FRAMEWORK OF INFORMATION SHARING AND PRIVACY PRESERVATION FOR INTEGRATING SOCIAL NETWORKS

Sharing Insensitive Information

- To maximize information sharing while preserving privacy, the following steps can be taken:
- Define the critical information needed by organization(OP) for the social network analysis task
- Apply data generalization techniques to GQ(social network of OQ) to protect privacy. Generalization techniques involve grouping or rounding data to a certain level of granularity to protect privacy. This will reduce the risk of sensitive information being revealed.

- **Use a privacy metric such as τ-tolerance**
  - degree of privacy leakage in the generalized GQ.
  - the amount of privacy leakage that is acceptable for the social network analysis task.

- The research problem involves sharing information between two organizations, while preserving the sensitivity of identities and network structure.

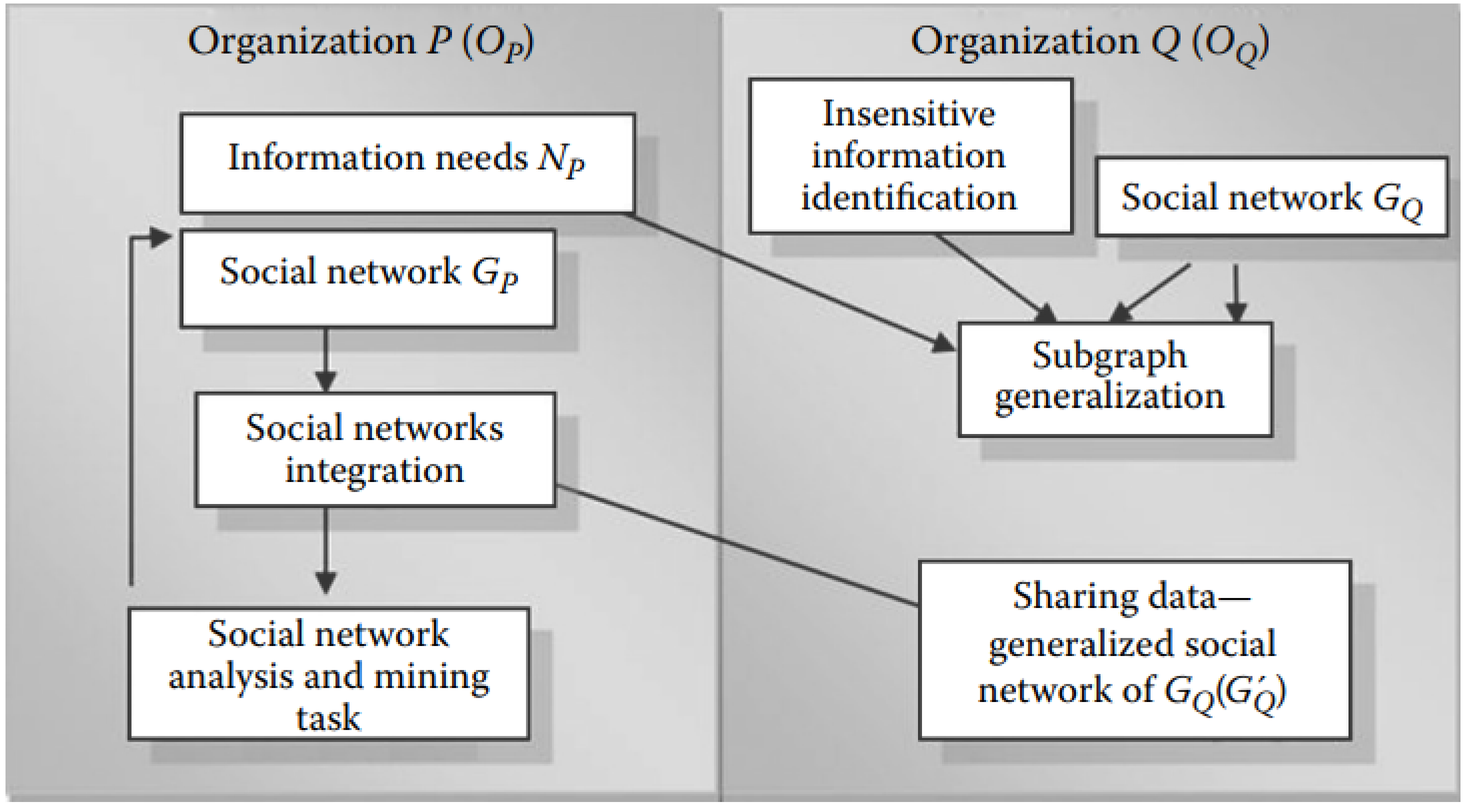
- Some identities are considered insensitive, and their information can be shared without compromising the confidentiality of the network.

- Sensitivity of an identity between two organizations is defined as sensitivity(u, Op, Oq) and is based on whether a referral or source of the identity can be obtained from the organizations.

- Fundamental centrality analysis is the focus of this work, and centrality measures such as degree and shortest distance between nodes are important attributes for computation.

- Generalization of information is required to preserve sensitive information while allowing for more accurate estimation of required information for centrality measures.

- Attributes that can be considered for generalization include the number of nodes in a subgroup, the diameter of a subgroup, the distributions of node degrees, and the eccentricity of insensitive nodes


Organization P (O_P)
Organization Q (O_Q)
Information needs N_P
Social network G_P
Social networks integration
Social network analysis and mining task
Insensitive information identification
Social network G_Q
Subgraph generalization
Sharing data—generalized social network of G_Q(G'_Q)

Definitions of τ-Tolerance of Privacy Leakage

- τ-Tolerance of privacy leakage on a sensitive node: 1. The identity of a sensitive node cannot be identified as one of τ or fewer possible known identities.

- τ-Tolerance of privacy leakage on the adjacency between an insensitive node and a sensitive node: 1. The identity of an insensitive node is known but its adjacency with other sensitive nodes is not known.

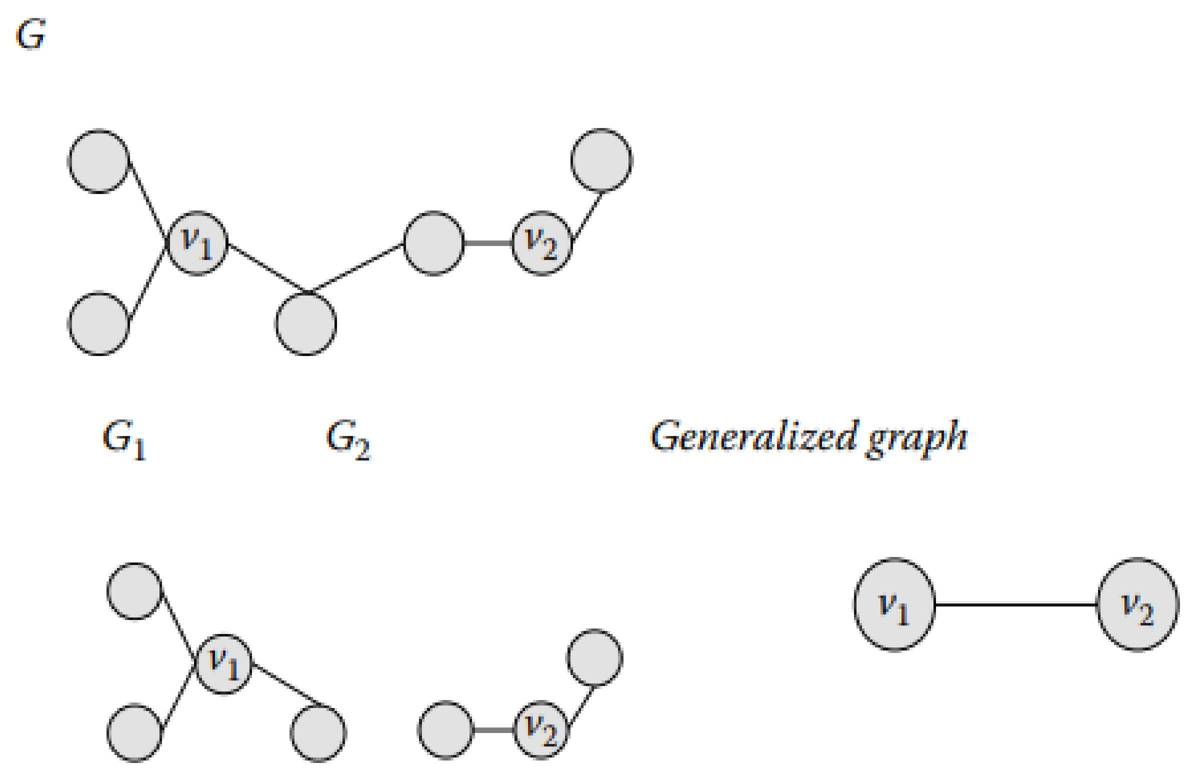
The sensitivity of an identity u

$$\text{sensitivity}(u, O_p, O_q) = \begin{cases} 0 & \text{if Refer}_{O_p}(u, O_q) = 1, \text{Refer}_{O_q}(u, O_p) = 1, \text{or Source}_{O_p O_q}(u) = 1 \\ 1 & \text{else} \end{cases}$$

Subgraph generalization generates a generalized version of a social network

- A connected subgraph is transformed into a generalized node with probabilistic model of attributes.
- A subgraph is denoted as $G' = (V', E')$, where $V' \subset V$, $E' \subset E$, $E' \subseteq V' \times V'$.
- The subgraphs must be mutually exclusive and exhaustive.
- The constructed subgraphs can be based on n-neighborhood or k-connectivity.
- The n-neighborhood graph of a node v is denoted as n-neighbor(v,G).
- The k-connectivity of a graph is the minimum number of nodes/edges whose removal results in a disconnected graph.

Illustrations of generating subgraphs



Probabilistic Model of Generalized Information

- determine the generalized information to be shared
- Achieve the following objectives:

(i) is useful for the social network analysis task after integration,

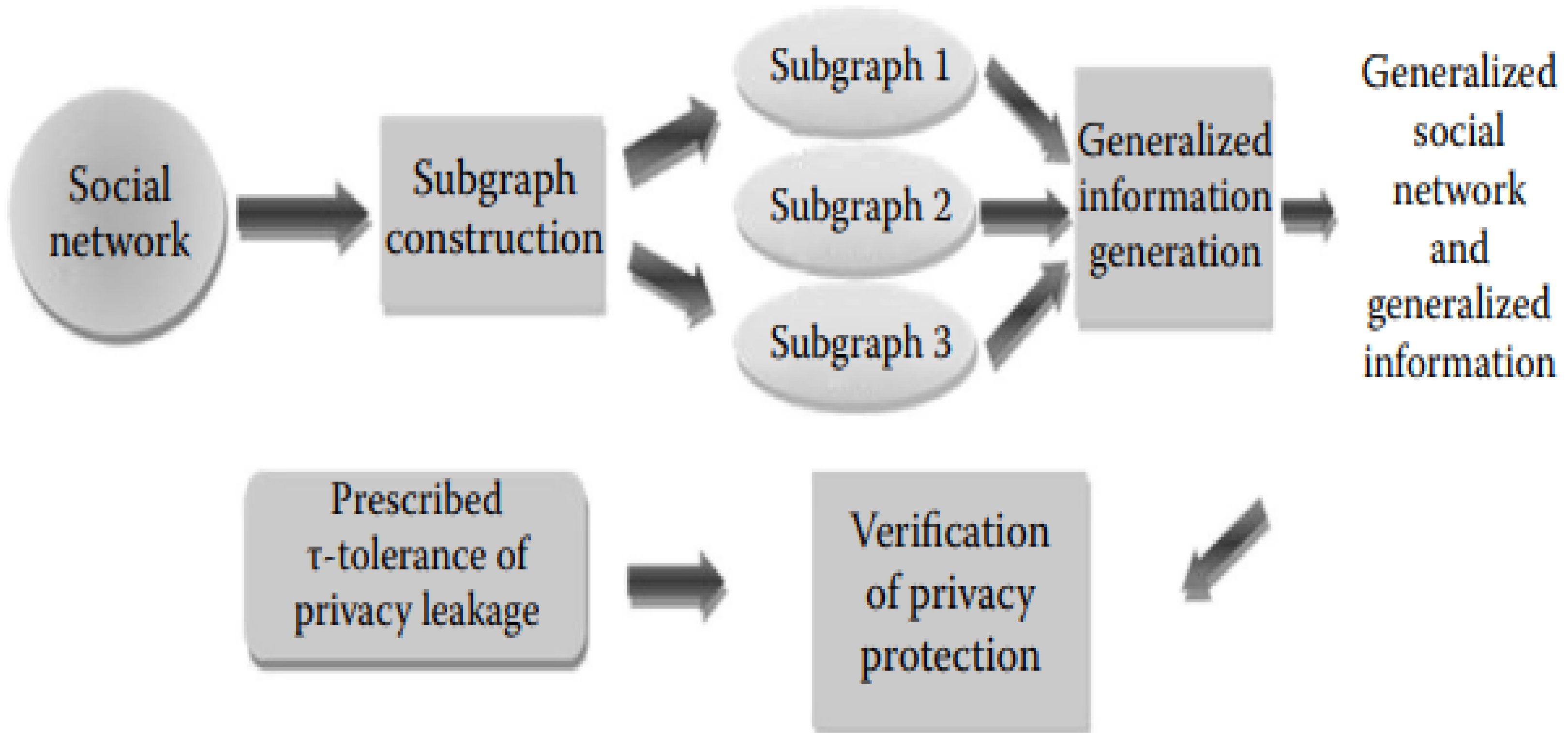
(ii) preserves the sensitive information, and

(iii) is minimal so that unnecessary information is not released

Integrating generalized social network for social network analysis task

1. The subgraph generalization approach is presented in Figure 34.5.
2. The approach involves developing techniques to make use of shared data from multiple organizations' social networks to accomplish social network analysis tasks.
3. The result of a social network analysis task is denoted as $\Im(\ )$ G, where G is a social network.
4. If organization Q shares its social network, GQ, with organization P, P can integrate GP and GQ to G and obtain a social network analysis result $\Im(\ )$ G with higher accuracy than $\Im(\ )$ GP.
5. However, Q can only share the generalized social network, GQ′, due to privacy concerns.
6. The integration technique, denoted as I G( ) P Q ,G′, should be capable of utilizing the useful information in GQ′ and integrating it with GP to achieve accuracy close to $\Im(\ )$ G.
7. An experiment was conducted using the Global Salafi Jihad terrorist social network with 366 nodes and 1275 ties, using the KNN subgraph generalization technique and closeness centrality as the social network analysis task.
8. The results showed that using the generalization approach improved the performance of closeness centrality by more than 35%.
9. The proposed approach of integrating social networks is promising

Framework of subgraph generalization approach


Social network
Subgraph construction
Subgraph 1
Subgraph 2
Subgraph 3
Generalized information generation
Generalized social network and generalized information
Prescribed τ-tolerance of privacy leakage
Verification of privacy protection

Social Network Integration and Analysis with Privacy Preservation

Introduction

1. Social networks have become popular with the advancement of Web 2.0 technologies and can be used in various domains such as marketing, epidemiology, homeland security, sociology, psychology, and management.

2. Social network data is usually owned by an individual organization or government agency, and sharing it between organizations is usually prohibited due to privacy concerns.
3. Techniques such as k-degree anonymity and k-anonymity achieved by edge or node perturbation are proposed for privacy preservation in social network data. However, such anonymization is not suitable for integration of social networks or other social network analysis and mining tasks.

- Sharing insensitive and generalized information can support social network analysis and mining while preserving privacy.
- The chapter discusses a generalization approach for integrating social networks with privacy preservation, motivated by the scenario of two local law enforcement units having their own criminal social networks.
- Social network analysis techniques such as centrality measures and similarity measures are commonly used to determine the relative significance of a node in a social network and compute the similarity between two subgroups within a social network.
- Recent development in link mining of social networks focuses on object ranking, object classification, group detection, entity resolution, and link prediction.
- Limitations of current approaches for social network analysis and privacy preservation are discussed.

1. An organization wants to publish anonymized relational data to enable data mining and analytics while ensuring privacy preservation.
2. Simply removing personally identifiable information (PII) such as names and identification numbers is not sufficient for privacy preservation.
3. Quasi-identifiers are a set of attributes that can support a linking attack and potentially reveal sensitive information about an individual.
4. An example linking attack is described where medical records and voter registration records are cross-checked using quasi-identifiers to identify a person with HIV.
5. Several approaches for privacy preservation of relational data have been developed, including k-anonymity, l-diversity, and t-closeness

k-anonymity

- Goal of k-anonymity
  - to ensure that at least k individuals share the same combination of age, gender, and ZIP code, making it impossible to uniquely identify any one of them based on this information.

One possible anonymized version of the dataset

- Original datasets

Age	Gender	Medical Condition
34	Female	Diabetes
45	Male	Hypertension
27	Female	Asthma
19	Male	Diabetes
50	Female	Hypertension
42	Male	Asthma

Age Group	Gender	Medical Condition	Count
20-39	Female	Diabetes	2
40-59	Male	Hypertensi on	2
20-39	Female	Asthma	1
20-39	Male	Diabetes	1
40-59	Female	Hypertensi on	1
40-59	Male	Asthma	1

PRIVACY PRESERVATION OF SOCIAL NETWORK DATA

- Naive approach of removing identities:
  - This approach simply removes the identities of nodes and only reveals the edges of a social network. However, Backstrom et al. (2007) proved that it is possible to discover whether edges between specific targeted pairs of nodes exist or not by active or passive attack
- Anonymization models
  - Anonymization models are proposed to tackle active and passive attacks and preserve the privacy of node identities in a social network.
  - These models include k-candidate anonymity, k-degree anonymity, and k-anonymity.
  - The main idea behind these models is to increase the difficulty of being attacked based on the notion of k-anonymity in relational data.
  - Edge anonymization models have also been proposed for social networks with labeled edges rather than labeled nodes.

PRIVACY PRESERVATION OF SOCIAL NETWORK DATA(contd)

- Node or edge perturbation:
  - The technique used to achieve the above anonymities is edge or node perturbation. By adding and/or deleting edges and/or nodes, a perturbed graph is generated to satisfy the anonymity requirement. Adversaries can only have a confidence level of 1/k of discovering the identity of a node by neighborhood attacks
- Secure multiparty computation (SMC):
  - Another approach to privacy-preserving social network
  - set of functions that multiple parties wish to jointly compute,
    - and each party has its own private inputs.
  - By preserving the private inputs, SMC uses cryptography technology to compute the joint function.
  - disadvantages of the cryptography approach
    - as the encrypted data can be attacked and recovered by the malicious party,
    - the complexity of SMC may not be computationally feasible for large-scale social network data

Cloud-Centric Assured Information Sharing for Social Networks (CCAIS)

R Devika

INTRODUCTION

- Cloud-Centric Assured Information Sharing for Social Networks
  - framework that provides a secure and efficient way to share information on social networks
  - cloud-:central platform to manage and control access to the shared information.
- Addresses challenges associated with social network information sharing
  - privacy, security, scalability, and accessibility.
- to provide a scalable and flexible platform for information sharing.
  - Virtualization
- includes auditing and monitoring capabilities
  - detect and prevent unauthorized access or usage of the shared information.

Several companies and organizations have adopted cloud-centric assured information sharing technology for social networks

- U.S. Department of Defense (DoD) uses a cloud-based platform called milCloud 2.0 to share information among different branches of the military
- Microsoft's Azure Active Directory
  - access control and identity management capabilities for cloud-based applications and services
  - Microsoft with its SharePoint and OneDrive services, Google with its Google Drive and Google Docs applications, and Dropbox

AIS:Assured Information Sharing

- the need for assured information sharing (AIS) in the cloud, particularly for government agencies.
- Existing AIS tools are not scalable enough to handle large amounts of data such as social media users.
- the design and development of a cloud-centric policy manager and data manager to enforce policies and securely store and retrieve data in the cloud, respectively.
- Semantic web technologies are used to represent and reason about social media data.
- the design and implementation of cloud-based assured information-sharing systems (CAISS) and CAISS++.
- Formal policy analysis and implementation approaches for these systems are presented.

Cloud-centric Assured Information Sharing System (CAISS)

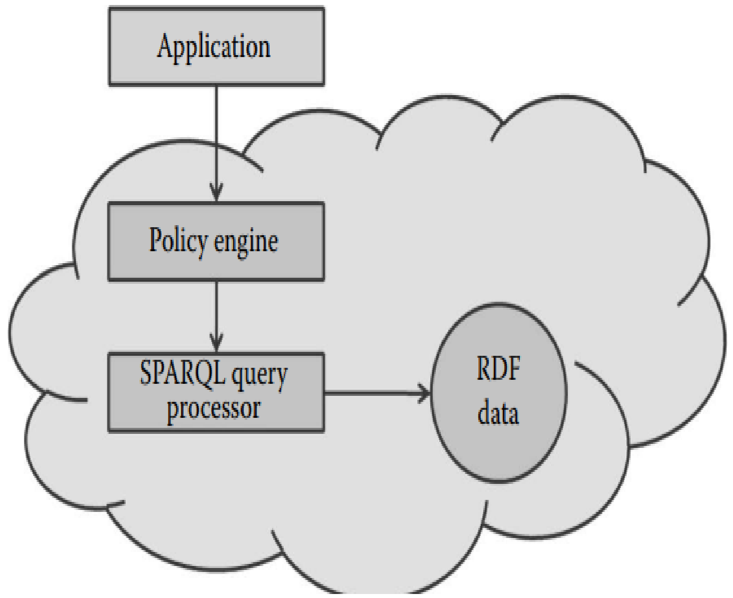
1. Enhancement of existing tools for AFOSR on secure cloud query processing with semantic web data and semantic web-based policy engine to develop CAISS.
2. Use of RDF-based policy engine to enforce access control, redaction, and inference control policies on data represented as RDF graphs.
3. Use of cloud SPARQL query engine for RDF data using the Hadoop/MapReduce framework.
4. Replacement of XACML-based policy engine with a semantic web-based policy engine for better expressiveness and to avoid performance bottleneck.
5. Porting of the current policy engine to the cloud environment and integration with the SPARQL query engine for federated query processing in the cloud.
6. Use of RDF as the data model for data interoperability, easy adaptability, and creating data-centric applications.
7. Incorporation of information-sharing policies into the policy engine.
8. Support for path queries in the enhanced SPARQL query processor.
9. Design of an algorithm to determine the candidate triples as an answer set in a distributed RDF graph for path queries

Enhanced Policy Engine.

1. The current policy engine does not operate in a cloud environment.
2. The RDF policy engine will be ported to the cloud environment and integrated with the SPARQL query engine for federated query processing.- **a single query that retrieves data from all the databases and combines the results into a unified view**.
3. RDF is used as the data model for achieving data interoperability, creating data-centric applications, and adapting to changing user requirements.
4. The policy engine is flexible and can handle any type of policy that could be represented using RDF and horn logic rules.
5. The policy engine currently addresses confidentiality, privacy, and redaction policies, but needs to incorporate information-sharing policies.
6. Simulation studies for incentive-based AIS and AIS prototypes in the cloud have been conducted.
7. Information-sharing policies, such as "US gives information to UK provided UK does not share it with India," have been defined in RDF and will be processed by the enhanced policy engine

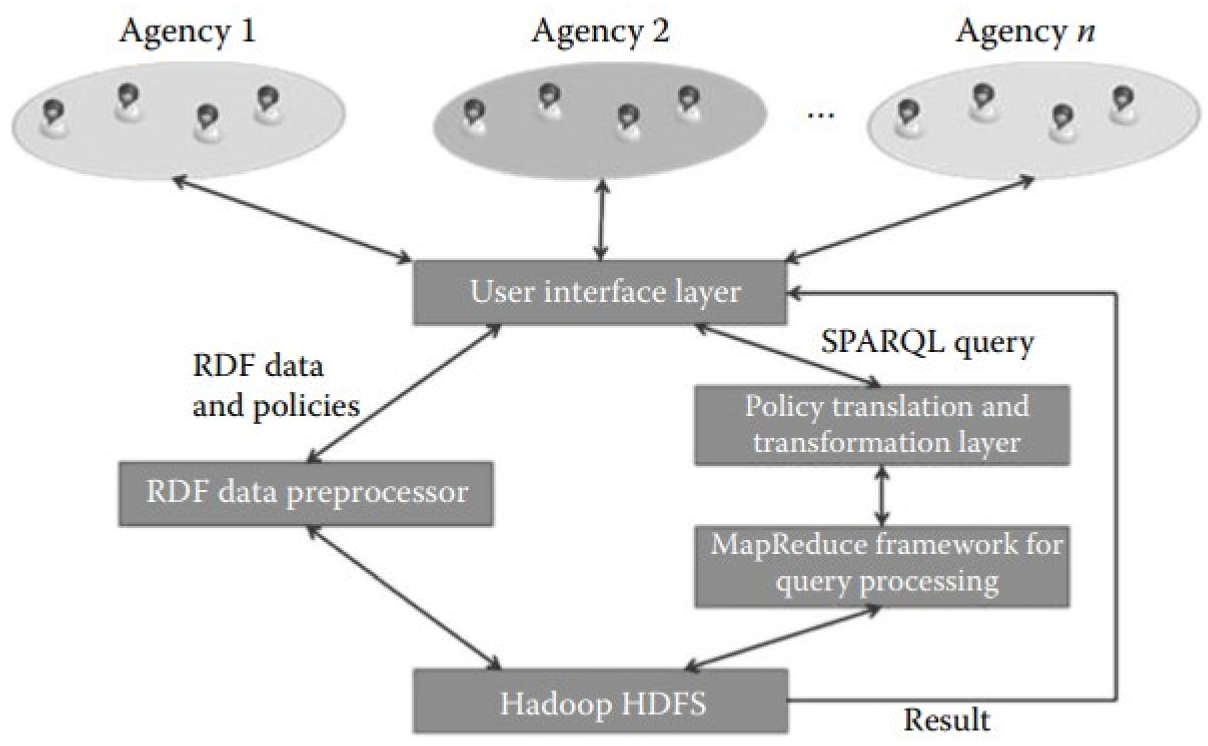
Overview of the CAISS architecture

- cloud-centric RDF policy engine + the enhanced SPARQL query processor



1.Ensure that RDF-based policies can be stored in the existing storage schema used by the query processor.(R2RML-relational database data into RDF which can be used in semantic web application)
2.Ensure that the enhanced query processor is able to efficiently evaluate policies (i.e., path queries) over the underlying RDF storage.
3.Conduct a performance evaluation of CAISS to verify that it meets the performance requirements of various participating agencie

The concept of operation of CAISS



- Multiple agencies will share data in a single cloud.
- The enhanced policy engine and the cloud-centric SPARQL query processor will enforce the information-sharing policies.

Some limitations of CAISS

- :

- i. Policy engine: The RDF-based policy engine used by CAISS has limited expressivity. In CAISS++, the plan is to make use of OWL, which is much more expressive than RDF, to model security policies through organization-specific domain ontologies as well as a system-wide upper ontology.
- ii. Hadoop storage architecture: CAISS uses a static storage model, which means that users can only provide RDF data during the initialization step and are not allowed to update the existing data. In CAISS++, a flexible storage model is provided where users are allowed to append new data to the existing RDF data stored in Hadoop Distributed File System (HDFS).
- iii. SPARQL query processor: CAISS only supports simple SPARQL queries that make use of basic graph patterns (BGPs). In CAISS++, support for other SPARQL query operators such as FILTER, GROUP BY, ORDER BY, etc., will be added. Additionally, CAISS uses a heuristic query optimizer that aims to minimize the number of MapReduce jobs required to answer a query. CAISS++ will incorporate a cost-based query optimizer that will minimize the number of triples that are accessed during the process of query execution

Cost-Based Query Optimizer Vs Heuristic Query Optimizer

Feature	Cost-Based Query Optimizer	Heuristic Query Optimizer
Optimization approach	Uses statistical models to estimate the cost of different execution plans	Uses a set of predefined rules or heuristics to select the best execution plan
Performance	Slower, but generally produces better execution plans for complex queries	Faster, but may not always produce the best execution plan
Resource requirements	Requires more resources (memory, CPU) to perform its calculations	Requires fewer resources than cost-based optimizer
Suitability for different types of queries	Better suited for complex queries with large data sets	Better suited for simple queries or queries with small data sets
Accuracy	More accurate due to its use of statistical models	Less accurate, as it relies on predefined rules and heuristics
Availability	Available in most modern RDBMS	May not be available in all RDBMS or may be limited in functionality compared to cost-based optimizer

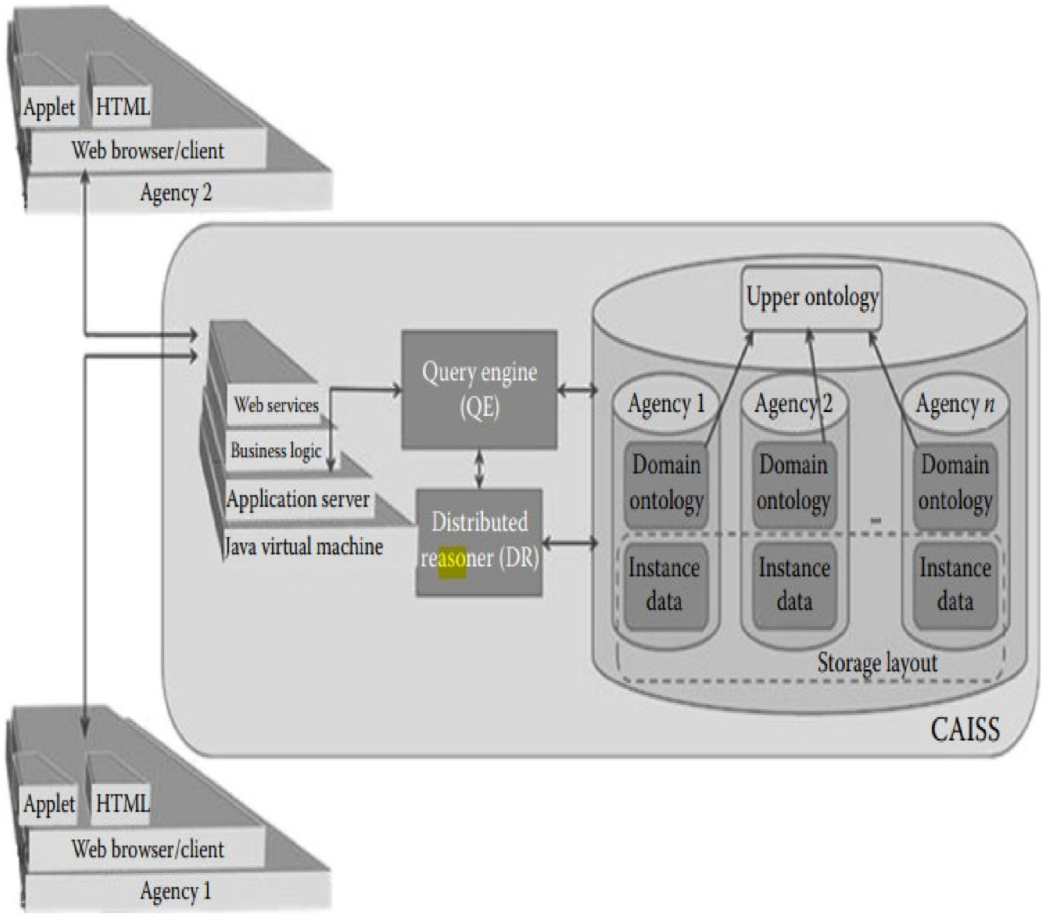
The design of CAISS++ involves three main tasks

- OWL-based policy engine
  - domain-specific ontologies
    - knowledge about a particular domain
  - upper ontology
    - high-level representation of concepts and relationships
      - common across multiple domains
    - what information can be shared and under what conditions.
- RDF processing engine
  - sophisticated storage architecture
    - Graph database-
  - efficient query processor for RDF data
    - Apache Jena
- Integration framework
  - The final task is to combine the policy engine
  - with the processing engine into an integrated framework.

CAISS++ trade-off between simplicity of design versus its scalability and efficiency

- Centralized CAISS++ prioritizes **simplicity**
- decentralized CAISS++ prioritizes **scalability and efficiency**

Centralized CAISS++

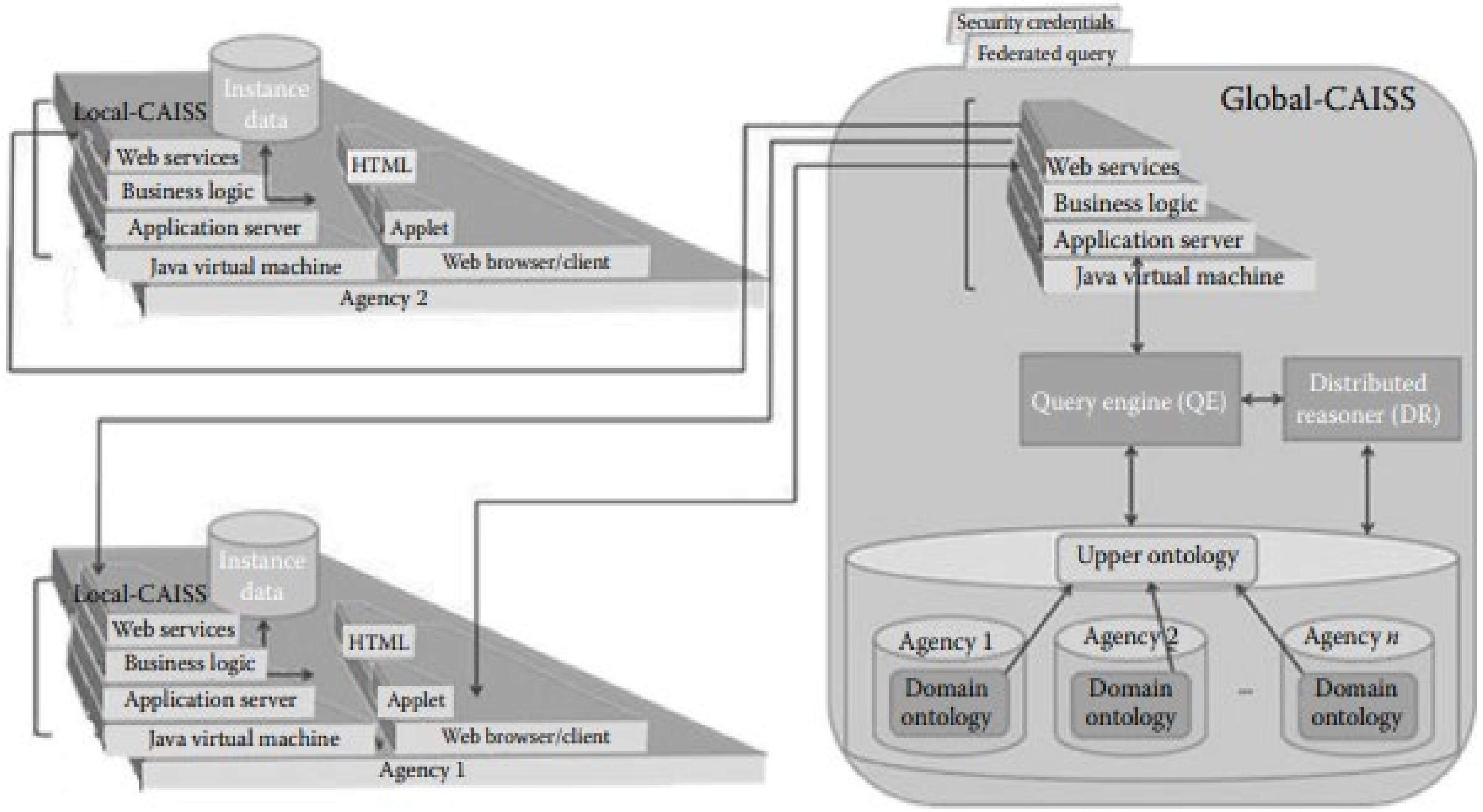


Features	CAISS	Centralized CAISS++
Security Policy Encoding	RDF	OWL sublanguage
SPARQL Query Processor	Limited subset	Maximum expressivity
Storage Architecture	Hadoop	Hadoop
Data Insertion	Initialization	Initialization
Data Update	Delete + Insert	Append

Pros and Cons of Centralized CAISS++ Approach

Pros	Cons
Easy to manage and maintain	Single point of failure
Supports complex security policies	High risk of data breach
Maximum expressivity of SPARQL	Limited scalability due to centralized nature
Supports data insertion and update	Limited fault tolerance

Decentralized CAISS++

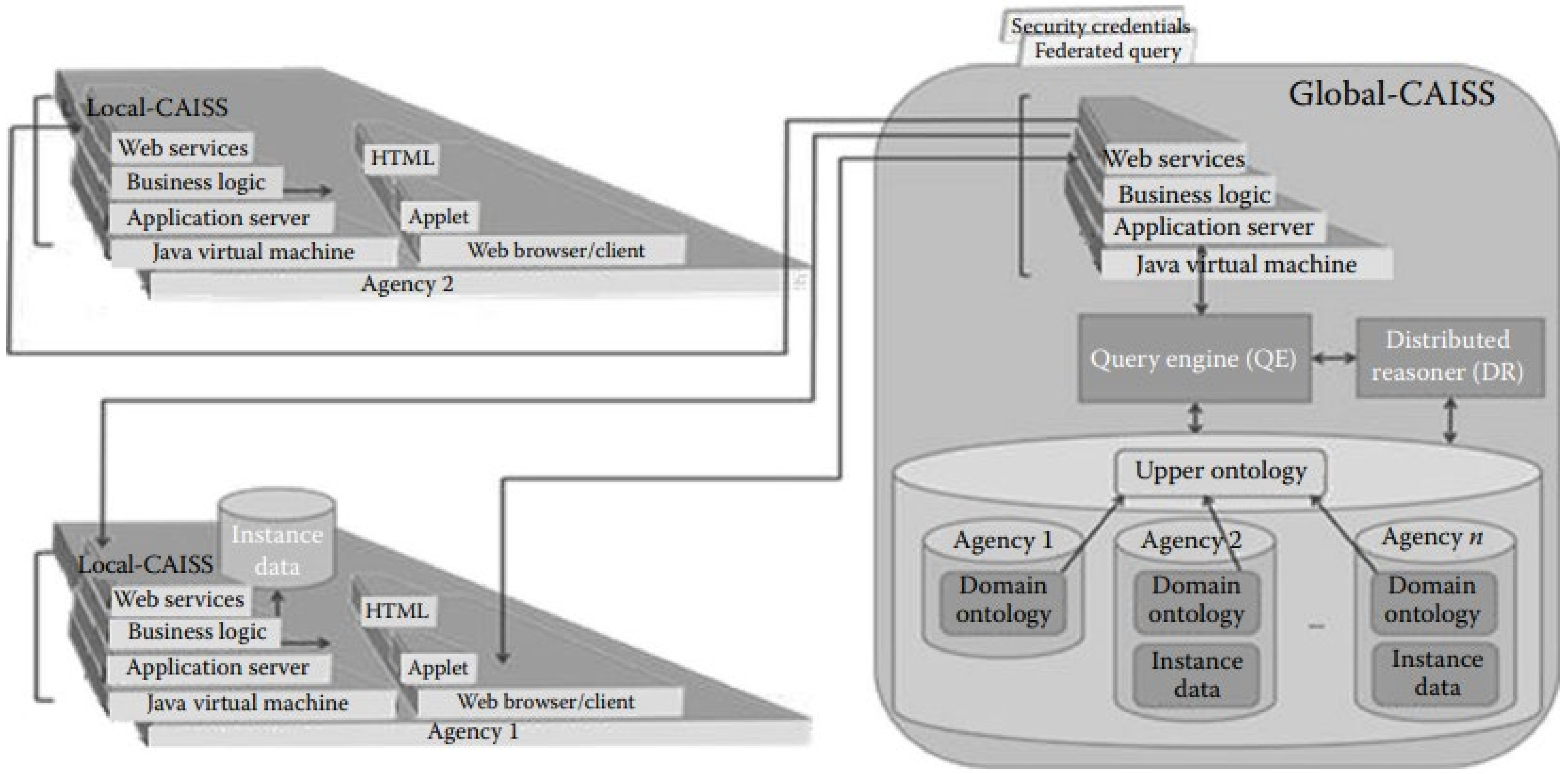


Pros	**Cons**
Distributed storage enhances scalability	Complexity in managing distributed domain ontologies
Increased fault tolerance	Additional overhead in query processing due to federation
Higher security due to separate storage	Limited expressivity of SPARQL
No single point of failure	Additional overhead in maintaining distributed storage

Hybrid CAISS++

- leverages the benefits
  - centralized CAISS++
  - decentralized CAISS++
- the freedom to choose the most suitable option for their needs
- made up of global CAISS++ and a set of local CAISS++

Hybrid CAISS++ architecture

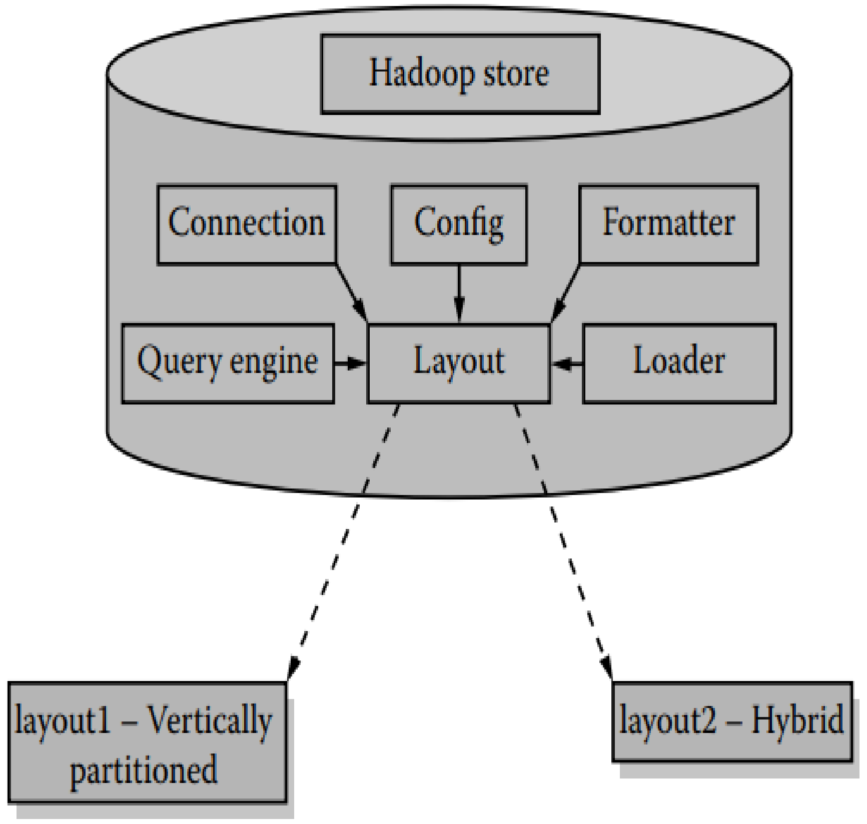


Illustrate the operation :Use-case scenarios

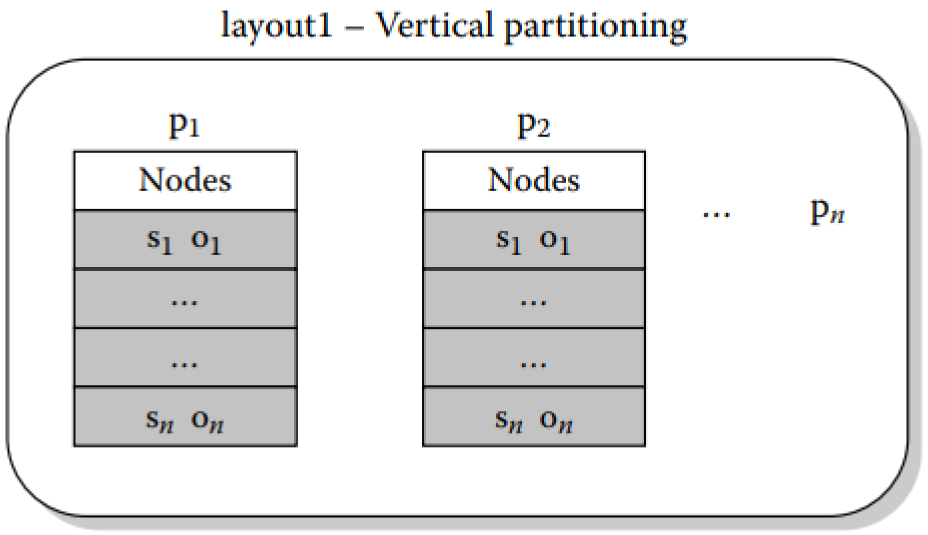
Case	Approach	Agencies	Knowledge Bases	Cloud Storage	Information Sharing
1	Centralized	Agency 1, Agency 2	Domain ontology and instance data	Shared cloud storage at global CAISS++	Proceeds as in centralized CAISS++
2	Decentralized	Agency 3, Agency 4, Agency 5	Respective domain ontologies	Central cloud storage accessible only to group members	Proceeds as in decentralized CAISS++
3	Mixed	User belonging to multiple groups	Data shipped to central cloud storage for centralized group	Local CAISS++ used for decentralized group	Depends on group approach

Hadoop storage architecture used by CAISS++

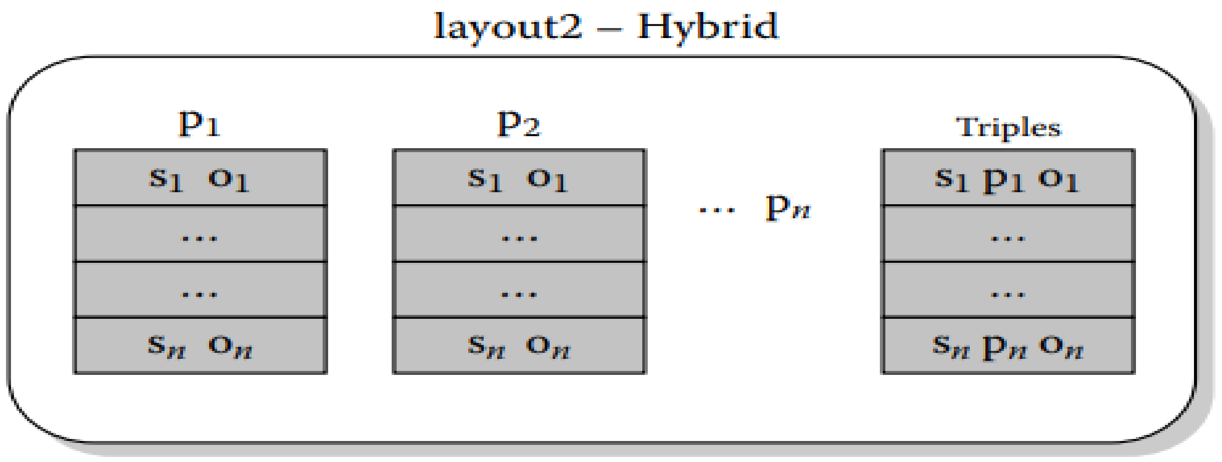
•A Store represents a single RDF dataset and can contain several RDF graphs.
•The framework includes a Layout formatter, Loader, Query engine, Connection, and Config blocks.
•All operations on an RDF graph are performed on the underlying layout.
•The Layout formatter block deletes all triples in an RDF graph while preserving the directory structure used to store that graph.
•The Loader block performs loading of triples into a layout.
•The Query engine block allows a user to query a layout using a SPARQL query.
•The querying mechanism involves translating a SPARQL query into a pipeline of MapReduce jobs and executing it on a layout.
•The Connection block maintains the necessary connections and configurations with the underlying HDFS.
•The Config block maintains configuration information such as graph names for each of the RDF graphs that make up a store.
•The framework uses naming conventions for storing RDF data in separate files under different HDFS folders.



Vertically partitioned layout



Hybrid layout.



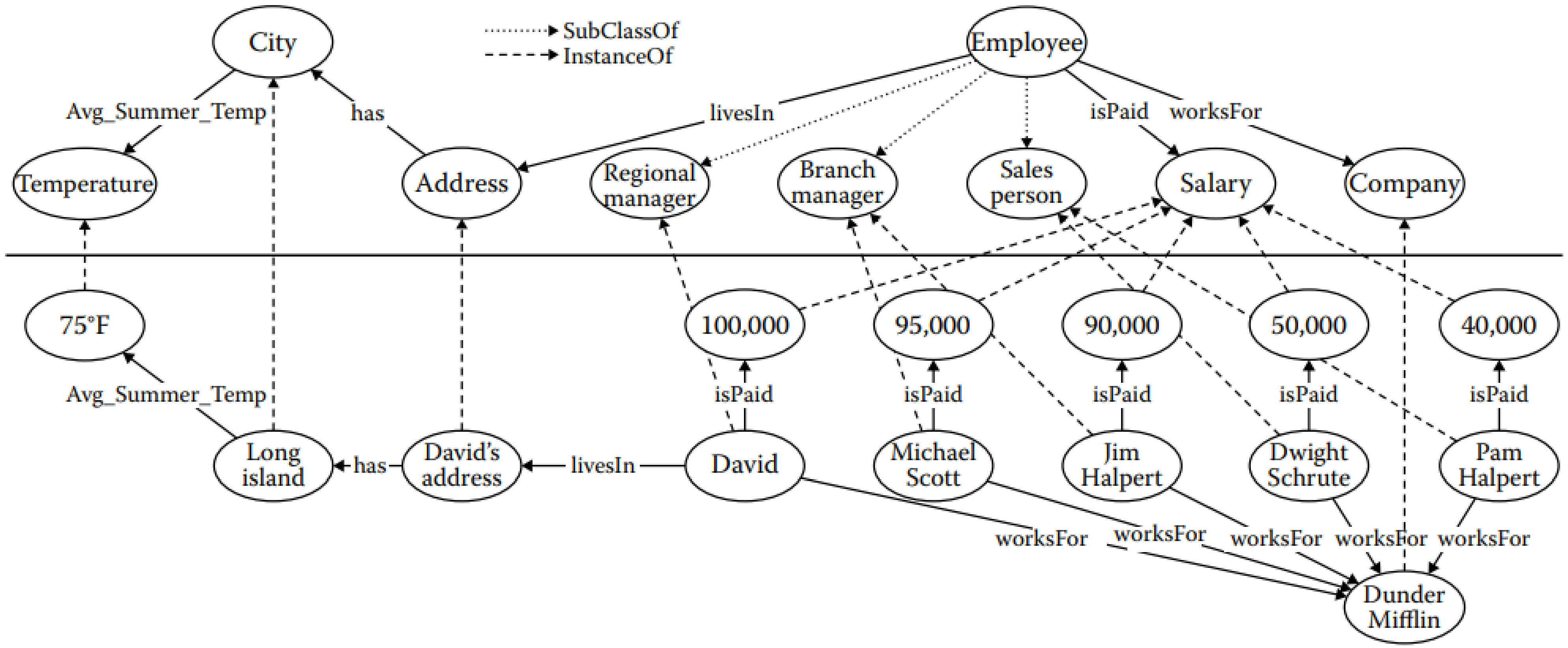
Secure Cloud Query Processing for Semantic Web-Based Social Media

R Devika

Introduction

- Semantic web technologies are being used by various businesses and industries
  - data integration and analysis, including social media data.
- **RDF is used to represent social networks**,
  - OWL is used to represent ontologies for understanding social networks.
  - SPARQL is used to query social networks,
  - SWRL is used to analyze them.
- **Ontologies are a robust way to represent knowledge and social media data**
  - describing concepts, properties, relationships, and restrictions.
- **Most RDF stores are not primarily concerned with security,**
  - to incorporate security features, especially in Jena.
- **Large RDF stores can be stored and retrieved from cloud computers using Hadoop,**
  - splits large jobs into smaller jobs and combines the results.
- **Access control for RDF stores in Hadoop has received little attention.**
- The chapter discusses a system that
  - implements access control for RDF data on Hadoop using a token-based system,
  - with conflicts resolved using time stamps of access tokens.
  - to scale well to extremely large data sets and addresses access control at the level of users, subjects, objects, and predicates.
- The architecture was implemented and tested on benchmark data in several stages: query rewriting, embedded enforcement, and postprocessing enforcement.
- The entire system is being implemented on Hadoop, an open-source cloud computing environment.

Knowledge base



Access tokens (AT)

- Access tokens (AT) are used to grant access to security-relevant data.
- An agent who possesses an AT is allowed to view the data that the token permits access to.
- The AT is represented by a positive integer value

Access token tuples (ATT)

- Access token tuples (ATT) have the form <AccessToken, Element, ElementType, ElementName>.
- Element can be Subject, Object, or Predicate, and ElementType can be URI, DataType, Literal, Model, or BlankNode.
- There are six access levels available, including Predicate data access, Predicate and subject data access, Predicate and object, Subject access, Object access, and Subject model level access.
- Agents possessing different access levels may access specific information associated with a particular subject, predicate, or object.
- The article provides an example of how an agent with model level access may read the average summer temperature of Long Island if they possess David's ATT.

Predicate data access:

•An agent can access a predicate file using an ATT associated with a specific predicate.

•The example given is <1, Predicate, isPaid, _>, which allows the agent to read the entire predicate file for "isPaid".
•Access level is defined for a specific predicate, giving the agent access to all information associated with it, subject to policy restrictions

.
•If the access level is limited to "isPaid", the agent can access all payment-related information.

•Additional restrictions or policies may apply, such as access to payment information for specific users or time periods.

Predicate and subject data access

1. Agents with a subject ATT can access data associated with a particular subject, which can be either a URI or a DataType.
2. Combining a subject ATT with a predicate data access ATT having the same AT grants the agent access to a specific subject of a specific predicate.
3. For example, combining the ATTs <1, Predicate, isPaid>requester has the permission to access resources with the predicate "isPaid"
4. and <1, Subject, URI, MichaelScott> subject of the resource as "MichaelScott" with a URI (Uniform Resource Identifier) that uniquely identifies it.

By combining these two attributes, an agent with attribute <1> would be able to access the resource that matches the predicate "isPaid" and subject "MichaelScott"

1. Similarly, Predicate and DataType ATTs can be combined to permit access to subjects of a specific data type over a specific predicate file.
2. The article omits descriptions of the different subject and object variations of each of the remaining access levels.

Predicate and object:

1.. This access level permits a principal to extract the names of subjects satisfying a particular predicate and object.

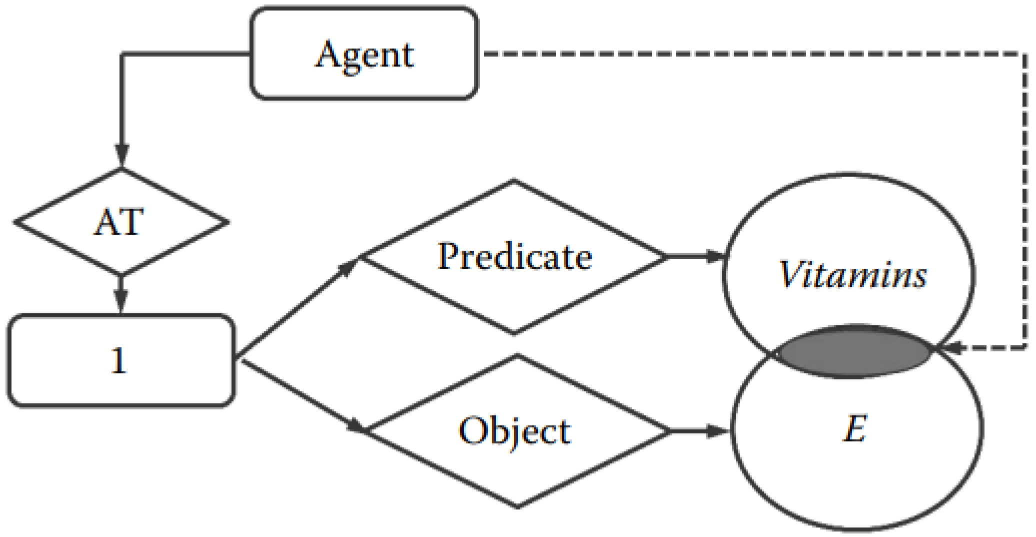
2.For example, with ATTs <1, Predicate, hasVitaminE> and <1, Object, vitaminE>, an agent possessing AT1 may view the names of subjects (e.g., foods) that have vitamin E.

3.More generally, if X1 and X2 are the set of triples generated by Predicate and Object triples (respectively) describing an AT, then agents possessing the AT can extract the names of subjects satisfying the predicate and object described by X1 and X2.

Subject access:

1. This access level allows an agent to read the subject's information over all the files.
2. It is one of the less restrictive access levels.
3. The subject can be a DataType or BlankNodE

Conjunctive combination of ATTS with a common AT



Object access

- Object access is a level of access control that grants an agent the ability to read the subjects of an object across all of its files.
- This access level is less restrictive than higher levels of access such as write or execute, as it does not allow the agent to modify or execute any actions on the object.
- The object in question can be any of the four types - a URI, a DataType, a Literal, or a BlankNode.

Subject model level access

- Subject model level access allows an agent to recursively extract all the predicates and objects associated with a subject, in order to generate a comprehensive model of the subject and its relationships with other entities.
- In the example of the book "To Kill a Mockingbird" by Harper Lee, the agent with Subject model level access could extract all predicates and objects associated with the book, author, and publisher.
- By treating each URI object as a new subject, the agent can recursively extract all predicates and objects associated with those entities as well.
- For example, by extracting the predicates and objects associated with the author "Harper Lee", the agent could discover other books written by the same author, as well as any other entities that the author is associated with.
- Similarly, by extracting the predicates and objects associated with the publisher "J. B. Lippincott & Co.", the agent could discover other authors that the publisher has worked with, as well as any other entities that the publisher is associated with.
- By continuing to recursively extract all predicates and objects associated with each entity, the agent can generate a comprehensive model that includes information such as the author's other books, the publisher's other authors, and the locations of all the publishers in the dataset.
- This model can be used to gain insights into the relationships between entities in the dataset, and can be useful for tasks such as data analysis and research. However, it can also raise privacy concerns if sensitive information is exposed

Subject model level access: extract average summer temperature of long island and provide it to david

- If an agent with model level access of David reads the average summer temperature of Long Island, it would involve recursively extracting all the predicates and objects associated with the LongIsland subject, and then extracting the object of the Avg_Summer_Temp predicate. Here are the steps that the agent would need to follow:

1. Identify the subject associated with David, which is "David lives in LongIsland".
2. Recursively extract all the predicates and objects associated with the LongIsland subject.
3. Identify the Avg_Summer_Temp predicate, which has an object of 75°F.
4. Extract the object of the Avg_Summer_Temp predicate, which is 75°F.
5. Provide the value of 75°F to David as the average summer temperature of Long Island.

Access Token Assignment

1. An access token list (AT list) is an array of one or more ATs granted to a given agent, along with a time stamp identifying the time at which each was granted.

Final Output of an Agent’s ATs.

- An agent's access to triples is determined by their AT list.
- Each AT in an agent's AT list permits access to a set of triples, which is called the AT's result set.
- The union of the result sets of all the ATs in an agent's AT list gives the set of triples accessible to the agent.
- If an agent has n ATs in their AT list, with result sets Y1, Y2, ..., Yn (respectively), then the agent may access the triples in set Y1 U Y2 U ... U Yn.
- This means that an agent's access to triples is determined by the combination of all the ATs in their AT list, with each AT granting access to a specific set of triples

Default security levels

1. Default security levels for data in the system can simplify an administrator's AT assignment burden.
2. All items in the data store have default security levels.
3. Personal information of individuals is kept private by denying access to any URI of data type Person by default.
4. This prevents agents from making inferences about any individual without explicit permission.
5. If an agent is granted explicit access to a particular type or property, they are also granted default access to the subtypes or subproperties of that type or property.

Example: Default security levels

- The example describes a file that lists things someone likes, such as hobbies or research interests.
- The file includes URIs of different types, including hobbies, research interests, and people.
- The default security level for any person's data (represented by the URI of type "Person") is set to "deny access".
- An agent named Ben has been granted an Access Triple (AT) that only allows him to see the "Likes" list, but not any specific information about individuals who are listed.
- Because of the default security level, Ben cannot see that Jim likes Jenny, who is a person and subject to the default security level of "deny access".
- However, if Ben is also granted an additional AT that explicitly allows him to see Jenny's URI, then he will also be able to see that Jenny is in Jim's "Likes" list.
- This is because the default security level system grants access to subproperties or subtypes of any property or type that an agent has explicit access to

Conflict Detection and Resolution

Suppose an agent has the following attributes in its currentAT array:

AT1: {"color": "blue", "size": "medium"}

AT2: {"color": "red", "shape": "square"}

AT3: {"color": "green", "size": "large", "shape": "circle"}

Now, the agent receives a new attribute with the following values and time stamp:

newAT: {"color": "blue", "shape": "circle"}

TSnewAT: 2022-01-01 12:00:00

The algorithm will first check for conflicts between the new attribute and the existing attributes. It will find that the new attribute is a subtype of AT3 (because it has the same "color" and a new "shape"), so a conflict occurs. The algorithm will then replace AT3 with the new attribute and its time stamp, as the time stamp of the new attribute is greater than the time stamp of AT3:

```
currentAT[] = [
{"color": "blue", "size": "medium"},
{"color": "red", "shape": "square"},
{"color": "blue", "shape": "circle", "size": "large"}, TSnewAT
]
```

The new attribute is added to the currentAT array with its time stamp, and the conflict is resolved

Algorithm

Input: AT newAT with time stamp TSnewAT
Result: Detect conflict and, if none exists, add (newAT, TSnewAT) to the agent’s AT list

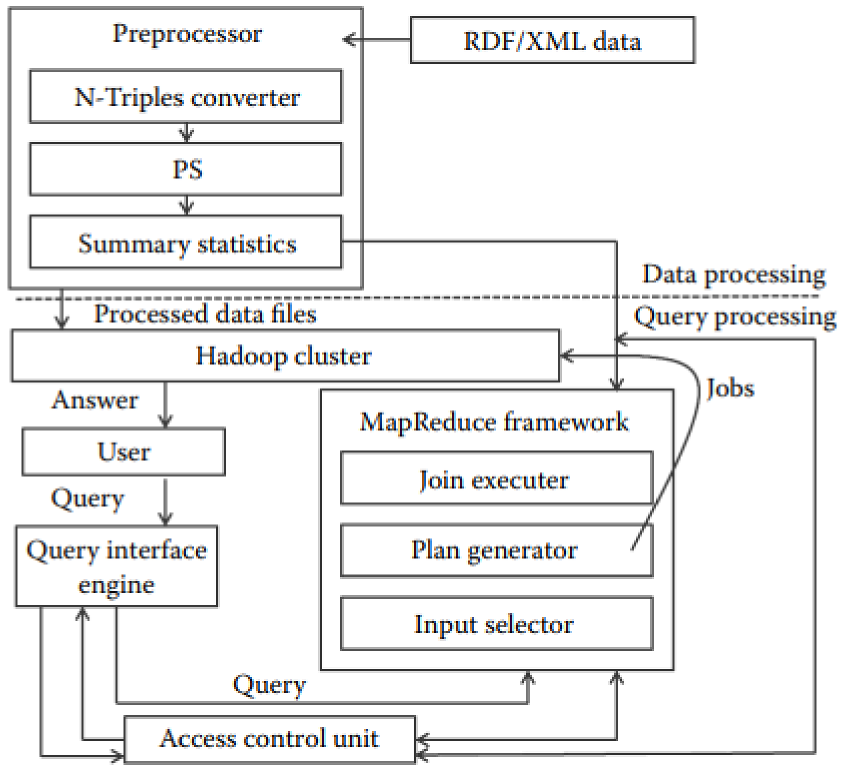
```
currentAT[] <- the ATs and their time stamps;
if no conflict exists between newAT and currentAT:
  add (newAT, TSnewAT) to currentAT
else:
  for each AT in currentAT:
    if newAT is a subset of AT and TSnewAT >= AT's time stamp:
      replace AT with newAT in currentAT
    else if AT is a subset of newAT and AT's time stamp < TSnewAT:
      replace AT with newAT in currentAT
    else if newAT is a subject or object subtype of AT and TSnewAT >= AT's time stamp:
      replace AT with newAT in currentAT
    else if AT is a subject or object subtype of newAT and AT's time stamp < TSnewAT:
      replace AT with newAT in currentAT
```

System Architecture: Scalable And Efficient Querying Of Large RDF Datasets Using The Mapreduce Framework, While Enforcing Access Control Policies To Protect Data Privacy And Security

- The LUBM dataset is used for experiments in this study.
    - evaluate the performance of Semantic Web repositories and reasoning systems
    - It consists of a synthetic dataset representing university domain data, including information about professors, courses, departments, students, and publications.
- The LUBM data generator generates data in RDF/XML serialization format.
- RDF/XML format is not suitable for the purpose of storing data in HDFS as flat files because parsing the entire file is needed to retrieve a single triple.
- N-Triples format is used instead of RDF/XML format for storing data as it contains complete RDF triples in one file line, making it convenient for MapReduce jobs.
- Data is partitioned by predicate to reduce the search space for SPARQL queries.
- In real-world RDF datasets, the number of distinct predicates is usually no more than 10 or 20.
- Files are named by predicate for simplicity; for example, all the triples containing predicate p1:pred are stored in a file named p1-pred.

Three steps: System Architecture

- ❑Data Preprocessing Component
- ❑ Query Answering Component
- ❑Data postprocessing



Data Preprocessing Component

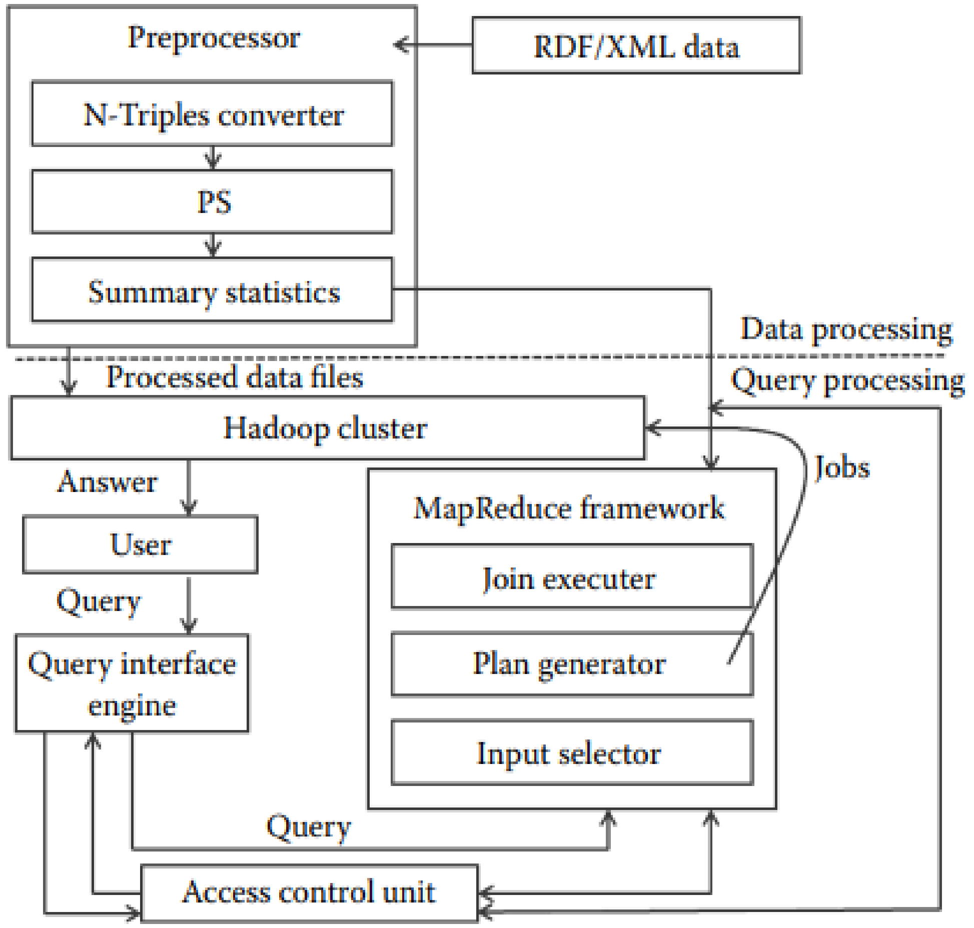
- Converts RDF/XML data to N-Triples serialization format using N-Triple converter
- Splits N-Triples data into predicate files using Predicate Split component
- Gathers summary statistics from the output of the Predicate Split component and delivers them to Hadoop Distributed File System (HDFS) using summary statistics generator component

Query Answering Component:

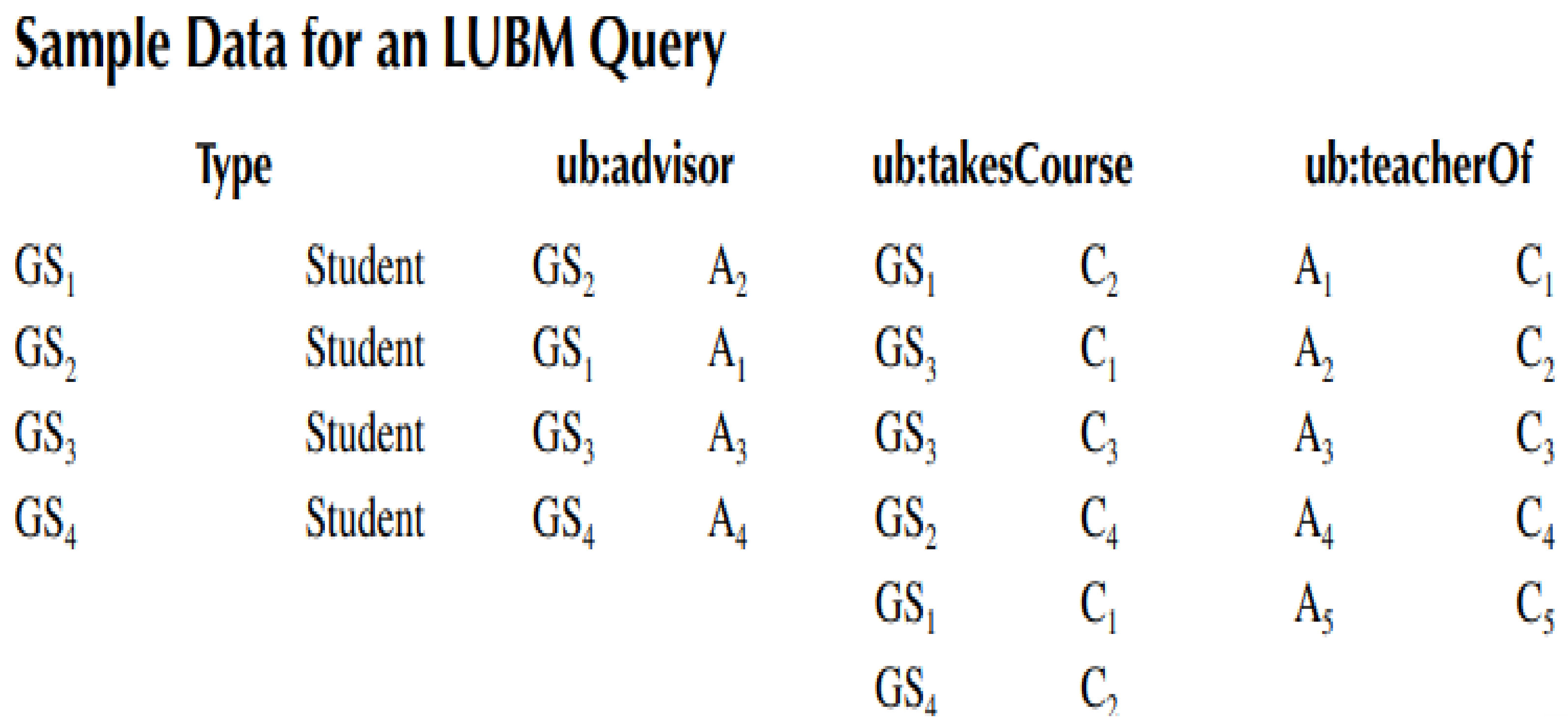
- Enforces access control policies during query execution using access control unit
- Executes queries using MapReduce framework with the help of input selector and plan generator, job executer, and Pellet OWL reasoner subcomponents
- The input selector and plan generator selects input files, decides how many jobs are needed, and passes the information to the job executer component
- The job executer component communicates with the access control unit to get relevant policies and runs jobs accordingly
- The Pellet OWL reasoner is used to answer queries that require inference
- Access control policies are stored in HDFS and loaded by the access control unit each time the framework is loaded

Data Generation and Storage

•The LUBM benchmark dataset is widely used for experiments in RDF data
•The RDF/XML format is not suitable for storing data in HDFS as files and for MapReduce jobs
•The data is converted to N-Triples format and partitioned by predicate to reduce the search space for SPARQL queries
•Predicates are named and stored in separate files, with each file containing complete RDF triples (subject, predicate, and object) in one line
•Sample data for three predicates (advisor, takesCourse, and teacherOf)



sample data for three predicates



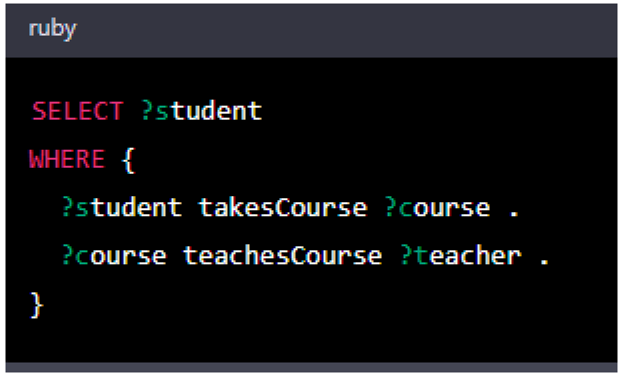
Sample Data for an LUBM Query

Type		ub:advisor		ub:takesCourse		ub:teacherOf	
GS_1	Student	GS_2	A_2	GS_1	C_2	A_1	C_1
GS_2	Student	GS_1	A_1	GS_3	C_1	A_2	C_2
GS_3	Student	GS_3	A_3	GS_3	C_3	A_3	C_3
GS_4	Student	GS_4	A_4	GS_2	C_4	A_4	C_4
				GS_1	C_1	A_5	C_5
				GS_4	C_2		

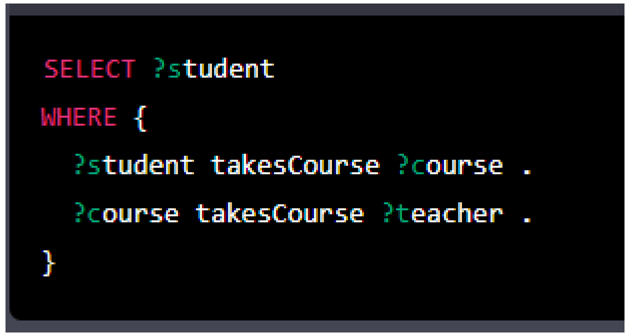
Policy Enforcement

- MapReduce framework enforces policies in two phases:
  - query-parsing phase
    - Some policies can be enforced by rewriting a SPARQL query during the query-parsing phase.
      - replacing predicate variables with the predicates to which a user has access.

Query Rewriting



```
SELECT ?student
WHERE {
  ?student takesCourse ?course .
  ?course teachesCourse ?teacher .
}
```



```
SELECT ?student
WHERE {
  ?student takesCourse ?course .
  ?course takesCourse ?teacher .
}
```

- query-answering phase
    - The remaining policies can be enforced during the query-answering phase in two ways
        - the first approach enforces policies during the MapReduce job execution
        - the second approach enforces policies after the job execution through a set of filter jobs

4. Predicate-level policies are enforced while selecting the input files by the input selector.
5. Query rewriting involves replacing predicate variables with the predicates to which a user has access.
6. An example is given to illustrate query rewriting.
7. After the query is rewritten, the framework can answer the query in two ways, which are detailed in the following sections.

Algorithm :Pseudo-Code for EEMAP 1:

- course:CS101
- subject:GS1
- instructor:Dr. Smith
- location:Building A, Room 101

- course:MA201
- subject:GS2
- instructor:Prof. Johnson
- location:Building B, Room 201

- course:EE301
- subject:GS3
- instructor:Dr. Lee
- location:Building C, Room 301

```
Algorithm 32.2: Pseudo-Code for EEMAP 1:
splits ← value.split ()
2: if Input file = sensitiveCourses then
3: output (splits [0], “S”)
 4: else if splits [0] = GS3 then
5: output (splits [1], “T”) 6: end if
```

- The program would first split each line into a key and a value, resulting in a list of key-value pairs:
- [("course", "CS101"), ("subject", "GS1"), ("instructor", "Dr. Smith"), ("location", "Building A, Room 101")]
- [("course", "MA201"), ("subject", "GS2"), ("instructor", "Prof. Johnson"), ("location", "Building B, Room 201")]
- [("course", "EE301"), ("subject", "GS3"), ("instructor", "Dr. Lee"), ("location", "Building C, Room 301")]

- the program would output the course and a flag ("S") to indicate that the course is confidential   EE301,S
- EE301,T

Algorithm :Pseudo-Code for EEREDUCE

```
1: count ← 0
2: iter ← values.iterator ()
3: while iter.hasNext () do
 4: count ++
5: t ← iter.next ()
 6: end while
7: if count = 1 AND t = "T" then
8: output (key)
9: end if
```

Postprocessing enforcement

- Postprocessing enforcement is a method of implementing access control policies in data processing jobs.
- The approach involves running a job without any access controls, and then filtering the output using additional jobs to enforce the policy.
- In the example given, the first job runs without any access controls and produces output that includes all courses, including confidential ones.
- The second job filters the output of the first job to remove any confidential courses based on the confidentialCourses file.
- The advantage of this approach is that it is simple to implement. However, it may take longer to answer the query because it involves running an extra job to enforce the policy

Secure Cloud Query Processing with Relational Data for Social Media

Unit 3

Introduction

1.Social media data contains sensitive personal information that should not be accessible to unauthorized parties.
2.Encryption can be used to protect data stored in a cloud-based database from unauthorized access.

overhead in terms of processing time and storage requirements
impact query performance and increase costs.

Query processing systems can perform computations on encrypted data without requiring it to be decrypted, ensuring that data remains secure.
4.Access control mechanisms can limit access to data based on user authorization levels.

4.complex to manage and configure

5.Efficient query processing techniques, such as parallel processing, can handle large volumes of data in real-time.
6.Caching techniques can improve query performance by storing frequently accessed data in memory.
7.A combination of encryption, access control, efficient query processing techniques, and caching can ensure the confidentiality, integrity, and availability of data while meeting performance requirements.

Introduction(Cont'd)

1. Secure large-scale data sharing in the cloud is an important issue for organizations.
2. Data sharing is necessary between different departments, partners, or customers while ensuring data privacy, confidentiality, and integrity.
3. Existing solutions face challenges and limitations.
4. The authors propose a new architecture that leverages cloud computing and XACML-based security mechanisms to address these challenges and limitations

XACML (eXtensible Access Control Markup Language)

- XML-based language for specifying and enforcing access control policies
- organizations to define and enforce fine-grained access control policies for their resource
- several benefits
  - Fine-grained access control- control access to specific resources based on a variety of factors
  - Flexibility-- organizations can customize it to meet their specific needs.
  - Interoperability- by many different vendors and systems, making it easier to integrate into existing environments.
  - Auditing and reporting- help organizations track access requests and identify potential security issues.

XACML is widely used in various industries and Applications

1. Healthcare: XACML can be used to enforce access control policies in healthcare systems, ensuring that sensitive patient data is only accessible to authorized personnel.
2. Finance: Financial institutions use XACML to enforce access control policies on sensitive financial information, ensuring that only authorized personnel can access the data.
3. Government: XACML is used in government systems to enforce access control policies for classified or sensitive data.
4. Cloud computing: XACML can be used to manage access control policies in cloud computing environments, ensuring that data stored in the cloud is protected and accessible only to authorized users.
5. Identity and access management (IAM): XACML is often used in IAM solutions to enforce access control policies for user authentication, authorization, and access management.

Key components of xacml-based security mechanisms

- Policy Decision Point (PDP)
  - evaluating access requests
  - access control decisions based on the policies defined in the system.
- Policy Enforcement Point (PEP)
  - for intercepting access requests
  - forwarding them to the PDP for evaluation
- Policy Information Point (PIP)
  - provides additional information to the PDP
    - user attributes, resource attributes, or other contextual information.
- Policy Administration Point (PAP)
  - managing the policies that are used by the system
    - to create, modify, and delete policies as necessary.

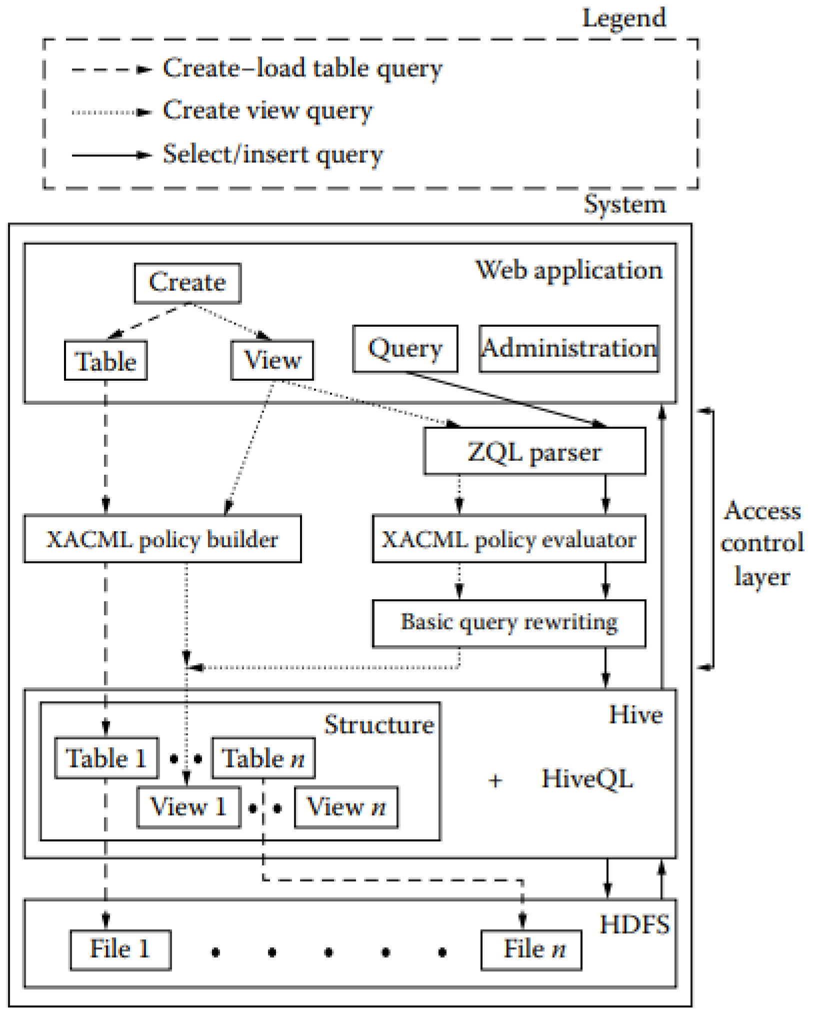
Secure Large-Scale Data Sharing in the Cloud

- Cloud computing technologies provide a scalable and distributed way to store and process large amounts of data.
- HDFS (Hadoop Distributed File System) and Hive are commonly used technologies in cloud computing for data storage and processing.
- XACML (eXtensible Access Control Markup Language) is a standard policy-based access control language that enables fine-grained authorizatio
  and attribute-based access control.
- Combining HDFS, Hive, and XACML can provide a secure and efficient way of sharing data in the cloud.
- The proposed architecture provides a mechanism to load and query shared data securely stored in HDFS using Hive.
- The additional layer of security using XACML policies ensures data privacy, confidentiality, and integrity.
- Fine-grained authorization and attribute-based access control provide better control over who can access data and what actions they can perform
  on it.
- This solution can benefit organizations that need to share data between different departments, partners, or customers while ensuring data security

SYSTEM ARCHITECTURE

1.The web application layer is the only interface provided by the system for user access to the cloud infrastructure.
2.Different functions are provided based on the user's permissions.
3.A log-in page is provided for user authentication, and passwords are stored using a salted hash technique.
4.The system supports three types of users:

those who can only query existing tables/views,
those who can create tables/views and define XACML policies,
and a special admin user who can assign new users to either of the above categories.



A Salted Hash Technique

1. User creates a password "MyPassword123" for their account on a website.
2. Website's salted hash function generates a random salt value, "Gc$7tYpL", and combines it with the password to create a salted password: MyPassword123 + Gc$7tYpL = MyPassword123Gc$7tYpL
3. Salted password is then hashed using a cryptographic hash function, such as SHA-256, to create a hash value: SHA-256(MyPassword123Gc$7tYpL) = 879c2d72ab62f7581947e9c9c72e7d0718d19d1cf95457af5da6b7bb6f20283c
4. Salt value "Gc$7tYpL" and hash value "879c2d72ab62f7581947e9c9c72e7d0718d19d1cf95457af5da6b7bb6f20283c" are stored in the website's database.
5. When the user attempts to log in, they enter their password "MyPassword123" again.
6. Website retrieves the salt value associated with the user's account from the database, combines it with the password, and hashes the salted password to compare it to the stored hash value.
7. Resulting hash value matches the stored hash value, and the user is granted access to their account.
8. This process makes it much more difficult for an attacker to guess the user's password, as they would need to know the random salt value used in the hashing process to generate the correct hash value

ZQL parser layer parses

- The ZQL parser layer is a component of the Zenoss monitoring software platform responsible for analyzing ZQL queries.
- ZQL is a query language used to retrieve data from data sources in Zenoss.
- The parser layer checks the syntax and structure of a ZQL query for errors.
- The parser layer identifies the data source and fields to be queried, and applies filters or conditions specified in the query.
- Once the query is parsed, it is transformed into an internal representation used by the Zenoss platform to retrieve the requested data.
- The next layer in the Zenoss platform executes the internal representation of the parsed query to retrieve the data.
- Finally, the requested data is returned to the user.
- The ZQL parser layer helps to ensure the accuracy and efficiency of data retrieval in Zenoss.

ZQL parser layer parses (CONT’D)

1. The ZQL parser layer parses user queries and sends them to the XACML policy evaluator or returns an error message if the query is invalid.
2. The ZQL parser is an SQL parser written in Java that constructs different Java vectors for different parts of the query.
3. In the system, the vector of attribute names in the SELECT clause is returned to the web application layer, and the vector of table/view names in the FROM clause is passed to the XACML policy evaluator to check the user's permissions.
4. The ZQL parser currently supports SQL DELETE, INSERT, SELECT, and UPDATE statements.
5. Future work involves adding support for other keywords such as CREATE, DROP, etc.

XACML Policy Layer

define access control policies on resources

used to determine whether access is allowed for a particular resource based on the policy

•Users can create tables/views and define XACML policies on them.

•Users can upload their own predefined XACML policy or have the system build a policy for them.

•If a user chooses to have the system build a policy, they must specify the kinds of queries allowed on the table/view.

•Sun's XACML implementation is used to build policies for tables/views with user-specified group

Policy: A policy is a set of rules that define how access to resources should be controlled. Policies can be defined at different levels, such as the system level, application level, or user level
Rule: A rule is a statement that specifies a condition and an action. It defines under what circumstances the action should be taken.
Target: A target specifies the resources to which the policy applies
Condition: A condition specifies additional criteria that must be met for a rule to apply.

XACML Policy Evaluator

- The XACML policy evaluator is a tool used to enforce access control policies for a system.
- The XACML policy evaluator is used in Hive to check whether the current user has access to all tables and views specified in a user query or view creation operation.
- Sun's XACML implementation is used to implement the policy evaluator in Hive.
- If the user has the required permissions for all the specified tables and views, the query or view creation operation is processed further. Otherwise, an error message is returned to the user.
- In Hive, the only way to create a view is by specifying a SELECT query on the existing tables/views.
- The XACML policy evaluator is used during view creation in Hive to ensure that the user has the required permissions to create the view

Basic Query-Rewriting Layer

- to add another level of abstraction between users and HiveQL.
- users to input SQL queries that are rewritten according to HiveQL's syntax.
- Currently, two rewriting rules are provided to transform the SQL query: a. Multiple tables in the FROM clause can be transformed into a sequence of JOIN statements. b. The traditional INSERT INTO <tablename> SELECT statement is rewritten into HiveQL's
  - INSERT OVERWRITE TABLE <tablename> SELECT statement.

1. In the future, a more complete query-rewriting engine will be added with more complicated rules.

Suppose a user wants to query data from two tables "orders" and "customers" using a traditional SQL query

- SELECT * FROM orders, customers WHERE orders.customer_id = customers.id;
- HiveQL syntax does not allow multiple tables in the FROM clause of a query
  - it expects such queries to be given as a sequence of JOIN statements
  - SELECT * FROM orders JOIN customers ON orders.customer_id = customers.id;
  - Note:
    - The user can now execute this query on HiveQL without needing to worry about the syntax differences

HiveQL uses a modified version of SQL's INSERT-SELECT

- INSERT OVERWRITE TABLE <tablename> SELECT
  - replaces the existing data in a table with the results of the SELECT statement.

  - SQL
    - INSERT INTO a SELECT * FROM b;
  - HiveQL-compatible syntax using "INSERT OVERWRITE TABLE
    - INSERT OVERWRITE TABLE a SELECT * FROM b;
- Suppose a user wants to insert the contents of a table "c" into another table "d" while specifying a subset of columns to be inserted
  - INSERT INTO d (id, name) SELECT id, name FROM c;
  - INSERT OVERWRITE TABLE d SELECT id, name FROM c;

Hive Layer

- Hive is a data warehouse infrastructure built on top of Hadoop.
- Hive structures data in the HDFS and enables querying of this data using HiveQL.
- Each table in Hive is stored as a file in the HDFS.
- Views in Hive are only a logical concept created with a SELECT query and are not stored as files in the HDFS.
- Hive is used in our framework to structure and share data between collaborating organizations.
- HiveQL enables users to query the data using a familiar SQL-like syntax

HDFS layer

- The HDFS is a distributed file system designed to run on basic hardware.
- The HDFS layer in our framework stores the data files corresponding to tables created in Hive.
- Our security assumption is that these files can only be accessed through our system and not through Hadoop's web or command line interfaces.