# Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 07-001

A Security-Enabled Grid System for Distributed Data Mining

Seonho Kim, Jinoh Kim, and Jon Weissman

January 10, 2007

# Technical Report

## A Security-Enabled Grid System for Distributed Data Mining

**Seonho Kim, Jinoh Kim, and Jon Weissman**

**January 9, 2007**

**Department of Computer Science & Engineering**
**University of Minnesota**
**4-192 EE/CS Building**
**200 Union Street SE**
**Minneapolis, MN 55455-0159 USA**

# A Security-Enabled Grid System for Distributed Data Mining

Seonho Kim, Jinoh Kim, and Jon Weissman
{shkim, jinohkim, jon}@cs.umn.edu
Department of Computer Science & Engineering
University of Minnesota

# 1. Introduction

In many areas of business, scientific, and engineering, interest has been increasing in exploration and mining of huge amount of data (generated everyday tera byte or peta byte in scale) to improve business processes, to make new scientific discoveries, or to detect possible network intrusions etc. The fast growing Internet and Web-based technologies as well as the capacity of hardware devices impact the nature of business and scientific computing. Data and computing resources are distributed across multiple sites in distributed environment and may belong to different organizations which may have different policies for data and resources in the organization. Even though a number of data analysis tools have been successfully developed for single site data exploration and mining, they are not suitable in distributed environments. It is mainly because security is of primary importance in such environments particularly where privacy related sensitive data such as patient medical information and log of network connections needs to be accessed and exchanged. Well-defined policies and mechanisms are required for secure communication, authentication, and authorization.

MINDS (Minnesota INtrusion Detection System) [1] [2] is a data mining based network intrusion detection system which is a combination of software and hardware that attempts to perform various network intrusion detection and raises an alarm when possible intrusion is detected. MINDS uses data mining techniques to detect novel attacks - anomaly detection, such as previously unseen/unknown/emerging attacks (in other words, anomalies) which are difficult to detect with the existing detection techniques (misuse detection), which build predictive models from labeled data sets to identify known intrusions and effectively detect many kinds of known attacks with high accuracy. The MINDS detects anomalies and attacks including scanning, worms, and non-standard behavior such as policy violations and insider abuse. Figure 1 shows MINDS system developed at AHPCRC (Army High Performance Computing Research Center) at the University of Minnesota [1]. The data capturing device such as network flow tools or network monitoring tools like *tcpdump* are used to collect the network traffic data from routers. The collected data is first filtered to remove unimportant network connections and to extract some features: 1) basic features such as source/destination IP address pair,

port number, and protocols and 2) some additional features such as the number of flows from the same source to specific destination IP addresses for a predetermined time period. The extracted features are input for MINDS system. While well-known intrusions are detected by the known attack detection module, the anomaly detection module is applied to the remaining network connections. During the process of anomaly detection, a training set is generated if not given as input. The anomaly detection module scores each network connection to reflect the level of anomalousness of the network connection compared to the normal network traffic. The highly scored network connections are analyzed by the association pattern analysis module to produce summary and characterization of the possible attacks. This summary and characterization is used to create new signatures and models for unknown emerging attacks.

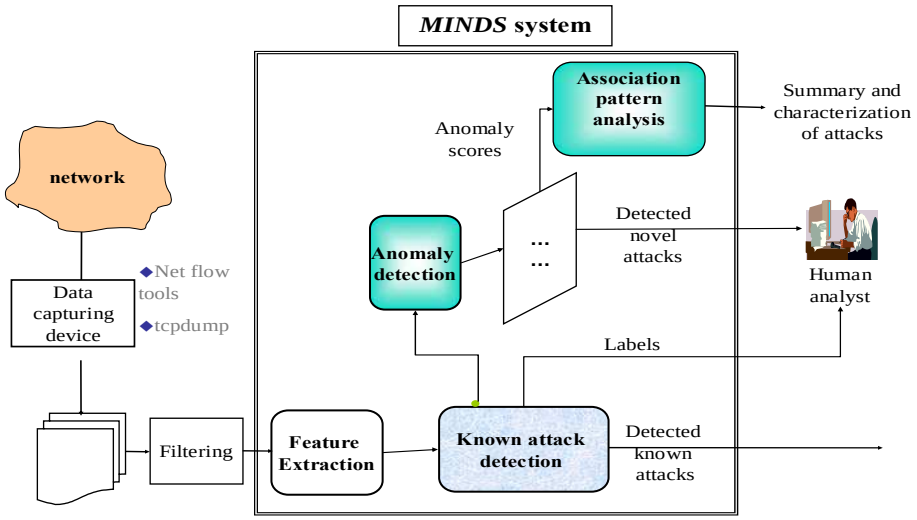



Figure 1: Minnesota INtrusion Detection System

As the network intrusions become more sophisticated hard to be identified by single site analyses, increases the need for coordination across multiple distributed sites to find the correlation of suspicious events and to share distributed anomaly data in different administrative domains. Grid and middleware technologies can be leveraged to address these newly emerging requirements. Grid technology has become a promising technology as an integration infrastructure for seamless sharing and coordinated use of geographically distributed huge amounts of data as well as heterogeneous high-

performance compute resources in Virtual Organizations (VOs) [3] [4]. Grid technology provides an infrastructure supporting secure, flexible, coordinated resource sharing among dynamic collections of individuals and institutions in a VO. Grid technology also addresses the sharing of data sources distributed diversely in their formats, access mechanisms/policies, and ownerships.

In this paper, we present a Grid system that enables distributed data mining, exploration and sharing to address issues described above which involve distributed data analysis and multiple ownerships. The system addresses the three main requirements of distributed data mining on Grid: 1) exploitation of geographically and organizationally distributed computing resources to solve data-intensive data mining problem, 2) ensuring the security and privacy of sensitive data, 3) supporting seamless data/computing resource sharing. We present the system architecture, specification of the component services, security consideration, a prototype of the system, and performance evaluation.

The rest of paper is organized as follows. In section 2, we present layered Grid system stack, system architecture, and specification and APIs of system components. Section 3 describes the security considerations for MINDS Grid system and we present a test-bed and prototype implementation of the system in section 4. In section 5, we present initial performance measurements. Conclusion is presented in section 6.

## 2. System Architecture

### 2.1 Layered Grid System Stack

Figure 2 shows a layered grid system stack which the MINDS Grid system is based on. Service oriented architecture (SOA) is at the bottom of the layer. SOA is an architectural style for building software applications using network-accessible services [5]. In SOA, services are defined as loosely-coupled, implementation-independent, and well-defined software interfaces. Web services is a way of realizing SOA based on XML-based open standards such as SOAP [6], UDDI [7], and WSDL [8] which provide a common approach for defining, publishing, and utilizing Web services.

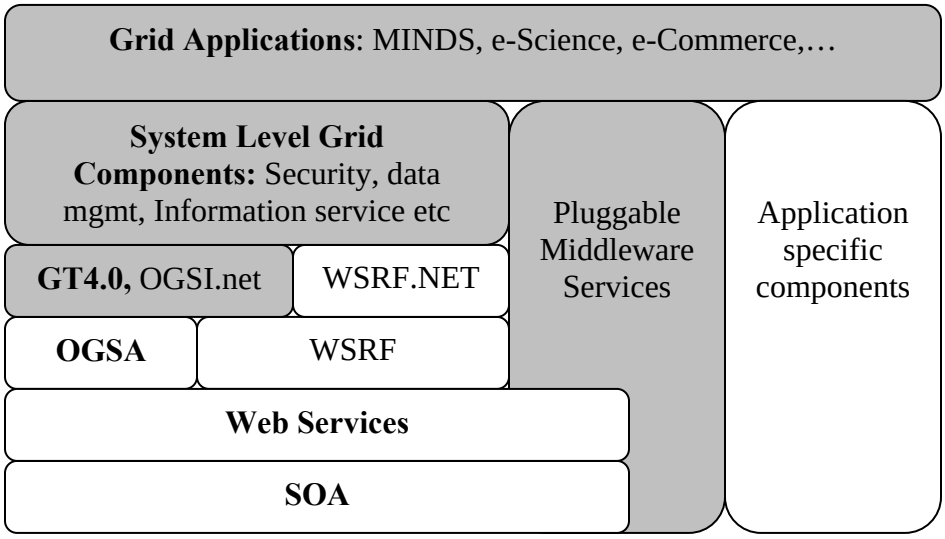



Figure 2: Layered Grid System Stack

The Open Grid Service Architecture (OGSA)[3][4], which is developed by the Global Grid Forum, requires stateful Web services specified by WSRF (Web Services Resource Framework). OGSA defines a common, standard, and open architecture for grid-based applications in Virtual Organizations (VOs) where secure, flexible, coordinated resource sharing among dynamic collections of individuals and institutions is required. The WSRF [10][11] has been led by the Globus Alliance [9] to define a generic framework for modeling and accessing persistent resources using Web services. The WSRF defines conventions for managing states and for integration and management of

multiple services. Globus Toolkit 4.0 [12], which implements OGSA and WSRF, is a software toolkit that can be used to program Grid-based applications. It is composed of several system-level components divided into five categories: security, data management, execution management, information services, and common runtime. These system level grid components provide core functionality that is required by application-level Grid services. Grid applications are built on top of system-level Grid components, application-level grid services, application specific components, and pluggable middleware services such as a dynamic OGSA-based Grid middleware service supporting dynamic service hosting [13][14] and a resource management middleware that dynamically makes decisions on resources allocation/scheduling [15].

### 2.2 MINDS Grid Service

The MINDS Grid system is built on top of the layered grid system stack described in the previous section. In the MINDS Grid system, the MINDS system is transformed into a Grid service and deployed on distributed service containers. By separating the process of training set generation (data pre-processing) from the given input data, the data can be decomposed into multiple fragments and distributed among different MINDS Grid services for parallel analysis. The figure 3 describes the workflow of our MINDS Grid system. The MINDS Grid service front-end is an entry point for the MINDS service requests. The scheduling module inside the front-end sets up a plan regarding where to pre-process data for training set generation if necessary, how to decompose the data, where to send the decomposed data (that is, where to run the MINDS Grid service back-end), whether to deploy new MINDS back-end Grid services if necessary, where to aggregate the result datasets, and finally where to store. The MINDS Grid service front-end may coordinate with different Grid services such as storage service, and with various middleware-level services such as replication service, data management service (e.g., GridFTP [12] for data transfer), and security infrastructure for authentication and authorization.

For regulated data access within the data mining community, a community level security authority (CSA, Community Security Authority) is required. The authority expresses policies regarding four main principals in the community – users (and user

groups), resource providers (storage provider and compute provider), data (and data groups), and applications - and relationships between them. The CSA uses a catalogue service to manage the catalogue of raw input data, processed data (alert and summary), and replicated datasets, and it maintains a database containing policies, as shown in figure 3. The CSA provides APIs to define a relationship between principals, APIs to set a security level of principals, and APIs for query and notification. For example, the scheduler queries the CSA to get a list of compute resource providers where the input data can be securely processed and a list of storage resource providers where to store result output securely. The CSA returns lists based on the user's ID and the group ID.

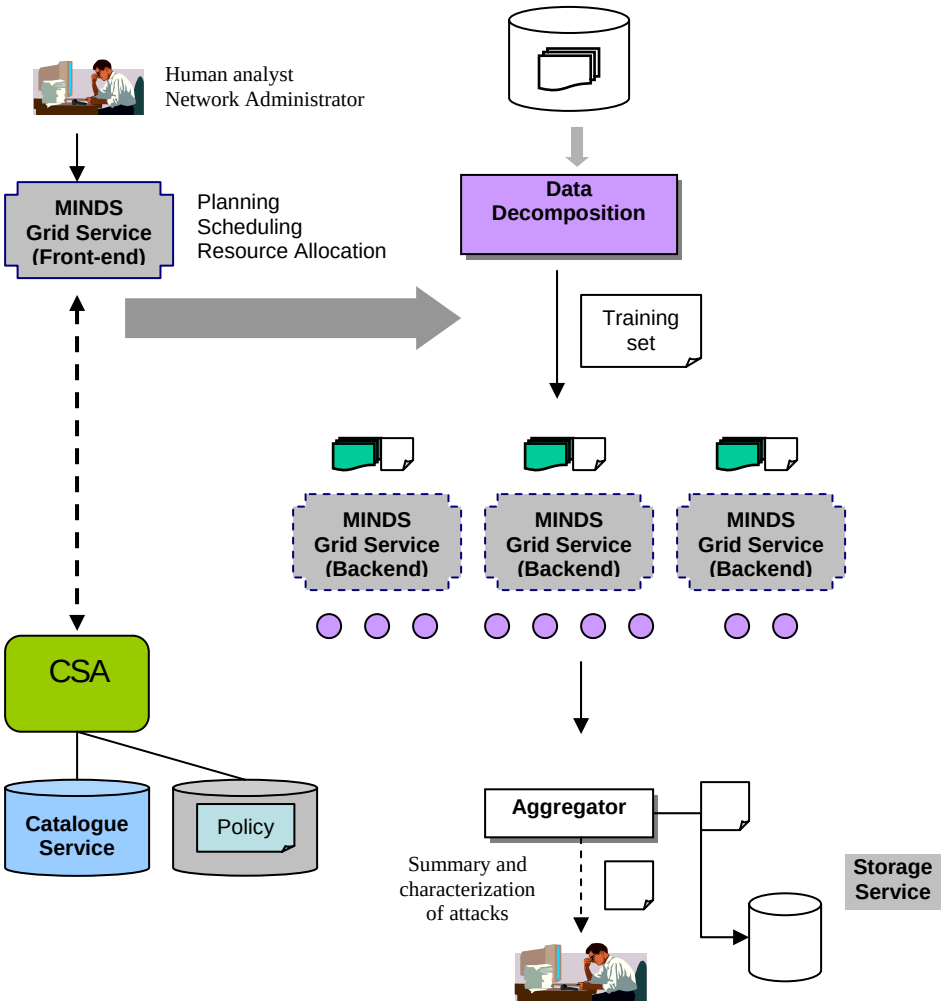



Figure 3: MINDS-Grid system workflow

The CSA is in charge of automated alert subscription. When an alert is generated, the owner of the alert registers it to the CSA with a unique ID. Then CSA automatically notifies the users in the group via email with a summary and a link to the detailed alerts. If a user receives a notification, the user accesses the alerts through authentication and authorization module of the CSA by contacting the front-end. A user might want to query the CSA to examine certain type of alerts stored in the community. The CSA returns alerts in one piece or subsets of them according to the security level of the user.

## 2.3 MINDS Grid System Architecture & APIs

In order to realize the workflow discussed in section 2.2, we designed a Grid system shown in figure 4. The system consists of three main component services: MINDS Service, Storage Service, and Community Security Authority. In order to address the first requirement - exploitation of geographically and organizationally distributed computing resources to solve data-intensive data mining problem, we designed the MINDS service as a composite service of a front-end and multiple back-ends.

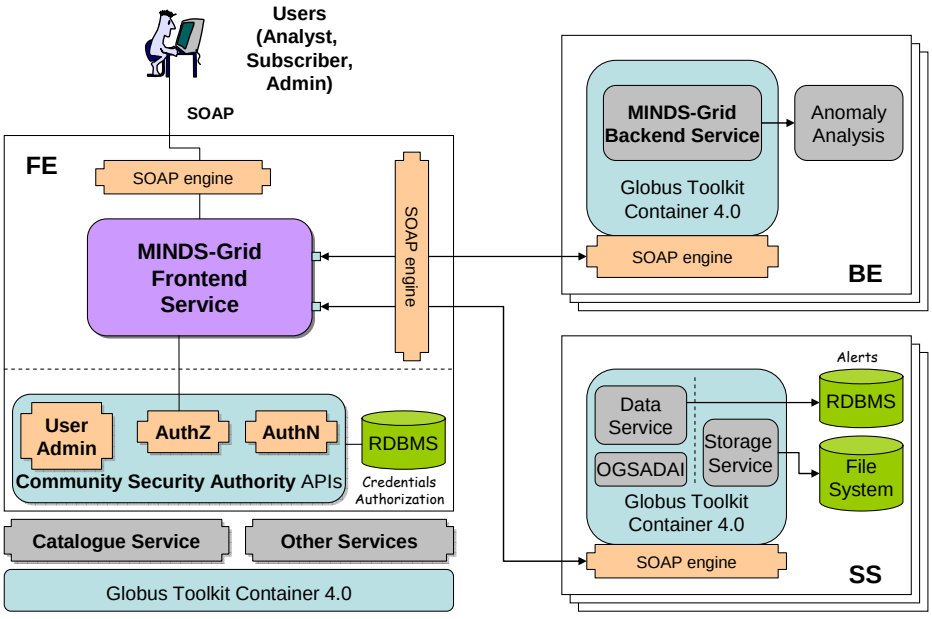



Figure 4: MINDS-Grid system architecture

The MINDS Grid service front-end supports planning, scheduling, and resource allocation for MINDS anomaly analysis. MINDS Grid service front-end service should adapt dynamically changing environment to make efficient decisions. We have developed runtime middleware frameworks that can be plugged in the front-end: a dynamic service hosting framework [14], a resource management middleware for dynamic resource allocation and job scheduling [15], a genetic algorithm based data-intensive application scheduling algorithms [16], and a dynamic resource leasing framework [17][18]. In collaboration with these runtime middleware frameworks, the front-end can set up a plan for data pre-processing (training set generation), input data decomposition, and parallel anomaly analysis, and aggregation of analysis results from MINDS back-end services, and finally store the aggregated analysis result into one or more storage services. Each MINDS back-end service does the actual MINDS anomaly analysis by taking one or more decomposed input data fragments and a training data. The MINDS front-end service coordinates with different Grid services such as storage service, Grid Security Infrastructure, and other various middleware-level services such as catalogue service, replica management service, data management service (e.g., GridFTP for data transfer) on top of Globus Toolkit infrastructure. To ensure the privacy and sensitivity of data, every communication between clients and Grid services are encrypted based on TLS (Transport Level Security)-based communication [22] and PKI (Public Key Infrastructure)-based authentication and authorization [24].

### 2.3.1 MINDS-Grid Service Front-end

The roles and APIs of the MINDS Grid service front-end are described in figure 5. The main role of the service front-end is planning, scheduling, and resource allocation. The service front-end provides users with interfaces for alert notification and sharing (storing/subscribing). The first interface is mindAnalysis(). This interface takes *NetworkFlowType* as input and returns *AlertType*. The *NetworkFlowType* contains raw network flow data and the security context information of the data. The *AlertType* contains a reference to an alert file and the security context information of the alert it represents. When a user wants to subscribe an alert, *UserType* needs to be given as input. The *UserType* defines the user's identity and the user security level with roles and

privileges. *AlertType* and *UserType* are associated with a specific *SecurityContextType* which defines the security context for principals such as users, services, and data resources. The security context contains information for authentication (AuthN) and authorization (AuthZ).

**Roles:**
Planning, scheduling, and resource allocation for MINDS anomaly analysis
Alert notification/sharing
Interfaces for Users

**APIs:**
AlertType **mindsAnalysis** (NetworkFlowType)
Decomposing data/job
Launching distributed MINDS analysis by calling service back-ends
Aggregating results
Setting a security context for the alert
Notifying/storing the alert by calling a Storage Service
AlertType **subscribeAlert** (AlertType, UserType)
Requesting alert retrieval from a storage service
Calling MINDS CSA for AuthN/AuthZ
Calling a storage service
void **storeAlert** (AlertType, SecurityContextType)
Requesting alert store to a storage service with a security context
Calling MINDS CSA for generating AuthZ data for the alert
Calling a storage service
Updating the catalogue with the newly stored alert
UserType **registerUser** (UserType, SecurityContextType)
Registering a new user with the security context given
Calling MINDS CSA to create AuthZ data for the user
void **unregisterUser** (UserType)
Calling MINDS CSA to delete authorization data for the user

Figure 5: MINDS-Grid Service front-end (MINDS Grid service front-end)

### 2.3.2 Storage Service

The storage service provides two interfaces: storeAlert () for analysts and retrieveAlert() for subscribers as shown in figure 6. The storage service is invoked by the MINDS front-end to store alerts. An alert can be stored in one or more storage services to improve subscription performance. When the MINDS front-end calls the retriveAlert(), it passes SecurityContextType data containing the user's security context. The storage service uses this information to filter the requested alert.

**Roles:**
Interface for storing/retrieving alerts
**APIs**
**void storeAlert (**AlertType**)**
AlertType **retrieveAlert (**AlertType**,** SecurityContextType**)**
Checking the security context of the entity requesting the alert
Filtering (or sub-setting) the alert according to the security context

Figure 6: Storage Service

### 2.3.3 Community Security Authority

The community security authority module [figure 7] consists of three components: authentication module, authorization module, and user provisioning module. The authentication module deals with user authentication mechanism. Each User and service in the MINDS Grid system is associated with *CredentialType* data to prove the authenticity to the system. The *CredentialType* defines credentials that are used for authentication, authorization, and secure communication (e.g., username/password pair, hardware generated key, or x509 certificates). In MINDS Grid system, the *ServiceType* defines the string representation of a service.

**Roles:**
Interface for security functionality
User authentication for the service
User authorization for alerts
Management of security context for principals (users, services, & data resources)
**APIs**
UserType **addUser (**UserType**,** SecurityContextType**)**
**void removeUser (**UserType**)**
UserType **setSecurityContext (**UserType**,** SecurityContextType**)**
SecurityContextType **getSecurityContext (**UserType**)**
AlertType **setSecurityContext** (AlertType, SecurityContextType)
SecurityContextType **getSecurityContext** (AlertType)
ServiceType **setSecurityContext** (ServiceType, SecurityContextType )
SecurityContextType **getSecurityContext** (ServiceType)
boolean **authenticate** (CredentialType)
Checking the authenticity of the user based on the credential submitted
SecurityContextType **getAuthZData** (UserType)
Checking whether the user is authenticated
Retrieving the general security context (roles and privileges) for the user

Figure 7: Community Security Authority Specification

## 3. Security Consideration in MINDS Grid system

When meaningful alerts are generated by the MINDS Grid service, summaries and alerts as characterization of potential attacks need to be notified to other permission-granted users (e.g., network administrators) immediately, stored for future user query, or replicated for efficient sharing. Since the privacy and sensitivity of data are critical factors that should be considered when scheduling data mining applications, handling and access of data (process, query, or subscription) should be carefully regulated according to appropriate access control policies. The distributed nature of data and computing resources as well as diverse ownership of data has important implications for four principals in a MINDS virtual organization which is consists of multiple physical sites:

1) [User: Data Analyst] Data analyst (that is, data owner) would like to mine information using state-of-the-art data exploration and data mining primitives, or develop their own application specific primitives by effectively utilizing the information and resources (compute resource and storage) from multiple organizations based on their privileges
2) [User: Data Subscriber] Data subscribers would like to effectively explore and utilize the information produced by data analysts from multiple organizations based on their privileges
3) [System Administrator] System administrators would like to effectively manage available compute/data resources and provide users with policy-based constrained access to the resources
4) [Data Resource] Data resources would need to be ensured that data access is provided to the user based on their privileges and to be guaranteed that a user does not compromise data privacy or the organization policy constraints

In this section, we discuss policies and mechanisms for secure communication, authentication, and authorization to address the implications on four principals discussed above.

### 3.1 Secure Communication

When communication is made between two entities over a network in MINDS VO, it is important that the data is securely transferred in its entirety without any security breach such as tampering, eavesdropping, or message compromise. Transport Layer Security (TLS) [22] and Secure Sockets Layer (SSL) [23] protocols provide a mechanism for secure communication between software client and server on the Internet using cryptography technology, that is, every message is encrypted before transmission and decrypted after transmission. In order to set up a secure channel between a client and a server, mutual authentication (identity verification on both sides) should be made. In TLS, mutual authentication is made by PKI certificate based mechanism.

### 3.2 Authentication

In order for a user to access MINDS Grid system, he or she should be authenticated to the system. There are a lot of different user authentication technologies such as password based Kerberos authentication [19], SASL [20] and GSSAPI [21], X.509 PKI [24] based authentication, and smart card technology etc. Followings are features required for user authentication in MINDS Grid system.

i) [*Orthogonality*] Authentication framework should be orthogonal to the other components of the system so that authentication mechanism can be plugged or updated easily without significant system modification

ii) [*Standard*] Authentication framework needs to be based on standards for easy integration and management

iii) [*Single Sign-On*] Authentication framework should support single sign on (SSO) so that a user authenticates once and can use the multiple services without re-signing-on

iv) [*Mutual Authentication*] Authentication mechanism should be mutual in the sense that it should provide a way to allow users (or software clients) to verify the identity of the server and vice versa

MINDS Grid system authentication framework is built based on X.509 PKI-based security scheme. It supports mutual authentication between clients and services, TLS-based secure communication, and authorization. Each user is supposed to have a X.509

certificate signed by a certificate authority (CA) of the virtual organization or by other third-party CA. The MINDS Grid front-end authenticates the user with the CA certificate.

### 3.3 Authorization: Access Control Framework

#### 3.3.1 Access Control models

Several access control models have been proposed for effective access control on resources, for example, Role Based Access Control (RBAC) [25] [26], Mandatory Access Control (MAC) [27], and Discretionary Access Control (DAC) [28] etc. In RBAC, roles are created for various operations on resources within an organization. Users are assigned one or more particular roles each of which is associated with one or more permissions to perform certain operations on resources. Since access to resources in RBAC is controlled not by the user identity but by his/her roles, user provisioning is simplified as a matter of assigning a set of appropriate roles to a specific user. MAC is an approach to control access to a certain sensitive resource by a user. A user is given certain security clearance properties and each resource is associated with specific security properties. In MAC, a user is allowed or denied to access a specific resource after the verification of the compatibility of the security properties of the security clearance properties of the user (or those of the proxy of the user) to the resource. DAC control access to resources based on the owner's (a user or group) identity which the resources belong to. Access control to a specific resource is regulated by the owner of the resource not by some administrator, that is, a user with certain access permission to a specific resource can give access to permission other users. None of these fully satisfies the requirements of the MINDS Grid system. The access control framework of MINDS Grid system needs a scheme to regulate users' activities as well as the access to data resources (summaries and alerts) and fields.

We developed two different access control schemes for MINDS-Grid authorization framework: Role-Privilege-Based Access Control (RPBAC) and MAC like Security-Clearance-Based Access Control (SCBAC). The RPBAC regulates the activities of the users while SCBAC regulates the access to data resource based on a set of security properties/policies. Figure 8 shows the ER diagram describing the relations between entities: User, Role, Privilege, and Alert. In RPBAC, roles are created for various operations on resources within an organization. Users are assigned one or more particular

roles each of which is associated with one or more permissions (privileges) to perform certain operations on resources. There is a non-zero many-to-many relationship between User and Role as well as between Role and Privilege. Every alert data has its owner but there may be a user (for example, a user as a Subscriber only) who does not own any alert data. There is 1-to-zero or more relationship between User and Alert as shown in figure 8. Each alert can be expressed as a vector of field elements. Each field is associated with a security classification level. The scheme SCBAC is applied when a user wants to access alert data. A user is given certain security clearance properties ($SL_{clearance}$) and each data resource is associated with specific security classification properties ($SL_{classification}$). In SCBAC, a user is allowed or denied to access a specific resource after the verification of the compatibility of the user.

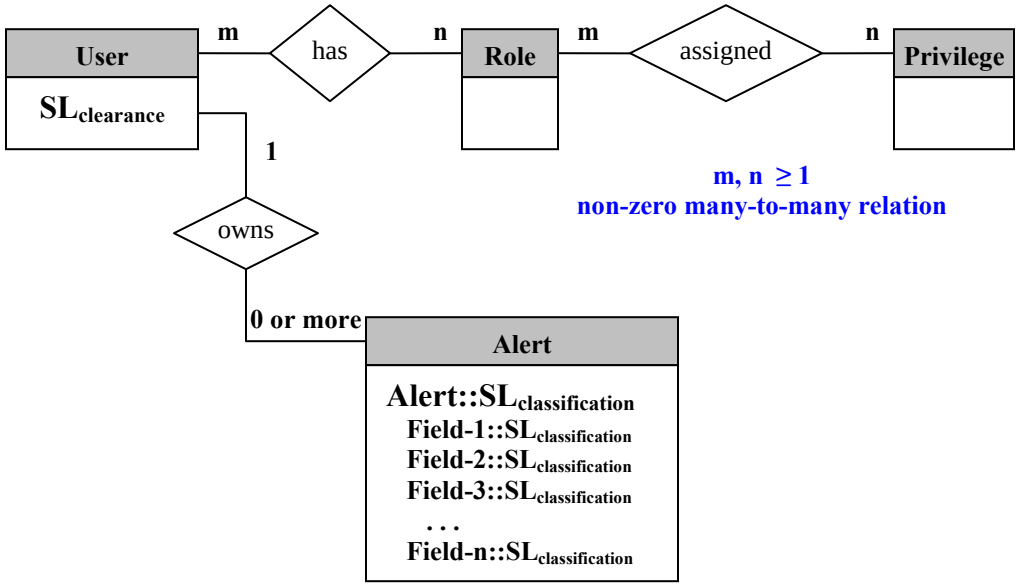



Figure 8: Entity relationship diagram for MINDS Grid access control framework

### 3.3.2 Mathematical Representation of Access Control Schemes

In this section, we mathematically present two access control models [RPBAC and SCBAC] discussed in section 3.3.1. We present notations and constraints followed by the access control mechanisms.

### 3.3.2.1 RPBAC access control scheme

**Notations and Definition**

i) $U = \{u_i\}$ : A set of users, where $|U| = k$ (cardinality of the set U)

ii) $R = \{r_i\}$ : A set of roles, $|R| = l$

iii) $P = \{p_i\}$: A set of privileges to access resources or services, $|P| = m$, where $p_i$ represents a permission to perform an atomic operational unit on a resource

iv) $o_i = \langle q_{i1}, q_{i2}, ..., q_{i,m-1}, q_{i,m} \rangle^T$, where $q_{ij} = \begin{cases} 1 \text{ if operation requires privilege } p_j \\ 0 \text{ otherwise} \end{cases}$

: a vector representation (transpose of $1 \times m$ vector) of a set of privileges required for user's request. For example, an analysis requires privileges to access back-end service and to retrieve the result

$\|o_i\|$: the number of non-zero elements of $o_i$, for example $\|\langle 1,0,0,1,1 \rangle\| = 3$

v) $AC = \{Allow, Deny\}$: Access control decisions

vi)Matrix representation of User-Role mapping

$$M = (m_{ij}) = \left[\begin{array}{cc|c|c} m_{11} & m_{12} & \cdots & m_{1l} \\ m_{21} & m_{22} & \cdots & m_{2l} \\ \hline \vdots & \vdots & & \vdots \\ \hline m_{k1} & m_{k2} & \cdots & m_{kl} \end{array}\right], \quad k \times l \text{ User-Role mapping matrix}$$

, where $m_{ij} = 1$ if $u_i$ is assigned $r_j$, 0 otherwise

vii) Matrix representation of Role-Privilege mapping

$$N = (n_{ij}) = \left[\begin{array}{cc|c|c} n_{11} & n_{12} & \cdots & n_{1m} \\ n_{21} & n_{22} & \cdots & n_{2m} \\ \hline \vdots & \vdots & & \vdots \\ \hline n_{l1} & n_{l2} & \cdots & n_{lm} \end{array}\right] : \ l \times m \text{ Role-Privilege mapping matrix}$$

, where $n_{ij} = 1$ if $r_i$ has the privilege $p_j$, 0 otherwise

viii) $f(A)$ : An element-wise function on a matrix $A = (a_{ij})$

$f(A) = A' = (a'_{ij})$, where $a'_{ij} = 1$ if $a_{ij} \neq 0$ and $a'_{ij} = 0$ otherwise

ix) $M_R = f(M \cdot N)$: $k \times m$ Reference matrix for RPBAC-based access control

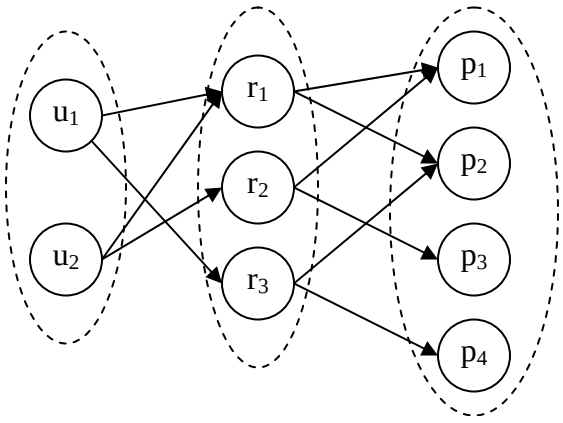
**Constraints**

i) $UR_{map} \subseteq U \times R$ : User-role mapping: non zero many-to-many relation

ii) $RP_{map} \subseteq R \times P$ : Role-privilege mapping: non zero many-to-many relation

**Access Control Decision**

The access control decision maker refers to a data structure containing User-Role-Privilege mappings. The User-Role-Privilege mapping can be represented as a compound bipartite graph or a reference matrix as shown in figure 9 (a) and (b). There is an equivalent reference matrix for each reference graph, but not vice versa. That is, there is a many-to-one relationship between graph representation and matrix representation. When certain operation on a resource or a service is invoked by a user, the user should have an appropriate role which is associated with a specific privilege to perform the operation. In order word, there should be a path from $u_i$ to $p_j$ in the graph representation of the compound mapping as shown figure 9 (a) or the value of (i,j) element of the reference matrix $M_R = f(M \cdot N)$ should be equal to 1. For example, from the example in figure 9, user $u_1$ has privilege $p_4$ but does not have privilege $p_3$.




(a)

$$\Leftrightarrow f(M \cdot N) = f\left(\begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

$M_R$

(b)

Figure 9: An example representation of RPBAC User-Role-Privilege mapping
(a) Graph representation (compound bipartite graph) (b) Matrix representation

Since user and requested operation can be expressed as a vector respectively, the RPBAC scheme based access control decision can be transformed into a matrix multiplication operation as follows:

*Request: user* $u_i$ *wants to perform a request consisting of a set of privilege* $o_j$

*Access control decision operator:*

*i) Access control decision on a privilege*

$$\psi(u_i, p_k) = \begin{cases} \textbf{Allow} \ \textit{if} \ e_i \cdot M_R \cdot e_k^{\ T} = 1 \\ \textbf{Deny} \ \textit{if} \ e_i \cdot M_R \cdot e_k^{\ T} = 0 \end{cases}$$

*, where* $e_i$ *is the i'th row of* $k \times k$ *identity matrix*

*and* $e_k^{\ T}$ *is the transpose of the k'th row of* $m \times m$ *identity matrix*

*ii) Access control decision on a set of privileges*

$$\psi(u_i, o_j) = \begin{cases} \textbf{Allow} \ \textit{if} \ e_i \cdot M_R \cdot o_j = \left\| o_j \right\| \\ \textbf{Deny} \ \textit{if} \ e_i \cdot M_R \cdot o_j < \left\| o_j \right\| \end{cases}$$

*, where* $e_i$ *is the i'th row of* $k \times k$ *identity matrix*

For example, let's suppose $u_2$ want to perform an operation $o_j = <1,0,1,1>^T$ that is the operation requires privileges: $p_1$, $p_2$, and $p_4$. The request will be denied because $\psi(u_1, o_j) = \begin{bmatrix} 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} = 2$ is not the same $\left\| o_j \right\| = 3$. If a user $u_2$ wants to perform an operational unit associated with the privilege $p_2$, this request will be allowed because the decision operator calculates 1, that is, $\psi(u_1, o_j) = \begin{bmatrix} 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = 1$.

**Considerations for matrix-based RPBAC**

The matrix representation of RPBAC simplifies the management of data structure for role based access control and access control decision making. Appendix A describes XML schema for User, Role, and Privilege. The schema defines the structure, elements, and data types of the metadata. User provisioning, policy management, and access control decision making depend on the metadata. Appendix B shows various activities related to user provisioning and policy management along with complexity. Since the number of users in the VO is far larger than that of R and P, that is $|U| >> |R|, |P|$, there is a possible scalability problem in user provisioning (add/remove). Since two mapping reference matrices (M and N) are maintained separately, the possible scalability problem can be resolved, in our access control model, simply by grouping users into multiple sub groups. Each group has its own User-Role reference matrix M but there is one global Role-Privilege reference matrix N. User grouping also reduces the cost of matrix operations: multiplication (complexity: $O(k \cdot l \cdot m)$) and element-wise operation $f(M_R)$ (complexity: $O(k \cdot m)$).

### 3.3.2.2 SCBAC access control scheme

**Notations and Definitions**

i) $U = \{u_i\}$ : A set of users, where $|U| = k$ (cardinality of the set U)

ii) $d_j = \langle f_{j1}, f_{j2}, ..., f_{j,n-1}, f_{j,n} \rangle$

Alert data: a vector consisting of one or more field elements,
,where $f_{jk}$ stands for k'th field of alert data $d_j$

iii) $SL = \{0,1,2,3,..N\}$ : Security Level: a linearly ordered set

$x \le y \Rightarrow SCL(x) \ge SCL(y)$ : level of security of x is higher than that of y
, where SCL stands for the security classification/clearance level and
$x, y \in SL$

iv) $AC = \{Allow, Deny\}$: Access control decisions

v) $\varphi(e) \in SL$ : A map of an entity e to a specific security level
where entity may be a user, an alert, or an alert field

vi) An entity-i, $e_i$ is said to ***clear*** the minimum security requirement of an entity-j, $e_j$ iff $SCL(e_i) \ge SCL(e_j)$, that is, iff $\varphi(e_i) \le \varphi(e_j)$

vii) Control decision function for the access request of entity-i to entity-j

$$\psi(e_i, e_j) = \begin{cases} \textbf{Allow} \text{ if } \varphi(e_i) \le \varphi(e_j) \\ \textbf{Deny} \text{ if } \varphi(e_i) \ge \varphi(e_j) \end{cases}$$

**Constraints**

i) $\varphi(d_i) \ge \max\{\varphi(f_{i,k}) \mid k = 1..n\}$: The security classification level of alert data itself should be larger than or equal to the maximum value of security classification levels of its fields.

**Access Control Decision**

When a user wants to access an alert data, the user should clear the minimum security requirement of the alert data and that of each field of the alert data. The SCBAC access control decision can be described as follows:

*Request: user* $u_i$ *wants to access an alert data* $d_j$

*Access control decision operator:*

$$\Psi(u_i, d_j) \in AC^{|d_j|}$$

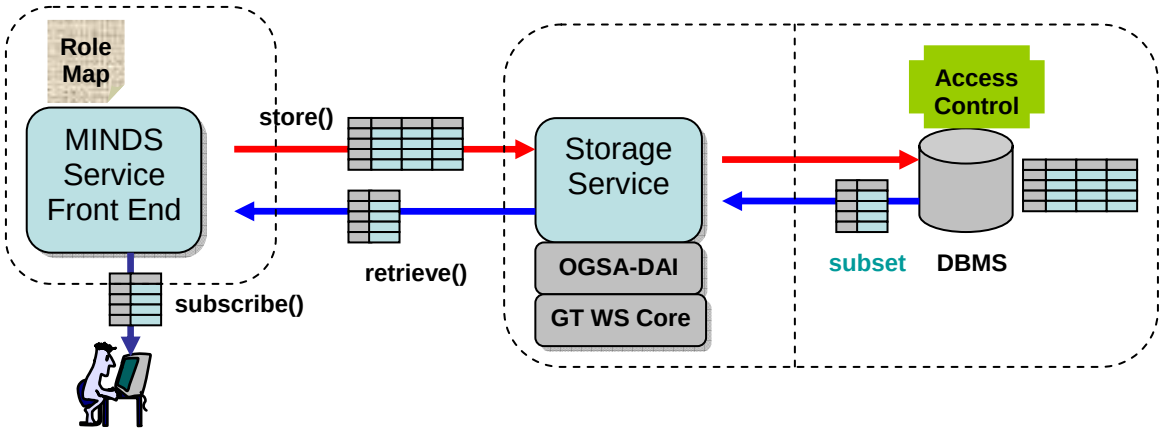
, where $\Psi(u_i, d_j) = < \psi(u_i, f_{j1}), \psi(u_i, f_{j2}), ..., \psi(u_i, f_{j,n}) >$

if $\psi(u_i, d_j)$ = Deny, then there is no need to perform piecewise access control decision operator for each field

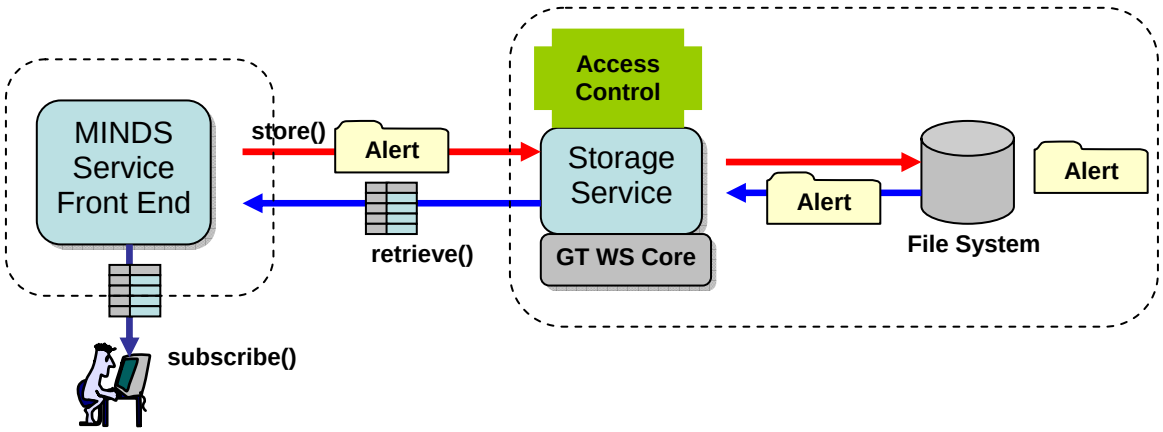
*Computational complexity:* $O(n)$

### 3.3.3 Access Control Architecture

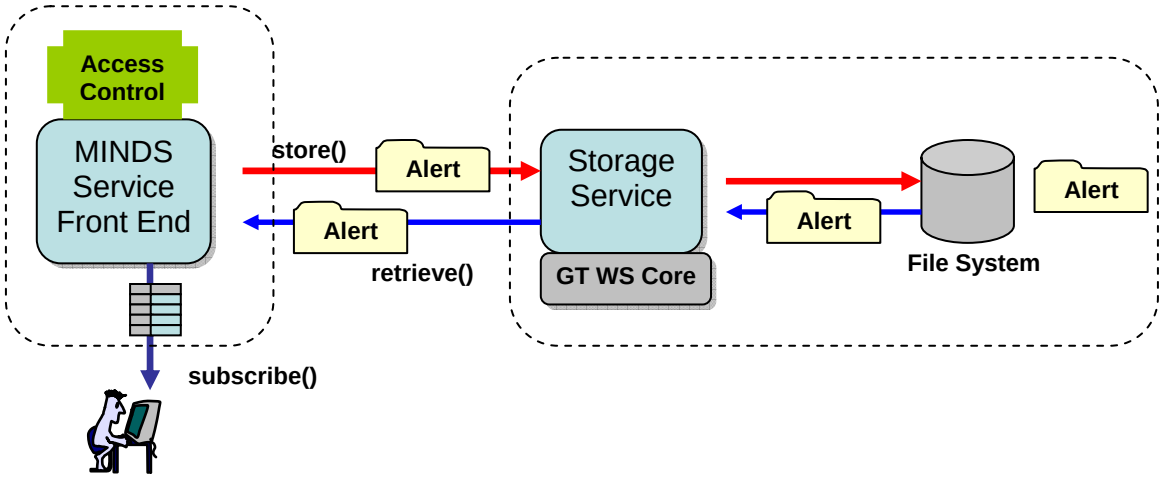
Figure 10 shows three different architectural alternatives considered for MINDS Grid access control framework: OGSA-DAI (Open Grid Services Architecture Data Access and Integration) [29] [30] based model and two file-system based models (centralized/decentralized). The main difference among three models is where the access control takes place. Table 1 describes the features of each model. The first model is based on OGSA-DAI. The OGSA-DAI enables access and integration of data across geographically distributed data sources through the Grid service interfaces for query, update, transform, and delivery of data. The data resources can be XML, relational database, or files. In this architecture, the alert data is stored in a database table.




(a) OGSA-DAI, DBMS based model




(b) File system based decentralized model




(c) File system based centralized model

Figure 10: Access control architectural alternatives: File system based centralized model

Each record in the alert file is stored in the table through the OGSA-DAI DataService interface in the first model. The main access control takes places in the DBMS level and leverages the built-in access privilege management system provided by DBMS. OGSA-DAI supports a simple role-based access control [25] on data stored in a relational DB, XML DB, or a file via role-mapping that maps individual Grid users to a database account which is given a set of privileges(SELECT, INSERT, UPDATE, DELETE, DROP etc) on databases, tables, and columns. Since each resource provider (in MINDS storage service) needs to maintain a role-map file, this model is not suitable for dynamic environments where users may join or leave dynamically and users' privileges may change frequently.

| Models | Features |
|---|---|
| OGSA-DAI Decentralized Model [ODDM] | • Decentralized access control<br>• Possible consistency maintenance problem with Authorization data<br>• Fine grained database level access control<br>• Limited control at MINDS service side<br>• Performance bottleneck (store operation)<br>• Not suitable for dynamic environments where users and resources are dynamic |
| File System based Decentralized model [FSDM] | • Decentralized access control<br>• Possible consistency maintenance problem with Authorization data<br>• Reduce unnecessary network bandwidth consumption (filtered alert) |
| File System based Centralized model [FSCM] | • Centralized access control<br>• No Consistency maintenance problem<br>• Possible network overhead |

Table 1: Comparison of three different architectural alternatives

The second model is based on the file system(FSDM). In this model, the alert data is stored as a whole in the local system securely. The access control in this model, takes place within the OGSA WSRF framework. Both ODDM and FSDM are a decentralized model in the sense that the access control decision takes place at the storage service side

and the authorization schema is managed in the storage service. Since the authorization schema is managed locally, there is a possible consistency maintenance problem.

On the other hand, the all access control decisions take place at the MINDS front-end side in FSCM. The storage service is not involved in any access control mechanism. Since alert file is transferred in its entirety without any filtering (or sub-setting), there is a possible network overhead.

# 4. Prototype Implementation & Testbed

## 4.1 Prototype Implementation

### MINDS Grid Service

We have developed a Java-based prototype of the MINDS grid service consisting of a front-end service and multiple back-end service using GT4.0. Through a user interface, user-custom configuration files and the location of input network flow data are submitted to the MINDS front-end service. The front-end sets up a plan and runs analysis in parallel on multiple back-end services. On completion of analysis, each back-end service returns the result to the front-end and the front-end aggregates the results and calls selected storage service to store the analysis result. The MINDS grid service is packaged as a GAR (Grid Application aRchive) to be deployed into Globus container or is packaged as a Web Application aRchive file (WAR) to be deployed into Tomcat service container.

### Storage Service

We implemented two different version of storage service to realize ODDM and FSDM. Storage service is a Grid service built on Globus Toolkit 4.0 support and running within a GT Java WS core container. Both version support two interfaces: storeAlert( ) and retrieveAlert( ). The ODDM-based storage service is developed as DataService exposing a relational database PostgreSQL as a data resource using OGSA-DAI toolkit and deployed in the container. FSDM-based storage service is implemented as a Grid service using GT4.0 API and deployed in the Globus Java WS core container.

### Realization of Security Consideration

For the secure sharing of summaries and alerts as characterization of potential attacks, our system ensures TLS-based encrypted communication (all SOAP messages are encrypted and transferred securely) and PKI-based mutual authentication/authorization. Services verify each other by exchanging certificates which are digitally signed by a trusted certificate authority (CA) in the organization. For restricted data access (query and subscription) within the data mining Grid community, we developed community security authority service which manages policies regarding three main principals in the community – users (service consumers such as analysts or

system administrators), services (MINDS service, Security service, and Storage service), and data resources (input flow data, alerts, and summaries). The security service regulates interactions between them.

**RPBAC and SCBAC in MINDS Grid system**

We defined three different roles in MINDS-Grid service for RPBAC: 1) Admin, 2) Analyst, and 3) Subscriber. Admin is the role for administrator of MINDS Grid system. The users with the role of Analyst can use MINDS analysis service and Storage Service when to store alerts. Subscribers can use the Storage Service to retrieve alerts they are subscribing.

| User | Role | Privilege (eligibility) | Security Clearance Level |
|---|---|---|---|
| /OU=UMN /CN=Kim | Admin | Super User | 1 |
| /OU=DTC /CN=Tom | Analyst, Subscriber | MINDS Analysis (using MINDS service)<br>Alert Subscriber (using Storage service) | 2 |
| /OU=PSU /CN=Amy | Subscriber | Alert Subscriber (using Storage service) | 4 |
| /OU=PSU /CN= Jon | Analyst | MINDS Analysis (using MINDS service)<br>Alert Subscriber (using Storage service) | 3 |
| /OU=PSU /CN=Peter | Subscriber | Alert Subscriber (using Storage service) | 2 |

Table2: An Example of User-Role-Privilege mapping

On the other hand, SCBAC scheme specifies the security clearance level for users, alerts, and alert fields. The security clearance level is on a scale of 1 to 5. Each user is given one ore more roles and a security clearance level as shown in table 2. Each alert file is associated with a security classification level and each field in the alert file is also associated with a security classification level. For example, the user with the distinguished name (/OU=DTC/CN=Tom) has two different roles (Analyst and Subscriber) and is given the security clearance level 2. This user can access alerts and alert fields of the security classification level lower than 2. The table 3 shows an example

of the alert catalogue. Each alert file has a globally unique identity in the community and is associated with the owner, security classification level, and the service handles of the storage service instances where the alert file is stored. An alert file may be replicated and stored on multiple storage services, then is associated with multiple service handles.

| Name | Owner | Security Classifica-tion Level | Location (Storage Grid Service Handle) |
|---|---|---|---|
| Alert-1 | /OU=UMN /CN=Kim | 1 | https://ss1.umn.edu:8443/wsrf/services/StorageService<br>https://ss3.umn.edu:8443/wsrf/services/StorageService |
| Alert-2 | /OU=DTC /CN=Jon | 3 | https://ss3.umn.edu:8443/wsrf/services/StorageService |
| | | | . . . . . |
| Alert-n | /OU=PSU /CN=Peter | 2 | https://ss3.umn.edu:8443/wsrf/services/StorageService |

Table 3: An example of Alert Catalogue

### 4.2 Testbed

We have built a testbed to deploy the prototype of the system we designed for the proof of concept. The testbed consists of several computing nodes. One Linux machine is used for deploying a MINDS service front-end and community security authority. As a fabric, Globus Toolkit 4.0 and a certificate authority (CA, simpleCA) have been installed. Three other Linux machines are used for deploying MINDS service back-ends. Apache Tomcat HTTP server has been installed on each machine and the Globus infrastructure is deployed as a Web application. Three windows machines (two laptops running Windows XP and one desktop running Windows server 2003) are used for deploying the Storage service. Globus Java WS (Web Service) core container is installed on each Windows machine and both ODDM-based storage service and FSDM-based storage service are deployed on the container.

# 5. Performance Evaluation

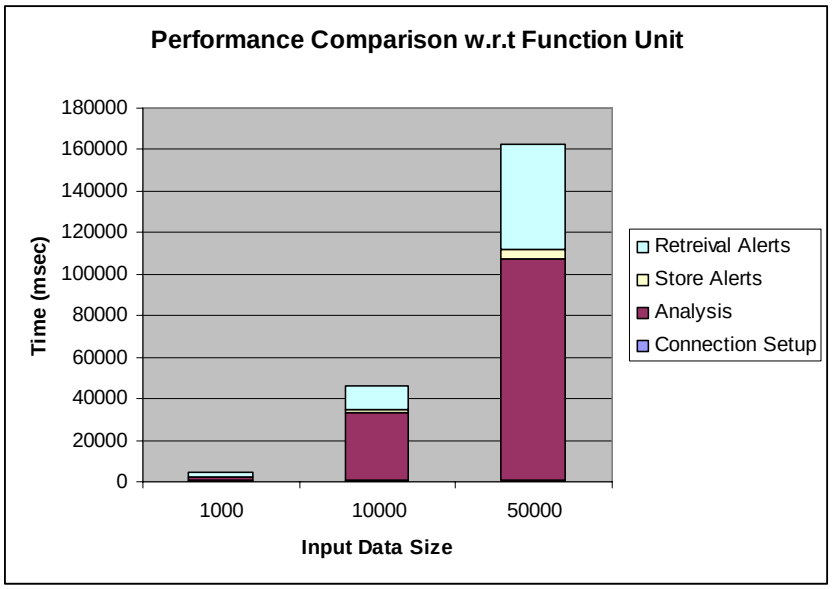
## 5.1 Experimental Setup

We measure performance of the system on a testbed built on four Linux systems (2x 652 MHz Intel Pentium III processor with 2GB memory running ubuntu 6.06) connected by 100MB Ethernet and a Windows system (1.8 GHz Intel Pentium Mobile processor with 1GB memory running Microsoft Windows XP) connected by Wireless LAN 802.11b. We deployed MINDS Grid service front-end on a Linux machine, three MINDS Grid service back-ends on other three Linux machines, and finally two different versions of storage services on a Windows machine. In all experiments, at least 10 trials are conducted and the measurements are averaged with error bars with a 95% confidence interval. The experiments consist of three parts. First of all, we measure the performance of each functional unit varying input data size. The functional units are 1) connection setup time, 2) application running time (MINDS analysis), 3) alert storing time, and 4) alert retrieval time. Second experiment is to evaluate the efficiency of distributed MINDS analysis. We use three different decomposition factors (1 to 3) in this experiment. The last experiment compares two different architectural alternatives for access control framework (ODDM and FSDM) described in section 3.3.3.

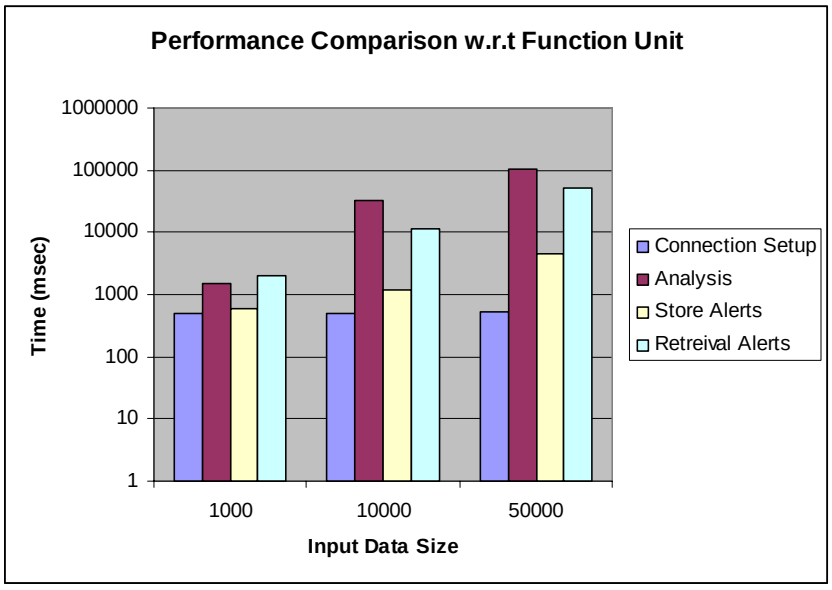
## 5.2 Performance evaluation of functional units

We decompose the whole workflow into multiple functional units and measure the time delay of each functional unit to understand which functional units are dominant. The decomposition factor is fixed as 1 and FSDM-based storage service is used in this experiment. With regard to input data, three different input data sets (1,000 records, 10,000 records, and 50,000 records) are used. As shown in figure 11 (a), the overall execution time linearly increases as the input data size increases. Figure 11 (b) shows detailed performance of each functional unit in a logarithmic scale. First of all, there is no difference in the connection setup time among different input size as expected. Secondly, MINDS analysis and Retrieval are dominant operations in the sense that the execution time of each operation increases sharply as the input size increases. On the other hand, the input size does not impact that much on the execution time of store operation. Hence,

we can see Analysis and Retrieval operations can be bottlenecks in the overall performance.




(a)




(b)

Figure 11: Performance comparison w.r.t Function Units

### 5.3 Efficiency of distributed MINDS analysis

The MINDS Grid service front-end decides the decomposition factor based on various system related status information. According to the decomposition factor, the input data

is decomposed into multiple sub datasets and sent to multiple MINDS Grid service back-ends for analysis. Figure 12 shows the relationship between the execution time and the decomposition factor. We used two different data sets: 10,000-record data and 50,000-record data. In overall, the decomposition factor affects the execution time but not in a dominant fashion. We observe that decomposition benefits are relatively large when the input data size is small. In case of the 10,000-record data set, the performance gain of decomposition is 70% and 80% for the decomposition factor 2 and 3 respectively. On the other hand, for 50,000-record data set, 20% and 25% performance gains are achieved. This result shows that the execution time significantly depends on the characteristics of the data mining scheme while it also depends on the input data size.

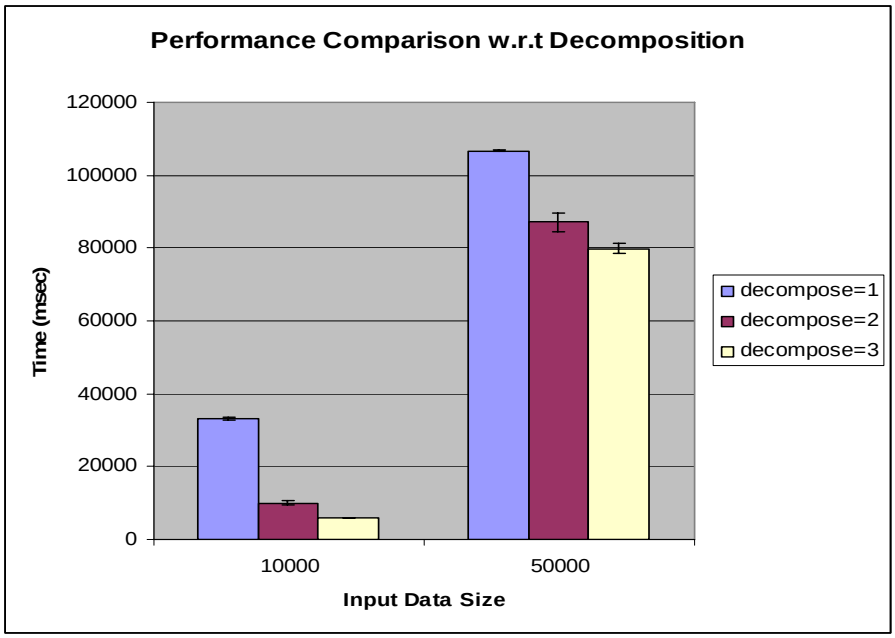



Figure 12: Performance comparison w.r.t Decomposition Factors

### 5.4 Comparison of architectural alternatives for access control

In this experiment, we compare two different architectural alternatives for access control: ODDM and FSDM. We used three sets of alerts with different size (10, 50, and 100 records) and measured the execution time of store and retrieval operation. Figure 13 shows the cost of store/retrieval operation in a logarithmic scale. For retrieval operation, there is slight difference (a factor of 3 to 4.5) between ODDM and FSDM. However, FSDM outperforms ODDM for store operation with the order of magnitude of 2 (by a

factor of 70 to 400). This is because the OGSA-DAI does not support bulk records store. In ODDM, each record is wrapped as a SOAP message and inserted into the database record by record basis. On the other hand, in FSDM, alerts are transferred and stored in its entirety in a single SOAP message.

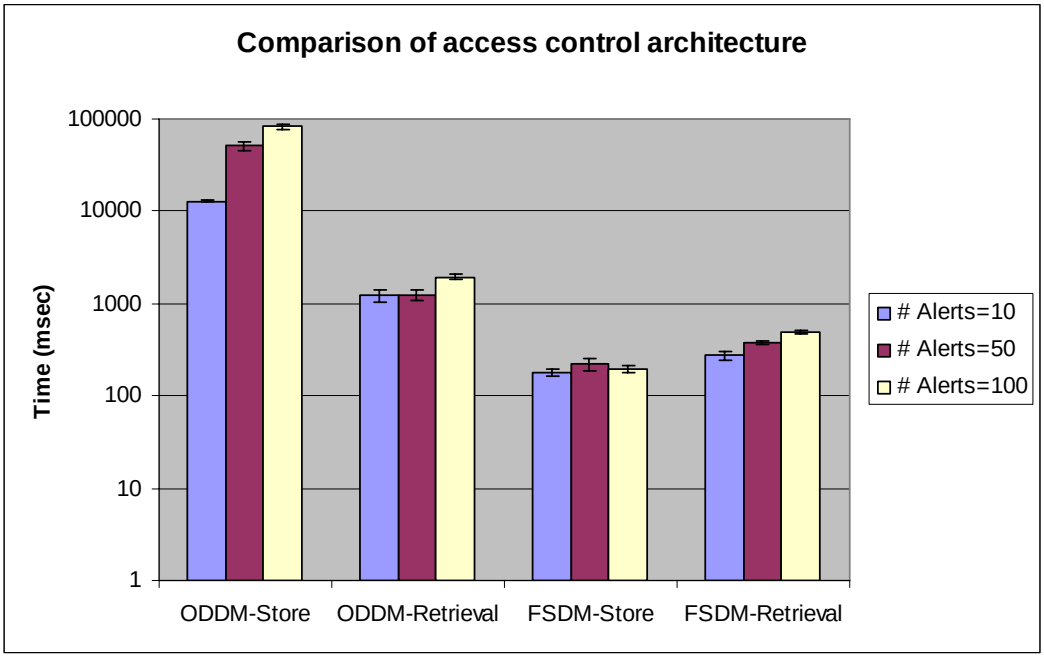



Figure 13: Performance comparison between Storage Service Schemes

## 6. Conclusion

We present a security enabled Grid system that supports distributed data mining, exploration and sharing. The system addresses issues pertaining to the three main requirements of distributed data mining on Grid: 1) exploiting of geographically and organizationally distributed computing resources to solve data-intensive data mining problem, 2) ensuring the security and privacy of sensitive data, 3) supporting seamless data/computing resource sharing. In order to address the first and third requirement, we design system architecture and present specifications of the component services which are built on a layered Grid system stack described in section 2.1. For the second requirement, we develop community security authority (CSA) which supports secure communication between entities, authentication, and authorized access control. We leveraged existing technologies such as TLS, PKI, and grid security infrastructure to support secure communication, user authentication, and mutual authentication between software clients and servers. Two access schemes - RPBAC and SCBAC are developed to effectively regulate various security related activities and access in MINDS Grid VO. For the proof of the concept, we present a prototype of MINDS Grid system along with some initial performance evaluation.

## Reference


[1] Ertoz, L., Eilertson, E., Lazarevic, A., Tan, P., Srivastava, J., Kumar, V., Dokas, P, *The MINDS - Minnesota Intrusion Detection System*, *"Next Generation Data Mining*, MIT Press, 2004".

[2] Dokas, P., Ertoz, L., Kumar, V., Lazarevic, A., Srivastava, J., Tan, P.: Data Mining for Network Intrusion Detection, Proc. NSF Workshop on Next Generation Data Mining, Baltimore, MD, November 2002

[3] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International J. Supercomputer Applications*, 15(3), 2001.

[4] I. Foster, C. Kesselman, J. Nick, S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," Open Grid Service Infrastructure WG, GGF, June 2002.

[5] OASIS Reference Model for Service Oriented Architecture 1.0, http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html

[6] Simple Object Access Protocol, http://www.w3.org/TR/soap/

[7] Universal Description, Discovery, & Integration, http://www.udd i.org/specification.html

[8] Web Services Definition Language, http://www.w3.org/TR/wsdl

[9] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," International Journal of Supercomputing Applications, 11(2), 1997

[10] The WS-Resource Framework, http://www.globus.org/wsrf/

[11] Web Services Resource Framework (WSRF) – Primer v1.2, http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-02.pdf, May 2006

[12] Globus GT4: www.globus.org, 2006

[13] Jon Weissman, Seonho Kim, and Darin England, A Framework for Dynamic Service Adaptation in the Grid: Next Generation Software Program Progress Report, NGS NSF Workshop, in conjunction with IPDPS 2005

[14] Jon Weissman, Seonho Kim, and Darin England, A Dynamic Grid Service Architecture, *IEEE International Symposium on Cluster Computing and the Grid (CCGrid2005)*, May, 2005, Cardiff, UK

[15] B. Lee and J.B. Weissman, ``Adaptive Resource Selection for Grid-Enabled Network Services", *2nd IEEE International Symposium on Network Computing and Applications*, April 2003.

[16] Seonho Kim and Jon Weissman, A Genetic Algorithm based Approach for Scheduling Decomposable Data Grid Applications, IEEE International Conference on Parallel Processing, August, 2004

[17] D. England and J.B. Weissman, "A Stochastic Control Model for the Deployment of Dynamic Grid Services," *5th IEEE/ACM International Workshop on Grid Computing* 2004.

[18] D. England and J.B. Weissman, "A Resource Leasing Policy for On-Demand Computing," invited to the I*nternational Journal of High Performance Computing and Application*s (IJHPCA) 2005.

[19] B. Clifford Neuman and Theodore Ts'o., Kerberos: An Authentication Service for Computer Networks, IEEE Communications, 32(9):33-38. September 1994

[20] Simple Authentication and Security Layer, Internet Engineering Task Force RFC-4222, http://www.ietf.org/rfc/rfc4422.txt

[21] Generic Security Services API, Internet Engineering Task Force RFC2743, http://www.ietf.org/rfc/rfc2743.txt

[22] The Transport Layer Security Protocol Version 1.0, IETF RFC 2246, http://www.ietf.org/rfc/rfc2246.txt

[23] The Secure Sockets Layer (SSL) 3.0 specification, http://wp.netscape.com/eng/ssl3/draft302.txt

[24] ITU-T Recommendation X.509: Information Technology – Open Systems Interconnection – The Directory: Authentication Framework

[25] R. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, Role-based access control models, IEEE Computer, Vol. 29, no. 2, pp. 38-47, 1996

[26] D. Ferraiolo and R. Kuhn, Role-Based Access Control, Proc. 15th National Computer Security Conference., 1992

[27] P. Loscocco, S. Smalley, P. Muckelbauer, R. Taylor, and J. Farrell., The Inevitability of Failure: The flawed assumption of security in modern computing environment., Proceedings of the 21st national Information Systems Security Conference, p303-314, 1998

[28] B. Lampson, Protection, in 5th Princeton Symposium on Information Sciences and Systems, pp. 437-443, 1971

[29] A.E. Arenas et al., The Design and Implementation of Grid Database Services in OGSA-DAI, Proc. UK e-Science All Hands Meeting, 2003

[30] The OGSA-DAI project, http://www.ogsadai.org.uk/

## Appendix A: XML schema for metadata for access control schemes

### **User.xsd: User metadata schema**

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="user" type="User"/>
  <xs:complexType name="User">
     <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="description" type="xs:string"/>
        <xs:element name="rpbac-index" type="xs:int"/>
        <xs:element name="security-clearance-level" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

The field *"rpbac-index"* matches to a row index of the reference matrix (the matrix M) of User-Role mapping. The field *"security-clearance-level"* is an integer value which specifies the user's security clearance level used in SCBAC access control scheme.

### **Role.xsd: Role metadata schema**

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="role" type="Role"/>
  <xs:complexType name="Role">
     <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="description" type="xs:string"/>
        <xs:element name="rpbac-index" type="xs:int"/>
     </xs:sequence>
  </xs:complexType>
</xs:schema>
```

The field *"rpbac-index"* matches to a column index of the reference matrix M or a row index of the reference matrix N of Role-Privilege mapping.

**Privilege.xsd: Privilege metadata schema**

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="user" type="User"/>
  <xs:complexType name="User">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="description" type="xs:string"/>
      <xs:element name="rpbac-index" type="xs:int"/>
      <xs:element name="assoication" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

The field *"rpbac-index"* matches to a column index of the reference matrix N. The field *"association"* specifies a permission which the privilege is associated with. It may be a path to an executable command, a path to a library, or a service handle to a Grid service

## Appendix B: User Provisioning and Control Policy Management

| Activity | | Description | Reference matrix to update | Action | Complexity |
|---|---|---|---|---|---|
| **User** related | ADD | Add a new user | M | Add a new row to M and add a new element to the user metadata | O(n) |
| | REMOVE | Remove a user | M | Remove a row from M and update the user metadata | O(n) |
| **Role** related | ADD | Add a new role | M and N | Add a new column to M, add a new row to N and update the role metadata | O(n) |
| | REMOVE | Remove a role | M and N | Remove a column from N, remove a row from N and | O(n) |
| | MODIFY | Modify privileges of an existing role | N | Modify values of the matching elements to the privileges | O(1) |
| **Privilege** related | ADD | Add a new privilege | N | Add a new column to N and update the privilege metadata | O(n) |
| | REMOVE | Remove an existing privilege | N | Remove a column from N and update the privilege metadata | O(n) |