შესავალი GRID-ტექნოლოგიებში

GRID გამოთვლები არის განაწილებული გამოთვლების ფორმა, რომელიც შეიძლება წარმოვადგინოთ როგორც „ვირტუალური სუპერკომპიუტერი“, რომელიც წარმოდგენილია ქსელით დაკავშირებული კლასტერების სახით (ერთმანეთთან დაკავშირებული კომპიუტერები, რომლებიც მუშაობენ ერთად ამოცანების უზარმაზარი რაოდენობის შესასრულებლად). ეს ტექნოლოგია გამოიყენება სამეცნიერო ამოცანების ამოსახსნელად, რომლებსაც ჭირდებათ მძლავრი რესურსები. GRID გამოთვლები ასევე გამოიყენება კომერციულ ინფრასტრუქტურებში ისეთი შრომატევადი ამოცანების შესასრულებლად როგორიცაა ეკონომიკური პროგნოზირება, სეისმოანალიზი, კლიმატის მოდელირება, ცილის ფოლდინგის, წამლის თვისებების დამუშავება და გამოკვლევა. მრავალპროცესორიან სუპერკომპიუტერებთან შედარებით შეიძლება გამოვყოთ GRID_ის უპირატესობები:

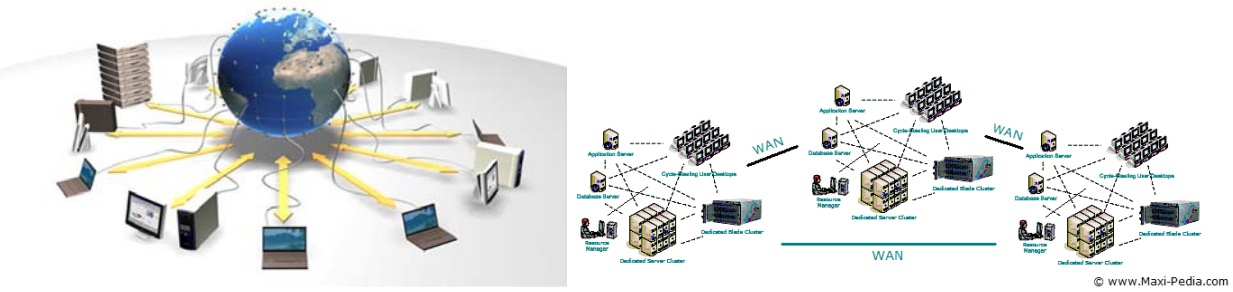
- შედარებით იაფია
- ხელმისაწვდომია ყოველდღე მთელი დღე-ღამის განმავლობაში
- ადვილად განახლებადია

ქსელური ორგანიზების თვალსაზრისით GRID წარმოადგენს შეთანხმებულ, გახსნილ და სტანდარტიზებულ სისტემას, რომელიც უზრუნველყოფს გამოთვლითი რესურსების მოქნილ, უსაფრთხო, კოორდინირებულ განაწილებას და ინფორმაციის შენახვას, რომელიც წარმოადგენს ვირტუალური ორგანიზაციის (Virtual Organization-VO ეს არის მომხმარებელთა საზოგადოება რომლებიც ერთობლივად გამოიყენებენ გამოთვლით რესურსებს იმ წესების მიხედვით, რომელიც დადგენილია რესურსების მფლობელებსა და მათ შორის მოქმედი შეთანხმების საფუძველზე. ყოველ ვირტუალურ ორგანიზაციას გააჩნია თავისი რეგისტრაციის ცენტრი) გარემოს ნაწილს. GRID ასევე წარმოადგენს პარალელური გამოთვლების სახეობას და გეოგრაფიულად განაწილებულ ინფრასტრუქტურას, რომელიც აერთიანებს სხვადასხვა ტიპის რესურსებს (პროცესორებს, მუდმივ და ოპერატიულ მეხსიერებებს, შემნახველებს და მონაცემთა ბაზებს), რომელთანაც წვდომა მომხმარებელს შეუძლია მსოფლიოს ნებისმიერი წერტილიდან, ისე რომ მას არ ჭირდება გამოყენებული რესურსების ადგილმდებარეობის ცოდნა. ხშირად მოჰყავთ მაგალითი: world wide web (WWW) უზრუნველყოფს ინფორმაციასთან წვდომას, რომელიც გეოგრაფიულად მთელ მსოფლიოში განაწილებულ მილიონობით სხვადასხვა სერვერებზეა განთავსებული. WWW-სგან განსხვავებით GRID არის ახალი გამოთვლითი ინფრასტრუქტურა რომელიც უზრუნველყოფს შეუფერხებელ და უსაფრთხო წვდომას გამოთვლით სიმძლავრეებთან, მონაცემთა შენახვის რესურსებთან, რომლებიც ასევე მთელ მსოფლიოშია განაწილებული.

GRID აღჭურვილია სტანდარტული ბიბლიოთეკებით თუმცა მომხმარებლების ამოცანების სფეციფიკიდან გამომდინარე შესაძლებელია მასზე სხვა არასტანდარტული პროგრამების და ბიბლიოთეკების დაყენება, მარტივად რომ ვთქვათ GRID-ზე შესაძლებელია

მოხდეს ნებისმიერი ამოცანის შესრულება, მთავარი მოთხოვნაა რომ ამოცანა განსაზღვრული იყოს ლინუქსის პლატფორმისთვის.

GRID ინფრასტრუქტურის მნიშვნელოვანი კომპონენტი არის შუალედური პროგრამული უზრუნველყოფა (middlware, ჩვენს შემთხვევაში EMI), რომლის დანიშნულებაა ამოცანების მართვა, დიდი მოცულობის მონაცემებთან უსაფრთხო წვდომა,მონაცემების სწრაფად გადაადგილება და ტირაჟირება ერთი გეოგრაფიული ადგილიდან მეორეში და დაშორებული ასლების სინქრონიზაციის ორგანიზება (ნახაზი 1).



ნახაზი 1

GRID-სერვისები

GRID დამოკიდებულია პროგრამულ უზრუნველყოფაზე რომელსაც middlware_ს უწოდებენ. იგი შეიძლება წარმოვიდგინოთ როგორც ინტერფეისი რესურსებსა და აპლიკაციებს შორის. GRID-ის სერვისები ეს არის პროგრამული შესაძლებლობების კრებული, აგრეთვე წესები და რეკომენდაციები მათი გამოყენების შესახებ. ესენია:

- User Interface, UI – მომხმარებლის ინტერფეისი
- Workload Management System,WMS – ამოცანათა მართვის სისტემა
- Logging and Bookkeeping, LB – შესრულებადი ამოცანების სტატუსის შემოწმების სერვისი
- Virtual Organization Management service, VOMS – ვირტუალური ორგანიზაციის მენეჯმენტის სერვისი
- Berkeley Database Information Index, BDII – ინფორმაციული სისტემა
- Computing Element, CE – გამომთვლელი ელემენტი
- Worker Nodes, WN – მუშა კვანძები
- Storage Element, SE – შემნახველი ელემენტი
- File catalog, LFC – ფაილების კატალოგი
- MyProxy– მინდობილობის ვადის გახანგრძლივების მექანიზმი

UI

მომხმარებლის GRID-ის რესურსებთან წვდომისთვის მომხმარებლის ინტერფეისი არის განკუთვნილი. მისი საშუალებით მომხმარებელს შეუძლია გაუშვას ამოცანა, გადააგზავნოს მონაცემები ერთი შემნახველი რესურსიდან მეორეზე, აკონტროლოს ამოცანის შესრულების პროცესი და მიიღოს შედეგი.

WMS

ამოცანათა მართვის სისტემის დანიშნულება არის ამოცანათა გაშვებაზე მოთხოვნების მიღება, შესაბამისი რესურსის მოძებნა (ანუ CE-ს და მასთან დაკავშირებული WN-ების), ამოცანის გაგზავნის შესასრულებლად, ანიჭებს მას სტატუსს და ახდენს მის შესრულებას და შედეგის მიღების კონტროლს. მომხმარებლის მიერ ამოცანის გაშვება ხდება მომხმარებელზე ორიენტირებული მაღალი დონის ენის (Job Description Language, JDL) საშუალებით. GRID-ში დასათვლელად დავალების გაგზავნამდე მისი აღწერა უნდა მოხდეს JDL-ენის საშუალებით. დავალების აღწერის ტიპიური ფაილი შეიცავს ინფორმაციას დავალების შესასრულებელი ფაილის მდებარეობაზე, შესავალი და გამოსავალი მონაცემების მდებარეობაზე და მოთხოვნებს რომლებსაც უნდა აკმაყოფილებდეს გამოთვლითი რესურსები (მინიმალური მეხსიერება, პროცესორის ტიპი და ა.შ.).

LB

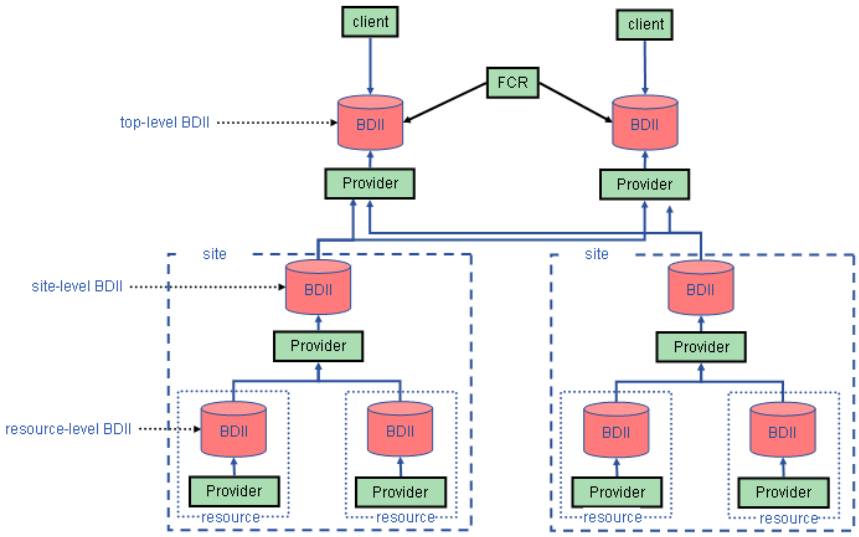
შესრულებადი ამოცანების სტატუსის შემოწმების სერვისი ამოწმებს WMS-ის მიერ მართული ამოცანის შესრულების პროცესს, აგროვებს WMS-ის სხვადასხვა კომპონენტებისგან შეტყობინებებს მოვლენათა შესახებ და ამუშავებს მათ, რათა წარმოადგინოს მიმდინარე ამოცანის სტატუსის ზოგადი მდგომარეობა.

VOMS

ვირტუალური ორგანიზაციის მენეჯმენტის სერვისი წარმოადგენს ინფორმაციას მომხმარებელთა ავტორიზაციისათვის. VOMS მონაცემების შესანახად იყენებს რელაციურ მონაცემთა ბაზას MySQL-ს ან ORACLE-ს. ის მართავს მომხმარებლის აუტენტიფიკაციასა და ავტორიზაციას და აძლევს საშუალებას ვირტუალური ორგანიზაციის წევრებს სხვადასხვა წვდომის უფლებებით გამოიყენონ GRID-ის რესურსები. ეს სერვისი შეიძლება შეიცავდეს მრავალ ვირტუალურ ორგანიზაციას. ის ანიჭებს მომხმარებელს უფლებებს მათი ვირტუალურ ორგანიზაციის წევრობის, ამ ორგანიზაციაში მომხმარებელთა ჯგუფის, როლის და ატრიბუტების მიხედვით. ყველა სერვისი შეიცავს VOMS კლიენტს. სერვისი მუშაობს ვარგისი სერტიფიკატის მქონე მომხმარებელთან. გააჩნია ინტერფეისი მომხმარებლის ვირტუალურ ორგანიზაციაში გაწევრიანების, ადმინისტრატორისთვის მომხმარებლის სამართავად და სხვადასხვა სერვისისთვის შენახული ინფორმაციის მოთხოვნის შემთხვევაში მის მისაწოდებლად (მომხმარებელთა სია, როლი და ა.შ.)

BDII

ინფორმაციული სისტემა წარმოგვიდგენს დეტალურ ინფორმაციას GRID სერვისებზე. მას აქვს სამ დონიანი იერარქიული სტრუქტურა: BDII core(resource level), BDII site და BDII top. BDII core აყენია GRID-ის კონკრეტულ სერვისთან ერთად და შეიცავს ინფორმაციას მის შესახებ. ყოველ GRID საიტზე გაშვებულია BDII site სერვისი. ის კრებს ინფორმაციას საიტის ყველა BDII core-დან. BDII top კრებს ინფორმაციას ყველა საიტის BDII site-დან. როგორც წესი VO-ში არის რამდენიმე BDII top-ი, რომ ერთ-ერთის დაზიანების შემთხვევაში მუშაობა არ შეფერხდეს და მოხდეს დატვირთვის გადანაწილება. ინფორმაციული სისტემის კლიენტი უგზავნის მოთხოვნას BDII top-ს რათა მიიღოს მისთვის საჭირო ინფორმაცია.

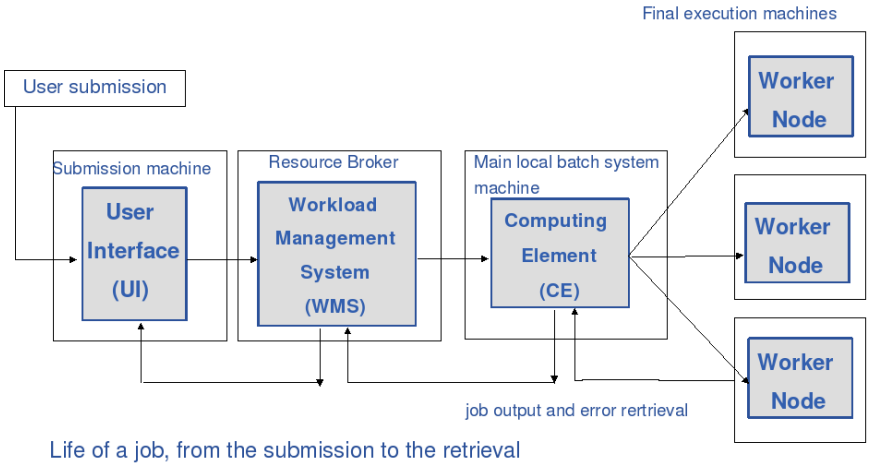


ნახაზი 2

BDII იღებს ინფორმაციას გაშვებული საინფორმაციო პროვაიდერებისგან. ეს არის სკრიპტები, რომლებიც იღებენ ინფორმაციას, აძლევენ მას LDAP (Lightweight Directory Access Protocol - პროტოკოლი ინტერნეტში განაწილებული დირექტორიების ინფორმაციის წვდომისა და შენახვისთვის) Data Interchange Format (LDIF) ფორმატს და გამოაქვთ შედეგი სტანდარტულ გამოსავალში (ნახ.2). ეს საინფორმაციო პროვაიდერები შეიძლება გამოყენებულ იქნეს სხვა BDII-სთვის მოთხოვნის გასაგზავნათ. Generic Information Provider (GIP) არის პროგრამა, რომელიც ამარტივებს საინფორმაციო პროვაიდერების შექმნას და ჩამოყალიბებას. ის მცირე ფლაგინების გამოყენების საშუალებას იძლევა, რაც აიოლებს ახალ სისტემებთან თავსებადობას. ინფორმაციული სისტემის ინფმორმაცია შეესაბამება სქემას რომელსაც GLUE სქემას უწოდებენ. Freedom of Choice for Resources mechanism (FCR) გამოიყენება BDII top-ში რაც აძლევს VO-ს საშუალებას გავლენა მოახდინოს მათი კონკრეტული სერვისის გამოყენებაზე. FCR პორტალი აყალიბებს VO-სთვის ხელმისაწვდომი სერვისების სიას. პორტალი შეიძლება VO მენეჯერის მიერ გამოყენებული იქნას საიტების თეთრი და შავის სიის შესადგენად (ანუ სანდო და საეჭვო საიტები). ამ ინფორმაციას შემდეგ იყენებს BDII top.

CE

გამომთვლელი ელემენტი არის პასუხიმგებელი გამომთვლელ რესურსებზე. მისი ძირითადი ფუნქცია არის ამოცანათა მენეჯმენტი (გაშვება, კონტროლი და ა.შ.). მომხმარებელს შეუძლია ინტერაქცია უშუალოდ მასთან ან WMS-თან, რომელიც გადასცემს ამოცანას შესაბამის CE-ს რომელსაც იპოვის matchmaking პროცესის საშუალებით. მოცანათა გადაცემისას CE-ს შეუძლია იმუშავოს push მოდელით (სადაც ამოცანა შესასრულებლად გადეცემა პირდაპირ CE-ს) ან pull მოდელით (სადაც CE უგზავნის თხოვნას WMS-ს რათა ამ უკანასკნელმა გადასცეს მას ამოცანა). გარდა ამოცანათა მენეჯმენტის ფუნქციისა, CE-მა უნდა წარმოადგინოს ინფორმაცია რომელიც განსაზღვრავს მის მფლობელობაში მყოფ რესურსებს. push მოდელის შემთხვევაში ეს ინფორმაცია ქვეყნდება ინფორმაციულ სისტემაში და გამოიყენება match making მიერ, რომელიც ძებნის საჭირო რესურსებს ამოცანების რიგში. pull მოდელის შემთხვევაში ეს ინფორმაცია ჯდება CE availability მესიჯში და გადაეცემა WMS-ს CE-გან. ამის შემდეგ matchmaker იყენებს ამ ინფორმაციას CE-სათვის შესაბამისი ამოცანის საპოვნელად (ნახ. 3).



Life of a job, from the submission to the retrieval

ნახაზი 3

WN

GRID გარემოს თვალთახედვით მუშა კვანძები წარმოადგენს CE-ს შემადგენელ ნაწილს. WN იმართებიან CE-ს მიერ. პროცესის განაწილების და გამოთვლის დეტალები მომხმარებლისთვის დამალულია, მაგრამ რეალურად WN ახორციელებენ გამოთვლებს, ამიტომ მათზე უნდა დაყენდეს გამომთვლელი პროგრამები.

SE

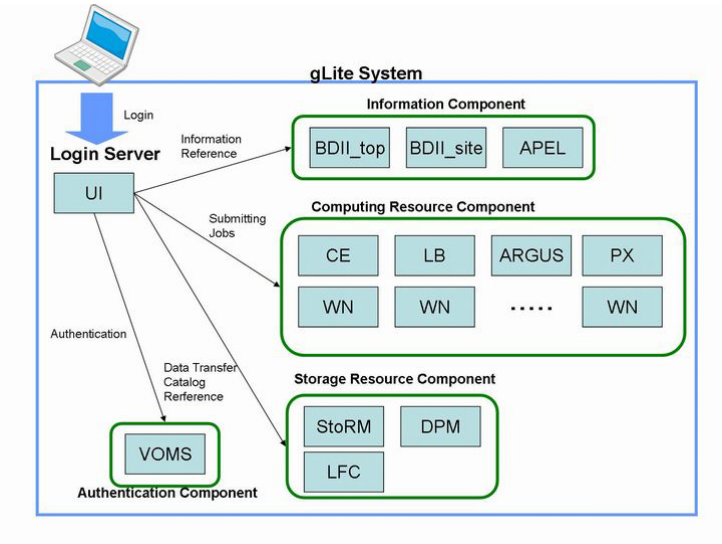
შემნახველი ელემენტი უზრუნველყოფს ერთგვაროვან წვდომას ნებისმიერ მონაცემთა შემნახველთან. ზოგადად SE-ს შეუძლია მართოს ჩვეულებრივი დისკის სერვერები, დისკების მასივებიანი ლენტური შემნახველი სისტემები (MSS). ეს ელემენტი მალავს მომხმარებლისგან მონაცემთა შემნახავის დეტალებს და უზრუნველყოფს მომხმარებლის ერთგვაროვან წვდომას მონაცემებზე.

SE შეიძლება გააჩნდეს მონაცემთა წვდომის სხვადასხვა პროტოკოლი და ინტერფეისი, მაგალითისთვის ლოკალური და დაშორებული პროტოკოლები: GSIFTP (a GSI-secure FTP), RFIO, (gsi)dcap, xroot, NFS 4.1. შენახვის რესურსის უმეტესობა იმართება Storage Resource Manager (SRM)-ის მიერ, რომელიც არის შუალედური სერვისი და იძლევა საშუალებას დისკიდან ლენტზე ფაილის ტრანსფერის, სივრცის დარეზერვების და ა.შ. მიუხედავად ამისა SRM იმპლემენტაცია სხვადასხვა შენახვის სიტემებსთვის შეიძლება განსხვავდებოდეს და გვთავაზობს სხვადასხვა შესაძლებლობებს.

არსებობს სხვადასხვა ტიპის შემნახველი სისტემები:

- Disk Pool Manager (DPM), რომელიც გამოიყენება მხოლოდ დისკების სისტემისთვის.
- CASTOR შექმნილია დიდ მასშტაბიანი MSS (ინტერფეისული დიკებით და შიდა ლენტური შემნახველებით) სამართავად.
- DCACHE განკუთვლია MSS და დისკის მასივების სიტემისთვის.
- არსებობს ასევე StoRM და BestMAN.

Type	Resources	File transfer	File I/O	SRM
CASTOR	MSS	GSIFTP	Insecure RFIO Yes	Yes
dCache	Disks/MSS	GSIFTP	gsidcap	Yes
DPM	Disks	GSIFTP	secure RFIO	Yes
StoRM	Disks/MSS	GSIFTP	direct file access	Yes
GrifFTP server	Disks	GSIFTP		No
Xrootd	Disks	xrootd	xrootd	No



ნახაზი 4

GRID-სისტემის უსაფრთხოება

GRID-სისტემაში მომხმარებლის შესვლა საკმაოდ მარტივია, თუმცა ამისათვის რამდენიმე მნიშვნელოვანი ამოცანის გადაჭრაა საჭირო.

პირველი ამოცანაა როგორ დავშიფროთ ის ინფორმაცია რომელიც გადაეცემა GRID-ს, განსაკუთრებით ის პარამეტრები რომლებიც დაკავშირებულია GRID-სისტემაში შესვლასთან. ამ ამოცანის გადასაწყვეტად გამოიყენება ასიმეტრიული დაშიფვრის ტექნოლოგია. ასიმეტრიული სისტემები ისეა მოწყობილი, რომ მათში არსებობს ორი რიცხვი:

- A მომხმარებლის ღია გასაღები (PublicKey), რომელიც ყველასათვის ხელმისაწვდომია და გამოიყენება A-მომხმარებლისათვის განკუთვნილი შეტყობინების დასაშიფრად.
- A მომხმარებლის ჩაკეტილი გასაღები (PrivateKey), რომელიც გამოიყენება მისთვის გამოგზავნილი შეტყობინებების გასაშიფრად.

ყოველი Private Key არის სპეციალურად დაგენერირებული რიცხვი, რომელიც გამოიყენება საიდენტიფიკაციოდ როგორც პაროლი. Private Key დაცული უნდა იყოს. იგი მათემატიკურად დაკავშირებულია Public Key-სთან. ფორმალურად Public Key-სგან შეიძლება Private Key-ის გამოთვლა, მაგრამ პრაქტიკულად ამას ძალიან დიდი დრო ჭირდება (გასაღების ზომის გაზრდით დრო ექსპონენციალურად იზრდება), ხოლო Private Key-დან Public Key-ს გამოთვლა მარტივია. ამის გამო მათ ასიმეტრიულ გასაღებებსაც უწოდებენ. ეს მეტად მნიშვნელოვანია, რადგან მომხმარებელს შეუძლია გამოაქვეყნოს თავისი ღია გასაღები იმისათვის რომ ნებისმიერმა შეძლოს მისთვის შეტყობინების დაშიფვრა და გამოგზავნა.

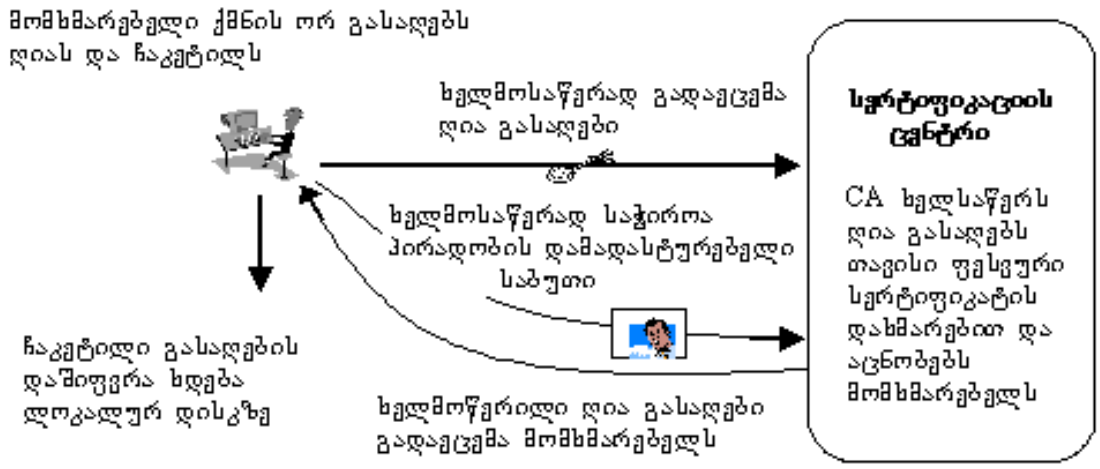
მომხმარებელმა სხვებისაგან საიდუმლოდ უნდა შეინახოს თავისი ჩაკეტილი გასაღები, რომ მხოლოდ თვითონ შეძლოს მისთვის გამოგზავნილი შეტყობინებების გაშიფვრა. აქვე ავღნიშნოთ, რომ ჩაკეტილი გასაღების გასაიდუმლება იმიტომაცაა საჭირო, რომ მხოლოდ მომხმარებელს უნდა შეეძლოს ციფრული ხელმოწერის შესრულება. ეს ხელმოწერა კი დამოკიდებულია ჩაკეტილი გასაღების მნიშვნელობაზე.

მომხმარებელს შეუძლია შექმნას ციფრული ხელმოწერა, რომელიც დასმული ციფრულ დოკუმენტზე, იძლევა იმის გარანტიას, რომ ხელმომწერი დოკუმენტზე მომხმარებელია. სქემა შემდეგია: მომხმარებელი იყენებს რა თავის ჩაკეტილ გასაღებს, აწარმოებს გარკვეულ ოპერაციას მონაცემებზე, რომელზეც უნდა დასვას ხელმოწერა და გადაცეს რეზულტატი საწყის მონაცემებთან ერთად ნებისმიერ სხვა ობიექტს. ეს უკანასკნელი მომხმარებელ A-ს ღია გასაღების გამოყენებით იოლად დარწმუნდება, რომ ციფრული ხელმოწერა ნამდვილია.

უნდა აღინიშოს, რომ წყვილი გასაღები ვერ უზრუნველყოფს სათანადო უსაფრთხოების დაცვას. როგორ შეიძლება დარწმუნება იმაში რომ დაშიფრული ჩაკეტილი გასაღებით გადაცემული ინფორმაცია ნამდვილად ეკუთვნის აღნიშნულ მომხმარებელს.

GRID-ის უსაფრთხოების ინფრასტრუქტურა (Grid Security Infrastructure, GSI) უზრუნველყოფს უსაფრთხო მუშაობას საერთო მოხმარების ქსელებში და თავაზობს მომხმარებელს ისეთ სერვისებს როგორიცაა იდენტიფიკაცია, ინფორმაციის გადაცემის კონფიდეციალობა და GRID-ში ერთიანი შესვლა. ერთიან შესვლაში იგულისხმება, რომ მომხმარებელმა საჭიროა მხოლოდ ერთხელ გაიაროს იდენტიფიცირების პროცედურა შემდეგ კი ყველა რესურსებზე წვდომისათვის, რომელიც მას დაჭირდება სისტემა თვითონ იზრუნებს მის იდენტიფიცირებაზე. მომხმარებლის და სერვისების იდენტიფიცირებისათვის GSI-ში გამოიყენება ციფრული სერტიფიკატები X.509. ციფრული სერტიფიკატი არის სერტიფიკატის მქონე მომხმარებლის ღია გასაღები, რომელშიც ინტეგრირებულია პერსონალური ინფორმაცია, როგორიცაა მომხმარებლის სახელი, მისი ელექტრონული მისამართი, სამუშაო ადგილი, სერტიფიკატის მოქმედების დრო და ა.შ. გარდა ამისა სერტიფიკატი შეიცავს სასერტიფიკაციო ცენტრის ციფრულ ხელმოწერას (Certification Authority - CA), რომელიც ადასტურებს, რომ სერტიფიკატი ეკუთვნის იმ მომხმარებელს ვისი მონაცემებიც ჩაწერილია სერტიფიკატში. ციფრული ხელმოწერა ეს არის ინფორმაცია სერტიფიკატის შესახებ, რომელიც დაშიფრულია CA მიერ ჩაკეტილი გასაღებით. ამრიგად ამ ინფორმაციის წაკითხვა შესაძლებელია მხოლოდ CA-ს ღია გასაღების საშუალებით (ნახ.5).

სერტიფიკატის გაცემამდე (ხელმოწერამდე) CA-ს ვალდებულებაა შეამოწმოს სერტიფიკატის კუთვნილება მოცემულ მომხმარებელზე. ყველა სასერტიფიკაციო ცენტრი ატარებს საკუთარ პოლიტიკას, რომელიც განსაზღვრავს სერტიფიკატების შექმნის და ხელმოწერის წესებს.



ნახაზი 5

როგორც წესი სერტიფიკატის მისაღებად საჭიროა მომხმარებელი შევიდეს სასერტიფიკაციო ცენტრის Web-გვერდზე და შეავსოს ფორმა, რომელშიც მითითებული იქნება პერსონალური მონაცემები, ორგანიზაციის დასახელება და ზუსტი ელექტრონული მისამართი. ზოგიერთი სასერტიფიკაციო ცენტრი თხოულობს დამატებით ინფორმაციას მაგალითად შესასრულებელი პროექტების შესახებ, ჩასატარებელი სამეცნიერო სამუშაოების შესახებ და ა.შ. მაგრამ როგორ უნდა დაამტკიცოს პიროვნების იდენტობა სასერტიფიკაციო ცენტრმა, რომელიც სხვა ქალაქშია ანდა სულაც სხვა ქვეყანაში? ამისათვის უმეტეს მაშტაბურ GRID პროექტებში გათვალისწინებულია რეგისტრატორების ინსტიტუტის არსებობა (Registration Authority - RA) რომლებიც დაადასტურებენ სერტიფიკატის მქონე მომხმარებლის პერსონალურ მონაცემებს. ამიტომ სანამ მოხდება ციფრული სერტიფიკატის ხელმოწერა CA უკავშირდება შესაბამის RA-ას და მისგან იღებს დასტურს, რომ მოთხოვნა სერტიფიკატის მიღებაზე ჭეშმარიტია.

Public Key ინახება სპეციფიურ ფორმატში რომელიც ცნობილია როგორც X.509 სერტიფიკატი. სერტიფიკატის მნიშვნელოვანი პარამეტრია Subject Name (SN), რომელიც ასე გამოიყურება:

/C=UK/O=eScience/OU=CLRC/L=RAL/CN=John Doe

ეს არის უფრო ზოგადი ფორამტის Distinguished Name (DN)-ის ნაწილი, რომელიც ფართოდ გამოიყენება GRID-ში. არ არსებობს DN-ის ჩაწერის მიღებული სტანდარტი, მაგრამ ითვლება რომ მას სულ მცირე უნდა გააჩნდეს უნიკალური სახელი, ხოლო SN სერტიფიკატს მფლობელის სახელი.

სერტიფიკატი ასევე შეიცავს სხვა ინფორმაციასაც, მაგალითად დროს რომლის შემდეგაც მას გასდის გამოყენების ვადა. მომხმარებლის სერტიფიკატის ნორმალური ვადა ერთი წელია, რომლის გასვლამდეც ის უნდა განახლდეს. განახლებული სერტიფიკატს აქვს ახალი Private and Public Key, მაგრამ SN იგივე რჩება. Private Key-ს მოპარვის შემთხვევაში შეიძლება მისი ცნობილი სერტიფიკატების სიაში შეტანა რომლებიც მოიაზრება როგორც უვარგისი.

იდენტობის დადასტურების შემდეგ გენერირდება სერტიფიკატი და ეგზავნება მომხმარებელს. ეს შეიძლება განხორციელდეს ელ-ფოსტით ან მომხმარებელმა შესაბამისი ინსტრუქციის მიხედვით უნდა გადმოწეროს ის საიტიდან. თუ სერტიფიკატი უშუალოდ დაინსტალირდება ბროუზრში უნდა მოხდეს მისი შენახვა დისკზე. თუ როგორ მოხდება ეს ბროუზერის სპეციფიკაციაზეა დამოკიდებული.

გამოგზავნილი სერტიფიკატი მიიიღება ჩვეულებრივ PEM (გაფართოება .pem) ან PKCS12 (გაფართოება .p12 ან .pfx) ფორმატში. შეიძლება მის ერთი ფორმატიდან მეორეში გადაყვანა openssl ბრძანებით.

თუ სერტიფიკატი PKCS12 ფორმატშია ის შეიძლება გადავიყვანოთ PEM-ში:

```
$ openssl pkcs12 -nocerts -in my_cert.p12 -out userkey.pem
$ openssl pkcs12 -clcerts -nokeys -in my_cert.p12 -out usercert.pem
```

სადაც:
my cert.p12 არის გადასაყვანი PKCS12 ფორმატის ფაილი;
userkey.pem არის მიღებული private key ფაილი;
usercert.pem არის მიღებული PEM სერტიფიკატის ფაილი.
userkey.pem და usercert.pem ფაილი უნდა დაკოპირდეს UI-ში.

ამოცანის აღწერის ენა JDL

JDL არის მაღალი დონის ენა რომელიც გამოიყენება ამოცანების აღწერისთვის. მომხმარებელმა უნდა აღწეროს მისი ამოცანა, დავალების აღწერის ტიპიური ფაილი შეიცავს ინფორმაციას დავალების შესასრულებელი ფაილის მდებარეობაზე, შესავალი და გამოსავალი მონაცემების მდებარეობაზე და მოთხოვნებს რომლებსაც უნდა აკმაყოფილებდეს გამოთვლითი რესურსები (მინიმალური მეხსიერება, პროცესორის ტიპი და ა,შ.) და მიიღოს გამოსავალი ფაილი როცა ამოცანა დასრულდება. აქ შევეხებით მოკლედ ენის სინტაქსს და მოვიყვანთ რამდენიმე მაგალითს. ამოცანის განმსაზღვრელი ფაილი არის JDL გაფართოების ფაილი რომელიც შედგება ასეთი ფორმატის ხაზებისგან:

attribute = expression;

გამოსახულება შეიძლება შედგებოდეს რამოდენიმე სტრიქონისაგან, მაგრამ მხოლოდ ბოლო სტრიქონი მთავრდება წერტილ-მძიმეთი. ლიტერალური სტრიქონი მოთავსებულია ორმაგ ბრჭყალებში. თუ სტრიქონი თვითონაც შეიცავს ბრჭყალებს ისინი უნდა გამოიყოს backslash-ით (მაგ: Arguments = "\"hello\" 10"). ერთ ხაზიან კომენტარს უნდა წინ უძღვოდეს # ან // სიმბოლო, ხოლო მრავალხაზიანი კომენტარი უნდა მოთავსებული იყოს შემდეგ /* */ სიმბოლოებს შორის.

ამოცანის განსაზღვრა რომელიც უშვებს hostname ბრძანებას WN-ზე უნდა დაიწეროს ასე:

```
Executable = "/bin/hostname";
StdOutput = "std.out";
StdError = "std.err";
```

Executable ატრიბუტი განსაზღვრავს ბრძანებას რომელიც უნდა გაუშვას ამოცანამ. თუ ბრძანება უკვე არის WN-ზე ის უნდა გამოისახოს როგორც სრული გზა (path); თუ ის უნდა გადაკოპირდეს UI-დან მაშინ მხოლოდ ფაილის სახელით უნდა გამოისახოს და მისი გზა UI-ზე უნდა მცემული უნდა იყოს InputSandbox ატრიბუტით. მაგალითად:

```
Executable = "test.sh";
InputSandbox = {"/home/doe/test.sh"};
StdOutput = "std.out";
StdError = "std.err";
```

ატრიბუტი Arguments შეიძლება შეიცავდეს სტრიქონს, რომელიც მოცემულია როგორც არგუმენტების სია Executable-სთვის:

```
Arguments = "fileA 10";
```

Executable და Arguments ატრიბუტებში შესაძლოა აუცილებელი იყოს ისეთი სპეციალური სიმბოლოების გამოყოფა როგორებიცაა &, \, |, >, <. ამ შემთხვევაში (თუ ისინი არიან ფაილის სახელის ნაწილი) მათ წინ უნდა უძღოდეთ შემდეგი სიმბოლო \.

StdOutput და StdError ატრიბუტები განსაზღვრავენ executable-ს სტანდარტული გამოსავლის და შეცდომების შემცველ ფაილებს. სტანდარტული შესავალი ფაილის განსასაზღვრად შესაბამისად StdInput ატრიბუტი გამოიყენება:

```
StdInput = "std.in";
```

თუ ფაილები უნდა გადაკოპირდეს UI-დან WN-ზე ისინი უნდა ჩაიწეროს InputSandbox ატრიბუტში:

```
InputSandbox = {"test.sh", "fileA", "fileB", ...};
```

მხოლოდ ფაილს რომელიც განისაზღვრება როგორც Executable აქვს ავტომატურად execution ბიტი ჩართული. თუ სხვა ფაილებს, რომლებიც მითითებულია InputSandbox-ში და აქვთ ასეთი ტიპის ბიტი, WN-ზე გადაკოპირებისას კარგავენ მას. ამოცანის დასრულების შემდეგ ფაილები რომელთა ტრანსფერი ხდება WN-დან უკან UI-ზე განსაზღვრულია OutputSandbox ატრიბუტით:

```
OutputSandbox = {"std.out", "std.err"};
```

სადაც ფაილების სახელები არის ფარდობითი გზა ამოცანის სამუშაო დირექტორიის მიმართ (ანუ მიმდინარე დირექტორია როდესაც შემსრულებელი ფაილი იშვება).

ჯგუფური სიმბოლოს (Wildcards) გამოყენება დაშვებულია მხოლოდ InputSandbox ატრიბუტში. ფაილების სია input sandbox-ში ფარდობითია UI-ზე მიმდინარე დირექტორიის მიმართ. OutputSandbox ატრიბუტში აბსოლუტური გზის განსაზღვრა შეუძლებელია. InputSandbox ატრიბუტი არ შეიძლება შეიცავდეს ორი ერთიდაიგივე დასახელების ფაილს,

მიუხედავათ მათი განსხვავებული აბსოლუტური გზისა, რადგან ისინი გადაეწერებიან ერთმანეთს WN-ზე.

ამოცანის shell გარემო შეიძლება შეიცვალოს Environment ატრიბუტით. მაგალითად:

```
Environment = {"CMS_PATH=$HOME/cms", "CMS_DB=$CMS_PATH/cmdb"};
```

საბოლოოდ, ტიპიური JDL ფაილი მარტივი ამოცანისთვის გამოიყურება ასე:

```
Executable = "test.sh";
Arguments = "fileA fileB";
StdOutput = "std.out";
StdError = "std.err";
InputSandbox = {"test.sh", "fileA", "fileB"};
OutputSandbox = {"std.out", "std.err"};
```

სადაც test.sh ფაილი შეიძლება გამოიყურებოდეს ასე:

```
#!/bin/sh
echo "First file:"
cat $1
echo "Second file:"
cat $2
```

Requirements ატრიბუტი შესაძლოა გამოყენებულ იქნეს ამოცანის გასაშვებად საჭირო რესურსების მითითებით. მისი ცვლადი არის ლოგიკური ტიპის (Boolean) გამოსახულება რომელიც იღებს ნამდვილ (true) მნიშვნელობას ამოცანისთვის რომელიც უნდა გაიშვას რომელიმე CE-ზე. ამ მიზნით BDII-ს ყველა GLUE ატრიბუტის გამოყენება შეიძლება, წინ other.-ის დაწერით. მხოლოდ ერთი Requirements ატრიბუტის გამოყენება შეიძლება. რამდენიმეს დაწერის შემთხვევაში მხოლოდ ბოლო გამოიყენება. თუ საჭიროა ამოცანაში რამდენიმე პირობის გათვალისწინება მაშინ უნდა მოხდეს მათი კომბინირება ერთ Requirements ატრიბუტში.

მაგალითად, ვთქვათ მომხმარებელს უნდა გაუშვას ამოცანა CE-ზე PBS-ის როგორც batch სისტემის გამოყენებით და რომლის WN-ებს აქვთ მინიმუმ ორი პროცესორი. მაშინ მას შეუძლია დაწეროს:

```
Requirements = other.GlueCEInfoLRMSType == "PBS" && other.GlueCEInfoTotalCPUs > 1;
```

WMS-ს ასევე შეიძლება მოეთხოვოს ამოცანის გაგზავნა CE-ის კონკრეტულ ამოცანათა რიგში:

```
Requirements = other.GlueCEUniqueID == "lxshare0286.cern.ch:2119/jobmanager-pbs-short";
```

ან CE-ის ამოცანათა ნებისმიერ რიგში:

```
Requirements = other.GlueCEInfoHostName == "lxshare0286.cern.ch";
```

თუ ამოცანის შესრულების ხანგრძლივობის მითითება მნიშვნელოვანია, მაშინ საჭიროა მაქსიმუმი CPU დროის ან wallclock დროის (წუთებში) მითითება. მაგალითად თუ საჭიროა რვა CPU საათი და 12 wallclock საათი ეს შეიძლება ასე გამოისახოს:

Requirements = other.GlueCEPolicyMaxCPUTime > 480 &&other.GlueCEPolicyMaxWallClockTime > 720;

თუ ამოცანამ გადააჭარბა იმ ამოცანათა რიგის დროის ლიმიტს რომელშიც ის არის გაშვებული მაშინ ის ნადგურდება batch სისტემის მიერ.

ამოცანისთვის საჭირო CPU დრო უკუპროპორციულია CPU სიჩქარის , რომელიც გამოისახება GlueHostBenchmarkSI00 ატრიბუტით. თუ მაგალითად ამოცანას ჭირდება 720 წუთი 1000-ანი სიჩქარის CPU-ზე, მაშინ ვწერთ:

Requirements = other.GlueCEPolicyMaxCPUTime >(720 * 1000 / other.GlueHostBenchmarkSI00);

თუ ამოცანა უნდა გაიშვას CE-ზე სადაც VO-ს სპეციფიური პროგრამაა დაყენებული და ეს ინფორმაცია გამოქვეყნებულია CE-ს მიერ, მაშინ შეიძლება საჭირო იყოს მსგავსი რამის მითითება:

Requirements = Member("VO-cms-CMSSW_2_0_0",other.GlueHostApplicationSoftwareRunTimeEnvironment);

Member ოპერატორი გამოიყენება ტესტისთვის თუ მისი პირველი არგუმენტი (სკალარული ცვლადი) არის მისი მეორე არგუმენტის (სია) წევრი.
GlueHostApplicationSoftwareRunTimeEnvironment ატრიბუტი ფაქტიურად არის სტრიქონების სია და გამოიყენება VO-ს სპეციფიური ინფორმაციისთვის CE-თან დაკავშირებით (მაგალითად ინფორმაცია პროგრამმის შესახებ რომელიც აყენია შესაბამისი VO-ს CE-ზე).

ასევე შესაძლებელია რეგულარული გამოსახულების გამოყენება მოთხოვნილი რესურსების გამოსახვისას:
Requirements = RegExp("cern.ch", other.GlueCEUniqueID);

საწინააღმდეგო შეიძლება მოთხოვნილ იქნეს ასე:

Requirements = (!RegExp("cern.ch", other.GlueCEUniqueID));

მომხმარებელს შეიძლება დაჭირდეს განსაზღვრა თუ რა ტიპის CE იქნას გამოყენებული ამოცანის შესასრულებლად. აქ გვაქვს ორი მაგალითი თუ როგორ შეიძლება გამოისახოს JDL-ით მოთხოვნა კონკრეტულ ოპერაციულ სისტემაზე და კომპიუტერის არქიტექტურაზე:

```
Requirements = ( other.GlueHostOperatingSystemName == "CentOS" ||
RegExp("Scientific", other.GlueHostOperatingSystemName) ||
RegExp("Enterprise", other.GlueHostOperatingSystemName)
) &&
( other.GlueHostOperatingSystemRelease >= 5 &&
other.GlueHostOperatingSystemRelease < 6
```

) ;
ან ასე:

```
SN = other.GlueHostOperatingSystemName ;
SR = other.GlueHostOperatingSystemRelease ;

SL5 = ( SN == "CentOS" ||
RegExp("Scientific", SN) ||
RegExp("Enterprise", SN)
) &&( SR >= 5 && SR < 6 ) ;
Requirements = SL5 ;
```

ქვემოთ მოყვანილია Requirements ატრიბუტის მაგალითები სპეციფიური CE-ს არქიტექტურისთვის:

Intel i386-i686

```
Requirements = other.GlueHostArchitecturePlatformType is UNDEFINED ||
RegExp("i[3456]86",other.GlueHostArchitecturePlatformType) ;
```

64-bit Xeon and Opteron

```
Requirements = (other.GlueHostArchitecturePlatformType == "x86_64") ;
```

64-bit Itanium

```
Requirements = (other.GlueHostArchitecturePlatformType == "ia64") ;
```

თუ საჭიროა მოცემულ CE-ის უახლოეს SE-ზე ჩაიწეროს თვლის შედეგები, მაშინ საჭიროა დავწეროთ:

```
Member("srm.pic.es", other.GlueCESEBindGroupSEUniqueID);
```

შემდეგი მაგალითი გამოიყენება alice VO პროექტში CE-ს მოსაძებნად, რომელზეც დაყენებულია რაიმე პროგრამა (მაგ:VO-alice-AliEn and VO-alice-ALICE-v4-01-Rev-01) და ამოცანას აძლევს საშუალებას გაიშვას ერთი დღის განმავლობაში (ანუ ამოცანა არ შეწყდება დროის ამოწურვამდე):

```
Requirements = other.GlueHostNetworkAdapterOutboundIP==true &&
Member("VO-alice-AliEn", other.GlueHostApplicationSoftwareRunTimeEnvironment) &&
Member("VO-alice-ALICE-v4-01", other.GlueHostApplicationSoftwareRunTimeEnvironment) &&
(other.GlueCEPolicyMaxWallClockTime> 1440 );
```

შეიძლება ასევე WMS-მა ავტომატურად თავიდან გაუშვას ამოცანა, რომელიც GRID-ის მიერ იქნა შეწყვეტილი. არსებობს deep resubmission და shallowresubmission. deep resubmission-ია როდესაც ამოცანა წყდება WN-ზე გაშვებისთანავე, სხვა შემთხვევაში shallowresubmission.

მომხმარებელს შეუძლია შეზღუდოს WMS-ის მიერ ამოცანის თავიდან გაშვების რაოდენობა RetryCount და ShallowRetryCount ატრიბუტით:

```
RetryCount = 0;
ShallowRetryCount = 3;
```

ხანდახან უმჯობესია deep resubmission-ის გათიშვა, რადგან შეიძლება ამოცანა შეწყდეს მის მიერ რამის შესრულების შემდეგ (მაგალითად ფაილის შექმნა). shallowresubmission-ის გამოყენება ზრდის ამოცანის სწორად შესრულების შანსს.

WMS-ის პროქსი სერტიფიკატის განახლების თვისება ავტომატურად ჩართულია, რამდენადაც მომხმარებელი გრძელვადიან პროქსი სერტიფიკატს ინახავს default MyProxy სერვერზე (ჩვეულებრივ განისაზღვრება MYPROXY_SERVER გარემოს ცვლადი MyProxy კლიენტის ბრძანებებისთვის) და UI-ის კონფიგურაციაში WMS-ის ბრძანებებისთვის. მიუხედავად ამისა შესაძლებელია WMS-ზე სხვა MyProxy სერვერის მითითება JDL ფაილში:

MyProxyServer = "myproxy.cern.ch";

შესაძლოა პროქსი სერტიფიკატის განახლების გამორთვა:

MyProxyServer = "";

არჩევა თუ სად უნდა გაიშვას ამოცანა რამოდენიმე CE-ს შორის ხდება CE-ის რანგის მიხედვით (rank), რომლის სიდიდე გამოისახება ათწილადის სახით.

თუ რანგი ცხადად არ არის მითითებული მაშინ რანგი უდრის - other.GlueCEStateEstimatedResponseTime, სადაც სავარაუდო პასუხის დრო არის დროის ინტერვალი ამოცანის გადაგზავნასა და მისი გაშვების დასაწყისის შორის. მიუხედავად ამისა მომხმარებელს შეუძლია თავიდან განსაზღვროს CE-ის რანგი Rank ატრიბუტით:

Rank = other.GlueCEStateFreeCPUs;

რომელიც რანგს განსაზღვრავს თავისუფალი CPU-ს მიხედვით. შეგვიძლია უფრო რთული გამოსახულებაც დავწეროთ:

Rank = (other.GlueCEStateWaitingJobs == 0 ? other.GlueCEStateFreeCPUs :
-other.GlueCEStateWaitingJobs);

აქ დამატებულია მოთხოვნა ისეთ CE-ზე სადაც საერთოდ არ არის გაშვებული ამოცანები ან გაშვებულია ყველაზე ნაკლები.

JDL ფაილში შესაბამისი პარამეტრების გამოყენებით დავალება შეიძლება გაიგზავნოს გამომთვლელ კვანძებზე რომლებსაც გააჩნიათ განსაზღვერული შემავალი ფაილების ასლები.

InputData = {"LF:file1.txt"};
ReplicaCatalog="ldap://lhc20.sinp.msu.ru:9011/rc=Testbed\Replica
Catalog,dc=lxshare0226,dc=cern,dc=ch";
DataAccessProtocol = {"file", "gridftp"};

პარამეტრი **Replica Catalog** საჭიროა მხოლოდ იმ შემთხვევაში როდესაც **InputData** შეიცავს ფაილის ლოგიკურ სახელს. მისი გამოყენება არ არის საჭირო თუ ფაილებს გააჩნიათ ფიზიკური სახელები. ანალოგიურად შეიძლება საიტების გამოყოფა განსაზღვრული შემნახველი ელემენტით (storage element):

OutputSE = "gppse05.gridpp.rl.ac.uk";

ხშირად მოსახერხებელია დავალების შესრულების დაწყებამდე რესურსების დავალებასთან თავსებადობის შემოწმება. ამისათვის გამოიყენება ბრძანება:

glite-wms-job-list-match <job.jdl>

მოცემული JDL ფაილისათვის იგი იძლევა რანგის მიხედვით დალაგებულ რესურსების ჩამონათვალს, რომლებიც აკმაყოფილებენ მოცემული დავალების პირობებს.

GRID სისტემაში შესვლა

GRID რესურსებთან დაშვების მისაღებად საჭიროა მომხმარებელს ჰქონდეს ვარგისი პროქსი სერტიფიკატი UI კომპიუტერზე. GRID-ის გარემოში შესვლა ხორციელდება სერტიფიკატში მითითებული სახელით და კონტროლდება სპეციალური პროგრამით (ელექტრონული მინდობილობით - proxy), რომელიც იქმნება შეზღუდული დროთ მომხმარებლის პერსონალური (userkey.pem) გასაღების გამოყენებით. GRID სისტემის სერვისული სამსახურებს შეუძლიათ შეასრულონ ნებისმიერი მოქმედებები თუ მათ გააჩნიათ ასეთი მინდობილობის ასლი.

თუ სერტიფიკატი სწორედაა დაყენებული, მაშინ ბრძანება **grid-proxy-init** შექმნის ახალ მინდობილობას. პასუხად ეკრანზე გამოვა მსგავსი შეტყობინება:

```
$ grid-proxy-init
Youridentity: /C=FR/O=CNRS/OU=LAL/CN=CharlesLoomis/Email=loomis@lal.in2p3.fr
Enter GRID pass phrase for this identity: **********
Creating proxy ..................................................... Done
Your proxy is valid until Tue Aug 13 03:15:11 2002
```

განსაკუთრებული მითითების გარეშე მინდობილობა მოქმედებს 12 საათის განმავლობაში. მინდობილობის შექმნის შემდეგ, აუცილებლობის შემთხვევაში ის გამოიყენება GRID-ის სხვადასხვა ბრძანებების მიერ. თუ საჭიროა შეიძლება მინდობილობის შექმნა სხვა ხანგრძლივობით, ამისათვის გამოიყენება პარამეტრი **–hours**. უნდა გვახსოვდეს, რაც მეტია მინდობილობის ხანგრძლივობის დრო მით მეტია მათი გატეხვის ალბათობა.
თუ სერტიფიკატი არასწორედ იყო დაყენებული, გამოვა შეტყობინება შეცდომის შესახებ: “user certificate not found”, ხოლო თუ შეცდომაა საიდენტიფიკაციო ფრაზაში შეტყობინება – “wrong pass phrase”.

მინდობილობის ვადის გასვლამდე გასანადგურებლად შეიძლება შემდეგი ბრძანების გამოყენება: grid-proxy-destroy. მაგრამ ამ ბრძანებით განადგურდება მინდობილობის მხოლოდ ლოკალური ასლი. იგი არ შეეხება იმ ასლებს, რომლებიც გამოიყენებოდა თქვენი დავალების შესრულებისას და აგრეთვე GRID-ის სამსახურების მიერ.

გაცემული მინდობილობის შესახებ ინფორმაციის მისაღებად საჭიროა გამოვიყენოთ ბრძანება grid-proxy-info პარამეტრით –all, რომელიც იძლევა სრულ ინფორმაციას მინდობილობის შესახებ:

```
$ grid-proxy-info -all
subject  : /C=FR/O=CNRS/OU=LAL/CN=Charles Loomis/Email=loomis@lal.in2p3.fr/CN=proxy
issuer   : /C=FR/O=CNRS/OU=LAL/CN=Charles Loomis/Email=loomis@lal.in2p3.fr
type     : full
strength : 512 bits
timeleft : 11:36:17
```

გარკვეულ ელემენტებზე ინფორმაციის მიღება შესაძლოა ბრძანების პარამეტრების შესაბამისი მნიშვნელობების საშუალებით.

უფრო დაწვრილებით ყველა ამ ბრძანების შესახებ ინფორმაციის მიღება შესაძლოა თუ გამოვიყენებთ ბრძანებებთან ერთად პარამეტრს**--help.**

დავალებების გაგზავნის ბრძანებები

WMProxy წარმოადგენს Glite WMS-ის მთავარ სერვისს ამოცანების სამართავად. მისი რამდენიმე ფუნქცია:

- პროქსის დელეგაციით აჩქარებს თანმიმდევრულ ოპერაციებს;
- Match-making (რომელი CE შეესაბამება JDL ფაილით აღწერილს);
- ერთი ან რამოდენიმე ამოცანის გადაგზავნა შესასრულებლად;
- ამოცანების გაუქმება;
- ამოცანების შედეგის გამოტანა.

ბრძანება რომელიც მოითხოვს ინფორმაციას LB სერვისისგან შეიცავს ამოცანის მიმდინარე სტატუსს ან მისი სტატუსის ისტორიას.

შემდეგ ცხრილში წერია ძირითადი ბრძანებები, მათი ხშირად გამოყენებადი ობციებით:

ფუნქცია	Glite WMS
ამოცანის გადაგზავნა შესასრულებლად	glite-wms-job-submit [-d delegID] [-a] [-o joblist] „jdl ფაილი“
ამოცანის სტატუსის შემოწმება	glite-wms-job-status [-v verbosity] [-i joblist] „ამოცანის ID“
ამოცანის logging ინფორმაცია	glite-wms-job-logging-info [-v verbosity] [-i joblist] „ამოცანის ID“
ამოცანის შედეგის გამოტანა	glite-wms-job-output [-diroutdir] [-i joblist] „ამოცანის ID“
ამოცანის გაუქმება	glite-wms-job-cancel [-i joblist] „ამოცანის ID“
მისაწვდომი რესურსების სია	glite-wms-job-list-match[-d delegID] [-a] „jdl ფაილი“

პროქსი დელეგაცია	glite-wms-job-delegate-proxy -d „დელეგაციის ID“

ამოცანის გაგზავნა ხდება ამ ბრძანებით

glite-wms-job-submit -a „jdl ფაილი“

-a ობცია საჭიროა მომხმარებლის პროქსის WMProxy სერვერზე ავტომატური დელეგირებისთვის.
https://lhc20.sinp.msu.ru:7846/137.138.181.214/152312203546264?lhc20.sinp.msu.ru:7771

მიაქციეთ ყურადღება, რომ მინიჭებული ნომერი შეიცავს კითხვის ნიშანს, რომლებიც ზოგიერთ სისტემურ გარსებში (მაგალითად csh-ში) აღიქმება როგორც ნებისმიერი სიმბოლოების კრებული (wildcard). ამ გარსებისათვის საიდენტიფიკაციო ნომრები ბრჭყალებში უნდა ჩაისვას.

სხვა ბრძანებების მიერ ხდება საიდენტიფიკაციო ნომრის გამოყენება შესაბამისი მოქმედებების ჩასატარებლად ჩამოთვლილი დავალებებიდან. ბრძანება dg-job-output აგზავნის შესრულებული დავალების რეზულტატს რესურსების ბროკერიდან მომხმარებლის კომპიუტერზე. რეზულტატი შეიძლება მიღებული იქნას, როდესაც დავალებას ექნება სტატუსი Output Ready.

Proxy

დაშორებულ სერვისთან ურთიერთობისთვის სერტიფიკატი შეიძლება გამოვიყენოთ საიდენტიფიკაციოდ. მიუხედავად ამისა ხშირად დაშორებული სერვისისთვის აუცილებელია იმოქმედოს მომხმარებელის სახელით, მაგალითად ამ სერვერზე გაშვებულ ამოცანას შეიძლება დასჭირდეს სხვა სერვერებზე ფაილის ტრანსფერი და ამისთვის მას სჭირდება დაამტკიცოს რომ მას აქვს უფლება იმოქმედოს ამ მომხმარებლის სახელით (ამას უწოდებენ Delegation-ს). მეორე მხრივ Private Key-ს გადაგზავნა სხვა სერვერზე უსაფრთხოებიდან გამომდინარე არ არის მიზანშეწონილი.

გამოსავალი არის proxy სერტიფიკატის გამოყენება. proxy სერტიფიკატი შეიცავს ახალ Public/PrivateKey წყვილს ხელმოწერილს მომხმარებლის პერსონალურ სერტიფიკათთან ამის მსგავსი SN-ით:

/C=UK/O=eScience/OU=CLRC/L=RAL/CN=John Doe/CN=proxy

ჩვეულებრივ proxy-ს აქვს მცირე ვადა (სტანდარტულად 12 საათი). Proxy ფაილები შეიცავენ proxy სერტიფიკატს, მის Private Key-ს და მომხმარებლის სერტიფიკატს (მაგრამ არა მომხმარებლის Private Key-ს).

მომხმარებლის proxy-სთან იდენტიფიკაციის მექანიზმი იგივეა როგორც პერსონალური სერტიფიკატის შემთხვევაში, მაგრამ დამატებით მოწმდება პროქსის ხელმოწერა მომხმარებლის სერტიფიკატთან (ჩართულია ფაილში).

უსაფრთხოების თვალსაზრისით proxy სუსტია. რადგან Private Key იგზავნება passphrase-ს გარეშე, ადვილია მისი მოპარვა. ასევე არ არსებობს proxy-ს გაუქმების მექანიზმი რომ შეუძლებელი იყოს მისი გამოყენება, ამიტომაც მას აქვს 12 საათიანი ვადა, რაც ამცირებს პოტენციური ზარალის რისკს.

VOMS Proxy

VOMS სისტემა მართავს ინფორმაციას, ორგანიზაციის შიგნით მომხმარებლის როლის და პრივილეგიების შესახებ. პროქსის შექმნისას რამდენიმე VOMS სერვერი ეკონტაქტება ერთმანეთს და ისინი აბრუნებენ "mini certificate"-ს რომელიც ცნობილია როგორც Attribute Certificate (AC). AC შეიცავს რა ინფორმაციას მომხმარებლის ჯგუფის წევრობისა და ამ ორგანიზაციის შიგნით როლის შესახებ.

VOMSproxy-ს შესაქმნელად AC ჩაშენებულია სტანდარტულ proxy-ში და ეს მთლიანად ხელმოწერილია ძირითადი სერთიფიკატის private key-თ. სერვისები შემდეგ შიფრავენ VOMS ინფორმაციას და იყენებენ დანიშნულებისამებრ, ანუ მომხმარებელს ეძლევა რაიმეს გაკეთების უფლება იმის მიხედვით თუ მას რა როლი აქვს შესაბამის VO-ში. ამ მეთოდიდან გამომდინარე VOMS ატრიბუტები შეიძლება გამოყენებულ იქნეს პროქსისთან ერთად და ისინი არ შეიძლება მიებას CA გაცემულ სერტიფიკატებს.

ყოველ AC-ს აქვს თავისი ვადა. ეს ვადა proxy-სთვის ჩვეულებრივ 12 საათია, მაგრამ AC-სთვის შესაძლებელია ამ დროის შეცვლა.

MyProxy

პროქსის შეზღუდული დრო ქმნის პრობლემას. თუ ამოცანა არ დამთავრდა პროქსის ვადის ამოწურვამდე ამოცანა შეწყდება. ეს შეიძლება მოხდეს იმ სემთხვევაში თუ ამოცანის შესრულება იკავებს ძალიან დიდ დროს ან ის იმყოფება გაშვებული ამოცანების რიგში დიდი დროის განმავლობაში. ამ პრობლემის მარტივი გადაწყვეტაა გრძელვადიანი პროქსის დაშვება, მაგრამ ეს ქმნის დამატებით რისკებს უსაფრთხოებისთვის. ასევე პროქსის ვადა განსაზღვრულია VOMS სერვერზე ამიტომ მისი შეცვლა შეუძლია მხოლოდ ამ სერვერის ადმინისტრატორს.

ამ შეზღუდვის გვერდის ასავლელად არსებობს MyProxy სერვერი, სადაც იქმნება გრძელვადიანი პროქსი. ეს შეიძლება იყოს ცალკე სერვერი. WMS-ს შეუძლია გამოიყენოს ეს გრძელვადიანი პროქსი და პერიოდულად განაახლოს იგი გაშვებული ამოცანისთვის, სანამ ეს ამოცანა დასრულდება ან გრძელვადიანი პროქსის დრო ამოიწურება.

ავტომატური განახლების მექანიზმის ჩართვისათვის საჭიროა შესრულდეს შემდეგი მოქმედებები:

1. შეიქმნას მინდობილობა grid-proxy-init-ის გამოყენებით.

2. დარეგისტრირდეს ეს მინდობილობა proxy-server-ზე შემდეგი ბრძანებით myproxy-init –s <server> [-t <cred> -c <proxy>], სადაც “server” ეს არის სერვერის მისამართი, “cred” – დრო (საათების რაოდენობა), რომლის განმავლობაშიც სერვერზე უნდა მოქმედებს მინდობილობა, ხოლო “proxy” - დრო (საათების რაოდენობა), რომლის განმავლობაშიც უნდა მოქმედებდეს განახლებული მინდობილობა.

3. დავალებები ხანმოკლე მინდობილობებით გაეშვება შემდეგი ბრძანების საშუალებით:

grid-proxy-init –hours<hours>.

ინფორმაცია შენახული მინდობილობების შესახებ შეიძლება მიღებული იქნეს შემდეგი ბრძანებით:

myproxy-info –s <server> -d,

ხოლო მინდობილობის განადგურება შესაძლებელია ბრძანებით

myproxy-destroy–s<server>.

მაგალითი 1. Hello from Script

ამ მაგალითში GRID გარემოში ხდება მცირე დავალების სკრიპტის გაგზავნა, ის სრულდება და რეზულტატი ეგზავნება მომხმარებელს.

შევქმნათ შესასრულებელი ფაილი დასახელებით HelloScript.sh, რომელიც შეიცავს შემდეგ სკრიპტს:

```
#!/bin/sh
/bin/echo "Hello From Script"
/bin/ls 9485968.txt
```

ეს სკრიპტი მოცემულ ფრაზას (ამ შემთხვევაში "HelloFromScript") ჩაწერს სტანდარტულ გამოსავალ ფაილში, ხოლო შეცდომას სტანდარტული შეცდომების ფაილში-9485968.txt (თუ კომპიუტერში რომლიდანაც გაიშვა დავალება არ არის ფაილი 9485968.txt)

JDL ფაილს ამ დავალებისათვის აქვს შემდეგი სახე:

```
Executable    = "HelloScript.sh";
StdOutput     = "std.out";
StdError      = "std.err";
InputSandbox  = {"HelloScript.sh"};
OutputSandbox = {"std.out","std.err"};
```

თუ სკრიპტი HelloScript.sh არ არის მოთავსებული მიმდინარე დირექტორიაში, მაშინ აუცილებელია მიეთითოს სრული გზა სკრიპტისაკენ.

თუ დავალება სწორედაა შესრულებული მაშინ გამოსავალი ფაილი შედგება ერთი სტრიქონისაგან

HelloFromScript

ხოლო შეცდომების ჩაწერის ფაილი-სტრიქონისაგან:

/bin/ls: 9485968.txt: No such file or directory

მაგალითი 2. დავალების შესრულებისათვის აუცილებელი მოთხოვნების სპეციფიკაცია.

სპეციფიკაციის მოთხოვნების საშუალებით, მომხმარებელს შეუძლია გააგზავნოს დავალება იმ საიტზე რომელიც აკმაყოფილებს აღნიშნულ მოთხოვნებს და რომელიც აუცილებელია დავალების შესასრულებლად. არასრულ სპეციფიკაციას შეუძლია გამოიწვიოს დავალების შესრულების გაჩერება და დროის კარგვები.

დავალების აღწერის JDL ფაილში სპეციფიკაციისათვის გამოიყენება ატრიბუტი "Requirements". ამ ატრიბუტის მნიშვნელობებს წარმოადგენენ ლოგიკური გამოსახულებები, რომელთა საშუალებით მიიღება საჭირო შეზღუდვები.

შესაძლო მნიშვნელობები (ან ცვლადები) რომლებიც გამოიყენება მოთხოვნების აღწერისათვის, განისაზღვრება გამოთვლითი ელემენტის ატრიბუტებით. ისინი განისაზღვრება GRID-ის საინფორმაციო სამსახურების საშუალებით.

```
ldapsearch -x \
        -Hldap://lxshare0225.cern.ch:2135 \
-b 'mds-vo-name=edg,o=grid' \
       '(objectclass=computingelement)'
```

ტიპიური ატრიბუტებია: "Architecture", "OpSys", "RunTimeEnvironment", "MaxCPUTime", "MaxWallClockTime" и "FreeCPUs".

მაგალითად თუ დავალების მოთხოვნაა, რომ CPU-ს სამუშაო დრო იყოს 25 წუთი და რეალური დრო 100 წუთი, ამისათვის საჭიროა შემდეგი სპეციფიკაციის მოთხოვნა:

```
Requirements = other.MaxCPUTime>=1500 &&other.MaxWallClockTime>=6000;
```

დრო მოცემულია წამებში. აუცილებელია ავღნიშნოთ, რომ რიცხვითი მნიშვნელობები არ ისმება ბრჭყალებში. მათი ბრჭყალებში ჩასმა გამოიწვევდა რესურსების არასწორ არჩევას.

ატრიბუტი "RunTimeEnvironment" გამოიყენება პროგრამული საშუალებების აღწერისათვის, რომლებიც უნდა იქნეს დაყენებული კვანძზე დავალების შესასრულებლად. მაგალითად მოთხოვნა

```
Requirements  =Member(other.RunTimeEnvironment,"ALICE-3.07.01");
```

გამოყოფს კვანძებს რომლებზეც დაყენებულია პროგრამული უზრუნველყოფა "ALICE-3.07.01".

მაგალითი 3. RankingResources

თუ GRID ქსელის რამდენიმე კვანძი აკმაყოფილებს დავალების მოთხოვნებს, მაშინ ის გაიგზავნება უმაღლესი რანგის მქონე რესურსზე. თუ ატრიბუტი "Rank" არ არის მითითებული JDL ფაილში, მაშინ გამოყენებული იქნება:

Rank = -other.EstimatedTraversalTime;

EstimatedTraversalTime პარამეტრის მნიშვნელობა იმ შეფასებული დროის ტოლია (წამებში) რომელიც გაივლის დავალების გაშვებამდე. ასეთი მითითება ყოველთვის ოპტიმალური ვერ იქნება და მომხმარებელს შეუძლია გამოიყენოს სხვა კრიტერიუმები, მაგალითად:

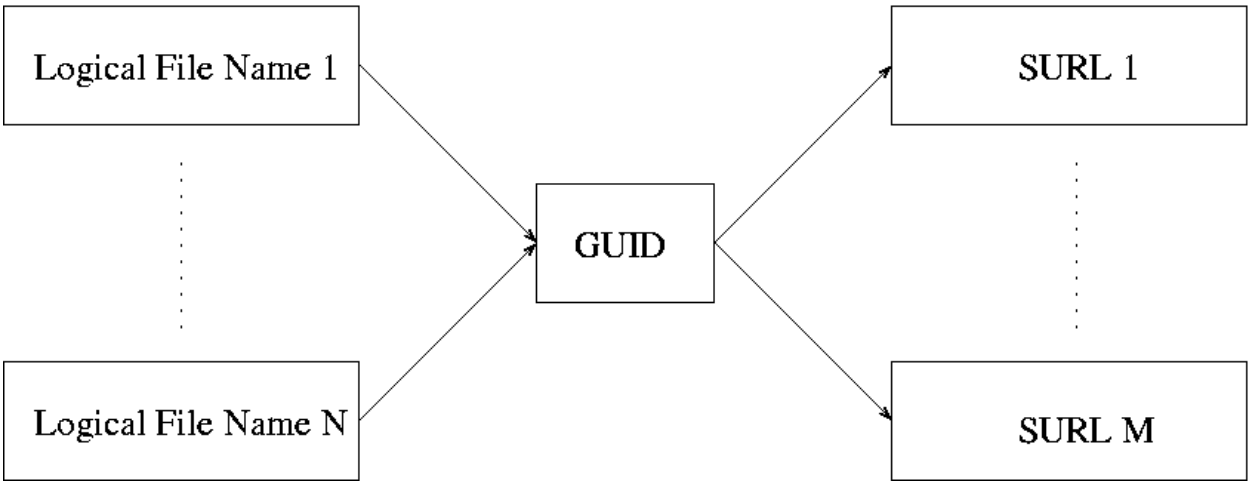
Rank = other.FreeCPUs;

იძლეა საშუალებას ავირჩიოთ კვანძი რომელსაც გააჩნია თავისუფალი პროცესორების უდიდესი რაოდენობა. რასაკვირველია რაც მეტია რანგი, მით უფრო სასურველია რესურსის გამოყენება.

მონაცემების მართვა

GRID-ის მონაცემთა მართვის მთავარი ერთეული ჩვეულებრივ არის ფაილი. GRID გარემოში ფაილებს შეიძლება ჰქონდეთ ასლები რამდენიმე GRID საიტზე. ამ ფაილებს შორის შესაბამისობის უზრუნველსაყოფად არ შეიძლება GRID-ის ფაილების განახლება შექმნის შემდეგ და მხოლოდ წაკითხვისა და წაშლის ოპერაციებია რეკომენდირებული. იდეალურ შემთხვევაში მომხმარებელს არ ჭირდება ფაილის ადგილსამყოფელის ცოდნა, რადგან ის იყენებს ფაილის ლოგიკურ სახელს.

ფაილებს GRID-ში შეიძლება ჰქონდეთ რამდენიმე სახელი. Grid Unique IDentifier (GUID), Logical File Name (LFN), Storage URL (SURL) და Transport URL (TURL). ფაილის GUID და LFN სახელი არ ასახავს ფაილის და მისი ასლის ფიზიკურ მდებარეობას განსხვავებით SURL და TURL-ისგან (ნახ.6).



ნახაზი 6

ფაილს შეიძლება ჰქონდეს GUID უნიკალური სახელი. ეს სახელი მას ეძლევა GRID-ში მისი თავდაპირველი რეგისტრაციისას UUID სტანდარტის საფუძველზე (UUID იყენებს MAC მისამართისა და დროის კომბინაციას). GUID-ის ფორმაა: guid:<unique string>(მაგ: guid:93bd772a-b282-4332-a0c5-c79e99fc2e9c).

მომხმარებელი ფაილზე ოპერაციებისთვის ძირითადად იყენებს LFN-ს. მისი ფორმაა: lfn:<any string>. GRID-ის ფაილს შეიძლება ჰქონდეს რამდენიმე lfn ლინკი ისევე როგორც UNIX ფაილურ სისტემაში.

SURL-ი იძლევა ინფორმაციას ფაილის ასლის ადგილსამყოფელის შესახებ. მას აქვს ასეთი ფორმატი:
srm://<SE hostname>:<port>/srm/managerv2?SFN=<path>

TURL იძლევა აუცილებელ ინფორმაციას ფაილის ასლის ჩასაწერად ან მისაღებად, ჰოსტის სახელის, ფაზის, პროტოკოლის და პორტის გამოყენებით. მისი ფორმატია <protocol>://<SE hostname>:<port>/<path>. არ არის აუცილებელი ერთი ფაილის ფაზა ან ჰოტის სახელი SURL-სთვის და TURL-სთვის ერთი და იგივე იყოს. მოცემული ფაილისთვის შეიძლება იყოს იმდენი TURL რამდენი მონაცემთა წვდომის პროტოკოლიც გააჩნია მოცემულ SE-ს. ასევე SE შეიძლება შეიცავდეს ფაილის მრავალ ასლს დატვირთვის ბალანსირებისათვის (load balancing).

სანამ ფაილები ინახება SE-ზე, LFN, GUID და SURL შორის შესაბამისობა ინახება File Catalogue სერვისში. EMI-ში ოფიციალური კატალოგი არის LCG File Catalogue (LFC).

File Transfer Service (FTS) არის სერვისი რომელიც უზრუნველყოფს SE-ს შორის ფაილების მიღებას, ტრანსფერის დაგეგმვას და განხორციელებას. ის ასევე იძლევა საშუალებას განსაზღვროს ფაილის ნაწილის ტრანსფერი სხვადასხვა ვირტუალურ ორგანიზაციებს შორის და მიანიჭოს სხვადასხვა პრიორიტეტები ინდივიდუალურ ფაილებს. FTS არის ძალიან მოსახერხებელი დიდი მოცულობის მონაცემების ტრანსფერისთვის.

არსებობს მონაცემთა მართვის სხვადასხვა კლიენტური პროგრამები ფაილების ატვირთვა ან ჩამოტვირთვისთვის GRID-ში ან პირიქით GRID-დან, მათი ასლის შექმნისთვის და ფაილების კატალოგთან ინტერაქციისთვის. შეგვიძლია გამოვიყენოთ lcg-util პროგრამა ანუ მისი lcg-* ბრძანებები. ეს არის მაღალი დონის ინტერფეისი (command line and API), რომელიც მალავს ფაილების კატალოგისა და SE-ების ურთიერთობის კომპლექსურობას, რაც ამცირებს გრიდის ფაილის დაზიანების რისკებს.

ასევე შეგვიძლია გამოვიყენოთ glite-gridftp-*, uberftp, globus-url-copy და srm* ბრძანებები. თუმცა მათი გამოყენება ჩვეულებრივი მომხმარებლისთვის არ არის რეკომენდირებული, რადგან ისინი არ უზრუნველყოფენ SE-ზე ფაილების და ფაილების კატალოგში ჩანაწერებს შორის შესაბამისობის დადგენას რამაც შეიძლება ცუდი შედეგი გამოიწვიოს.

ვირტუალური ორგანიზაციაში ხელმისაწვდომი SE-ს მოსაძებნად გამოიყენება „lcg-infosites --vo MyVO se“ ბრძანება. შედეგი იქნება ამის მსგავსი:

Avail Space(Kb)	Used Space(Kb)	Type	SEs
16070000000	n.a	n.a	se.phy.bg.ac.yu
888143872	84934656	n.a	se001.ipp.acad.bg
1634215924	68738609	n.a	se03.grid.acad.bg
2300000000	20226331	n.a	plethon.grid.ucy.ac.cy
75518264	152634624	n.a	node004.grid.auth.gr
2247308768	194503424	n.a	se.hep.ntua.gr

lcg-infosites ბძანებით ასევე შესაძლებელია სხვა სერვერების პოვნა, მათ შორის მოცემულ SE ელემენტთან ახლო მყოფი CE-ის პოვნა. მაგ: lcg-infosites --vo lhcb ce.

დირექტორიის შექმის ბრძანება:

lfc-mkdir -p /grid/MyVO/MyUserName

ფაილის დირექტორიაში ატვირთვა:

lcg-cr -d MySEName -l lfn:/grid/MyVO/MyUserName/MyFileName --vo MyVO MySrcFilePath

MySEName არის SE სახელი, lfn- ფაილის ლოგიკური სახელი, MySrcFilePath – UI-დან ასატვირთი ფაილის სახელი. მას უნდა ჰქონდეს შემდეგი ფორმატი: file:AbsolutePath.

მაგალითად ფაილის U-დან GRID-ში ატვირთვის დასრულებისას:

lcg-cr -d MySEName -l lfn:/grid/MyVO/MyUserName/MyFileName --vo MyVO file:/home/MyUserName/MyFileName

საბოლოო გამოსავალი გამოიყურება ასე:

guid:399099e7-6333-45e9-8029-67aa6097d16a.

ჩვენ lcg-cr ბრძანებას ვიყენებთ ფაილის გრიდში განსათავსებლად მის გაშვებამდე. ასევე შეიძლება გამოვიყენოთ bash სკრიპტი:

#!/bin/sh

lcg-cr -d MySEName --vo MyVO -l lfn:/grid/MyVO/MyUserName/proteins1.fasta file:`pwd`/proteins1.fasta

lcg-cr -d MySEName --vo MyVO -l lfn:/grid/MyVO/MyUserName/proteins2.fasta file:`pwd`/proteins2.fasta

lcg-cr -d MySEName --vo MyVO -l lfn:/grid/MyVO/MyUserName/blosum62.txt file:`pwd`/blosum62.txt

lcg-cr -d MySEName --vo MyVO -l lfn:/grid/MyVO/MyUserName/alignment.exe file:`pwd`/ariadne

დირექტორიის დათვალიერება:

lfc-ls -l /grid/MyVO/MyUserName

ჩვენ ყოველთვის შეგვიძლია lfn-იდან guid-ის მიღება და პირიქით. მაგალითად lcg-lg (GUID სია) ბრძანება გვიბრუნებს lfn ან surl-თან ასოცირებულ guid-ს. SE-ზე ატვირთული ფაილისთვის, რომელსაც დავარქვით lfn-ი შეგვიძლია ვნახოთ მისი guid:

lcg-lg --vo MyVO lfn:/grid/MyVO/MyUserName/MyFileName

შედეგი იქნება:

guid:399099e7-6333-45e9-8029-67aa6097d16a

თუ ჩვენ არ გვინდა მუდმივად სრული მისამართების აკრეფა, შეგვიძლია ჩვენი ძირითადი დირექტორია გავწეროთ გარემოს ცვლადის სახით:

export LFC_HOME=/grid/MyVO/MyUserName

lfc ბრძანებების სია:

lfc-chmod	წვდომის უფლებების შეცვლა LFC ფაილზე/დირექტორიაზე
lfc-chown	LFC ფაილის ან დირექტორიის მფლობელი მომხმარებლის და ჯგუფის შეცვლა
lfc-chgrp	LFC ფაილის ან დირექტორიის მფლობელი ჯგუფის შეცვლა
lfc-setcomment	კომენტარის დამატება/შეცვლა
lfc-delcomment	ფაილთან ან დირექტორიასთან ასოცირებული კომენტარის წაშლა
lfc-setacl	ფაილის/დირექტორიის წვდომის სიის შექმნა
lfc-getacl	ფაილის/დირექტორიის წვდომის სიის ნახვა
lfc-ln	ფაილის/დირექტორიის სიმბოლური ლინკის შექმნა
lfc-ls	დირექტორიაში ფაილების/დირექტორიების სია
lfc-rm	ფაილის/დირექტორიის წაშლა
lfc-mkdir	დირექტორიის შექმნა
lfc-rename	ფაილის/დირექტორიის სახელის შეცვლა
lfc-ping	ამოწმებს დომენური სახელების სერვერის მდგომარეობას.

lcg ბრძანებების სია:

ასლების მართვა

lcg-cp	ფაილების SE -დან/-ზე კოპირება
lcg-cr	ფაილს კოპირება SE-ზე და კატალოგში დარეგისტრირება
lcg-del	ფაილების წაშლა (ერთი ან ყველა ასლის)
lcg-rep	ფაილის კოპირება ერთი SE-დან სხვა SE-ზე და ასლის კატალოგში დარეგისტრირება (replicate)
lcg-gt	მოცემული ფაილის SURL-ის და ტრანსფერ პროტოკოლისთვის TURL-ის მიღება

მოქმედებები ფაილების კატალოგზე

lcg-aa	კატალოგში მოცემული GUID-სთვის ფსევდონიმის მიცემა
lcg-ra	კატალოგში მოცემული GUID-თვის ფსევდონიმის წაშლა
lcg-rf	SE-ზე განთავსებული ფაილის კატალოგში დარეგისტრირება
lcg-uf	SE-ზე განთავსებული ფაილის კატალოგიდან რეგისტრირებული ფაილის ამოშლა
lcg-la	მოცემული LFN, GUID ან SURL-სთვის ფსევდონიმების სია
lcg-lg	მოცემული LFN ან SURL-სთვის GUID-ის ნახვა
lcg-lr	მოცემული LFN, GUID ან SURL-სთვის ასლების სია
lcg-ls	მოცემული SURL-ის ან LFN-სთვის ფაილების სია

ხშირად საჭიროა რაღაც ფაილების გადაგზავნა ერთი კვანძიდან მეორეზე. ამის გაკეთება შესაძლებელია შემდეგი ბრძანებით:

globus-url-copy [options] sourceURL destURL

შესაძლო ოპციების სიის მიღება შესაძლებელია ამ ბრძანების გამოყენებით ოპცია--- help-თან ერთად. შესაძლოა შემდეგი პროტოკოლების გამოყენება: file, gsiftp и http. მაგალითად:

file:///home/loomis/stuff.txt
gsiftp://testbed011.cern.ch/~/stuff.txt
gsiftp://testbed011.cern.ch//tmp/stuff.txt
http://marianne.in2p3.fr/datagrid/documentation/daemon-guidelines.html

პროტოკოლ file-სათვის დასაშვებია მხოლოდ აბსოლუტური სახელი. პროტოკოლ gsiftp-შემთხვევაში home დირექტორიის აღნიშვნისათვის გამოიყენება ნიშანი ტილდა.

ერთ-ერთი მნიშვნელოვანი უპირატესობა globus-url-copy ბრძანების გამოყენების არის ის, რომ მისი საშუალებით შესაძლებელია მონაცემების პირდაპირი გადაცემა ორ დაშორებულ კომპიუტერს შორის. ეს შესაძლებლობას იძლევა თავიდან ავიცილოთ ფაილების ორმაგი კოპირება (განსაკუთრებით იმ შემთხვევაში თუ ბრძანება გაიცემა კომპიუტერიდან რომელიც ქსელს უერთდება დაბალსიჩქარიანი ხაზით).