on 30-40MB of memory and can process 13,000 events/second/core.
★★★ **Reliability** – supports memory- and file-based buffering to prevent inter-node data loss. The buffering logic is highly tunable and can be customized for various throughput/latency requirements.
★★★ **Cost economy** - released as open source under the terms of the Apache License

RabbitMQ

Technology/Integration/Messaging/Distributed Message Broker

Description: RabbitMQ is open source message broker software (sometimes called message-oriented middleware) that implements the Advanced Message Queuing Protocol (AMQP).

Consequences:
★★ **Performance –** good response rate and quite high memory consumption, scales with clustering well
★★ **Reliability –** implemented distributed from scratch (Erlang), built-in clustering, can persist and replicate configuration and messages, allows acknowledgements. Can exhaust RAM.
★★★ **Cost economy** - released as open source under the terms of the Mozilla Public License

Apache Kafka

Technology/Integration/Messaging/Distributed Message Broker

Description: Apache Kafka is an open-source message broker project developed by the Apache Software Foundation written in Scala. The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds.

Consequences:
★★★ **Performance –** very fast response rate, scales with clustering
★★★ **Reliability –** durable and fault tolerant by design and implementation language (Scala), persists and replicates messages
★★★ **Cost economy** - released as open source under the terms of the Apache License

Amazon SQS

Technology/Integration/Messaging/Distributed Message Broker

Description: Amazon Simple Queue Service (SQS) is a fast, reliable, scalable, fully managed message queuing service. SQS makes it simple and cost-effective to decouple the components of a cloud application.

Consequences:
★★½ **Performance –** serves up to 35K msg/sec, auto-scales by Amazon
★★★ **Reliability –** designed for high availability, works as a service, very reliable, durable messages
★★ **Cost economy** - $0.50 per 1 million requests per month

Apache ActiveMQ

Technology/Integration/Messaging/Distributed Message Broker

Description: Apache ActiveMQ ™ is an open source messaging and Integration Patterns server. Apache ActiveMQ is fast, supports many Cross Language Clients and Protocols, it comes with easy to use Enterprise Integration Patterns and many advanced features while fully supporting JMS 1.1 and J2EE 1.4.

Consequences:
★★ **Performance –** good response rate on par with RabbitMQ but slower than Kafka
★★ **Reliability –** high availability is supported, persistent messaging supported, some issues reported with clustering and occasional message loss
★★★ **Cost economy** - released as open source under the terms of the Apache License

Apache Cassandra

Technology/Data Storage/NoSQL Database/Column-Family

Description: The Apache Cassandra is an open source distributed database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. Cassandra offers robust support for clusters spanning multiple datacenters, with asynchronous masterless replication allowing low latency operations for all clients.

Consequences:

★★★ **Performance** – Cassandra is 30%-100% faster (avg) than HBase for both reads and writes due to efficient memory/caching, SSD support, online snapshots, locally-managed storage, effective compaction, etc. It is a winner of most performance benchmarks in its class.
★★★ **Reliability** – one of the most reliable and mature NoSQL databases
★★★ **Real-time analysis** – fast access to data makes it a good solution for real-time analysis
★★★ **Cost economy** - released as open source under the terms of the Apache License

Apache HBase

Technology/Data Storage/NoSQL Database/Column-Family

Description: HBase is a NoSQL databases which uses HDFS and Hadoop as a storage engine, and integrates well with most Hadoop products. Typical use case is random, real-time read/write access to your Big Data, scaling up data to billions of rows, and millions of columns. Apache HBase is an open-source, distributed, versioned, non-relational database modeled after Google's BigTable.

Consequences:

★★½ **Performance** – is still fast compared to other databases, but slower than Cassandra due to underlying HDFS, ineffective compaction, etc.
★★ **Reliability** – very slow crash recovery, compactions that disrupt operations, unreliable splitting, and very small write-ahead logs makes it less reliable than Cassandra despite reliable HDFS storage
★★★ **Real-time analysis** – one of the fastest random access to data in NoSQL world makes it a good solution for real-time analysis
★★★ **Cost economy** - released as open source under the terms of the Apache License

MongoDB

Technology/Data Storage/NoSQL Database/Document-Oriented

Description: MongoDB (from "hu**mongo**us") is an open-source document database, and the leading NoSQL database. Written in C++, MongoDB features: JSON-style documents with dynamic schemas offer simplicity and power, indexes on any attribute, mirrors across LANs and WANs for scale, scales horizontally without compromising functionality, flexible aggregation and data processing, etc.

Consequences:

★★ **Performance** – not as fast as simplest key-value storages, but features like auto-sharding, full index support, map-reduce makes it fast enough. Written in C++.
★★ **Reliability** – durability is a known problem (being fixed though), issues with repairing databases, requires replication setup to implement reliability
★★★ **Real-time analysis** – one of the most common use-cases, supports schema design, indexing and sharding for real time analytics workloads
★★★ **Cost economy** - released as open source under the terms of the GNU AGPL license, commercial licenses are also available

Apache CouchDB

Technology/Data Storage/NoSQL Database/Document-Oriented

Description: CouchDB is a database that embraces the web by storing data with JSON documents; allowing accessing data via HTTP; indexing, combining, and transforming your documents with JavaScript. CouchDB works well with modern web and mobile apps, supports incremental replication and master-master setups with automatic conflict detection.

Consequences:

★½ **Performance** – fast for direct ID lookups and map-reduce jobs, but that's it. Users reported performance issues.
★ **Reliability** – serious problems with reliability and availability were reported by users despite functionality like replication and automatic conflict resolution. Not yet suitable for highly-available or heavy-loaded solutions.
★★★ **Real-time analysis** – fast ID lookups and fast aggregation calculation using map-reduce
★★★ **Cost economy** - released as open source under the terms of the Apache License

Impala

Technology/Analytics/Search & Query/Interactive Query Engine

Description: Cloudera's open source massively parallel processing (MPP) SQL query engine for data stored in a computer cluster running Apache Hadoop.

Consequences:

★★★ **Performance –** considered as one of fastest technologies at the moment, significantly faster than Hive

★★ **Processing capabilities** – supports the SQL-92 standard, but overall features are limited compared to HiveQL

★★ **Reliability** – designed for short queries; queries must be restarted if a node fails

★★★ **Cost economy** - released as open source under the terms of the Apache License

Spark SQL

Technology/Analytics/Search & Query/Interactive Query Engine

Description: Based on Spark – an in-memory distributed computing engine (alternative to Hadoop MapReduce), Spark SQL allows running SQL and HiveQL queries over large datasets. Spark SQL is an ancestor of Shark.

Consequences:

★★★ **Performance –** considered as one of fastest technologies at the moment, significantly faster than Hive

★★ **Processing capabilities –** based on SQL-like query language supporting most of HiveQL features including UDFs and SerDes

★★★ **Reliability –** supports long-running queries and mid-query fault recovery

★★★ **Cost economy** - released as open source under the terms of the Apache License

Apache Hive

Technology/Analytics/Search & Query/Interactive Query Engine

Description: The Apache Hive data warehouse software facilitates querying and managing large datasets residing in distributed storage.

Consequences:

★½ **Performance –** even with Stinger initiative Hive is still slow compared to other alternatives such as Impala or Spark SQL

★★★ **Processing capabilities** – are based on HiveQL, a subset of SQL-92 which offers extensions such as non-scalar data types, XML/JSON functions, UDFs, custom SerDes and other features

★★★ **Reliability** – supports long-running queries and mid-query fault recovery

★★★ **Cost economy** - released as open source under the terms of the Apache License

Splunk (Indexer)

Technology/Analytics/Search & Query/Distributed Search Engine

Description: Splunk Indexer is a component responsible for indexing and correlation of real-time data in a searchable repository. Along with other Splunk components it creates a larger solution which provides graphs, reports, alerts, dashboards and visualizations.

Consequences:

★★½ **Ad-hoc analysis** – provides REST-like API and CLI with proprietary SPL language, supports faceted search, abnormalities finding, SQL-like joins, stats functions and functions that support chart visualization

★★½ **Real-time analysis** – enables near real time indexing with about a second latency

★★★ **Reliability** – implements sharding, load balancing, automatic failover and recovery

★ **Cost economy** - the freeware version is limited to 500 MB of data a day, $1,800 for a 1 GB/day annual term license or $4,500 for a 1 GB/day perpetual

Elasticsearch

Technology/Analytics/Search & Query/Distributed Search Engine

Description: An open source search and analytics engine based on Lucene. It provides distributed, multitenant-capable full-text search capabilities with a RESTful web interface and schema-free JSON documents. Along with Logstash and Kibana creates ELK stack to collect, index and visualize data.

Consequences:

★★ **Ad-hoc analysis** – provides JSON-based query language Query DSL, supports aggregations and filtering, allows joins through *has_child* queries

★★½ **Real-time analysis** – enables near real time indexing with about a second latency

★★★ **Reliability** – implements sharding, load balancing, automatic failover and recovery

★★★ **Cost economy** - released as open source under the terms of the Apache License

Apache Solr

Technology/Analytics/Search & Query/Distributed Search Engine

Description: An open source enterprise search platform from the Apache Lucene project. Providing distributed search and index replication, Solr is highly scalable and adopted by a number of Big Data vendors such as Cloudera and Amazon Web Services.

Consequences:

★★ **Ad-hoc analysis** – provides REST-like API and extends Lucene query language, supports faceted search and filtering including pivot facets

★★½ **Real-time analysis** – enables near real time indexing with about a second latency

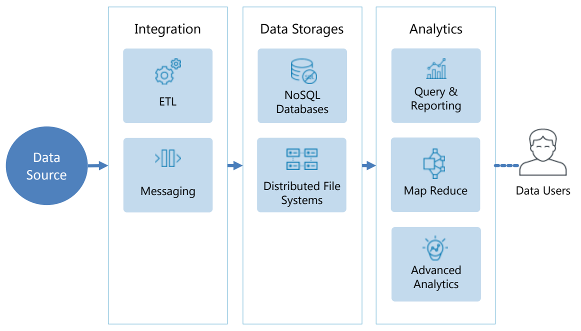
★★★ **Reliability** – implements sharding, load balancing, automatic failover and recovery

★★★ **Cost economy** - released as open source under the terms of the Apache License

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

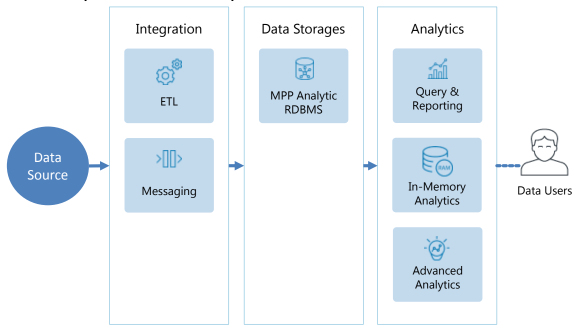
★★★ **Cost economy** – cost minimized due to open-source technologies

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

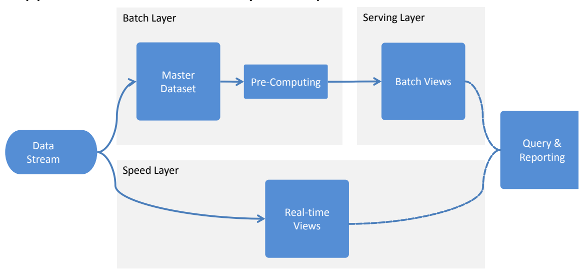
★ **Cost economy** – MPP RDBMS license cost is quite expensive

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

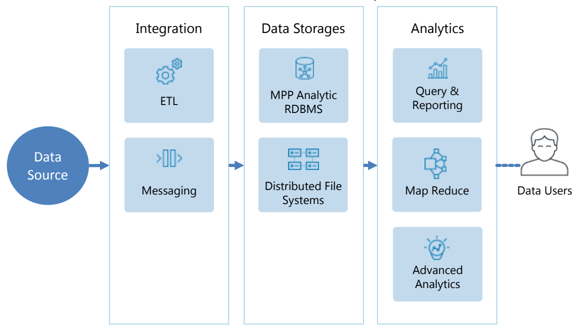
★★★ **Cost economy** – cost minimized due to open-source technologies

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

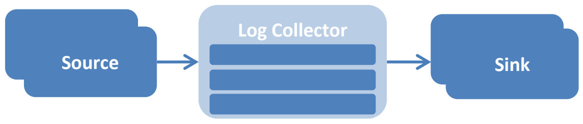
★ **Cost economy** – MPP RDBMS license cost is quite expensive

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

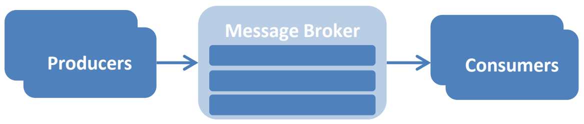
★★ **Flexibility –** imposes limitations on usage scenarios compared to the Message Broker pattern

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

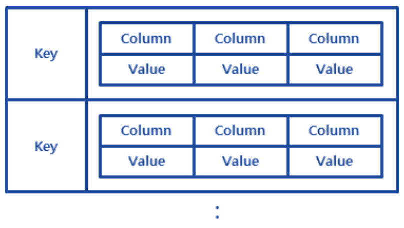
★★★ **Flexibility –** can be used for multiple purposes – routing, transformation, aggregation, pub-sub, etc.

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.



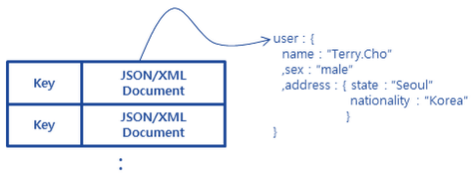
Consequences:
★★★ **Performance** – is extremely fast due to absence of schema definition, relational, transactional or referential integrity functionality
★★★ **Scalability** – can be linearly scaled by splitting data across servers using hash value calculated based on a row key
★★★ **Availability** – 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.

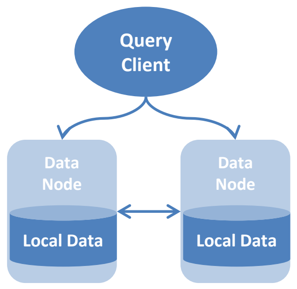
Consequences:
★★ **Performance** – designed for fast sequential read/write access, really good for batch processing (Map Reduce). For random read/write recommended using NoSQL databases (e.g. HBase on top of HDFS).
★★★ **Scalability** – massively and linearly scalable, number of nodes theoretically is 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.



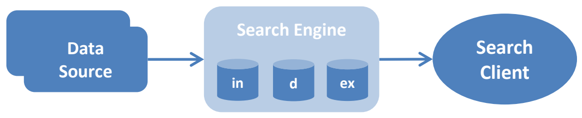
Consequences:
★★ **Performance** – can query large amounts of data in human-time (2-30 seconds), however still not as fast as relational data warehouse
★★½ **Reliability** – some implementations provide long-running query support and mid-query fault recovery
★★★ **Ad-hoc analysis** – best in class, similar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