Discrepancy Analysis - Apache Hadoop

EECS 4314 - Advanced Software Engineering

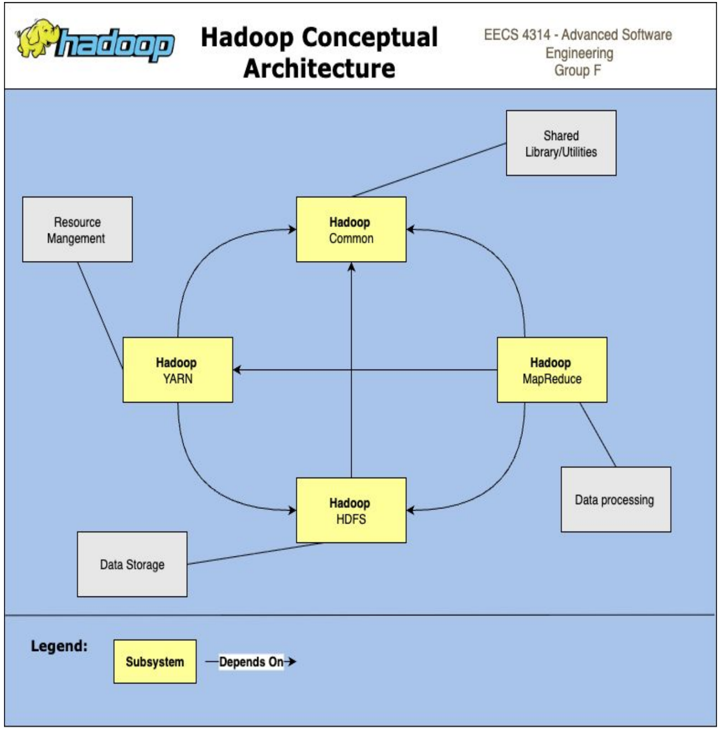
Group F

Overview

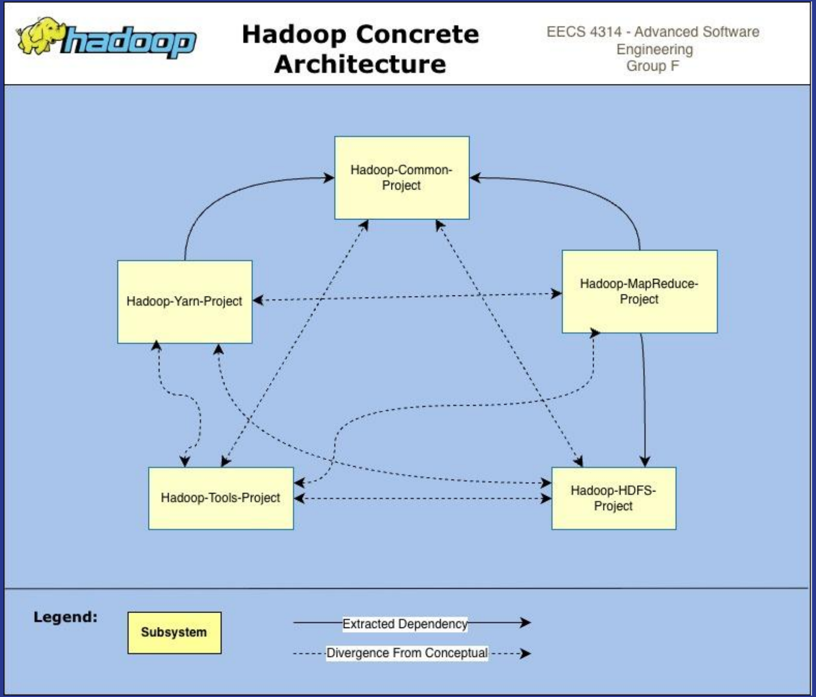
- Compare Conceptual (A1) vs Concrete (A2) architecture of Apache Hadoop
- Analyze discrepancies at two levels:
  - Top-level subsystem interactions (HDFS, YARN, MapReduce, Common)
  - Internal MapReduce subsystem
- Highlight absences, divergences, and rationale + revisions
- Sequence diagram for word count tool
- Discuss concurrency + team implications on architecture
- Present limitations and lessons learned

Top Level Analysis

Conceptual vs Concrete



Conceptual A1



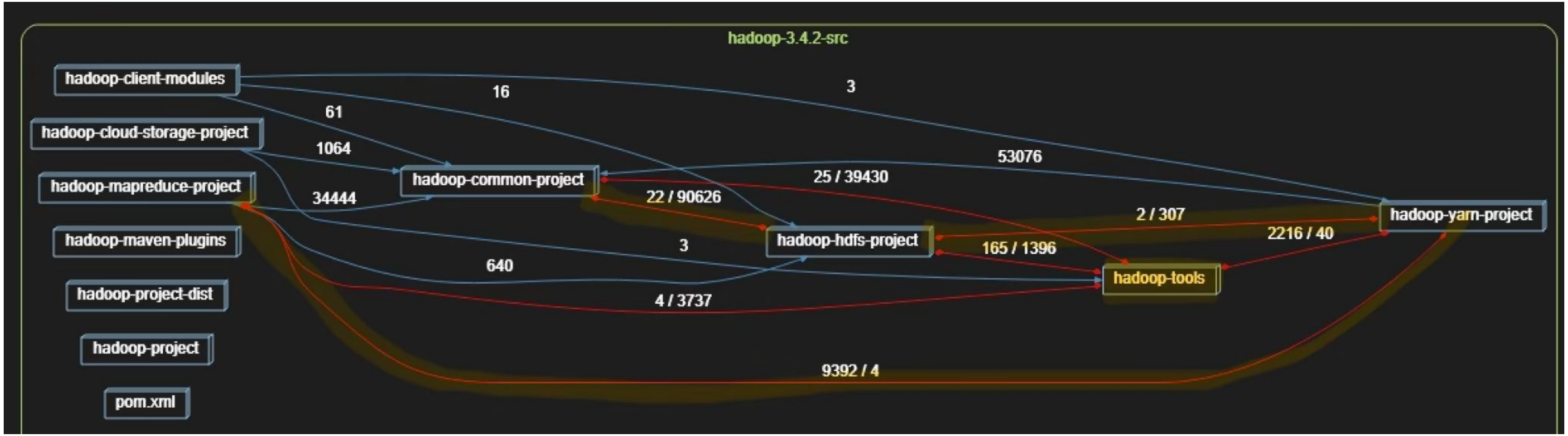
——Concrete A2 (revised)

Discrepancies

Structural Differences
A1 has a missing subsystem: Hadoop-Tools-Project

New Dependencies
A2 includes dependencies not shown in A1:
HDFS <-> YARN
HDFS <-> Common
MapReduce <-> YARN

Complexity Difference
A2 contains more mutual dependencies then the conceptual.



Rationale

Conceptual vs concrete Differences
A1 derived from documentation and abstracts and hides low-level implementation.
A2 reflects the real code, function calls and relations.

Unexpected Dependencies

Hadoop-Tools-Project:

- Contains collection of tools that share configuration handling used across subsystems.
- Necessary for configuration handling

MapReduce <-> YARN

- YARN manages MapReduce jobs using the Application Master.
- Executes tasks through NodeManager and works with ResourceManager.

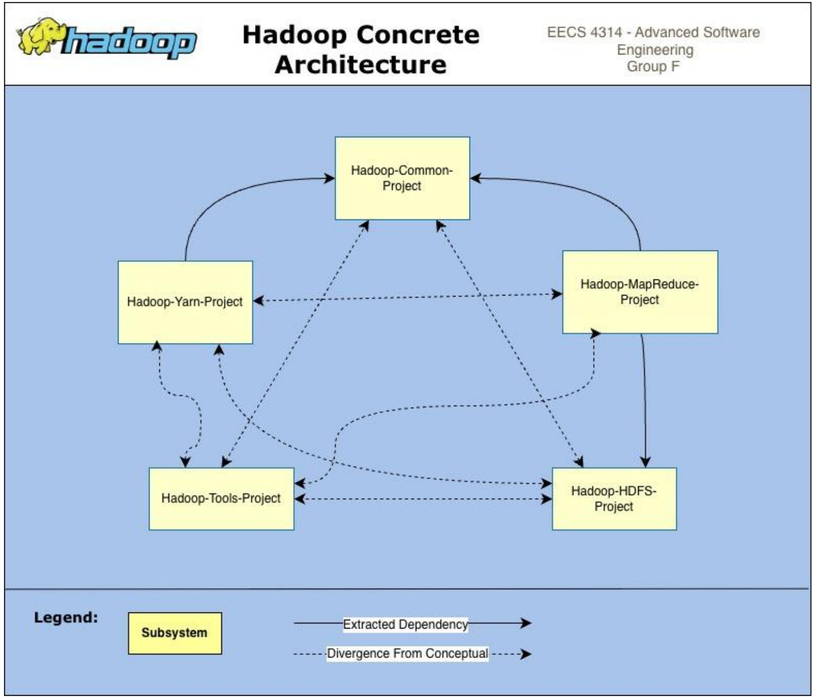
HDFS <-> YARN and HDFS <-> Common

- YARN manages the data access applications to interact with HDFS.

All <-> Tools-Project

- New subsystem

Fixes



Updates Made to the Conceptual Architecture

- A1 was modified to include Hadoop-Tools
- Added missing dependencies:
  HDFS <-> YARN
  HDFS <-> Common
  MapReduce <-> YARN

Exclusions

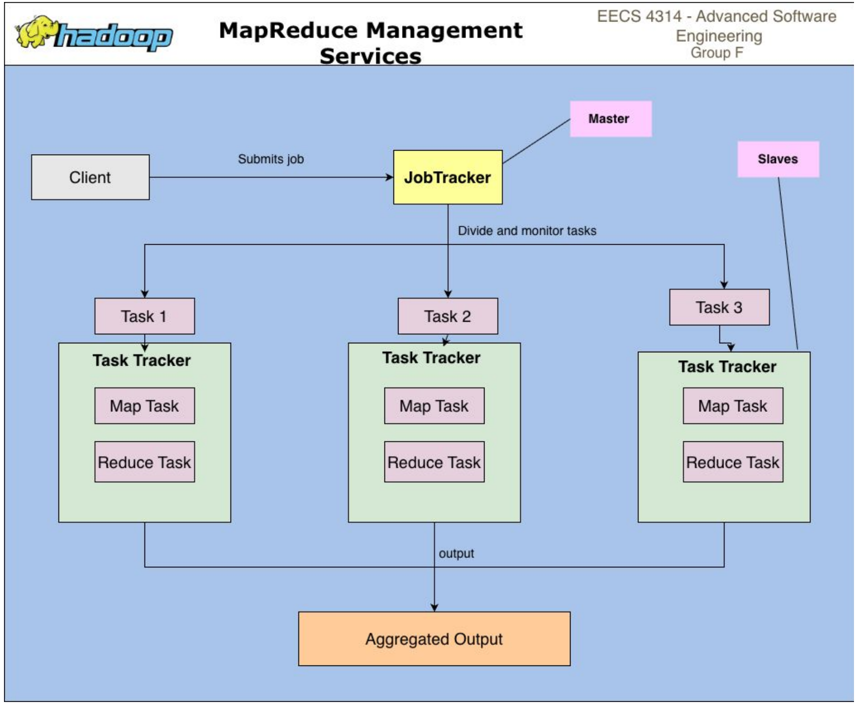
- Excluded non-architecture modules from the conceptual view: build-tools, client-modules, project, pom.xml, assemblies, cloud-storage, maven-plugins, project-dist, minicluster, and dist.
- Including them would hinder clarity and make it harder to focus on the main five subsystem.

Revision

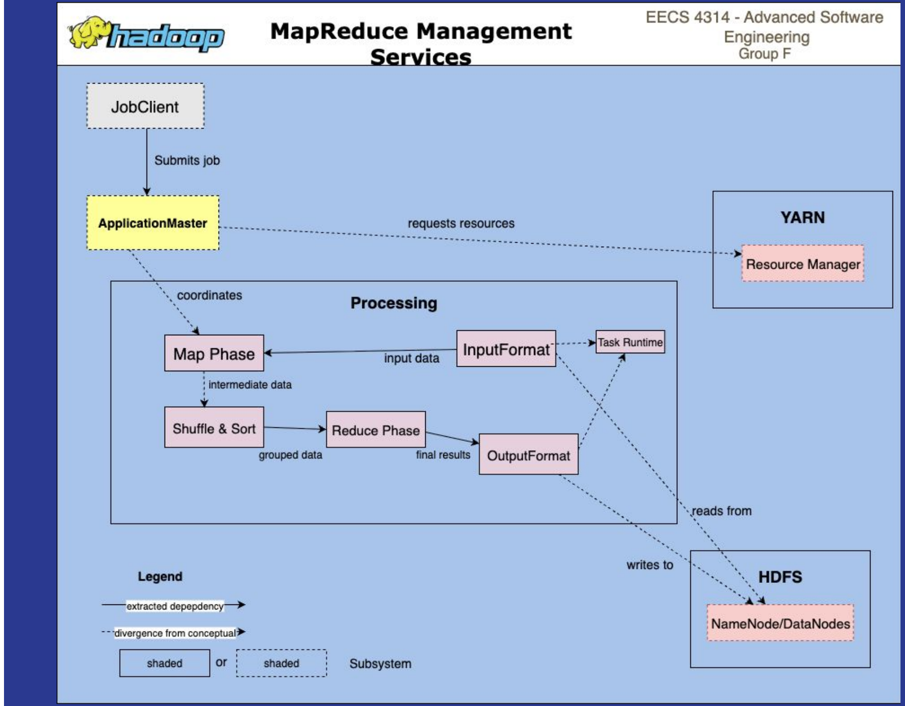
- Revised conceptual architecture to reflect A2.
- Added dashed lines to indicate divergences between A1 and A2.
- Maintains abstraction and reliability.

Mapreduce Analysis

Conceptual vs Concrete (MapReduce)



Conceptual A1



——Concrete A2 (revised)

Discrepancies

A1 Model (Conceptual):

- Uses JobTracker/TaskTracker master-slave control & No ApplicationMaster or JobClient shown
- Represents MapReduce as only a data processing pipeline *(→ Input → Map → Shuffle → Reduce → Output)*
- Does not include any shared execution layer

A2 Model (Concrete) - Has Revision + Fix:

- Shows JobClient + ApplicationMaster running on YARN
- HDFS is tightly coupled with job execution (input splits, intermediate storage support, final output)
- Reveals three architectural layers, not just the pipeline: Processing Layer, Client/Control Layer & Infrastructure Layer
- Reveals a shared Task Runtime execution layer

Rationale -> Evolution

- A1 still depicts the JobTracker/TaskTracker model because it's easier for explaining MapReduce but A2 reflects the actual current implementation in system.
- Architectural shift: Control moved out of MapReduce into YARN to improve scalability and cluster resource sharing
- A1 matches older Hadoop MapReduce v1 illustration while A2 reflects Hadoop v2+ (MRv2) as extracted from version 3.4.2
- MapTask and ReduceTask rely on the Task Runtime execution layer for:
    - Failure detection & retries
    - Progress monitoring & status reporting

Discrepancies Contd.

A1 Model (Conceptual):

- Shows Shuffle as a single intermediate step in the pipeline (placed between Map and Reduce)
- Treated as just an internal data rearrangement

A2 Model (Concrete) - Has Revision + Fix:

- Identifies Shuffle & Sort as a dedicated subsystem (hadoop-mapreduce-client-shuffle)
- Elevated to its own client module, independent of Map/Reduce logic

Rationale -> Algorithm simplification

- Shuffle & Sort is responsible for:
  - Large-scale data transfer across nodes
  - Merge, buffer, partition, and grouping of key/value pairs
  - Supporting HDFS + network boundaries during intermediate output
- It needs special handling and tuning because this is what determines whether MapReduce can handle large workloads successfully
- Making it a separate subsystem allows performance tuning and fault handling independent of Map/Reduce code

Discrepancies Contd.

A1 Model (Conceptual):

- Shows Input/Split and Output as fixed steps in a pipeline
- Implies that MapReduce reads data uniformly, regardless of data format or source

A2 Model (Concrete) - Has Revision + Fix:

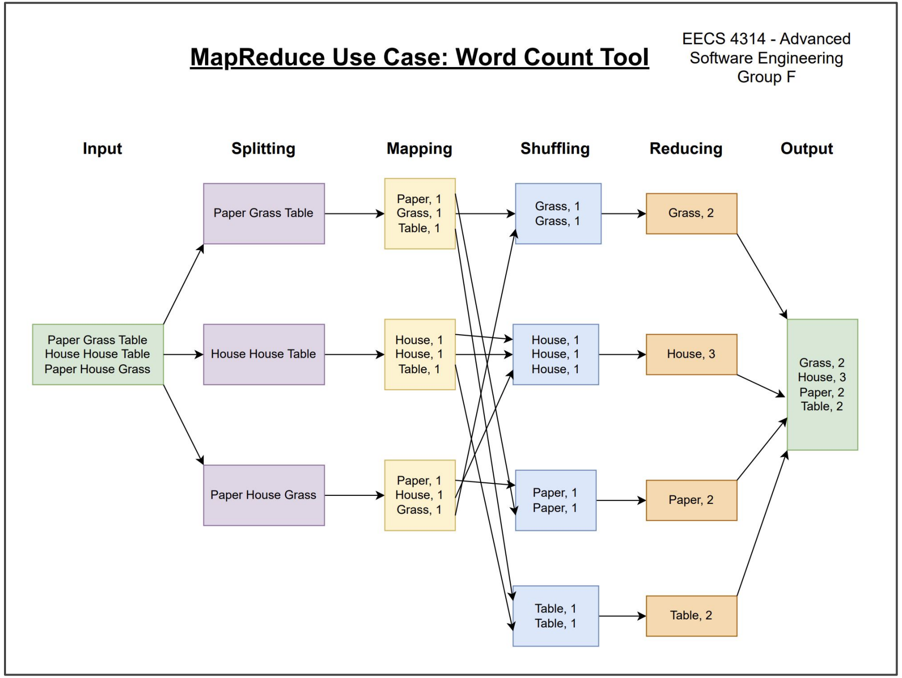
- Reveals InputOutputFormat as a pluggable subsystem, with:
  - InputFormat (controls how data is split and read)
  - OutputFormat (controls how results are written/stored)
- Supports multiple inputs/outputs (e.g., sequence files, text, logs, DB exports)

Rationale -> System Scalability

- Hadoop needs to work with diverse data sources/formats at scale
- I/O must be extensible to support new formats without rewriting Map/Reduce logic

Sequence Diagram

- No changes were made to the sequence diagram from A2 since it reflects the MapReduce workflow from the conceptual model
- It includes the core MapReduce phases in the same order as the conceptual model
- The diagram focuses on the data flow, which matches the level of abstraction in the conceptual model



MapReduce Use Case: Word Count Tool

EECS 4314 - Advanced
Software Engineering
Group F

Input

Splitting

Mapping

Shuffling

Reducing

Output

Paper Grass Table
House House Table
Paper House Grass

Paper Grass Table

House House Table

Paper House Grass

Paper, 1
Grass, 1
Table, 1

House, 1
House, 1
Table, 1

Paper, 1
House, 1
Grass, 1

Grass, 1
Grass, 1

House, 1
House, 1
House, 1

Paper, 1
Paper, 1

Table, 1
Table, 1

Grass, 2

House, 3

Paper, 2

Table, 2

Grass, 2
House, 3
Paper, 2
Table, 2

Concurrency and Development Teams

Parallel execution of tasks

- Conceptual: map and reduce tasks run in parallel across cluster nodes
- Concrete: MapReduce schedules independent map and reduce tasks across worker nodes to run concurrently

Phase coordination

- Conceptual: reducers start after map tasks complete
- Concrete: MapReduce manages the shuffle phase and sorts outputs before reducers start

Task scheduling

- Conceptual: Task distribution for MapReduce jobs is handled by YARN, which allocates cluster resources
- Concrete: MapReduce coordinates with YARN's ResourceManager and NodeManagers to schedule map and reduce tasks concurrently across worker nodes

Concurrency and Development Teams

Parallel development

- The MapReduce architecture is modular, with clear subsystem boundaries and defined APIs
- This modularity allows developer teams to implement components independently without conflicts

Special maintenance

- Complex parts of MapReduce, such as the shuffle and sort phase, are isolated as separate modules with clear responsibilities
- This allows specialized teams to maintain or extend these parts without affecting other components

Limitations of our findings

- We only looked at the code structure, not how Hadoop behaves at runtime
- We left out features that weren't directly related to MapReduce (like security, native libraries, or extra storage tools).
- Interactions between YARN and HDFS are much more dynamic than what our static diagrams can show, so our results simplify how resources and data are actually managed on a cluster.
- The conceptual model used older MapReduce ideas, while the concrete model reflects newer versions, so part of the "difference" is due to version changes

Lessons Learned

- Software architecture is not static, and instead evolves over time based on needs, as seen by the difference in Mapreduce v1 and v2
- Divergences tend to be much more common than absences, due to concrete models revealing higher levels of complexity
- Conceptual models focus on clarity and help in understanding the overall flow of a system, while concrete models expose the real dependencies and interactions required for the system to function

Closing Statements

- Performing a reflection analysis is an important step in understanding a system as it outlines the reasoning behind concrete design decisions
- Hadoop and MapReduce architectures ended up being significantly more interconnect that we initially expected due to unexpected dependencies
- Performing all three part of the project revealed that real world software systems evolve and adapt in order to meet scalability and performance demands

References

- https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/CommandsManual.html
- https://hadoop.apache.org/docs/r2.7.1/hadoop-yarn/hadoop-yarn-site/