



CRESCO

**Centro computazionale di RicErca sui Sistemi
COMplessi**

**Dipartimento di Ingegneria dell'Informazione e Matematica
Applicata**

Università di Salerno

Responsabile Unità Operativa

Prof. Ciro D'Apice

LA1-SP I.2 - Sviluppo ed integrazione dell'architettura GRID e di grafica 3 D

Attività I.2.1 Integrazione ENEA GRID con altre architetture di GRID computing

**1a - Analisi dell'infrastruttura GRID ENEA ed
individuazione dei punti critici ai fini
dell'interoperabilità e dell'estensione delle
caratteristiche di gestione e controllo
dell'infrastruttura stessa**

V 1.0

Indice

1. Contesto, obiettivi e scopo	3
2. La Rete ENEA	4
3. La GRID ENEA.....	6
3.1. Obiettivi della GRID ENEA	7
3.2. Architettura della GRID ENEA.....	8
3.2.1. Breve panoramica.....	9
3.2.2. Sottosistema di Accesso	10
3.2.3. File System Distribuito	13
3.2.3.1. La sicurezza in AFS.....	16
3.2.3.2. Il protocollo Kerberos.....	18
3.2.3.3. La cella ENEA.IT	29
3.2.4. Il Gestore delle Risorse LSF	30
3.2.5. Interazione fra LSF e AFS.....	32
3.3. Interoperabilità con altre GRID.....	33
4. Analisi delle criticità e possibili approcci risolutivi.....	35
4.1. Integrazione UNIX-Windows.....	35
4.1.1. Autenticazione	35
4.1.2. Autorizzazioni	35
4.2. Interoperabilità	36
5. Conclusioni	37

1. Contesto, obiettivi e scopo

Il Progetto CRESCO: Centro computazionale di RicErca sui Sistemi COmplessi ha come obiettivo la realizzazione, presso il Centro Ricerche ENEA di Portici (NA), di un importante Polo di calcolo multidisciplinare per lo studio dei sistemi complessi di natura biologica e tecnologica, collegato con *facilities* di elaborazione, archiviazione e visualizzazione 3D di dati ubicate presso i Centri ENEA di Brindisi e Trisaia (MT).

Sul piano delle applicazioni e dei contenuti scientifici, il Progetto si focalizza sulle seguenti linee d'attività:

- LA1. Realizzazione del Polo di Calcolo e sviluppo di nuove funzionalità di GRID Computing; l'implementazione di soluzioni innovative in tema di architetture di sistemi di calcolo e di GRID computing per le attività di R&S di punta dell'ENEA che richiedano l'utilizzo di risorse computazionali estremamente importanti. Sul versante infrastrutturale il Progetto prevede la realizzazione di una piattaforma di calcolo di assoluta eccellenza in grado di erogare una potenza dell'ordine del Teraflop, corredata di una nuova classe di funzionalità GRID innovative;
- LA2. Sviluppo di tecnologie e modelli computazionali per la descrizione di sistemi complessi di origine biologica e di materiali innovativi; lo studio di oggetti biologici dal punto di vista "sistemico" (in silico cell) e lo studio di sistemi naturali (comunità animali e sociali) secondo il paradigma dei sistemi complessi;
- LA3. Sviluppo di modelli di simulazione ed analisi delle Reti tecnologiche complesse e delle loro interdipendenze; lo studio di sistemi tecnologici complessi e delle loro mutue interazioni, e la realizzazione di opportuni strumenti per il modelling, la simulazione e il controllo.

Nell'ambito delle attività previste dalla Linea di Attività 1 il sottoprogetto identificato come SP I.2, *Sviluppo ed integrazione dell'architettura GRID e di grafica 3D*, prevede, fra l'altro, l'attività di integrazione dell'ENEA-GRID con altre architetture di *GRID computing* e nell'ambito di tale attività, nella quale il Dipartimento di Ingegneria dell'Informazione e Matematica Applicata dell'Università degli Studi di Salerno è coinvolto, è prevista, primariamente, un'analisi dell'infrastruttura ENEA-GRID con l'individuazione dei punti critici ai fini dell'interoperabilità e dell'estensione delle caratteristiche di gestione e controllo dell'infrastruttura stessa.

L'obiettivo che ci si pone è quindi quello di studiare la GRID implementata in ENEA, evidenziandone le caratteristiche salienti, gli eventuali punti di forza e di debolezza con lo scopo di incrementarne le potenzialità sia in termini generali sia, più specificatamente, per la parte gestionale amministrativa, non tralasciando quelli che possono essere gli aspetti essenziali nell'ottica dell'interazione con GRID di altra natura.

Il fine del presente documento è quindi quello di presentare i risultati derivanti dalla sopra descritta attività.

2. La Rete ENEA

L'ENEA, ha 12 centri di ricerca distribuiti sul territorio nazionale in 6 dei quali sono dislocate risorse computazionali di rilievo.

La rete ENEA che interconnette i Centri di Ricerca e le Sedi ENEA è denominata ENET ed è parte della Rete GARR Gruppo Armonizzazione Reti Ricerca ossia la rete per l'Università e la Ricerca Scientifica Italiana. A tale Rete sono connessi anche il [C.N.R.](#), l'[I.N.F.N.](#), le Università e altri Enti afferenti al mondo della ricerca (pubblica e privata)

I 6 centri che ospitano le risorse computazionali sono:

- Brindisi
- Bologna
- Casaccia
- Frascati
- Portici
- Trisaia

Tali centri ospitano anche i PoP Garr ossia i punti di accesso della WAN ENEA alla rete GARR; altri POP GARR sono dislocati presso i centri ENEA di

- Palermo
- Pisa
- Roma Sede

La seguente figura mostra la rete ENET



Figura 1: Topologia della Rete ENEA

I 6 centri computazionali ospitano quindi rilevanti risorse di elaborazione e sono connessi direttamente con link di rete ad alte prestazioni

3. La GRID ENEA

Il Grid Computing (letteralmente Calcolo a Griglia) è un paradigma di computazione distribuita (ideato dai fondatori del progetto Globus) nel quale tutte le risorse di un numero indeterminato di computer (sia risorse di calcolo, sia risorse di *storage*) geograficamente distribuite vengono inglobate – per mezzo di una tecnologia di rete standard – per essere trattate come un unico supercomputer, in maniera trasparente per gli utenti.

Il modello di computazione implementato per Grid ha come obiettivo quello di fornire i mezzi necessari per risolvere problemi computazionali complessi e troppo grandi per un unico supercomputer –come quelli di Protein Folding, modellazione climatica, ecc– mentre è anche usabile per problemi di tipo più semplice.

Questo paradigma si fonda su cinque principi:

1. **Condivisione di risorse:** L'idea di partenza di Grid è la esistenza e condivisione di differenti risorse – di differenti proprietari, con diverse politiche di accesso e con software eterogeneo – e la problematica che questo comporta.
2. **Sicurezza del sistema:** Bisogna definire una politica di accessi alle risorse, la autenticazione e la autorizzazione degli utenti.
3. **Uso efficiente delle risorse:** siccome le risorse sono disponibili per vari utenti è necessario definire un meccanismo per ripartire i job in modo automatico ed efficiente. Il software che gestisce questo aspetto è conosciuto come “middleware”.
4. **Reti di comunicazioni veloci ed affidabili** che rendono possibile la esistenza di Grid.
5. **Uso di standard aperti:** Come visto prima, il Grid nasce con la filosofia di condividere risorse. È per questo che l'uso di standard aperti comporta una migliore utilizzazione della struttura creata. Attualmente, gli standard per Grid provengono dal Globus Toolkit4.

In generale l'architettura dei sistemi GRID può essere rappresentata mediante una struttura a livelli

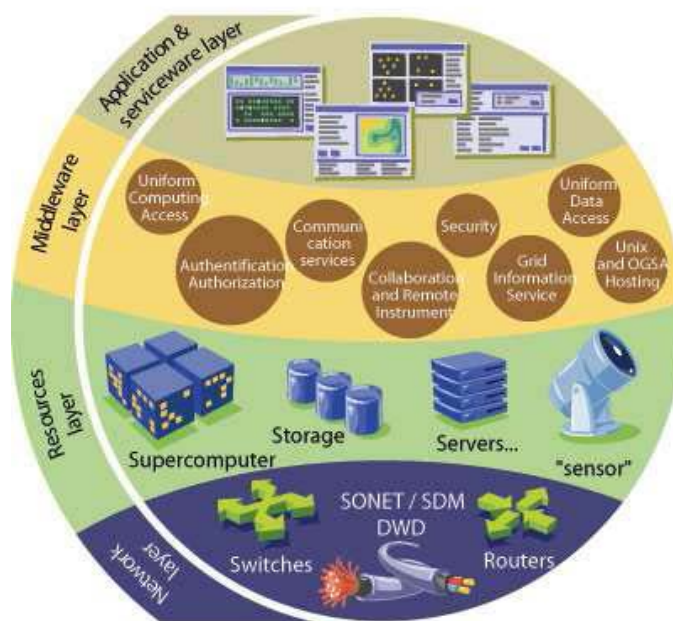


Figura 2: Livelli di un'architettura Grid. (Immagine: CERN)

Nella parte alta della architettura si incontra il livello applicativo, il software (che deve essere adattato al Grid) con il quale l'utente si interfaccia. Un livello più in basso, il livello di servizi (*middleware layer*) fornisce i tools per permettere che i differenti elementi –cioè servers, storage, ecc– vengano usati nell'ambiente Grid, essendo questo livello l'artefice di tutto il funzionamento del Grid. Sotto il livello di servizi vi è il livello di risorse, formato dalle differenti risorse disponibili nella griglia. Per finire, il livello di rete comprende la connettività fra tutte le risorse della Grid.

3.1. Obiettivi della GRID ENEA

ENEA GRID è un sistema finalizzato al conseguimento dei seguenti obiettivi principali:

- mettere a disposizione un sistema di produzione capace di offrire i servizi necessari al calcolo scientifico;
- integrare l'insieme di risorse informatiche di ENEA-INFO, distribuite su WAN;
- fornire un ambiente di lavoro unificato e metodi di accesso omogenei per tutti i ricercatori dell'ENEA indipendentemente dalla loro sede di lavoro;
- mettere a disposizione strumenti capaci di permettere l'integrazione delle risorse informatiche dipartimentali ed individuali e di fornire supporto alle attività sperimentali.

Tali obiettivi vengono conseguiti tramite l'erogazione di una serie di servizi che possono essere così catalogati:

Sistemi paralleli/seriali multiplatforma

AIX, Linux (Alpha,x86,x86_64,IA64), IRIX, Mac OS X, Solaris

Risorse 3D e di post-processamento grafico

SGI + 3D immersivo

Risorse Software:

Codici commerciali: Fluent, Gambit, Abacus, Catia, Ansys

Codici di "ricerca": mcnp/x, eranos, fluka,....

Ambienti di sviluppo & analisi: IDL, Matlab, Mathematica, SAS

Applicazioni Windows

In altre parole l'approccio basato sul concetto di griglia computazionale ha permesso di delineare un quadro unitario per l'insieme dei sistemi di calcolo di alte prestazioni disponibili all'interno dell'ENEA, al fine di ottimizzare il loro utilizzo all'interno di un ente distribuito sul tutto il territorio nazionale.

L'attività dell'ENEA infatti si svolge in 12 centri situati in Italia settentrionale, centrale e meridionale. Le sedi maggiori sono dotate ognuna di un centro di calcolo che gestisce le risorse installate localmente e che coprono esigenze di calcolo seriale, calcolo parallelo e richieste di grafica avanzata.

Il sistema ENEA GRID permette accedere all'insieme di tali risorse distribuite geograficamente, come ad un unico sistema virtuale, con una capacità integrata globale di varie centinaia di Gflops, distribuita su sistemi paralleli IBM SP, cluster Linux e macchine SGI.

3.2. Architettura della GRID ENEA

La scelta architetturale adottata nella realizzazione della GRID ENEA non è quella canonica basata su un *middleware* quale GLOBUS o g-Lite, adottato in diversi progetti GRID. In effetti alla base della GRID di ENEA ci sono tre componenti di base, sicuramente maturi, e che afferiscono sia ad ambiti proprietari che Open Source.

I tre componenti chiave sono:

File system distribuito: AFS/OpenAFS

Gestore delle risorse: LSF Multicluster

Interfaccia grafica: Java and Citrix Technologies

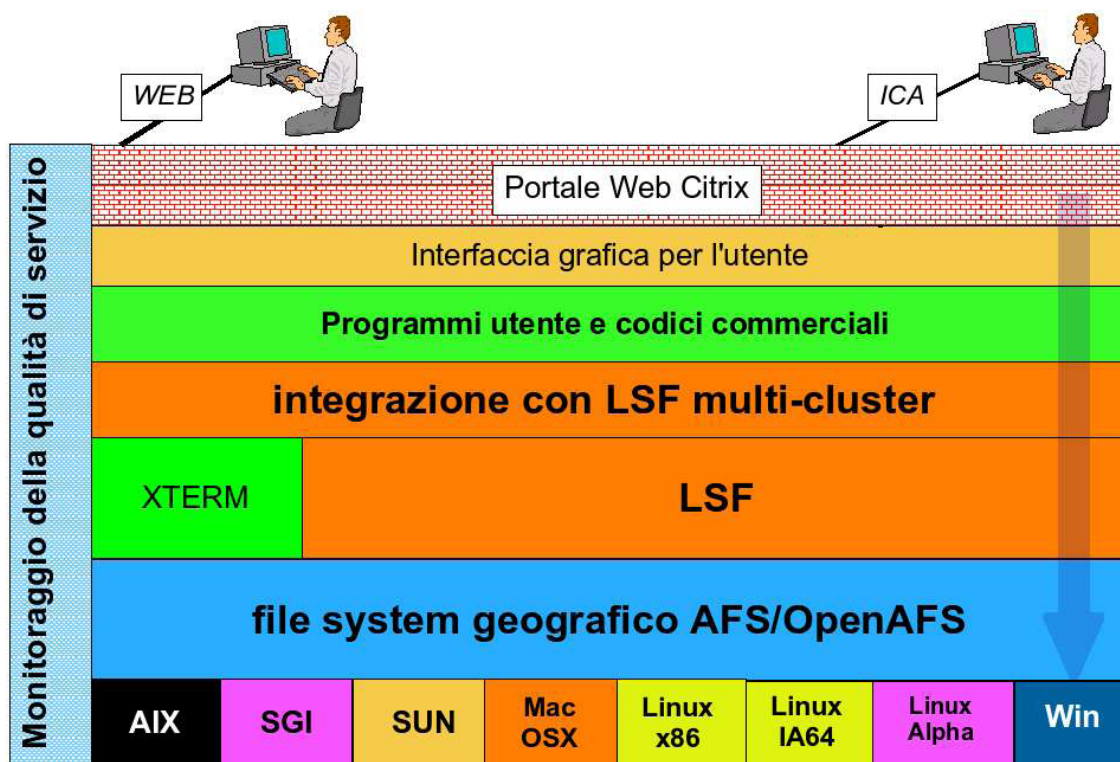


Figura 3: Architettura della GRID in ENEA

3.2.1. Breve panoramica

Per comprendere il funzionamento di massima della GRID ENEA, prima di passare ad una esamina più dettagliata, descriviamo brevemente le modalità operative che consentono la fruizione dei servizi erogati tramite la GRID.

E' possibile connettersi alla GRID ENEA con due principali modalità:

mediante Citrix Presentation Server Client;

mediante SSH (Secure SHell).

Citrix Presentation Server Client è un software, liberamente scaricabile dal sito del produttore ossia Citrix Systems¹ che consente, previa una opportuna configurazione, di collegarsi ad uno dei server Citrix della GRID ENEA ossia uno dei sistemi:

sp4-1.frascati.enea.it

pace.bologna.enea.it

¹ Citrix Systems, Inc. (Nasdaq:CTXS) è il leader globale nelle soluzioni infrastrutturali di accesso ed è il nome più noto nel settore dell'accesso enterprise. Quasi 60 milioni di persone in più di 180.000 organizzazioni in tutto il mondo usano Citrix ogni giorno.

dafne.casaccia.enea.it

triafs.trisaia.enea.it

kleos.portici.enea.it

ercules.brindisi.enea.it

che, come si nota sono dislocati in ognuna delle sedi dell'ENEA nelle quali risultano presenti le risorse computazionali sfruttabili tramite la GRID.

Tali Server Citrix ospitano il prodotto **Citrix Presentation Server** (precedentemente denominato Citrix MetaFrame Server) che è un prodotto dedicato all'accesso remoto e alla pubblicazione di applicazioni che consente agli utenti di collegarsi alle applicazioni disponibili sui server centrali.

Dopo aver effettuato l'accesso, l'utente ha la possibilità di eseguire una serie di applicazioni disponibili sui vari sistemi della GRID avendo la possibilità di accedere in ogni caso ai propri dati in quanto tutti i sistemi vedranno tramite il file system distribuito AFS (**Andrew File System**) la directory dell'utente stesso. Ciò significa che per l'utente risulta trasparente la scelta dell'host sul quale girerà il suo processo. In effetti la selezione dell'host sul quale verrà eseguito il servizio/applicazione richiesto dall'utente avverrà mediante il prodotto di gestione delle risorse **Platform LSF** che sulla base di opportune politiche (ad esempio disponibilità delle risorse, carico dei sistemi, ecc) stabilirà l'host più idoneo ed avvierà su di esso il processo.

3.2.2. Sottosistema di Accesso

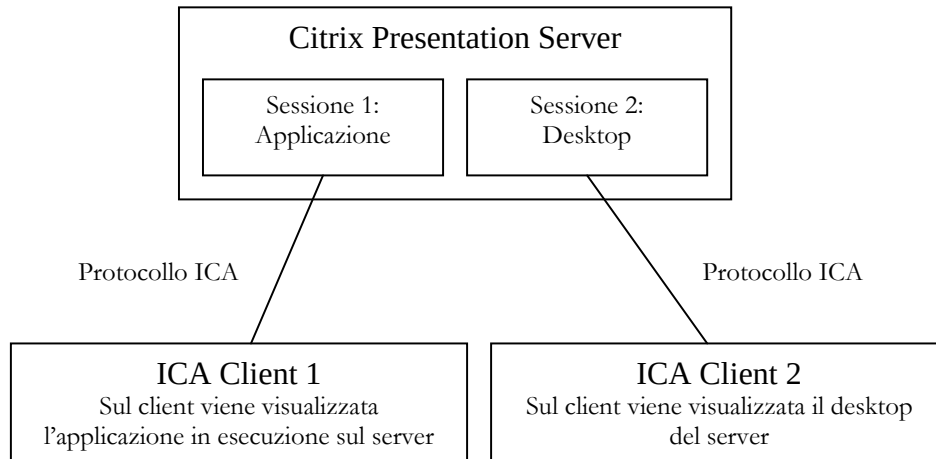
Sui cosiddetti Citrix Server viene dunque installato il prodotto Citrix Presentation Server e nel caso specifico di ENEA tali sistemi sono dei sistemi UNIX. I client attraverso il prodotto Citrix Presentation Server Client anche detto ICA Client (Independent Computing Architecture²) possono collegarsi ai server Citrix. I client ICA sono distribuiti free per le diverse piattaforme (Windows, UNIX, LINUX, MAC).

Il processo di autenticazione impiegato in fase di collegamento è basato sulle autorizzazioni utilizzate dal sistema operativo dell'host e quindi gli utenti del sistema potranno tranquillamente connettersi senza necessità di dover creare gli utenti. Tramite l'ICA client sarà quindi possibile lanciare le applicazioni distribuite tramite il sistema o una sorta di desktop remoto sul server. Le applicazioni che possono essere distribuite sono di varia natura ed ovviamente verranno eseguite sul server con un carico minimo per il client in quanto il protocollo ICA prevede solo l'invio di *keystrokes*, mouse clicks e

² Independent Computing Architecture (ICA) è un protocollo proprietario per sistemi di tipo application server, progettato da Citrix Systems. Il protocollo stabilisce una specifica per trasferire i dati fra il client ed il server ma non è legato a nessuna piattaforma specifica. Ciò consente ad esempio che applicazioni Windows ordinarie siano eseguite su un server Windows e che i client supportati possano accedere a tali applicazioni; tali client non dovranno eseguire necessariamente Windows ma potrebbero essere Mac, Unix, Linux. ICA is broadly similar in purpose to window servers such as the X Window System. It also provides for the feedback of user input from the client to the server, and a variety of means for the server to send graphical output from the running application to the client.

screen updates tra il client e il server. Le applicazioni da distribuire dovranno essere specificate tramite la configurazione del Citrix Server.

Quindi la situazione può essere schematizzata come segue:



Nel caso di ENEA l'applicazione che viene distribuita tramite il Citrix Presentation Server (forse sarebbe meglio dire i Citrix Presentation Server) è la cosiddetta Interfaccia ENEA ossia una particolare configurazione del prodotto della NISE EnginFrame che, essenzialmente, permette di accedere alle risorse di calcolo da una semplice interfaccia java. Il prodotto EnginFrame è stato installato sul file system geografico AFS, di cui si parlerà dettagliatamente in seguito, al fine di poter distribuire agevolmente l'Interfaccia ENEA in maniera univoca attraverso tutti i Server Citrix.

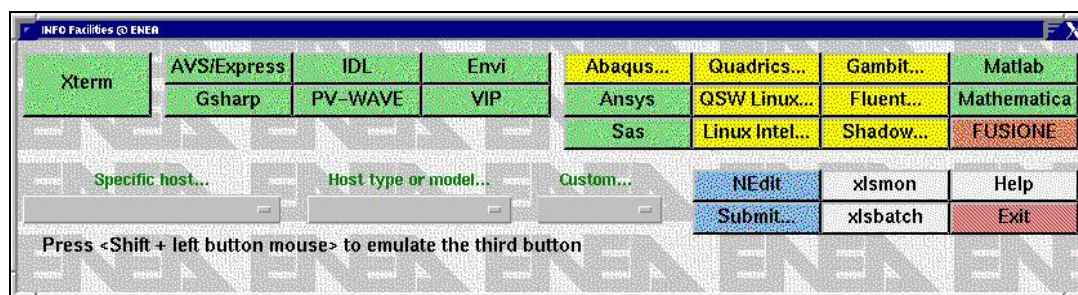


Figura 4: L'Interfaccia ENEA distribuita tramite Citrix Presentation Server

L'accesso alla GRID ENEA può anche essere realizzato mediante il protocollo SSH (Secure Shell) un protocollo di comunicazione nato per rimpiazzare i comandi Berkeley³ ed in grado di sostituire anche le, in maniera sicura, anche le funzionalità dei sopra citati telnet ed ftp. Esso, concepito ed inizialmente sviluppato dal finlandese Tatu Ylonen nel 1995, consente delle autenticazioni forti fra i vari host e connessioni crittografate.

Esistono due versioni del protocollo SSH. La prima (SSH1) mostra le seguenti caratteristiche generali:

- sistema di codifica ed autenticazione RSA con chiavi da 768 a 2048 bit;

- possibilità di compressione della comunicazione;

- utilizzo di uno dei seguenti algoritmi di cifratura:

- IDEA (chiave a 128 bit)

- 3DES (chiave a 168bit)

- Blowfish (chiave a 128bit)

La seconda (SSH2) è una completa riscrittura della vecchia versione, è più sicura, fornisce controlli di integrità, fa uso dell'algoritmo Diffie-Hellman per lo scambio delle chiavi tra client e server, di nuovi algoritmi pubblici come il DSA, ha una implementazione sicura di tutta la suite di comandi ftp.

Quindi mediante un qualsiasi client SSH ad esempio PuTTY, sarà possibile collegarsi ad uno dei server ENEA ed accedere da remoto al corrispondente sistema UNIX avendo a disposizione sempre la stessa home directory (grazie al file system distribuito) e tramite LSF avere la possibilità di sottoporre i job sulle macchine del cluster.

³ All'università di Berkeley furono sviluppati un insieme di comandi detti r-utilities ove la r indicava remote, giacché tali comandi consentivano l'esecuzione di azioni su host remoti. La fase di autenticazione dell'utente è basata sul file rhosts, presente nella home directory dell'utente sul sistema remoto, che contiene un elenco di utenti e degli host su cui essi risiedono ed indica che essi possono accedere al sistema senza specificare alcuna password. Tale protocollo è fondamentalmente insicuro in quanto basato su tali relazioni di fiducia.

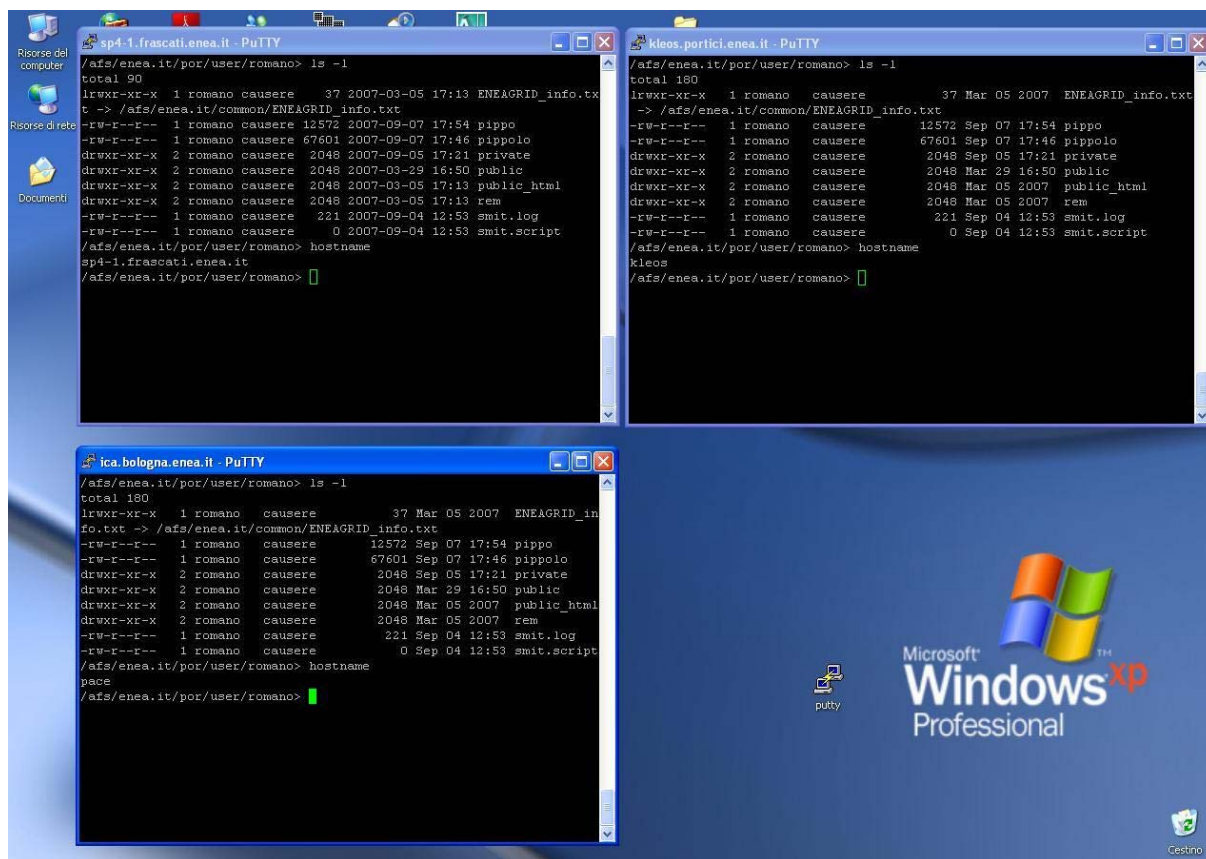


Figura 5: Accesso a vari server della GRID ENEA da macchina Windows mediante PuTTY (SSH)

3.2.3. File System Distribuito

AFS (Andrew File System) è un file system distribuito che permette l'accesso a files o directory residenti su macchine connesse in rete locale e geografica.

La struttura e i meccanismi di indirizzamento di AFS consentono la creazione di file system distribuiti su un gran numero di nodi che possono risiedere in siti geografici diversi.

AFS fu originariamente sviluppato alla Carnegie Mellon University nell'ambito dell'Andrew Project⁴ e deve il suo nome ad Andrew Carnegie ed Andrew Mellon. Fu in seguito sviluppato come prodotto commerciale dalla Transarc Corporation, successivamente acquisita dall'IBM. Nel 2000 l'IBM annunciò l'intenzione di rilasciare una versione del prodotto commerciale AFS sotto la IBM Public License e ciò diventò OpenAFS. Attualmente OpenAFS è attivamente sviluppato per un grande range di famiglie di sistemi operative sia UNIX che Windows

⁴ The Andrew Project was a distributed computing environment begun in 1983, driven by the Information Technology Center, a joint Carnegie Mellon University and IBM project.

AFS si basa sul modello client/server; i dati sono memorizzati su una o più macchine server e vengono trasferiti alle macchine client (cached) soltanto quando vengono richiesti. Da questo momento in poi, tutte le modifiche sono eseguite sulla copia locale e sono riportate sul server solo alla chiusura del file.

Cache Manager

La macchina client è un comune computer che ospita un software client che si occupa della connessione al server. Su di essa è attivo il demone *Cache Manager* che copia i dati dal server alla macchina locale e, dopo l'eventuale modifica da parte dell'utente locale, li ritrasmette al server che li aggiorna nel proprio file system al fine di non saturare la rete con continue richieste ma soprattutto per evitare l'interruzione del servizio nel caso di crash imprevisti dei server e per rendere più veloci le operazioni. Il demone inoltre gestisce l'accesso, la visualizzazione, e la modifica dei dati dalla directory principale */afs* che rappresenta la radice di una struttura organizzata ad albero.

Celle

Al secondo livello della struttura vi sono le celle, il cui nome, solitamente, viene assegnato in base al nome del loro dominio (ad esempio */afs/enea.it*). Una cella è un dominio amministrativo nello spazio AFS, ognuna di esse viene amministrata in modo autonomo da chi la gestisce che può quindi assegnare la quantità di spazio agli utenti a proprio piacimento. Ogni cella possiede il proprio File Space e, con le dovute autorizzazioni, può accedere al File Space delle altre celle.

Token

L'autorizzazione avviene mediante l'assegnazione di un token all'atto dell'accesso. Grazie ad esso il sistema sa che l'utente in questione è un utente autenticato. Se un utente non dovesse avere il token, egli viene riconosciuto dal sistema come un utente anonimo con la conseguenza che i suoi permessi saranno ristretti ai permessi del gruppo *anyuser*. Un caso del genere si può verificare quando il token assegnatogli in precedenza scade, visto che questo identificativo ha una durata limitata nel tempo, che va da 25 ore (valore di default) a trenta giorni, a discrezione dell'amministratore del sistema. Per prevenire questo problema esiste il comando *tokens* che mostra la data di scadenza del token, e il comando *klog*, che serve per rinnovarlo ed evitare quindi che il token scada durante una sessione in cui è stato lanciato un job molto grande.

Volumi

AFS suddivide in le partizioni in sezioni chiamate volumi. Ogni cartella home si trova in volume AFS diverso. Ogni volume inoltre ha uno spazio limitato, che viene definito quota. L'amministratore del sistema assegna una quota agli utenti che possono verificarla tramite il comando *fs listquota*; inoltre con questo comando può conoscere il nome del volume che ospita un determinato file o directory, la sua dimensione, la percentuale di quota usata.

Nel caso in cui un utente dovesse superare il limite di spazio consentito, il sistema restituirà un errore.

L'accesso ad un volume avviene attraverso un mount point che punta al server e alla partizione a cui il volume appartiene. Un mount point appare all'utente come una directory; per spostarsi si utilizzano i comuni comandi di UNIX. Per conoscere l'indirizzo fisico di un file è sufficiente impartire il comando `fs whereis [-path ]`.

Access Control List

Una fondamentale caratteristica di AFS è la sicurezza poiché deve seguire uno schema di autenticazione molto preciso e un controllo degli accessi molto rigido grazie alla Access Control List, una tabella associata ad ogni directory che definisce i permessi per ogni gruppo o utente che accede a questi dati. L'intera gestione delle autorizzazioni è assegnato al Kerberos Authentication Server.

AFS è caratterizzato inoltre da due tipi di comando, quelli per il file system (fs), che servono per avere, come detto prima, informazioni sulle quote, sulle celle, sulla cache ecc; poi ci sono quelli per la gestione delle directory (pts): possiamo avere informazioni sui gruppi, sugli utenti, sui permessi dei file e cartelle, ecc.

A differenza di UNIX, che possiede solo tre tipi di permessi da controllare, AFS ne ha ben sette:

1. Lookup (l): permette all'utente di eseguire il comando `ls` per visualizzare il listato della directory corrente e di esaminare il suo ACL;
2. Insert (i): permette all'utente di creare nuovi file e sottodirectory nella cartella corrente;
3. Delete (d): permette all'utente di cancellare file e sottodirectory dalla directory corrente;
4. Administer (a): permette all'utente di cambiare l'ACL.
5. Read (r): permette all'utente di leggere e copiare i contenuti dei file;
6. Write (w): permette all'utente di modificare i contenuti dei file e cambiare i loro permessi con il comando `chmod`;
7. Lock (k): permette all'utente di effettuare il lock del file.

Ovviamente ogni utente, per poter accedere ad una directory, deve avere almeno il permesso lookup.

Oltre al normale accesso ai file, un utente che ne ha diritto (permessi lookup e read), può anche eseguire un programma, ovviamente a patto che esso sia presente nella propria directory di AFS. Come avviene per i file, il Cache Manager copia il programma sul disco locale e poi lo esegue localmente.

3.2.3.1. La sicurezza in AFS

Come per ogni file system, anche per AFS, la sicurezza è un fattore fondamentale e quindi AFS incorpora diverse features che evitano che utenti non autorizzati possano accedere ai dati. Peraltro in un ambiente di rete la tecnica più efficiente per evitare il furto di informazioni è quella di accertarsi dell'identità dell'altro. Quindi la natura della rete costringe tutti gli attori presenti sulla rete stessa a considerare che gli altri attori coinvolti in una comunicazione non siano effettivamente loro fino a che non riescano a provarlo. Il meccanismo con il quale le parti provano la loro effettiva identità è detto *Mutual Authentication* la cui implementazione è abbastanza complessa anche se l'idea di base sottostante è abbastanza semplice basandosi sul fatto che le parti provano la loro identità mostrando la conoscenza di uno *shared secret*.

Uno *shared secret* è un pezzo di informazione conosciuto solo dalle parti che si stanno mutuamente autenticando e che potrebbero venirne a conoscenza per la prima volta da una terza parte che le due parti considerano ambedue fidata.

Quindi la parte che avvia la transazione presenta lo *shared secret* e rifiuta di accettarlo l'altra parte fino a che non dimostra che la stessa conosce anche essa il segreto

La forma più comune di *shared secret* nelle transazioni AFS è la chiave di cifratura spesso riferita semplicemente come chiave. Le due parti impiegano la chiave condivisa per cifrare i pacchetti che inviano e per decifrare i pacchetti che ricevono. La cifratura raggiunge quindi due scopi. In primis protegge i furti di informazioni durante il transito sulla rete e secondo assicura sull'identità dell'altra parte che, essendo in grado di decifrare i messaggi dimostra la conoscenza della chiave e quindi dello *shared secret* ed in conclusione dimostra la sua identità

Simple Mutual Authentication

Tale tipo di autenticazione coinvolge solo una chiave di cifratura e due parti generalmente un client ed un server. Il client contatta il server inviando un challenge message cifrati con la chiave noto solo a loro due. Il server decifra il messaggio usando la sua chiave che è la stessa del client se le due parti condividono lo stesso segreto. Poi il server risponde al challenge ed usa la sua chiave per cifrare la sua risposta. Il client usa la sua chiave per decifrare la risposta del server e se la risposta è corretta allora il client è sicuro dell'identità del server perché solo qualcuno che conosce la stessa chiave può decifrare il challenge e rispondere correttamente. D'altra parte, il server conclude che anche l'identità del server è sicura in quanto il challenge message decodificato aveva senso dopo la decodifica. AFS impiega la simple mutual authentication per verificare l'identità dell'utente durante la prima fase della procedura di login. In tal caso, la chiave è basata sulla password dell'utente

Complex Mutual Authentication

La *Complex Mutual Authentication* coinvolge tre chiavi di cifratura e tre parti. Tutte le transazioni sicure AFS (eccetto la prima fase relativa al processo di login) impiegano tale autenticazione

Quando un *client* desidera comunicare con un server, per prima cosa contatta una terza parte denominata *ticket-granter*; *ticket-granter* e *client* si autenticano mutuamente impiegando la *Simple Mutual Authentication*. Quando hanno finito, il *ticket granter* dà al *client* un *server ticket* o semplicemente *ticket* come prova che esso, ossia il *ticket granter*, ha preverificato l'identità del *client*. Il *ticket granter* cifra il ticket con la prima di 3 chiavi chiamata *server encryption key* perché è nota solo al *ticket granter* ed al server al quale il client si vuole collegare. Il client non conosce questa chiave. Il *ticket granter* invia anche altri pezzi di informazione insieme al ticket che mettono in condizione il client di usare il ticket sebbene il client stesso non sia in grado di decifrarlo. Insieme con il ticket, questi items costituiscono il **token**

- Una *session key*, che è la seconda chiave di cifratura coinvolta in tale autenticazione. Il *ticket-granter* crea tale chiave at random, come il shared secret fra il client ed il server. Per motivi esposti in seguito, il ticket granter mette una copia della session key all'interno del ticket. Il client ed il server impiegano la session key per cifrare i messaggi che si spediscono l'uno all'altro durante la comunicazione. Il *ticket-granter* inventa una differente chiave di sessione per ogni connessione fra client e server (ci possono essere diverse transazioni durante una singola connessione).
- Il nome del server per il quale il ticket è valido (e di conseguenza quale server encryption key cifrerà il ticket stesso).
- Un indicatore del tempo di vita del ticket. Il tempo di vita predefinito dei server AFS per i ticket è 100 ore. Se il client desidera contattare il server nuovamente dopo che il ticket è scaduto, deve contattare il ticket granter per averne uno nuovo.

Il *ticket-granter* cifra l'intero token con la terza chiave di cifratura coinvolta nell'autenticazione – chiave nota solo al ticket-granter ed al client. In qualche caso tale chiave è derivata dalla password dell'utente.

Ora che il client ha un ticket valido, è pronto per contattare il server, spedendogli due cose:

- Il ticket cifrato con la *server encryption key*.
- La sua richiesta cifrata con la *session key*.

A questo punto, il server non conosce la *session key*, in quanto è stata appena creata dal *ticket-granter*. Comunque, il ticket-granter ha messo, come detto in precedenza, una copia della *session key* all'interno del ticket. Il server usa la server encryption key per decifrare il ticket ed imparare la session key che poi usa per decifrare il messaggio di richiesta del client. Genera la risposta e la trasmette al client cifrandola con sempre la session key.

Tale step costituisce il cuore della *mutual authentication* fra client e server perché prova ad ambo le parti che essi condividono lo stesso segreto:

Il server conclude che il client è autorizzato ad inoltrargli la richiesta in quanto la richiesta stessa ha senso quando la decifra usando la session key. Se il client impiegasse una differente *session key* da quella che il server ha trovato nel ticket, allora la richiesta risulterebbe incomprensibile dopo la decifratura. Le due

copie della session key (quella all'interno del ticket e quella usata dal client) possono essere uguali solo se coincidono con quelle generate dal server. Il client non può fingere la conoscenza della session key perchè non può guardare all'interno del ticket, cifrato con la server encryption key nota solo al server ed al *ticket-granter*. Il server confida sul fatto che il *ticket-granter* da i token solo ai client che il *ticket-granter* stesso ha autenticato e quindi il server decide che il client è legittimo. Nota che non c'è nessuna comunicazione diretta fra il *ticket-granter* ed il server, sebbene il loro rapporto sia centrale nell'autenticazione basata su ticket. Essi interagiscono solo indirettamente attraverso il possesso da parte del client di un ticket spedito e comprensivo del loro shared secret.

Il client conclude che il server è autentico e si fida delle risposta che gli arriva dal server in quanto essa ha senso dopo che la stessa è stata decifrata impiegando la session key. Ciò infatti indica che il server ha cifrato la risposta con la stessa chiave di sessione conosciuta dal client e l'unica situazione affinché ciò sia possibile è che il server ha dovuto prima decifrare il ticket. Ciò è ovviamente possibile solo se il server stesso condivide con il ticket granter il segreto della *server encryption key*. Il client si fida che il ticket granter fornisca ticket solo per i legittimi server, e quindi considera autentico un server che è in grado di decifrare il ticket e in conclusione accetta le sue risposte

In effetti l'autenticazione impiegata da AFS fu sviluppata a partire dai documenti di Kerberos versione 4, ma prima che il protocollo venisse formalizzato⁵. Risulta comunque possibile impiegare il Kerberos standard invece di quello di AFS sia nella versione 4 che nella versione 5.

In ogni caso visto la stretta interazione fra AFS e Kerberos si è ritenuto opportuno approfondire lo studio su tale protocollo.

3.2.3.2. Il protocollo Kerberos

Kerberos è un protocollo di autenticazione per host fidati su reti non di fiducia i cui principali obiettivi sono:

- La password dell'utente non deve mai viaggiare sulla rete;
- La password dell'utente non deve mai essere memorizzata in nessuna forma sulla macchina client: essa dopo essere utilizzata deve essere subito scartata;
- Le password dell'utente non dovrebbero essere memorizzate in chiaro neanche nel database dei server di autenticazione;
- All'utente è richiesto di inserire la password una sola volta per sessione di lavoro. Potrà pertanto accedere, durante detta sessione, a tutti i servizi per il quale è autorizzato, in modo trasparente, senza dover reinserire la password. Questa caratteristica è nota come Single Sign-On

⁵ Come conseguenza il Kerberos AFS impiega il protocollo RX per tutte le comunicazioni fra i client ed i database servers (che nella terminologia Kerberos agiscono da KDC). I client AFS standard che eseguono l'autenticazione scartano il TGT dopo aver acquisito un ticket di servizio. Ciò significa che non puoi ottenere ticket per altri servizi impiegando il token AFS

- La gestione delle informazioni di autenticazione è centralizzata e risiede sul server di autenticazione. I server applicativi non devono contenere informazioni di autenticazione dei loro utenti. Ciò è essenziale per ottenere i seguenti risultati:
 - o L'amministratore potrà disabilitare l'account di un utente agendo in un solo posto senza dover agire su più server applicativi che forniscono i vari servizi;
 - o L'utente, cambiando la password, lo fa per tutti i servizi contemporaneamente;
 - o Non vi è nessuna ridondanza di informazioni di autenticazione che altrimenti bisognerebbe difendere in più posti;
- Non solo gli utenti devono dimostrare di essere chi dicono di essere, ma, quando richiesto, anche i server applicativi devono provare la loro autenticità ai client. Questa caratteristica prende il nome di Mutua autenticazione;
- Se richiesto, dopo che l'autenticazione e l'autorizzazione sono avvenute, client e server devono poter stabilire una connessione criptata. A tal scopo, Kerberos fornisce supporto alla generazione e allo scambio di una chiave crittografica da utilizzare per cifrare i dati.

Prima di proseguire nella descrizione del protocollo è opportuno introdurre i principali termini che ricorrono nella trattazione dell'argomento tenendo comunque presente che kerberos è un protocollo abbastanza complicato ed i dettagli precisi di funzionamento sono decritti nell' RFC1510

Realm	Con il termine realm (regno) si indica un dominio amministrativo di autenticazione. Si intende cioè fissare i confini entro cui un server di autenticazione è autoritario nell'autenticare un utente, un host o un servizio
Principal	Un principal è il nome con cui si fa riferimento alle entry nel database del server di autenticazione. Ad ogni utente, host o servizio di un dato realm viene associato un principal ed il formato generico di un principal per un utente è : Nome[/Istanza]@REALM mentre per un servizio è Servizio/Hostname@REALM
Ticket	Un ticket è un biglietto che un client presenta ad un server applicativo per dimostrargli l'autenticità della sua identità. I ticket vengono emessi dal server di autenticazione e vengono criptati con la chiave segreta del server a cui sono destinati. Poiché tale chiave è un segreto condiviso soltanto tra il server di autenticazione e il server che fornisce il servizio, neanche il client stesso che ha richiesto il ticket potrà conoscerne o modificarne il contenuto. Le principali informazioni contenute in un ticket sono: • Il principal del richiedente (in genere è lo username); • Il principal del servizio a cui è destinato; • L'indirizzo IP della macchina client da cui il ticket può essere usato. In Kerberos 5 questo campo è opzionale e può essere anche multiplo allo scopo di far funzionare anche client sotto NAT o multihomed.

	• La data e l'ora (in formato di timestamp) in cui ha inizio la validità del ticket; • Il tempo massimo di vita (lifetime) del ticket • La chiave di sessione (ha un ruolo fondamentale che viene descritto più avanti); Ogni ticket ha una scadenza (in genere 10 ore). Ciò è essenziale in quanto su un ticket già emesso, il server di autenticazione non ha più nessun controllo. L'amministratore del realm, cioè, benchè possa in qualsiasi momento impedire l'emissione di nuovi ticket per un certo utente, non potrà impedire a quest'ultimo di utilizzare quelli che già possiede. Da qui la necessità di limitare il tempo di vita dei ticket in modo da limitare nel tempo eventuali abusi.
Tipo di cifratura	Kerberos 4 implementa un unico tipo di cifratura che è il DES a 56 bit. La fragilità di questa encryption unitamente ad altre vulnerabilità del protocollo, hanno di fatto reso obsoleto Kerberos 4. La versione 5 di Kerberos, invece, non fissa a priori il numero e il tipo di encryption supportate. Tra le encryption supportate sono degne di nota il triplo DES (3DES), le più nuove AES128 e AES256. e RC4-HMAC presente anche in Windows
Chiave di cifratura	Le chiavi di cifratura possono essere ottenute a partire dalle password mediante una funzione hash denominata string2key
Salt	In Kerberos 5, a differenza che nella versione 4, si è introdotto il concetto di salt della password. Si tratta di una stringa da concatenare alla password in chiaro prima di applicare la funzione string2key per ottenere la chiave. Per default, Kerberos 5 usa come salt il principal stesso dell'utente
KDC	Il server di autenticazione in un ambiente Kerberos, in virtù della sua funzione di distributore di ticket per l'accesso ai servizi, prende il nome di Key Distribution Center o più brevemente KDC. Benchè risieda interamente su un unico server fisico (spesso coincide con un unico processo) lo si può pensare logicamente diviso in tre parti: il Database, l'Authentication Server (AS) e il Ticket Granting Server (TGS). Database Il database è il contenitore delle entry associate a utenti e servizi ed ogni entry contiene le seguenti informazioni: • Il principal a cui la entry è associata; • Le encryption key ed il relativo kvno; • La durata massima di validità per un ticket associato al principal; • Il tempo massimo in cui un ticket associato al principal può essere rinnovato (solo Kerberos 5); • Gli attributi o flags che caratterizzano il comportamento dei ticket; • La data di scadenza della password; • La data di scadenza del principal dopo la quale non verranno più emessi ticket.

	Authentication Server (AS) L'Authentication Server è la parte del KDC che risponde a richieste di autenticazione iniziale da parte dei client, quando l'utente, non essendo ancora autenticato, deve digitare la password. L'AS, in risposta ad una richiesta di autenticazione, emette uno speciale ticket chiamato Ticket Granting Ticket o più brevemente TGT e se l'utente è veramente chi dice di essere (e capiremo in seguito come fa a dimostrarlo) potrà usare il TGT per ottenere altri ticket di servizio, senza dover ridigitare la password. Ticket Granting Server (TGS) Il Ticket Granting Server è la componente del KDC che distribuisce ai client con un TGT valido, i ticket di servizio che garantiscono l'autenticità dell'identità per ottenere la risorsa richiesta sui server applicativi. Si potrebbe pensare al TGS come ad un server applicativo (visto che per accedervi è necessario presentare il TGT) che fornisce come servizio l'emissione di ticket di servizio. Si faccia poi attenzione a non confondere le sigle TGT e TGS: la prima indica un ticket mentre la seconda un servizio.
Chiave di sessione	Utenti e i servizi condividono un segreto con il KDC. Tale segreto è, per gli utenti, la chiave derivata dalla loro password, mentre per i servizi, la loro chiave segreta (impostata dall'amministratore). Queste chiavi sono dette a lungo termine poiché non cambiano al variare della sessione di lavoro. È necessario però, almeno per il tempo in cui un client ha una sessione di lavoro aperta su di un server, che anche l'utente condivida con il servizio un segreto: questa chiave, generata dal KDC al momento dell'emissione di un ticket, prende il nome di Session Key (o chiave di sessione). La copia destinata al servizio viene imbustata dal KDC nel ticket (tanto il server applicativo conoscendo la chiave a lungo termine del servizio potrà decifrarlo ed estrarre la session key), mentre la copia destinata all'utente viene incapsulata in un pacchetto cifrato con la chiave a lungo termine dell'utente. La session key ha un ruolo fondamentale nel dimostrare l'autenticità dell'utente che vediamo nel paragrafo successivo.
Autenticazione	Benchè in un ticket sia presente il principal dell'utente e soltanto il server applicativo possa estrarre ed eventualmente manipolare tale informazione (essendo il ticket criptato con la chiave segreta del servizio), ciò non basta a garantire l'autenticità del client. Un impostore, infatti, potrebbe catturare (si ricordi l'ipotesi di rete aperta e insicura) il ticket quando questo viene inviato dal client legittimo al server applicativo e, al momento opportuno, inviarlo per ottenere illegittimamente il servizio. D'altra parte, mettere nel ticket anche gli indirizzi IP delle macchine da cui è possibile utilizzarlo non serve a molto: si sa che in una rete aperta e insicura gli indirizzi sono facilmente falsificabili. Per risolvere il problema, si deve sfruttare il fatto che client e server, almeno durante la sessione hanno in comune la session key che solo loro conoscono (anche il KDC la conosce visto che l'ha generata, ma tanto lui è fidato per definizione!!!). Si applica

	allora la seguente strategia: insieme alla richiesta contenente il ticket, il client affianca un altro pacchetto (l'autheticator) in cui ci mette il principal dell'utente e il timestamp (il suo in quel momento) e lo cripta con la session key; il server che deve offrire il servizio, ricevendo la richiesta, spacchetta prima il ticket, estrae la session key e se l'utente è veramente chi dice di essere riesce a decryptare l'autheticator estraendo il timestamp. Se quest'ultimo differisce dall'orario del server per meno di 2 minuti (ma la tolleranza è configurabile) l'autenticazione va a buon fine. Da tutto questo si evince la criticità della sincronizzazione tra le macchine appartenenti ad un realm.
Cache delle credenziali	Il client non conserva mai la password dell'utente così come non ne memorizza la chiave segreta ottenuta applicando la string2key: esse vengono utilizzate per decryptare le risposte del KDC ed immediatamente scartate. D'altra parte però, per implementare la caratteristica del single sign-on (SSO), in cui all'utente è richiesto di digitare la password una sola volta per sessione di lavoro, è necessario memorizzare i ticket e le relative session key. Il luogo dove vengono conservati questi dati è detto "Cache delle Credenziali". Dove tale cache debba essere collocata non riguarda il protocollo, ma varia da implementazione a implementazione. Spesso per questioni di portabilità si sceglie di collocarla nel filesystem (MIT e Heimdal). In altre implementazioni (AFS ed Active Directory), allo scopo di aumentare la sicurezza in caso di client vulnerabili, la cache delle credenziali è posta in una zona della memoria accessibile solo al kernel e non paginabile su disco.

Dopo aver illustrato i termini fondamentali per il protocollo Kerberos passiamo a descriverne il funzionamento illustrando i vari **messaggi** che le parti si scambiano durante le comunicazioni. Facendo anche riferimento alla figura seguente

I principali messaggi che le parti si scambiano sono:

Di seguito sono elencati i messaggi di cui ci occuperemo (vedi anche la figura sotto):

- AS_REQ è la richiesta iniziale di autenticazione dell'utente⁶. Tale messaggio è diretto alla componente del KDC nota come Authentication Server (AS);
- AS_REP è la risposta dell'AS alla richiesta precedente. Sostanzialmente contiene il TGT (criptato con la chiave segreta del TGS) e la chiave di sessione (criptata con la chiave segreta dell'utente richiedente);
- TGS_REQ è la richiesta da parte del client rivolta al Ticket Granting Server (TGS) per un ticket di servizio. Dentro questo pacchetto viaggia il TGT ottenuto dal messaggio precedente e un autenticatore generato dal client e criptato con la session key;
- TGS_REP è la risposta del Ticket Granting Server alla richiesta precedente. Ci si trova dentro il service ticket richiesto (criptato con la chiave segreta del servizio) e una chiave di sessione di servizio generata dal TGS e criptata con la precedente chiave di sessione generata dall'AS;

⁶

È realizzata tramite il comando *kinit*.

- AP_REQ è la richiesta che il client manda ad un server applicativo per accedere ad un servizio. Le componenti sono il ticket di servizio ottenuto dal TGS con la risposta precedente e un autenticatore generato sempre dal client, ma questa volta criptato con la chiave di sessione del servizio (generata dal TGS);
- AP_REP è la risposta che il server applicativo dà al client per provare di essere veramente il server che il client si aspetta. Questo pacchetto non è sempre richiesto. Il client lo richiede al server solo quando è necessaria la mutua autenticazione.

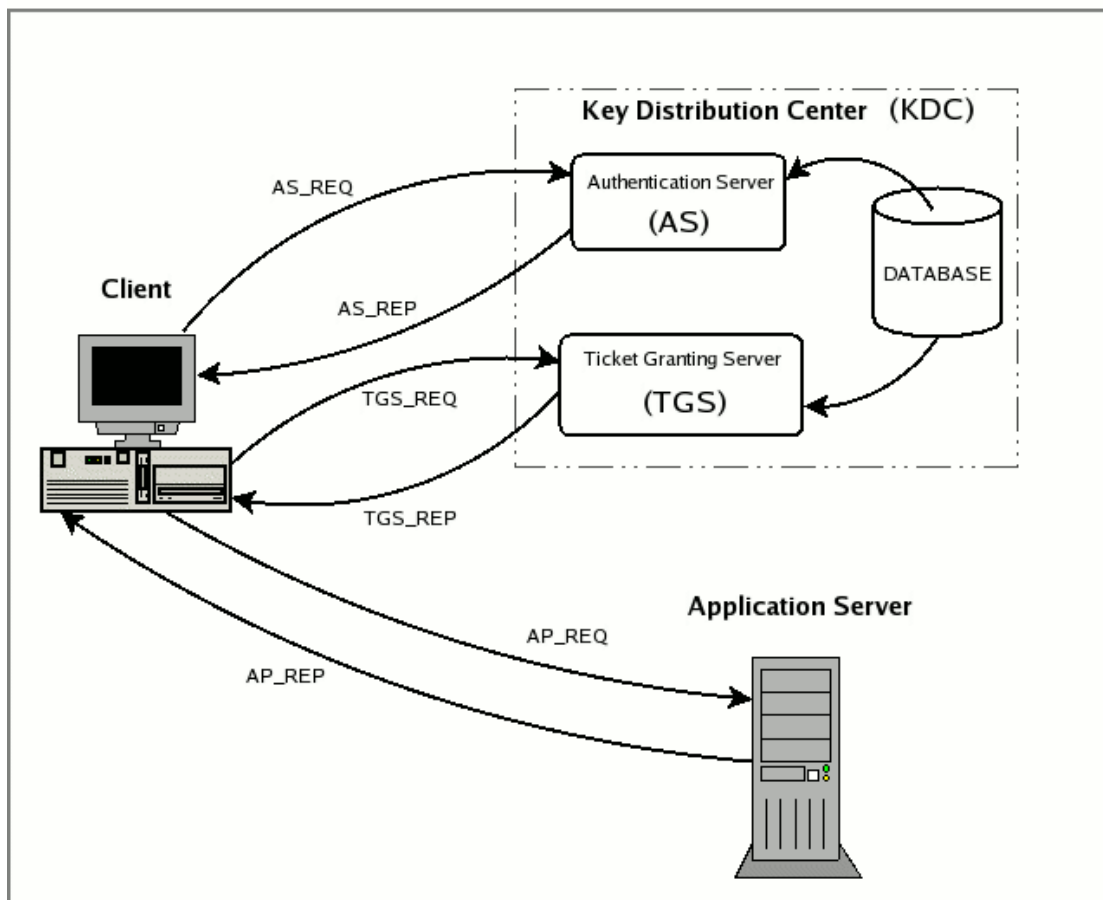


Figura 6: Schema di funzionamento del protocollo **Kerberos**

AS_REQ il client chiede all'AS del KDC (più precisamente all'AS) un TGT. La richiesta è completamente in chiaro ed appare così:

$AS_REQ = (Principal_{Client}, Principal_{Service}, IP_list, Lifetime)$

$Principal_{Client}$ è il principal associato all'utente che chiede di autenticarsi (per es. pippo@EXAMPLE.COM);

$Principal_{Service}$ è il principal associato al servizio per cui si chiede il ticket e quindi è la stringa "krbtgt/REALM@REALM"

IP_list è una lista di indirizzi IP che indica gli host da cui è possibile usare il ticket che verrà emesso

$Lifetime$ è il tempo massimo di validità (richiesto) per il ticket da emettere

AS_REQ All'arrivo della richiesta precedente, l'AS controlla se $\text{Principal}_{\text{Client}}$ e $\text{Principal}_{\text{Service}}$ esistono nel database del KDC: se almeno uno dei due non esiste viene inviato al client un messaggio d'errore, altrimenti l'AS elabora la risposta come segue:

- Crea in maniera random una chiave di sessione che sarà il segreto condiviso tra il client e il TGS. Diciamola SKTGS;
- Crea il TGT mettendogli dentro il principal del richiedente, il principal del servizio, la lista di indirizzi IP (queste prime tre informazioni sono copiate così come arrivano dal pacchetto AS_REQ), la data e l'ora (del KDC) in formato di timestamp, il lifetime e infine la session key SKTGS; il Ticket Granting Ticket appare cioè così:

$\text{TGT} = (\text{Principal}_{\text{Client}}, \text{krbtgt/REALM@REALM}, \text{IP_list}, \text{Timestamp}, \text{Lifetime}, \text{SKTGS})$

- Genera e spedisce la risposta contenente: il ticket creato prima, cifrato con la chiave segreta del servizio (diciamola KTGS); il principal del servizio, il timestamp, il lifetime e la session key tutto criptato con la chiave segreta dell'utente richiedente il servizio (diciamola KUser). In sintesi:

$\text{AS_REP} = \{ \text{Principal}_{\text{Service}}, \text{Timestamp}, \text{Lifetime}, \text{SKTGS} \} \text{KUser} \{ \text{TGT} \} \text{KTGS}$

Potrebbe sembrare che tale messaggio contenga informazioni ridondanti ($\text{Principal}_{\text{Service}}$, il timestamp, lifetime, la session key). Ma non è così: le informazioni presenti nel TGT, essendo criptate con la chiave segreta del server non potranno essere lette dal client e pertanto vanno ripetute. A questo punto, quando il client riceve il messaggio di risposta, chiederà all'utente di immettere la password. A quest'ultima viene concatenato il salt e poi applicata la funzione string2key con il cui risultato si tenta di decriptare la parte di messaggio cifrata dal KDC con la chiave segreta dell'utente. Se l'utente è veramente chi dice di essere e quindi ha immesso la password giusta, l'operazione di decifratura andrà a buon fine e si potrà estrarre la session key che insieme al TGT (che invece rimane criptato) viene conservata nella cache delle credenziali dell'utente.

TGS_REQ A questo punto, l'utente, che ha già dimostrato di essere chi dice di essere (quindi ha nella sua cache delle credenziali un TGT e una chiave di sessione SKTGS) e vuole accedere ad un servizio per il quale non ha ancora un ticket adatto, spedisce al Ticket Granting Server una richiesta (TGS_REQ) costruendola nel seguente modo:

- Crea un autenticatore con il principal dell'utente, il timestamp della macchina client e cripta tutto con la chiave di sessione condivisa con il TGS, cioè:

$\text{Authenticator} = \{ \text{Principal}_{\text{Client}}, \text{Timestamp} \} \text{SKTGS}$

- Crea il pacchetto di richiesta contenente: in chiaro il principal del servizio per cui si vuole il ticket e un lifetime; il Ticket Granting Ticket che è già criptato con la chiave del TGS; l'autenticatore appena creato. In sintesi:

$TGS_REQ = (Principal_{Service}, Lifetime, Authenticator) \{ TGT \} K_{TGS}$

TGS_REP Quando arriva la richiesta precedente, il TGS verifica dapprima che il principal del servizio richiesto ($Principal_{Service}$) esista nel database del KDC. Se esiste, apre il TGT utilizzando la chiave del $krbtgt/REALM@REALM$ ed estrae la session key (SK_{TGS}) con la quale decifra l'autenticatore. Affinché l'emissione del ticket di servizio possa avvenire controlla che abbiano esito positivo le condizioni:

- Il TGT non sia scaduto;
- Il $Principal_{Client}$ presente nell'autenticatore coincida con quello presente nel TGT;
- L'autenticatore non sia presente nella replay cache e non sia scaduto;
- Se IP_list non è nulla verifica che l'indirizzo IP sorgente del pacchetto di richiesta (TGS_REQ) sia uno di quelli contenuti nella lista;

Le precedenti condizioni verificate dimostrano che il TGT appartiene veramente all'utente che ha fatto la richiesta e pertanto il TGS inizia a processare la risposta come segue:

- Crea in maniera random una chiave di sessione che sarà il segreto condiviso tra il client e il servizio. Diciamola $SK_{Service}$;
- Crea il ticket di servizio mettendogli dentro il principal del richiedente, il principal del servizio, la lista di indirizzi IP, la data e l'ora (del KDC) in formato di timestamp, il lifetime (come minimo tra il lifetime del TGT e quello associato al principal del servizio) e infine la session key $SK_{Service}$. Detto $T_{Service}$ il nuovo ticket si ha:
- $T_{Service} = (Principal_{Client}, Principal_{Service}, IP_list, Timestamp, Lifetime, SK_{Service})$
- Spedisce il messaggio di risposta contenente: il ticket creato prima cifrato con la chiave segreta del servizio (diciamola $K_{Service}$); il principal del servizio, il timestamp, il lifetime e la nuova session key tutto criptato con la chiave di sessione estratta dal TGT. In sintesi:

$TGS_REP = \{ Principal_{Service}, Timestamp, Lifetime, SK_{Service} \} K_{TGS} \{ T_{Service} \} K_{Service}$

Quando il client riceve la risposta, avendo nella cache delle credenziali la session key SK_{TGS} , potrà decifrare la parte di messaggio contenente l'altra chiave di sessione e memorizzarla insieme al ticket di servizio $T_{Service}$ che però rimane criptato.

AP_REQ Il client, avendo ormai le credenziali per accedere al servizio (cioè il ticket e la relativa session key), può chiedere al server applicativo l'accesso alla risorsa mediante un messaggio AP_REQ . Si tenga presente, che a differenza dei precedenti messaggi in cui era coinvolto il KDC, l' AP_REQ non è standard, ma varia da applicazione ad applicazione. Si lascia cioè al programmatore applicativo il compito di stabilire la strategia con cui il client sfrutta le sue credenziali per provare al server la

sua identità. Tuttavia, come esempio possiamo considerare la seguente strategia:

- Il client crea un autenticatore contenente il principal dell'utente, il timestamp e cripta tutto con la chiave di sessione SKService che condivide con il server applicativo, cioè:
- Authenticator = { PrincipalClient , Timestamp }SKService
- Crea il pacchetto di richiesta contenente il ticket di servizio TService che è criptato con sua chiave segreta e l'autenticatore appena creato.

In sintesi:

AP_REQ = Authenticator { TService }KService

Quando arriva la richiesta precedente, il server applicativo apre il ticket utilizzando la chiave segreta del servizio richiesto ed estrae la session key SKService con la quale decripta l'autenticatore. Per stabilire che l'utente richiedente è autentico e quindi concedere accesso al servizio, il server verifica le seguenti condizioni:

- Il ticket non sia scaduto;
- Il PrincipalClient presente nell'autenticatore coincida con quello presente nel ticket;
- L'autenticatore non sia presente nella replay cache e non sia scaduto;
- Se IP_list (estratta dal ticket) non è nulla verifica che l'indirizzo IP sorgente del pacchetto di richiesta (AP_REQ) sia uno di quelli contenuti nella lista;

PreAutenticazione Come si è visto nella descrizione dell'Authentication Server Reply (AS_REP), il KDC, prima di distribuire un ticket, controlla semplicemente che il principal del richiedente e del fornitore il servizio esistano nel database. In particolare poi, se si tratta della richiesta di un TGT la cosa è ancora più facile perché krbtgt/REALM@REALM esiste sicuramente e quindi basta conoscere l'esistenza del principal di un utente per poter ottenere con una semplice richiesta di autenticazione iniziale un TGT. Ovviamente, tale TGT, se la richiesta è avvenuta da parte di un utente illegittimo, non potrà essere usato poiché questi non conoscendo la password non potrà ottenere la session key con cui forgiare un autenticatore valido. Tuttavia, questo ticket, ottenuto in maniera così tanto gratuita può essere sottoposto ad un attacco brute-force nel tentativo di indovinare la chiave a lungo termine del servizio a cui il ticket è destinato. È chiaro, che indovinare il segreto di un servizio non è una cosa banale neanche con le attuali potenze di calcolo, ma comunque, con Kerberos 5, allo scopo di rafforzare la sicurezza, si introduce il concetto di pre-autenticazione. Se cioè, le policy (configurabili) del KDC richiedono la preautenticazione, alla richiesta iniziale da parte di un client, l'Authentication Server risponde con un pacchetto di errore indicante la necessità di preautenticarsi. Il client, visto l'errore, chiede all'utente l'immissione della password e riformula la richiesta ma questa volta

aggiungendovi il timestamp criptato con la chiave a lungo termine dell'utente che, come sappiamo, si ottiene applicando la string2key alla password in chiaro dopo averci aggiunto l'eventuale salt. Questa volta, il KDC, poiché conosce la chiave segreta dell'utente, tenta di decriptare il timestamp presente nella richiesta e se ci riesce ed il timestamp è coerente, cioè compreso nella tolleranza stabilita, conclude che il richiedente è autentico e il processo di autenticazione continua normalmente. Si badi bene che la preautenticazione è una policy del KDC e quindi il protocollo non la richiede necessariamente. A livello implementativo, MIT Kerberos 5 ed Heimdal hanno per default la preautenticazione disabilitata, mentre il Kerberos di Windows Active Directory e il kaserver di AFS (che è appunto un Kerberos 4 preautenticato) la richiedono

Tipi di ticket

Esaminiamo ora in maggior dettaglio la struttura e le caratteristiche dei ticket. Come già anticipato, ora che sono noti il funzionamento del KDC e i messaggi tra gli host che partecipano all'autenticazione, si può riparlare di ticket. Questi, a seconda che abbiano o meno al loro interno settati degli attributi (noti anche come flag) si comportano in una determinata maniera. Elenchiamo di seguito i tipi di ticket più importanti, e, anche se non troppo corretto visto che qui si parla di protocollo, faremo riferimento (quel che basta per fissare le idee) alla versione 1.3.6 di Kerberos 5 del MIT.

Ticket iniziali

Un ticket iniziale è quello che si ottiene direttamente dall'AS, cioè quando l'utente deve autenticarsi immettendo la password. Da qui si deduce che il TGT è sempre un ticket iniziale. I ticket di servizio, d'altra parte, vengono distribuiti dal TGS su presentazione di un TGT e quindi non sono ticket iniziali. C'è tuttavia un'eccezione a questa regola: alcune applicazioni Kerberos potrebbero richiedere, per garantirsi che l'utente abbia solo qualche istante prima digitato la password, che il ticket di servizio sia iniziale; in questo caso il ticket pur non essendo un TGT viene richiesto all'AS invece che al TGS e quindi è un ticket iniziale. Operativamente, l'utente pippo che vuole ottenere un ticket che sia iniziale (quindi senza sfruttare un TGT) per il servizio imap sulla macchina mbox.example.com usa il comando:

```
[pippo@client01 pippo]$ kinit -s
imap/mbox.example.com@EXAMPLE.COM
pippo@EXAMPLE.COM
Password for pippo@EXAMPLE.COM:
[pippo@client01 pippo]$
[pippo@client01 pippo]$ klist -f
Ticket cache: FILE:/tmp/krb5cc_500
Default principal: pippo@EXAMPLE.COM
Valid starting Expires Service
principal
01/27/05 14:28:59 01/28/05 14:28:39
imap/mbox.example.com@EXAMPLE.COM
Flags: I
```

```
Kerberos 4 ticket cache: /tmp/tkt500
```

```
klist: You have no tickets cached
```

Si noti la presenza del flag I indicante che si tratta di un ticket iniziale.

Ticket rinnovabili

Un ticket rinnovabile può essere rimandato al KDC perché venga rinnovato, cioè gli venga riassegnato l'intero lifetime. È ovvio che il KDC onorerà la richiesta di rinnovo solo se il ticket non è ancora scaduto e non ha superato il tempo massimo di rinnovo (definito nel database del Key Distribution Center). La rinnovabilità di un ticket concilia la necessità di avere ticket di breve durata per ragioni di sicurezza, con quella di non dover ridigitare la password per lunghi periodi: si pensi ad esempio ad un job che deve essere processato per giorni e non può richiedere l'intervento umano. Nell'esempio successivo pippo chiede un ticket che duri al massimo un'ora ma rinnovabile per 8 giorni:

```
kinit -l 1h -r 8d pippo
Password for pippo@EXAMPLE.COM:
[pippo@client01 pippo]$
[pippo@client01 pippo]$ klist -f
Ticket cache: FILE:/tmp/krb5cc_500
Default principal: pippo@EXAMPLE.COM
Valid starting      Expires              Service
principal
01/27/05 15:35:14   01/27/05 16:34:54
krbtgt/EXAMPLE.COM@EXAMPLE.COM
renew until 02/03/05 15:35:14, Flags:
```

```
RI
```

```
Kerberos 4 ticket cache: /tmp/tkt500
```

```
klist: You have no tickets cached
```

mentre pippo per rinnovare il suo ticket senza ridigitare la password:

```
[pippo@client01 pippo]$ kinit -R
[pippo@client01 pippo]$
[pippo@client01 pippo]$ klist -f
Ticket cache: FILE:/tmp/krb5cc_500
Default principal: pippo@EXAMPLE.COM
Valid starting      Expires              Service
principal
01/27/05 15:47:52   01/27/05 16:47:32
krbtgt/EXAMPLE.COM@EXAMPLE.COM
renew until 02/03/05 15:35:14, Flags:
```

```
RIT
```

```
Kerberos 4 ticket cache: /tmp/tkt500
```

```
klist: You have no tickets cached
```

Ticket forwardabili

Supponiamo di avere una sessione di lavoro su di una macchina con il relativo TGT e di voler da questa fare login su di un'altra conservando il

ticket. La soluzione a questo problema sono i ticket forwardabili. Un ticket forwardato da un host all'altro è esso stesso forwardabile per cui si conclude che una volta autenticati è possibile accedere al login su tutte le macchine che si vuole senza dover ridigitare nessuna password. Per ottenere lo stesso risultato senza Kerberos, bisognerebbe ricorrere a metodi molto meno sicuri come rsh o all'autenticazione a chiave pubblica con ssh. Tuttavia, quest'ultimo metodo può essere impraticabile su sistemi in cui le home directory dell'utente sono su filesystem di rete (es. NFS o AFS) poiché la chiave privata (che appunto dovrebbe essere privata) verrebbe fatta passare sulla rete.

3.2.3.3. La cella ENEA.IT

La cella AFS ENEA.IT è caratterizzata da 6 db-server basati su OpenAFS 1.4.0 e sistema operativo Scientific Linux 4.2⁷ ubicati nelle sedi di Frascati (3), Bologna, Casaccia, Trisaia e da 15 File-servers basati su OpenAFS e su diversi sistemi operativi (fra i quali Scientific Linux, Solaris, AIX) dislocati in 6 sedi

Nel corso del 2006 è stata effettuata, la migrazione dell'autenticazione da quella nativa di AFS, basata su una implementazione pre-standard di Kerberos 4, alla versione Kerberos 5.

L'implementazione Kerberos 5 scelta è quella prodotta da MIT e disponibile all'URL:

<http://web.mit.edu/Kerberos>

mentre l'architettura individuata sarà costituita da:

- master KDC [afskdc.frascati.enea.it] del realm ENEA.IT
- i dbserver afs sono sostituiti dai server slave di autenticazione Kerberos 5
- kaserver disattivato e sostituito da fakeka che emula la funzionalità di kaserver in ambiente kerberos 5

Con tale approccio tutte le applicazioni che fanno richiesta di token (ad es. klog, ssh,...) continuano a funzionare senza problemi. Prima di procedere con un'analisi di maggior dettaglio della situazione realizzata è opportuno chiarire i termini impiegati.

kaserver Kerberos Authentication server è il server di autenticazione in AFS

fakeka è un emulatore di kaserver che tratta il protocollo AFS RX⁸ al solo fine di supportare il funzionamento del comando klog. Tale emulatore è impiegato insieme ad un KDC Kerberos 5 al fine di supportare i client

⁷ Scientific Linux è una distribution GNU/Linux free e open source free GNU/Linux, sviluppata congiuntamente dal Fermi National Accelerator Laboratory e dal CERN (European Organization for Nuclear Research), basata su Red Hat Enterprise Linux e che punta ad essere compatibile al 100% con RHEL. Tale prodotto è derivato dai sorgenti free e open source resi disponibili da Red Hat, Inc., ma non è prodotta, mantenuta o supportata da Red Hat. Nello specifico, questo prodotto è costruito a partire dal codice sorgente delle versioni di RHEL, sotto i termini e le condizioni degli EULA di RHEL e di GPL.

⁸ Per esaminare i protocolli impiegati da ASF si consideri che quello di livello più basso è l'UDP al di sopra del quale troviamo il Remote Procedure Call (RPC). In generale l'RPC consente allo sviluppatore di costruire le applicazioni impiegando il modello client/server, nascondendo i sottostanti meccanismi di rete. Nello specifico AFS impiega Rx, un protocollo RPC sviluppato specificatamente per AFS durante la fase di sviluppo alla Carnegie Mellon University. Al disopra dell'RPC sussistono una serie di processi server e di interface che impiegano il protocollo Rx per la comunicazione fra i sistemi.

AFS ed è usualmente usato con ka-forwarder; fakeka deve girare sullo stesso host sul quale gira il KDC Kerberos 5 dacchè deve poter accedere al database del KDC.

ka-forwarder rimane in ascolto per le richieste verso un AFS Authentication Server e le inoltra a un server fakeka; il processo dovrebbe quindi girare su ogni AFS database server

In effetti a valle di tale migrazione i servizi che utilizzano direttamente il token o che hanno la capacità di richiederlo ad una macchina dbserver dovrebbero mantenere le loro funzionalità. Caso specifico è costituito da LSF che utilizza le utility *gettok* e *puttok* per trasmettere i token).

3.2.4. Il Gestore delle Risorse LSF

Come già accennato in precedenza il gestore delle risorse nella GRID ENEA è costituito da LSF. LSF sta per Load Sharing Facility ed è un sistema distribuito general purpose della Platform Computing Corporation che unisce un cluster di computer in un singolo sistema virtuale per sfruttare al meglio le risorse disponibili in rete. Elaboratori di vari produttori possono essere integrati in un unico sistema dove è possibile delegare all'infrastruttura stessa la selezione del miglior host per eseguire i propri applicative

LSF può selezionare automaticamente gli host in un ambiente eterogeneo in base alle correnti condizioni ed alle risorse richieste dall'applicazione. Con LSF, quindi risulta possibile eseguire dei job su sistemi remoti come se gli stessi fossero eseguiti sul sistema locale. LSF è in grado di supportare sia elaborazioni interattive che di tipo batch. LSF gestisce una serie di informazioni sia relative alle risorse sia relative al carico di tutti gli host presenti nel cluster: Le informazioni sulle risorse includono, ad esempio, il numero di processori per ciascun host, la memoria fisica totale disponibile per i job dell'utente, il tipo, il modello e la velocità relativa di ogni host, l'eventuale finestra temporale in cui l'host è disponibile ecc. mentre quelle dinamiche e relative al carico possono essere il carico della CPU, la memoria reale disponibile, lo spazio di swap disponibile, l'attività di paginazione e di I/O, il numero di utenti interattivi loggati ecc.

L'elaborazione può essere di tipo batch o di tipo interattivo. Nel primo caso LSF sottomette i job di tipo batch in una coda dalla quale vengono automaticamente eseguiti non appena un host adeguato risulterà disponibile. Nel caso di job interattivi essi saranno lanciati immediatamente e di solito richiederanno qualche input tramite un'interfaccia utente grafica o alfanumerica. Sia l'input che l'output verrà automaticamente spedito fra il sistema locale e quello remoto sul quale il job è realmente eseguito. E' anche possibile sottomettere i job interattivi tramite batch per sfruttare i vantaggi delle code e delle loro policy ma in tal caso il job sarà sottoposto alle policy di schedulazione della coda e quindi potrebbe non essere avviato immediatamente.

Il funzionamento di LSF è basato su una serie di demoni che sono in esecuzione sui vari nodi del cluster.

Descriviamo sommariamente le loro funzionalità:

- mbatchd** (Master Batch Daemon – MBD) gira sul cosiddetto Master Host ed è avviato dal demone *sbatchd*. Riceve le richieste di sottomissione dei job nonché le richieste di query informative. Invia i job agli host in base alle scelte di *mbschd*.
- mbschd** (Master Batch Scheduler Daemon) gira sul master host, interagisce con *mbatchd* ed è avviato da *sbatchd*. E' deputato all'attivazione delle politiche di schedulazione basate sulle richieste dei job e sulle policy
- sbatchd** (Slave Batch Daemon – SBD) è in esecuzione su ogni server del cluster. Accetta le richieste di esecuzione di job dall'MBD, e gestisce l'esecuzione locale del job. Controlla l'esecuzione dei job, fa rispettare le policy dei batch, riporta lo stato dei job all'MBD e lancia l'MBD
- res** Remote Execution Server – è in esecuzione su ogni server del cluster. Accetta le richieste di esecuzione da remoto per fornire l'esecuzione trasparente e sicura di job e task
- lim** Load Information Manager – è in esecuzione su ogni server del cluster. Raccoglie informazioni relative al carico dell'host ed alla sua configurazione e li inoltra al master LIM che è in esecuzione sul master host
- pim** Process Information Manager – è in esecuzione su ogni server del cluster ed è attivato dal LIM. Raccoglie le informazioni sui processi in esecuzione sull'host quali la percentuale di CPU e la memoria impiegata dai job e riporta tali informazioni al *sbatchd*.

In sintesi in ogni cluster LSF c'è un MBD e su ogni host c'è un SBD. I job degli utenti sono tenuti in code di batch da *mbatchd*, che periodicamente controlla le informazioni di carico di tutti gli host candidati. Quando un host con le necessarie risorse diventa disponibile, *mbatchd* spedisce il job all'*sbatchd* presente su quell'host. Se più di un host risulta disponibile viene scelto il migliore. Il daemon *sbatchd* controlla l'esecuzione dei job e riporta lo stato dei job a *mbatchd*. La LSF Batch Library (LSBLIB) è l'API per LSF Batch, e fornisce un facile accesso ai servizi di *mbatchd* e *sbatchd*. LSBLIB fornisce una interfaccia potente per gli utenti avanzati che vogliano sviluppare nuove applicazioni di processamento batch in linguaggio C. In ulteriore dettaglio si consideri che LSF consente l'esecuzione sia di job interattivi che di job batch. Il demone che di occupa dell'esecuzione dei job interattivi è denominato Remote Execution Server.

3.2.5. Interazione fra LSF e AFS

In generale LSF gestisce i permessi utente per vari tipi di File System in modo che l'utente non debba preoccuparsi di dove siano memorizzati fisicamente i file. In effetti LSF è progettato per reti nelle quali gli host abbiano file system condivisi, ed i file abbiano gli stessi nomi sui vari hosts; in effetti LSF include anche il supporto per copiare i dati dell'utente all'execution host prima di eseguire un job batch e per copiare i risultati nell'altro verso dopo l'esecuzione dello stesso. Nelle reti dove i file system non sono condivisi tale meccanismo può essere impiegato per fornire l'accesso ai dati locali da parte dei job remoti. I file system supportati sono:

in ambiente Unix:

- Network File System (NFS)⁹
- Andrew File System (AFS)
- Distributed File System (DCE/DFS)

in ambiente Windows:

- le directory contenenti i file LSF possono essere condivise fra gli host tramite un sistema Windows server

In maggior dettaglio la versione 6.1 di Platform LSF consente il trasferimento del token AFS per i job LSF sui seguenti sistemi¹⁰

HP Alpha/AXP systems running Tru64 UNIX (Digital UNIX) 4.0

HP systems running HP-UX 10.20

IBM RS/6000 systems running AIX 4.1, 4.2, 4.3, 5.0, 5.1, and 5.2

x86 systems running Linux Kernel 2.2 and 2.4, compiled with glibc 2.1.x (Transarc AFS 3.5)

SGI systems running IRIX 6.2, 6.4, and 6.5

Sun SPARC systems running Solaris 2.5.1 and 2.6

Sun SPARC systems running 32-bit Solaris 7 and 8 (rtokd not supported)

Per supportare AFS bisogna essenzialmente copiare una serie di file nella directory LSF_SERVERDIR¹¹

res vedi descrizione del demone nel paragrafo precedente

sbatchd vedi descrizione del demone nel paragrafo precedente

⁹ I file system di tipo NFS possono essere montati sia in maniera permanente che a richiesta utilizzando automount

¹⁰ A tal proposito si vede la pagina <http://www2.platform.com/services/support/services/Platforms6.1.asp>

¹¹ LSF_SERVERDIR è la directory per tutti i file binari e gli script del server e gli eseguibili esterni invocati dai demoni LSF, deve essere di proprietà di root e condivisa da tutti gli host dello stesso tipo

i due eseguibili `res` e `sbatchd` sono gli stessi della classica distribution di LSF eccetto che per il fatto che essi sono linkati con le librerie AFS.

Altri file da copiare nella directory

- gettok** `gettok` prende il/i token AFS dal kernel e lo/li visualizza in formato ASCII sullo standard output.
- puttok** `puttok` legge dallo standard input il/i token AFS nel formato generato da `gettok`, e setta il/i token nel kernel.
- esub, eexec** Si tratta di shell scripts invocati da LSF per supportare l'inoltro del token dall'host che sottomette a quello di esecuzione e per supportare il rinnovo del token. E' possibile modificare ulteriormente tali script per customizzare ulteriormente l'elaborazione del token (ad esempio utilizzare un software di cifratura specifico).

Inoltre si consideri che se la `LSB_SHAREDIRE` è definita in AFS il demone `mbatchd` deve essere settato in modo che possieda il token per poter scrivere nella directory `LSB_SHAREDIRE`.

Per poter impiegare tale funzionalità è necessario configurare un opportuno kit detto *token renewal kit* che consiste di una parte client ed una server costituite, rispettivamente, dagli eseguibili `rtok` e `rtokd`. Essenzialmente **rtok** è impiegato per richiedere al demone **rtokd**, in esecuzione sul server AFS, il rinnovo del token per un certo utente. I binari `rtok/rtokd` impiegano anche `eexec`, `esub`, `gettok`, e `puttok`.

3.3. Interoperabilità con altre GRID

Spesso le iniziative che si succedono nel mondo delle GRID portano alla nascita di diverse GRID che presentano interazioni reciproche limitate se non nulle anche perché il modello di Virtual Organisation (VO) che viene adottato nelle varie realtà è pensato per uno specifico tipo di dominio e tutte le interazioni, quali la condivisione delle risorse sono limitate all'interno della VO presa in esame. Inoltre, le tecnologie abilitanti in generale e i middleware più in particolare sono spesso differenti.

Nel momento in cui si desidera realizzare l'interazione fra due differenti GRID i problemi che si incontrano sono generalmente relativi alla differente gestione delle risorse, alle diverse policy di sicurezza, ai differenti requirement di QoS e di SLA (Service Level Agreement), ai diversi protocolli utilizzati.

E' chiaro che l'entità dei problemi che bisogna affrontare per assicurare una interoperabilità più o meno stretta diventa funzione delle differenti architetture di GRID considerate.

Nel caso specifico di ENEA i principali problemi di integrazione dovrebbero afferire a tre settori che sono poi i punti in qualche modo qualificanti e caratteristici dell'architettura di GRID adottata dall'ente:

Condivisione di file

La scelta di ENEA è quella di affidarsi al prodotto OpenSource AFS che consente di avere un file system distribuito per cui nel momento in cui si richiede la fruizione di una risorsa non si presenta la necessità di dover procedere con un trasferimento di dati fra i due sistemi che partecipano alla GRID. Altre architetture GRID potrebbero invece non prevedere file system distribuiti e quindi contemplare un trasferimento di file prima dell'esecuzione del servizio.

Autenticazione

Nella GRID ENEA un ruolo fondamentale in tale ambito è dato dai token previsti da AFS e che nell'ambito delle piattaforme supportate possono essere trasferiti al fine di garantire un accesso sicuro ed un ambiente di lavoro omogeneo fra i vari sistemi della GRID. In altri contesti GRID il momento dell'autenticazione potrebbe essere basato sull'esistenza di altri meccanismi quali i certificati.

Gestione delle Risorse

La gestione delle risorse per ENEA-GRID è realizzata dal prodotto multipiattaforma LSF in grado di essere eseguito su ambienti eterogenei Linux, AIX, SUN, SGI, Windows mentre middleware di GRID potrebbero essere supportati solo in alcuni ambienti (es. Linux)

Quindi nel momento in cui si desidera sfruttare le risorse della ENEA-GRID da altre architetture di GRID un possibile approccio potrebbe prevedere un opportuno gateway che si occupi di risolvere i problemi di compatibilità e quindi, in sintesi di mappare l'utente della GRID in un utente in possesso di un token AFS, spostare i dati sul File System AFS, sottomettere il job tramite i comandi di LSF ed alla fine trasferire, nel senso opposto, i dati di output disponibili in AFS.

4. Analisi delle criticità e possibili approcci risolutivi

4.1. Integrazione UNIX-Windows

Dall'analisi riportata emerge che uno degli aspetti più rimarchevoli sia quello della non completa integrazione nella GRID ENEA dei sistemi elaborativi con s.o. Windows .

I prodotti chiave sui quali è basata l'architettura della GRID supportano i due mondi quindi sia i sistemi operativi UNIX sia Windows; ciò vale per il gestore delle risorse LSF, per il file system AFS e per il sistema di distribuzione delle applicazioni Citrix Presentation Manager.

Il problema nasce nel momento in cui , si vuole accedere a risorse e servizi della Grid provenienti dal mondo Microsoft Windows , bisogna condividere fra i due mondi il sistema di accounting e il file system distribuito.

4.1.1. Autenticazione

L'accesso alla GRID è subordinato ad una autenticazione che è basata sulle credenziali tipiche di UNIX; in effetti tutti i sistemi UNIX hanno il file `/etc/passwd` che contiene l'intera lista degli utenti della GRID, vale a dire un elenco completo degli utenti di tutta l'ENEA (al momento sono registrati più di mille utenti). La presenza di un file system distribuito quale AFS consente di poter modificare tale file una sola volta e le modifiche sono direttamente viste da tutti i sistemi della GRID. Inoltre un meccanismo di SSO consente che al momento dell'accesso, fornendo le proprie credenziali, l'utente veda come propria home directory il proprio volume AFS quindi una home del tipo `/afs/enea.it/sedeenea/nomeutente` e dunque, indipendentemente dal sistema sul quale avviene l'accesso, l'utente è in grado di ritrovare i propri file ed avere a disposizione lo stesso ambiente a prescindere dall'host.

Pertanto tale sistema è ovviamente limitato al mondo UNIX e per quanto riguarda il mondo Windows il problema dell'integrazione è in fase di studio mediante una convergenza con il sistema Active Directory impiegando come anello di congiunzione con il protocollo di autenticazione Kerberos 5 che è supportato dai due mondi.

4.1.2. Autorizzazioni

Altro aspetto riguarda quello delle autorizzazioni ossia un sistema che consenta di selezionare gli utenti che possono accedere alle applicazioni presenti sulla GRID. L'interfaccia distribuita dai server Citrix sembra essere in qualche modo statica ossia sembra che tutti gli utenti, abbiano a disposizione e possano impiegare le stesse risorse. Allo stato per effettuare una politica di autorizzazioni si potrebbe solo agire sulle

Access Control List del file system AFS visto che è plausibile che le applicazioni siano installate sul file system distribuito. Come abbiamo visto in precedenza le ACL di AFS sono abbastanza potenti in quanto consentono ben 7 diversi profili di accesso ai vari file. Le politiche di autorizzazione potrebbero quindi essere realizzate limitando l'accesso fisico ai file sui quali la risorsa poggia. Una soluzione più adeguata, flessibile e opportuna è invece quella di introdurre un *policy manager* con il fine specifico di gestire tali politiche. Un file system distribuito quale AFS consente di avere a disposizione le proprie home directory su tutti i sistemi siano essi Unix e Windows in quanto il prodotto è come già detto multiplatforma. Peraltro esiste allo stato una forte limitazione che è dovuta al non supportato trasferimento del token da parte del gestore delle risorse LSF dal mondo Unix a quello Windows. In effetti LSF non supporta AFS in ambiente Windows e quindi nel momento in cui si desidera attivare un job su una macchina Windows, il mancato trasferimento del token AFS (che ovviamente consente all'utente di poter accedere al proprio volume) determina di fatto l'impossibilità di avere il proprio spazio a disposizione sull'host Windows e, nel caso il sistema di autorizzazione risultasse basato sulle ACL di AFS, l'operazione sarebbe di fatto impossibile. E' dunque opportuno verificare se sia possibile effettuare l'operazione di trasferimento del token da e verso sistemi Windows, partendo dai codici sorgenti delle routine a ciò destinate e disponibili per il solo mondo Unix.

4.2. Interoperabilità

Per quanto attiene alle problematiche connesse all'interoperabilità con altre GRID, l'assenza di un middleware nell'architettura della GRID determina la necessità di predisporre dei gateway che consentano l'interazione di ENEA-GRID con altre GRID. In effetti, i problemi che si pongono nel settore dell'interoperabilità sono di varia natura e possono essere più o meno critici quanto più differenti siano le architetture delle GRID che si vogliono integrare. Nel caso specifico per l'accesso alla GRID ENEA da parte di altre griglie dovranno essere previsti dei meccanismi che provvedano alla conversione od al *wrapping* dei comandi esterni in comandi di LSF mentre per quanto riguarda l'accesso al file system si renderà necessario porre in essere dei meccanismi che consentano la conversione delle autorizzazioni esterne in token AFS. La soluzione ottimale sarebbe quella di costruire un gateway generalizzato in grado di agire come un framework che, opportunamente configurato, sia in grado di esibire, nei confronti delle GRID che si desidera integrare, le interfacce per le interazioni più comuni.

5. Conclusioni

L'attuale infrastruttura dell'ENEA-GRID utilizza in maniera quasi esclusiva dei server di tipo UNIX e risulta di fondamentale importanza, nel prossimo futuro, l'utilizzo di risorse basate su sistema operativo Microsoft Windows. Nell'ambito del conseguimento di tale obiettivo un'altra unità convenzionata, il CASPUR, si sta occupando di integrare nel modo migliore le security di Windows, e nello specifico quelle di Active Directory, con OpenAFS per permettere l'accesso, con credenziali uniche, ad entrambi gli ambienti ed alle risorse disponibili attraverso OpenAFS.

L'attività del DIIMA sarà tesa all'introduzione di un *policy manager*, al fine di poter disporre di meccanismi di autorizzazione che, al momento, non sono disponibili nella ENEA-GRID che effettua controlli nella sola sfera dell'autenticazione

.

Nell'introduzione di un tale componente si possono seguire due strade:

- il *policy manager* integra l'attuale interfaccia ENEA-GRID distribuita dai Citrix Presentation Server per fare in modo che l'attivazione del servizio sia disponibile solo per gli utenti autorizzati. In tale ottica si consideri che l'attuale interfaccia è sviluppata con tecnologia Java.
- l'interfaccia di accesso viene completamente ridisegnata in ambiente Windows anche in considerazione del fatto che quella attuale ha un *feel* abbastanza obsoleto. Tale approccio determina la necessità di analizzare in dettaglio il problema del trasferimento del token AFS da e verso sistemi Windows visto che la sottomissione delle applicazioni avverrebbe da sistemi Microsoft.

Si ritiene quindi opportuno, *in primis*, realizzare uno studio di fattibilità teso ad accertare la possibile portabilità in ambiente Microsoft delle routine *gettok/puttok* sviluppate dalla casa produttrice di LSF al fine di garantire alla routine di sottomissione dei job di LSF (*bsub* e *lrun*) un'autenticazione basata su quella di AFS.

Nel caso in cui tale studio di fattibilità sancisca una eccessiva complessità per il *porting* delle suddette procedure si renderà necessario progettare e realizzare un *policy manager* che sostituisca/integri il layer di autorizzazione attualmente presente nell'interfaccia di accesso all'ENEA GRID.