Cloud-Centric Assured Information Sharing

Bhavani Thuraisingham, Vaibhav Khadilkar, Jyothsna Rachapalli,
Tyrone Cadenhead, Murat Kantarcioglu, Kevin Hamlen,
Latifur Khan, and Farhan Husain

The University of Texas at Dallas, Richardson, Texas, USA
{bhavani.thuraisingham,vvk072000,jxr061100,tyrone.cadenhead,
muratk,hamlen,latifur.khan,farhan}@utdallas.edu

Abstract. In this paper we describe the design and implementation of cloud-based assured information sharing systems. In particular, we will describe our current implementation of a centralized cloud-based assured information sharing system and the design of a decentralized hybrid cloud-based assured information sharing system of the future. Our goal is for coalition organizations to share information stored in multiple clouds and enforce appropriate policies.

1 Introduction

The advent of *cloud computing* and the continuing movement toward *software as a service* (SaaS) paradigms has posed an increasing need for *assured information sharing* (AIS) as a service in the cloud. The urgency of this need has been voiced as recently as April 2011 by NSA CIO Lonny Anderson in describing the agency's focus on a "cloud-centric" approach to information sharing with other agencies [1]. Likewise, the DoD has been embracing cloud computing paradigms to more efficiently, economically, flexibly, and scalably meet its vision of "delivering the power of information to ensure mission success through an agile enterprise with freedom of maneuverability across the information environment" [2-5]. Both agencies therefore have a tremendous need for effective AIS technologies and tools for cloud environments.

Although a number of AIS tools have been developed over the past five years for policy-based information sharing [5-8], to our knowledge none of these tools operate in the cloud and hence do not provide the scalability needed to support large numbers of users utilizing massive amounts of data. Recent prototype systems for supporting cloud-based AIS have applied cloud-centric engines that query large amounts of data in relational databases via non-cloud policy engines that enforce policies expressed in XACML [9-10]. While this is a significant improvement over prior efforts (and has given us insights into implementing cloud-based solutions), it nevertheless has at least three significant limitations. First, XACML-based policy specifications are not expressive enough to support many of the complex policies needed for AIS missions like those of the NSA and DoD. Second, to meet the scalability and efficiency requirements of mission-critical tasks, the policy engine needs to operate in the cloud rather than externally. Third, secure query processing based on relational technology has limitations in representing and processing unstructured data needed for command and control applications.

M. Chau et al. (Eds.): PAISI 2012, LNCS 7299, pp. 1–26, 2012.

To share the large amounts of data securely and efficiently, there clearly needs to be a seamless integration of the policy and data managers in the cloud. Therefore, in order to satisfy the cloud-centric AIS needs of the DoD and NSA, we need (i) a cloud-resident policy manager that enforces information sharing policies expressed in a semantically rich language, and (ii) a cloud-resident data manager that securely stores and retrieves data and seamlessly integrates with the policy manager. To our knowledge, no such system currently exists. Therefore, our project is designing and developing such cloud-based assured information sharing system is proceeding in two phases.

During phase 1, we are developing a proof of concept prototype of a Cloud-centric Assured Information Sharing System (CAISS) that is utilizes the technology components we have designed in-house as well as open source tools. CAISS consists of two components: a cloud-centric policy manager that enforces policies specified in RDF (resource description framework), and a cloud-centric data manager that will store and manage data also specified in RDF. This RDF data manager is essentially a query engine for SPARQL (SPARQL Protocol and RDF Query Language), a language widely used by the semantic web community to query RDF data. RDF is a semantic web language that is considerably more expressive than XACML for specifying and reasoning about policies. Furthermore, our policy manager and data manager will have seamless integration since they both manage RDF data. We have chosen this RDF-based approach for cloud-centric AIS during Phase 1 because it satisfies the two necessary conditions stated in the previous paragraph, and we have already developed an RDF-based non-cloud centric policy manager [11] and an RDF-based cloud-centric data manager for AFOSR [12]. Having parts of the two critical components needed to build a useful cloud-centric AIS system puts us in an excellent position to build a useful proof of concept demonstration system CAISS. Specifically, we are enhancing our RDF-based policy engine to operate on a cloud, extend our cloud-centric RDF data manager to integrate with the policy manager, and build an integrated framework for CAISS.

While our initial CAISS design and implementation will be the first system supporting cloud-centric AIS, it will operate only on a single trusted cloud and will therefore not support information sharing across multiple clouds. Furthermore, while CAISS's RDF-based, formal semantics approach to policy specification will be significantly more expressive than XACML-based approaches, it will not support an enhanced machine interpretability of content since RDF does not provide a sufficiently rich vocabulary (e.g., support for classes and properties). Phase 2 will therefore develop a fully functional and robust AIS system called CAISS++ that addresses these deficiencies. The preliminary design for CAISS++ is completed and will be discussed later in this paper. CAISS is an important stepping-stone towards CAISS++ because CAISS can be used as a baseline framework against which CAISS++ can be compared along several performance dimensions, such as storage model efficiency and OWL-based policy expressiveness. Furthermore, since CAISS and CAISS++ share the same core components (policy engine and query processor), the lessons learned from the implementation and integration of these components in CAISS will be invaluable during the development of CAISS++. Finally, the evaluation and testing of CAISS will provide us with important insights into the shortcomings of CAISS, which can then be systematically addressed in the implementation of CAISS++.

We will also conduct a formal analysis of policy specifications and the software-level protection mechanisms that enforce them to provide exceptionally high-assurance security guarantees for the resulting system. We envisage CAISS++ to be used in highly mission-critical applications. Therefore, it becomes imperative to provide guarantees that the policies are enforced in a provably correct manner. We have extensive expertise in formal policy analysis [13-14] and their enforcement via machine-certified, in-line reference monitors [15-17]. Such analyses will be leveraged to model and certify security properties enforced by core software components in the trusted computing base of CAISS++.

CAISS++ will be a breakthrough technology for information sharing due to the fact that it uses a novel combination of cloud-centric policy specification and enforcement along with a cloud-centric data storage and efficient query evaluation. CAISS++ will make use of ontologies, a sublanguage of the Web Ontology Language (OWL), to build policies. A mixture of such ontologies with a Semantic Web based rule language (e.g. SWRL) facilitates distributed reasoning on the policies to enforce security. Additionally, CAISS++ will include a RDF processing engine that provides cost-based optimization for evaluating SPARQL queries based on information sharing policies.

We will discuss the design and implementation of CAISS in Section 2.1 and the design of CAISS++ in Section 2.2. Formal policy analysis and the implementation approach for CAISS++ will be provided in Sections 2.3 and 2.4, respectively. Related efforts are discussed in Section 3. The paper is concluded in Section 4.

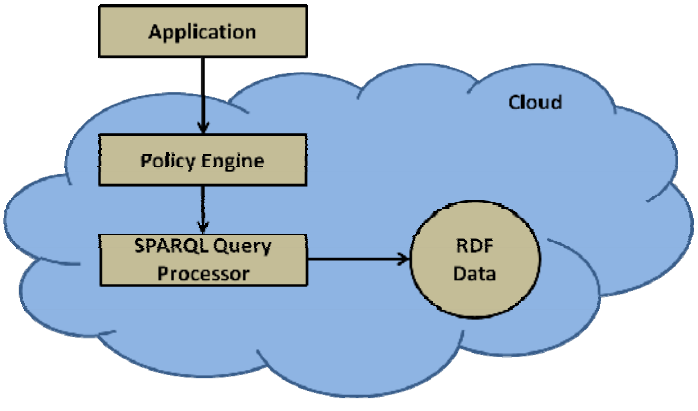

Fig. 1. CAISS Prototype Overview

2 System Design and Implementation

2.1 Proof of Concept Prototype of CAISS

We are enhancing our tools developed for AFOSR on (i) secure cloud query processing with semantic web data, and (ii) semantic web-based policy engine, to develop CAISS. Details of our tools are given in Section 4 (under related work). 1 In this section we will discuss the enhancements to be made to our tools to develop CAISS.

First, our RDF-based policy engine enforces access control, redaction and inference control policies on data represented as RDF graphs. Second, our cloud SPARQL query engine for RDF data uses the Hadoop/Mapreduce framework. Note that Hadoop is the Apache distributed file system and MapReduce sits on top of Hadoop and carries out job scheduling. As in the case of our cloud-based relational query processor prototype [9], our SPARQL query engine also handles policies are specified in XACML and the policy engine implements the XACML protocol. The use of XACML as a policy language requires extensive knowledge about the general concepts used in the design of XACML. Thus, policy authoring in XACML requires a steep learning curve, and is therefore a task that is left to an experienced administrator. A second disadvantage of using XACML is related with performance. Current implementations of XACML require an access request to be evaluated against every policy in the system until a policy applies to the incoming request. This strategy is sufficient for systems with a relatively few users and policies. However, for systems with a large number of users and a substantial number of access requests, the aforementioned strategy becomes a performance bottleneck. Finally, XACML is not sufficiently expressive to capture the semantics of information sharing policies. Prior research has shown that semantic web-based policies are far more expressive. This is because semantic web technologies are based on description logic and have the power to represent knowledge as well as reason about knowledge. Therefore our first step is to replace the XACML-based policy engine with a semantic web-based policy engine. Since we already have our RDF-based policy engine, for the Phase 1 prototype we will enhance this engine and integrate it with our SPARQL query processor. Since our policy engine is based on RDF and our query processor also manages large RDF graphs there will be no impedance mismatch between the data and the policies.

Enhanced Policy Engine. Our current policy engine has a limitation in that it does not operate in a cloud. Therefore, we will port our RDF policy engine to the cloud environment and integrate it with the SPARQL query engine for federated query processing in the cloud. Our policy engine will benefit from the scalability and the distributed platform offered by Hadoop's MapReduce framework to answer SPARQL queries over large distributed RDF triple stores (billions of RDF triples). The reasons for using RDF as our data model are as follows: (1) RDF allows us to achieve data interoperability between the seemingly disparate sources of information that are catalogued by each agency/organization separately. (2) The use of RDF allows participating agencies to create data-centric applications that make use of the integrated data that is now available to them. (3) Since RDF does not require the use of an explicit schema for data generation, it can be easily adapted to ever-changing user requirements. The policy engine's flexibility is based on its accepting high-level policies and executing them as query rules over a directed RDF graph representation of the data. While our prior work focuses on provenance data and access control policies, our CAISS prototype will be flexible enough to handle data represented in RDF and will include information sharing policies. The strength of our policy engine is that it can handle any type of policy that could be represented using RDF and horn logic rules.

The second limitation of our policy engine is that it currently addresses certain types of policies such as confidentiality, privacy and redaction policies. We need to incorporate information sharing policies into our policy engine. We have however conducted simulation studies for incentive-based AIS as well as AIS prototypes in the cloud.

We have defined a number of information sharing policies such as "US gives information to UK provided UK does not share it with India". We propose to specify such policies in RDF and incorporate them to be processed by our enhanced policy engine.

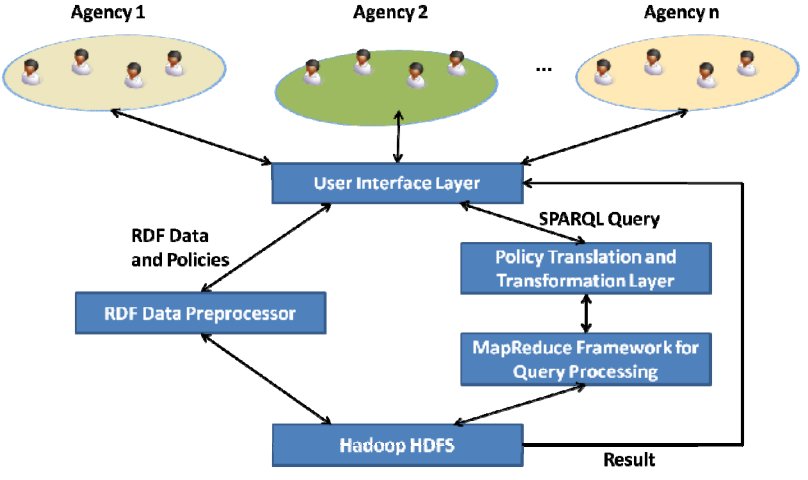

Fig. 2. Operation of CAISS

Enhanced SPARQL Query Processor. While we have a tool that will execute SPARQL queries over large RDF graphs on Hadoop, there is still the need for supporting path queries (that is, SPARQL queries that provide answers to a request for paths in a RDF graph). A RDF triple can be viewed as an arc from the Subject to Object with the Predicate used to label the arc. The answers to the SPARQL query are based on reachability (that is, the paths between a source node and a target node). The concatenation of the labels on the arcs along a path can be thought of as a word belonging to the answer set of the path query. Each term of a word is contributed by some predicate label of a triple in the RDF graph. We propose an algorithm to determine the candidate triples as an answer set in a distributed RDF graph. First, the RDF document is converted to a N-triple file that is split based on predicate labels. A term in a word could correspond to some predicate file. Second, we form the word by tracing an appropriate path in the distributed RDF graph. We use MapReduce jobs to build the word and to get the candidate RDF triples as an order set. Finally we return all of the set of ordered RDF triples as the answers to the corresponding SPARQL query.

Integration Framework. Figure 1 provides an overview of the CAISS architecture. The integration of the cloud-centric RDF policy engine with the enhanced SPARQL query processor must address the following. First, we need to make sure that RDF-based policies can be stored in the existing storage schema used by the query processor. Second, we need to ensure that the enhanced query processor is able to efficiently evaluate policies (i.e., path queries) over the underlying RDF storage. Finally, we need to conduct a performance evaluation of CAISS to verify that it meets the

performance requirements of various participating agencies. Figure 2 illustrates the concept of operation of CAISS. Here, 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. This proof of concept system will drive the detailed design and implementation of CAISS++.

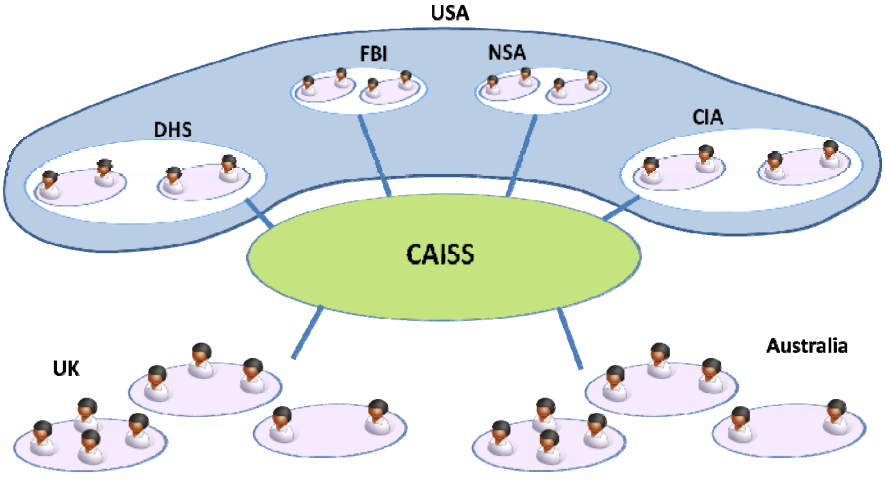

Fig. 3. CAISS++ Scenario

There are several benefits in developing a proof of concept prototype such as CAISS before we embark on CAISS++. First CAISS itself is useful to share data within a single cloud. Second, we will have a baseline system that we can compare against with respect to efficiency and ease-of-use when we implement CAISS++. Third, this will give us valuable lessons with respect to the integration of the different pieces required for AIS in the cloud. Finally, by running different scenarios on CAISS, we can identify potential performance bottlenecks that need to be addressed in CAISS++.

2.2 Design of CAISS++

We have examined alternatives and carried out a preliminary design of CAISS++. Based on the lessons learned from the CAISS prototype and the preliminary design of CAISS++, we will carry out a detailed design of CAISS++ and subsequently implement an operational prototype of CAISS++ during Phase 2. In this section we will first discuss the limitations of CAISS and then discuss the design alternatives for CAISS++.

Limitations of CAISS. 1. *Policy Engine:* CAISS uses an RDF-based policy engine which has limited expressivity. The purpose of RDF is to provide a structure (or framework) for describing resources. OWL is built on top of RDF and it is designed for use by applications that need to process the content of information instead of just

presenting information to human users. OWL facilitates greater machine interpretability of content than that supported by RDF by providing additional vocabulary for describing properties and classes along with a formal semantics. OWL has three increasingly-expressive sublanguages: OWL Lite, OWL DL and OWL Full and one has the freedom to choose a suitable sublanguage based on application requirements. In CAISS++, we plan 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 (note that CAISS++ will reuse an organization's existing domain ontology or facilitate the creation of a new domain ontology if it does not exist. Additionally, engineer the upper ontology that will be used by the centralized component of CAISS++). Additionally, CAISS++ will make use of a distributed reasoning algorithm which will leverage ontologies to enforce security policies.

2. Hadoop Storage Architecture: CAISS uses a static storage model wherein a user provides the system with RDF data only once during the initialization step. Thereafter, a user is not allowed to update the existing data. On the other hand, CAISS++ attempts to provide a flexible storage model to users. In CAISS++, a user is allowed to append new data to the existing RDF data stored in HDFS. Note that, only allowing a user to append new data rather than deleting/modifying existing data comes from the append-only restriction for files that is enforced by HDFS.

3. *SPARQL Query Processor*: CAISS only supports simple SPARQL queries that make use of basic graph patterns (BGP). 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.

Design of CAISS++. CAISS++ overcomes the limitations of CAISS. The detailed design of CAISS++ and its implementation will be carried out during Phase 2. The lessons learned from CAISS will also drive the detailed design of CAISS++. We assume that the data is encrypted with appropriate DoD encryption technologies and therefore will not conduct research on encryption in this project. The concept of operation for CAISS++ is shown in interaction with several participating agencies in Figure 3 where multiple organizations share data in a single cloud.

The design of CAISS++ is based on a novel combination of an OWL-based policy engine with a RDF processing engine. Therefore, this design is composed of several tasks each of which is solved separately after which all tasks are integrated into a single framework. (1) **OWL-based policy engine:** The policy engine uses a set of agency-specific domain ontologies as well as an upper ontology to construct policies for the task of AIS. The task of enforcing policies may require the use of a distributed reasoner, therefore, we will evaluate existing distributed reasoners. (2) **RDF processing engine:** The processing engine requires the construction of sophisticated storage architectures as well as an efficient query processor. (3) **Integration Framework:** The final task is to combine the policy engine with the processing engine into an integrated framework. The initial design of CAISS++ will be based on a trade-off between simplicity of design vs. its scalability and efficiency. The first design alternative is known as Centralized CAISS++ and it chooses simplicity as the trade-off whereas the second design alternative known as Decentralized CAISS++ chooses

scalability and efficiency as the trade-off. Finally, we also provide a Hybrid CAISS++ architecture that tries to combine the benefits of both, Centralized and Decentralized CAISS++. Since CAISS++ follows a requirements-driven design, the division of tasks that we outlined above to achieve AIS are present in each of the approaches that we present next.

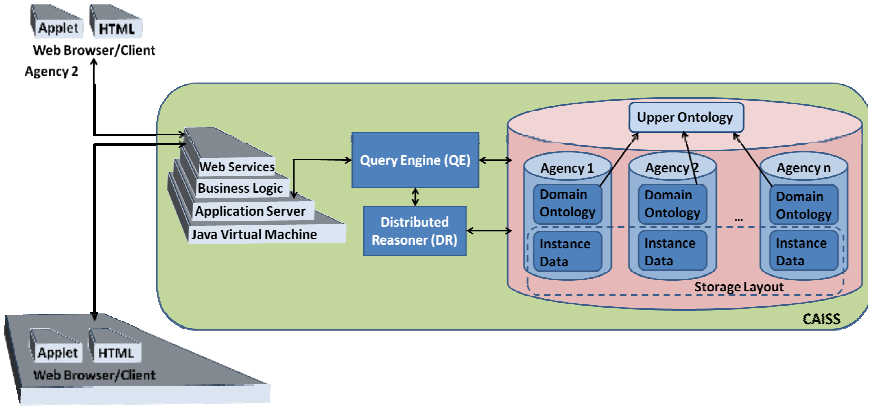

Fig. 4. Centralized CAISS++

Centralized CAISS++. Figure 4 illustrates two agencies interacting hrough Centralized CAISS++. Centralized CAISS++ consists of a shared cloud storage to store the shared data. All the participating agencies store their respective knowledge bases consisting of domain ontology with corresponding instance data. Centralized CAISS++ also consists of an upper ontology, a query engine (QE) and a distributed reasoner (DR). The upper ontology is used to capture the domain knowledge that is common across the domains of participating agencies whereas, domain ontology captures the knowledge specific to a given agency or a domain. Note that the domain ontology for a given agency will be protected from the domain ontologies of other participating agencies. Policies can either be captured in the upper ontology or in any of the domain ontologies depending on their scope of applicability. Note that the domain ontology for a given agency will be protected from domain ontologies of other participating agencies.The design of an upper ontology as well as domain ontologies that capture the requirements of the participating agencies is a significant research area and is the focus of the ontology engineering problem. Ontologies will be created using suitable dialects of OWL which are based on Description Logics. Description Logics are usually decidable fragments of First Order Logic and will be the basis for providing sound formal semantics. Having represented knowledge in terms of ontologies, reasoning will be done using existing optimized reasoning algorithms. Query answering will leverage reasoning algorithms to formulate and answer intelligent queries. The encoding of policies in OWL will ensure that they are enforced in a provably correct manner. In Section 3.1 we present an on-going research project at UTD that focuses on providing a general framework for enforcing policies in a provably correct manner using the same underlying technologies. This work can be leveraged towards modeling and enforcement of security policies in CAISS++.

The instance data can choose between several available data storage formats (discussed later on). The QE receives queries from the participating agencies, parses the query and determines whether or not the computation requires the use of a DR. If the query is simple and does not require the use of a reasoner, the query engine executes the query directly over the shared knowledge base. Once the query result has been computed the result is returned to the querying agency. If however, the query is complex and requires inferences over the given data the query engine uses the distributed reasoner to compute the inferences and then returns the result to the querying agency. A distributed DL reasoner differs from a traditional DL reasoner in its ability to perform reasoning over cloud data storage using the MapReduce framework. During the preliminary design of CAISS++ in Phase 1, we will conduct a thorough investigation of the available distributed reasoners using existing benchmarks such as LUBM [17]. The goal of this investigation is to determine if we can use one of the existing reasoners or whether we need to build our own distributed reasoner. In Figure 4, an agency is illustrated as a stack consisting of a web browser, an applet and HTML. An agency uses the web browser to send the queries to CAISS++ which are handled by the query processor.

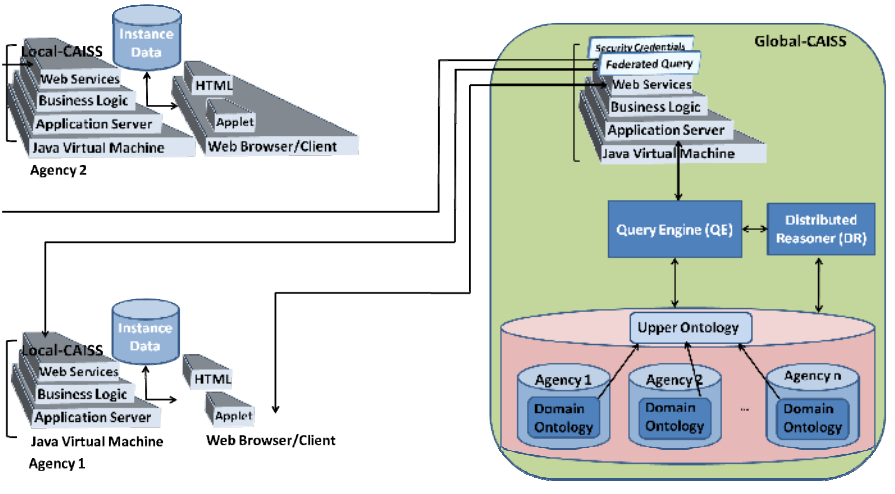

Fig. 5. Decentralized CAISS++

The main differences between Centralized CAISS++ and CAISS (described in Section 2.1) are as follows: (1) CAISS will use RDF to encode security policies whereas Centralized CAISS++ will use a suitable sublanguage of OWL which is more expressive than RDF and can therefore capture the security policies better. (2) The SPARQL query processor in CAISS will support a limited subset of SPARQL expressivity i.e. it will provide support only for Basic Graph Patterns (BGP), whereas the SPARQL query processor in Centralized CAISS++ will be designed to support maximum expressivity of SPARQL. (3) The Hadoop storage architecture used in CAISS only supports data insertion during an initialization step. However, when data needs to be updated, the entire RDF graph is deleted and a new dataset is inserted in its place. On the other

hand, Centralized CAISS++, in addition to supporting the previous feature, also opens up Hadoop HDFS's append-only feature to users. This feature allows users to append new information to the data that they have previously uploaded to the system.

Decentralized CAISS++. Figure 5 illustrates two agencies in interaction with Decentralized CAISS++. Decentralized CAISS++ consists of two parts namely Global CAISS++ and Local CAISS++. Global CAISS++ consists of a shared cloud storage which is used by the participating agencies to store only their respective domain ontologies and not the instance data unlike centralized CAISS++. Note that domain ontologies for various organizations will be sensitive, therefore, CAISS++ will make use of its own domain ontology to protect a participating agency from accessing other domain ontologies. When a user from an agency queries the CAISS++ data store, Global CAISS++ processes the query in two steps. In the first step, it performs a check to verify whether the user is authorized to perform the action specified in the query. If the result of step 1 verifies the user as an authorized user, then it proceeds to step 2 of query processing. In the second step, Global CAISS++ federates the actual query to the participating agencies. The query is then processed by the Local CAISS++ of a participating agency. The result of computation is then returned to the Global CAISS++ which aggregates the final result and returns it to the user. The step 2 of query processing may involve query splitting if the data required to answer a query spans multiple domains. In this case the results of sub-queries from several agencies (their Local CAISS++) will need to be combined for further query processing. Once the results are merged and the final result is computed the result is returned to the user of the querying agency. The figure illustrates agencies with a set of two stacks, one of which corresponds to the Local CAISS++ and the other consisting of a web browser, an applet and HTML, which is used by an agency to query Global CAISS++. Table 1 shows the pros and cons of the Centralized CAISS++ approach while Table 2 shows the pros and cons of the Decentralized CAISS++ approach.

Hybrid CAISS++. Figure 6 illustrates an overview of Hybrid CAISS++ which leverages the benefits of Centralized CAISS++ as well as Decentralized CAISS++. Hybrid CAISS++ architecture is illustrated in Figure 7. It is a flexible design alternative as the users of the participating agencies have the freedom to choose between Centralized CAISS++ or Decentralized CAISS++. Hybrid CAISS++ is made up of Global CAISS++ and a set of Local CAISS++'s located at each of the participating agencies. Global CAISS++ consists of a shared cloud storage which is used by the participating agencies to store the data they would like to share with other agencies.

Table 1. The pros and cons of Centralized CAISS++

PROS	CONS
Simple approach	Difficult to update data. Expensive approach as data needs to be migrated to central storage on each update or a set of updates.
Ease of implementation	Leads to data duplication
Easier to query	If data is available in different formats it needs to be homogenized by translating it to RDF

Table 2. The pros and cons of Decentralized CAISS++

Advantages	Disadvantages
No duplication of data	Complex query processing.
Scalable and Flexible	Difficult to implement
Efficient	May require query rewriting and query splitting

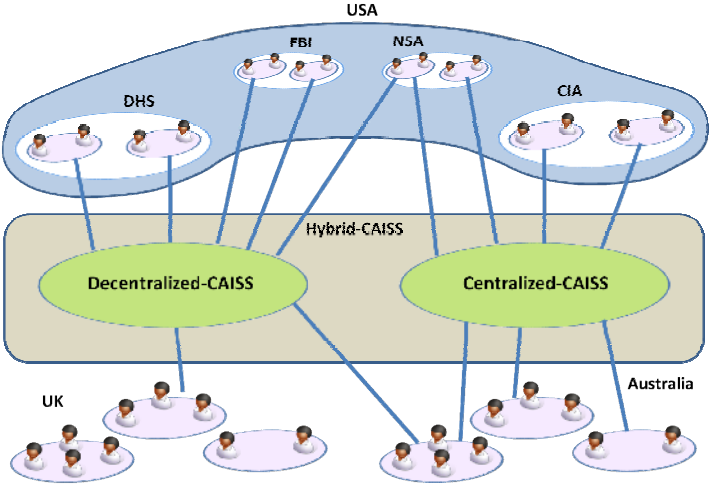

Fig. 6. Hybrid CAISS++ Overview

A Local CAISS++ of an agency is used to receive and process a federated query on the instance data located at the agency. A participating group is a group comprising of users from several agencies who want to share information with each other. The members of a group arrive on a mutual agreement on whether they opt for centralized or decentralized approach. Additional users can join a group at a later point in time if the need arises. Hybrid CAISS++ will be designed to simultaneously support a set of participating groups. Additionally, a user can belong to several participating groups at the same time. We describe few use-case scenarios which illustrate the utility of Hybrid CAISS+.

1) This case corresponds to the scenario where a set of users who want to securely share information with each other opt for a centralized approach. Suppose users from Agency 1 want to share information with users of Agency 2 and vice versa, then both the agencies store their knowledge bases comprising of domain ontology and instance data on the shared cloud storage located at Global CAISS++. The centralized CAISS++ approach works by having the participating agencies arrive at mutual trust on using the central cloud storage. Subsequently, information sharing proceeds as in Centralized CAISS++.

2) This case corresponds to the scenario where a set of users opt for a decentralized approach. For example, Agencies 3, 4 and 5 wish to share information with each other

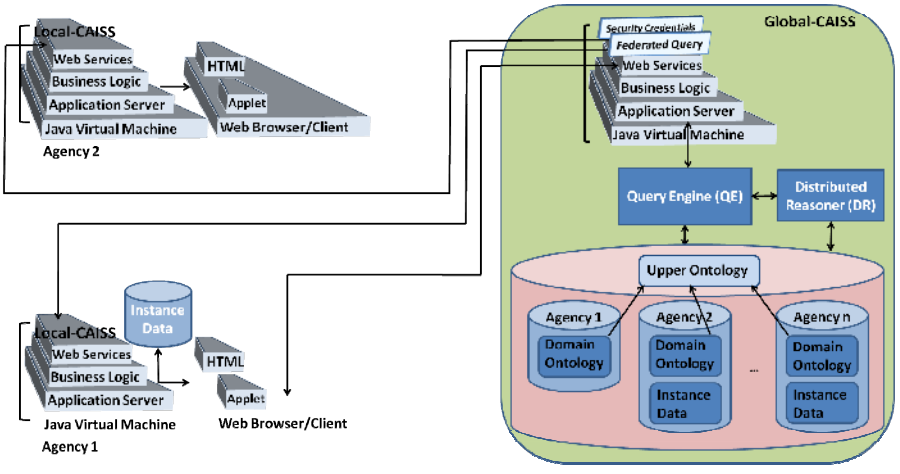

Fig. 7. Hybrid CAISS++ Architecture

and mutually opt for the decentralized approach. All the three agencies store their respective domain ontologies at the central cloud storage and this information is only accessible to members of this group. The subsequent information sharing process proceeds in the manner described earlier for the Decentralized CAISS++ approach.

3) This case corresponds to the scenario where a user of an agency belongs to multiple participating groups, some of which opt for the centralized approach and others for the decentralized approach. Since the user is a part of a group using the centralized approach to sharing, he/she needs to make his/her data available to the group by shipping his/her data to the central cloud storage. Additionally, since the user is also a part of a group using the decentralized approach for sharing he/she needs to respond to the federated query with the help of the Local CAISS++ located at his/her agency.

Table 3 shows the trade-offs between the different approaches and this will enable users to choose a suitable approach of AIS based on their application requirements. Next we describe details of the cloud storage mechanism that makes use of Hadoop to store the knowledge bases from various agencies and then discuss the details of distributed SPARQL query processing over the cloud storage.

Table 3. A comparison of the three approaches based on functionality

Functionality	**Centralized CAISS++**	**Decentralized CAISS++**	**Hybrid CAISS++**
No Data Duplication	X	√	Maybe
Flexibility	X	X	√
Scalablility	X	√	√
Efficiency	√	√	√
Simplicity - No query rewriting	√	X	X
Trusted Centralized Cloud Data Storage	√	X	X

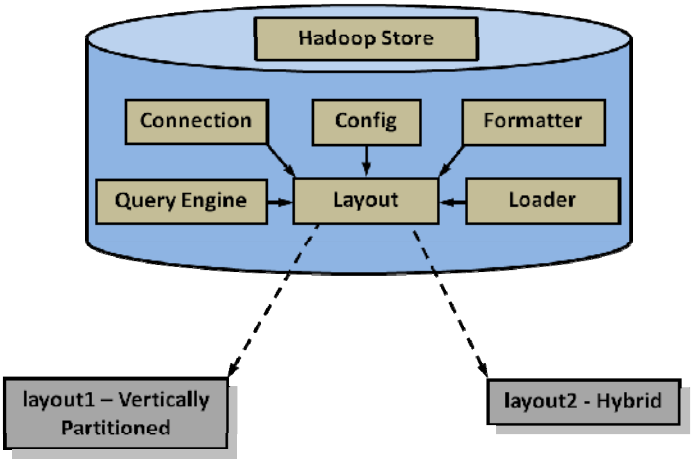

Fig. 8. Hadoop Storage Architecture used by CAISS++

Hadoop Storage Architecture. In Figure 8, we present an architectural overview of our Hadoop-based RDF storage and retrieval framework. We use the concept of a "Store" to provide data loading and querying capabilities on RDF graphs that are stored in the underlying HDFS. A store represents a single RDF dataset and can therefore contain several RDF graphs, each with its own separate layout. All operations on a RDF graph are then implicitly converted into operations on the underlying layout including the following:

- Layout Formatter: This block performs the function of formatting a layout, which is the process of deleting all triples in a RDF graph while preserving the directory structure used to store that graph.
- Loader: This block performs loading of triples into a layout.
- Query Engine: This block allows a user to query a layout using a SPARQL query. Since our framework operates on the underlying HDFS, the querying mechanism on a layout involves translating a SPARQL query into a possible pipeline of MapReduce jobs and then executing this pipeline on a layout.
- Connection: This block maintains the necessary connections and configurations with the underlying HDFS.
- Config: This block maintains configuration information such as graph names for each of the RDF graphs that make up a store.

Since RDF data will be stored under different HDFS folders in separate files as a part of our storage schema, we need to adopt certain naming conventions for such folders and files.

Naming Conventions: A Hadoop Store can be composed of several distinct RDF graphs in our framework. Therefore, a separate folder will be created in HDFS for each such Hadoop Store. The name of this folder will correspond to the name that has been selected for the given store. Furthermore, a RDF graph is divided into several files in our framework depending on the storage layout that is selected. Therefore, a separate folder will be created in HDFS for each distinct RDF graph. The name of this folder is

defined to be “default” for the default RDF graph while for a named RDF graph; the URI of the graph is used as the folder name. We use the abstraction of a store in our framework for the reason that this will simplify the management of data belonging to various agencies. Two of the layouts to be supported by our framework are given below. These layouts use a varying number of HDFS files to store RDF data.

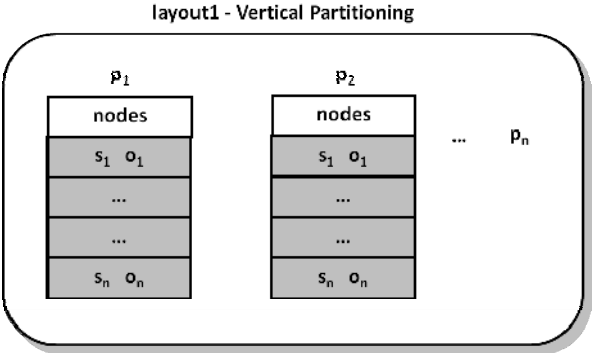

Fig. 9. Vertically Partitioned Layout

Vertically Partitioned Layout: Figure 9 presents the storage schema for the vertically partitioned layout. For every unique predicate contained in a RDF graph, this layout creates a separate file using the name of the predicate as the file name, in the underlying HDFS. Note that only the local name part of a predicate URI is used in a file name and a separate mapping exists between a file name and the predicate URI. A file for a given predicate contains a separate line for every triple that contains that predicate. This line stores the subject and object values that make up the triple. This schema will lead to significant storage space savings since moving the predicate name to the name of a file completely eliminates the storage of this predicate value. However, multiple occurrences of the same resource URI or literal value will be stored multiple times across all files as well as within a file. Additionally, a SPARQL query may need to lookup multiple files to ensure that a complete result is returned to a user, for example, a query to find all triples that belong to a specific subject or object.

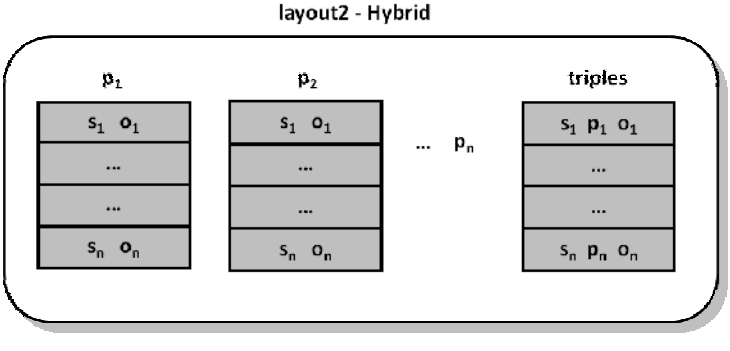

Fig. 10. Hybrid Layout

Hybrid Layout: Figure 10 presents the storage schema for the hybrid layout. This layout is an extension of the vertically partitioned layout, since in addition to the separate files that are created for every unique predicate in a RDF graph, it also creates a separate triples file containing all the triples in the SPO (Subject, Predicate, Object) format. The advantage of having such a file is that it directly gives us all triples belonging to a certain subject or object. Recall that such a search operation required scanning through multiple files in the vertically partitioned layout. The storage space efficiency of this layout is not as good as the vertically partitioned layout due to the addition of the triples file. However, a SPARQL query to find all triples belonging to a certain subject or object could be performed more efficiently using this layout.

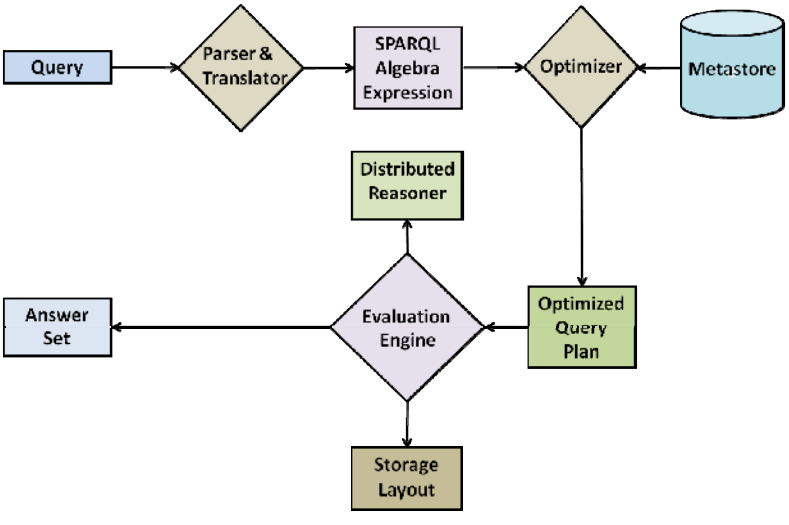

Fig. 11. Distributed processing of SPARQL in CAISS++

Distributed Processing of SPARQL. Query processing in CAISS++ comprises of several steps (Figure 11). The first step is query parsing and translation where a given SPARQL query is first parsed to verify syntactic correctness and then a parse tree corresponding to the input query is built. The parse tree is then translated into a SPARQL algebra expression. Since a given SPARQL query can have multiple equivalent SPARQL algebra expressions, we annotate each such expression with instructions on how to evaluate each operation in this expression. Such annotated SPARQL algebra expressions correspond to query-evaluation plans which serve as the input to the optimizer. The optimizer selects a query plan that minimizes the cost of query evaluation. In order to optimize a query, an optimizer must know the cost of each operation. To compute the cost of each operation, the optimizer uses a Metastore that stores statistics associated with the RDF data. The cost of a given query-evaluation plan is alternatively measured in terms of the number of MapReduce jobs or the number of triples that will be accessed as a part of query execution. Once the query plan is chosen, the query is evaluated with that plan and the result of the query is output. Since we use a cloud-centric framework to store RDF data, an evaluation engine needs to convert SPARQL

algebra operators into equivalent MapReduce jobs on the underlying storage layouts (described earlier). Therefore, in CAISS++ we will implement a MapReduce job for each of the SPARQL algebra operators. Additionally, the evaluation engine uses a distributed reasoner to compute inferences required for query evaluation.

Framework Integration: The components that we have outlined that are a part of CAISS++ need to be integrated to work with another. Furthermore, this process of integration depends on a user's selection of one of the three possible design choices provided with CAISS++, namely, Centralized CAISS++, Decentralized CAISS++ or Hybrid CAISS++. The integration of the various pieces of CAISS++ that have been presented so far needs to take into account several issues. First, we need to make sure that our ontology engineering process has been successful in capturing an agency's requirements and additionally, the ontologies can be stored in the storage schema used by the Hadoop Storage Architecture. Secondly, we need to ensure that the distributed SPARQL query processor is able to efficiently evaluate queries (i.e., user-generated SPARQL queries as well as SPARQL queries that evaluate policies) over the underlying RDF storage. Finally, we need to conduct a performance evaluation of CAISS++ to verify that it meets the performance requirements of various participating agencies as well as leads to significant performance advantages when compared with CAISS.

Policy Specification and Enforcement: The users of CAISS++ can use a language of their choice (e.g., XACML, RDF, Rei, etc) to specify their information sharing policies. These policies will be translated into a suitable sub-language of OWL using existing or custom-built translators. We will extend our policy engine for CAISS to handle policies specified in OWL. In addition to RDF policies, our current policy engine can handle policies in OWL for implementing role-based access control, inference control, and social network analysis (please see Section 4).

2.3 Formal Policy Analysis

Our proposed framework is applicable to a variety of mission-critical, high-assurance applications that span multiple possibly mutually-distrusting organizations. In order to provide maximal security assurance in such settings, it is important to establish strong formal guarantees regarding the correctness of the system and the policies it enforces. To that end, we propose to examine the development of an infrastructure for constructing formal, machine-checkable proofs of important system properties and policy analyses for our system. While machine-checkable proofs can be very difficult and time-consuming to construct for many large software systems, our choice of SPARQL, RDF, and OWL as query, ontology, and policy languages, opens unique opportunities to elegantly formulate such proofs in a logic programming environment. We will encode policies, policy-rewriting algorithms, and security properties as a rule based, logical derivation system in Prolog, and will apply model-checking and theorem-proving systems such as ACL2 to produce machine-checkable proofs that these properties are obeyed by the system. Properties that we intend to consider in our model include soundness, transparency, consistency and completeness. The results of our formal policy analysis will drive our detailed design and implementation of CAISS++. To our knowledge, none of the prior work has focused on such formal policy analysis for SPARQL, RDF and OWL. Our extensive research on formal policy analysis with in-line reference monitors is discussed under related work.

2.4 Implementation Approach

The implementation of CAISS is being carried out in Java and is based on a flexible design where we can plug and play multiple components. A service provide and/or user will have the flexibility to use the SPARQL query processor as well as the RDF-based policy engine as separate components or combine them. The open source component used for CAISS will include the Pellet reasoned as well as our in-house tools such as the SPARQL query processor on the Hadoop/MapReduce framework as well as the Cloud-centric RDF policy engine. CAISS will allow us to demonstrate basic AIS scenarios on our cloud based framework.

In the implementation of CAISS++, we will again use Java as the programming language. We will use Protégé as our ontology editor during the process of ontology engineering which includes designing domain ontologies as well as the upper ontology. We will also evaluate several existing distributed reasoning algorithms such as WebPIE and QueryPIE to determine the best algorithm that matches an agency's requirements. The selected algorithm will then be used to perform reasoning over OWL-based security policies. Additionally, the design of the Hadoop Storage Architecture is based on Jena's SPARQL Database (SDB) architecture and will feature some of the functionalities that are available with Jena SDB. The SPARQL query engine will also feature code written in Java. This code will consist of several modules including query parsing and translation, query optimization and query execution. The query execution module will consist of MapReduce jobs for the various operators of the SPARQL language. Finally, our Web-based user interface will make use of several components such as JBoss, EJB, JSF, among others.

3 Related Work

We will first provide an overview of our research directly relevant to our project and then discuss overall related work. We will also discuss product/technology competition.

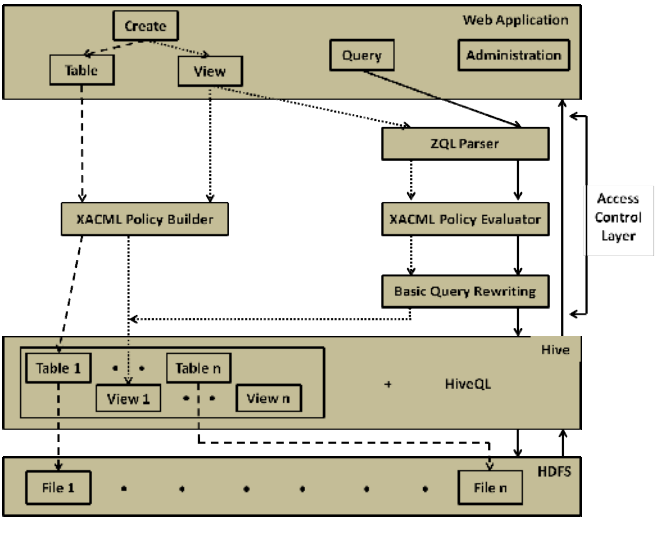

Fig. 12. HIVE-based Assured Cloud Query Processing

3.1 Our Related Research

Secure Data Storage and Retrieval in the Cloud. We have built a web-based application that combines existing cloud computing technologies such as Hadoop an open source distributed file system and Hive data warehouse infrastructure built on top of Hadoop with a XACML policy based security mechanism to allow collaborating organizations to securely store and retrieve large amounts of data [9, 12, 19]. Figure 12 presents the architecture of our system. We use the services provided by the HIVE layer and Hadoop including the **Hadoop Distributed File System (HDFS)** layer that makes up the storage layer of Hadoop and allows the storage of data blocks across a cluster of nodes. The layers we have implemented include the web application layer, the ZQL parser layer, the XACML policy layer, and the query rewriting layer. The **Web Application layer** is the only interface provided by our system to the user to access the cloud infrastructure. The **ZQL Parser** [20] layer takes as input any query submitted by a user and either proceeds to the XACML policy evaluator if the query is successfully parsed or returns an error message to the user.

The **XACML Policy Layer** is used to build (XACML Policy Builder) and evaluate (XACML Policy Evaluation) XACML policies. The **Basic Query Rewriting Layer** rewrites SQL queries entered by the user. The **Hive** layer is used to manage relational data that is stored in the underlying Hadoop HDFS [21]. In addition, we have also designed and implemented secure storage and query processing in a hybrid cloud [22].

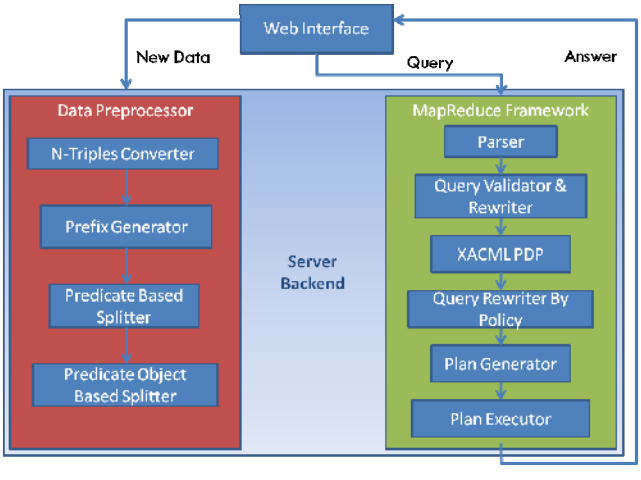

Fig. 13. SPARQL-based Assured Cloud Query Processing

Secure SPARQL Query Processing on the Cloud. We have developed a framework to query RDF data stored over Hadoop as shown in Figure 13. We used the Pellet reasoner to reason at various stages. We carried out real-time query reasoning using the pellet libraries coupled with Hadoop's MapReduce functionalities. Our RDFquery processing is composed of two main steps: 1) the preprocessing and 2) the query optimization and execution.

Pre-processing: In order to execute a SPARQL query on RDF data, we carried out data pre-processing steps and stored the pre-processed data into HDFS. A separate MapReduce task was written to perform the conversion of RDF/XML data into N-Triples as well as for prefix generation. Our storage strategy is based on predicate splits [12].

Query Execution and Optimization: We have developed a SPARQL query execution and optimization module for Hadoop. As our storage strategy is based on predicate splits, first, we examine the predicates present in the query. Second, we examine a subset of the input files that are matched with predicates. Third, SPARQL queries generally have many joins in them and all of these joins may not be possible to perform in a single map-reduce job. Therefore, we have developed an algorithm that decides the number of jobs required for each kind of query. As part of optimization, we applied a greedy strategy and cost-based optimization to reduce query processing time. We have also developed a XACML-based centralized policy engine that will carry out federated RDF query processing on the cloud. Details of the enforcement strategy are given in [12, 23, 24].

RDF Policy Engine. In our prior work [11], we have developed a policy engine to processes RDF-based access control policies for RDF data. The policy engine is designed with the following features in mind: scalability, efficiency and interoperability. This framework (Figure 14) can be used to execute various policies, including access control policies and redaction policies. It can also be used as a testbed for evaluating different policy sets over RDF data and to view the outcomes graphically. Our framework presents an interface that accepts a high level policy, which is then translated into the required format. It takes a user's input query and returns a response which has been pruned using a set of user-defined policy constraints. The architecture is built using a modular approach, therefore it is very flexible in that most of the

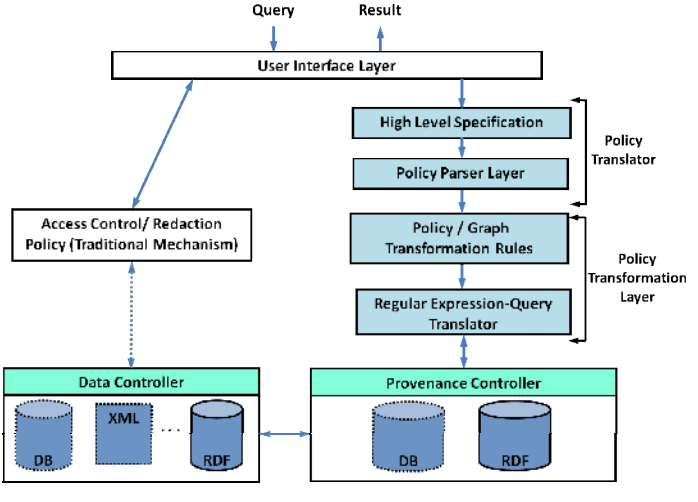

Fig. 14. RDF Policy Engine

modules can be extended or replaced by another application module. For example, a policy module implementing a discretionary access control (DAC) could be replaced entirely by a RBAC module or we may decide to enforce all our constraints based on a generalized redaction model. It should be noted that our policy engine also handles role-based access control policies specified in OWL and SWRL [25]. In addition, it handles certain policies specified in OWL for inference control such as association based policies where access to collections of entities is denied and logical policies where A implies B and if access to B is denied then access to A should also be denied [25-27]. This capability of our policy engine will be useful in our design and implementation of CAISS++ where information is shared across multiple clouds.

Assured Information Sharing Prototypes. We have developed multiple systems for AIS at UTD. Under an AFOSR funded project (between 2005-2008) we developed an XACML based policy engine to function on top of relational databases and demonstrated the sharing of (simulated) medical data [6]. In this implementation, we specified the policies in XACML and stored the data in multiple Oracle database. When one organization request data from another organization, the policies are examined and authorized data is released. In addition, we also conducted simulation studies on the amount of data that would be lost by enforcing the policies while information sharing. Under our current MURI project, also funded by AFOSR, we have conducted simulation studies for incentive based information sharing [28]. We have also examined risk based access control in an information sharing scenario [29]. In addition to access control policies, we have specified different types of policies including need to share policies and trust policies (e.g., A shared data with B provided B does not share the data with C). Note that the 9/11 commission report calls for the migration from the more restrictive need-to-know to the less restrictive need-to-share policies. These policies are key to support the specification of directive concerning AIS obligations.

Formal Policy Analysis: UTD PI Hamlen is an expert in the emerging field of language-based security, which leverages techniques from programming language theory and compilers to enforce software security and policy analysis. By reducing high-level security policy specifications and system models to the level of the denotational and operational semantics of their binary-level implementations, our past work has developed formally machine-certifiable security enforcement mechanisms of a variety of complex software systems, including those implemented in .NET [16], ActionScript [19], Java [13], and native code [31]. Working at the binary level provides extremely high formal guarantees because it permits the tool chain that produces mission-critical software components to remain untrusted; the binary code produced by the chain can be certified directly. This strategy is an excellent match for CAISS++ because data security specification languages such as XACML and OWL can be elegantly reflected down to the binary level of bytecode languages with XML-aware system APIs, such as Java bytecode. Our past work has applied binary-instrumentation (e.g., in-lined reference monitoring) and a combination of binary type-checking [30], model-checking [18], and automated theorem proving (e.g., via ACL2) to achieve fully automated machine certification of binary software in such domains.

3.2 Overall Related Research

While there are some related efforts none of the efforts have provided a solution to AIS in the cloud, nor have they conducted such a formal policy analysis.

Secure Data Storage and Retrieval in the Cloud. Security for cloud has received recent attention [31]. Some efforts on implementing at the infrastructure level have been reported [32]. Such development efforts are an important step towards securing cloud infrastructures but are only in their inception stages. The goal of our system is to add another layer of security above the security offered by Hadoop [19]. Once the security offered by Hadoop becomes robust it will only strengthen the effectiveness of our system. Similar efforts have been undertaken by Amazon and Microsoft for their cloud computing offerings [33-34]. However, this work falls in the public domain whereas our system is designed for a private cloud infrastructure. This distinguishing factor makes our infrastructure "trusted" over public infrastructures where the data must be stored in an encrypted format.

SPARQL Query Processor. Only a handful of efforts have been reported on SPARQL query processing. These include BioMANTA [35] and SHARD [36]. Bio-MANTA proposes extensions to RDF Molecules [37] and implements a MapReduce based Molecule store [38]. They use MapReduce to answer the queries. They have queried a maximum of 4 million triples. Our work differs in the following ways: first, we have queried 1 billion triples. Second, we have devised a storage schema which is tailored to improve query execution performance for RDF data. To our knowledge, we are the first to come up with a storage schema for RDF data using flat files in HDFS, and a MapReduce job determination algorithm to answer a SPARQL query. SHARD (Scalable, High-Performance, Robust and Distributed) is a RDF triple store using the Hadoop Cloudera distribution. This project shows initial results demonstrating Hadoop's ability to improve scalability for RDF datasets. However, SHARD stores its data only in a triple store schema. It does no query planning or reordering, and its query processor will not minimize the number of Hadoop jobs. None of the efforts have incorporated security policies.

RDF-Based Policy Engine. There exists prior research devoted to the study of enforcing policies over RDF stores. These include the work in [39], which uses RDF for policy specification and enforcement. In addition, the policies are generally written in RDF. In [40], the authors propose an access control model for RDF. Their model is based on RDF data semantics and incorporates RDF and RDF Schema (RDFS) entailments. Here protection is provided at the resource level, which adds granularity to their framework. Other frameworks enforcing policies over RDF\OWL include [41-42]. [41] describes KAoS, a policy and domain services framework that uses OWL both, to represent policies and domains. [42] introduces Rei, a policy framework that is flexible and allows different kinds of policies to be stated. Extensions to Rei have been proposed recently [43]. The policy specification language allows users to develop declarative policies over domain specific ontologies in RDF, DAML+OIL and OWL. The authors in [44] also introduced a prototype, RAP, for implementation of an RDF store with integrated maintenance capabilities and access control. These frameworks, however do not address cases where the RDF store can become very large or the case where the policies do not scale with the data. Under an IARPA funded project, we have developed techniques for very large RDF graph processing [45].

Hadoop Storage Architecture. There has been significant interest in large-scale distributed storage and retrieval techniques for RDF data. The theoretical designs of a parallel processing framework for RDF data are presented in the work done by Castagna et al. [46]. This work advocates the use of a data distribution model with varying levels of granularity such as triple level, graph level and dataset level. A query over such a distributed model is then divided into a set of sub-queries over machines containing the distributed data. The results of all sub-queries will then be merged to return a complete result to a user application. Several implementations of this theoretical concept exist in the research community. These efforts include the work done by Choi et al. [47] and Abraham et al. [48]. A separate technique that has been used to store and retrieve RDF data makes use of peer-to-peer systems [49-52]. However, there are some drawbacks with such systems as peer-to-peer systems need to have super peers that store information about the distribution of RDF data among the peers. Another disadvantage is a need to federate a SPARQL query to every peer in the network.

Distributed Reasoning. InteGrail system uses distributed reasoning, whose vision is to shape the European railway organization of the future [53]. In [54] authors have shown a scalable implementation of RDFS reasoning based on MapReduce which can infer 30 billion triples from a real-world dataset in less than two hours, yielding an input and output throughput of 123.000 triples/second and 3.27 million triples/second respectively. They have presented some non-trivial optimizations for encoding the RDFS ruleset in MapReduce and have evaluated the scalability of their implementation on a cluster of 64 compute nodes using several real-world datasets.

Access Control and Policy Ontology Modeling. There have been some attempts to model access control and policy models using semantic web technologies. In [55], authors have shown how OWL and Description Logic can be used to build an access control system. They have developed a high level OWL-DL ontology that expresses the elements of a role based access control system and have built a domain-specific ontology that captures the features of a sample scenario. Finally, they have joined these two artifacts to take into account attributes in the dentition of the policies and in the access control decision. In [56], authors first presented a security policy ontology based on the DOGMA which is a formal ontology engineering framework. This ontology covers the core elements of security policies (i.e. Condition, Action, Resource) and can easily be extended to represent specific security policies, such as access control policies. In [57], authors present an ontologically-motivated approach to multi-level access control and provenance for information systems.

3.3 Commercial Developments

RDF Processing Engines: Research and commercial RDF processing engines include Jena by HP labs, BigOWLIM and RDF-3X. Although the storage schemas and query processing mechanisms for some of these tools are proprietary, they are all based on some type of indexing strategy for RDF data. However, only a few tools exist that use a cloud-centric architecture for processing RDF data and moreover, these tools are not salable to a very large number of triples. In contrast, our proposed query processor in CAISS++, will be built as a planet-scale RDF processing engine

that supports all SPARQL operators and will provide optimized execution strategies for SPARQL queries and can scale to billions of triples. **Semantic Web based Security Policy engines**: As stated in Section 3.2, the current work on semantic web-based policy specification and enforcement does not address the issues of policy generation and enforcement for massive amounts of data and support large number of users. **Cloud:** To the best of our knowledge there is no significant commercial competition for cloud-centric AIS. Since we have taken a modular approach to the creation of our tools, we can iteratively refine each component (policy engine, storage architecture and query processor) separately. Due to the component-based approach we have taken, we will be able to adapt to changes in the platforms we use (e.g., Hadoop, RDF, OWL and SPARQL) without having to depend on the particular features of a given platform.

4 Summary and Directions

This paper has described our design and implementation of a cloud-based information sharing system that called CAISS. CAISS utilizes several of the technologies we have developed for AFOS as well as open source tools. We also described the design of an ideal cloud-based assured information sharing system called CAISS++. Based on the lessons learned from the implementation of CAISS we will then carry out a detailed design of CAISS++ and subsequently implement the system that will be the first of its kind for cloud-based assured information sharing.

References

1. NSA Pursues Intelligence-Sharing Architecture, `http://www.informationweek.com/news/government/cloud-saas/229401646`
2. DoD Information Enterprise Strategic Plan (2010-2012), `http://cio-nii.defense.gov/docs/DodIESP-r16.pdf`
3. Department of Defense Information Sharing Strategy (2007), `http://dodcio.defense.gov/docs/InfoSharingStrategy.pdf`
4. DoD Embraced Cloud Computing, http://www.defensemarket.com/?p=67
5. Finin, T., Joshi, A., Kargupta, L., Yesha, Y., Sachs, J., Bertino, E., Li, N., Clifton, C., Spafford, G., Thuraisingham, B., Kantarcioglu, M., Bensoussan, A., Berg, N., Khan, L., Han, J., Zhai, C., Sandhu, R., Xu, S., Massaro, J., Adamic, L.: Assured Information Sharing Life Cycle. In: Proc. Intelligence and Security Informatics (2009)
6. Thuraisingham, B., Kumar, H., Khan, L.: Design and Implementation of a Framework for Assured Information Sharing Across Organizational Boundaries. Journal of Information Security and Privacy (2008)
7. Awad, M., Khan, L., Thuraisingham, B.: Policy Enforcement System for Inter-Organizational Data Sharing. Journal of Information Security and Privacy 4(3) (2010)
8. Rao, P., Lin, D., Bertino, E., Li, N., Lobo, J.: EXAM: An Environment for Access Control Policy Analysis and Management. In: Proc. POLICY 2008 (2008)
9. Thuraisingham, B., Khadilkar, V., Gupta, A., Kantarcioglu, M., Khan, L.: Secure Data Storage and Retrieval in the Cloud. In: CollaborateCom 2010 (2010)

10. Thuraisingham, B., Khadilkar, V.: Assured Information Sharing in the Cloud, UTD Tech. Report (September 2011)
11. Cadenhead, T., Khadilkar, V., Kantarcioglu, M., Thuraisingham, B.: Transforming provenance using redaction. In: Proc. ACM SACMAT (2011)
12. Husain, M.F., McGlothlin, J., Masud, M., Khan, L., Thuraisingham, B.: Heuristics-Based Query Processing for Large RDF Graphs Using Cloud Computing. IEEE Trans. Knowl. Data Eng. 23 (2011)
13. Jones, Hamlen: Disambiguating aspect-oriented security policies. In: Proc. 9th Int. Conf. Aspect-Oriented Software Development, pp. 193–204 (2010)
14. Jones, M., Hamlen, K.: A service-oriented approach to mobile code security. In: Proc. 8th Int. Conf. Mobile Web Information Systems (2011)
15. Hamlen, K., Morrisett, G., Schneider, F.: Computability classes for en-forcement mechanisms. ACM Trans. Prog. Lang. and Systems 28(1), 175–205 (2006)
16. Hamlen, K., Morrisett, G., Schneider, F.: Certified in-lined reference monitoring on.NET. In: Proc. ACM Workshop on Prog. Lang. and Analysis for Security, pp. 7–16 (2006)
17. Guo, Y., Heflin, J.: LUBM: A Benchmark for OWL Knowledge Base Systems. Web Semantics 3 (2005)
18. Sridhar, M., Hamlen, K.W.: Model-Checking In-Lined Reference Monitors. In: Barthe, G., Hermenegildo, M. (eds.) VMCAI 2010. LNCS, vol. 5944, pp. 312–327. Springer, Heidelberg (2010)
19. UTD Secure Cloud Repository, `http://cs.utdallas.edu/secure-cloud-repository/`
20. Zql: a Java SQL parser, `http://www.gibello.com/code/zql/`
21. Thusoo, A., Sharma, J., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., Murthy, R.: Hive - A Warehousing Solution Over a Map-Reduce Framework. In: PVLDB (2009)
22. Khadilkar, V., Kantarcioglu, M., Thuraisingham, B., Mehrotra, S.: Secure Data Processing in a Hybrid Cloud Proc. CoRR abs/1105.1982 (2011)
23. Hamlen, K., Kantarcioglu, M., Khan, L., Thuraisingham, B.: Security Issues for Cloud Computing. Journal of Information Security and Privacy 4(2) (2010)
24. Khaled, A., Husain, M., Khan, L., Hamlen, K., Thuraisingham, B.: A To-ken-Based Access Control System for RDF Data in the Clouds. In: CloudCom 2010 (2010)
25. Cadenhead, T., De Meuter, W., Thuraisingham, B.: Scalable and Efficient Reasoning for Enforcing Role-Based Access Control. In: Foresti, S., Jajodia, S. (eds.) Data and Applications Security XXIV. LNCS, vol. 6166, pp. 209–224. Springer, Heidelberg (2010)
26. Cadenhead, T., Khadilkar, V., Kantarcioglu, M., Thuraisingham, B.: A language for provenance access control. In: Proc. ACM CODASPY 2011 (2011)
27. Carminati, B., Ferrari, E., Heatherly, R., Kantarcioglu, M., Thuraisingham, B.: A semantic web based framework for social network access control. In: SACMAT 2009 (2009)
28. Kantarcioglu, M.: Incentive-based Assured Information Sharing. AFOSR MURI Review (October 2010)
29. Celikel, E., Kantarcioglu, M., Thuraisingham, B., Bertino, E.: Managing Risks in RBAC Employed Distributed Environments. In: Meersman, R. (ed.) OTM 2007, Part II. LNCS, vol. 4804, pp. 1548–1566. Springer, Heidelberg (2007)
30. Hamlen, K., Mohan, V., Wartell, R.: Reining in Windows API abuses with in-lined reference monitors. Tech. Rep. UTDCS-18-10, Comp. Sci. Dept., U. Texas at Dallas (2010)
31. Talbot, D.: How Secure is Cloud Computing?, `http://www.technologyreview.com/computing/23951/`

32. O'Malley, O., Zhang, K., Radia, S., Marti, R., Harrell, C.: Hadoop Security Design, http://bit.ly/75011o
33. Amazon Web Services: Overview of Security Processes, http://awsmedia.s3.amazonaws.com/pdf/AWSSecurityWhitepaper.pdf
34. Marshall, A., Howard, M., Bugher, G., Harden, B.: Security best practices in developing Windows Azure Applications, Microsoft Corp. (2010)
35. BioMANTA: Modelling and Analysis of Biological Network Activity, http://www.itee.uq.edu.au/reresearch/projects/biomanta
36. SHARD, http://www.cloudera.com/blog/2010/03/how-raytheonresearchers-are-using-hadoop-to-build-a-scalable-distributed-triplestore
37. Ding, L., Finin, T., Peng, Y., da Silva, P., Mcguinness, D.: Tracking RDF Graph Provenance using RDF Molecules. In: Proc. International Semantic Web Conference (2005)
38. Newman, A., Hunter, J., Li, Y., Bouton, C., Davis, M.: A Scale-Out RDF Molecule Store for Distributed Processing of Biomedical Data. In: Semantic Web for Health Care and Life Sciences Workshop, WWW 2008 (2008)
39. Carminati, B., Ferrari, E., Thuraisingham, B.: Using RDF for policy specification and enforcement. In: DEXA 2004 (2004)
40. Jain, A., Farkas, C.: Secure resource description framework: an access control model. In: ACM SACMAT 2006 (2006)
41. Uszok, A., Bradshaw, J., Johnson, R., Jeffers, M., Tate, A., Dalton, J., Aitken, S.: KAoS policy management for semantic web services. Intelligent Systems (2004)
42. Kagal, L.: Rei: A policy language for the me-centric project. In, HP Labs (2002), accessible online, http://www.hpl.hp.com/techreports/2002/HPL-2002-270.html
43. Khandelwal, A., Bao, J., Kagal, L., Jacobi, I., Ding, L., Hendler, J.: Analyzing the AIR Language: A Semantic Web (Production) Rule Language. In: Hitzler, P., Lukasiewicz, T. (eds.) RR 2010. LNCS, vol. 6333, pp. 58–72. Springer, Heidelberg (2010)
44. Reddivari, P., Finin, T., Joshi, A.: Policy-based access control for an RDF store. In: Policy Management for the Web, IJCAI Workshop (2005)
45. UTD Semantic Web Repository, http://cs.utdallas.edu/semanticweb/
46. Castagna, P., Seaborne, A., Dollin, C.: A Parallel Processing Framework for RDF Design and Issues. Technical report, HP Laboratories (2009)
47. Choi, H., Son, J., Cho, Y., Sung, M., Chung, Y.: SPIDER: A System for Scalable, Parallel / Distributed Evaluation of large-scale RDF Data. In: Proceedings ACM CIKM (2009)
48. Abraham, J., Brazier, P., Chebotko, A., Navarro, J., Piazza, A.: Distributed Storage and Querying Techniques for a Semantic Web of Scientific Workflow Provenance. In: Proceedings IEEE SCC (2010)
49. Aberer, K., Cudré-Mauroux, P., Hauswirth, M., Van Pelt, T.: GridVine: Building Internet-Scale Semantic Overlay Networks. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 107–121. Springer, Heidelberg (2004)
50. Cai, M., Frank, M.: RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network. In: Proceedings ACM WWW (2004)
51. Harth, A., Umbrich, J., Hogan, A., Decker, S.: YARS2: A Federated Reposi-tory for Searching and Querying Graph Structured Data. Technical report, DERI (2007)
52. Della Valle, E., Turati, A., Ghioni, A.: *PAGE*: A Distributed Infrastructure for Fostering RDF-Based Interoperability. In: Eliassen, F., Montresor, A. (eds.) DAIS 2006. LNCS, vol. 4025, pp. 347–353. Springer, Heidelberg (2006)

53. Distributed Reasoning: Seamless integration and processing of distributed knowledge, http://www.integrail.eu/documents/fs04.pdf
54. Urbani, J.: Scalable Distributed Reasoning using MapReduce, http://www.few.vu.nl/~jui200/papers/ISWC09-Urbani.pdf
55. Cirio, L., Cruz, I., Tamassia, R.: A Role and Attribute Based Access Control System Using Semantic Web Technologies. In: IFIP Workshop on Semantic Web and Web Semantics (2007)
56. Reul, Q., Zhao, G., Meersman, R.: Ontology-based access control policy inter-operability. In: Proc. 1st Conference on Mobility, Individualisation, Socialisation and Connectivity, MISC 2010 (2010)
57. Andersen, B., Neuhaus, F.: An ontological approach to information access control and provenance. In: Proceedings of Ontology for the Intelligence Community, Fairfax, VA (October 2009)