Національний університет «Полтавська політехніка імені Юрія Кондратюка»

(повне найменування вищого навчального закладу)

Навчально науковий інститут інформаційних технологій та робототехніки

(повна назва факультету)

Кафедра комп’ютерних та інформаційних технологій і систем

(повна назва кафедри)

**Пояснювальна записка**

**до дипломного проекту (роботи)**

\_\_\_\_\_\_\_\_\_\_\_\_\_бакалавра\_\_\_\_\_\_\_\_\_\_\_\_\_

(освітньо-кваліфікаційний рівень)

на тему

Конструювання обчислювальної GRID-системи на базі компонентів UNICORE.

Виконав: студент 4 курсу, групи 401-ТК

спеціальності

\_\_123 Комп’ютерна інженерія \_

(шифр і назва напряму)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_Давиденко А. Ю.\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Керівник \_\_\_\_\_Скакаліна О. В. \_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Рецензент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Полтава – 2021 року

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ « ПОЛТАВСЬКА ПОЛІТЕХНІКА ІМЕНІ ЮРІЯ КОНДРАТЮКА»**

**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА РОБОТОТЕХНІКИ**

**КАФЕДРА КОМП’ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ**

**КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА спеціальність 123 «Комп’ютерна інженерія»**

**на тему**

**«Конструювання обчислювальної GRID-системи на базі компонентів UNICORE»**

### Студента групи 401-ТК Давиденка Антона Юрійовича

Керівник роботи кандидат технічних наук, доцент Скакаліна О. В.

Завідувач кафедри кандидат технiчнix наук, доцент Головко Г.В.

Полтава – 2021

### РЕФЕРАТ

Кваліфікаційна робота бакалавра: 70 с., 17 рисунків, 1 додаток, 25 джерел.

**Об’єкт дослідження**: модель обчислювальної GRID-системи на базі компонентів UNICORE.

**Мета роботи**: конструювання обчислювальної GRID-системи для об’єднання децентралізованих і/або віддалених обчислювальних ресурсів в одну мережу для подальшого централізованого доступу до їх обчислювальних потужностей, забезпечення адміністрування і подальшого масштабування цієї системи.

**Методи**: встановлення компонентів UNICORE на цільові робочі станції, редагуання конфігураційних файлів, тестування готової мережі.

Ключові слова: GRID-система, грід, мережа, UNICORE, розподілені обчислення, конструювання.

**ABSTRACT**

Bachelor's qualification work: 70 p., 17 drawings, 1 appendix, 25 sources.

**Object of research:** model of computing GRID-system based on UNICORE components.

**Purpose:** designing a computing GRID system to combine decentralized and/or remote computing resources into a single network for further centralized access to their computing power, providing administration and further scaling of this system.

**Methods:** installation of UNICORE components on target workstations, configuration files editing, testing of created network..

Keywords: GRID-system, grid, network, UNICORE, distributed computing, designing.

**ЗМІСТ**

[ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 5](#_Toc74683756)

[ВСТУП 7](#_Toc74683757)

[РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД ОБЧИСЛЮВАЛЬИХ ГРІД-СИСТЕМ 10](#_Toc74683758)

[1.1. Опис предметної області 10](#_Toc74683759)

[1.2. Властивості і основні поняття грід-систем 12](#_Toc74683760)

[1.3. Сервіси грід-систем 14](#_Toc74683761)

[1.4. Архітектура грід-систем 17](#_Toc74683762)

[1.5. Вирішення питань безпеки в грід 22](#_Toc74683763)

[1.6. Огляд існуючих рішень 25](#_Toc74683764)

[РОЗДІЛ 2. ФУНКЦІОНАЛЬНІ МОЖЛИВОСТІ СИСТЕМИ UNICORE І ЇЇ КОМПОНЕНТІВ 31](#_Toc74683765)

[2.1. Загальна інформація про UNICORE 31](#_Toc74683766)

[2.2. Архітектура і основні компоненти UNICORE 33](#_Toc74683767)

[РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ОБЧИСЛЮВАЛЬНОЇ ГРІД-СИСТЕМИ 41](#_Toc74683768)

[3.1. Встановлення операційних систем для сервера UNICORE 41](#_Toc74683769)

[3.2. Встановлення додаткового ПЗ, конфігурація ОС 41](#_Toc74683770)

[3.3. Встановлення і налаштування сервера UNICORE 42](#_Toc74683771)

[3.4. Встановлення компонентів системи UNICORE 43](#_Toc74683772)

[РОЗДІЛ 4. ТЕСТУВАННЯ СТВОРЕНОЇ ГРІД-СИСТЕМИ 46](#_Toc74683773)

[4.1. Запуск компонентів системи UNICORE 46](#_Toc74683774)

[4.2. Тестування роботи грід-системи 52](#_Toc74683775)

[4.3. Введення в експлуатацію 54](#_Toc74683776)

[ВИСНОВКИ 55](#_Toc74683777)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 56](#_Toc74683778)

[ДОДАТОК А 59](#_Toc74683779)

[КОНФІГУРАЦІЙНИЙ ФАЙЛ CONFIGURE.PROPERTIES 59](#_Toc74683780)

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

**ЕОМ –** електронно-обчислювальна машина

**ППЗ –** проміжне програмне забезпечення

**ПЗ –** програмне забезпечення

**API –** Application Programming Interface

**FTP –** File Transfer Protocol

**HTTP –** HyperText Transfer Protocol

**IP –** Internet Protocol

**TCP –** Transmission Control Protocol

**UDP –** User Datagram Protocol

**CVS –** Concurrent Versions System

**PSI –** Problem Solving Environment

**MWF –** Meta-workflow

**MPI –** Message Passing Interface

**IETF –** Internet Engineering Task Force

**TLS –** Transport Layer Security

**CMS –** Content Management System

**GSS –** Generic Security Service

**PKIX –** Public-Key Infrastructure X.509

**JDL –** Java Declarative Language

**WLCG –** Worldwide LHC Computing Grid

**ВАК –** Великий адронний колайдер

**OGSA –** Open Grid Services Architecture

**WS-RF –** Web Services Resource Framework

**SOA –** Service-Oriented Architecture

**URC –** Unicore Rich Client

**UCC –** Unicore Comandline Client

**HiLA –** High Level API

**XUUDB –** X.509 Unicore User Database

**BES –** Basic Execution Services

**ACL –** Access Control List

**SAML –** Security Assertion Markup Language

**CA –** Certification Authorities

**SOAP –** Simple Object Access Protocol

**XNJS –** eXtended Network Job Supervisor

**SSL –** Secure Sockets Layer

**TSI –** Target System Interface

**XACML –** eXtensible Access Control Markup Language

**VOMS –** Virtual Organization Membership Service

**CIS –** Common Information Service

**XML –** eXtensible Markup Language

**HPC-BP –** High Performance Computing Basic Profile

**HPC-FSP –** High Performance Computing File Staging Profile

**JSDL –** Job Submission Description Language

**DRMAA –** Distributed Resource Management Application API

**ОС –** операційна система

# ВСТУП

Нині інформаційні технології розвиваються надзвичайно швидко. Компютерні технології, штучний інтелект досить швидко впроваджуються в усі сфери народного господарства і людської діяльності. Також вони з великою активністю використовуються в научних дослідженнях, в тому числі і фундаментальних. Кожне нове дослідження потребує обробки і обчислень великого, іноді гігантського обсягу даних. Одним із найпростіших і очевидних рішень даної проблеми останні десятки років є використання суперкомп'ютерів. Але зараз вони не завжди забезпечують потреби інженерів і вчених.

Суперкомп'ютери завжди були достатньо дорогими. Вони потребують великих витрат при проектуванні, виробництві, і немалих коштів на обслуговування. Окрім того, комп'ютери вдосконалюються і еволюціонують дуже швидко: процесорна потужність подвоюється приблизно кожні 1,5 роки. Тобто, суперкомп'ютери, які були еталоном потужності і швидкодії на момент їх введення в експлуатацію, можуть через 5-10 років поступатися деякими персональним комп'ютерам, а поставлені перед ними завдання лише ускладнюються. Через це суперкомп'ютери з кожним роком все гірше справляються з поставленими їм задачами: потужність машин залишається сталою, а час, потрібний для отримання результату, збільшується через постійно зростаюче ускладнення поставлених завдань. У спробі вирішити дані проблеми, технічний прогрес дійшов до застосування розподілених обчислень.

Розподілені обчислення - це метод вирішення важких, ресурсомістких обчислювальних задач і операцій з використанням декількох електронно-обчислювальних машин (ЕОМ), об'єднаних в паралельну обчислювальну систему. Вони дозволяли використовувати кілька комп'ютерів одночасно, розпаралелюючи частини задач, де це можливо, для зменшення часу обчислень і збільшення ефективності роботи. Робочі станції, об'єднані в подібну мережу частіше всього називають обчислювальними кластерами. Такий підхід дозволяє істотно скоротити витрати на підтримання роботи і обслуговування такої мережі через використання вже існуючих комп'ютерів. Однією із особливостей розподілених багатопроцесорних обчислювальних систем, якої немає в суперкомп'ютерів, є можливість нарощування продуктивності масштабуванням. Можна досить просто збільшити обчислювальну потужність системи за рахунок підключення більшої кількості робочих станцій. Вважається, що збільшення продуктивності розподіленої системи може відбуватися необмежено, проте на практичному рівні нарощування корисної потужності є обмеженим, як правило, законом Амдала і законом Густавсона-Барсіса.

Однак даний підхід має певні мінуси, один з яких - географічне розташування. Далеко не кожна організація або користувач має можливість тримати необхідну кількість обчислювальних машин в одному місці. Завдяки науково-технічному прогресу, швидкому інтернету і захищеними каналами зв'язку нині не потрібно тримати всі частини обчислювальної мережі в одному місці. Вони можуть бути розташовані по всій земній кулі і діяти з ефективністю, яка не уступає системам, розташованим в одній споруді. Для таких обчислювальних систем з високим ступенем розподілу було виділено окремий клас розподілених систем – грід.

Грід-обчислення є однією з форм розподілених обчислень, в якій суперкомп'ютер складається з одинарних ЕОМ і кластерів, географічно розподілених в різних місцях. Вони працюють разом для виконання необхідних операцій. Грід – це технологія, яка використовується для вирішення наукових завдань, що потребують немалих обчислювальних ресурсів. Грід-обчислення використовуються і в комерційній інфраструктурі для вирішення таких завдань, як сейсмоаналіз, економічне прогнозування, розробка та вивчення властивостей нових ліків, перевірка фундаментальних теорій про походження і побудову Всесвіту тощо. Всі обчислювальні ресурси робочих станцій, що беруть участь в обчисленні, об'єднуються в одну єдину грід-мережу.

Метою даної дипломної роботи є вивчення грід-обчислень як концепції, аналіз існуючих рішень для організації гріду, побудова власної грід-мережі на базі компонентів системи UNICORE.

# РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД ОБЧИСЛЮВАЛЬИХ ГРІД-СИСТЕМ

## **Опис предметної області**

Останні кілька десятків років є часом зародження та розвитку іноваційного напрямку в IT, назву якому дали у 1998 році Ян Фостер та Карл Кессельман – «Грід» (англ. grid). Свою ідею вони пояснювали через аналогію з електричними мережами, які називаються Power GRID. Power GRID – це мережа електроживлення, побудована таким чином, що кожен може підключиться до розетки і отримати електричну енергію, не роздумуючи про те, звідки ця енергія з’явилася. Точно такий же принцип закладений і в грід-мережах – користувач підключається до такої мережі і отримує необхні обчислювальні ресурси, не роздумуючи, звідки вони з'являються і де знаходяться [1].

Ідея використання мережі комп‘ютерів для вирішення різноманітних обчислювальних задач з’явилася набагато раніше, спроби створити подібні системи робилися з 1960-х років, однак нині вона набула завершеної форми «концепції грід» [2].

Концепція грід-систем виникла на базі успіхів, досягнутих в наступних напрямках:

* різке підвищення продуктивності мікропроцесорів масового виробництва;
* появи високошвидкісних оптоволоконних ліній зв'язку;
* феномена WWW/internet-глобалізації процесу обміну інформацією і інтеграції світової економіки;
* розвитку методів організації вирішення складних обчислювальних завдань;
* вдосконалення технології і засобів інформаційної безпеки [3].

Вважається, що фізики-ядерники спричнили розвиток грід-обчислень. Їх потреба в обробці колосальних об‘ємів даних, отриманих в результаті різних досліджень, є рушійною силою для втілення різних програм з впровадження грід і сьогодні. Однак грід має достатню кількість й інших сфер для застосування, оскільки пропонує універсальний підхід до вирішення проблеми недостатньої кількості обчислювальних ресурсів. Очевидно, що мережа комп‘ютерів має спроможність вирішити набагато складніші і ресурсомісткіші задачі, ніж кожен з її складових вузлів окремо [2].

Грід як засіб для сумісного використання обчислювальних ресурісв та сховищ даних дозволяє не лише обмінюватися даними між комп’ютерами, а й, зрештою, перетворити комп’ютерну мережу в глобальну, в певний величезний віртуальний комп’ютер, доступний віддалено з будь-якої точки на земній кулі [2].

Нині, до сфери застосування грід відносяться:

* ядерна фізика;
* захист навколишнього середовища;
* передбачення погоди і моделювання кліматичних змін;
* чисельне моделювання в машинобудуванні і авіабудуванні;
* біологічне моделювання.
* фармацевтика [4].

Крім того, застосування грід може дати якісно нову вирішення наступних класів задач:

* масова обробка потоків даних великого обсягу;
* багатопараметричний аналіз даних;
* моделювання на віддалених суперкомп'ютерах;
* математичне моделювання з високоточної візуалізацією обчислювальних експериментів;
* складні бізнес-додатки з великими об'ємами обчислень [4].

Аналогічно до інфраструктури електромереж (підстанцій, трансформаторів, ліній електропередач та іншої інфраструктури) в грід існує своя інфраструктура – з проміжним програмним забезпеченням (ППЗ, з англ. middleware), з допомогою якого виконується «віртуалізація» ресурсів. Розвиток ППЗ пройшов шлях розвитку від базових засобів, що підтримують віддалений доступ до ресурсів системи, до створення платформ, тобто узгоджених наборів інструментів і засобів, здатних дати комплексне рішення завдання обслуговування грід-інфраструктури виробничого призначення [2].

Нині, грід претендує на роль універсальної інфраструктури для обробки даних, в якій функціонує безліч служб (Grid Services), які дозволяють вирішувати не тільки конкретні прикладні завдання, а й пропонують такі послуги як: пошук необхідних ресурсів, збір інформації про стан ресурсів, зберігання і доставку даних [5].

* 1. **Властивості і основні поняття грід-систем**

Ничже перераховані та розшифровані основні властивості грід, до яких відносять:

* *cпільне використання розподілених ресурсів* – дана властивість відкриває можливості для співробітництва, яких було б досить важко досягти іншими засобами. Попри це виникають питання «справедливості» розподілу ресурсів, створення та керування віртуальними організаціями (ВО);
* *об‘єднання потужностей* – проводиться побудова системи з більшою сумарною потужністю обчислень, ніж у її окремих вузлів, також відбувається більш ефективне використання апаратних засобів (зменшується час бездіяльності системи). Постають жорсткі вимоги до каналів зв‘язку;
* *віртуалізація* – включає в себе наступні поняття: приховування або маскування від користувача апаратного та програмного влаштування системи та її складових, реальних відстаней між вузлами, приналежності вузлів до різних організацій, компаній і країн, створюється певна ілюзія роботи з «віртуальним суперкомп‘ютером»;
* *неоднорідність (гетерогенність)* – зазвичай грід складається з множини різноманітних апаратних засобів та різноманітного програмного забезпечення (ПЗ) і він має успішно виконувати поставлені завдання в таких умовах;
* *децентралізоване управління* – не існує єдиного власника грід-системи, що вимагає використання механізмів розподіленого управління;
* *інтероперабельність* – функціональна сумісність роботи різних частин грід та навіть різних грід-інфраструктур основується на стандартизації інтерфейсів між цими частинами. Підходи побудови розподілених мереж, без врахування існуючих стандартів, є досить складними і майже не перспективними;
* *прозорість доступу* – грід має давати доступ до ресурсів мережі користувачам, незалежно від конкретної топології мережі чи локальної реалізації механізмів доступу до тих чи інших вузлів та їх ресурсів;
* *масштабованість* – грід має надавати механізми під’єднання нових джерел обчислювальних ресурсів, сховищ даних, користувачів не впливаючи на вже наявних учасників мережі: грід повинен мати можливість динамічної зміни своєї конфігурації;
* *безпека* – одна з головних вимог до грід-системи – безпека доступу до обчислювальних ресурсів, що обмежує набір дозволених операцій у авторизованих користувачів та програм [6].

Проте слід зазначити, що попри високу динаміку розвитку цієї галузі, що дозволяє сподіватися на успішне широке впровадження грід-технологій у найближчому майбутньому, є достатня кількість питань, які все ще є предметом дискусій і досліджень.

Нижче приведені основні поняття грід:

* *ресурс* – ресурсами являються обчислювальні ресурси, мережеві ресурси, системи зберігання даних, каталоги тощо. Ресурси зазвичай поділяють на фізичні і логічні. До фізичних відносяться: оперативна пам'ять, постійна пам'ять (жорсткі, твердотільні накопичувачи, флеш-карти тощо), продуктивність і кількість процесорів і т.д. Логічними ресурсами є комп'ютерний кластер, розподілена файлова система, розподілений набір комп'ютерів;
* *мережевий протокол* – це формальний опис форматів повідомлень і набір правил, що визначає процедури обміну повідомленнями між частинами грід-мережі. Розрізняють мережеві протоколи нижнього (наприклад Ethernet), середнього (IP, TCP, UDP тощо) і високого рівня (FTP, HTTP та ін.);
* *синтаксис* – це правила, що визначають порядок і форму запису інформації в повідомленнях;
* *сервіс* – це сутність, яка надає специфічну функціональність;
* *інтерфейс прикладного програмування* (англ. application programming interface, API) – це набір функцій (сервісів), що дозволяють застосунку здійснювати доступ до ресурсів через обслуговуючу операційну систему (ОС) [6];
  1. **Сервіси грід-систем**

В розподіленому обчислювальному середовищі має бути набір сервісів, для контрольованого виконання прикладних програм авторизованих користувачів. Специфіка сервісів грід визначається, перш за все, характером самого обчислювального середовища: грід є динамічною структурою. З часом можуть змінюватись як кількість об'єднаних ЕОМ (наприклад, через непередбачуваний збій, планове вимкнення або розміщення в мережі нових робочих станцій), так і їх якісні характеристики (зміна навантаження на окремий вузол системи).

Для нормальної роботи грід-системи необхідно забезпечити:

* *ідентифікацію виконуваної програми* – для забезпечення контролю виконання прикладної програми вона повинна забезпечуватися унікальним номером (ідентифікатором). Присвоєння такого ідентифікатора і контроль його унікальності мають виконуватися спеціальним сервісом грід;
* *авторизацію користувача* – в грід завдання авторизації користувача є досить складною проблемою. Необхідне забезпечення унікальності ідентифікатора користувача сервісом авторизаціх користувачів на всіх обчислювальних вузлах, що утворюють розподілене середовище, а також необхідно визначати відносний пріоритет користувача, необхідний для нормальної роботи служби розподілення обчислювальних ресурсів між прикладними програмами;
* *пошук ресурсів* – даний сервіс визначає розміри доступного в даний момент часу обчислювального ресурсу, результати роботи цього сервісу можуть бути використані для моніторингу поточного стану системи;
* *опис ресурсів* – даний сервіс здійснює єдинй опис різнорідних ресурсів, що знаходяться в поточний момент часу в складі грід;
* *резервування ресурсів* – даний сервіс повинен забезпечити зайняття вільних ресурсів для розміщення на ньому прикладної програми. Його робота сильно пов'язана з роботою сервісу пошуку і сервісу опису ресурсів;
* *розподіл ресурсів* – менеджер ресурсів здійснює поділ існуючих ресурсів між прикладними програмами;
* *виконання розподілених алгоритмів* – цей сервіс забезпечує виконання паралельних програм;
* *доступ до віддалених даних* – сервіс забезпечує роботу розподілених баз даних;
* *виявлення несправностей* – сервіс визначає працездатність включених в обчислювальний процес вузлів системи. Як тільки обчислювальне середовище однієї з прикладних програм виявилося пошкодженим, даний сервіс сигналізує про це менеджеру ресурсів і менеджеру управління завданнями.

Кількість і характер сервісів, що забезпечують роботу грід мережі, можна змінювати в залежності від призначення конкретного обчислювального середовища [6].

* 1. **Архітектура грід-систем**

Важливою частиною грід-системи є її архітектура. Для опису рівнів архітектури грід, буде використана модель «пісочного годинника». Як показано на рисунку 1.1, горловина пісочного годинника відповідає невеликій кількості основних протоколів (TCP і HTTP), необхідних для зв'язку служб високого рівня зі службами низького рівня [7].

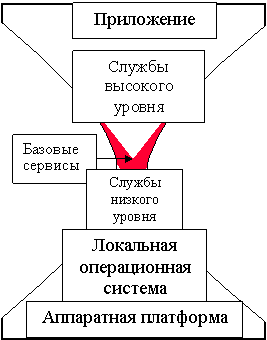


Рисунок 1.1 – Архітектура GRID-системи за моделлю пісочного годинника

Якщо деталізувати рисункок 1.1, архітектуру грід можна відобразити як ієрархічну структуру (рисунок 1.2), що включає в себе декілька рівнів. На кожному рівнів наявні свої сервіси, які взаємодіють за допомогою певних визначених протоколів [8].

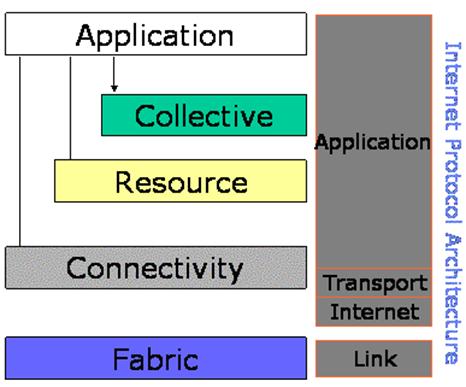


Рисунок 1.2 – Багаторівнена архітектура грід

**Рівень Fabric** пропонує ресурси, спільний використання яких забезпечується через грід протоколи.

Доступ до обчислювальних ресурсів потребує:

* механізмів для запуску програми і відстежування стану її виконання;
* механізмів для визначення апаратно-програмних характеристик і також визначення поточного стану (наприклад, робочого завантаження ресурсів).

Для доступу до ресурсів, що забезпечують зберігання даних потрібні:

* механізми для запуску програм і відстежування їх виконання;
* механізми для відправки/отримання файлів, в тому числі високошвидкісні механізми з підтримкою багатопотоковості, механізми для читання/запису підмножин файлів;
* механізми менеджменту, які дозволяють керувати ресурсами, які займаються пересиланням даних;
* механізми, що дозволяють дізнатися апаратні і програмні характеристики та поточне завантаження;
* механізми для проведення віддалених вибірок даних;
* механізми резервування.

Для отримання доступу до мережевих ресурсів необхідні:

* механізм менеджменту і керування ресурсами, призначені для мережевого трафіку (встановленн пріоритетів, проведення резервування);
* механізми визначення характеристик і завантаженості мережі;

Для отримання доступу до ресурсів сховища кодів програм потрібен механізм керування версіями вихідного та об'єктного коду, наприклад, Concurrent Versions System (CVS) або GIT.

Для доступу до ресурсів каталогів потрібен механізм зберігання даних, обробки запитів і проведення оновлення даних [6].

**Рівень Connectivity** включає основні комунікаційні та ідентифікаційні протоколи, що потрібні для проведення специфічних для грід операцій. Комунікаційні протоколи дозволяють ресурсам рівня Fabric обмінюватися даними. Ідентифікаційні протоколи, побудовані на комунікаційних сервісах, дають криптографічно захищений механізм для верифікації і ідентифікації користувачів і ресурсів. Проблеми безпеки на рівні Connectivity має вирішуватися за рахунок вже існуючих стандартів безпеки [6].

Ідентифікація користувача в системі може бути проведена різними способами. Зокрема, може бути використано, так зване, одноразове реєстрування при використанні якого користувачам необхідно реєструватися в обчислювальному середовищі всього один раз і, після успішного закінчення реєстрації, отримати доступ до обчислювальних ресурсів рівня Fabric [6].

Також існує такий спосіб ідентифікації як делегування. Програма користувача повинна мати доступ до ресурсів, на яких він авторизований. За певних умов програма може передати підмножину своїх прав іншій дочірній програмі [6].

Нормально, що кожен провайдер обчислювальних ресурсів може мати певну кількість локальних рішень в сфері безпеки ресурсів. Система безпеки обчислювального середовища повинна коректно взаємодіяти з цими рішеннями [6].

**Рівень Resource** базується на авторизаційному і комунікаційному протоколі рівня Connectivity. Він визначає протоколи для таких операцій:

* створення облікових записів користувачів;
* здійснення безпечного обміну інформацією;
* ініціалізації, моніторингу та проведення спільних операцій на індивідуальних обчислювальних ресурсах;
* проведення обліку використаного часу для кожного користувача окремо.

Доступу та контроль над локальними ресурсами сервіси рівня Resource проводиться за допомогою функцій рівня Fabric. Необхідно відмітити, що протоколи рівня Resource повністю зв’язані з індивідуальними ресурсами.

Протоколи рівня Resource можуть бути розбиті за призначенням в два класи:

* протоколи менеджменту, які використовуються для надання доступу до спільних ресурсів. Вони забезпечують виконання резервування і контролю відповідності ресурсів певним вимогам;
* інформаційні протоколи, необхідні для моніторингу інформації про структуру і стан ресурсу (наприклад, конфігурацію, поточне завантаження тощо).

Протоколи рівнів Resource і Connectivity є базовими і повинні бути обмежені невеликю і добре продуманою множиною. Ці протоколи повинні бути сконструйовані для спільного використання різнорідних ресурсів, але й не повинні занадто сильно обмежувати допустимі протоколи вищого рівня і їх продуктивність [6].

**Рівень Collective** згруповує протоколи і сервіси, які не зв’язані з конкретними обчислювальними ресурсами, він більш глобальний за своєю суттю і забезпечує колективну взаємодію між ресурсами [6].

Прикладами сервісів рівня Collective можуть бути:

* сервіс директорії, який дозволяє учасникам грід досліджувати представлені ресурси і їх властивості. В цьому сервісі використовуються протоколи грід рівня Resource;
* сервіси розподілу і планування ресурсів, який оброблює заявоки користувачів, що дозволяє учасникам грід отримувати доступ до необхідних ресурсів, а також розпланувувати виконання завдань на них. Прикладами реалізації таких сервісів можуть бути AppLeS, Condor-G, Nimrod-G і DRM брокер;
* сервіси моніторингу та діагностики, які надають інформацію про збої, спроби обійти систему безпеки або зламати її, перенавантаження ресурсів тощо;
* сервіси реплікації даних, які дозволяють контролювати зберігання інформації, оптимізувати розташування і кількість копій даних в залежності від надійності, часу доступу, вартості;
* сервіс завантаження і спільних робіт, відомий як Problem Solving Environment (PSI), який надає засоби для управління потоком завдань;
* сервіси пошуку програмного забезпечення, дають можливість дослідити і вибрати необхідне для вирішення проблеми ПЗ. Прикладами можуть бути Nimf і NetSovle;
* сервіси авторизації, які визначають правила використання ресурсів для користувачів грід. Ці сервіси використовують протоколи рівня Resource і локальні протоколи безпеки;
* сервіси реєстрації, які збирають інформацію про використання ресурсів користувачами;
* сервіси співпраці, які забезпечують синхронний і асинхронний скоординований обмін інформацією між великими спільнотами користувачів. Прикладами можуть служити Access GRID і CAVERNsoft [6].

Системи грід-програмування дозволяють використовувати вже відомі програмні моделі в грід-середовищі. Вони використовують різноманітні грід-сервіси, такі як використання ресурсів, призначення ресурсів тощо. Прикладами можуть бути грід-реалізації інтерфейсу передачі повідомлень MWF і MPI [6].

**Рівень Application** включає в себе застосунок користувача, який функціонує в грід-середовищі (рисунок 1.3).

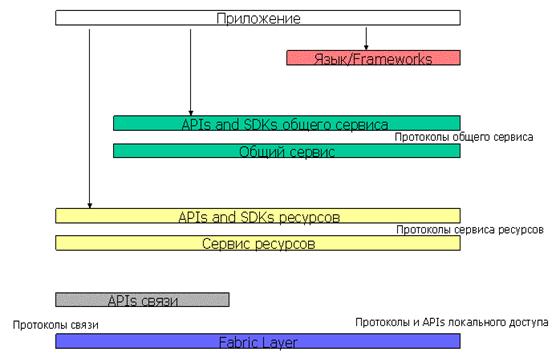


Рисунок 1.3 – Взаємодія прикладної програми з різними рівнями грід-системи

Взаємодія прикладної програми з сервісами різних рівнів здійснюється через інтерфейси прикладних програм цих сервісів [6].

* 1. **Вирішення питань безпеки в грід**

Вкрай важливою частиною будь-якої грід-системи є забепечення безпеки. Очевидно, що питання безпеки в інформаційних системах мають велику роль. Захист інформації в грід має свої особливості. Зокрема, необхідно забезпечити безпеку спільного використання ресурсів. Для забезпечення безпеки, необхідно передбачити встановлення механізму відключення анонімного несанкціонованого доступу в обчислювальне середовище [6].

На кожному рівні архітектури грід рішення щодо забезпечення безпеки мають свою особливості. При не можна упускати той факт, що повна система безпеки грід повинна успішно взаємодіяти з уже існуючими локальними рішеннями. Попри це, різнорідність локальних рішень сильно ускладнює створення комплексних систем безпеки. Проте, одним з можливих шляхів усунення подібних труднощів є використання єдиних стандартизованих підходів реалізації локальних безпекових рішень [8].

Існує безліч стандартів для забезпечення безпеки, але жоден з них не задовольняє вимогам систем безпеки грід в повній мірі. Нижче будуть розглянуті основні з існуючих стандартів і ступінь їх відповідності вимогам, що пред'являються до грід-систем.

1. **Kerberos.** Kerberos – IETF стандарт, який підтримує безпеку системи через встановлення аутентичності, цілісності і конфіденційності повідомлення, створеного при використанні методу роздільної секретної криптографії. Цей протокол може забезпечити одноразову авторизацію і гнучкий захист повідомлень. Попри це, виконання вимоги інтеграції з локальними безпековими рішеннями для цього протоколу ускладнено, так як різноманітні реалізації Kerberos мають тенденцію замінювати локальні рішення безпеки.
2. **TLS**. TLS (Transport Layer Security) (спочатку відомий як Secure Sockets Layer, SSL) – IETF стандарт для встановлення аутентичності, цілісності і конфіденційності повідомлення, яке створено з використанням технології криптографії з відкритим ключем. Використання даного протоколу складне при реалізації одноразової авторизації і делегування.
3. **CMS.** CMS (Cryptografic Message syntax) – стандарт IETF визначає синтаксис, який дозволяє в цифровій формі підписати, підтвердити справжність, або зашифрувати будь-які повідомлення.
4. **GSS-API.** GSS-API (Generic Security Service API) – стандарт IETF, який визначає інтерфейс прикладних програм, що забезпечує встановлення автентичності, цілісності і конфіденційності повідомлення. Він передбачає, що дві сторони, що обмінюються повідомленнями, мають зв'язок, засновану на протоколі, що забезпечує надійність доставки інформаційних пакетів (наприклад, TCP / IP). Даний стандарт може бути використаний при реалізації будь-якого з основних механізмів безпеки і може бути рекомендований для застосування в розподіленої обчислювальної середовищі. Розширенням стандарту є стандарт IDUP-GSS-API, який забезпечує підтримку захисту незалежних одиниць даних.
5. **PKIX.** PKIX (Public-Key Infrastructure X.509) – набір IETF стандартів, які описують протоколи і синтаксис управління сертифікатами X.509 в інфраструктурах систем безпеки, що використовують технологію відкритих ключів. Стандарти X.509 зазвичай використовуються в поєднанні з іншими стандартами безпечного зв'язку, наприклад, TLS [3].

Світовий досвіду розробок розподілених обчислювальних систем показує, що система безпеки грід має включати в себе властивості:

1. Можливість здійснення одноразової реєстрації користувачів. Така реєстрація користувачів передбачає, що процедура ідентифікації проводиться тільки в момент входу користувача в систему. Всі переходи даного користувача між ресурсами розподіленої обчислювальної середовища відбуваються без повторної ідентифікації.
2. Можливість делегування прав. Делегування прав полягає в тому, що програма користувача повинна мати доступ до ресурсів, на яких користувач авторизований, і так само повинна мати можливість передати підмножину своїх прав дочірньому процесу.
3. Можливість інтеграції з локальними системами безпеки. Кожен ресурс може використовувати будь-який з існуючих способів вирішення проблеми безпеки системи. Реалізація системи безпеки грід повинна взаємодіяти з цими локальними системами безпеки, не змінюючи їх.
4. Можливість реалізації довірчих відносин. Довірчі відносини припускають, що користувач авторизований на ресурсі X, повинен вважатися авторизованим і на ресурсі Y, якщо ресурс X взаємодіє з ресурсом Y [6].

Окремо необхідно розглядати проблему захисту зв'язку між різними ресурсами обчислювального середовища. На цьому рівні повинні бути забезпечені:

* гнучкий захист повідомлень. Передбачається, що прикладна програма повинна мати можливість динамічно конфігурувати протокол сервісу безпеки, щоб використовувати різні рівні захисту повідомлення: або визначати тільки цілісність повідомлення, або цілісність і конфіденційність. Вибір рівня захисту може обиратися за різними критеріями: типом переданих повідомлень, вимогами сторін сеансу зв'язку, інфраструктурою, в якій здійснюється передача повідомлення;
* підтримка різних протоколів зв'язку, які гарантують доставку інформаційних пакетів. У зв'язку з розвитком Мережі Інтернет, в даний час велике поширення отримав протокол TCP, що забезпечує гарантовану доставку інформаційних пакетів. Система безпеки грід повинна мати можливість використання інших протоколів, що володіють подібними властивостями [6].

## **Огляд існуючих рішень**

В даний час в світі проводяться близько 300 проектів по використанню обчислювальних потужностей розподілених комп’ютерних систем, до числа яких можна віднести: SETI@home, DISTRIBUTED.NET, GIMPS, ANTHRAX, GLOBUS, CONDOR, CERN і ін. Далі будуть розглянуті деякі реальні проекти і існуючі технології грід-систем.

**Проект SETI@home** (SEARCH FOR EXTRATERRESTRIAL INTELLIGENCE) - пошук позаземних цивілізацій за допомогою розподіленої обробки даних, що надходять з радіотелескопа. Приєднатися до проекту міг кожен охочий, завантаживши на свій комп'ютер програму для обробки радіосигналів. Доступні клієнтські програми для WINDOWS, Масintosh, UNIX, OS/2. Добровольці, які брали участь в проекті, отримували порціями по 384 Кб радіоастрономічні дані в файлах. Ці дані оброблялися у фоновому режимі програмою, завантаженої з web-сайту SET@home.com. Вона виконувала досить об'ємні обчислення, пов'язані з швидким перетворенням Фур'є, але які саме - для добровольця залишається невідомим. В процесі роботи комп'ютер користувача має зворотний зв'язок, що свідчить про хід виконання процесу. Після закінчення результати розрахунків поверталися в [SET@home.com](mailto:SET@home.com). Згідно наведеній на сайті статистиці, сумарна продуктивність задіяних в проекті комп'ютерів у багато разів перевершувала продуктивність всіх комп'ютерів зі списку Тор 500 [9]. 31 березня 2020 року проект був заморожений. Всього, за більше ніж 20 роботи проекту в ньому взяло участь декілька мільйонів користувачів [10].

**DISTRIBUTED.NET** також є одним з найбільших об'єднань користувачів Інтернету, що надають свої комп'ютери для вирішення складних завдань в розподіленому режимі. Ці проекти пов'язані з завданнями визначення шифрів (RSA CHALLENGES). З моменту початку проекту в ньому зареєструвалися близько 200 тис. людей [9].

**Проект GIMPS** (GREAT INTERNET MERSENNE PRIME SEARCH) ставить задачу пошуку простих чисел Мерса, тобто числа виду *2Р-1*, де *p* є простим числом. У грудні 2018 року в рамках даного проекту було знайдено максимальне на даний час число Мерса – 282589933-1. Десятки тисяч комп'ютерів по всьому світу, віддаючи частину своїх обчислювальних ресурсів, працювали над цим завданням більше дев’ятнадцяти з половиною років [9].

У **проекті ANTHRAX** (Сибірська виразка), який виник у зв'язку з існуючими в США випадками поширення поштою спор сибірської виразки і був направлений на пошук протиотрути. Проект був запущений після того, як була виділена ключова білкова компонента спор. Обчислювальна частина завдання полягала в тому, щоб зробити на виділеному білку скринінг 3,57 млрд. потенційних інгібіторів токсину. Завдання вирішувалася на спеціальному варіанті платформи GRID MP компанії UNITED DEVICES. В розподілені обчислення було залучено близько 1,9 млн. серверів і ПК. Висока точність і якість забезпечувалися п'ятикратним рівнем надмірності при скринінгу кожної молекули. Повний скринінг був закінчений за 24 дня. За свідченням фахівців Оксфордського університету, якби ця робота робилася традиційними методами, вона б тривала кілька десятків років, а не чотири тижні [9].

**Проект GLOBUS** орієнтований на надання інфраструктури для грід-обчислень. Метою його створення була надання користувачам можливості працювати з розподіленими обчислювальними ресурсами як з окремим елементом. Проект Globus є продовженням розвитку проекту I-WAY, в ході якого основний акцент був перенесений з підтримки високопродуктивних обчислень в сторону сервісів підтримки віртуальних організацій [11].

Основним елементом проекту Globus виступає Globus Toolkit, який в 1998 році компанія Globus Alliance вперше запропонувала спільноті як ППЗ. Остання версія цього пакету (GT5) включає набір сервісів: Execution Management (управління завданнями), Information Services (інформаційні сервіси) і Data Management (управління даними) [11].

Globus Toolkit містить набір модулів, які використовуються для побудови віртуальної організації при розподілених обчисленнях. Кожен модуль визначає інтерфейс, використовуваний високорівневими компонентами, і має реалізацію для різних середовищ виконання [11].

Велике поширення отримала **система Condor**, створена в університеті штату Вісконсін (США) для об'єднання комп'ютерів університету. Вона використовується для вирішення ряду завдань, які потребують великої кількість обчислювальних ресурсів. Система Condor дозволяє розподіляти завдання як на відчужувані, так і не невідчужувані ресурси, що дозволяє ефективно використовувати їх обчислювальні потужності. ПЗ системи Condor є безкоштовним для користувачів [11].

Condor володіє гнучкою системою розподілу ресурсів, яка реалізована за допомогою мови ClassAd (Classified Advertisement). ClassAd одна з перших мов опису завдань, розроблена для систем управління робочими потоками, заснованих на Condor. ClassAd є функціональною мовою, яка дозволяє користувачеві створювати опис ресурсів і завдань, що надходять на виконання. Дана мова легко розширюється, що дозволяє легко нарощувати необхідний список атрибутів. В Condor на основі мови ClassAd реалізований механізм matchmaking, який здійснює пошук ресурсів для кожного завдання з черги шляхом зіставлення інформації про ресурси і запитів завдань. Основною одиницею мови є вирази (expression), тому що виконання завдання вимагає обчислення всіх виразів. Мова ClassAd надалі була використана як основа для побудови мови Java Declarative Language (JDL) [11].

Condor підтримує механізми міграції завдань, створення контрольних точок, віддалений виклик процедур. У разі зростання активності власника виконавчого комп'ютера механізм міграції спільно з механізмом створення контрольних точок дозволяють продовжити обробку завдання на іншому комп'ютері з того місця, на якому вона призупинилася [11].

**Проект Legion** вперше був представлений в 1993 році. Основна його мета полягала в побудові метакомп’ютерного середовища, незалежного від масштабу, географічного положення, мови або операційної системи. В архітектурі системи використовується підхід, заснований на принципах об'єктно-орієнтованого програмування. Всі сутності в системі є об'єктами (включаючи комп'ютери, мережі, файли) зі специфічними процедурами доступу, які виконуються на віртуальній машині. Однак даний підхід не набув широкого поширення в через те, що застосунки для грід не є об'єктно-орієнтованими [11].

Legion надає користувачеві набір об'єктів, що надають базові сервіси:

* об'єкти обчислювачів – реалізують базові принципи роботи з обчислювальними ресурсами;
* об'єкти систем зберігання даних – надають базові методи роботи з системами зберігання даних;
* об'єкти зв'язування – що забезпечують зв'язок між абстрактним ідентифікатором об'єкта і його фізичною адресою;
* об'єкти контексту – об'єкти, що реалізують проекцію користувальницьких імен об'єктів на абстрактні ідентифікатори об'єктів в системі Legion.

Legion багато в чому розділяє з Globus Toolkit базові принципи побудови грід-систем, проте на відміну від Globus Toolkit, в якому акцент робиться на стандартизацію віддаленої взаємодії, орієнтований на об'єктно-орієнтовану модель [11].

В даний час проект Legion розвивається в двох напрямах: дослідному та комерційному. Компанія Avaki поставляє комерційні рішення на основі моделі системи Legion. Є віртуальна машина для Legion-об'єктів – грід Centurion [11].

**The Worldwide LHC Computing Grid** (WLCG) - це глобальна обчислювальна інфраструктура, метою якої є надання обчислювальних ресурсів для зберігання, розподілу та аналізу даних, що генеруються Великим адронним колайдером (ВАК), роблячи дані однаково доступними для всіх партнерів, незалежно від фізичного розташування [12].

WLCG - це найбільша в світі обчислювальна мережа. Вона підтримується багатьма асоційованими національними та міжнародними мережами по всьому світу, такими як European Grid Initiative (заснована в Європі) та Open Science Grid (заснована в США), а також багатьма іншими регіональними мережами [12].

WLCG поєднує в собі обчислювальні ресурси близько 900 000 комп'ютернів з понад 170 сайтів у 42 країнах, створюючи величезну розподілену обчислювальну інфраструктуру, яка забезпечує більш ніж 12 000 фізиків по всьому світу доступом майже в реальному часі до даних ВАК та потужності для їх обробки. WLCG виконує понад 2 мільйони завдань на день і наприкінці 2 запуску ВАК загальна швидкість передачі регулярно перевищувала 60 ГБ/с. Ці цифри будуть збільшуватися з часом, а обчислювальні ресурси та нові технології ставатимуть дедалі доступнішими у всьому світі [12].

WLCG координується CERN (з фр. Conseil Europeen pour la Recherche Nucleaire – Європейська рада з ядерних досліджень). Вона управляється всесвітньою міжекспериментальною співпрацею (ALICE, ATLAS, CMS та LHCb) та комп'ютерними центрами-учасниками. WLCG розглядається радою делегатів агентств фінансування країн-партнерів та науково перевіряється Експериментальним комітетом LHC. CERN забезпечує близько 20% ресурсів WLCG [12].

# РОЗДІЛ 2. ФУНКЦІОНАЛЬНІ МОЖЛИВОСТІ СИСТЕМИ UNICORE І ЇЇ КОМПОНЕНТІВ

## **Загальна інформація про UNICORE**

UNICORE (UNiform Interface to COmputing REsources) - це комплекс проміжного програмного забезпечення, що дозволяє організувати безпечний і простий доступ до ресурсів грід-системи, яка може включати в себе різнорідні обчислювальні вузли і сховища даних [13].

Проект по розробці UNICORE був ініційований в 1997 році міністерством освіти Німеччини і головною його метою було створення системи, дозволила б забезпечити захищений і прозорий доступ до суперкомп'ютерів провідних німецьких університетів. UNICORE розвивався окремо від аналогічних проектів, таких, наприклад, як Globus Toolkit, так як робота над ним розпочалася ще до прийняття нової парадигми грід-обчислень. Перший прототип системи був отриманий після закінчення в 2002 році проектів «UNICORE» і «UNICORE Plus», відтоді UNICORE активно розвивається на базі європейських проектів, що змінюють один одного, з надання вченим з різних країн доступу до високопродуктивних системам [13].

UNICORE - одна з небагатьох оригінальних платформ, що не використовувала Globus Toolkit, проте останнім часом відбувається трансформація її протоколів в сторону загальновизнаних для грід. Також можна виділити наступні особливості UNICORE:

* відноситься до відкритого ПЗ (ліцензія BSD);
* відповідає стандартам організацій Open Grid Forum (OGF), W3C, OASIS, і IETF; зокрема: Open Grid Services Architecture (OGSA) і Web Services Resource Framework (WS-RF 1.2);
* реалізовано з використанням сучасної відкритої і розширюваної архітектури Service-Oriented Architecture (SOA), яка дозволяє використовувати різні варіанти компонентів для роботи;
* здатний взаємодіяти з іншими програмними продуктами грід;
* пропонує компоненти для всіх рівнів архітектури організації доступу до грід-систем, в тому числі і клієнтські програми для користувачів. Може бути легко інтегрований в існуючу інфраструктуру;
* використовує сучасні протоколи забезпечення безпеки, підходящі для використання в суперкомп'ютерних комплексах і грід-системах. Доступ здійснюється за допомогою сертифікатів стандарту X.509;
* підтримує технологію workflow і дозволяє використовувати різні варіанти її реалізації, враховуючи специфіку поточної конфігурації гріда;
* дозволяє використовувати вбудовані наукові додатки різних типів;
* пропонує користувачам два види доступу: через командний рядок і за допомогою графічного клієнта, що підтримує побудову складних завдань;
* спрощене встановлення і налаштування внаслідок однотипного формату конфігураційних файлів компонентів системи;
* підтримка різних операційних систем (Windows, MacOS, Linux, Unix) і планувальників завдань (LoadLeveler, Torque, SLURM, LSF, OpenCCS, і т.д.) [13].

Перераховані переваги, схилили до вибору його як ППЗ для реалізації поставленого завдання.

## **Архітектура і основні компоненти UNICORE**

Архітектура UNICORE формується з клієнтського, сервісного і системного шарів і представлена на рис. 2.1.

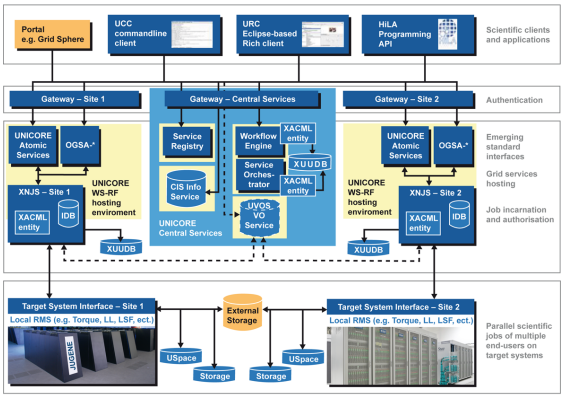


Рисунок 2.1 – Архітектура UNICORE

Верхньою частиною архітектури є **клієнтський шар**. На цьому рівні користувачам доступні різноманітні клієнти, починаючи з графічних клієнтів, таких як Unicore Rich Client (URC) на основі Eclipse, інтерфейсу командного рядка з назвою Unicore Comandline Client (UCC), веб-браузерного компонент Unicore Portal до API під назвою High Level API for Grid Applications (HiLA) [10]. Клієнтське ПЗ дозволяє користувачам створювати описи робіт (jobs), що виконуються в гріді. Одне завдання зазвичай відповідає виконанню комп’ютерної програми на одній із доступних комп’ютерних систем у грід [14]. Розглянемо клієнтське ПЗ UNICORE більш детально.

**Unicore Portal –** це веб-клієнт ППЗ UNICORE. Портал представляє зручний інтерфейс для всіх основних служб UNICORE, а також основну функціональність системи workflow UNICORE. Він пропонує такі функції:

* подання роботи та управління роботами через власні інтерфейси UNICORE;
* переміщення даних (завантаження на сервер і з сервера, копіювання з сервера на сервер тощо) за допомогою функцій управління сховищем UNICORE та доступних протоколів передачі даних;
* функції зберігання, такі як виведення списку, створення, редагування та видалення файлів і папок;
* часткова підтримка системи workflow UNICORE - є основний редактор робочих процесів та таблиця, що відображає workflow користувачів та робочі каталоги. Портал також підтримує надсилання шаблонів робочого циклу;
* інформація про сайти, до яких користувач має доступ;
* різні методи аутентифікації [14].

**Unicore Comandline Client (UCC)** – повнофункціональний клієнт ППЗ UNICORE. UCC має команди клієнта для всіх основних служб UNICORE, інтерфейсу OGSA-Basic Execution Services (OGSA-BES) та системи робочих процесів UNICORE. Є надбудовою над стандартним терміналом Linux або cmd Microsoft Windows. Він пропонує наступні функції:

* подання та управління робочими процесами як через власний інтерфейс UNICORE, так і через OGSA-BES;
* подання та обробка завдань в пакетному режимі з безліччю параметрів налаштування продуктивності;
* переміщення даних (завантаження на сервер і з сервера, копіювання з сервера на сервер тощо) за допомогою функцій управління сховищем UNICORE та доступних протоколів передачі даних;
* функції зберігання (ls, mkdir,…), включаючи створення екземплярів сховищ через Storage Factory;
* повна підтримка системи робочих процесів UNICORE, включаючи можливість запуску окремих завдань через систему посередницьких послуг;
* підтримка системи метаданих UNICORE;
* підтримка спільного використання ресурсів UNICORE через Access Control List (ACL);
* інформація про доступні послуги надається за допомогою команди «system-info»;
* доступ до різних утиліти, таких як «shell» mode, генерування довірчого делегування через Security Assertion Markup Language (SAML), виконання низькорівневих операцій WSRF тощо;
* розширюваність за допомогою користувацьких команд і можливість запускати сценарії, написані мовою програмування Groovy;
* вбудована довідка [15];

**Unicore Rich Client (URC)** – це UCC з графічним інтерфейсом, розроблений в середовищі Eclipse. Надає більшу кількість можливостей для роботи з грід-системою, ніж UCC і UNICORE Portal. інтерфейс дозволяє переглядати повну схему грід-системи, формувати як одиночні завдання, так і workflow-проекти. Для спрощення роботи з workflow, до складу URC входить спеціальний графічний редактор, що дозволяє створювати workflow шляхом створення блок-схеми. Крім того, за допомогою URC можна управляти цифровими сертифікатами [16].

Наступним елементом архітектури є **сервісний шар**. Він включає в себе всі послуги та компоненти UNICORE Service-Oriented Architecture (Сервісно-орієнтованої архітектури, SOA). Рисунок 2.1 показує три набори послуг, лівий і правий містять сервіси розташовані на одному сайті, а середній показує центральні сервіси, такі як Service Registry, Workflow Service та Common Information Service (CIS), які обслуговують всі сайти та користувачів у системі UNICORE [10]. Розглянемо компоненти сервісного шару більш детально.

**UNICORE Gateway** – шлюз, точка входу на сайт UNICORE. Він спрямовує HTTPS трафік на сервери, такі як UNICORE/X. Він встановлюється перед будь-яким мережевим брандмауером. Шлюз аутентифікує вхідні повідомлення та пересилає їх за призначенням. Отримавши відповідь шлюз відправляє її назад клієнту. Другою функцією шлюзу є аутентифікація вхідних запитів. Для проходження аутентифікації необхідна наявність сертифікатів X.509 як у клієнта, так і у компонентів системи. Trust CAs (Certification Authorities) – це довірені центри сертифікатів, підпис яких потрібно для визнання сертифіката X.509 дійсним. Також клієнт повинен бути зареєстрований в базі XUUDB. Інформація про аутентифікованого клієнта передається службам, що знаходяться за шлюзом, у власному форматі UNICORE (як заголовочний елемент Simple Object Access Protocol, SOAP). Компонент Gateway використовується як для центральних сервісів так і для сервісів, розміщених на одному сайті [17].

**XUUDB (X.509 Unicore User Database)** база даних, що містить в собі відповідності між сертифікатами користувачів X.509 і їх параметрами (атрибутами), в число яких входить:

* ім'я облікового запису користувача;
* категорії прав доступу;
* інші дані для авторизації.

В XUUDB входять два веб-сервіси – один для адміністрування бази даних, а інший для обробки вхідних запитів. XUUDB передбачає два види використання сертифікатів – ручний (адміністратор сам контролює всі відповідності) і динамічний (адміністратор формує список правил, згідно з якими будуть призначатися відповідності) [18].

**UNICORE/X** – центральний компонент системи UNICORE. Він виконує такі функції як подання робіт, управління роботами, доступ до сховища та забезпечує перехід до функціональних можливостей цільових ресурсів, наприклад пакетних або файлових систем [19].

Головний компонент UNICORE/X це eXtended Network Job Supervisor (XNJS). Він виконує наступні функції:

* обробка запитів, що надійшли з Gateway;
* збір інформації про доступність ресурсів грід-системи;
* управління завданнями;
* моніторинг виконуваних завдань і передача їх статусу;
* зберігання інформації про виконані завдання, для подальшої передачі користувачеві [19].

Всі завдання описані за допомогою стандартизованої мови JDL (Job Description Language), завдяки чому до складу завдання, крім вхідних даних, входить вся супутня інформація – назва програми і її версія (або шлях до необхідного виконуваного файлу), аргументи і змінні оточення, обсяг запрошених обчислювальних ресурсів, налаштування передачі даних. Використовуючи цю інформацію UNICORE/X визначає, чи можливо в даний момент передати дане завдання на виконання. Якщо так, то завдання відправляється на конкретний обчислювальний вузол, де відбувається його виконання і ведеться моніторинг прогресу, інакше завдання потрапляє в чергу [19].

Для забезпечення безпеки в UNICORE/X передбачені наступні процедури:

* аутентифікація вхідних завдань за допомогою SSL протоколу;
* повідомлення містять інформацію для авторизації, яка вилучається і обробляється сервером (ім'я відправника, рівень доступу до ресурсів тощо);
* регулярні цикли перевірок сертифікатів.

Крім того, UNICORE/X містить компоненти, що дозволяють забезпечити інтероперабельність з іншими грід-системами [19].

**UNICORE Registry** представляє сервісний реєстр, де різні служби реєструються після їх запуску. UNICORE Registry необхідний для зберігання інформації про поточну архітектуру грід-системи і її сервісів. Клієнти звертаються до цього реєстру, щоб підключитися до мережі. Інфраструктура UNICORE може мати кілька реєстрів [20].

**UNICORE Workflow System** – допоміжний компонент грід-системи. Забезпечує розширені можливості обробки робочих процесів із використанням ресурсів UNICORE. Його основними компонентами є Workflow Engine та Service Orchestrator. Workflow Engine забезпечує високорівневі конструкції управління (for-each, while, if-then-else тощо), Service Orchestrator містить потужний, розширюваний ресурсний посередник і займається виконанням окремих UNICORE робіт (single UNICORE jobs) [21]. Спочатку підтримка Workflow System мала головне значення для UNICORE. Двошаровий дизайн з відокремленням в основному зроблений для кращої масштабованості, але також пропонує можливість додавати специфічні для доменів workflow-мови та workflow-механізми. Можливості workflow пропонуються користувачам на клієнтському рівні через URC. Крім того, визначення, подання, моніторинг та контроль робочих процесів також можливі через UCC. Служба відстеження (Tracing Service) збирає інформацію про час роботи workflow-системи та дозволяє генерувати показники продуктивності. Кінцеві користувачі можуть скористатися службою відстеження для візуалізації виконання складного робочого процесу всередині URC [13].

Останнім рівнем в архітектурі UNICORE є **системний шар**. У ньомузнаходяться компоненти UNICORE TSI і USpace.

**UNICORE TSI (Target System Interface)** являє собою Perl демон, який працює на інтерфейсі цільового ресурсу. TSI забезпечує віддалений інтерфейс операційної системи, пакетної системи та файлової системи цільового ресурсу. Він використовується сервером UNICORE/X для виконання завдань на цільовому ресурсі, таких як подання та моніторинг завдань, обробка даних, управління каталогами тощо [22]. В TSI абстрактні команди з рівня грід перетворюються в команди, специфічні для системи, наприклад у разі подання завдання, викликаються специфічні команди системи управління ресурсами (resource management system). TSI виконує роботу від імені користувачів UNICORE, тому він повиннен мати можливість виконувати процеси під різними uids та gids. Тому в робочих умовах його слід запускати з достатніми привілеями (частіше за все з привілеями суперкористувача root) для того, щоб TSI міг повноцінно виконувати свої функції (всі інші компоненти UNICORE на сайті виконуються із стандартного облікового запису користувача, бажано зі спеціально створеного і пов’язаного з UNICORE). TSI доступний для різноманітних часто використовуваних пакетних систем, таких як Torque, LoadLeveler, LSF, SLURM, OpenCCS тощо [13].

**USpace** - це каталог завдань UNICORE. Окремий каталог існує для кожного завдання, де XNJS та TSI зберігають всі вхідні і вихідні дані, включаючи записи із потоків *stdout* і *stderr*. TSI також дозволяє мати доступ до локальних файлових систем, таких як директорія Home користувача. Вони називаються UNICORE Storage і можуть використовуватися в операціях поетапного введення та виведення системи з експлуатації [13].

* 1. **Стандарти UNICORE**

Спільні відкриті стандарти мають головне значення для реалізації взаємодіючих у всьому світі мережевих інфраструктур. З метою підвищення сумісності з грід-інфраструктурами, що мають інші рішення щодо ППЗ, в UNICORE використовуються кілька стандартів від Open Grid Forum та OASIS у різних сферах. Повний стек веб-служб на основі WS-RF 1.2, SOAP та WS-I реалізований для побудови сервісно-орієнтованої архітектури UNICORE [23].

Для безпеки в якості основи під час доступу використовуються повні сертифікати X.509, в той час як контроль доступу базується на політиці eXtensible Access Control Markup Language (XACML). Підтримуються VOMS (Virtual Organization Membership Service) на основі SAML, а також проксі сертифікатів [23].

Стандарти відіграють значну роль не лише в галузі безпеки, але й у сфері інформації. У цьому контексті інформаційний сервіс UNICORE, що має назву CIS, базується на інформаційній моделі GLUE 2.0 і дозволяє стандартизувати спосіб експонування обчислювальних ресурсів. Сервіс збирає як статичну, так і динамічну інформацію з усіх підключених XNJS, які потім відображаються або у вихідній формі XML (eXtensible Markup Language) або іншій зручній для читання текстовій формі. Оскрім цього також зберігається інформація про довготу та широту, Google Maps може ілюструвати географічне представлення інфраструктури грід-мережі [23].

Для управління роботами OGSA-BES та декілька профілів (High Performance Computing Basic Profile, HPC-BP і High Performance Computing File Staging Profile, HPC-FSP) для створення, моніторингу та контролю робіт, включаючи індексування даних, тоді як визначенню завдання відповідає стандарт Job Submission Description Language (JSDL) та його профілі [23].

У галузі управління даними та передачі, може використовуватися OGSA-ByteIO для передачі даних як з сайту на сайт, так і від клієнта на сайт. Для передачі даних із зовнішнього сховища та на нього може бути використаний протокол передачі GridFTP у якості альтернативи протоколу HTTP, що встановлений за замовчуванням. На системному рівні доступна версія TSI для Distributed Resource Management Application API (DRMAA) стандарту, що забезпечує стандартизований інтерфейс між ТSI та пакетною системою (batch system) [23].

# РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ОБЧИСЛЮВАЛЬНОЇ ГРІД-СИСТЕМИ

## **Встановлення операційних систем для сервера UNICORE**

Для установки сервісів UNICORE (за винятком графічного клієнта) необхідна UNIX-подібна система. Вона може використовуватися як з підключеним графічним інтерфейсом, так і без нього. В рамках реалізації поставленого завдання перші компоненти грід-системи будуть встановлені на віртуальну машину Debian 10.9.0 amd64, додаткові обчислювальні ресурси будуть використовувати віртуальну машину на Ubuntu 18.04 LTS minimal. Unicore Rich Client для керування грідом буде встановлений на машину під керуванням Windows 10. Налаштування грід-мережі буде виконуватись через термінал.

## **Встановлення додаткового ПЗ, конфігурація ОС**

Оскільки всі компоненти грід-системи UNICORE за своєю суттю є Java-процесами, то для старту їх роботи потрібний встановлений на системах Java-пакет. Версія установчого пакета UNICORE безпосередньо залежить від версії Java, і вимагає лише певної версії Java JDK. Остання (На момент написання даної роботи) версія установчого пакета UNICORE (8.1.0) вимагає Java версії не нижче 1.8.0. В даному випадку Java потрібно встановлювати окремо на обидві машини. Для встановлення достатньо скористатись утилітою aptitude.

UNICORE - це не тільки Java-застосунок, але також і bash і python-скрипти для конфігурації і установки сервісів. І якщо bash інтерпретатор йде за замовчуванням у всіх дистрибутивах linux, то python потрібно встановлювати

окремо. Для коректної роботи всіх скриптів досить python-minimal (версія 2.5-2.7), для роботи TSI необхідна версія 3.4. або більш пізня.

Для того, щоб розширити кількість видів компільованих програм (не обмежуючись стандартним вибором) хорошим рішенням буде установка linux

пакету "Development Tools" - він встановлює пакет компіляторів, в число яких входять компілятори для С, С ++, Perl, а також кілька системних компіляторів для системи linux. Так само корисною установкою буде утиліта htop – розширений системний моніторинг для процесів і обчислювальних потужностей ОС Linux, дані якої можна буде використовувати при аналізі роботи і адмініструванні вузлів гріда UNICORE. Встановити можна через утиліта aptitude.

## **Встановлення і налаштування сервера UNICORE**

Необхідний для установки сервера UNICORE дистрибутив можна скачати

з репозиторію SourceForge, перейти до якого можна з офіційного сайту UNICORE. Для установки системи потрібно завантажити пакет UNICORE Servers, що включає в себе такі компоненти як UNICORE/X, Registry, Gateway, XUUDB і UNICORE TSI.

Є два способи установки сервісів UNICORE - установка за допомогою Java-установника (unicore-servers-X.X.X.jar) і ручна збірка, компіляція і установка з вихідних файлів (unicore-servers-X.X.X.tar.gz). Перший варіант більш простий в установці і подальшої деінсталяції пакета, однак накладає певні обмеження на подальше адміністрування вузла, з огляду на відсутню частина керуючих скриптів і вихідного конфігураційного файлу.

Виходячи з вищесказаного, для виконання поставленого завдання доцільна установка пакета сервісів UNICORE шляхом ручного збірки і компіляції.

## **Встановлення компонентів системи UNICORE**

Після завантаження та розпакування програмного пакета необхідно виконати його конфігурацію. Для цього необхідно виконати наступні кроки:

1. Відредагувати файл конфігурації configure.properties. Можна стверджувати, що цей файл - найбільш важлива частина всього пакету. Він містить в собі всі параметри установки. Через редагування цього файлу коригуються всі налаштування компонентів UNICORE. Згідно офіційної документації, даний конфігураційний файл – єдиний параметр, який вимагає регулярної архівації з метою його збереження від втрати. Маючи на руках готовий конфігураційний файл configure.properties, можна з легкістю підняти ті ж самі UNICORE сервіси на іншій обчислювальній машині.
2. Запустити скрипт configure.py. Даний скрипт застосує змінені параметри з configure.properties, помістивши їх в конфігураційні файли кожного з сервісів.

Для установки обраних сервісів необхідно запустити скрипт install.py, який розташує файли згідно заданої конфігурації (повинно бути виконано лише один раз). Вміст конфігураційного файлу configure.properties знаходиться в додатку A. При необхідності змінити параметри системи після її установки можна через редагування файлу configure.properties і подальшого запуску configure.py, або ж можна відредагувати конфігураційні файли в каталозі conf кожного з сервісів.

Для перевірки працездатності системи будуть використані тестові сертифікати, які включені в програмний пакет. Під час конфігурації вони були поміщені в XUUDB.

Налаштування компонента Gateway проводиться за допомогою конфігураційних файлів, що зберігаються в каталозі unicore/gateway/conf. У файлі connections.properties було поставлено порт для шлюзу, а в файлі gateway.properties була включена можливість динамічної реєстрації користувачів. В результаті цього ми позбавляємося від необхідності вручну модифікувати XUUDB, додаючи дані про користувачів і їхні сертифікати, все це буде зроблено автоматично при першому підключенні клієнта до системи. У цьому ж файлі можна налаштувати фільтри, щоб обмежити можливість реєстрації для клієнтів. Для запуску системи необхідно спочатку ініціювати XUUDB, а потім запустити сервер UNICORE/X за допомогою bash-скрипта. У момент запуску XUUDB підключить тестові сертифікати. Після цього буде ініціалізовано і підключено Gateway і сервер почне роботу. Компонент Registry інтегрований в UNICORE/X, немає потреби використовувати його в якості окремого сервісу.

Для підключення обчислювальних ресурсів до грід-системі використовується компонент UNICORE TSI (Target System Interface). Саме він відповідає за передачу фізичних ресурсів на виконання завдань, отриманих від компонента XNJS (EXecution Network Job Supervisor) сервера UNICORE/X. На відміну від інших компонентів системи, TSI необхідно обов'язково запускати від імені root-користувача, так як він повинен мати всі права для виконання своїх функцій.

Установку компонента TSI можна проводити двома шляхами - вказати його параметри в configure.properties, або ж за допомогою скрипта Install.sh. Під час установки необхідно вибрати використовуваний планувальник завдань. UNICORE TSI підтримує інтеграцію планувальників Torque, Slurm, LSF і LoadLeveler. Використання сторонніх планувальників виходить за рамки поставленого завдання, тому при інсталяції було обрано стандартний планувальник "nobatch".

Потрібно враховувати той факт, що після установки доступ до конфігураційних файлів TSI і його скриптів можливий тільки з використанням прав root-користувача.

Запуск системи UNICORE для початку роботи з нею може проводитися декількома шляхами. У кореневому каталозі пакета UNICORE розташований bash-скрипт start.sh, який запускає в певному порядку всі сервіси, але також кожен з сервісів містить свій власний start.sh скрипт в /bin директорії, який можна запускати в індивідуальному порядку в / bin (зручно, якщо сервіс з яких-небудь причин припинив роботу, і його потрібно перезапустити, не зачіпаючи інші працюють сервіси). Так само кожен сервіс пише свій власний інформативний лог в директорії /logs, що спрощує процес адміністрування.

Схема створеної грід-мережі на базі компонентів UNICORE показана на рисунку 3.1.



Рисунок 3.1 – Схема створеної грід-мережі

# РОЗДІЛ 4. ТЕСТУВАННЯ СТВОРЕНОЇ ГРІД-СИСТЕМИ

Тестування реалізованої грід-мережі буде включати в себе:

* тестовий запуск компонентів грід-мережі на декількох машинах;
* демонстрація графічного інтерфейсу для доступу в грід-мережу;
* запуск тестового завдання на віддалених машинах за допомогою графічного інтерфейсу;
  1. **Запуск компонентів системи UNICORE**

Як було сказано раніше, запускати сервіси UNICORE можна декількома способами: головним керуючим скриптом, або ж кожен сервіс окремо. Для демонстрації запуск буде здійснюватись за допомогою головного керуючого скрипта. Процес запуску здійснюється з командного рядка операційної системи Linux, за допомогою команди "./start.sh" з директорії з файлами системи UNICORE. Результат запуску системи на ОС Debian представлений на рис. 4.1.

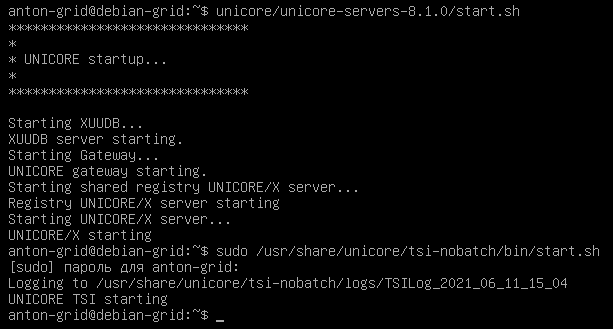


Рисунок 4.1 – Запуск сервісів UNICORE з командного рядка ОС Linux Debian

Результат запуску системи на ОС Ubuntu представлений на рис. 4.2.

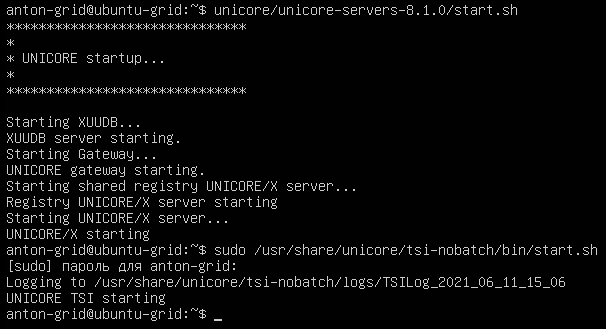


Рисунок 4.2 – Запуск сервісів UNICORE з командного рядка ОС Linux Ubuntu

Після запуску вузол (комп'ютер або кластер комп'ютерів) буде готовий до роботі. Для перевірки того, що запуск сервісів пройшов успішно, необхідно перевірити файл startup.log (у кожного сервісу він свій).

Після запуску всіх сервісів UNICORE і старту роботи всіх вузлів грід-системи, можна отримати доступ до реалізованої грід-мережі за допомогою UNICORE Rich Client. Після аутентифікації грід-мережа буде виглядати наступному чином (рис. 4.3):

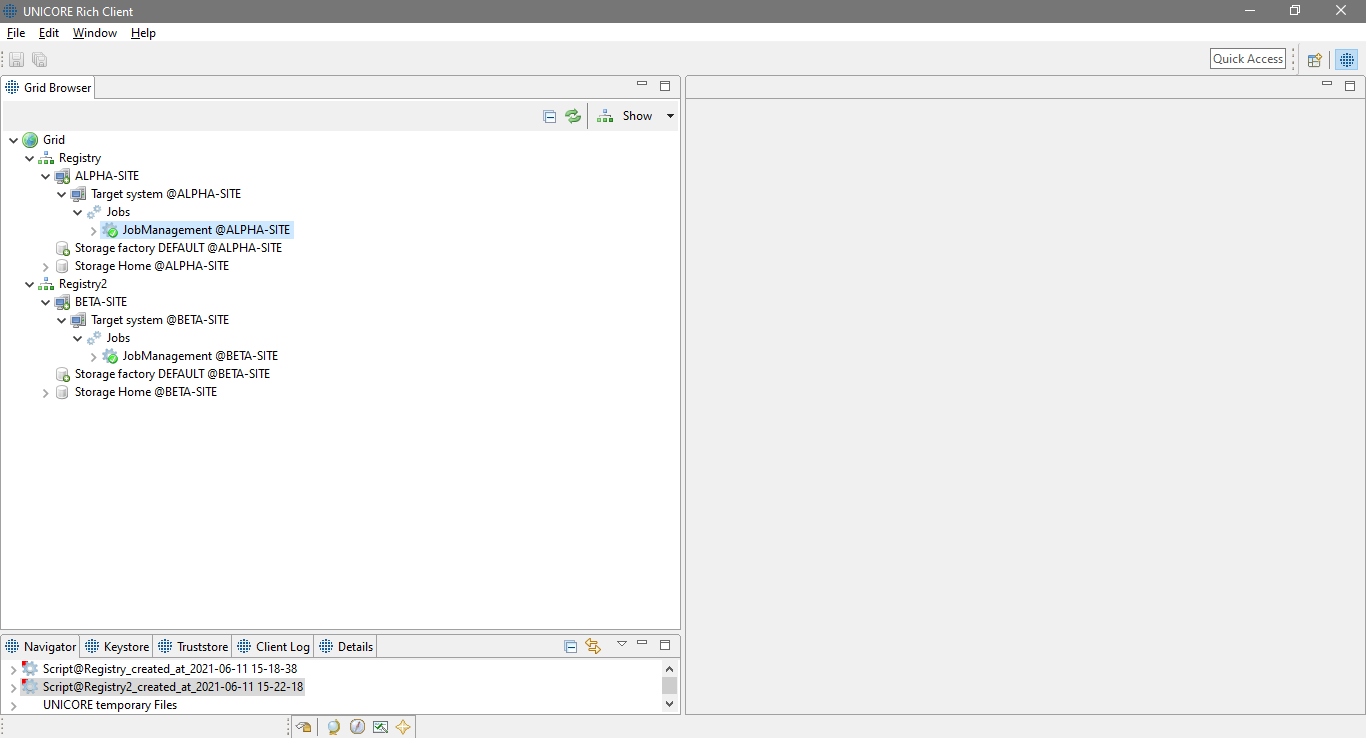


Рисунок 4.3 – Графічний клієнт для роботи з грід-системою UNICORE

На рисунку 4.3. представлений базовий інтерфейс для роботи з грід-системою. У вкладці "Grid Browser" представлена схема існуючої грід-мережі у вигляді списку. Вся грід-мережу ділиться на "сайти", кожен з яких містить в собі деяку кількість «цільових систем» – ними можуть бути як один комп'ютер, так і їх об'єднання. На кожному «сайті» існує власна база даних, вона містить в собі інформацію про цільові системи і їх користувачів. Вкладка "Jobs" містить в собі список виконаних завдань. Кожна така "робота" зберігає в собі повну інформацію по виконане завдання: вхідні дані, час виконання, вузол, на якому вона обчислювалася і вихідні дані.

Більш детальний вид списку представлений на рис. 4.4.

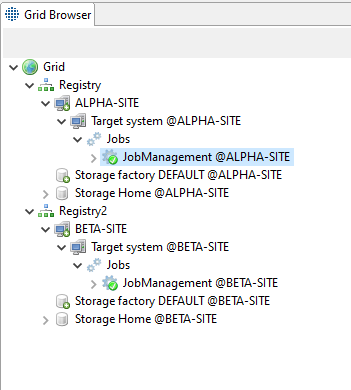


Рисунок 4.4 – Схема реалізованої грід-системи

Для нової виконуваної завдання перш за все потрібно створити для неї оболонку. Для цього необхідно зайти в створення нової роботи (можна зробити як в загальному інтерфейсі, так і в окремих цільових системах).

Після цього вибрати необхідний обробник (по замовчуванню є bash, python, java, але список може бути розширений за рахунок встановлених компіляторів на цільових системах). Після чого вибирається назва для завдання і система (або сайт), на яких вона буде виконуватися. Якщо вузол не вказано вручну, то грід-мережа вибере його самостійно.

Процес створення нової роботи представлений на рисунках 4.5-4.6:

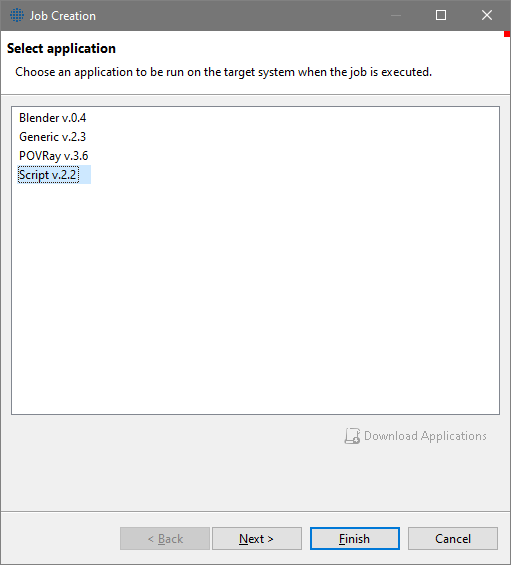


Рисунок 4.5 – Вибір типу оброблюваної завдання



Рисунок 4.6 – Вибір назви та місця зберігання виконуваної завдання

Процес вибору регіону для обчислення завдання представлений на рисунку 4.7.

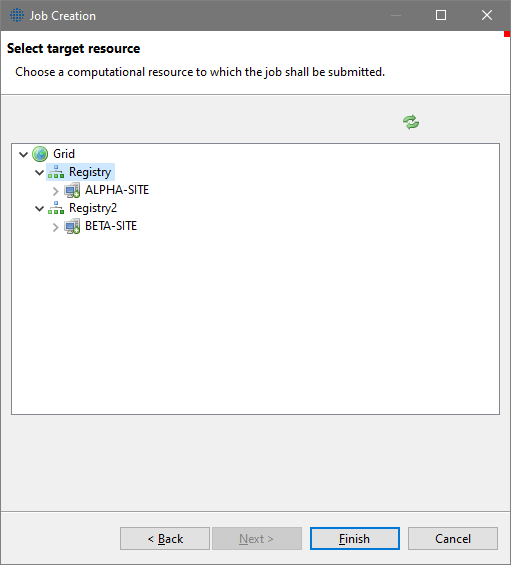


Рисунок 4.7 – Вибір регіону для обчислення завдання

Після успішного створення нового завдання в головному вікні з'явиться вкладка з створеної роботою, в тіло якої поміщається програмний код обчислювальної завдання. В якості додаткових параметрів можуть бути вказані розширені параметри компіляції, час роботи, конкретні обчислювальні ресурси в цільової системі. Після закінчення налаштувань, робота зберігається і відправляється натисканням на кнопку Submit на верхній панелі керування. Після запуску робота відправляється на обраний обчислювальний вузол (або визначається сайтом автоматично, якщо конкретна система вказана не була), і запускається на обробку. По закінченню обчислень, робота завершується з позитивним або негативним результатом. Незалежно від успішності обчислень, робота зберігає в собі всі вихідні дані для подальшого їх аналізу і можливого редагування. Також передбачений експорт підсумкових даних в файл для спрощення звітності.

* 1. **Тестування роботи грід-системи**

Тестування коректності роботи проведемо на прикладі обчислення факторіалу числа 10. Спочатку виконаємо це завдання на ALPHA-SITE (ОС Debian) Процес відправки завдання на виконання показаний на рисунках 4.8-4.9.

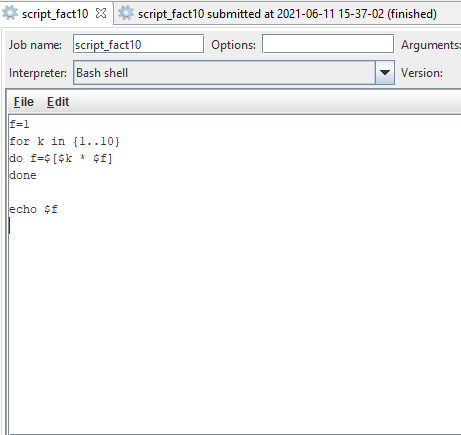


Рисунок 4.8 – Приклад програмного коду, готового до запуску

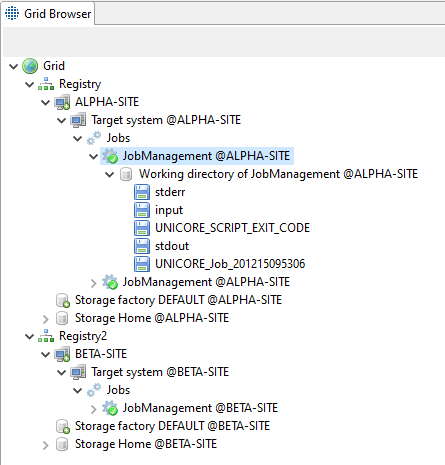


Рисунок 4.9 – Вид успішно завершеного завдання, яке зберігає в собі результати обчислення

Результат проведеного обчислення показаний на рисунку 4.10.

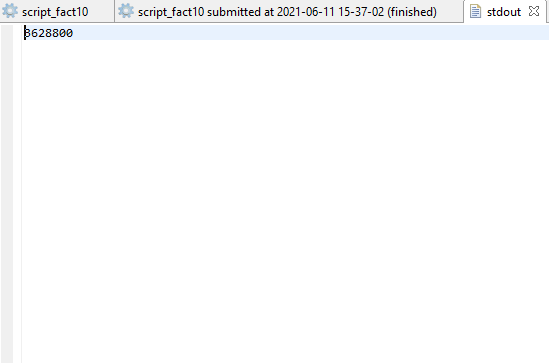


Рисунок 4.10 – Результат проведеного обчислення

Тепер виконаємо те ж завдання на BETA-SITE (ОС Ubuntu). Процес відправки завдання на виконання показаний на рисунках 4.11-4.12.

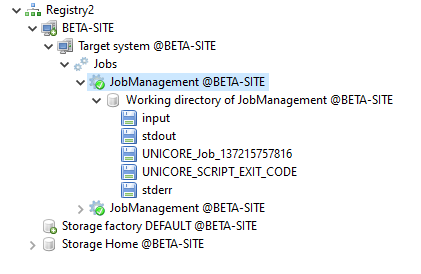


Рисунок 4.11 – Вид успішно завершеного завдання, яке зберігає в собі результати обчислення

Результат проведеного обчислення показаний на рисунку 4.12.



Рисунок 4.12 – Результат проведених обчислень

## **Введення в експлуатацію**

Існуючу грід-мережу можна без труднощів збільшити за рахунок нових робочих станцій, встановивши на нові машини дистрибутив Unicore Servers і налаштувавши файл configuration-properties. Також, за допомогою цього ж дистрибутиву і файлу конфігурації можна легко створити нову грід-мережу будь-якої конфігурації.

# ВИСНОВКИ

Мета дипломної роботи полягала в конструювання обчислювальної грід-системи на базі компонентів UNICORE для об’єднання децентралізованих і/або віддалених обчислювальних ресурсів в одну мережу для подальшого централізованого доступу до їх обчислювальних потужностей, забезпечення адміністрування і подальшого масштабування цієї системи.

В процесі виконання роботи була вивчена фундаментальна теорія сучасних розподілених обчислень, їх типи. Досліджено концепцію грід-обчислень, їх стандартизація. Досліджені компоненти обраної для виконання завдання системи UNICORE, їх розподіл, призначення, правила використання.

Отримані знання дозволили вибрати оптимальний варіант для реалізації власної грід-мережі, з'ясувати слабкі або вразливі місця для їх подальшого удосконалення.

Грід-мережа була реалізована з урахуванням всіх необхідних стандартів для розподілених обчислень, встановлення та налаштування всіх її компонентів на практиці дозволило продемонструвати основні і необхідні елементи для розподілених обчислень, такі як гетерогенне і легко масштабоване середовище, об'єднання децентралізованих ресурсів, можливість використання графічного інтерфейсу, створення робіт по обчисленню завдань на різних мовах програмування.

Після завершення практичної частини роботи була розроблена достатньо універсальна грід-мережа, з легкою масштабованістю. Установка сервісів не залежить від апаратної частини обчислювальних машин. Розроблена система здатна виконувати різноманітні обчислювальні задачі на різних її компонентах.

Робота грід-мережі була перевірена в тестових умовах, і відповідає всім вимогам, заявленим в поставленому завданню. Масштабуючи отриману мережу, доповнюючи її більшою кількістю цільових систем, вона буде готова до вирішення складніших обчислювальних задач.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Foster I. The Grid: Blueprint for a New Computing Infrastructure / I. Foster, C. Kesselman. – San Francisco: Morgan Kaufmann Publishers, 1998. – pp. 259-278.
2. Конспект лекцій з дисципліни «Грід-системи та технології хмарних обчислень» для студентів освітніх рівнів «бакалавр», «магістр» / Г. В.Шимчук, О. В. Маєвський, О. Б. Назаревич, М. А. Стадник. – Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2016. – 340 с.
3. Крюков Ю.А. Вычислительная инфраструктура для прикладных задач, будущее и настоящее / Ю.А. Крюков // Геоинформатика. – 2004. – № 9. – С. 57-61.
4. Methods and Experiences of Parallelizing Flood Models / L. Hluchy, V.D. Tran, D. Froehlich [etc.] // The 10th Euro PVM/MPI conference. LNCS 2840. – Sept. 2003. – P. 677-681.
5. Devid M. Model-Based Evaluation: From Dependability to Security / M. Devid, H. William, S. Kishor // IEEE Transactions on Dependable and Security Computinq. – January-March, 2004. – Vol. 1, № 1. – P. 48-65.
6. Сервисы GRID, как объекты стандартизации [Електронний ресурс] – Режим доступу до ресурсу: <http://jre.cplire.ru/jre/dec03/4/text.html>.
7. National Research Council. Realizing the Information Future: The Internet and Beyond / National Research Council.. – Washington, DC: The National Academies Press, 1994. – 320 p.
8. Foster I., Kesselman C., Tuecke S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. The International Journal of High Performance Computing Applications. 2001. Vol. 15, No. 3. pp. 200–222.
9. Организация распределенных вычислений на базе GRID-технологии / Р.К. Алекперов // Штучний інтелект. – 2011. – № 1. – С. 6-14.
10. For 21 years, millions of people helped a university search for alien life. Now it's time to analyze the results [Електронний ресурс] // CNN. – 2020. – Режим доступу до ресурсу: <https://edition.cnn.com/2020/03/05/us/seti-home-hibernation-alien-trnd-scn/index.html>.
11. Анализ современного состояния и развития Grid-технологий и языков описания заданий / М. А. Волк, Т. В Филимончук, Шиблак Муаз. Ал // Збірник наукових праць Харківського університету Повітряних сил. – 2013. – № 2. – С. 75-81.
12. The Worldwide LHC Computing Grid (WLCG) [Електронний ресурс] – Режим доступу до ресурсу: <https://home.cern/science/computing/grid>.
13. UNICORE Homepage [Електронний ресурс] – Режим доступу до ресурсу: <https://www.unicore.eu>.
14. UNICORE Web Portal: Administrator Manual [Електронний ресурс] – Режим доступу до ресурсу: <https://www.unicore.eu/docstore/portal-2.2.0/portal-manual.html>.
15. UNICORE Commandline Client: User Manual [Електронний ресурс] – Режим доступу до ресурсу: <https://www.unicore.eu/docstore/ucc-7.11.0/ucc-manual.html>.
16. UNICORE Rich Client user manual [Електронний ресурс] – Режим доступу до ресурсу: <https://www.unicore.eu/docstore/urc-7.1.3/manual.html>.
17. UNICORE Gateway [Електронний ресурс] – Режим доступу до ресурсу: https://www.unicore.eu/docstore/gateway-1.5.11/manual.html.
18. XUUDB Manual [Електронний ресурс] – Режим доступу до ресурсу: <https://www.unicore.eu/docstore/xuudb-2.5.0/manual.html>.
19. UNICORE/X Manual [Електронний ресурс] – Режим доступу до ресурсу: <https://www.unicore.eu/docstore/unicorex-7.12.0/unicorex-manual.html>.
20. UNICORE Registry Manual [Електронний ресурс] – Режим доступу до ресурсу: <https://www.unicore.eu/docstore/registry-7.9.0/registry-manual.html>.
21. UNICORE Workflow System manual [Електронний ресурс] – Режим доступу до ресурсу: <https://www.unicore.eu/docstore/workflow-6.4.1/workflow-manual.html>.
22. UNICORE TSI: Manual [Електронний ресурс] – Режим доступу до ресурсу: <https://www.unicore.eu/docstore/tsi-6.5.1/tsi-manual.html>.
23. UNICORE 6 – Recent and Future Advancements / A. Streit et al. annals of telecommunications - annales des télécommunications. – 2010. – Т.65, №. 11-12. – С. 757–762.
24. Ubuntu Desktop Guide [Електронний ресурс] – Режим доступу до ресурсу: <https://help.ubuntu.com/18.04/ubuntu-help/index.html>.
25. Руководство по установке Debian GNU/Linux [Електронний ресурс] – Режим доступу до ресурсу: https://www.debian.org/releases/buster/amd64/.

# ДОДАТОК А

# КОНФІГУРАЦІЙНИЙ ФАЙЛ CONFIGURE.PROPERTIES

#

# Configuration properties

#

# this file is read by the configure.py script.

#

# Note:: the special parameter value "hostname" will be replaced

# by the hostname specified on the command line or

# found by lookup.

#

[parameters]

#

# Target base directory, use "currentdir" to leave as-is

# (otherwise use an absolute path, such as "/opt/unicore")

#

INSTALL\_PATH=currentdir

################################################################

#

# The next settings control which components shall be configured

#

################################################################

# the Gateway

gateway=true

# the UNICORE/X server

unicorex=true

# the UNICORE TSI (check also the uxTSIMode property below)

tsi=true

#

# the shared Registry server

# (if you want UNICORE/X to use this or another external registry,

# make sure to edit the relevant settings below)

#

registry=true

#

# Workflow server

#

workflow=false

# the XUUDB user database

xuudb=true

#

# shall the docs be copied by install.py?

#

docs=true

#

# Shall the demo certificates be installed?

#

# Setting to "true" will install demo server certs and will add

# the "demouser" to the XUUDB and map files

#

installdemocerts=true

#

# Java command to use

#

JAVA\_CMD=java

###################################################

#

# Settings for configuring the Gateway server

#

###################################################

#

# Gateway host and port

#

gwHost=192.168.139.130

gwPort=8080

#

# Allow auto-registration with the gateway?

#

gwAutoRegistration=true

gwAutoRegistrationSecret=\_SET\_ME\_

#

# Add a line to connection.properties

# for registry and workflow

#

gwAddRegistryEntry=${registry}

gwAddWFEntry=${workflow}

#

# Security settings

#

# keystore is a JKS or PKCS12 file

gwKeystore=${INSTALL\_PATH}/certs/gateway.p12

gwKeystorePassword=the!gateway

# truststore type is either "directory" or "keystore"

gwTruststoreType=directory

gwTruststore=${INSTALL\_PATH}/certs/trusted/\*.pem

# password in case for type "keystore"

gwTruststorePassword=n/a

###################################################

#

# Settings for configuring the UNICORE/X server

#

###################################################

#

# UNICORE/X server host and port

#

uxHost=localhost

uxPort=7777

# how much memory to allocate for the UNICORE/X process

uxMemory=256

#

# VSite name

#

uxName=ALPHA-SITE

#

# Security settings

#

# keystore is a JKS or PKCS12 file

uxKeystore=${INSTALL\_PATH}/certs/unicorex.p12

uxKeystorePassword=the!njs

# truststore type is either "directory" or "keystore"

uxTruststoreType=directory

uxTruststore=${INSTALL\_PATH}/certs/trusted/\*.pem

# password required for type "keystore"

uxTruststorePassword=n/a

#

# Gateway host/port. By default these are in sync with the

# gateway server settings defined above

#

uxGatewayHost=${gwHost}

uxGatewayPort=${gwPort}

#

# TSI mode

#

# remote : uses the UNICORE TSI server

# embedded : uses the internal Java TSI

#

uxTSIMode=remote

#

# Settings for defining the external registry

#

#

# register with an external Registry?

#

uxUseExternalRegistry=${registry}

#

# Give the URL of the Registry the UNICORE/X should register

# with (defaults to the one we are installing)

#

uxUrlExternalRegistry=https://${gwHost}:${gwPort}/${registryName}/services/Registry?res=default\_registry

#

# Enable the StorageFactory service for this UNICORE/X server?

# A StorageFactory should ideally be running on sites with a powerful,

# high capacity filesystem

#

uxEnableStorageFactory=true

#

# For the default storage factory, where should the files be created

# This has to be world executable (like the filespace directory)

#

uxDefaultStorageFactoryPath=${INSTALL\_PATH}/unicorex/storage-factory

#

# settings for interfacing to an XUUDB

#

# whether to use the XUUDB

# - if set to "XUUDB", the XUUDB will used

# - if set to "FILE", a map file 'conf/simpleuudb' will be used

# - if set to "GRID-MAPFILE", a gridmap file 'conf/grid-mapfile will be used

uxUseXUUDB=XUUDB

# host/port, by default the XUUDB server values given below are used

uxXUUDBHost=${xuudbHost}

uxXUUDBPort=${xuudbPort}

# GCID: the ID used by the UNICORE/X server for querying the XUUDB

uxGCID=${uxName}

#

# Settings for interfacing UNICORE/X to the TSI server

# By default, these are in sync with the corresponding TSI properties

# defined below

# The TSI base directory

uxTSIDirectory=${tsiInstallDirectory}

uxTSIHost=${tsiHost}

uxTSIPort=${tsiPort}

uxTSICallbackPort=${tsiNjsPort}

# The user id for querying job info from the batch system (cannot be root!)

uxTSIQstatUser=${USER\_NAME}

# The interval between batch system status checks (in milliseconds)

uxTSIStatusUpdateInterval=60000

#

# The directory on the target system where the job directories will be created

# (On a cluster, this should be a shared filesystem!)

# The directory must be accessible for every user id (chmod 1777)

#

uxTSIWorkingDirectoriesBasedir=${INSTALL\_PATH}/FILESPACE

#

# Resource settings, these will go into the IDB file at

# 'unicorex/conf/idb.json'.

# All resource settings have a minimum, a maximum and a default value.

# Please consult the IDB documentation (docs/unicorex/xnjs-idb.txt)

# for further details and options

#

# total CPUs

uxTotalCPUsMin=1

uxTotalCPUsMax=1

uxTotalCPUsDefault=1

# nodes

uxNodesMin=1

uxNodesMax=1

uxNodesDefault=1

# CPUs per node

uxCPUsPerNodeMin=1

uxCPUsPerNodeMax=1

uxCPUsPerNodeDefault=1

# wall clock time (seconds) for the job

uxTimeMin=10

uxTimeMax=86000

uxTimeDefault=3600

# memory per node (in bytes)

uxMemoryPerNodeMin=1048576

uxMemoryPerNodeMax=1073741824

uxMemoryPerNodeDefault=268435456

###################################################

#

# Settings for interfacing UNICORE/X to a VO server

# e.g. Unity, which is used as an attribute source

#

###################################################

# The following settings are enough for an UVOS server. For others you

# may need to adapt the generated configuration

# Which group is used to store Xlogins and Roles for this site?

voGroup=/demo-vo/DEMO-SITE

# whether to query the VO server (aka PULL mode)?

# set to "VO-PULL" to enable, leave empty to disable

voEnablePull=

#voEnablePull=VO-PULL

voPullServerURL=https://localhost:2443/unicore-soapidp/saml2unicoreidp-soap/AssertionQueryService

###################################################

#

# Settings related to the UNICORE TSI server

#

###################################################

# TSI host and port

# (i.e. port on which the TSI process will listen for XNJS requests)

tsiHost=localhost

tsiPort=4433

# The port on which the XNJS is listening for TSI worker connections

tsiNjsPort=7654

#

# The selected TSI (i.e. directory containing platform-specific

# TSI files), one of

#

# nobatch (Simple fork)

# torque (Torque / PBS)

# slurm (Slurm)

# lsf (LSF)

# loadleveler (LoadLeveler)

#

tsiSelected=nobatch

#

# Where to install the TSI server

# (absolute path)

#

tsiInstallDirectory=${INSTALL\_PATH}/tsi\_selected

###################################################

#

# Settings related to the XUUDB server

#

###################################################

#

# XUUDB server host and port

#

xuudbHost=hostname

xuudbPort=34463

#

# Security settings

#

# keystore is a JKS or PKCS12 file

xuudbKeystore=${INSTALL\_PATH}/certs/xuudb.p12

xuudbKeystorePassword=the!xuudb

# truststore is either "directory" or "keystore"

xuudbTruststoreType=directory

xuudbTruststore=${INSTALL\_PATH}/certs/trusted/\*.pem

xuudbTruststorePassword=n/a

# ACL entry: to use the XUUDB admin.sh client, its certificate DN

# needs to be in the xuudb.acl file

# (see docs/xuudb/xuudb.txt for full information)

xuudbAdminDN=CN=Demo XUUDB,O=UNICORE,C=EU

###################################################

#

# Settings related to the Registry server

#

###################################################

#

# Registry server host and port

#

registryHost=192.168.139.130

registryPort=7778

#

# VSite name

#

registryName=ALPHA-REGISTRY

#

# Gateway host/port. By default these are in sync with the

# gateway server settings defined above

#

registryGatewayHost=${gwHost}

registryGatewayPort=${gwPort}

#

# settings for interfacing to an XUUDB

# (only relevant if accesscontrol is enabled)

# whether to use the XUUDB

# - if set to "XUUDB", the XUUDB will used

# - if set to "FILE", a map file 'conf/simpleuudb' will be used

# - if set to "GRID-MAPFILE", a grid-map file 'conf/grid-mapfile' will be used

registryUseXUUDB=XUUDB

# host/port, by default the XUUDB server values given below are used

registryXUUDBHost=${xuudbHost}

registryXUUDBPort=${xuudbPort}

# registry GCID: the ID used by the Registry for querying the XUUDB

registryGCID=REGISTRY

#

# Security settings

#

# keystore is a JKS or PKCS12 file

registryKeystore=${INSTALL\_PATH}/certs/registry.p12

registryKeystorePassword=the!registry

# truststore is either "directory" or "keystore"

registryTruststoreType=directory

registryTruststore=${INSTALL\_PATH}/certs/trusted/\*.pem

# password required for type "keystore"

registryTruststorePassword=n/a

###################################################

#

# Settings related to the Workflow server

#

###################################################

# server host

wfHost=hostname

# server port

wfPort=7700

# memory in megabytes

wfMemory=256

# workflow site name

wfSitename=WORKFLOW

# keystore is a JKS or PKCS12 file

wfKeystore=${INSTALL\_PATH}/certs/workflow.p12

wfKeystorePassword=the!workflow

# truststore type is either "directory" or "keystore"

wfTruststoreType=directory

wfTruststore=${INSTALL\_PATH}/certs/trusted/\*.pem

# Use XUUDB? Or set to "FILE" for simple map file

wfUseXUUDB=XUUDB

wfExternalRegistry=https://${gwHost}:${gwPort}/${registryName}/services/Registry?res=default\_registry

# host/port, by default the XUUDB server values given below are used

wfXUUDBHost=${xuudbHost}

wfXUUDBPort=${xuudbPort}

# GCID: the ID used by the UNICORE/X server for querying the XUUDB

wfGCID=${wfSitename}